Hewlett-Packard Company

TPC Benchmark™ H
Full Disclosure Report
for
HP ProLiant DL785 G6
using
Microsoft SQL Server 2008 Enterprise Edition SP1 for x64
and
Windows Server 2008 Enterprise Edition SP1 for x64

First Edition
July 2009

© 2009 Hewlett-Packard Company. All rights reserved.

Hewlett-Packard Company (HP), the Sponsor of this benchmark test, believes that the information in this document is accurate as of the publication date. The information in this document is subject to change without notice. The Sponsor assumes no responsibility for any errors that may appear in this document.

The pricing information in this document is believed to accurately reflect the current prices as of the publication date. However, the Sponsor provides no warranty of the pricing information in this document.

Benchmark results are highly dependent upon workload, specific application requirements, and system design and implementation. Relative system performance will vary as a result of these and other factors. Therefore, the TPC Benchmark H should not be used as a substitute for a specific customer application benchmark when critical capacity planning and/or product evaluation decisions are contemplated.

All performance data contained in this report was obtained in a rigorously controlled environment. Results obtained in other operating environments may vary significantly. No warranty of system performance or price/performance is expressed or implied in this report.

Copyright 2009 Hewlett-Packard Company.

All rights reserved. Permission is hereby granted to reproduce this document in whole or in part provided the copyright notice printed above is set forth in full text or on the title page of each item reproduced.

Printed in the United States, July, 2009

HP and HP StorageWorks are registered trademarks of Hewlett-Packard Company.

Microsoft, Windows 2008 and SQL Server 2008 are registered trademarks of Microsoft Corporation.

TPC Benchmark, TPC-H, QppH, QthH and QphH are registered certification marks of the Transaction Processing Performance Council.

All other brand or product names mentioned herein must be considered trademarks or registered trademarks of their respective owners.

© 2009 Hewlett-Packard Company. All rights reserved.

Abstract

Overview

This report documents the methodology and results of the TPC Benchmark™ H test conducted on the HP ProLiant DL785 G6 using Microsoft SQL Server 2008 Enterprise Edition SP1 for x64 in conformance with the requirements of the TPC Benchmark™ H Standard Specification, Revision 2.8.0. The operating system used for the benchmark was Microsoft Windows Server 2008 Enterprise Edition SP1 for x64.

The TPC Benchmark™ H was developed by the Transaction Processing Performance Council (TPC). The TPC was founded to define transaction processing benchmarks and to disseminate objective, verifiable performance data to the industry.

Requests for this TPC Benchmark H Full Disclosure Report should be sent to:

Transaction Processing Performance Council (TPC)
c/o Shanley Public Relations
777 North First Street, Suite 600
San Jose, CA 95112 USA
Telephone: (408) 295-8894
Fax: (408) 295-9768

Standard and Executive Summary Statements

Pages iv - vii contain the Executive Summary and Numerical Quantities Summary of the benchmark results for the HP ProLiant DL785 G6.

Auditor

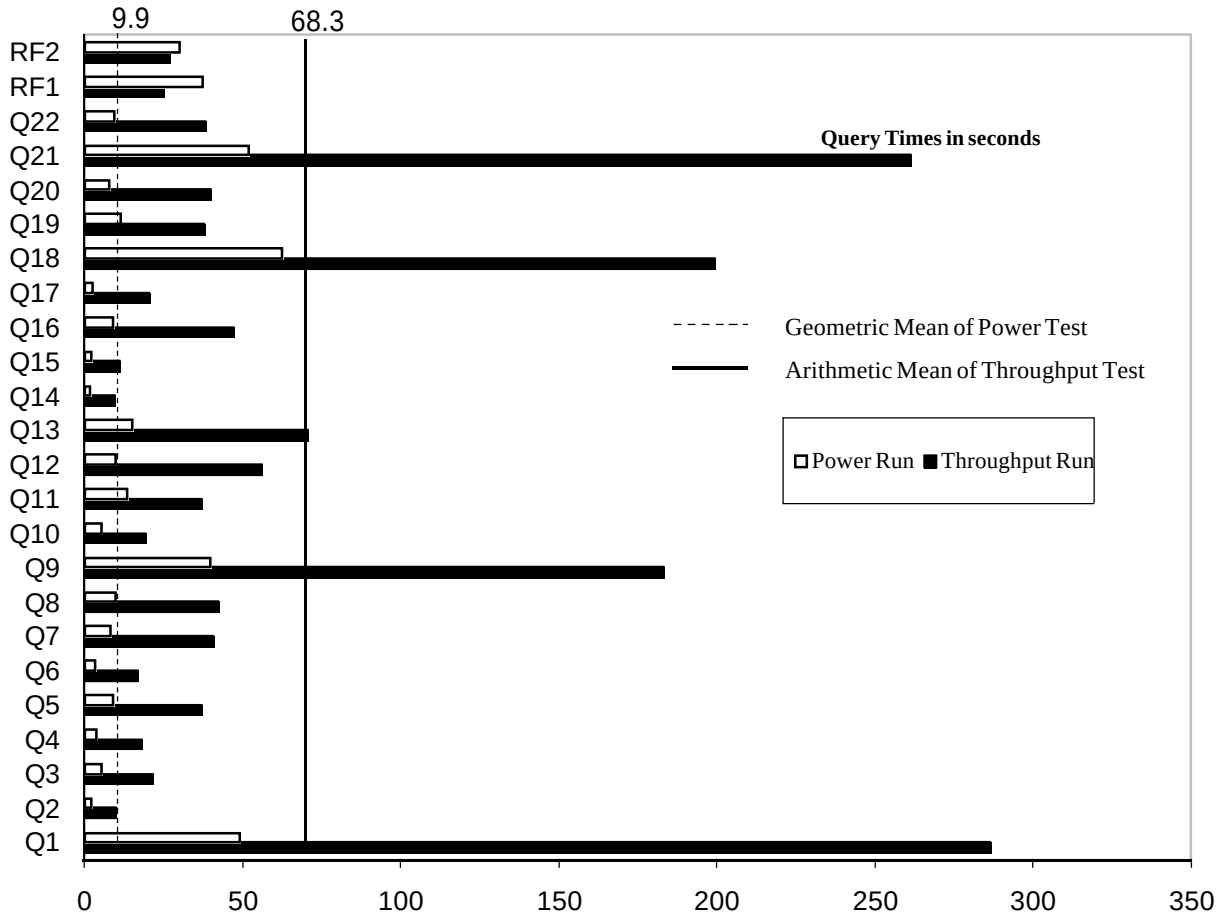
The benchmark configuration, environment and methodology used to produce and validate the test results, and the pricing model used to calculate the cost per QppH and QthH were audited by Lorna Livingtree of Performance Metrics, Inc. to verify compliance with the relevant TPC specifications.

The auditor's letter of attestation is attached in Section 9.1 "Auditors' Report."

© 2009 Hewlett-Packard Company. All rights reserved.

	HP ProLiant DL785 G6	TPC-H Rev. 2.8.0 Report Date: July 30, 2009
Total System Cost	Composite Query per Hour Rating	Price Performance
$ 177,550 USD	**91,558.2 QphH** **@ 300G**	**$ 1.94 USD** **$ / QphH @300G**

Database size	Database Manager	Operating System	Other Software	Availability Date
300GB	**Microsoft SQL Server 2008 Enterprise Edition SP1 for x64**	**Windows Server 2008 Enterprise Edition SP1 for x64**	**None**	**10/30/2009**



Database Load Time = 13:19:50	Load included backup: Y	Total Data Storage / Database Size = 44.66
RAID(Base Tables): N	RAID (Base Tables and Auxiliary Data Structure): N	RAID(All)= N

System Components	System Total	Per Node
Nodes:	1	n.a.
AMD Opteron™ 8439 Six-Core 2.8GHz:	8	8
Cores:	48	48
Threads:	48	48
Memory:	256GB	256GB
OS Disk Drives(internal:4 x 36GB 15Krpm SAS):	4	4
Network Interfaces (on-board GigE):	2	2
Storage Controllers P800:	6	6
Storage Shelves MSA70:	12	n.a.
Storage Subsystem Disk Drives:	194	n.a.
Total Storage:	13,398.8 GB	n.a.

© 2009 Hewlett-Packard Company. All rights reserved.

HP ProLiant DL785 G6

TPC-H Rev. 2.8.0
Report Date: July 30, 2009

Description	Price Key	Part Number	Unit Price	Qty.	Extended Price	3 Yr Maint Price
Server Hardware						
HP DL785 CTO Chassis	1	AH233A	$9,999	1	$9,999	
HP O8439SE 2.8GHz DL785 4p FIO Kit	1	575261-L21	$19,499	1	$19,499	
HP O8439SE 2.8GHz DL785 4p Kit	1	575261-B21	$19,499	1	$19,499	
HP 8GB Reg PC2-6400 2x4GB	1	497767-B21	$259	32	$8,288	
HP 36GB 15k 2.5 Single Port HP SAS Drive	1	431933-B21	$349	4	$1,396	
HP 3year 4h 24x7 Proliant DL785 HW Support	1	HA104A3 Opt.6XP	$3,208	1		$3,208
HP s7540 17" CRT Monitor	1	PF997AA#ABA	$139	1	$139	
HP USB Keyboard	1	GM321AA#ABA	$14	1	$14	
HP Optical 4 Button USB Mouse	1	EW208AA#ABA	$7	1	$7	
			Subtotal		**$58,841**	**$3,208**
Storage						
HP 5642 Unassembled Rack	1	358254-B21	$865	1	$865	
HP Smart Array Controller P800	1	381513-B21	$949	6	$5,694	
Storage MSA70	1	418800-B21	$3,199	12	$38,388	
Storage MSA70 10% Spares	1	418800-B21	$3,199	2	$6,398	
HP ext Mini SAS 4m Cable	1	432238-B21	$142	12	$1,704	
HP 72GB 15k 2.5 Single Port HP SAS Drive	1	431935-B21	$359	194	$69,646	
HP 72GB 15k 2.5 Single Port HP SAS Drive 10% spares	1	431935-B21	$359	20	$7,180	
			Subtotal		**$129,875**	**$0**
Server Software						
Microsoft Windows Server 2008 Enterprise Edition (x64)	2	P72-03168	$2,310	1	$2,310	
Microsoft SQL Server 2008 Enterprise x64 Edition with 25 CALs	2	810-07578	$8,318	1	$8,318	$245
Microsoft SQL Server 2008 Client License	2	359-01912	$156	35	$5,460	
			Subtotal		**$16,088**	**$245**
		Total Extended Price			**$204,804**	**$3,453**
		Total Discounts			**$30,195**	**$513**
Price Key: 1 - HP , 2 - Microsoft, Audited by Lorna Livingtree of Performance Metrics, Inc		**Grand Total**			**$174,609**	**$2,940**

All discounts are based on US list prices and for similar quantities and configurations. A 16% discount was based on the overall specific components pricing from vendor 1 in this single quotation. Discounts for similarly sized configurations will be similar to those quoted here, but may vary based on the components in the configuration.

3 year cost of ownership USD:	**$177,550**
QphH @ 300GB:	**91,558.2**
$ USD/QphH @ 300GB	**$1.94**

Sales contact: HP Sales Development, 19111 Pruneridge Ave., Cupertino, CA 95014 (408) 447 2320 or HP direct: 800-203-6748

Prices used in TPC benchmarks reflect the actual prices a customer would pay for a one-time purchase of the stated components. Individually negotiated discounts are not permitted. Special prices based on assumptions about past or future purchases are not permitted. All discounts reflect standard pricing policies for the listed components. For complete details, see the pricing sections of the TPC benchmark specifications. If you find that the stated prices are not available according to these terms, please inform at pricing@tpc.org. Thank you.

© 2009 Hewlett-Packard Company. All rights reserved.

Numerical Quantities

Measurement Results

Database Scale Factor	300GB
Total Data Storage / Database Size	44.66
Start of Database Load	07/11/09 14:38:47
End of Database Load	07/12/09 03:58:37
Database Load Time	13h19m50s
Query Streams for Throughput Test	6
TPC-H Power	109,067.1
TPC-H Throughput	76,860.0
TPC-H Composite Query-per-Hour (QphH@300GB)	91,558.2
Total System Price over 3 Years	$177,550
TPC-H Price/Performance Metric (USD / QphH@300GB)	$1.94 / QphH@300GB

Measurement Interval

Measurement Interval in Throughput Test (Ts) = seconds 1855 seconds

Duration of Stream Execution

Power Stream	Seed	RF1 Start Time RF1 End Time	Query Start Time Query End Time	RF2 Start Time RF2 End Time	Duration
	712035837	07/12/09 10:07:28	07/12/09 10:08:18	07/12/09 10:13:52	0:06:55
		07/12/09 10:08:05	07/12/09 10:13:52	07/12/09 10:14:23	

Throughput Stream	Seed	Query Start Time Query End Time	Duration	RF1 Start Time RF1 End Time	RF2 Start Time RF2 End Time
1	712035838	07/12/09 10:14:43	0:26:34	07/12/09 10:40:20	07/12/09 10:40:46
		07/12/09 10:39:50		07/12/09 10:40:46	07/12/09 10:41:17
2	712035839	07/12/09 10:14:43	0:27:32	07/12/09 10:41:18	07/12/09 10:41:47
		07/12/09 10:39:54		07/12/09 10:41:46	07/12/09 10:42:14
3	712035840	07/12/09 10:14:43	0:28:25	07/12/09 10:42:15	07/12/09 10:42:41
		07/12/09 10:39:06		07/12/09 10:42:40	07/12/09 10:43:08
4	712035841	07/12/09 10:14:43	0:29:15	07/12/09 10:43:09	07/12/09 10:43:32
		07/12/09 10:40:19		07/12/09 10:43:32	07/12/09 10:43:58
5	712035842	07/12/09 10:14:43	0:30:05	07/12/09 10:43:59	07/12/09 10:44:23
		07/12/09 10:39:51		07/12/09 10:44:22	07/12/09 10:44:48
6	712035843	07/12/09 10:14:43	0:30:54	07/12/09 10:44:49	07/12/09 10:45:13
		07/12/09 10:39:34		07/12/09 10:45:12	07/12/09 10:45:38

© 2009 Hewlett-Packard Company. All rights reserved.

TPC-H Timing Intervals (in seconds):

Query	Stream 0	Stream 1	Stream 2	Stream 3	Stream 4	Stream 5	Stream 6	Min Qi	Max Qi	Avg. Qi
Q1	49.0	285.0	287.3	294.7	264.5	290.7	297.3	264.5	297.3	286.6
Q2	2.2	5.5	9.1	10.4	7.1	14.9	11.9	5.5	14.9	9.8
Q3	5.6	24.4	21.2	26.5	6.9	28.1	23.3	6.9	28.1	21.7
Q4	3.8	17.2	16.0	17.2	20.8	22.0	14.8	14.8	22.0	18.0
Q5	9.2	43.6	33.8	41.8	34.6	29.5	37.6	29.5	43.6	36.8
Q6	3.5	30.4	12.6	15.1	12.0	18.3	11.1	11.1	30.4	16.6
Q7	8.4	51.3	36.1	40.3	32.1	45.7	39.5	32.1	51.3	40.8
Q8	9.7	33.6	48.4	44.7	46.8	36.7	42.8	33.6	48.4	42.2
Q9	39.9	159.3	148.6	161.8	317.8	143.6	166.9	143.6	317.8	183.0
Q10	5.3	17.3	20.2	18.6	18.3	22.1	19.3	17.3	22.1	19.3
Q11	13.6	40.4	32.6	34.8	35.3	32.5	46.1	32.5	46.1	37.0
Q12	9.9	54.6	38.2	58.5	59.2	57.5	68.6	38.2	68.6	56.1
Q13	15.0	65.4	74.0	63.9	67.7	80.5	72.1	63.9	80.5	70.6
Q14	1.8	7.7	7.1	15.5	9.4	8.4	8.6	7.1	15.5	9.5
Q15	2.2	12.2	11.5	9.5	11.1	11.3	11.5	9.5	12.2	11.2
Q16	9.2	52.2	50.9	43.0	46.3	43.3	47.5	43.0	52.2	47.2
Q17	2.6	18.4	22.8	14.0	19.7	23.4	24.5	14.0	24.5	20.5
Q18	62.4	194.7	284.4	183.1	182.9	183.9	166.1	166.1	284.4	199.2
Q19	11.6	32.1	19.2	30.2	50.7	48.5	45.3	19.2	50.7	37.7
Q20	8.0	47.1	46.6	48.0	8.2	44.9	44.3	8.2	48.0	39.9
Q21	51.8	275.9	247.9	241.4	275.1	276.3	250.9	241.4	276.3	261.3
Q22	9.6	38.9	42.9	49.9	10.0	46.5	40.7	10.0	49.9	38.2
UF1	37.3	26.0	28.1	24.9	23.0	23.4	23.4	23.0	28.1	24.8
UF2	30.2	30.7	27.9	26.8	25.5	25.2	24.9	24.9	30.7	26.8

© 2009 Hewlett-Packard Company. All rights reserved.

Abstract iii
Overview iii
Standard and Executive Summary Statements iii
Auditor iii
1.0 General Items 10
1.1 Test Sponsor 10
1.2 Parameter Settings 10
1.3 Configuration Items 10
2.0 Clause 1: Logical Database Design 12
2.1 Table Definitions 12
2.2 Physical Organization of Database 12
2.3 Horizontal Partitioning 12
2.4 Replication 12
3.0 Clause 2: Queries and Refresh Functions - Related Items 13
3.1 Query Language 13
3.2 Random Number Generation 13
3.3 Substitution Parameters Generation 13
3.4 Query Text and Output Data from Database 13
3.5 Query Substitution Parameters and Seeds Used 13
3.6 Isolation Level 13
3.7 Refresh Functions 13
4.0 Clause 3: Database System Properties 14
4.1 Atomicity Requirements 14
4.1.1 Atomicity of the Completed Transactions 14
4.1.2 Atomicity of Aborted Transactions 14
4.2 Consistency Requirements 14
4.3 Isolation Requirements 15
4.4 Durability Requirements 17
5.0 Clause 4: Scaling and Database Population 18
5.1 Initial Cardinality of Tables 18
5.2 Distribution of Tables and Logs Across Media 18
5.3 Mapping of Database Partitions/Replications 20
5.4 Implementation of RAID 20
5.5 DBGEN Modifications 20
5.6 Database Load time 20
5.7 Data Storage Ratio 20
5.8 Database Load Mechanism Details and Illustration 20
5.9 Qualification Database Configuration 21
5.10 Dataset Verification 21
6.0 Clause 5: Performance Metrics and Execution Rules Related Items 22
6.1 Steps in the Power Test 22
6.2 Timing Intervals for Each Query and Refresh Function 22
6.3 Number of Streams for The Throughput Test 22
6.4 Start and End Date/Times for Each Query Stream 22
6.5 Total Elapsed Time for the Measurement Interval 22
6.6 Refresh Function Start Date/Time and Finish Date/Time 22
6.7 Timing Intervals for Each Query and Each Refresh Function for Each Stream 23
6.8 Performance Metrics 23
6.9 The Performance Metric and Numerical Quantities from Both Runs 23

© 2009 Hewlett-Packard Company. All rights reserved.

6.10 System Activity Between Tests 23
7.0 Clause 6: SUT and Driver Implementation Related Items 24
7.1 Driver 24
7.2 Implementation Specific Layer (ISL) 24
7.3 Profile-Directed Optimization 25
8. Clause 7: Pricing Related Items 26
8.1 Hardware and Software Used 26
8.2 Three-Year Cost of System Configuration 26
8.3 Availability Dates 26
9. Clause 8: Audit Related Items 27
9.1 Auditors' Report 27
Appendix A: Tunable Parameters 30
Appendix B: Database Build Scripts 87
Appendix C: Query Text and Output 91
Appendix D: Seeds and Query Substitution Parameters 98
Appendix E: Refresh Function Source Code 100
Appendix F: Implementation Specific Layer and Source Code 112
Appendix G: Price Quotations 466

© 2009 Hewlett-Packard Company. All rights reserved.

1.0 General Items

1.1 Test Sponsor

A statement identifying the benchmark sponsor(s) and other participating companies must be provided.

This benchmark was sponsored by Hewlett-Packard Company. The benchmark was developed and engineered by Hewlett-Packard Company. Testing took place at Microsoft facilities in Redmond, Washington.

1.2 Parameter Settings

Settings must be provided for all customer-tunable parameters and options which have been changed from the defaults found in actual products, including by not limited to:

- *Database Tuning Options*
- *Optimizer/Query execution options*
- *Query processing tool/language configuration parameters*
- *Recovery/commit options*
- *Consistency/locking options*
- *Operating system and configuration parameters*
- *Configuration parameters and options for any other software component incorporated into the pricing structure*
- *Compiler optimization options*

This requirement can be satisfied by providing a full list of all parameters and options, as long as all those which have been modified from their default values have been clearly identified and these parameters and options are only set once.

Appendix A, “Tunable Parameters,” contains a list of all database parameters and operating system parameters.

1.3 Configuration Items

Diagrams of both measured and priced configurations must be provided, accompanied by a description of the differences. This includes, but is not limited to:

- *Number and type of processors*
- *Size of allocated memory, and any specific mapping/partitioning of memory unique to the test.*
- *Number and type of disk units (and controllers, if applicable).*
- *Number of channels or bus connections to disk units, including their protocol type.*
- *Number of LAN (e.g. Ethernet) Connections, including routers, workstations, terminals, etc., that were physically used in the test or are incorporated into the pricing structure.*
- *Type and the run-time execution location of software components (e.g., DBMS, query processing tools /languages, middle-ware components, software drivers, etc.).*

© 2009 Hewlett-Packard Company. All rights reserved.

The System Under Test (SUT), an HP ProLiant DL785 G6, depicted in Figure 1.1, consisted of :

8 AMD Opteron™ 8439 Quad-Core 2.8GHz
256 GB RAM
6 P800 Smart Array Controllers
12 HP StorageWorks MSA 70 Enclosures
194 72GB Pluggable SAS Disks
4 36GB Pluggable SAS Disks

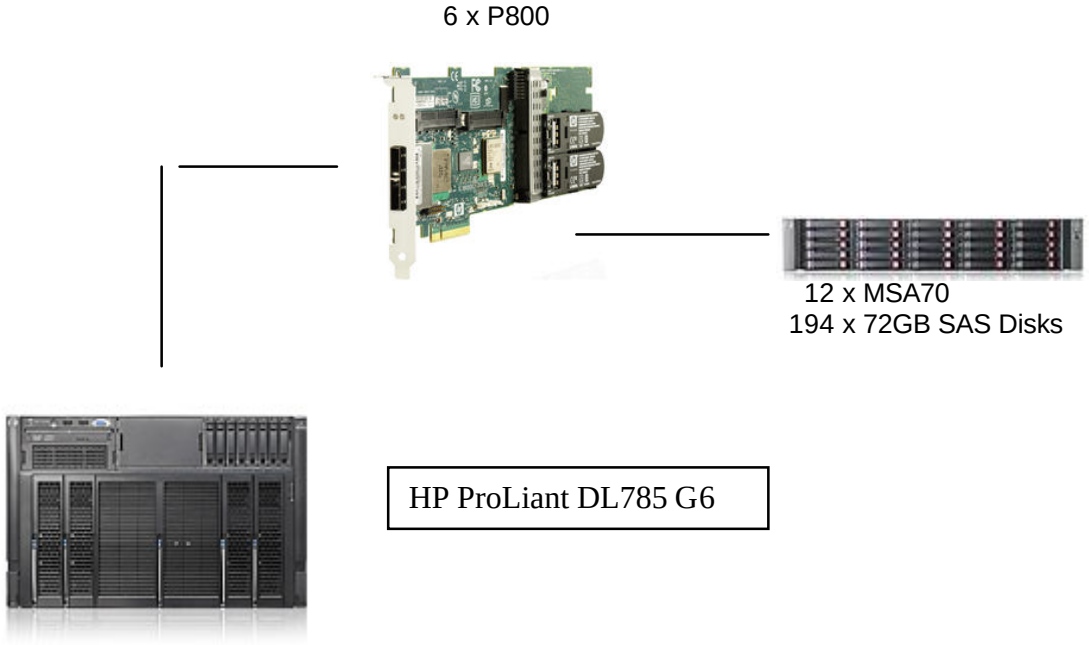


Figure 1.1 Benchmark and priced configuration for HP ProLiant DL785 G6

© 2009 Hewlett-Packard Company. All rights reserved.

2.0 Clause 1: Logical Database Design

2.1 Table Definitions

Listings must be provided for all table definition statements and all other statements used to set up the test and qualification databases. (8.1.2.1)

Appendix B, “Database Build Scripts,” contains the table definitions and the program used to load the database.

2.2 Physical Organization of Database

The physical organization of tables and indices, within the test and qualification databases, must be disclosed. If the column ordering of any table is different from that specified in Clause 1.4, it must be noted.

Appendix B, “Database Build Scripts,” contains the DDL for the index definitions.

2.3 Horizontal Partitioning

Horizontal partitioning of tables and rows in the test and qualification databases (see Clause 1.5.4) must be disclosed.

Horizontal partitioning was not used

2.4 Replication

Any replication of physical objects must be disclosed and must conform to the requirements of Clause 1.5.6.

No replication was used.

© 2009 Hewlett-Packard Company. All rights reserved.

3.0 Clause 2: Queries and Refresh Functions - Related Items

3.1 Query Language

The query language used to implement the queries must be identified.

T-SQL was the query language used.

3.2 Random Number Generation

The method of verification for the random number generation must be described unless the supplied DBGEN and QGEN were used.

DBGEN version 2.8.0 and QGEN version 2.8.0 were used to generate all database populations.

3.3 Substitution Parameters Generation

The method used to generate values for substitution parameters must be disclosed. If QGEN is not used for this purpose, then the source code of any non-commercial tool used must be disclosed. If QGEN is used, the version number, release number, modification number and patch level of QGEN must be disclosed.

The TPC source based QGEN version 2.8.0 was used to generate the substitution parameters

3.4 Query Text and Output Data from Database

The executable query text used for query validation must be disclosed along with the corresponding output data generated during the execution of the query text against the qualification database. If minor modifications (see Clause 2.2.3) have been applied to any functional query definitions or approved variants in order to obtain executable query text, these modifications must be disclosed and justified. The justification for a particular minor query modification can apply collectively to all queries for which it has been used. The output data for the power and throughput tests must be made available electronically upon request..

Appendix C contains the query text and query output. The following modifications were used:

- The “dateadd” function is used to perform date arithmetic in Q1, Q4, Q5, Q6, Q10, Q12, Q14 , Q15 and Q20.
- The “datepart” function is used to extract part of a date (“YY”) in Q7, Q8 and Q9.
- The “top” function is used to restrict the number of output rows in Q2, Q3, Q10, Q18 and Q21.

3.5 Query Substitution Parameters and Seeds Used

All the query substitution parameters used during the performance test must be disclosed in tabular format, along with the seeds used to generate these parameters.

Appendix D contains the seed and query substitution parameters used.

3.6 Isolation Level

The isolation level used to run the queries must be disclosed. If the isolation level does not map closely to one of the isolation levels defined in Clause 3.4, additional descriptive detail must be provided.

The queries and transactions were run with isolation level Read Committed.

3.7 Refresh Functions

The details of how the refresh functions were implemented must be disclosed

Appendix E contains the source code for the refresh functions.

© 2009 Hewlett-Packard Company. All rights reserved.

4.0 Clause 3: Database System Properties

4.1 Atomicity Requirements

The results of the ACID tests must be disclosed along with a description of how the ACID requirements were met. This includes disclosing the code written to implement the ACID Transaction and Query.

All ACID tests were conducted according to specification. The steps performed are outlined below.

4.1.1 Atomicity of the Completed Transactions

Perform the ACID Transaction for a randomly selected set of input data and verify that the appropriate rows have been changed in the ORDER, LINEITEM, and HISTORY tables.

The following steps were performed to verify the Atomicity of completed transactions:

1. The total price from the ORDER table and the extended price from the LINEITEM table were retrieved for a randomly selected order key.
2. The ACID Transaction was performed using the order key from step 1.
3. The ACID Transaction committed.
4. The total price from the ORDER table and the extended price from the LINEITEM table were retrieved for the same order key. It was verified that the appropriate rows had been changed.

4.1.2 Atomicity of Aborted Transactions

Perform the ACID transaction for a randomly selected set of input data, submitting a ROLLBACK of the transaction for the COMMIT of the transaction. Verify that the appropriate rows have not been changed in the ORDER, LINEITEM, and HISTORY tables.

The following steps were performed to verify the Atomicity of the aborted ACID transaction:

1. The total price from the ORDER table and the extended price from the LINEITEM table were retrieved for a randomly selected order key.
2. The ACID Transaction was performed using the order key from step 1. The transaction was stopped prior to the commit.
3. The ACID Transaction was ROLLED BACK. .
4. The total price from the ORDER table and the extended price from the LINEITEM table were retrieved for the same order key used in steps 1 and 2. It was verified that the appropriate rows had not been changed.

4.2 Consistency Requirements

Consistency is the property of the application that requires any execution of transactions to take the database from one consistent state to another. A consistent state for the TPC-H database is defined to exist when:

*O_TOTALPRICE = SUM(trunc(trunc((L_EXTENDEDPRICE – L_DISCOUNT) * (1 + L_TAX)))*

for each ORDER and LINEITEM defined by (O_ORDERKEY = L_ORDERKEY)

4.2.1 Consistency Tests

Verify that ORDER and LINEITEM tables are initially consistent as defined in Clause 3.3.2.1, based upon a random sample of at least 10 distinct values of O_ORDERKEY.

The following steps were performed to verify consistency:

1. The consistency of the ORDER and LINEITEM tables was verified based on a sample of O_ORDERKEYs.
2. One hundred ACID Transactions were submitted from each of six execution streams.
3. The consistency of the ORDER and LINEITEM tables was re-verified.

© 2009 Hewlett-Packard Company. All rights reserved.

4.3 Isolation Requirements

Operations of concurrent transactions must yield results which are indistinguishable from the results which would be obtained by forcing each transaction to be serially executed to completion in some order.

4.3.1 Isolation Test 1 - Read-Write Conflict with Commit

Demonstrate isolation for the read-write conflict of a read-write transaction and a read-only transaction when the read-write transaction is committed)

The following steps were performed to satisfy the test of isolation for a read-only and a read-write committed transaction:

1. An ACID Transaction was started for a randomly selected O_KEY, L_KEY and DELTA. The ACID Transaction was suspended prior to Commit.
2. An ACID query was started for the same O_KEY used in step 1. The ACID query blocked and did not see any uncommitted changes made by the ACID Transaction.
3. The ACID Transaction was resumed and committed. The ACID query completed. It returned the data as committed by the ACID Transaction.

4.3.2 Isolation Test 2 - Read-Write Conflict with Rollback

Demonstrate isolation for the read-write conflict of a read-write transaction and a read-only transaction when the read-write transaction is rolled back.

The following steps were performed to satisfy the test of isolation for read-only and a rolled back read-write transaction:

1. An ACID transaction was started for a randomly selected O_KEY, L_KEY and DELTA. The ACID Transaction was suspended prior to Rollback.
2. An ACID query was started for the same O_KEY used in step 1. The ACID query did not see any uncommitted changes made by the ACID Transaction.
3. The ACID Transaction was ROLLED BACK.
4. The ACID query completed.

4.3.3 Isolation Test 3 - Write-Write Conflict with Commit

Demonstrate isolation for the write-write conflict of two update transactions when the first transaction is committed.

The following steps were performed to verify isolation of two update transactions:

1. An ACID Transaction T1 was started for a randomly selected O_KEY, L_KEY and DELTA. The ACID transaction T1 was suspended prior to Commit.
2. Another ACID Transaction T2 was started using the same O_KEY and L_KEY and a randomly selected DELTA.
3. T2 waited.
4. The ACID transaction T1 was allowed to Commit and T2 completed.
5. It was verified that: T2.L_EXTENDEDPRICE = T1.L_EXTENDEDPRICE +(DELTA1*(T1.L_EXTENDEDPRICE/T1.L_QUANTITY))

4.3.4 Isolation Test 4 - Write-Write Conflict with Rollback

Demonstrate isolation for the write-write conflict of two update transactions when the first transaction is rolled back.

The following steps were performed to verify the isolation of two update transactions after the first one is rolled back:

1. An ACID Transaction T1 was started for a randomly selected O_KEY, L_KEY and DELTA. The ACID Transaction T1 was suspended prior to Rollback.
2. Another ACID Transaction T2 was started using the same O_KEY and L_KEY used in step 1 and a randomly selected DELTA.
3. T2 waited.
4. T1 was allowed to ROLLBACK and T2 completed.

© 2009 Hewlett-Packard Company. All rights reserved.

5. It was verified that T2.L_EXTENDEDPRICE = T1.L_EXTENDEDPRICE.

4.3.5 Isolation Test 5 – Concurrent Read and Write Transactions on Different Tables

Demonstrate the ability of read and write transactions affecting different database tables to make progress concurrently.

The following steps were performed:

1. An ACID Transaction T1 for a randomly selected O_KEY, L_KEY and DELTA. The ACID Transaction T1 was suspended prior to Commit.
2. Another ACID Transaction T2 was started using random values for PS_PARTKEY and PS_SUPPKEY.
3. T2 completed.
4. T1 completed and the appropriate rows in the ORDER, LINEITEM and HISTORY tables were changed.

4.3.6 Isolation Test 6 – Update Transactions During Continuous Read-Only Query Stream

Demonstrate the continuous submission of arbitrary (read-only) queries against one or more tables of the database does not indefinitely delay update transactions affecting those tables from making progress.

The following steps were performed:

1. An ACID Transaction T1 was started, executing Q1 against the qualification database. The substitution parameter was chosen from the interval [0..2159] so that the query ran for a sufficient amount of time.
2. Before T1 completed, an ACID Transaction T2 was started using randomly selected values of O_KEY, L_KEY and DELTA.
3. T2 completed before T1 completed.
4. It was verified that the appropriate rows in the ORDER, LINEITEM and HISTORY tables were changed.

© 2009 Hewlett-Packard Company. All rights reserved.

4.4 Durability Requirements

The tested system must guarantee durability: the ability to preserve the effects of committed transactions and insure database consistency after recovery from any one of the failures listed in Clause 3.5.2.

4.4.1 Permanent Unrecoverable Failure of Any Durable Medium and Loss of System Power

Guarantee the database and committed updates are preserved across a permanent irrecoverable failure of any single durable medium containing TPC-H database tables or recovery log tables

The test database log was stored on a logical volume on a RAID-1 array of 2 physical disks. The tables for the test database were stored on 48 logical volumes on RAID-0 arrays. A backup of the test database was taken. The backup was spread across 48 logical volumes on RAID-5 arrays.

The tests were conducted on the qualification database. The qualification database was loaded in the same fashion as the test database but used fewer disks. The steps performed are shown below:

1. The complete database was backed up.
2. Seven streams of ACID transactions were started. Each stream executed a minimum of 100 transactions.
3. While the test was running, one of the disks from the RAID-1 based log was removed.
4. After it was determined that the test would still run with the loss of a log disk, one physical drive of a RAID-0 data volume was removed.
5. A checkpoint was issued to force a failure.
6. The seven streams of ACID transactions failed and recorded their number of committed transaction in the success files.
7. The database log was dumped to disk and the database was shut down.
8. The database and log disks were replaced with new disks and the RAID rebuild process started
9. When the RAID rebuild process finished a database restore was done.
10. A command was issued causing the database to run through its recovery.
11. The counts in the success files and the HISTORY table count were compared and were found to match.

4.4.2 System Crash

Guarantee the database and committed updates are preserved across an instantaneous interruption (system crash/system hang) in processing which requires the system to reboot to recover.

1. Seven streams of ACID transactions were started. Each stream executed a minimum of 100 transactions.
2. While the streams of ACID transactions were running a system hardware reset was issued.
3. The system rebooted and the database was restarted.
4. The database went through a recovery period.
5. The success file and the HISTORY table counts were compared and were found to match.

4.4.3 Memory Failure

Guarantee the database and committed updates are preserved across failure of all or part of memory (loss of contents).

See section 4.4.2

© 2009 Hewlett-Packard Company. All rights reserved.

5.0 Clause 4: Scaling and Database Population

5.1 Initial Cardinality of Tables

The cardinality (i.e., the number of rows) of each table of the test database, as it existed at the completion of the database load (see clause 4.2.5) must be disclosed.

Table 5.1 lists the TPC-H Benchmark defined tables and the row count for each table as they existed upon completion of the build.

TABLE	# of Rows
Lineitem	**1,799,989,091**
Orders	**450,000,000**
Partsupp	**240,000,000**
Part	**60,000,000**
Customer	**45,000,000**
Supplier	**3,000,000**
Nation	25
Region	5

Table 5.1 Initial Number of Rows

5.2 Distribution of Tables and Logs Across Media

The distribution of tables and logs across all media must be explicitly described for the tested and priced systems.

Microsoft SQL Server was configured on an HP ProLiant DL785 G6 with the following configuration:

6 P800 Smart Array Controllers
12 HP StorageWorks MSA 70 Enclosures
194 72GB SAS Disks
4 36GB SAS Disks

194 disks were used to hold table data, indexes, database log and the temporary database (TempDB).

The raw partitions for the database and the NTFS partition for the backup and flat files were mounted to a folder on the G: drive

G:\mnt\li
G:\mnt\gen
G:\mnt\temp
G:\mnt\ntfs

A description of distribution of database filegroups and log can be found in the Table below.

© 2009 Hewlett-Packard Company. All rights reserved.

Slot Controller/Channel	# of Disks	Array Fault Tolerance	Size in GB	Partition Format	Content
Slot 0	2	RAID 1	36	NTFS	OS, SQL
P400 (onboard)	2	RAID 1	36	NTFS	Tools, Mount Points
	2	RAID 1	72	NTFS	Logs
Slot 1 A/B	36	RAID 0	9 x 17	RAW	TPCH300G LI
P800				RAW	TPCH300G GEN
				RAW	TPCH300G TempDB
		RAID 5	9 x 197	NTFS	Backup, Flatfiles
Slot 5 A/B	36	RAID 0	9 x 17	RAW	TPCH300G LI
P800				RAW	TPCH300G GEN
				RAW	TPCH300G TempDB
		RAID 5	9 x 197	NTFS	Backup, Flatfiles
Slot 8 A/B	40	RAID 0	10 x 17	RAW	TPCH300G LI
P800				RAW	TPCH300G GEN
				RAW	TPCH300G TempDB
		RAID 5	10 x 197	NTFS	Backup, Flatfiles
Slot 9 A/B	24	RAID 0	6 x 17	RAW	TPCH300G LI
P800				RAW	TPCH300G GEN
				RAW	TPCH300G TempDB
		RAID 5	6 x 197	NTFS	Backup, Flatfiles
Slot 10 A/B	32	RAID 0	8 x 17	RAW	TPCH300G LI
P800				RAW	TPCH300G GEN
				RAW	TPCH300G TempDB
		RAID 5	8 x 197	NTFS	Backup, Flatfiles
Slot 11 A/B	24	RAID 0	6 x 17	RAW	TPCH300G LI
P800				RAW	TPCH300G GEN
				RAW	TPCH300G TempDB
		RAID 5	6 x 197	NTFS	Backup, Flatfiles

© 2009 Hewlett-Packard Company. All rights reserved.

5.3 Mapping of Database Partitions/Replications

The mapping of database partitions/replications must be explicitly described.

Database partitioning/replication was not used.

5.4 Implementation of RAID

Implementations may use some form of RAID to ensure high availability. If used for data, auxiliary storage (e.g. indexes) or temporary space, the level of RAID used must be disclosed for each device.

RAID 0 was used for database filegroups and TempDB, RAID 1 for database recovery logs, RAID 5 for the Backup drives.

5.5 DBGEN Modifications

The version number, release number, modification number, and patch level of DBGEN must be disclosed. Any modifications to the DBGEN (see Clause 4.2.1) source code must be disclosed. In the event that a program other than DBGEN was used to populate the database, it must be disclosed in its entirety.

DBGEN version 2.8.0 was used, no modifications were made.

5.6 Database Load time

The database load time for the test database (see clause 4.3) must be disclosed.

The database load time was 13h19m50sec

5.7 Data Storage Ratio

The data storage ratio must be disclosed. It is computed by dividing the total data storage of the priced configuration (expressed in GB) by the size chosen for the test database as defined in 4.1.3.1. The ratio must be reported to the nearest 1/100th, rounded up.

Disk Type	# of Disks	Total (GB)
36 GB 15K rpm SAS	4	135.68
72 GB 15K rpm SAS	194	13,263.08
Total		13,398.76

Size of test database: 300G Data Storage Ratio: 44.66

5.8 Database Load Mechanism Details and Illustration

The details of the database load must be disclosed, including a block diagram illustrating the overall process. Disclosure of the load procedure includes all steps, scripts, input and configuration files required to completely reproduce the test and qualification databases.

Flat files for each of the tables were created using DBGEN. The tables were loaded as depicted in Figure 5-8.

© 2009 Hewlett-Packard Company. All rights reserved.

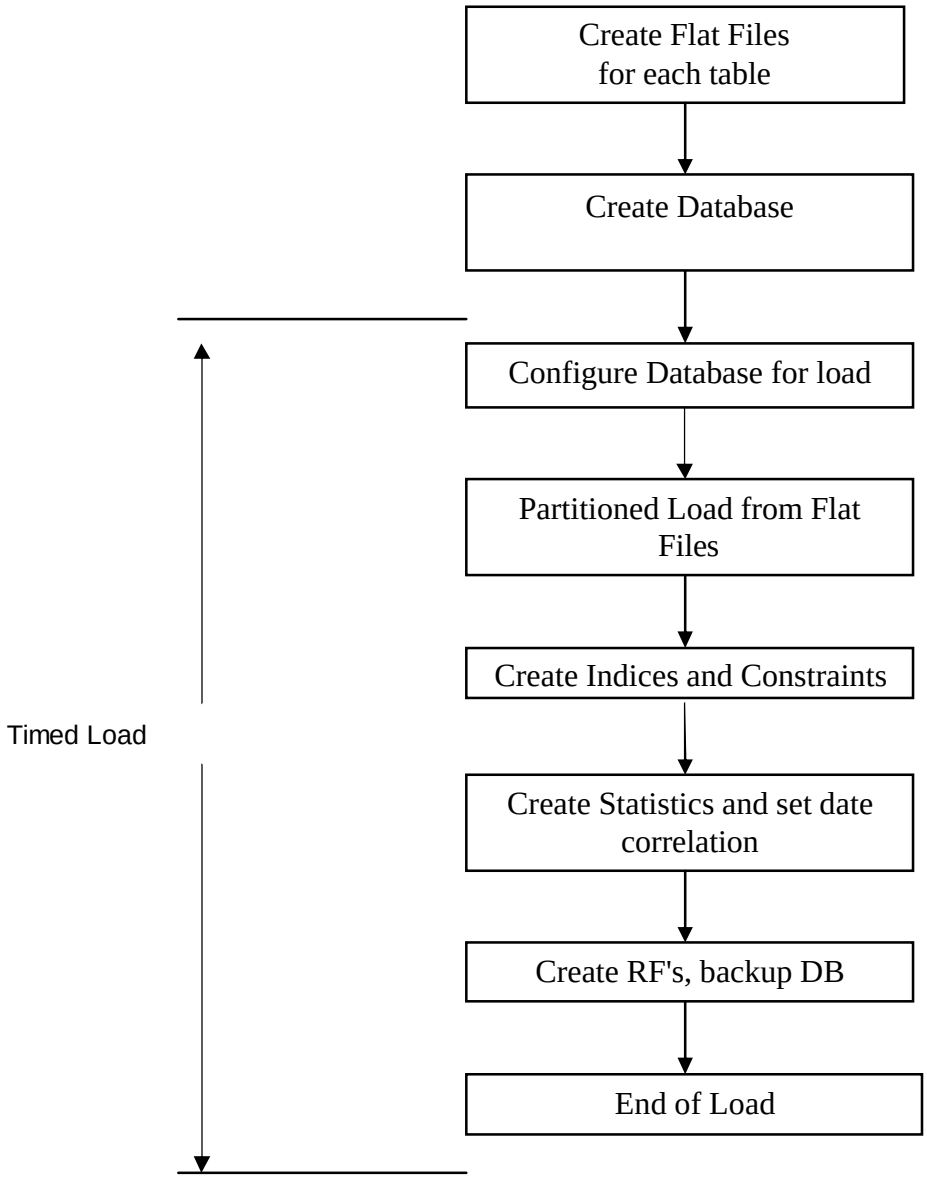


Figure 5.8: Block Diagram of Database Load Process

5.9 Qualification Database Configuration

The details of the database load must be disclosed, including a block diagram illustrating the overall process. Disclosure of the load procedure includes all steps, scripts, input and configuration files required to completely reproduce the test and qualification databases.

The qualification database used identical scripts to create and load the data with changes to adjust for the database scale factor.

5.10 Dataset Verification

Verify that the rows in the loaded database after the performance test are correct by comparing some small number of rows extracted at random from any two files of the corresponding Base, Insert and Delete reference data set files for each table and the corresponding rows of the database.

Verified according to the specification.

© 2009 Hewlett-Packard Company. All rights reserved.

6.0 Clause 5: Performance Metrics and Execution Rules Related Items

6.1 Steps in the Power Test

The details of the steps followed to implement the power test (e.g., system boot, database restart, etc.) must be disclosed.

The following steps were used to implement the power test:

1. RF1 Refresh Transaction
2. Stream 0 Execution
3. RF2 Refresh Transaction.

6.2 Timing Intervals for Each Query and Refresh Function

The timing intervals (see Clause 5.3.6) for each query of the measured set and for both refresh functions must be reported for the power test.

The timing intervals for each query and both refresh functions are given in the Numerical Quantities Summary earlier in this document on page vi.

6.3 Number of Streams for The Throughput Test

The number of execution streams used for the throughput test must be disclosed.

6 streams were used for the Throughput Test.

6.4 Start and End Date/Times for Each Query Stream

The start time and finish time for each query execution stream must be reported for the throughput test.

The Numerical Quantities Summary on page vi contains the start and stop times for the query execution streams run on the system reported.

6.5 Total Elapsed Time for the Measurement Interval

The total elapsed time of the measurement interval(see Clause 5.3.5) must be reported for the throughput test.

The Numerical Quantities Summary on page vi contains the timing intervals for the throughput test run on the system reported.

6.6 Refresh Function Start Date/Time and Finish Date/Time

Start and finish time for each update function in the update stream must be reported for the throughput test.

The Numerical Quantities Summary on page vi contains the start and finish times for the refresh functions of each stream.

© 2009 Hewlett-Packard Company. All rights reserved.

6.7 Timing Intervals for Each Query and Each Refresh Function for Each Stream

The timing intervals (see Clause 5.3.6) for each query of each stream and for each update function must be reported for the throughput test.

The timing intervals for each query and each update function are given in the Numerical Quantities Summary earlier in this document on page vii.

6.8 Performance Metrics

The computed performance metrics, related numerical quantities and the price performance metric must be reported.

The Numerical Quantities Summary contains the performance metrics, related numerical quantities, and the price/performance metric for the system reported.

6.9 The Performance Metric and Numerical Quantities from Both Runs

The performance metric (QphH@Size) and the numerical quantities (TPC-H Power@Size and TPC-H Throughput@ Size) from both of the runs must be disclosed

Run ID	QppH@300G	QthH@300G	QphH@300G
Run 1	109,067.1	76,860.0	91,558.2
Run 2	118,877.2	77,072.0	95,718.9

6.10 System Activity Between Tests

Any activity on the SUT that takes place between the conclusion of Run1 and the beginning of Run2 must be disclosed.

Auditor requested queries were run against the database and the log was truncated.

© 2009 Hewlett-Packard Company. All rights reserved.

7.0 Clause 6: SUT and Driver Implementation Related Items

7.1 Driver

A detailed description of how the driver performs its functions must be supplied, including any related source code or scripts. This description should allow an independent reconstruction of the driver.

The TPC-H benchmark was implemented using a Microsoft tool called StepMaster. StepMaster is a general purpose test tool which can drive ODBC and shell commands. Within StepMaster, the user designs a workspace corresponding to the sequence of operations (or steps) to be executed. When the workspace is executed, StepMaster records information about the run into a database as well as a log file for later analysis.

StepMaster provides a mechanism for creating parallel streams of execution. This is used in the throughput tests to drive the query and refresh streams. Each step is timed using a millisecond resolution timer. A timestamp T1 is taken before beginning the operation and a timestamp T2 is taken after completing the operation. These times are recorded in a database as well as a log file for later analysis.

Two types of ODBC connections are supported. A dynamic connection is used to execute a single operation and is closed when the operation finishes. A static connection is held open until the run completes and may be used to execute more than one step. A connection (either static or dynamic) can only have one outstanding operation at any time.

In TPC-H, static connections are used for the query streams in the power and throughput tests. StepMaster reads an Access database to determine the sequence of steps to execute. These commands are represented as the Implementation Specific Layer. StepMaster records its execution history, including all timings, in the Access database. Additionally, StepMaster writes a textual log file of execution for each run.

The refresh functions were executed using multiple batch scripts. The initial script is invoked by StepMaster, subsequent scripts are called from within the scripts.

The source code for both StepMaster and the RF Scripts is disclosed on Appendix E/F

7.2 Implementation Specific Layer (ISL)

If an implementation-specific layer is used, then a detailed description of how it performs its functions must be supplied, including any related source code or scripts. This description should allow an independent reconstruction of the implementation-specific layer.

The following steps are performed, to accomplish the Power and Throughput Runs:

1. Power Run

- Execute 96 concurrent RF1 threads, each of which will apply a segment of a refresh set generated by DBGen. Each thread submits multiple transactions, where a transaction spans a set of orders and their associated line items. A checkpoint was issued after RF1 completion.

- Execute the Stream 0 queries in the order according to TPC Benchmark H Specification, Appendix A

- Execute 96 concurrent RF2 threads, each of which will apply a segment of a refresh set generated by DBGen. Each thread submits multiple transactions, where a transaction spans a set of orders and their associated line items. A checkpoint was issued after RF2 completion.

© 2009 Hewlett-Packard Company. All rights reserved.

2. Throughput Run

- Execute 6 concurrent query streams. Each stream executes queries in the order according to TPC Benchmark H Specification, Appendix A, for the appropriate Stream ID (01-06). Upon completion of each stream, a semaphore is signaled to indicate completion.
- Execute 6 consecutive RF1/RF2 transactions, against ascending Refresh sets produced by DBGen. The first RF1 waits on a semaphore prior to beginning its insert operations.

Each step during the query execution is timed by StepMaster. The timing information, together with an activity log, are stored for later analysis. The inputs and results of steps are stored in text files for later analysis.

7.3 Profile-Directed Optimization

If profile-directed optimization as described in Clause 5.2.9 is used, such use must be disclosed.

Profile-directed optimization was not used.

© 2009 Hewlett-Packard Company. All rights reserved.

8. Clause 7: Pricing Related Items

8.1 Hardware and Software Used

A detailed list of hardware and software used in the priced system must be reported. Each item must have a vendor part number, description, and release/revision level, and indicate General Availability status or committed delivery date. If package pricing is used, contents of the package must be disclosed. Pricing source(s) and effective date(s) of price(s) must also be reported.

The pricing summary sheet is given on page *v* in the Executive Summary at the front of this report. The source for all prices is indicated.

The HP ProLiant DL785 G6 with additional processors will be available 10/30/2009
Disk controllers and hard drives are available at the time of publication.

The pricing and availability of the Microsoft software used is given in a quote from Microsoft, which is included in this report in Appendix H.

8.2 Three-Year Cost of System Configuration

The total 3-year price of the entire configuration must be reported, including: hardware, software, and maintenance charges. Separate component pricing is required.

The pricing summary sheet on page v in the front of this report contains all details.

8.3 Availability Dates

The committed delivery date for general availability (availability date) of products used in the priced calculations must be reported. When the priced system includes products with different availability dates, the single availability date reported on the first page of the executive summary must be the date by which all components are committed to being available. The full disclosure report must report availability dates individually for at least each of the categories for which a pricing subtotal must be provided (see Clause 7.3.1.4). All availability dates, whether for individual components or for the SUT as a whole, must be disclosed to a precision of 1 day, but the precise format is left to the test sponsor.

Category	Available
Server Hardware	10/30/2009
Storage	Now (date of publication)
Server Software	Now (date of publication)
SQL Server	Now (date of publication)

© 2009 Hewlett-Packard Company. All rights reserved.

9. Clause 8: Audit Related Items

9.1 Auditors' Report

The auditor's agency name, address, phone number, and Attestation letter with a brief audit summary report indicating compliance must be included in the full disclosure report. A statement should be included specifying who to contact in order to obtain further information regarding the audit process.

This implementation of the TPC Benchmark H was audited by Lorna Livingtree of Performance Metrics, a certified TPC-H auditor. Further information regarding the audit process may be obtained from:

Lorna Livingtree
Performance Metrics Inc.
PO Box 984
Klamath, CA 95548
(707) 482 0523

Requests for this TPC Benchmark H Full Disclosure Report should be sent to:

Transaction Processing Performance Council
c/o Shanley Public Relations
777 North First Street, Suite 6000
San Jose, CA 95112-6311 USA
Telephone: (408) 295-8894
Fax: (408) 295-9768

© 2009 Hewlett-Packard Company. All rights reserved.

PERFORMANCE METRICS INC.
TPC Certified Auditors

July 20, 2009

Mr. Mike Fitzner
Senior Performance Engineer
Hewlett-Packard Company
14475 24th Street NE
Bellevue, WA 98007

I have verified the TPC Benchmark™ H for the following configuration:

Platform: HP Proliant DL785
Database Manager: Microsoft SQL Server 2008 Enterprise Edition SP1 for x64
Operating System: Microsoft Windows Server 2008 SP1 for x64

CPU's	Memory	Total Disks	Qpph@300GB	QthH@300GB	QphH@300GB
8 AMD hex core @ 2.8 Ghz	256 GB	194 @ 72GB 4 @ 36GB	**109,067.1**	**76,860.0**	**91,558.2**

In my opinion, these performance results were produced in compliance with the TPC requirements for the benchmark. The following attributes of the benchmark were given special attention:

- The database tables were defined with the proper columns, layout and sizes.
- The tested database was correctly scaled and populated for 300 GB using DBGEN. The version of DBGEN was 2.8.0.
- The sample data produce by DBGEN was successfully compared to the reference data for this scale factor.
- The qualification database layout was identical to the tested database except for the number and size of the files.
- The query text was verified to use only compliant variants and minor modifications.
- The executable query text was generated by QGEN and submitted through SQL Server's standard interactive interface. The version of QGEN was 2.8.0. The sample

PO Box 984 Klamath, CA 95548
(707) 482-0523 fax: (707) 482-0575

email: LornaL@PerfMetrics.com

© 2009 Hewlett-Packard Company. All rights reserved.

parameters produced by QGEN were successfully compared to the reference data for this scale factor.

- The validation of the query text against the qualification database produced compliant results.
- The refresh functions were properly implemented and executed the correct number of inserts and deletes.
- The load timing was properly measured and reported.
- The execution times were correctly measured and reported.
- The performance metrics were correctly computed and reported.
- The repeatability of the measurement was verified.
- The ACID properties were demonstrated and verified.
- The system pricing was checked for major components and maintenance.
- The executive summary pages of the FDR were verified for accuracy.

Auditor's Notes: None

Sincerely,

Lorna Livingtree

Lorna Livingtree
Auditor

PO Box 984 Klamath, CA 95548
(707) 482-0523 fax: (707) 482-0575

email: LornaL@PerfMetrics.com

© 2009 Hewlett-Packard Company. All rights reserved.

Appendix A: Tunable Parameters

System Info

System Information report written at:
07/14/09 11:20:31
System Name: OCTANE1
[System Summary]

Item Value
OS Name Microsoft® Windows Server® 2008 Enterprise
Version 6.0.6001 Service Pack 1 Build 6001
Other OS Description Not Available
OS Manufacturer Microsoft Corporation
System Name OCTANE1
System Manufacturer HP
System Model ProLiant DL785 G6
System Type x64-based PC
Processor Six-Core AMD Opteron(tm) Processor 8439 SE, 2800 Mhz, 6 Core(s), 6 Logical Processor(s)
Processor Six-Core AMD Opteron(tm) Processor 8439 SE, 2800 Mhz, 6 Core(s), 6 Logical Processor(s)
Processor Six-Core AMD Opteron(tm) Processor 8439 SE, 2800 Mhz, 6 Core(s), 6 Logical Processor(s)
Processor Six-Core AMD Opteron(tm) Processor 8439 SE, 2800 Mhz, 6 Core(s), 6 Logical Processor(s)
Processor Six-Core AMD Opteron(tm) Processor 8439 SE, 2800 Mhz, 6 Core(s), 6 Logical Processor(s)
Processor Six-Core AMD Opteron(tm) Processor 8439 SE, 2800 Mhz, 6 Core(s), 6 Logical Processor(s)
Processor Six-Core AMD Opteron(tm) Processor 8439 SE, 2800 Mhz, 6 Core(s), 6 Logical Processor(s)
BIOS Version/Date HP A15, 6/23/2009
SMBIOS Version 2.5
Windows Directory E:\Windows
System Directory E:\Windows\system32
Boot Device\Device\HarddiskVolume56
Locale United States
Hardware Abstraction Layer Version = "6.0.6001.18000"
User Name OCTANE1\Administrator
Time Zone Pacific Daylight Time
Installed Physical Memory (RAM) 256 GB
Total Physical Memory 4.00 GB
Available Physical Memory 692 MB
Total Virtual Memory 252 GB
Available Virtual Memory 3.84 GB
Page File Space 4.00 GB
Page File E:\pagefile.sys

[Hardware Resources]

[Conflicts/Sharing]

Resource Device
Memory Address 0xFD800000-0xFDBFFFFF PCI bus
Memory Address 0xFD800000-0xFDBFFFFF ServerWorks (Broadcom) HT-2100 HT-PCI-E Bridge 0

IRQ 41 HP iLO Management Channel Interface Driver
IRQ 41 Standard Universal PCI to USB Host Controller

I/O Port 0x00000000-0x000003AF PCI bus
I/O Port 0x00000000-0x000003AF Direct memory access controller

Memory Address 0xFDC00000-0xFDFFFFFF PCI bus
Memory Address 0xFDC00000-0xFDFFFFFF ServerWorks (Broadcom) HT-2100 HT-PCI-E Bridge 0

I/O Port 0x000003C0-0x000003DF PCI bus
I/O Port 0x000003C0-0x000003DF ATI ES1000

IRQ 5 Standard OpenHCD USB Host Controller
IRQ 5 Standard OpenHCD USB Host Controller
IRQ 5 Standard Enhanced PCI to USB Host Controller

Memory Address 0xE8000000-0xFD5FFFFF PCI bus
Memory Address 0xE8000000-0xFD5FFFFF ATI ES1000

I/O Port 0x00009000-0x0000FFFF PCI bus
I/O Port 0x00009000-0x0000FFFF ServerWorks (Broadcom) HT-2100 HT-PCI-E Bridge 0

I/O Port 0x00006000-0x00006FFF PCI bus
I/O Port 0x00006000-0x00006FFF ServerWorks (Broadcom) HT-2100 HT-PCI-E Bridge 0

Memory Address 0xFD600000-0xFD7FFFFF PCI bus
Memory Address 0xFD600000-0xFD7FFFFF ServerWorks (Broadcom) HT-2100 HT-PCI-E Bridge 0

I/O Port 0x000000A0-0x000000BF Motherboard resources
I/O Port 0x000000A0-0x000000BF Programmable interrupt controller

Memory Address 0xA0000-0xBFFFF PCI bus
Memory Address 0xA0000-0xBFFFF ATI ES1000

I/O Port 0x00007000-0x00008FFF PCI bus
I/O Port 0x00007000-0x00008FFF ServerWorks (Broadcom) HT-2100 HT-PCI-E Bridge 0

Memory Address 0xF8000000-0xFBFFFFFF ServerWorks (Broadcom) HT-1000 HT-PCI-X Bridge 0
Memory Address 0xF8000000-0xFBFFFFFF ServerWorks (Broadcom) HT-1000 HT-PCI-X Bridge
Memory Address 0xF8000000-0xFBFFFFFF Broadcom BCM5706C NetXtreme II GigE

I/O Port 0x000003B0-0x000003BB PCI bus
I/O Port 0x000003B0-0x000003BB ATI ES1000

I/O Port 0x00001000-0x00005FFF PCI bus
I/O Port 0x00001000-0x00005FFF ATI ES1000

I/O Port 0x00000600-0x0000067F Motherboard resources
I/O Port 0x00000600-0x0000067F Extended IO Bus

I/O Port 0x00000020-0x0000003F Motherboard resources
I/O Port 0x00000020-0x0000003F Programmable interrupt controller

[DMA]

Resource Device Status
Channel 7 Direct memory access controller OK

[Forced Hardware]

Device PNP Device ID

[I/O]

Resource Device Status
0x00001000-0x00005FFF PCI busOK
0x00001000-0x00005FFF ATI ES1000 OK
0x00000000-0x000003AF PCI busOK
0x00000000-0x000003AF Direct memory access controller OK
0x000003E0-0x00000CF7 PCI busOK
0x00000D00-0x00000FFF PCI busOK
0x000003B0-0x000003BB PCI busOK
0x000003B0-0x000003BB ATI ES1000 OK
0x000003C0-0x000003DF PCI busOK
0x000003C0-0x000003DF ATI ES1000 OK
0x00002800-0x000028FF HP ProLiant iLO 2 Legacy Support Function OK

© 2009 Hewlett-Packard Company. All rights reserved.

0x00001400-0x000014FF HP iLO Management Channel Interface Driver OK
0x00001800-0x0000181F Standard Universal PCI to USB Host Controller OK
0x00000500-0x0000050F Standard Dual Channel PCI IDE Controller OK
0x000001F0-0x000001F7 ATA Channel 0 OK
0x000003F6-0x000003F6 ATA Channel 0 OK
0x00000170-0x00000177 ATA Channel 1 OK
0x00000376-0x00000376 ATA Channel 1 OK
0x00000010-0x0000001F Motherboard resources OK
0x00000020-0x0000003F Motherboard resources OK
0x00000020-0x0000003F Programmable interrupt controller OK
0x000000A0-0x000000BF Motherboard resources OK
0x000000A0-0x000000BF Programmable interrupt controller OK
0x00000050-0x00000053 Motherboard resources OK
0x00000070-0x00000079 Motherboard resources OK
0x00000090-0x0000009F Motherboard resources OK
0x000000F0-0x000000F0 Motherboard resources OK
0x00000379-0x0000037A Motherboard resources OK
0x00000408-0x0000040F Motherboard resources OK
0x0000040B-0x0000040B Motherboard resources OK
0x000004D0-0x000004D1 Motherboard resources OK
0x000004D6-0x000004D6 Motherboard resources OK
0x00000520-0x00000520 Motherboard resources OK
0x00000580-0x0000059F Motherboard resources OK
0x00000600-0x0000067F Motherboard resources OK
0x00000600-0x0000067F Extended IO Bus OK
0x00000700-0x00000703 Motherboard resources OK
0x00000820-0x0000082F Motherboard resources OK
0x00000900-0x000009FE Motherboard resources OK
0x00000C06-0x00000C07 Motherboard resources OK
0x00000C14-0x00000C14 Motherboard resources OK
0x00000C4A-0x00000C4A Motherboard resources OK
0x00000C50-0x00000C52 Motherboard resources OK
0x00000C6C-0x00000C6C Motherboard resources OK
0x00000C6F-0x00000C6F Motherboard resources OK
0x00000C80-0x00000C83 Motherboard resources OK
0x00000C90-0x00000C9F Motherboard resources OK
0x00000CA0-0x00000CA5 Motherboard resources OK
0x00000CD4-0x00000CD7 Motherboard resources OK
0x00000F50-0x00000F58 Motherboard resources OK
0x000002F8-0x000002FF Motherboard resources OK
0x00000CA2-0x00000CA3 Microsoft Generic IPMI Compliant Device OK
0x00000040-0x00000043 System timer OK
0x00000080-0x0000008F Direct memory access controller OK
0x000000C0-0x000000DF Direct memory access controller OK
0x00000061-0x00000061 System speakerOK
0x00000C00-0x00000C01 Programmable interrupt controller OK
0x00000060-0x00000060 Standard PS/2 Keyboard OK
0x00000064-0x00000064 Standard PS/2 Keyboard OK
0x0000002E-0x0000002F Extended IO Bus OK
0x0000004E-0x0000004F Extended IO Bus OK
0x00000620-0x0000065F Extended IO Bus OK
0x00000680-0x0000069F Extended IO Bus OK
0x00000660-0x0000067F Extended IO Bus OK
0x00000300-0x0000030F Extended IO Bus OK
0x000003F8-0x000003FF Communications Port (COM1) OK
0x00001C00-0x00001CFF Standard OpenHCD USB Host Controller OK
0x00003000-0x000030FF Standard OpenHCD USB Host Controller OK
0x00003400-0x000034FF Standard Enhanced PCI to USB Host Controller OK
0x00005000-0x00005FFF ServerWorks (Broadcom) HT-2100 HT-PCI-E Bridge 0 OK
0x00004000-0x00004FFF ServerWorks (Broadcom) HT-2100 HT-PCI-E Bridge 1&3 OK
0x00006000-0x00006FFF PCI busOK
0x00006000-0x00006FFF ServerWorks (Broadcom) HT-2100 HT-PCI-E Bridge 0 OK
0x00007000-0x00008FFF PCI busOK
0x00007000-0x00008FFF ServerWorks (Broadcom) HT-2100 HT-PCI-E Bridge 0 OK
0x00008000-0x00008FFF ServerWorks (Broadcom) HT-2100 HT-PCI-E Bridge 1&3 OK
0x00009000-0x0000FFFF PCI busOK
0x00009000-0x0000FFFF ServerWorks (Broadcom) HT-2100 HT-PCI-E Bridge 0 OK
0x0000A000-0x0000AFFF ServerWorks (Broadcom) HT-2100 HT-PCI-E Bridge 1&3 OK

[IRQs]

Resource Device Status
IRQ 81 Microsoft ACPI-Compliant System OK
IRQ 82 Microsoft ACPI-Compliant System OK
IRQ 83 Microsoft ACPI-Compliant System OK
IRQ 84 Microsoft ACPI-Compliant System OK
IRQ 85 Microsoft ACPI-Compliant System OK
IRQ 86 Microsoft ACPI-Compliant System OK
IRQ 87 Microsoft ACPI-Compliant System OK
IRQ 88 Microsoft ACPI-Compliant System OK
IRQ 89 Microsoft ACPI-Compliant System OK
IRQ 90 Microsoft ACPI-Compliant System OK
IRQ 91 Microsoft ACPI-Compliant System OK
IRQ 92 Microsoft ACPI-Compliant System OK
IRQ 93 Microsoft ACPI-Compliant System OK
IRQ 94 Microsoft ACPI-Compliant System OK
IRQ 95 Microsoft ACPI-Compliant System OK
IRQ 96 Microsoft ACPI-Compliant System OK
IRQ 97 Microsoft ACPI-Compliant System OK
IRQ 98 Microsoft ACPI-Compliant System OK
IRQ 99 Microsoft ACPI-Compliant System OK
IRQ 100 Microsoft ACPI-Compliant System OK
IRQ 101 Microsoft ACPI-Compliant System OK
IRQ 102 Microsoft ACPI-Compliant System OK
IRQ 103 Microsoft ACPI-Compliant System OK
IRQ 104 Microsoft ACPI-Compliant System OK
IRQ 105 Microsoft ACPI-Compliant System OK
IRQ 106 Microsoft ACPI-Compliant System OK
IRQ 107 Microsoft ACPI-Compliant System OK
IRQ 108 Microsoft ACPI-Compliant System OK
IRQ 109 Microsoft ACPI-Compliant System OK
IRQ 110 Microsoft ACPI-Compliant System OK
IRQ 111 Microsoft ACPI-Compliant System OK
IRQ 112 Microsoft ACPI-Compliant System OK
IRQ 113 Microsoft ACPI-Compliant System OK
IRQ 114 Microsoft ACPI-Compliant System OK
IRQ 115 Microsoft ACPI-Compliant System OK
IRQ 116 Microsoft ACPI-Compliant System OK
IRQ 117 Microsoft ACPI-Compliant System OK

© 2009 Hewlett-Packard Company. All rights reserved.

IRQ 118 Microsoft ACPI-Compliant System OK
IRQ 119 Microsoft ACPI-Compliant System OK
IRQ 120 Microsoft ACPI-Compliant System OK
IRQ 121 Microsoft ACPI-Compliant System OK
IRQ 122 Microsoft ACPI-Compliant System OK
IRQ 123 Microsoft ACPI-Compliant System OK
IRQ 124 Microsoft ACPI-Compliant System OK
IRQ 125 Microsoft ACPI-Compliant System OK
IRQ 126 Microsoft ACPI-Compliant System OK
IRQ 127 Microsoft ACPI-Compliant System OK
IRQ 128 Microsoft ACPI-Compliant System OK
IRQ 129 Microsoft ACPI-Compliant System OK
IRQ 130 Microsoft ACPI-Compliant System OK
IRQ 131 Microsoft ACPI-Compliant System OK
IRQ 132 Microsoft ACPI-Compliant System OK
IRQ 133 Microsoft ACPI-Compliant System OK
IRQ 134 Microsoft ACPI-Compliant System OK
IRQ 135 Microsoft ACPI-Compliant System OK
IRQ 136 Microsoft ACPI-Compliant System OK
IRQ 137 Microsoft ACPI-Compliant System OK
IRQ 138 Microsoft ACPI-Compliant System OK
IRQ 139 Microsoft ACPI-Compliant System OK
IRQ 140 Microsoft ACPI-Compliant System OK
IRQ 141 Microsoft ACPI-Compliant System OK
IRQ 142 Microsoft ACPI-Compliant System OK
IRQ 143 Microsoft ACPI-Compliant System OK
IRQ 144 Microsoft ACPI-Compliant System OK
IRQ 145 Microsoft ACPI-Compliant System OK
IRQ 146 Microsoft ACPI-Compliant System OK
IRQ 147 Microsoft ACPI-Compliant System OK
IRQ 148 Microsoft ACPI-Compliant System OK
IRQ 149 Microsoft ACPI-Compliant System OK
IRQ 150 Microsoft ACPI-Compliant System OK
IRQ 151 Microsoft ACPI-Compliant System OK
IRQ 152 Microsoft ACPI-Compliant System OK
IRQ 153 Microsoft ACPI-Compliant System OK
IRQ 154 Microsoft ACPI-Compliant System OK
IRQ 155 Microsoft ACPI-Compliant System OK
IRQ 156 Microsoft ACPI-Compliant System OK
IRQ 157 Microsoft ACPI-Compliant System OK
IRQ 158 Microsoft ACPI-Compliant System OK
IRQ 159 Microsoft ACPI-Compliant System OK
IRQ 160 Microsoft ACPI-Compliant System OK
IRQ 161 Microsoft ACPI-Compliant System OK
IRQ 162 Microsoft ACPI-Compliant System OK
IRQ 163 Microsoft ACPI-Compliant System OK
IRQ 164 Microsoft ACPI-Compliant System OK
IRQ 165 Microsoft ACPI-Compliant System OK
IRQ 166 Microsoft ACPI-Compliant System OK
IRQ 167 Microsoft ACPI-Compliant System OK
IRQ 168 Microsoft ACPI-Compliant System OK
IRQ 169 Microsoft ACPI-Compliant System OK
IRQ 170 Microsoft ACPI-Compliant System OK
IRQ 171 Microsoft ACPI-Compliant System OK
IRQ 172 Microsoft ACPI-Compliant System OK
IRQ 173 Microsoft ACPI-Compliant System OK
IRQ 174 Microsoft ACPI-Compliant System OK
IRQ 175 Microsoft ACPI-Compliant System OK
IRQ 176 Microsoft ACPI-Compliant System OK
IRQ 177 Microsoft ACPI-Compliant System OK
IRQ 178 Microsoft ACPI-Compliant System OK
IRQ 179 Microsoft ACPI-Compliant System OK
IRQ 180 Microsoft ACPI-Compliant System OK
IRQ 181 Microsoft ACPI-Compliant System OK
IRQ 182 Microsoft ACPI-Compliant System OK
IRQ 183 Microsoft ACPI-Compliant System OK
IRQ 184 Microsoft ACPI-Compliant System OK
IRQ 185 Microsoft ACPI-Compliant System OK
IRQ 186 Microsoft ACPI-Compliant System OK
IRQ 187 Microsoft ACPI-Compliant System OK
IRQ 188 Microsoft ACPI-Compliant System OK
IRQ 189 Microsoft ACPI-Compliant System OK
IRQ 190 Microsoft ACPI-Compliant System OK
IRQ 39 ATI ES1000OK
IRQ 11 HP ProLiant iLO 2 Legacy Support Function OK
IRQ 41 HP iLO Management Channel Interface Driver OK
IRQ 41 Standard Universal PCI to USB Host Controller OK
IRQ 40 HP ProLiant iLO 2 Management Controller Driver OK
IRQ 37 Broadcom BCM5706C NetXtreme II GigE OK
IRQ 38 Broadcom BCM5706C NetXtreme II GigE OK
IRQ 14 ATA Channel 0 OK
IRQ 15 ATA Channel 1 OK
IRQ 0 System timer OK
IRQ 1 Standard PS/2 Keyboard OK
IRQ 12 PS/2 Compatible MouseOK
IRQ 4 Communications Port (COM1) OK
IRQ 5 Standard OpenHCD USB Host Controller OK
IRQ 5 Standard OpenHCD USB Host Controller OK
IRQ 5 Standard Enhanced PCI to USB Host Controller OK
IRQ 22 ServerWorks (Broadcom) HT-2100 HT-PCI-E Bridge 0 OK
IRQ 4294967294 Smart Array P800 Controller OK
IRQ 4294967293 Smart Array P800 Controller OK
IRQ 4294967292 Smart Array P800 Controller OK
IRQ 4294967291 Smart Array P800 Controller OK
IRQ 25 ServerWorks (Broadcom) HT-2100 HT-PCI-E Bridge 1&3 OK
IRQ 4294967290 Smart Array P400 Controller OK
IRQ 4294967289 Smart Array P400 Controller OK
IRQ 4294967288 Smart Array P400 Controller OK
IRQ 4294967287 Smart Array P400 Controller OK
IRQ 30 ServerWorks (Broadcom) HT-2100 HT-PCI-E Bridge 2&4 OK
IRQ 31 ServerWorks (Broadcom) HT-2100 HT-PCI-E Bridge 1&3 OK
IRQ 32 ServerWorks (Broadcom) HT-2100 HT-PCI-E Bridge 2&4 OK
IRQ 16 ServerWorks (Broadcom) HT-2100 HT-PCI-E Bridge 0 OK
IRQ 4294967286 Smart Array P800 Controller OK
IRQ 4294967285 Smart Array P800 Controller OK
IRQ 4294967284 Smart Array P800 Controller OK
IRQ 4294967283 Smart Array P800 Controller OK
IRQ 34 ServerWorks (Broadcom) HT-2100 HT-PCI-E Bridge 1&3 OK
IRQ 26 ServerWorks (Broadcom) HT-2100 HT-PCI-E Bridge 2&4 OK
IRQ 19 ServerWorks (Broadcom) HT-2100 HT-PCI-E Bridge 1&3 OK
IRQ 20 ServerWorks (Broadcom) HT-2100 HT-PCI-E Bridge 2&4 OK
IRQ 17 ServerWorks (Broadcom) HT-2100 HT-PCI-E Bridge 0 OK
IRQ 4294967282 Smart Array P800 Controller OK
IRQ 4294967281 Smart Array P800 Controller OK
IRQ 4294967280 Smart Array P800 Controller OK

© 2009 Hewlett-Packard Company. All rights reserved.

IRQ 4294967279 Smart Array P800 Controller OK
IRQ 35 ServerWorks (Broadcom) HT-2100 HT-PCI-E Bridge 1&3 OK
IRQ 29 ServerWorks (Broadcom) HT-2100 HT-PCI-E Bridge 2&4 OK
IRQ 21 ServerWorks (Broadcom) HT-2100 HT-PCI-E Bridge 1&3 OK
IRQ 4294967278 Smart Array P800 Controller OK
IRQ 4294967277 Smart Array P800 Controller OK
IRQ 4294967276 Smart Array P800 Controller OK
IRQ 4294967275 Smart Array P800 Controller OK
IRQ 27 ServerWorks (Broadcom) HT-2100 HT-PCI-E Bridge 2&4 OK
IRQ 18 ServerWorks (Broadcom) HT-2100 HT-PCI-E Bridge 0 OK
IRQ 4294967274 Smart Array P800 Controller OK
IRQ 4294967273 Smart Array P800 Controller OK
IRQ 4294967272 Smart Array P800 Controller OK
IRQ 4294967271 Smart Array P800 Controller OK
IRQ 36 ServerWorks (Broadcom) HT-2100 HT-PCI-E Bridge 1&3 OK
IRQ 33 ServerWorks (Broadcom) HT-2100 HT-PCI-E Bridge 2&4 OK
IRQ 23 ServerWorks (Broadcom) HT-2100 HT-PCI-E Bridge 1&3 OK
IRQ 4294967270 Smart Array P800 Controller OK
IRQ 4294967269 Smart Array P800 Controller OK
IRQ 4294967268 Smart Array P800 Controller OK
IRQ 4294967267 Smart Array P800 Controller OK
IRQ 28 ServerWorks (Broadcom) HT-2100 HT-PCI-E Bridge 2&4 OK

[Memory]

Resource Device Status
0xE8000000-0xFD5FFFFF PCI busOK
0xE8000000-0xFD5FFFFF ATI ES1000 OK
0xA0000-0xBFFFF PCI busOK
0xA0000-0xBFFFF ATI ES1000OK
0xF7FF0000-0xF7FFFFFF ATI ES1000 OK
0xF7FE0000-0xF7FE01FF HP ProLiant iLO 2 Legacy Support Function OK
0xF7FD0000-0xF7FD07FF HP iLO Management Channel Interface Driver OK
0xF7FC0000-0xF7FC3FFF HP iLO Management Channel Interface Driver OK
0xF7F00000-0xF7F7FFFF HP iLO Management Channel Interface Driver OK
0xF7EF0000-0xF7EF00FF HP ProLiant iLO 2 Management Controller Driver OK
0xF8000000-0xFBFFFFFF ServerWorks (Broadcom) HT-1000 HT-PCI-X Bridge 0 OK
0xF8000000-0xFBFFFFFF ServerWorks (Broadcom) HT-1000 HT-PCI-X Bridge OK
0xF8000000-0xFBFFFFFF Broadcom BCM5706C NetXtreme II GigE OK
0xFA000000-0xFBFFFFFF Broadcom BCM5706C NetXtreme II GigE OK
0xD0000000-0xDFFFFFFF Motherboard resources OK
0xFED00000-0xFED003FF High precision event timer OK
0xF7EE0000-0xF7EE0FFF Standard OpenHCD USB Host Controller OK
0xF7ED0000-0xF7ED0FFF Standard OpenHCD USB Host Controller OK
0xF7EC0000-0xF7EC0FFF Standard Enhanced PCI to USB Host Controller OK
0xFD400000-0xFD5FFFFF ServerWorks (Broadcom) HT-2100 HT-PCI-E Bridge 0 OK
0xFD500000-0xFD5FFFFF Smart Array P800 Controller OK
0xFD4F0000-0xFD4F0FFF Smart Array P800 Controller OK
0xFD200000-0xFD3FFFFF ServerWorks (Broadcom) HT-2100 HT-PCI-E Bridge 1&3 OK
0xFD300000-0xFD3FFFFF Smart Array P400 Controller OK
0xFD2F0000-0xFD2F0FFF Smart Array P400 Controller OK
0xFD600000-0xFD7FFFFF PCI busOK
0xFD600000-0xFD7FFFFF ServerWorks (Broadcom) HT-2100 HT-PCI-E Bridge 0 OK
0xFD700000-0xFD7FFFFF Smart Array P800 Controller OK
0xFD6F0000-0xFD6F0FFF Smart Array P800 Controller OK
0xFD800000-0xFDBFFFFF PCI busOK
0xFD800000-0xFDBFFFFF ServerWorks (Broadcom) HT-2100 HT-PCI-E Bridge 0 OK
0xFD900000-0xFD9FFFFF Smart Array P800 Controller OK
0xFD8F0000-0xFD8F0FFF Smart Array P800 Controller OK
0xFDA00000-0xFDBFFFFF ServerWorks (Broadcom) HT-2100 HT-PCI-E Bridge 1&3 OK
0xFDB00000-0xFDBFFFFF Smart Array P800 Controller OK
0xFDAF0000-0xFDAF0FFF Smart Array P800 Controller OK
0xFDC00000-0xFDFFFFFF PCI busOK
0xFDC00000-0xFDFFFFFF ServerWorks (Broadcom) HT-2100 HT-PCI-E Bridge 0 OK
0xFDD00000-0xFDDFFFFF Smart Array P800 Controller OK
0xFDCF0000-0xFDCF0FFF Smart Array P800 Controller OK
0xFDE00000-0xFDFFFFFF ServerWorks (Broadcom) HT-2100 HT-PCI-E Bridge 1&3 OK
0xFDF00000-0xFDFFFFFF Smart Array P800 Controller OK
0xFDEF0000-0xFDEF0FFF Smart Array P800 Controller OK

[Components]

[Multimedia]

[Audio Codecs]

CODEC Manufacturer Description Status File Version Size Creation Date
e:\windows\system32\imaadp32.acm Microsoft Corporation OK E:\Windows\system32\IMAADP32.ACM 6.0.6000.16386 21.00 KB (21,504 bytes) 1/18/2008 10:43 PM
e:\windows\system32\msg711.acm Microsoft Corporation OK E:\Windows\system32\MSG711.ACM 6.0.6000.16386 14.00 KB (14,336 bytes) 1/18/2008 10:43 PM
e:\windows\system32\msadp32.acm Microsoft Corporation OK E:\Windows\system32\MSADP32.ACM 6.0.6000.16386 22.00 KB (22,528 bytes) 1/18/2008 10:43 PM
e:\windows\system32\msgsm32.acm Microsoft Corporation OK E:\Windows\system32\MSGSM32.ACM 6.0.6000.16386 28.00 KB (28,672 bytes) 1/18/2008 10:43 PM

[Video Codecs]

CODEC Manufacturer Description Status File Version Size Creation Date
e:\windows\system32\msyuv.dll Microsoft Corporation OK E:\Windows\system32\MSYUV.DLL 6.0.6000.16386 24.50 KB (25,088 bytes) 1/18/2008 10:34 PM
e:\windows\system32\iyuv_32.dll Microsoft Corporation OK E:\Windows\system32\IYUV_32.DLL 6.0.6000.16386 52.50 KB (53,760 bytes) 1/18/2008 10:34 PM
e:\windows\system32\tsbyuv.dll Microsoft Corporation OK E:\Windows\system32\TSBYUV.DLL 6.0.6000.16386 13.50 KB (13,824 bytes) 1/18/2008 10:34 PM
e:\windows\system32\msvidc32.dll Microsoft Corporation OK E:\Windows\system32\MSVIDC32.DLL 6.0.6001.18000 37.50 KB (38,400 bytes) 1/18/2008 10:43 PM
e:\windows\system32\msrle32.dll Microsoft Corporation OK E:\Windows\system32\MSRLE32.DLL 6.0.6000.16386 15.50 KB (15,872 bytes) 1/18/2008 10:43 PM

[CD-ROM]

Item Value
Drive D:
Description CD-ROM Drive
Media Loaded No
Media Type UNKNOWN
Name TSSTcorp DVD-ROM TS-L332A ATA Device
Manufacturer (Standard CD-ROM drives)
Status OK
Transfer Rate -1.00 kbytes/sec
SCSI Target ID 0
PNP Device ID IDE\CDROMTSSTCORP_DVD-ROM_TS-

© 2009 Hewlett-Packard Company. All rights reserved.

L332A_____HG00____\5&
2274845F&0&0.0.0
Driver
e:\windows\system32\drivers\cdrom.
sys (6.0.6001.18000, 78.00 KB (79,872
bytes), 1/18/2008 10:29 PM)

[Sound Device]

Item Value

[Display]

Item Value
Name ATI ES1000
PNP Device ID
PCI\VEN_1002&DEV_515E&SUBS
YS_31FB103C&REV_02\3&C04C483&
0&18
Adapter Type Not Available, ATI
Technologies Inc. compatible
Adapter Description ATI ES1000
Adapter RAM Not Available
Installed DriversNot Available
Driver Version 6.14.10.6744
INF File oem1.inf (ati2mtag_RN50
section)
Color Planes Not Available
Color Table Entries Not Available
Resolution Not Available
Bits/Pixel Not Available
Memory Address 0xE8000000-
0xFD5FFFFF
I/O Port 0x00001000-0x00005FFF
Memory Address 0xF7FF0000-
0xF7FFFFFF
IRQ Channel IRQ 39
I/O Port 0x000003B0-0x000003BB
I/O Port 0x000003C0-0x000003DF
Memory Address 0xA0000-0xBFFFF
Driver
e:\windows\system32\drivers\ati2mt
ag.sys (6.14.10.6744, 2.11 MB
(2,210,816 bytes), 3/11/2008 11:37 PM)

[Infrared]

Item Value

[Input]

[Keyboard]

Item Value
Description USB Human Interface
Device
Name Enhanced (101- or 102-key)
Layout 00000409
PNP Device ID
USB\VID_03F0&PID_1027&MI_00\
6&174A8EC&0&0000
Number of Function Keys 12
Driver
e:\windows\system32\drivers\hidusb
.sys (6.0.6001.18000, 15.50 KB (15,872
bytes), 1/18/2008 10:33 PM)

Description Standard PS/2 Keyboard
Name Enhanced (101- or 102-key)
Layout 00000409
PNP Device ID
ACPI\PNP0303\4&63ACF31&0
Number of Function Keys 12
I/O Port 0x00000060-0x00000060
I/O Port 0x00000064-0x00000064
IRQ Channel IRQ 1
Driver
e:\windows\system32\drivers\i8042p
rt.sys (6.0.6001.18000, 62.50 KB
(64,000 bytes), 1/18/2008 10:28 PM)

[Pointing Device]

Item Value
Hardware Type USB Human Interface
Device
Number of Buttons 0
Status OK
PNP Device ID
USB\VID_03F0&PID_1027&MI_01\
6&174A8EC&0&0001
Power Management Supported No
Double Click Threshold Not Available
Handedness Not Available
Driver
e:\windows\system32\drivers\hidusb
.sys (6.0.6001.18000, 15.50 KB (15,872
bytes), 1/18/2008 10:33 PM)

Hardware Type PS/2 Compatible Mouse
Number of Buttons 0
Status OK
PNP Device ID
ACPI\PNP0F13\4&63ACF31&0
Power Management Supported No
Double Click Threshold Not Available
Handedness Not Available
IRQ Channel IRQ 12
Driver
e:\windows\system32\drivers\i8042p
rt.sys (6.0.6001.18000, 62.50 KB
(64,000 bytes), 1/18/2008 10:28 PM)

[Modem]

Item Value

[Network]

[Adapter]

Item Value
Name [00000000] WAN Miniport
(SSTP)
Adapter Type Not Available
Product Type WAN Miniport (SSTP)
Installed Yes
PNP Device ID
ROOT\MS_SSTPMINIPORT\0000
Last Reset 7/13/2009 2:55 PM
Index 0
Service Name RasSstp
IP Address Not Available
IP Subnet Not Available
Default IP Gateway Not Available
DHCP Enabled No
DHCP Server Not Available
DHCP Lease Expires Not Available
DHCP Lease Obtained Not Available
MAC Address Not Available
Driver
e:\windows\system32\drivers\rassst
p.sys (6.0.6001.18000, 76.50 KB
(78,336 bytes), 1/18/2008 10:37 PM)

Name [00000001] WAN Miniport
(L2TP)
Adapter Type Not Available
Product Type WAN Miniport (L2TP)
Installed Yes
PNP Device ID
ROOT\MS_L2TPMINIPORT\0000
Last Reset 7/13/2009 2:55 PM
Index 1
Service Name Rasl2tp
IP Address Not Available
IP Subnet Not Available
Default IP Gateway Not Available
DHCP Enabled No
DHCP Server Not Available
DHCP Lease Expires Not Available
DHCP Lease Obtained Not Available
MAC Address Not Available
Driver
e:\windows\system32\drivers\rasl2tp
.sys (6.0.6001.18000, 122.00 KB
(124,928 bytes), 1/18/2008 10:37 PM)

Name [00000002] WAN Miniport
(PPTP)
Adapter Type Wide Area Network
(WAN)
Product Type WAN Miniport (PPTP)
Installed Yes
PNP Device ID
ROOT\MS_PPTPMINIPORT\0000
Last Reset 7/13/2009 2:55 PM
Index 2
Service Name PptpMiniport
IP Address Not Available
IP Subnet Not Available
Default IP Gateway Not Available
DHCP Enabled No
DHCP Server Not Available
DHCP Lease Expires Not Available
DHCP Lease Obtained Not Available
MAC Address 50:50:54:50:30:30
Driver
e:\windows\system32\drivers\rasppt
p.sys (6.0.6001.18000, 96.50 KB
(98,816 bytes), 1/18/2008 10:37 PM)

Name [00000003] WAN Miniport
(PPPOE)
Adapter Type Wide Area Network
(WAN)
Product Type WAN Miniport (PPPOE)
Installed Yes
PNP Device ID
ROOT\MS_PPPOEMINIPORT\0000
Last Reset 7/13/2009 2:55 PM
Index 3
Service Name RasPppoe
IP Address Not Available
IP Subnet Not Available
Default IP Gateway Not Available
DHCP Enabled No
DHCP Server Not Available
DHCP Lease Expires Not Available
DHCP Lease Obtained Not Available
MAC Address 33:50:6F:45:30:30
Driver
e:\windows\system32\drivers\rasppp
oe.sys (6.0.6001.18000, 49.00 KB
(50,176 bytes), 1/18/2008 10:37 PM)

Name [00000004] WAN Miniport (IPv6)
Adapter Type Not Available
Product Type WAN Miniport (IPv6)
Installed Yes

© 2009 Hewlett-Packard Company. All rights reserved.

PNP Device ID
ROOT\MS_NDISWANIPV6\0000
Last Reset 7/13/2009 2:55 PM
Index 4
Service Name NdisWan
IP Address Not Available
IP Subnet Not Available
Default IP Gateway Not Available
DHCP Enabled No
DHCP Server Not Available
DHCP Lease Expires Not Available
DHCP Lease Obtained Not Available
MAC Address Not Available
Driver
e:\windows\system32\drivers\ndiswan.sys (6.0.6001.18000, 165.50 KB (169,472 bytes), 1/18/2008 10:37 PM)

Name [00000005] WAN Miniport (Network Monitor)
Adapter Type Not Available
Product Type WAN Miniport (Network Monitor)
Installed Yes
PNP Device ID
ROOT\MS_NDISWANBH\0000
Last Reset 7/13/2009 2:55 PM
Index 5
Service Name NdisWan
IP Address Not Available
IP Subnet Not Available
Default IP Gateway Not Available
DHCP Enabled No
DHCP Server Not Available
DHCP Lease Expires Not Available
DHCP Lease Obtained Not Available
MAC Address Not Available
Driver
e:\windows\system32\drivers\ndiswan.sys (6.0.6001.18000, 165.50 KB (169,472 bytes), 1/18/2008 10:37 PM)

Name [00000006] Broadcom BCM5706C NetXtreme II GigE (NDIS VBD Client)
Adapter Type Ethernet 802.3
Product Type Broadcom BCM5706C NetXtreme II GigE (NDIS VBD Client)
Installed Yes
PNP Device ID
B06BDRV\L2ND&PCI_164A14E4&SUBSYS_1709103C&REV_02\6&2FFF06&0&20050203
Last Reset 7/13/2009 2:55 PM
Index 6
Service Name l2nd
IP Address 192.168.1.1, fe80::b188:1e1d:ee03:837a
IP Subnet 255.255.255.0, 64
Default IP Gateway 0.0.0.0
DHCP Enabled No
DHCP Server Not Available
DHCP Lease Expires Not Available
DHCP Lease Obtained Not Available
MAC Address 00:1C:C4:F3:97:2C
Driver
e:\windows\system32\drivers\bxnd60a.sys (3.7.19.0, 63.00 KB (64,512 bytes), 1/19/2008 5:53 AM)

Name [00000007] Broadcom BCM5706C NetXtreme II GigE (NDIS VBD Client)
Adapter Type Ethernet 802.3
Product Type Broadcom BCM5706C NetXtreme II GigE (NDIS VBD Client)
Installed Yes
PNP Device ID
B06BDRV\L2ND&PCI_164A14E4&SUBSYS_1709103C&REV_02\6&23BF5E65&0&20050204
Last Reset 7/13/2009 2:55 PM
Index 7
Service Name l2nd
IP Address 15.1.101.44, fe80::29ba:ecb9:ded9:1825
IP Subnet 255.255.0.0, 64
Default IP Gateway Not Available
DHCP Enabled Yes
DHCP Server 15.1.101.1
DHCP Lease Expires 7/21/2009 2:59 PM
DHCP Lease Obtained 7/13/2009 2:59 PM
MAC Address 00:1C:C4:F3:97:2E
Driver
e:\windows\system32\drivers\bxnd60a.sys (3.7.19.0, 63.00 KB (64,512 bytes), 1/19/2008 5:53 AM)

Name [00000008] WAN Miniport (IP)
Adapter Type Not Available
Product Type WAN Miniport (IP)
Installed Yes
PNP Device ID
ROOT\MS_NDISWANIP\0000
Last Reset 7/13/2009 2:55 PM
Index 8
Service Name NdisWan
IP Address Not Available
IP Subnet Not Available
Default IP Gateway Not Available
DHCP Enabled No
DHCP Server Not Available
DHCP Lease Expires Not Available
DHCP Lease Obtained Not Available
MAC Address Not Available
Driver
e:\windows\system32\drivers\ndiswan.sys (6.0.6001.18000, 165.50 KB (169,472 bytes), 1/18/2008 10:37 PM)

Name [00000009] Microsoft ISATAP Adapter
Adapter Type Tunnel
Product Type Microsoft ISATAP Adapter
Installed Yes
PNP Device ID ROOT*ISATAP\0000
Last Reset 7/13/2009 2:55 PM
Index 9
Service Name tunnel
IP Address Not Available
IP Subnet Not Available
Default IP Gateway Not Available
DHCP Enabled No
DHCP Server Not Available
DHCP Lease Expires Not Available
DHCP Lease Obtained Not Available
MAC Address Not Available
Driver
e:\windows\system32\drivers\tunnel.sys (6.0.6001.18000, 27.50 KB (28,160 bytes), 1/18/2008 10:36 PM)

Name [00000010] Microsoft ISATAP Adapter
Adapter Type Tunnel
Product Type Microsoft ISATAP Adapter
Installed Yes
PNP Device ID ROOT*ISATAP\0002
Last Reset 7/13/2009 2:55 PM
Index 10
Service Name tunnel
IP Address Not Available
IP Subnet Not Available
Default IP Gateway Not Available
DHCP Enabled No
DHCP Server Not Available
DHCP Lease Expires Not Available
DHCP Lease Obtained Not Available
MAC Address Not Available
Driver
e:\windows\system32\drivers\tunnel.sys (6.0.6001.18000, 27.50 KB (28,160 bytes), 1/18/2008 10:36 PM)

Name [00000011] RAS Async Adapter
Adapter Type Wide Area Network (WAN)
Product Type RAS Async Adapter
Installed Yes
PNP Device ID SW\{EEAB7790-C514-11D1-B42B-00805FC1270E}\ASYNCMAC
Last Reset 7/13/2009 2:55 PM
Index 11
Service Name AsyncMac
IP Address Not Available
IP Subnet Not Available
Default IP Gateway Not Available
DHCP Enabled No
DHCP Server Not Available
DHCP Lease Expires Not Available
DHCP Lease Obtained Not Available
MAC Address 20:41:53:59:4E:FF
Driver
e:\windows\system32\drivers\asyncmac.sys (6.0.6001.18000, 21.50 KB (22,016 bytes), 1/18/2008 10:37 PM)

Name [00000012] Microsoft 6to4 Adapter
Adapter Type Tunnel
Product Type Microsoft 6to4 Adapter
Installed Yes
PNP Device ID ROOT*6TO4MP\0000
Last Reset 7/13/2009 2:55 PM
Index 12
Service Name tunnel
IP Address Not Available
IP Subnet Not Available
Default IP Gateway Not Available
DHCP Enabled No
DHCP Server Not Available
DHCP Lease Expires Not Available
DHCP Lease Obtained Not Available
MAC Address Not Available
Driver
e:\windows\system32\drivers\tunnel.sys (6.0.6001.18000, 27.50 KB (28,160 bytes), 1/18/2008 10:36 PM)

[Protocol]

Item Value
Name MSAFD Tcpip [TCP/IP]
Connectionless Service No
Guarantees Delivery Yes
Guarantees Sequencing Yes
Maximum Address Size 16 bytes
Maximum Message Size 0 bytes

© 2009 Hewlett-Packard Company. All rights reserved.

Message Oriented No
Minimum Address Size 16 bytes
Pseudo Stream Oriented No
Supports Broadcasting No
Supports Connect Data No
Supports Disconnect Data No
Supports Encryption No
Supports Expedited Data Yes
Supports Graceful Closing Yes
Supports Guaranteed Bandwidth No
Supports Multicasting No

Name MSAFD Tcpip [UDP/IP]
Connectionless Service Yes
Guarantees Delivery No
Guarantees Sequencing No
Maximum Address Size 16 bytes
Maximum Message Size 63.99 KB (65,527 bytes)
Message Oriented Yes
Minimum Address Size 16 bytes
Pseudo Stream Oriented No
Supports Broadcasting Yes
Supports Connect Data No
Supports Disconnect Data No
Supports Encryption No
Supports Expedited Data No
Supports Graceful Closing No
Supports Guaranteed Bandwidth No
Supports Multicasting Yes

Name MSAFD Tcpip [TCP/IPv6]
Connectionless Service No
Guarantees Delivery Yes
Guarantees Sequencing Yes
Maximum Address Size 28 bytes
Maximum Message Size 0 bytes
Message Oriented No
Minimum Address Size 28 bytes
Pseudo Stream Oriented No
Supports Broadcasting No
Supports Connect Data No
Supports Disconnect Data No
Supports Encryption No
Supports Expedited Data Yes
Supports Graceful Closing Yes
Supports Guaranteed Bandwidth No
Supports Multicasting No

Name MSAFD Tcpip [UDP/IPv6]
Connectionless Service Yes
Guarantees Delivery No
Guarantees Sequencing No
Maximum Address Size 28 bytes
Maximum Message Size 63.99 KB (65,527 bytes)
Message Oriented Yes
Minimum Address Size 28 bytes
Pseudo Stream Oriented No
Supports Broadcasting Yes
Supports Connect Data No
Supports Disconnect Data No
Supports Encryption No
Supports Expedited Data No
Supports Graceful Closing No
Supports Guaranteed Bandwidth No
Supports Multicasting Yes

Name RSVP TCPv6 Service Provider
Connectionless Service No
Guarantees Delivery Yes
Guarantees Sequencing Yes
Maximum Address Size 28 bytes
Maximum Message Size 0 bytes
Message Oriented No
Minimum Address Size 28 bytes
Pseudo Stream Oriented No
Supports Broadcasting No
Supports Connect Data No
Supports Disconnect Data No
Supports Encryption Yes
Supports Expedited Data Yes
Supports Graceful Closing Yes
Supports Guaranteed Bandwidth No
Supports Multicasting No

Name RSVP TCP Service Provider
Connectionless Service No
Guarantees Delivery Yes
Guarantees Sequencing Yes
Maximum Address Size 16 bytes
Maximum Message Size 0 bytes
Message Oriented No
Minimum Address Size 16 bytes
Pseudo Stream Oriented No
Supports Broadcasting No
Supports Connect Data No
Supports Disconnect Data No
Supports Encryption Yes
Supports Expedited Data Yes
Supports Graceful Closing Yes
Supports Guaranteed Bandwidth No
Supports Multicasting No

Name RSVP UDPv6 Service Provider
Connectionless Service Yes
Guarantees Delivery No
Guarantees Sequencing No
Maximum Address Size 28 bytes
Maximum Message Size 63.99 KB (65,527 bytes)
Message Oriented Yes
Minimum Address Size 28 bytes
Pseudo Stream Oriented No
Supports Broadcasting Yes
Supports Connect Data No
Supports Disconnect Data No
Supports Encryption Yes
Supports Expedited Data No
Supports Graceful Closing No
Supports Guaranteed Bandwidth No
Supports Multicasting Yes

Name RSVP UDP Service Provider
Connectionless Service Yes
Guarantees Delivery No
Guarantees Sequencing No
Maximum Address Size 16 bytes
Maximum Message Size 63.99 KB (65,527 bytes)
Message Oriented Yes
Minimum Address Size 16 bytes
Pseudo Stream Oriented No
Supports Broadcasting Yes
Supports Connect Data No
Supports Disconnect Data No
Supports Encryption Yes
Supports Expedited Data No
Supports Graceful Closing No
Supports Guaranteed Bandwidth No
Supports Multicasting Yes

[WinSock]

Item Value
File e:\windows\syswow64\wsock32.dll
Size 15.00 KB (15,360 bytes)
Version 6.0.6001.18000

File e:\windows\system32\wsock32.dll
Size 18.00 KB (18,432 bytes)
Version 6.0.6001.18000

[Ports]

[Serial]

Item Value
Name Communications Port (COM1)
Status OK
PNP Device ID ACPI\PNP0501\0
Maximum Input Buffer Size 0
Maximum Output Buffer Size No
Settable Baud Rate Yes
Settable Data Bits Yes
Settable Flow Control Yes
Settable Parity Yes
Settable Parity Check Yes
Settable Stop Bits Yes
Settable RLSD Yes
Supports RLSD Yes
Supports 16 Bit Mode No
Supports Special Characters No
Baud Rate 9600
Bits/Byte 8
Stop Bits 1
Parity None
Busy No
Abort Read/Write on Error No
Binary Mode Enabled Yes
Continue XMit on XOff No
CTS Outflow Control No
Discard NULL Bytes No
DSR Outflow Control 0
DSR Sensitivity 0
DTR Flow Control Type Enable
EOF Character 0
Error Replace Character 0
Error Replacement Enabled No
Event Character 0
Parity Check Enabled No
RTS Flow Control Type Enable
XOff Character 19
XOffXMit Threshold 512
XOn Character 17
XOnXMit Threshold 2048
XOnXOff InFlow Control 0
XOnXOff OutFlow Control 0
IRQ Channel IRQ 4
I/O Port 0x000003F8-0x000003FF
Driver
e:\windows\system32\drivers\serial.sys (6.0.6001.18000, 92.00 KB (94,208 bytes), 1/18/2008 10:28 PM)

[Parallel]

Item Value

[Storage]

[Drives]

Item Value
Drive C:
Description Local Fixed Disk
Compressed No
File System NTFS
Size 33.87 GB (36,372,967,424 bytes)

© 2009 Hewlett-Packard Company. All rights reserved.

Free Space 12.26 GB (13,159,514,112 bytes)
Volume Name WS03
Volume Serial Number 1C26471D

Drive D:
Description CD-ROM Disc

Drive E:
Description Local Fixed Disk
Compressed No
File System NTFS
Size 33.88 GB (36,382,437,376 bytes)
Free Space 8.54 GB (9,166,856,192 bytes)
Volume Name WS08
Volume Serial Number 88E38FFD

Drive G:
Description Local Fixed Disk
Compressed No
File System NTFS
Size 2.90 GB (3,114,266,624 bytes)
Free Space 2.23 GB (2,390,970,368 bytes)
Volume Name G-Drive
Volume Serial Number 8656BFD8

Drive L:
Description Local Fixed Disk
Compressed Not Available
File System Not Available
Size Not Available
Free Space Not Available
Volume Name Not Available
Volume Serial Number Not Available

Drive T:
Description Local Fixed Disk
Compressed No
File System NTFS
Size 24.41 GB (26,214,395,904 bytes)
Free Space 10.33 GB (11,087,134,720 bytes)
Volume Name TempLog
Volume Serial Number 764D6C88

[Disks]

Item Value
Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 3
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 4
SCSI Target ID 4
Sectors/Track 63
Size 16.59 GB (17,815,956,480 bytes)
Total Cylinders 2,166
Total Sectors 34,796,790
Total Tracks 552,330
Tracks/Cylinder 255
Partition Disk #21, Partition #0
Partition Size 4.88 GB (5,242,880,000 bytes)
Partition Starting Offset 135,266,304 bytes
Partition Disk #21, Partition #1
Partition Size 2.44 GB (2,621,440,000 bytes)
Partition Starting Offset 5,378,146,304 bytes
Partition Disk #21, Partition #2
Partition Size 9.15 GB (9,823,059,968 bytes)
Partition Starting Offset 7,999,586,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 1
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 4
SCSI Target ID 5
Sectors/Track 63
Size 192.55 GB (206,750,638,080 bytes)
Total Cylinders 25,136
Total Sectors 403,809,840
Total Tracks 6,409,680
Tracks/Cylinder 255
Partition Disk #22, Partition #0
Partition Size 192.43 GB (206,616,657,920 bytes)
Partition Starting Offset 135,266,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 3
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 4
SCSI Target ID 6
Sectors/Track 63
Size 16.59 GB (17,815,956,480 bytes)
Total Cylinders 2,166
Total Sectors 34,796,790
Total Tracks 552,330
Tracks/Cylinder 255
Partition Disk #23, Partition #0
Partition Size 4.88 GB (5,242,880,000 bytes)
Partition Starting Offset 135,266,304 bytes
Partition Disk #23, Partition #1
Partition Size 2.44 GB (2,621,440,000 bytes)
Partition Starting Offset 5,378,146,304 bytes
Partition Disk #23, Partition #2
Partition Size 9.15 GB (9,823,059,968 bytes)
Partition Starting Offset 7,999,586,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 1
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 4
SCSI Target ID 7
Sectors/Track 63
Size 192.55 GB (206,750,638,080 bytes)
Total Cylinders 25,136
Total Sectors 403,809,840
Total Tracks 6,409,680
Tracks/Cylinder 255
Partition Disk #24, Partition #0
Partition Size 192.43 GB (206,616,657,920 bytes)
Partition Starting Offset 135,266,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 3
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 4
SCSI Target ID 8
Sectors/Track 63
Size 16.59 GB (17,815,956,480 bytes)
Total Cylinders 2,166
Total Sectors 34,796,790
Total Tracks 552,330
Tracks/Cylinder 255
Partition Disk #25, Partition #0
Partition Size 4.88 GB (5,242,880,000 bytes)
Partition Starting Offset 135,266,304 bytes
Partition Disk #25, Partition #1
Partition Size 2.44 GB (2,621,440,000 bytes)
Partition Starting Offset 5,378,146,304 bytes
Partition Disk #25, Partition #2
Partition Size 9.15 GB (9,823,059,968 bytes)
Partition Starting Offset 7,999,586,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 1
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 4
SCSI Target ID 9
Sectors/Track 63
Size 192.55 GB (206,750,638,080 bytes)
Total Cylinders 25,136
Total Sectors 403,809,840

© 2009 Hewlett-Packard Company. All rights reserved.

Total Tracks 6,409,680
Tracks/Cylinder 255
Partition Disk #26, Partition #0
Partition Size 192.43 GB (206,616,657,920 bytes)
Partition Starting Offset 135,266,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 3
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 4
SCSI Target ID 10
Sectors/Track 63
Size 16.59 GB (17,815,956,480 bytes)
Total Cylinders 2,166
Total Sectors 34,796,790
Total Tracks 552,330
Tracks/Cylinder 255
Partition Disk #27, Partition #0
Partition Size 4.88 GB (5,242,880,000 bytes)
Partition Starting Offset 135,266,304 bytes
Partition Disk #27, Partition #1
Partition Size 2.44 GB (2,621,440,000 bytes)
Partition Starting Offset 5,378,146,304 bytes
Partition Disk #27, Partition #2
Partition Size 9.15 GB (9,823,059,968 bytes)
Partition Starting Offset 7,999,586,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 1
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 4
SCSI Target ID 11
Sectors/Track 63
Size 192.55 GB (206,750,638,080 bytes)
Total Cylinders 25,136
Total Sectors 403,809,840
Total Tracks 6,409,680
Tracks/Cylinder 255
Partition Disk #28, Partition #0
Partition Size 192.43 GB (206,616,657,920 bytes)
Partition Starting Offset 135,266,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 3
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 4
SCSI Target ID 12
Sectors/Track 63
Size 16.59 GB (17,815,956,480 bytes)
Total Cylinders 2,166
Total Sectors 34,796,790
Total Tracks 552,330
Tracks/Cylinder 255
Partition Disk #29, Partition #0
Partition Size 4.88 GB (5,242,880,000 bytes)
Partition Starting Offset 135,266,304 bytes
Partition Disk #29, Partition #1
Partition Size 2.44 GB (2,621,440,000 bytes)
Partition Starting Offset 5,378,146,304 bytes
Partition Disk #29, Partition #2
Partition Size 9.15 GB (9,823,059,968 bytes)
Partition Starting Offset 7,999,586,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 1
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 4
SCSI Target ID 13
Sectors/Track 63
Size 192.55 GB (206,750,638,080 bytes)
Total Cylinders 25,136
Total Sectors 403,809,840
Total Tracks 6,409,680
Tracks/Cylinder 255
Partition Disk #30, Partition #0
Partition Size 192.43 GB (206,616,657,920 bytes)
Partition Starting Offset 135,266,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 3
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 4
SCSI Target ID 14
Sectors/Track 63
Size 16.59 GB (17,815,956,480 bytes)
Total Cylinders 2,166
Total Sectors 34,796,790
Total Tracks 552,330
Tracks/Cylinder 255
Partition Disk #31, Partition #0
Partition Size 4.88 GB (5,242,880,000 bytes)
Partition Starting Offset 135,266,304 bytes
Partition Disk #31, Partition #1
Partition Size 2.44 GB (2,621,440,000 bytes)
Partition Starting Offset 5,378,146,304 bytes
Partition Disk #31, Partition #2
Partition Size 9.15 GB (9,823,059,968 bytes)
Partition Starting Offset 7,999,586,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 1
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 4
SCSI Target ID 15
Sectors/Track 63
Size 192.55 GB (206,750,638,080 bytes)
Total Cylinders 25,136
Total Sectors 403,809,840
Total Tracks 6,409,680
Tracks/Cylinder 255
Partition Disk #32, Partition #0
Partition Size 192.43 GB (206,616,657,920 bytes)
Partition Starting Offset 135,266,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 3
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 4
SCSI Target ID 16
Sectors/Track 63
Size 16.59 GB (17,815,956,480 bytes)
Total Cylinders 2,166
Total Sectors 34,796,790
Total Tracks 552,330
Tracks/Cylinder 255
Partition Disk #33, Partition #0
Partition Size 4.88 GB (5,242,880,000 bytes)
Partition Starting Offset 135,266,304 bytes
Partition Disk #33, Partition #1
Partition Size 2.44 GB (2,621,440,000 bytes)
Partition Starting Offset 5,378,146,304 bytes
Partition Disk #33, Partition #2
Partition Size 9.15 GB (9,823,059,968 bytes)
Partition Starting Offset 7,999,586,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)

© 2009 Hewlett-Packard Company. All rights reserved.

Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 1
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 4
SCSI Target ID 17
Sectors/Track 63
Size 192.55 GB (206,750,638,080 bytes)
Total Cylinders 25,136
Total Sectors 403,809,840
Total Tracks 6,409,680
Tracks/Cylinder 255
Partition Disk #34, Partition #0
Partition Size 192.43 GB (206,616,657,920 bytes)
Partition Starting Offset 135,266,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 3
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 4
SCSI Target ID 18
Sectors/Track 63
Size 16.59 GB (17,815,956,480 bytes)
Total Cylinders 2,166
Total Sectors 34,796,790
Total Tracks 552,330
Tracks/Cylinder 255
Partition Disk #35, Partition #0
Partition Size 4.88 GB (5,242,880,000 bytes)
Partition Starting Offset 135,266,304 bytes
Partition Disk #35, Partition #1
Partition Size 2.44 GB (2,621,440,000 bytes)
Partition Starting Offset 5,378,146,304 bytes
Partition Disk #35, Partition #2
Partition Size 9.15 GB (9,823,059,968 bytes)
Partition Starting Offset 7,999,586,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 1
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 4
SCSI Target ID 19
Sectors/Track 63
Size 192.55 GB (206,750,638,080 bytes)
Total Cylinders 25,136
Total Sectors 403,809,840
Total Tracks 6,409,680
Tracks/Cylinder 255
Partition Disk #36, Partition #0
Partition Size 192.43 GB (206,616,657,920 bytes)
Partition Starting Offset 135,266,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 3
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 4
SCSI Target ID 20
Sectors/Track 63
Size 16.59 GB (17,815,956,480 bytes)
Total Cylinders 2,166
Total Sectors 34,796,790
Total Tracks 552,330
Tracks/Cylinder 255
Partition Disk #37, Partition #0
Partition Size 4.88 GB (5,242,880,000 bytes)
Partition Starting Offset 135,266,304 bytes
Partition Disk #37, Partition #1
Partition Size 2.44 GB (2,621,440,000 bytes)
Partition Starting Offset 5,378,146,304 bytes
Partition Disk #37, Partition #2
Partition Size 9.15 GB (9,823,059,968 bytes)
Partition Starting Offset 7,999,586,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 1
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 4
SCSI Target ID 21
Sectors/Track 63
Size 192.55 GB (206,750,638,080 bytes)
Total Cylinders 25,136
Total Sectors 403,809,840
Total Tracks 6,409,680
Tracks/Cylinder 255
Partition Disk #38, Partition #0
Partition Size 192.43 GB (206,616,657,920 bytes)
Partition Starting Offset 135,266,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 3
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 6
SCSI Target ID 4
Sectors/Track 63
Size 16.59 GB (17,815,956,480 bytes)
Total Cylinders 2,166
Total Sectors 34,796,790
Total Tracks 552,330
Tracks/Cylinder 255
Partition Disk #59, Partition #0
Partition Size 4.88 GB (5,242,880,000 bytes)
Partition Starting Offset 135,266,304 bytes
Partition Disk #59, Partition #1
Partition Size 2.44 GB (2,621,440,000 bytes)
Partition Starting Offset 5,378,146,304 bytes
Partition Disk #59, Partition #2
Partition Size 9.15 GB (9,823,059,968 bytes)
Partition Starting Offset 7,999,586,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 1
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 6
SCSI Target ID 5
Sectors/Track 63
Size 192.55 GB (206,750,638,080 bytes)
Total Cylinders 25,136
Total Sectors 403,809,840
Total Tracks 6,409,680
Tracks/Cylinder 255
Partition Disk #60, Partition #0
Partition Size 192.43 GB (206,616,657,920 bytes)
Partition Starting Offset 135,266,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 3
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 6
SCSI Target ID 6
Sectors/Track 63
Size 16.59 GB (17,815,956,480 bytes)
Total Cylinders 2,166
Total Sectors 34,796,790
Total Tracks 552,330
Tracks/Cylinder 255
Partition Disk #61, Partition #0
Partition Size 4.88 GB (5,242,880,000 bytes)

© 2009 Hewlett-Packard Company. All rights reserved.

Partition Starting Offset 135,266,304 bytes
Partition Disk #61, Partition #1
Partition Size 2.44 GB (2,621,440,000 bytes)
Partition Starting Offset 5,378,146,304 bytes
Partition Disk #61, Partition #2
Partition Size 9.15 GB (9,823,059,968 bytes)
Partition Starting Offset 7,999,586,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 1
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 6
SCSI Target ID 7
Sectors/Track 63
Size 192.55 GB (206,750,638,080 bytes)
Total Cylinders 25,136
Total Sectors 403,809,840
Total Tracks 6,409,680
Tracks/Cylinder 255
Partition Disk #62, Partition #0
Partition Size 192.43 GB (206,616,657,920 bytes)
Partition Starting Offset 135,266,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 3
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 6
SCSI Target ID 8
Sectors/Track 63
Size 16.59 GB (17,815,956,480 bytes)
Total Cylinders 2,166
Total Sectors 34,796,790
Total Tracks 552,330
Tracks/Cylinder 255
Partition Disk #63, Partition #0
Partition Size 4.88 GB (5,242,880,000 bytes)
Partition Starting Offset 135,266,304 bytes
Partition Disk #63, Partition #1
Partition Size 2.44 GB (2,621,440,000 bytes)
Partition Starting Offset 5,378,146,304 bytes
Partition Disk #63, Partition #2
Partition Size 9.15 GB (9,823,059,968 bytes)
Partition Starting Offset 7,999,586,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 1
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 6
SCSI Target ID 9
Sectors/Track 63
Size 192.55 GB (206,750,638,080 bytes)
Total Cylinders 25,136
Total Sectors 403,809,840
Total Tracks 6,409,680
Tracks/Cylinder 255
Partition Disk #64, Partition #0
Partition Size 192.43 GB (206,616,657,920 bytes)
Partition Starting Offset 135,266,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 3
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 6
SCSI Target ID 10
Sectors/Track 63
Size 16.59 GB (17,815,956,480 bytes)
Total Cylinders 2,166
Total Sectors 34,796,790
Total Tracks 552,330
Tracks/Cylinder 255
Partition Disk #65, Partition #0
Partition Size 4.88 GB (5,242,880,000 bytes)
Partition Starting Offset 135,266,304 bytes
Partition Disk #65, Partition #1
Partition Size 2.44 GB (2,621,440,000 bytes)
Partition Starting Offset 5,378,146,304 bytes
Partition Disk #65, Partition #2
Partition Size 9.15 GB (9,823,059,968 bytes)
Partition Starting Offset 7,999,586,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 1
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 6
SCSI Target ID 11
Sectors/Track 63
Size 192.55 GB (206,750,638,080 bytes)
Total Cylinders 25,136
Total Sectors 403,809,840
Total Tracks 6,409,680
Tracks/Cylinder 255
Partition Disk #66, Partition #0
Partition Size 192.43 GB (206,616,657,920 bytes)
Partition Starting Offset 135,266,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 3
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 6
SCSI Target ID 12
Sectors/Track 63
Size 16.59 GB (17,815,956,480 bytes)
Total Cylinders 2,166
Total Sectors 34,796,790
Total Tracks 552,330
Tracks/Cylinder 255
Partition Disk #67, Partition #0
Partition Size 4.88 GB (5,242,880,000 bytes)
Partition Starting Offset 135,266,304 bytes
Partition Disk #67, Partition #1
Partition Size 2.44 GB (2,621,440,000 bytes)
Partition Starting Offset 5,378,146,304 bytes
Partition Disk #67, Partition #2
Partition Size 9.15 GB (9,823,059,968 bytes)
Partition Starting Offset 7,999,586,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 1
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 6
SCSI Target ID 13
Sectors/Track 63
Size 192.55 GB (206,750,638,080 bytes)
Total Cylinders 25,136
Total Sectors 403,809,840
Total Tracks 6,409,680
Tracks/Cylinder 255
Partition Disk #68, Partition #0
Partition Size 192.43 GB (206,616,657,920 bytes)
Partition Starting Offset 135,266,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk

© 2009 Hewlett-Packard Company. All rights reserved.

Partitions 3
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 6
SCSI Target ID 14
Sectors/Track 63
Size 16.59 GB (17,815,956,480 bytes)
Total Cylinders 2,166
Total Sectors 34,796,790
Total Tracks 552,330
Tracks/Cylinder 255
Partition Disk #69, Partition #0
Partition Size 4.88 GB (5,242,880,000 bytes)
Partition Starting Offset 135,266,304 bytes
Partition Disk #69, Partition #1
Partition Size 2.44 GB (2,621,440,000 bytes)
Partition Starting Offset 5,378,146,304 bytes
Partition Disk #69, Partition #2
Partition Size 9.15 GB (9,823,059,968 bytes)
Partition Starting Offset 7,999,586,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 1
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 6
SCSI Target ID 15
Sectors/Track 63
Size 192.55 GB (206,750,638,080 bytes)
Total Cylinders 25,136
Total Sectors 403,809,840
Total Tracks 6,409,680
Tracks/Cylinder 255
Partition Disk #70, Partition #0
Partition Size 192.43 GB (206,616,657,920 bytes)
Partition Starting Offset 135,266,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 3
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 2
SCSI Target ID 4
Sectors/Track 63
Size 16.59 GB (17,815,956,480 bytes)
Total Cylinders 2,166
Total Sectors 34,796,790
Total Tracks 552,330
Tracks/Cylinder 255
Partition Disk #0, Partition #0
Partition Size 4.88 GB (5,242,880,000 bytes)
Partition Starting Offset 135,266,304 bytes
Partition Disk #0, Partition #1
Partition Size 2.44 GB (2,621,440,000 bytes)
Partition Starting Offset 5,378,146,304 bytes
Partition Disk #0, Partition #2
Partition Size 9.15 GB (9,823,059,968 bytes)
Partition Starting Offset 7,999,586,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 1
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 2
SCSI Target ID 5
Sectors/Track 63
Size 192.55 GB (206,750,638,080 bytes)
Total Cylinders 25,136
Total Sectors 403,809,840
Total Tracks 6,409,680
Tracks/Cylinder 255
Partition Disk #1, Partition #0
Partition Size 192.43 GB (206,616,657,920 bytes)
Partition Starting Offset 135,266,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 3
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 2
SCSI Target ID 6
Sectors/Track 63
Size 16.59 GB (17,815,956,480 bytes)
Total Cylinders 2,166
Total Sectors 34,796,790
Total Tracks 552,330
Tracks/Cylinder 255
Partition Disk #2, Partition #0
Partition Size 4.88 GB (5,242,880,000 bytes)
Partition Starting Offset 135,266,304 bytes
Partition Disk #2, Partition #1
Partition Size 2.44 GB (2,621,440,000 bytes)
Partition Starting Offset 5,378,146,304 bytes
Partition Disk #2, Partition #2
Partition Size 9.15 GB (9,823,059,968 bytes)
Partition Starting Offset 7,999,586,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 1
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 2
SCSI Target ID 7
Sectors/Track 63
Size 192.55 GB (206,750,638,080 bytes)
Total Cylinders 25,136
Total Sectors 403,809,840
Total Tracks 6,409,680
Tracks/Cylinder 255
Partition Disk #3, Partition #0
Partition Size 192.43 GB (206,616,657,920 bytes)
Partition Starting Offset 135,266,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 3
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 2
SCSI Target ID 8
Sectors/Track 63
Size 16.59 GB (17,815,956,480 bytes)
Total Cylinders 2,166
Total Sectors 34,796,790
Total Tracks 552,330
Tracks/Cylinder 255
Partition Disk #4, Partition #0
Partition Size 4.88 GB (5,242,880,000 bytes)
Partition Starting Offset 135,266,304 bytes
Partition Disk #4, Partition #1
Partition Size 2.44 GB (2,621,440,000 bytes)
Partition Starting Offset 5,378,146,304 bytes
Partition Disk #4, Partition #2
Partition Size 9.15 GB (9,823,059,968 bytes)
Partition Starting Offset 7,999,586,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 1
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 2
SCSI Target ID 9
Sectors/Track 63
Size 192.55 GB (206,750,638,080 bytes)
Total Cylinders 25,136
Total Sectors 403,809,840

© 2009 Hewlett-Packard Company. All rights reserved.

Total Tracks 6,409,680
Tracks/Cylinder 255
Partition Disk #5, Partition #0
Partition Size 192.43 GB (206,616,657,920 bytes)
Partition Starting Offset 135,266,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 3
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 2
SCSI Target ID 10
Sectors/Track 63
Size 16.59 GB (17,815,956,480 bytes)
Total Cylinders 2,166
Total Sectors 34,796,790
Total Tracks 552,330
Tracks/Cylinder 255
Partition Disk #6, Partition #0
Partition Size 4.88 GB (5,242,880,000 bytes)
Partition Starting Offset 135,266,304 bytes
Partition Disk #6, Partition #1
Partition Size 2.44 GB (2,621,440,000 bytes)
Partition Starting Offset 5,378,146,304 bytes
Partition Disk #6, Partition #2
Partition Size 9.15 GB (9,823,059,968 bytes)
Partition Starting Offset 7,999,586,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 1
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 2
SCSI Target ID 11
Sectors/Track 63
Size 192.55 GB (206,750,638,080 bytes)
Total Cylinders 25,136
Total Sectors 403,809,840
Total Tracks 6,409,680
Tracks/Cylinder 255
Partition Disk #7, Partition #0
Partition Size 192.43 GB (206,616,657,920 bytes)
Partition Starting Offset 135,266,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 3
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 2
SCSI Target ID 12
Sectors/Track 63
Size 16.59 GB (17,815,956,480 bytes)
Total Cylinders 2,166
Total Sectors 34,796,790
Total Tracks 552,330
Tracks/Cylinder 255
Partition Disk #8, Partition #0
Partition Size 4.88 GB (5,242,880,000 bytes)
Partition Starting Offset 135,266,304 bytes
Partition Disk #8, Partition #1
Partition Size 2.44 GB (2,621,440,000 bytes)
Partition Starting Offset 5,378,146,304 bytes
Partition Disk #8, Partition #2
Partition Size 9.15 GB (9,823,059,968 bytes)
Partition Starting Offset 7,999,586,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 1
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 2
SCSI Target ID 13
Sectors/Track 63
Size 192.55 GB (206,750,638,080 bytes)
Total Cylinders 25,136
Total Sectors 403,809,840
Total Tracks 6,409,680
Tracks/Cylinder 255
Partition Disk #9, Partition #0
Partition Size 192.43 GB (206,616,657,920 bytes)
Partition Starting Offset 135,266,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 3
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 2
SCSI Target ID 14
Sectors/Track 63
Size 16.59 GB (17,815,956,480 bytes)
Total Cylinders 2,166
Total Sectors 34,796,790
Total Tracks 552,330
Tracks/Cylinder 255
Partition Disk #10, Partition #0
Partition Size 4.88 GB (5,242,880,000 bytes)
Partition Starting Offset 135,266,304 bytes
Partition Disk #10, Partition #1
Partition Size 2.44 GB (2,621,440,000 bytes)
Partition Starting Offset 5,378,146,304 bytes
Partition Disk #10, Partition #2
Partition Size 9.15 GB (9,823,059,968 bytes)
Partition Starting Offset 7,999,586,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 1
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 2
SCSI Target ID 15
Sectors/Track 63
Size 192.55 GB (206,750,638,080 bytes)
Total Cylinders 25,136
Total Sectors 403,809,840
Total Tracks 6,409,680
Tracks/Cylinder 255
Partition Disk #11, Partition #0
Partition Size 192.43 GB (206,616,657,920 bytes)
Partition Starting Offset 135,266,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 3
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 2
SCSI Target ID 16
Sectors/Track 63
Size 16.59 GB (17,815,956,480 bytes)
Total Cylinders 2,166
Total Sectors 34,796,790
Total Tracks 552,330
Tracks/Cylinder 255
Partition Disk #12, Partition #0
Partition Size 4.88 GB (5,242,880,000 bytes)
Partition Starting Offset 135,266,304 bytes
Partition Disk #12, Partition #1
Partition Size 2.44 GB (2,621,440,000 bytes)
Partition Starting Offset 5,378,146,304 bytes
Partition Disk #12, Partition #2
Partition Size 9.15 GB (9,823,059,968 bytes)
Partition Starting Offset 7,999,586,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)

© 2009 Hewlett-Packard Company. All rights reserved.

Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 1
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 2
SCSI Target ID 17
Sectors/Track 63
Size 192.55 GB (206,750,638,080 bytes)
Total Cylinders 25,136
Total Sectors 403,809,840
Total Tracks 6,409,680
Tracks/Cylinder 255
Partition Disk #13, Partition #0
Partition Size 192.43 GB (206,616,657,920 bytes)
Partition Starting Offset 135,266,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 3
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 2
SCSI Target ID 18
Sectors/Track 63
Size 16.59 GB (17,815,956,480 bytes)
Total Cylinders 2,166
Total Sectors 34,796,790
Total Tracks 552,330
Tracks/Cylinder 255
Partition Disk #14, Partition #0
Partition Size 4.88 GB (5,242,880,000 bytes)
Partition Starting Offset 135,266,304 bytes
Partition Disk #14, Partition #1
Partition Size 2.44 GB (2,621,440,000 bytes)
Partition Starting Offset 5,378,146,304 bytes
Partition Disk #14, Partition #2
Partition Size 9.15 GB (9,823,059,968 bytes)
Partition Starting Offset 7,999,586,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 1
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 2
SCSI Target ID 19
Sectors/Track 63
Size 192.55 GB (206,750,638,080 bytes)
Total Cylinders 25,136
Total Sectors 403,809,840
Total Tracks 6,409,680
Tracks/Cylinder 255
Partition Disk #15, Partition #0
Partition Size 192.43 GB (206,616,657,920 bytes)
Partition Starting Offset 135,266,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 3
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 2
SCSI Target ID 20
Sectors/Track 63
Size 16.59 GB (17,815,956,480 bytes)
Total Cylinders 2,166
Total Sectors 34,796,790
Total Tracks 552,330
Tracks/Cylinder 255
Partition Disk #16, Partition #0
Partition Size 4.88 GB (5,242,880,000 bytes)
Partition Starting Offset 135,266,304 bytes
Partition Disk #16, Partition #1
Partition Size 2.44 GB (2,621,440,000 bytes)
Partition Starting Offset 5,378,146,304 bytes
Partition Disk #16, Partition #2
Partition Size 9.15 GB (9,823,059,968 bytes)
Partition Starting Offset 7,999,586,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 1
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 2
SCSI Target ID 21
Sectors/Track 63
Size 192.55 GB (206,750,638,080 bytes)
Total Cylinders 25,136
Total Sectors 403,809,840
Total Tracks 6,409,680
Tracks/Cylinder 255
Partition Disk #17, Partition #0
Partition Size 192.43 GB (206,616,657,920 bytes)
Partition Starting Offset 135,266,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 3
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 5
SCSI Target ID 4
Sectors/Track 63
Size 16.59 GB (17,815,956,480 bytes)
Total Cylinders 2,166
Total Sectors 34,796,790
Total Tracks 552,330
Tracks/Cylinder 255
Partition Disk #39, Partition #0
Partition Size 4.88 GB (5,242,880,000 bytes)
Partition Starting Offset 135,266,304 bytes
Partition Disk #39, Partition #1
Partition Size 2.44 GB (2,621,440,000 bytes)
Partition Starting Offset 5,378,146,304 bytes
Partition Disk #39, Partition #2
Partition Size 9.15 GB (9,823,059,968 bytes)
Partition Starting Offset 7,999,586,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 1
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 5
SCSI Target ID 5
Sectors/Track 63
Size 192.55 GB (206,750,638,080 bytes)
Total Cylinders 25,136
Total Sectors 403,809,840
Total Tracks 6,409,680
Tracks/Cylinder 255
Partition Disk #40, Partition #0
Partition Size 192.43 GB (206,616,657,920 bytes)
Partition Starting Offset 135,266,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 3
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 5
SCSI Target ID 6
Sectors/Track 63
Size 16.59 GB (17,815,956,480 bytes)
Total Cylinders 2,166
Total Sectors 34,796,790
Total Tracks 552,330
Tracks/Cylinder 255
Partition Disk #41, Partition #0
Partition Size 4.88 GB (5,242,880,000 bytes)

© 2009 Hewlett-Packard Company. All rights reserved.

Partition Starting Offset 135,266,304 bytes
Partition Disk #41, Partition #1
Partition Size 2.44 GB (2,621,440,000 bytes)
Partition Starting Offset 5,378,146,304 bytes
Partition Disk #41, Partition #2
Partition Size 9.15 GB (9,823,059,968 bytes)
Partition Starting Offset 7,999,586,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 1
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 5
SCSI Target ID 7
Sectors/Track 63
Size 192.55 GB (206,750,638,080 bytes)
Total Cylinders 25,136
Total Sectors 403,809,840
Total Tracks 6,409,680
Tracks/Cylinder 255
Partition Disk #42, Partition #0
Partition Size 192.43 GB (206,616,657,920 bytes)
Partition Starting Offset 135,266,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 3
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 5
SCSI Target ID 8
Sectors/Track 63
Size 16.59 GB (17,815,956,480 bytes)
Total Cylinders 2,166
Total Sectors 34,796,790
Total Tracks 552,330
Tracks/Cylinder 255
Partition Disk #43, Partition #0
Partition Size 4.88 GB (5,242,880,000 bytes)
Partition Starting Offset 135,266,304 bytes
Partition Disk #43, Partition #1
Partition Size 2.44 GB (2,621,440,000 bytes)
Partition Starting Offset 5,378,146,304 bytes
Partition Disk #43, Partition #2
Partition Size 9.15 GB (9,823,059,968 bytes)
Partition Starting Offset 7,999,586,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 1
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 5
SCSI Target ID 9
Sectors/Track 63
Size 192.55 GB (206,750,638,080 bytes)
Total Cylinders 25,136
Total Sectors 403,809,840
Total Tracks 6,409,680
Tracks/Cylinder 255
Partition Disk #44, Partition #0
Partition Size 192.43 GB (206,616,657,920 bytes)
Partition Starting Offset 135,266,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 3
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 5
SCSI Target ID 10
Sectors/Track 63
Size 16.59 GB (17,815,956,480 bytes)
Total Cylinders 2,166
Total Sectors 34,796,790
Total Tracks 552,330
Tracks/Cylinder 255
Partition Disk #45, Partition #0
Partition Size 4.88 GB (5,242,880,000 bytes)
Partition Starting Offset 135,266,304 bytes
Partition Disk #45, Partition #1
Partition Size 2.44 GB (2,621,440,000 bytes)
Partition Starting Offset 5,378,146,304 bytes
Partition Disk #45, Partition #2
Partition Size 9.15 GB (9,823,059,968 bytes)
Partition Starting Offset 7,999,586,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 1
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 5
SCSI Target ID 11
Sectors/Track 63
Size 192.55 GB (206,750,638,080 bytes)
Total Cylinders 25,136
Total Sectors 403,809,840
Total Tracks 6,409,680
Tracks/Cylinder 255
Partition Disk #46, Partition #0
Partition Size 192.43 GB (206,616,657,920 bytes)
Partition Starting Offset 135,266,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 3
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 5
SCSI Target ID 12
Sectors/Track 63
Size 16.59 GB (17,815,956,480 bytes)
Total Cylinders 2,166
Total Sectors 34,796,790
Total Tracks 552,330
Tracks/Cylinder 255
Partition Disk #47, Partition #0
Partition Size 4.88 GB (5,242,880,000 bytes)
Partition Starting Offset 135,266,304 bytes
Partition Disk #47, Partition #1
Partition Size 2.44 GB (2,621,440,000 bytes)
Partition Starting Offset 5,378,146,304 bytes
Partition Disk #47, Partition #2
Partition Size 9.15 GB (9,823,059,968 bytes)
Partition Starting Offset 7,999,586,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 1
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 5
SCSI Target ID 13
Sectors/Track 63
Size 192.55 GB (206,750,638,080 bytes)
Total Cylinders 25,136
Total Sectors 403,809,840
Total Tracks 6,409,680
Tracks/Cylinder 255
Partition Disk #48, Partition #0
Partition Size 192.43 GB (206,616,657,920 bytes)
Partition Starting Offset 135,266,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk

© 2009 Hewlett-Packard Company. All rights reserved.

Partitions 3
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 5
SCSI Target ID 14
Sectors/Track 63
Size 16.59 GB (17,815,956,480 bytes)
Total Cylinders 2,166
Total Sectors 34,796,790
Total Tracks 552,330
Tracks/Cylinder 255
Partition Disk #49, Partition #0
Partition Size 4.88 GB (5,242,880,000 bytes)
Partition Starting Offset 135,266,304 bytes
Partition Disk #49, Partition #1
Partition Size 2.44 GB (2,621,440,000 bytes)
Partition Starting Offset 5,378,146,304 bytes
Partition Disk #49, Partition #2
Partition Size 9.15 GB (9,823,059,968 bytes)
Partition Starting Offset 7,999,586,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 1
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 5
SCSI Target ID 15
Sectors/Track 63
Size 192.55 GB (206,750,638,080 bytes)
Total Cylinders 25,136
Total Sectors 403,809,840
Total Tracks 6,409,680
Tracks/Cylinder 255
Partition Disk #50, Partition #0
Partition Size 192.43 GB (206,616,657,920 bytes)
Partition Starting Offset 135,266,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 3
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 5
SCSI Target ID 16
Sectors/Track 63
Size 16.59 GB (17,815,956,480 bytes)
Total Cylinders 2,166
Total Sectors 34,796,790
Total Tracks 552,330
Tracks/Cylinder 255
Partition Disk #51, Partition #0
Partition Size 4.88 GB (5,242,880,000 bytes)
Partition Starting Offset 135,266,304 bytes
Partition Disk #51, Partition #1
Partition Size 2.44 GB (2,621,440,000 bytes)
Partition Starting Offset 5,378,146,304 bytes
Partition Disk #51, Partition #2
Partition Size 9.15 GB (9,823,059,968 bytes)
Partition Starting Offset 7,999,586,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 1
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 5
SCSI Target ID 17
Sectors/Track 63
Size 192.55 GB (206,750,638,080 bytes)
Total Cylinders 25,136
Total Sectors 403,809,840
Total Tracks 6,409,680
Tracks/Cylinder 255
Partition Disk #52, Partition #0
Partition Size 192.43 GB (206,616,657,920 bytes)
Partition Starting Offset 135,266,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 3
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 5
SCSI Target ID 18
Sectors/Track 63
Size 16.59 GB (17,815,956,480 bytes)
Total Cylinders 2,166
Total Sectors 34,796,790
Total Tracks 552,330
Tracks/Cylinder 255
Partition Disk #53, Partition #0
Partition Size 4.88 GB (5,242,880,000 bytes)
Partition Starting Offset 135,266,304 bytes
Partition Disk #53, Partition #1
Partition Size 2.44 GB (2,621,440,000 bytes)
Partition Starting Offset 5,378,146,304 bytes
Partition Disk #53, Partition #2
Partition Size 9.15 GB (9,823,059,968 bytes)
Partition Starting Offset 7,999,586,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 1
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 5
SCSI Target ID 19
Sectors/Track 63
Size 192.55 GB (206,750,638,080 bytes)
Total Cylinders 25,136
Total Sectors 403,809,840
Total Tracks 6,409,680
Tracks/Cylinder 255
Partition Disk #54, Partition #0
Partition Size 192.43 GB (206,616,657,920 bytes)
Partition Starting Offset 135,266,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 3
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 5
SCSI Target ID 20
Sectors/Track 63
Size 16.59 GB (17,815,956,480 bytes)
Total Cylinders 2,166
Total Sectors 34,796,790
Total Tracks 552,330
Tracks/Cylinder 255
Partition Disk #55, Partition #0
Partition Size 4.88 GB (5,242,880,000 bytes)
Partition Starting Offset 135,266,304 bytes
Partition Disk #55, Partition #1
Partition Size 2.44 GB (2,621,440,000 bytes)
Partition Starting Offset 5,378,146,304 bytes
Partition Disk #55, Partition #2
Partition Size 9.15 GB (9,823,059,968 bytes)
Partition Starting Offset 7,999,586,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 1
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 5
SCSI Target ID 21
Sectors/Track 63
Size 192.55 GB (206,750,638,080 bytes)
Total Cylinders 25,136
Total Sectors 403,809,840

© 2009 Hewlett-Packard Company. All rights reserved.

Total Tracks 6,409,680
Tracks/Cylinder 255
Partition Disk #56, Partition #0
Partition Size 192.43 GB (206,616,657,920 bytes)
Partition Starting Offset 135,266,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 3
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 5
SCSI Target ID 22
Sectors/Track 63
Size 16.59 GB (17,815,956,480 bytes)
Total Cylinders 2,166
Total Sectors 34,796,790
Total Tracks 552,330
Tracks/Cylinder 255
Partition Disk #57, Partition #0
Partition Size 4.88 GB (5,242,880,000 bytes)
Partition Starting Offset 135,266,304 bytes
Partition Disk #57, Partition #1
Partition Size 2.44 GB (2,621,440,000 bytes)
Partition Starting Offset 5,378,146,304 bytes
Partition Disk #57, Partition #2
Partition Size 9.15 GB (9,823,059,968 bytes)
Partition Starting Offset 7,999,586,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 1
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 5
SCSI Target ID 23
Sectors/Track 63
Size 192.55 GB (206,750,638,080 bytes)
Total Cylinders 25,136
Total Sectors 403,809,840
Total Tracks 6,409,680
Tracks/Cylinder 255
Partition Disk #58, Partition #0
Partition Size 192.43 GB (206,616,657,920 bytes)
Partition Starting Offset 135,266,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 2
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 3
SCSI Target ID 4
Sectors/Track 32
Size 33.89 GB (36,385,505,280 bytes)
Total Cylinders 8,709
Total Sectors 71,065,440
Total Tracks 2,220,795
Tracks/Cylinder 255
Partition Disk #18, Partition #0
Partition Size 7.81 MB (8,193,024 bytes)
Partition Starting Offset 32,256 bytes
Partition Disk #18, Partition #1
Partition Size 33.87 GB (36,372,971,520 bytes)
Partition Starting Offset 8,355,840 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 1
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 3
SCSI Target ID 5
Sectors/Track 32
Size 33.89 GB (36,385,505,280 bytes)
Total Cylinders 8,709
Total Sectors 71,065,440
Total Tracks 2,220,795
Tracks/Cylinder 255
Partition Disk #19, Partition #0
Partition Size 33.88 GB (36,382,441,472 bytes)
Partition Starting Offset 1,048,576 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 3
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 3
SCSI Target ID 6
Sectors/Track 32
Size 68.33 GB (73,372,631,040 bytes)
Total Cylinders 17,562
Total Sectors 143,305,920
Total Tracks 4,478,310
Tracks/Cylinder 255
Partition Disk #20, Partition #0
Partition Size 41.02 GB (44,040,192,000 bytes)
Partition Starting Offset 1,048,576 bytes
Partition Disk #20, Partition #1
Partition Size 24.41 GB (26,214,400,000 bytes)
Partition Starting Offset 44,041,240,576 bytes
Partition Disk #20, Partition #2
Partition Size 2.90 GB (3,114,270,720 bytes)
Partition Starting Offset 70,255,640,576 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 3
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 8
SCSI Target ID 4
Sectors/Track 63
Size 16.59 GB (17,815,956,480 bytes)
Total Cylinders 2,166
Total Sectors 34,796,790
Total Tracks 552,330
Tracks/Cylinder 255
Partition Disk #87, Partition #0
Partition Size 4.88 GB (5,242,880,000 bytes)
Partition Starting Offset 135,266,304 bytes
Partition Disk #87, Partition #1
Partition Size 2.44 GB (2,621,440,000 bytes)
Partition Starting Offset 5,378,146,304 bytes
Partition Disk #87, Partition #2
Partition Size 9.15 GB (9,823,059,968 bytes)
Partition Starting Offset 7,999,586,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 1
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 8
SCSI Target ID 5
Sectors/Track 63
Size 192.55 GB (206,750,638,080 bytes)
Total Cylinders 25,136
Total Sectors 403,809,840
Total Tracks 6,409,680
Tracks/Cylinder 255
Partition Disk #88, Partition #0
Partition Size 192.43 GB (206,616,657,920 bytes)
Partition Starting Offset 135,266,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 3
SCSI Bus 0
SCSI Logical Unit 0

© 2009 Hewlett-Packard Company. All rights reserved.

SCSI Port 8
SCSI Target ID 6
Sectors/Track 63
Size 16.59 GB (17,815,956,480 bytes)
Total Cylinders 2,166
Total Sectors 34,796,790
Total Tracks 552,330
Tracks/Cylinder 255
Partition Disk #89, Partition #0
Partition Size 4.88 GB (5,242,880,000 bytes)
Partition Starting Offset 135,266,304 bytes
Partition Disk #89, Partition #1
Partition Size 2.44 GB (2,621,440,000 bytes)
Partition Starting Offset 5,378,146,304 bytes
Partition Disk #89, Partition #2
Partition Size 9.15 GB (9,823,059,968 bytes)
Partition Starting Offset 7,999,586,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 1
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 8
SCSI Target ID 7
Sectors/Track 63
Size 192.55 GB (206,750,638,080 bytes)
Total Cylinders 25,136
Total Sectors 403,809,840
Total Tracks 6,409,680
Tracks/Cylinder 255
Partition Disk #90, Partition #0
Partition Size 192.43 GB (206,616,657,920 bytes)
Partition Starting Offset 135,266,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 3
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 8
SCSI Target ID 8
Sectors/Track 63
Size 16.59 GB (17,815,956,480 bytes)
Total Cylinders 2,166
Total Sectors 34,796,790
Total Tracks 552,330
Tracks/Cylinder 255
Partition Disk #91, Partition #0
Partition Size 4.88 GB (5,242,880,000 bytes)
Partition Starting Offset 135,266,304 bytes
Partition Disk #91, Partition #1
Partition Size 2.44 GB (2,621,440,000 bytes)
Partition Starting Offset 5,378,146,304 bytes
Partition Disk #91, Partition #2
Partition Size 9.15 GB (9,823,059,968 bytes)
Partition Starting Offset 7,999,586,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 1
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 8
SCSI Target ID 9
Sectors/Track 63
Size 192.55 GB (206,750,638,080 bytes)
Total Cylinders 25,136
Total Sectors 403,809,840
Total Tracks 6,409,680
Tracks/Cylinder 255
Partition Disk #92, Partition #0
Partition Size 192.43 GB (206,616,657,920 bytes)
Partition Starting Offset 135,266,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 3
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 8
SCSI Target ID 10
Sectors/Track 63
Size 16.59 GB (17,815,956,480 bytes)
Total Cylinders 2,166
Total Sectors 34,796,790
Total Tracks 552,330
Tracks/Cylinder 255
Partition Disk #93, Partition #0
Partition Size 4.88 GB (5,242,880,000 bytes)
Partition Starting Offset 135,266,304 bytes
Partition Disk #93, Partition #1
Partition Size 2.44 GB (2,621,440,000 bytes)
Partition Starting Offset 5,378,146,304 bytes
Partition Disk #93, Partition #2
Partition Size 9.15 GB (9,823,059,968 bytes)
Partition Starting Offset 7,999,586,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 1
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 8
SCSI Target ID 11
Sectors/Track 63
Size 192.55 GB (206,750,638,080 bytes)
Total Cylinders 25,136
Total Sectors 403,809,840
Total Tracks 6,409,680
Tracks/Cylinder 255
Partition Disk #94, Partition #0
Partition Size 192.43 GB (206,616,657,920 bytes)
Partition Starting Offset 135,266,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 3
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 8
SCSI Target ID 12
Sectors/Track 63
Size 16.59 GB (17,815,956,480 bytes)
Total Cylinders 2,166
Total Sectors 34,796,790
Total Tracks 552,330
Tracks/Cylinder 255
Partition Disk #95, Partition #0
Partition Size 4.88 GB (5,242,880,000 bytes)
Partition Starting Offset 135,266,304 bytes
Partition Disk #95, Partition #1
Partition Size 2.44 GB (2,621,440,000 bytes)
Partition Starting Offset 5,378,146,304 bytes
Partition Disk #95, Partition #2
Partition Size 9.15 GB (9,823,059,968 bytes)
Partition Starting Offset 7,999,586,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 1
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 8
SCSI Target ID 13
Sectors/Track 63
Size 192.55 GB (206,750,638,080 bytes)
Total Cylinders 25,136
Total Sectors 403,809,840
Total Tracks 6,409,680
Tracks/Cylinder 255
Partition Disk #96, Partition #0

© 2009 Hewlett-Packard Company. All rights reserved.

Partition Size 192.43 GB (206,616,657,920 bytes)
Partition Starting Offset 135,266,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 3
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 8
SCSI Target ID 14
Sectors/Track 63
Size 16.59 GB (17,815,956,480 bytes)
Total Cylinders 2,166
Total Sectors 34,796,790
Total Tracks 552,330
Tracks/Cylinder 255
Partition Disk #97, Partition #0
Partition Size 4.88 GB (5,242,880,000 bytes)
Partition Starting Offset 135,266,304 bytes
Partition Disk #97, Partition #1
Partition Size 2.44 GB (2,621,440,000 bytes)
Partition Starting Offset 5,378,146,304 bytes
Partition Disk #97, Partition #2
Partition Size 9.15 GB (9,823,059,968 bytes)
Partition Starting Offset 7,999,586,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 1
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 8
SCSI Target ID 15
Sectors/Track 63
Size 192.55 GB (206,750,638,080 bytes)
Total Cylinders 25,136
Total Sectors 403,809,840
Total Tracks 6,409,680
Tracks/Cylinder 255
Partition Disk #98, Partition #0
Partition Size 192.43 GB (206,616,657,920 bytes)
Partition Starting Offset 135,266,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 3
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 7
SCSI Target ID 4
Sectors/Track 63
Size 16.59 GB (17,815,956,480 bytes)
Total Cylinders 2,166
Total Sectors 34,796,790
Total Tracks 552,330
Tracks/Cylinder 255
Partition Disk #71, Partition #0
Partition Size 4.88 GB (5,242,880,000 bytes)
Partition Starting Offset 135,266,304 bytes
Partition Disk #71, Partition #1
Partition Size 2.44 GB (2,621,440,000 bytes)
Partition Starting Offset 5,378,146,304 bytes
Partition Disk #71, Partition #2
Partition Size 9.15 GB (9,823,059,968 bytes)
Partition Starting Offset 7,999,586,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 1
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 7
SCSI Target ID 5
Sectors/Track 63
Size 192.55 GB (206,750,638,080 bytes)
Total Cylinders 25,136
Total Sectors 403,809,840
Total Tracks 6,409,680
Tracks/Cylinder 255
Partition Disk #72, Partition #0
Partition Size 192.43 GB (206,616,657,920 bytes)
Partition Starting Offset 135,266,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 3
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 7
SCSI Target ID 6
Sectors/Track 63
Size 16.59 GB (17,815,956,480 bytes)
Total Cylinders 2,166
Total Sectors 34,796,790
Total Tracks 552,330
Tracks/Cylinder 255
Partition Disk #73, Partition #0
Partition Size 4.88 GB (5,242,880,000 bytes)
Partition Starting Offset 135,266,304 bytes
Partition Disk #73, Partition #1
Partition Size 2.44 GB (2,621,440,000 bytes)
Partition Starting Offset 5,378,146,304 bytes
Partition Disk #73, Partition #2
Partition Size 9.15 GB (9,823,059,968 bytes)
Partition Starting Offset 7,999,586,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 1
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 7
SCSI Target ID 7
Sectors/Track 63
Size 192.55 GB (206,750,638,080 bytes)
Total Cylinders 25,136
Total Sectors 403,809,840
Total Tracks 6,409,680
Tracks/Cylinder 255
Partition Disk #74, Partition #0
Partition Size 192.43 GB (206,616,657,920 bytes)
Partition Starting Offset 135,266,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 3
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 7
SCSI Target ID 8
Sectors/Track 63
Size 16.59 GB (17,815,956,480 bytes)
Total Cylinders 2,166
Total Sectors 34,796,790
Total Tracks 552,330
Tracks/Cylinder 255
Partition Disk #75, Partition #0
Partition Size 4.88 GB (5,242,880,000 bytes)
Partition Starting Offset 135,266,304 bytes
Partition Disk #75, Partition #1
Partition Size 2.44 GB (2,621,440,000 bytes)
Partition Starting Offset 5,378,146,304 bytes
Partition Disk #75, Partition #2
Partition Size 9.15 GB (9,823,059,968 bytes)
Partition Starting Offset 7,999,586,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512

© 2009 Hewlett-Packard Company. All rights reserved.

Media Loaded Yes
Media Type Fixed hard disk
Partitions 1
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 7
SCSI Target ID 9
Sectors/Track 63
Size 192.55 GB (206,750,638,080 bytes)
Total Cylinders 25,136
Total Sectors 403,809,840
Total Tracks 6,409,680
Tracks/Cylinder 255
Partition Disk #76, Partition #0
Partition Size 192.43 GB (206,616,657,920 bytes)
Partition Starting Offset 135,266,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 3
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 7
SCSI Target ID 10
Sectors/Track 63
Size 16.59 GB (17,815,956,480 bytes)
Total Cylinders 2,166
Total Sectors 34,796,790
Total Tracks 552,330
Tracks/Cylinder 255
Partition Disk #77, Partition #0
Partition Size 4.88 GB (5,242,880,000 bytes)
Partition Starting Offset 135,266,304 bytes
Partition Disk #77, Partition #1
Partition Size 2.44 GB (2,621,440,000 bytes)
Partition Starting Offset 5,378,146,304 bytes
Partition Disk #77, Partition #2
Partition Size 9.15 GB (9,823,059,968 bytes)
Partition Starting Offset 7,999,586,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 1
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 7
SCSI Target ID 11
Sectors/Track 63
Size 192.55 GB (206,750,638,080 bytes)
Total Cylinders 25,136
Total Sectors 403,809,840
Total Tracks 6,409,680
Tracks/Cylinder 255
Partition Disk #78, Partition #0
Partition Size 192.43 GB (206,616,657,920 bytes)
Partition Starting Offset 135,266,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 3
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 7
SCSI Target ID 12
Sectors/Track 63
Size 16.59 GB (17,815,956,480 bytes)
Total Cylinders 2,166
Total Sectors 34,796,790
Total Tracks 552,330
Tracks/Cylinder 255
Partition Disk #79, Partition #0
Partition Size 4.88 GB (5,242,880,000 bytes)
Partition Starting Offset 135,266,304 bytes
Partition Disk #79, Partition #1
Partition Size 2.44 GB (2,621,440,000 bytes)
Partition Starting Offset 5,378,146,304 bytes
Partition Disk #79, Partition #2
Partition Size 9.15 GB (9,823,059,968 bytes)
Partition Starting Offset 7,999,586,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 1
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 7
SCSI Target ID 13
Sectors/Track 63
Size 192.55 GB (206,750,638,080 bytes)
Total Cylinders 25,136
Total Sectors 403,809,840
Total Tracks 6,409,680
Tracks/Cylinder 255
Partition Disk #80, Partition #0
Partition Size 192.43 GB (206,616,657,920 bytes)
Partition Starting Offset 135,266,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 3
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 7
SCSI Target ID 14
Sectors/Track 63
Size 16.59 GB (17,815,956,480 bytes)
Total Cylinders 2,166
Total Sectors 34,796,790
Total Tracks 552,330
Tracks/Cylinder 255
Partition Disk #81, Partition #0
Partition Size 4.88 GB (5,242,880,000 bytes)
Partition Starting Offset 135,266,304 bytes
Partition Disk #81, Partition #1
Partition Size 2.44 GB (2,621,440,000 bytes)
Partition Starting Offset 5,378,146,304 bytes
Partition Disk #81, Partition #2
Partition Size 9.15 GB (9,823,059,968 bytes)
Partition Starting Offset 7,999,586,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 1
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 7
SCSI Target ID 15
Sectors/Track 63
Size 192.55 GB (206,750,638,080 bytes)
Total Cylinders 25,136
Total Sectors 403,809,840
Total Tracks 6,409,680
Tracks/Cylinder 255
Partition Disk #82, Partition #0
Partition Size 192.43 GB (206,616,657,920 bytes)
Partition Starting Offset 135,266,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 3
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 7
SCSI Target ID 16
Sectors/Track 63
Size 16.59 GB (17,815,956,480 bytes)
Total Cylinders 2,166
Total Sectors 34,796,790
Total Tracks 552,330
Tracks/Cylinder 255
Partition Disk #83, Partition #0
Partition Size 4.88 GB (5,242,880,000 bytes)
Partition Starting Offset 135,266,304 bytes
Partition Disk #83, Partition #1

© 2009 Hewlett-Packard Company. All rights reserved.

Partition Size 2.44 GB (2,621,440,000 bytes)
Partition Starting Offset 5,378,146,304 bytes
Partition Disk #83, Partition #2
Partition Size 9.15 GB (9,823,059,968 bytes)
Partition Starting Offset 7,999,586,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 1
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 7
SCSI Target ID 17
Sectors/Track 63
Size 192.55 GB (206,750,638,080 bytes)
Total Cylinders 25,136
Total Sectors 403,809,840
Total Tracks 6,409,680
Tracks/Cylinder 255
Partition Disk #84, Partition #0
Partition Size 192.43 GB (206,616,657,920 bytes)
Partition Starting Offset 135,266,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 3
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 7
SCSI Target ID 18
Sectors/Track 63
Size 16.59 GB (17,815,956,480 bytes)
Total Cylinders 2,166
Total Sectors 34,796,790
Total Tracks 552,330
Tracks/Cylinder 255
Partition Disk #85, Partition #0
Partition Size 4.88 GB (5,242,880,000 bytes)
Partition Starting Offset 135,266,304 bytes
Partition Disk #85, Partition #1
Partition Size 2.44 GB (2,621,440,000 bytes)
Partition Starting Offset 5,378,146,304 bytes
Partition Disk #85, Partition #2
Partition Size 9.15 GB (9,823,059,968 bytes)
Partition Starting Offset 7,999,586,304 bytes

Description Disk drive
Manufacturer (Standard disk drives)
Model HP LOGICAL VOLUME SCSI Disk Device
Bytes/Sector 512
Media Loaded Yes
Media Type Fixed hard disk
Partitions 1
SCSI Bus 0
SCSI Logical Unit 0
SCSI Port 7
SCSI Target ID 19
Sectors/Track 63
Size 192.55 GB (206,750,638,080 bytes)
Total Cylinders 25,136
Total Sectors 403,809,840
Total Tracks 6,409,680
Tracks/Cylinder 255
Partition Disk #86, Partition #0
Partition Size 192.43 GB (206,616,657,920 bytes)
Partition Starting Offset 135,266,304 bytes

[SCSI]

Item Value
Name Smart Array P800 Controller
Manufacturer Hewlett-Packard Company
Status OK
PNP Device ID
PCI\VEN_103C&DEV_3230&SUBSYS_3223103C&REV_03\4&359FAD9&0&0098
Memory Address 0xFD500000-0xFD5FFFFF
Memory Address 0xFD4F0000-0xFD4F0FFF
IRQ Channel IRQ 4294967294
IRQ Channel IRQ 4294967293
IRQ Channel IRQ 4294967292
IRQ Channel IRQ 4294967291
Driver
e:\windows\system32\drivers\hpcisss2.sys (6.13.0.64, 84.04 KB (86,056 bytes), 3/11/2008 11:11 PM)

Name Smart Array P400 Controller
Manufacturer Hewlett-Packard Company
Status OK
PNP Device ID
PCI\VEN_103C&DEV_3230&SUBSYS_3234103C&REV_03\4&322FE8E2&0&00A0
Memory Address 0xFD300000-0xFD3FFFFF
Memory Address 0xFD2F0000-0xFD2F0FFF
IRQ Channel IRQ 4294967290
IRQ Channel IRQ 4294967289
IRQ Channel IRQ 4294967288
IRQ Channel IRQ 4294967287
Driver
e:\windows\system32\drivers\hpcisss2.sys (6.13.0.64, 84.04 KB (86,056 bytes), 3/11/2008 11:11 PM)

Name Smart Array P800 Controller
Manufacturer Hewlett-Packard Company
Status OK
PNP Device ID
PCI\VEN_103C&DEV_3230&SUBSYS_3223103C&REV_03\4&31E04343&0&0098
Memory Address 0xFD700000-0xFD7FFFFF
Memory Address 0xFD6F0000-0xFD6F0FFF
IRQ Channel IRQ 4294967286
IRQ Channel IRQ 4294967285
IRQ Channel IRQ 4294967284
IRQ Channel IRQ 4294967283
Driver
e:\windows\system32\drivers\hpcisss2.sys (6.13.0.64, 84.04 KB (86,056 bytes), 3/11/2008 11:11 PM)

Name Smart Array P800 Controller
Manufacturer Hewlett-Packard Company
Status OK
PNP Device ID
PCI\VEN_103C&DEV_3230&SUBSYS_3223103C&REV_03\4&234B80F1&0&0098
Memory Address 0xFD900000-0xFD9FFFFF
Memory Address 0xFD8F0000-0xFD8F0FFF
IRQ Channel IRQ 4294967282
IRQ Channel IRQ 4294967281
IRQ Channel IRQ 4294967280
IRQ Channel IRQ 4294967279
Driver
e:\windows\system32\drivers\hpcisss2.sys (6.13.0.64, 84.04 KB (86,056 bytes), 3/11/2008 11:11 PM)

Name Smart Array P800 Controller
Manufacturer Hewlett-Packard Company
Status OK
PNP Device ID
PCI\VEN_103C&DEV_3230&SUBSYS_3223103C&REV_03\4&16D9858&0&00B0
Memory Address 0xFDB00000-0xFDBFFFFF
Memory Address 0xFDAF0000-0xFDAF0FFF
IRQ Channel IRQ 4294967278
IRQ Channel IRQ 4294967277
IRQ Channel IRQ 4294967276
IRQ Channel IRQ 4294967275
Driver
e:\windows\system32\drivers\hpcisss2.sys (6.13.0.64, 84.04 KB (86,056 bytes), 3/11/2008 11:11 PM)

Name Smart Array P800 Controller
Manufacturer Hewlett-Packard Company
Status OK
PNP Device ID
PCI\VEN_103C&DEV_3230&SUBSYS_3223103C&REV_03\4&E9161B2&0&0098
Memory Address 0xFDD00000-0xFDDFFFFF
Memory Address 0xFDCF0000-0xFDCF0FFF
IRQ Channel IRQ 4294967274
IRQ Channel IRQ 4294967273
IRQ Channel IRQ 4294967272
IRQ Channel IRQ 4294967271
Driver
e:\windows\system32\drivers\hpcisss2.sys (6.13.0.64, 84.04 KB (86,056 bytes), 3/11/2008 11:11 PM)

Name Smart Array P800 Controller

© 2009 Hewlett-Packard Company. All rights reserved.

Manufacturer Hewlett-Packard Company
Status OK
PNP Device ID
PCI\VEN_103C&DEV_3230&SUBSYS_3223103C&REV_03\4&30576D44&0&00B0
Memory Address 0xFDF00000-0xFDFFFFFF
Memory Address 0xFDEF0000-0xFDEF0FFF
IRQ Channel IRQ 4294967270
IRQ Channel IRQ 4294967269
IRQ Channel IRQ 4294967268
IRQ Channel IRQ 4294967267
Driver
e:\windows\system32\drivers\hpcisss2.sys (6.13.0.64, 84.04 KB (86,056 bytes), 3/11/2008 11:11 PM)

Name Microsoft iSCSI Initiator
Manufacturer Microsoft
Status OK
PNP Device ID ROOT\ISCSIPRT\0000
Driver
e:\windows\system32\drivers\msiscsi.sys (6.0.6001.18000, 210.05 KB (215,096 bytes), 1/18/2008 10:30 PM)

[IDE]

Item Value
Name Standard Dual Channel PCI IDE Controller
Manufacturer (Standard IDE ATA/ATAPI controllers)
Status OK
PNP Device ID
PCI\VEN_1166&DEV_0214&SUBSYS_320B103C&REV_00\3&C04C483&0&31
I/O Port 0x00000500-0x0000050F
Driver
e:\windows\system32\drivers\pciide.sys (6.0.6000.16386, 13.10 KB (13,416 bytes), 1/18/2008 10:28 PM)

Name ATA Channel 0
Manufacturer (Standard IDE ATA/ATAPI controllers)
Status OK
PNP Device ID
PCIIDE\IDECHANNEL\4&148384DD&0&0
I/O Port 0x000001F0-0x000001F7
I/O Port 0x000003F6-0x000003F6
IRQ Channel IRQ 14
Driver
e:\windows\system32\drivers\atapi.sys (6.0.6001.18000, 22.05 KB (22,584 bytes), 1/18/2008 10:28 PM)

Name ATA Channel 1
Manufacturer (Standard IDE ATA/ATAPI controllers)
Status OK
PNP Device ID
PCIIDE\IDECHANNEL\4&148384DD&0&1
I/O Port 0x00000170-0x00000177
I/O Port 0x00000376-0x00000376
IRQ Channel IRQ 15
Driver
e:\windows\system32\drivers\atapi.sys (6.0.6001.18000, 22.05 KB (22,584 bytes), 1/18/2008 10:28 PM)

[Printing]

Can't Collect Information

[Problem Devices]

Device PNP Device ID Error Code
Not Available ACPI\PNP0C0F\1 The drivers for this device are not installed.

[USB]

Device PNP Device ID
Standard Universal PCI to USB Host Controller
PCI\VEN_103C&DEV_3300&SUBSYS_3305103C&REV_00\3&C04C483&0&24
Standard OpenHCD USB Host Controller
PCI\VEN_1166&DEV_0223&SUBSYS_320C103C&REV_01\3&C04C483&0&38
Standard OpenHCD USB Host Controller
PCI\VEN_1166&DEV_0223&SUBSYS_320C103C&REV_01\3&C04C483&0&39
Standard Enhanced PCI to USB Host Controller
PCI\VEN_1166&DEV_0223&SUBSYS_320D103C&REV_01\3&C04C483&0&3A

[Software Environment]

[System Drivers]

Name Description File Type Started Start Mode State Status Error Control Accept Pause Accept Stop
acpi Microsoft ACPI Driver
e:\windows\system32\drivers\acpi.sys Kernel Driver Yes Boot Running OK Critical No Yes
adp94xx adp94xx
e:\windows\system32\drivers\adp94xx.sys Kernel Driver No Disabled Stopped OK Normal No No
adpahci adpahci
e:\windows\system32\drivers\adpahci.sys Kernel Driver No Disabled Stopped OK Normal No No
adpu160m adpu160m
e:\windows\system32\drivers\adpu160m.sys Kernel Driver No Disabled Stopped OK Normal No No
adpu320 adpu320
e:\windows\system32\drivers\adpu320.sys Kernel Driver No Disabled Stopped OK Normal No No
afd Ancilliary Function Driver for Winsock
e:\windows\system32\drivers\afd.sys Kernel Driver Yes System Running OK Normal No Yes
agp440 Intel AGP Bus Filter
e:\windows\system32\drivers\agp440.sys Kernel Driver No Manual Stopped OK Normal No No
aic78xx aic78xx
e:\windows\system32\drivers\djsvs.sys Kernel Driver No Disabled Stopped OK Normal No No
aliide aliide
e:\windows\system32\drivers\aliide.sys Kernel Driver No Disabled Stopped OK Critical No No
amdide amdide
e:\windows\system32\drivers\amdide.sys Kernel Driver No Disabled Stopped OK Critical No No
amdk8 AMD K8 Processor Driver
e:\windows\system32\drivers\amdk8.sysKernel Driver No Disabled Stopped OK Normal No No
amdtools64 AMD Special Tools Driver
e:\windows\system32\drivers\amdtools64.sys Kernel Driver Yes Manual Running OK Normal No Yes
arc arc
e:\windows\system32\drivers\arc.sys Kernel Driver No Disabled Stopped OK Normal No No
arcsas arcsas
e:\windows\system32\drivers\arcsas.sysKernel Driver No Disabled Stopped OK Normal No No
asyncmac RAS Asynchronous Media Driver
e:\windows\system32\drivers\asyncmac.sys Kernel Driver Yes Manual Running OK Normal No Yes
atapi IDE Channel
e:\windows\system32\drivers\atapi.sys Kernel Driver Yes Boot Running OK Critical No Yes
ati2mtag ati2mtag
e:\windows\system32\drivers\ati2mtag.sys Kernel Driver Yes Manual Running OK Ignore No Yes
b06bdrv Broadcom NetXtreme II VBD
e:\windows\system32\drivers\bxvbda.sys Kernel Driver Yes Manual Running OK Normal No Yes
blbdrive blbdrive
e:\windows\system32\drivers\blbdrive.sys Kernel Driver No Disabled Stopped OK Normal No No
bowser Bowser
e:\windows\system32\drivers\bowser.sys File System Driver Yes Manual Running OK Normal No Yes
brfiltlo Brother USB Mass-Storage Lower Filter Driver
e:\windows\system32\drivers\brfiltlo.sys Kernel Driver No Manual Stopped OK Normal No No
brfiltup Brother USB Mass-Storage Upper Filter Driver
e:\windows\system32\drivers\brfiltup.sysKernel Driver No Manual Stopped OK Normal No No
brserid Brother MFC Serial Port Interface Driver (WDM)
e:\windows\system32\drivers\brserid.sysKernel Driver No Disabled Stopped OK Normal No No
brserwdm Brother WDM Serial driver
e:\windows\system32\drivers\brserwdm.sys Kernel Driver No Disabled Stopped OK Normal No No

© 2009 Hewlett-Packard Company. All rights reserved.

brusbmdm Brother MFC USB Fax Only Modem
e:\windows\system32\drivers\brusbmdm.sys Kernel Driver No
Disabled Stopped OK Normal No No
brusbser Brother MFC USB Serial WDM Driver
e:\windows\system32\drivers\brusbser.sys Kernel Driver No Manual
Stopped OK Normal No No
cdfs CD/DVD File System Reader
e:\windows\system32\drivers\cdfs.sys File System Driver Yes Disabled
Running OK Normal No Yes
cdrom CD-ROM Driver
e:\windows\system32\drivers\cdrom.sys Kernel Driver Yes System
Running OK Normal No Yes
circlass Consumer IR Devices
e:\windows\system32\drivers\circlass.sys Kernel Driver No Disabled
Stopped OK Normal No No
clfs Common Log (CLFS)
e:\windows\system32\clfs.sys
Kernel Driver Yes Boot
Running OK Critical No Yes
cmdide cmdide
e:\windows\system32\drivers\cmdide.sys Kernel Driver No Disabled
Stopped OK Critical No No
compbatt Microsoft Composite Battery Driver
e:\windows\system32\drivers\compbatt.sys Kernel Driver No Disabled
Stopped OK Critical No No
cpqcidrv HP iLO Management Channel Interface Driver
e:\windows\system32\drivers\cpqcidrv.sys Kernel Driver Yes Manual
Running OK Normal No Yes
cpuspy3 CpuSpy3 Driver
\??\e:\windows\system32\drivers\cpuspy3.sys Kernel Driver No Manual
Stopped OK Normal No No
crcdisk Crcdisk Filter Driver
e:\windows\system32\drivers\crcdisk.sys Kernel Driver Yes Boot
Running OK Normal No Yes
csc Offline Files Driver
e:\windows\system32\drivers\csc.sys Kernel Driver No Disabled
Stopped OK Normal No No
dfsc DFS Namespace Client Driver
e:\windows\system32\drivers\dfsc.sys File System Driver Yes System
Running OK Normal No Yes
disk Disk Driver
e:\windows\system32\drivers\disk.sys Kernel Driver Yes Boot
Running OK Normal No Yes
dxgkrnl LDDM Graphics Subsystem
e:\windows\system32\drivers\dxgkrnl.sys Kernel Driver No Manual
Stopped OK Ignore No No
e1g60 Intel(R) PRO/1000 NDIS 6 Adapter Driver
e:\windows\system32\drivers\e1g6032e.sys Kernel Driver No Manual
Stopped OK Normal No No
elxstor elxstor
e:\windows\system32\drivers\elxstor.sys Kernel Driver No Disabled
Stopped OK Normal No No
errdev Microsoft Hardware Error Device Driver
e:\windows\system32\drivers\errdev.sys Kernel Driver No Disabled
Stopped OK Normal No No
exfat exFAT File System Driver
e:\windows\system32\drivers\exfat.sys File System Driver No Manual
Stopped OK Normal No No
fastfat FAT12/16/32 File System Driver
e:\windows\system32\drivers\fastfat.sys File System Driver No Manual
Stopped OK Normal No No
fdc Floppy Disk Controller Driver
e:\windows\system32\drivers\fdc.sys
Kernel Driver No Disabled
Stopped OK Normal No No
fileinfo File Information FS MiniFilter
e:\windows\system32\drivers\fileinfo.sys File System Driver No Manual
Stopped OK Normal No No
filetrace FileTrace
e:\windows\system32\drivers\filetrace.sys File System Driver No Manual
Stopped OK Normal No No
fiodrive Fusion ioDrive Device Driver
e:\windows\system32\drivers\fiodrive.sys Kernel Driver Yes Boot
Running OK Normal No Yes
flpydisk Floppy Disk Driver
e:\windows\system32\drivers\flpydisk.sys Kernel Driver No Disabled
Stopped OK Normal No No
fltmgr FltMgr
e:\windows\system32\drivers\fltmgr.sys File System Driver Yes Boot
Running OK Critical No Yes
gagp30kx Microsoft Generic AGPv3.0 Filter for K8 Processor Platforms
e:\windows\system32\drivers\gagp30kx.sys Kernel Driver No Manual
Stopped OK Normal No No
hdaudbus Microsoft UAA Bus Driver for High Definition Audio
e:\windows\system32\drivers\hdaudbus.sys Kernel Driver No Disabled
Stopped OK Normal No No
hidbth Microsoft Bluetooth HID Miniport
e:\windows\system32\drivers\hidbth.sys Kernel Driver No Disabled
Stopped OK Ignore No No
hidir Microsoft Infrared HID Driver
e:\windows\system32\drivers\hidir.sys Kernel Driver No Disabled
Stopped OK Ignore No No
hidusb Microsoft HID Class Driver
e:\windows\system32\drivers\hidusb.sys Kernel Driver Yes Manual
Running OK Ignore No Yes
hpcisss HpCISSs
e:\windows\system32\drivers\hpcisss.sys Kernel Driver Yes Boot
Running OK Normal No Yes
hpcisss2 HpCISSs2
e:\windows\system32\drivers\hpcisss2.sys Kernel Driver Yes Boot
Running OK Normal No Yes
hpqilo2 hpqilo2
e:\windows\system32\drivers\hpqilo2.sys Kernel Driver Yes Manual
Running OK Normal No Yes
http HTTP
e:\windows\system32\drivers\http.sys Kernel Driver Yes Manual
Running OK Normal No Yes
i2omp i2omp
e:\windows\system32\drivers\i2omp.sys Kernel Driver No Disabled
Stopped OK Normal No No
i8042prt i8042 Keyboard and PS/2 Mouse Port Driver
e:\windows\system32\drivers\i8042prt.sys Kernel Driver Yes System
Running OK Normal No Yes
iastorv Intel RAID Controller Vista
e:\windows\system32\drivers\iastorv.sys Kernel Driver No Disabled
Stopped OK Normal No No
iirsp iirsp
e:\windows\system32\drivers\iirsp.sys Kernel Driver No Disabled
Stopped OK Normal No No
intelide intelide
e:\windows\system32\drivers\intelide.sys Kernel Driver No Disabled
Stopped OK Critical No No
intelppm Intel Processor Driver
e:\windows\system32\drivers\intelppm.sys Kernel Driver No Disabled
Stopped OK Normal No No
ioatdma Intel(R) QuickData Technology Device
e:\windows\system32\drivers\qd260x64.sys Kernel Driver No Disabled
Stopped OK Normal No No
ipfilterdriver IP Traffic Filter Driver
e:\windows\system32\drivers\ipfltdrv.sys Kernel Driver No Manual
Stopped OK Normal No No
ipmidrv IPMIDRV
e:\windows\system32\drivers\ipmidrv.sys Kernel Driver Yes Manual
Running OK Normal No Yes
ipnat IP Network Address Translator
e:\windows\system32\drivers\ipnat.sys Kernel Driver No Manual
Stopped OK Normal No No
irenum IR Bus Enumerator
e:\windows\system32\drivers\irenum.sys Kernel Driver No Manual
Stopped OK Ignore No No
isapnp PnP ISA/EISA Bus Driver
e:\windows\system32\drivers\isapnp.sys Kernel Driver No Disabled
Stopped OK Critical No No
iscsiprt iScsiPort Driver
e:\windows\system32\drivers\msiscsi.sys Kernel Driver Yes Manual
Running OK Normal No Yes
iteatapi ITEATAPI_Service_Install
e:\windows\system32\drivers\iteatapi.sys Kernel Driver No Disabled
Stopped OK Normal No No
iteraid ITERAID_Service_Install
e:\windows\system32\drivers\iteraid.sys Kernel Driver No Disabled
Stopped OK Normal No No
kbdclass Keyboard Class Driver
e:\windows\system32\drivers\kbdclass.sys Kernel Driver Yes System
Running OK Normal No Yes
kbdhid Keyboard HID Driver
e:\windows\system32\drivers\kbdhid.sys Kernel Driver Yes System
Running OK Ignore No Yes
ksecdd KSecDD
e:\windows\system32\drivers\ksecdd.sys Kernel Driver Yes Boot
Running OK Critical No Yes

© 2009 Hewlett-Packard Company. All rights reserved.

ksthunk Kernel Streaming Thunks
e:\windows\system32\drivers\ksthun
k.sys Kernel Driver No Manual
Stopped OK Normal No No
l2nd Broadcom NetXtreme II BXND
e:\windows\system32\drivers\bxnd6
0a.sys Kernel Driver Yes Manual
Running OK Normal No Yes
lltdio Link-Layer Topology Discovery
Mapper I/O Driver
e:\windows\system32\drivers\lltdio.s
ys Kernel Driver Yes Auto
Running OK Normal No Yes
lsi_fc LSI_FC
e:\windows\system32\drivers\lsi_fc.s
ys Kernel Driver No Disabled
Stopped OK Normal No No
lsi_sas LSI_SAS
e:\windows\system32\drivers\lsi_sas
.sysKernel Driver No Disabled
Stopped OK Normal No No
lsi_scsi LSI_SCSI
e:\windows\system32\drivers\lsi_scs
i.sys Kernel Driver No Disabled
Stopped OK Normal No No
luafv UAC File Virtualization
e:\windows\system32\drivers\luafv.s
ys File System Driver Yes Auto
Running OK Normal No Yes
megasas megasas
e:\windows\system32\drivers\megas
as.sys Kernel Driver No Disabled
Stopped OK Normal No No
megasr MegaSR
e:\windows\system32\drivers\megas
r.sys Kernel Driver No Disabled
Stopped OK Normal No No
modem Modem
e:\windows\system32\drivers\mode
m.sys Kernel Driver No Manual
Stopped OK Ignore No No
monitor Microsoft Monitor Class
Function Driver Service
e:\windows\system32\drivers\monito
r.sys Kernel Driver Yes Manual
Running OK Normal No Yes
mouclass Mouse Class Driver
e:\windows\system32\drivers\moucl
ass.sys Kernel Driver Yes System
Running OK Normal No Yes
mouhid Mouse HID Driver
e:\windows\system32\drivers\mouhi
d.sys Kernel Driver Yes Manual
Running OK Ignore No Yes
mountmgr Mount Point Manager
e:\windows\system32\drivers\mount
mgr.sys Kernel Driver Yes Boot
Running OK Critical No Yes
mpio Microsoft Multi-Path Bus Driver
e:\windows\system32\drivers\mpio.s
ys Kernel Driver No Disabled
Stopped OK Normal No No
mpsdrv Windows Firewall Authorization
Driver
e:\windows\system32\drivers\mpsdr
v.sys Kernel Driver No Manual
Stopped OK Normal No No
mraid35x Mraid35x
e:\windows\system32\drivers\mraid3
5x.sys Kernel Driver No Disabled
Stopped OK Normal No No
mrxsmb SMB MiniRedirector Wrapper
and Engine
e:\windows\system32\drivers\mrxsm
b.sys File System Driver Yes Manual
Running OK Normal No Yes
mrxsmb10 SMB 1.x MiniRedirector
e:\windows\system32\drivers\mrxsm
b10.sys File System Driver Yes Manual
Running OK Normal No Yes
mrxsmb20 SMB 2.0 MiniRedirector
e:\windows\system32\drivers\mrxsm
b20.sys File System Driver Yes Manual
Running OK Normal No Yes
msahci msahci
e:\windows\system32\drivers\msahc
i.sys Kernel Driver No Disabled
Stopped OK Critical No No
msdsm Microsoft Multi-Path Device
Specific Module
e:\windows\system32\drivers\msds
m.sys Kernel Driver No Disabled
Stopped OK Normal No No
msfs Msfs
e:\windows\system32\drivers\msfs.s
ys File System Driver Yes System
Running OK Normal No Yes
msisadrv ISA/EISA Class Driver
e:\windows\system32\drivers\msisa
drv.sys Kernel Driver Yes Boot
Running OK Critical No Yes
msrpc MsRPC
e:\windows\system32\drivers\msrpc.
sys Kernel Driver No Manual
Stopped OK Normal No No
mssmbios Microsoft System
Management BIOS Driver
e:\windows\system32\drivers\mssm
bios.sys Kernel Driver Yes Manual
Running OK Normal No Yes
multevent MultEvent Driver
\??\e:\windows\system32\drivers\mu
lteventdriver.sys Kernel Driver No
Manual Stopped OK Normal No
No
mup Mup
e:\windows\system32\drivers\mup.s
ys File System Driver Yes Boot
Running OK Normal No Yes
ndis NDIS System Driver
e:\windows\system32\drivers\ndis.sy
s Kernel Driver Yes Boot
Running OK Critical No Yes
ndistapi Remote Access NDIS TAPI
Driver
e:\windows\system32\drivers\ndista
pi.sys Kernel Driver Yes Manual
Running OK Normal No Yes
ndisuio NDIS Usermode I/O Protocol
e:\windows\system32\drivers\ndisui
o.sys Kernel Driver No Manual
Stopped OK Normal No No
ndiswan Remote Access NDIS WAN
Driver
e:\windows\system32\drivers\ndiswa
n.sys Kernel Driver Yes Manual
Running OK Normal No Yes
ndproxy NDIS Proxy
e:\windows\system32\drivers\ndprox
y.sys Kernel Driver Yes Manual
Running OK Normal No Yes
netbios NetBIOS Interface
e:\windows\system32\drivers\netbio
s.sys File System Driver Yes System
Running OK Normal No Yes
netbt NETBT
e:\windows\system32\drivers\netbt.s
ys Kernel Driver Yes System
Running OK Normal No Yes
nfrd960 nfrd960
e:\windows\system32\drivers\nfrd96
0.sys Kernel Driver No Disabled
Stopped OK Normal No No
npfs Npfs
e:\windows\system32\drivers\npfs.s
ys File System Driver Yes System
Running OK Normal No Yes
nsiproxy NSI proxy service
e:\windows\system32\drivers\nsipro
xy.sys Kernel Driver Yes System
Running OK Normal No Yes
ntfs Ntfs
e:\windows\system32\drivers\ntfs.sy
s File System Driver Yes Manual
Running OK Normal No Yes
null Null
e:\windows\system32\drivers\null.sy
s Kernel Driver Yes System
Running OK Normal No Yes
nvraid NVIDIA nForce RAID Driver
e:\windows\system32\drivers\nvraid.
sys Kernel Driver No Disabled
Stopped OK Normal No No
nvstor nvstor
e:\windows\system32\drivers\nvstor.
sys Kernel Driver No Disabled
Stopped OK Critical No No
nv_agp NVIDIA nForce AGP Bus Filter
e:\windows\system32\drivers\nv_ag
p.sys Kernel Driver No Manual
Stopped OK Normal No No
ohci1394 NEC FireWarden OHCI
Compliant IEEE 1394 Host Controller
e:\windows\system32\drivers\ohci13
94.sys Kernel Driver No Disabled
Stopped OK Normal No No
parport Parallel port driver
e:\windows\system32\drivers\parpor
t.sys Kernel Driver No Disabled
Stopped OK Normal No No
partmgr Partition Manager
e:\windows\system32\drivers\partmg
r.sys Kernel Driver Yes Boot
Running OK Critical No Yes
pci PCI Bus Driver
e:\windows\system32\drivers\pci.sys
Kernel Driver Yes Boot
Running OK Critical No Yes
pciide pciide
e:\windows\system32\drivers\pciide.
sys Kernel Driver Yes Boot
Running OK Critical No Yes
pcmcia pcmcia
e:\windows\system32\drivers\pcmci
a.sys Kernel Driver No Disabled
Stopped OK Normal No No
peauth PEAUTH
e:\windows\system32\drivers\peauth
.sysKernel Driver Yes Auto
Running OK Normal No Yes
pptpminiport WAN Miniport (PPTP)
e:\windows\system32\drivers\rasppt
p.sys Kernel Driver Yes Manual
Running OK Normal No Yes
processor Processor Driver
e:\windows\system32\drivers\proces
sr.sys Kernel Driver Yes Manual
Running OK Normal No Yes
procinfoProcInfo
e:\windows\system32\drivers\procinf
o.sys Kernel Driver No Manual
Stopped OK Normal No No
psched QoS Packet Scheduler
e:\windows\system32\drivers\pacer.

© 2009 Hewlett-Packard Company. All rights reserved.

sys Kernel Driver Yes System
Running OK Normal No Yes
ql2300 QLogic Fibre Channel Miniport Driver
e:\windows\system32\drivers\ql2300.sysKernel Driver No Disabled
Stopped OK Normal No No
ql40xx QLogic iSCSI Miniport Driver
e:\windows\system32\drivers\ql40xx.sysKernel Driver No Disabled
Stopped OK Normal No No
rasacd Remote Access Auto Connection Driver
e:\windows\system32\drivers\rasacd.sysKernel Driver Yes System
Running OK Normal No Yes
rasl2tp WAN Miniport (L2TP)
e:\windows\system32\drivers\rasl2tp.sysKernel Driver Yes Manual
Running OK Normal No Yes
raspppoe Remote Access PPPOE Driver
e:\windows\system32\drivers\rasppoe.sys Kernel Driver Yes Manual
Running OK Normal No Yes
rassstp WAN Miniport (SSTP)
e:\windows\system32\drivers\rasstp.sys Kernel Driver Yes Manual
Running OK Normal No Yes
rdbss Redirected Buffering Sub Sysytem
e:\windows\system32\drivers\rdbss.sys File System Driver Yes System
Running OK Normal No Yes
rdpcdd RDPCDD
e:\windows\system32\drivers\rdpcdd.sysKernel Driver Yes System
Running OK Ignore No Yes
rdpdr Terminal Server Device Redirector Driver
e:\windows\system32\drivers\rdpdr.sys Kernel Driver Yes Manual
Running OK Normal No Yes
rdpencdd RDP Encoder Mirror Driver
e:\windows\system32\drivers\rdpencdd.sys Kernel Driver Yes System
Running OK Ignore No Yes
rdpwd RDP Winstation Driver
e:\windows\system32\drivers\rdpwd.sys Kernel Driver Yes Manual
Running OK Ignore No Yes
rsfx0103 RsFx0103 Driver
e:\windows\system32\drivers\rsfx0103.sys File System Driver No
Disabled Stopped OK Normal
No No
rspndr Link-Layer Topology Discovery Responder
e:\windows\system32\drivers\rspndr.sys Kernel Driver Yes Auto
Running OK Normal No Yes
s3cap Microsoft Emulated S3 Device Cap Driver
e:\windows\system32\drivers\s3cap.sys Kernel Driver No Disabled
Stopped OK Normal No No
sacdrv sacdrv
e:\windows\system32\drivers\sacdrv.sysKernel Driver No Boot
Stopped OK Ignore No No
sbp2port SBP-2 Transport/Protocol Bus Driver
e:\windows\system32\drivers\sbp2port.sys Kernel Driver No Disabled
Stopped OK Normal No No
secdrv Security Driver
e:\windows\system32\drivers\secdrv.sysKernel Driver Yes Auto
Running OK Normal No Yes
serenum Serenum Filter Driver
e:\windows\system32\drivers\serenum.sys Kernel Driver Yes Manual
Running OK Normal No Yes
serial Serial port driver
e:\windows\system32\drivers\serial.sys Kernel Driver Yes System
Running OK Ignore No Yes
sermouse Serial Mouse Driver
e:\windows\system32\drivers\sermouse.sys Kernel Driver No Disabled
Stopped OK Normal No No
sffdisk SFF Storage Class Driver
e:\windows\system32\drivers\sffdisk.sys Kernel Driver No Disabled
Stopped OK Normal No No
sffp_mmc SFF Storage Protocol Driver for MMC
e:\windows\system32\drivers\sffp_mmc.sys Kernel Driver No Manual
Stopped OK Normal No No
sffp_sd SFF Storage Protocol Driver for SDBus
e:\windows\system32\drivers\sffp_sd.sys Kernel Driver No Manual
Stopped OK Normal No No
sfloppy High-Capacity Floppy Disk Drive
e:\windows\system32\drivers\sfloppy.sys Kernel Driver No Manual
Stopped OK Normal No No
sisraid2 SiSRaid2
e:\windows\system32\drivers\sisraid2.sys Kernel Driver No Disabled
Stopped OK Normal No No
sisraid4 SiSRaid4
e:\windows\system32\drivers\sisraid4.sys Kernel Driver No Disabled
Stopped OK Normal No No
slowmemory Slow Memory Measurement
e:\windows\system32\drivers\slowmemory.sys Kernel Driver No Manual
Stopped OK Normal No No
smb Message-oriented TCP/IP and TCP/IPv6 Protocol (SMB session)
e:\windows\system32\drivers\smb.sys Kernel Driver Yes System
Running OK Normal No Yes
spldr Security Processor Loader Driver
e:\windows\system32\drivers\spldr.sys Kernel Driver Yes Boot
Running OK Critical No Yes
srv srv
e:\windows\system32\drivers\srv.sys File System Driver Yes Manual
Running OK Normal No Yes
srv2 srv2
e:\windows\system32\drivers\srv2.sys File System Driver Yes Manual
Running OK Normal No Yes
srvnet srvnet
e:\windows\system32\drivers\srvnet.sys File System Driver Yes Manual
Running OK Normal No Yes
storflt Disk VMBUS Acceleration Filter Driver
e:\windows\system32\drivers\storflt.sys Kernel Driver Yes Boot
Running OK Normal No Yes
storvsc storvsc
e:\windows\system32\drivers\storvsc.sys Kernel Driver No Disabled
Stopped OK Normal No No
storvsp Microsoft Virtual Disk Server Driver
e:\windows\system32\drivers\storvsp.sys Kernel Driver No Disabled
Stopped OK Normal No No
swenum Software Bus Driver
e:\windows\system32\drivers\swenum.sys Kernel Driver Yes Manual
Running OK Normal No Yes
symc8xx Symc8xx
e:\windows\system32\drivers\symc8xx.sys Kernel Driver No Disabled
Stopped OK Normal No No
sym_hi Sym_hi
e:\windows\system32\drivers\sym_hi.sys Kernel Driver No Disabled
Stopped OK Normal No No
sym_u3 Sym_u3
e:\windows\system32\drivers\sym_u3.sys Kernel Driver No Disabled
Stopped OK Normal No No
tcpip TCP/IP Protocol Driver
e:\windows\system32\drivers\tcpip.sys Kernel Driver Yes Boot
Running OK Normal No Yes
tcpip6 Microsoft IPv6 Protocol Driver
e:\windows\system32\drivers\tcpip.sys Kernel Driver No Manual
Stopped OK Normal No No
tcpipreg TCP/IP Registry Compatibility
e:\windows\system32\drivers\tcpipreg.sys Kernel Driver Yes Auto
Running OK Normal No Yes
tdpipe TDPIPE
e:\windows\system32\drivers\tdpipe.sys Kernel Driver No Manual
Stopped OK Normal No No
tdtcp TDTCP
e:\windows\system32\drivers\tdtcp.sys Kernel Driver Yes Manual
Running OK Normal No Yes
tdx NetIO Legacy TDI Support Driver
e:\windows\system32\drivers\tdx.sys Kernel Driver Yes System
Running OK Normal No Yes
termdd Terminal Device Driver
e:\windows\system32\drivers\termdd.sysKernel Driver Yes System
Running OK Normal No Yes
tssecsrv Terminal Services Security Filter Driver
e:\windows\system32\drivers\tssecsrv.sys Kernel Driver Yes Manual
Running OK Ignore No Yes
tunnel Microsoft IPv6 Tunnel Miniport Adapter Driver
e:\windows\system32\drivers\tunnel.sys Kernel Driver Yes Manual
Running OK Normal No Yes
uagp35 Microsoft AGPv3.5 Filter
e:\windows\system32\drivers\uagp35.sys Kernel Driver No Manual
Stopped OK Normal No No
udfs udfs
e:\windows\system32\drivers\udfs.sys File System Driver No Disabled
Stopped OK Normal No No
uliagpkx Uli AGP Bus Filter
e:\windows\system32\drivers\uliagpkx.sys Kernel Driver No Manual
Stopped OK Normal No No

© 2009 Hewlett-Packard Company. All rights reserved.

uliahci uliahci
e:\windows\system32\drivers\uliahci.
sys Kernel Driver No Disabled
Stopped OK Normal No No
ulsata UlSata
e:\windows\system32\drivers\ulsata.
sys Kernel Driver No Disabled
Stopped OK Normal No No
ulsata2 ulsata2
e:\windows\system32\drivers\ulsata
2.sys Kernel Driver No Disabled
Stopped OK Normal No No
umbus UMBus Enumerator Driver
e:\windows\system32\drivers\umbus
.sysKernel Driver Yes Manual
Running OK Normal No Yes
umpass Microsoft UMPass Driver
e:\windows\system32\drivers\umpas
s.sys Kernel Driver No Disabled
Stopped OK Normal No No
usbccgp Microsoft USB Generic
Parent Driver
e:\windows\system32\drivers\usbcc
gp.sys Kernel Driver Yes Manual
Running OK Normal No Yes
usbcir eHome Infrared Receiver
(USBCIR)
e:\windows\system32\drivers\usbcir.
sys Kernel Driver No Disabled
Stopped OK Normal No No
usbehci Microsoft USB 2.0 Enhanced
Host Controller Miniport Driver
e:\windows\system32\drivers\usbeh
ci.sys Kernel Driver Yes Manual
Running OK Normal No Yes
usbhub Microsoft USB Standard Hub
Driver
e:\windows\system32\drivers\usbhu
b.sys Kernel Driver Yes Manual
Running OK Normal No Yes
usbohci Microsoft USB Open Host
Controller Miniport Driver
e:\windows\system32\drivers\usboh
ci.sys Kernel Driver Yes Manual
Running OK Normal No Yes
usbprint Microsoft USB PRINTER Class
e:\windows\system32\drivers\usbpri
nt.sys Kernel Driver No Disabled
Stopped OK Normal No No
usbstor USB Mass Storage Driver
e:\windows\system32\drivers\usbsto
r.sys Kernel Driver No Manual
Stopped OK Normal No No
usbuhci Microsoft USB Universal Host
Controller Miniport Driver
e:\windows\system32\drivers\usbuh
ci.sys Kernel Driver Yes Manual
Running OK Normal No Yes
vga vga
e:\windows\system32\drivers\vgapn
p.sys Kernel Driver No Manual
Stopped OK Ignore No No
vgasave VgaSave
e:\windows\system32\drivers\vga.sy
s Kernel Driver Yes System
Running OK Ignore No Yes
viaide viaide
e:\windows\system32\drivers\viaide.
sys Kernel Driver No Disabled
Stopped OK Critical No No
vid Virtualization Infrastructure Driver
e:\windows\system32\drivers\vid.sys
Kernel Driver No Disabled
Stopped OK Normal No No
vmbus VMBus
e:\windows\system32\drivers\vmbus
.sysKernel Driver No Disabled
Stopped OK Normal No No
volmgr Volume Manager Driver
e:\windows\system32\drivers\volmgr
.sysKernel Driver Yes Boot
Running OK Critical No Yes
volmgrx Dynamic Volume Manager
e:\windows\system32\drivers\volmgr
x.sys Kernel Driver Yes Boot
Running OK Critical No Yes
volsnap Storage volumes
e:\windows\system32\drivers\volsna
p.sys Kernel Driver Yes Boot
Running OK Critical No Yes
vsmraid vsmraid
e:\windows\system32\drivers\vsmrai
d.sys Kernel Driver No Disabled
Stopped OK Normal No No
wacompen Wacom Serial Pen HID
Driver
e:\windows\system32\drivers\waco
mpen.sys Kernel Driver No
Disabled Stopped OK Normal
No No
wanarp Remote Access IP ARP Driver
e:\windows\system32\drivers\wanar
p.sys Kernel Driver No Manual
Stopped OK Normal No No
wanarpv6 Remote Access IPv6 ARP
Driver
e:\windows\system32\drivers\wanar
p.sys Kernel Driver Yes System
Running OK Normal No Yes
wd Microsoft Watchdog Timer Driver
e:\windows\system32\drivers\wd.sys
Kernel Driver No Disabled
Stopped OK Normal No No
wdf01000 Kernel Mode Driver
Frameworks service
e:\windows\system32\drivers\wdf01
000.sys Kernel Driver Yes Boot
Running OK Normal No Yes
wmiacpi Microsoft Windows
Management Interface for ACPI
e:\windows\system32\drivers\wmiac
pi.sys Kernel Driver No Disabled
Stopped OK Normal No No
ws2ifsl Winsock IFS driver
e:\windows\system32\drivers\ws2ifsl
.sysKernel Driver No Disabled
Stopped OK Normal No No

[Signed Drivers]

Device Name Signed Device Class
Driver Version Driver Date
Manufacturer INF Name Driver
Name Device ID
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not
Available
STORAGE\VOLUME\1&19F7E59C
&0&GPTPARTITION{4B93D93E-48C3-
4F39-A0CF-A277301EF17C}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not
Available
STORAGE\VOLUME\1&19F7E59C
&0&GPTPARTITION{E97BE405-826B-
41B8-BD8B-7DA720D80CD4}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not
Available
STORAGE\VOLUME\1&19F7E59C
&0&GPTPARTITION{B837FE09-0B76-
46B0-9B46-EC8CCEAEBCC7}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not
Available
STORAGE\VOLUME\1&19F7E59C
&0&GPTPARTITION{6305769C-C2BE-
424C-87D6-BD9A80DED490}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not
Available
STORAGE\VOLUME\1&19F7E59C
&0&GPTPARTITION{BC57CDEA-0704-
4E0D-9FA2-1B46E8394B6A}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not
Available
STORAGE\VOLUME\1&19F7E59C
&0&GPTPARTITION{0DCCBF09-E775-
4B32-AAF9-6F142AC9B848}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not
Available
STORAGE\VOLUME\1&19F7E59C
&0&GPTPARTITION{CA87ED56-BF8D-
425C-A399-4D8360F84559}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not
Available
STORAGE\VOLUME\1&19F7E59C
&0&GPTPARTITION{3FF12247-A6EF-
46A5-ABEC-1C753DAF314F}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not
Available
STORAGE\VOLUME\1&19F7E59C
&0&GPTPARTITION{7B2BA133-4AD4-
4504-B975-7D3692E0C897}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not
Available
STORAGE\VOLUME\1&19F7E59C
&0&GPTPARTITION{7D760F2E-9742-
49DF-AFA3-DA6487CCA0AC}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not
Available
STORAGE\VOLUME\1&19F7E59C
&0&GPTPARTITION{B2613F2F-909C-
4573-A944-E976A799881D}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not
Available
STORAGE\VOLUME\1&19F7E59C
&0&GPTPARTITION{DAAD5AD3-0B98-
44D1-80D1-50097307EAEF}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not
Available
STORAGE\VOLUME\1&19F7E59C

© 2009 Hewlett-Packard Company. All rights reserved.

&0&GPTPARTITION{D9EE2691-CA29-4C29-A68B-95AD50D76020}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{B4970699-04E8-4AF9-9C5A-7D93326B01AB}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{EA060795-295C-4E56-98C1-8FAB923907B5}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{99EAA0CC-6CA0-4810-A145-CB2E0E073FBB}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{9506CFDE-6749-4037-9FDD-5E892744B18A}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{AF5310F7-8546-4BE4-858E-87B9514F0AEC}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{7BCCAF13-4450-430D-A465-A9ED3ACB5639}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{F4E845E9-26A6-4799-BF28-3B7E2DDFA362}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{393C53D5-74AA-443B-980C-A40D36AD7BB6}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{D6EB380C-2CDC-48E5-A06F-D92B725BC61F}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{8381BBF3-264D-4B51-A0F5-BDDCB2BC98B1}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{39483FBE-FC66-4E50-80F2-99EC3CAB8B96}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{6715A772-F637-4712-B78B-35A67E5E0A26}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{9AD23D85-894E-439C-B391-DFF2A56191AA}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{6535A8AF-F973-4EAA-AA2B-0EB91B03C912}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{81310378-E88F-414A-B791-45D2E228A64E}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{6669E91A-F25A-49A6-B6CD-EB538DF15CCB}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{7A37522F-AB92-45ED-A362-6858060DA500}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{AAD9F068-D931-49D2-943D-EF0589352057}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{4EF01AD5-228D-4BE3-AAEC-288211529D57}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{C98E1B60-7A7F-43B1-AB32-1F67EEBB9A48}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{44EC9F2C-3777-4CE0-ADCE-D29B53DA5C0D}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{C4A487D6-2EDC-4A97-A860-9EE611067E09}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{2E9F1832-182D-4DA1-9DF0-602FA3CEFFA5}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{5F0AB577-F26D-4AB5-BF7A-E69E51769BC7}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{3065697A-A192-4C3C-A1A3-24D1364F11E3}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{8632D096-F56E-4AC1-8364-FBDB65A94C6A}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{849E5745-73ED-4DBC-9F42-E0D85F5F0726}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{8B77636A-F646-428A-9D06-1FD8C2818DA1}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{20D8E37B-39F7-42E8-8F0B-0DDA6B2E97AE}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{3F44F9B5-6409-46BE-86A9-3ECE984499EF}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{84D53AC7-C90B-428B-B68E-EF14C4E4C33A}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C

© 2009 Hewlett-Packard Company. All rights reserved.

&0&GPTPARTITION{EF8C7D61-1C34-458B-A7A8-F057D43F7858}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{0AD05864-5242-4969-B319-13F89C14A38F}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{3CB1302D-06B3-4553-9EBF-59102E07F959}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{4E4222C8-73D8-4950-8169-67AC47019915}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{5A4F639A-A6AA-4E77-A346-BBEF877438C0}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{01090A46-8C71-49EC-9FA7-8988D7D6DD1C}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{59889536-EB31-4BCA-B3F8-61D4425517F1}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{329EE2CF-98DD-434A-AE71-C15575B17C4D}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{253F20B7-0B18-48F0-8866-B1999DA63E64}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{5380E3B8-D843-42C4-A2F0-E068190D7D5B}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{D40B8FF6-DCFA-49C9-86B1-742136996983}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{27A3DC16-31F0-4E0E-8C07-78180CCA4666}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{17ED7D52-9DB7-4A02-AC30-F83BAFDC4012}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{97004622-03B0-44D4-9B1E-72DA3676C09F}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{3CB905CD-92C7-4AC5-9CF3-C7F9BA22F3AE}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{E7826464-A3D4-44E5-856E-30ADBF95511C}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{1F51C4EB-A130-4EF4-9BF7-8A0DDD12E2BB}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{77E821F4-DD65-43FE-ABB1-CE020E650E31}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{16B14FE5-FC84-4073-B792-503808C62DDD}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{CC67D9A2-EA89-47F4-A7A3-CD96D84903B4}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{53106143-5B39-4BBC-9BAD-21578D8CFD8C}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{DAE75721-217E-410B-BF8D-34EB32FC1AEB}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{ACE7A525-17FD-4C27-B2E4-47DAF8A89BDA}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{89482684-66C3-42C5-8B88-D5AFE353B244}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{4DE943C6-3C67-461E-B733-CAFBB840C90C}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{033474E5-D265-47E2-A523-83C5DFE54C3B}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{4F110C36-E43C-4EC4-8F2C-CA51BB97B701}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{03DCBD39-19F5-4EDD-9408-490CB2E71081}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{F93CC9CA-E18A-405F-863C-39E4C289C014}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{7B9C084A-3216-4D66-BFFE-85943C7C3B0B}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{CA3F5D90-9AAC-430B-80B5-2476DC8186C8}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{542BF55B-E560-48AA-B1E0-B5159302B237}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C

© 2009 Hewlett-Packard Company. All rights reserved.

&0&GPTPARTITION{0B2B946B-F2D3-453C-9C1D-6FE90A2F57C7}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{20B7213F-3841-49F8-9EF8-ABB3CC807A6A}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{5E513B1B-B0B1-4716-B109-8ED1EDE39763}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{7EC122C2-5F31-4C0B-97B8-884ADECE2622}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{E4134BD4-3227-491A-998C-1762AE09BE06}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{C9985740-9B89-4085-A75F-7CCD814CC45A}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{97E95094-77A9-4A18-95F6-A9F60694FDD1}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{BF00D0A0-94B5-4BF6-B3D9-4C1F9ED8E0EF}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{01915DEB-29D8-4B89-8B0D-06804968A4C9}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{5EE68E7F-1358-447C-A539-6759AA08BD12}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{122866FF-EC16-4AFA-809E-EBCB235AD77A}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{8C217342-74A3-4EE3-B48D-F176A456C22B}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{DBEE5F03-A629-46C7-978B-111061ADE2E8}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{6D895A40-C3FE-48ED-A965-75BF4E9B07AE}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{57849715-DA85-4F3A-81CF-13FBF3EF2335}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{9A4C6299-56E8-4528-89B5-42B4F69355AA}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{9BEC5AFC-5B12-41A7-8F09-3503F1CD6ACA}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{D445B5B5-5A1B-4118-BF35-723FEFD5998A}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{93FBD65B-2E60-4173-86FC-1A87273DA23A}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{E982CE4B-8D01-43D4-99D4-28BB62DA402E}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{566F5BD4-D777-4AE8-897B-053DA618DD33}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{C57FA9D1-F049-4ED8-9700-F85DCFB060EE}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{F62CC78B-6F61-4BCD-B05A-477E0A235126}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{DC7EF092-0751-4AAA-A78B-566491CDD506}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{F1457548-19E5-4BD7-BB48-C69B7961C1E7}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{D3188EFE-C634-4A72-9325-028703039D91}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{3AC04FB8-CC5D-45A0-9501-7090F79E1298}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{9890E5A9-F28B-4CA7-83B9-A10B3963DC82}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{00B20FA2-54CC-4714-8B39-DA4E43AB8FBD}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{18E51B35-FD4B-4AB1-9501-952AFFEAFFE8}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{48F35AB4-3FE9-4E14-B498-C2B91779491D}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{7FFBCB35-2CBB-41FB-A852-DD9D373F1D33}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C

© 2009 Hewlett-Packard Company. All rights reserved.

&0&GPTPARTITION{79B9E55C-E010-4FD8-A9B0-35B578D95ADB}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{024AFB5F-A8AE-4658-AAA6-9DDE12FD2A4B}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{A6F95A5B-A3F8-4875-B910-599D786B4327}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{24A6707A-F3E8-4F20-B626-B88E68484713}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{C1C98B13-039B-49EA-95DE-758003E4C894}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{FFB46B07-E6DE-4F3C-9E63-49D30E7B14A7}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{A485345B-F932-48DD-A814-64F0A51ECBA3}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{C4148FC4-57F4-41D2-86B2-7516F4005423}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{25937809-6E06-4B17-8197-F050C21454C9}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{E278FACE-675B-45AF-8CE7-20DD46237E31}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{EA621784-FC0B-42DC-8EB1-96E977B0F430}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{FA665EB2-45EC-4EBD-876A-E14747E6C491}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{CF45ECF4-2806-4B73-8F60-BA8DEDEC669E}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{FE585E36-703B-4E6A-B659-483EAB8145CD}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{CEAD7738-1ABC-4C4C-B89D-5B44948C6EEA}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{DFA608AB-CCB0-4D0E-86B9-7A73D1DC493C}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{8B38CAFD-E5DB-4FB1-B1D7-765BE7A06E29}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{70D0618A-49F6-429A-8563-7CD3760A5A57}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{C68968AD-F434-4290-AEE9-4B38D5C9C113}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{25F0B673-85BF-49EE-8A72-0635CB85A622}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{FCD12613-C53A-48E2-85E0-74AE69A180D6}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{CF1F4CB3-937A-44B9-B462-C32CD032F513}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{46660FDE-8B82-459A-8B33-D79F59E81504}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{AB7EAA44-AD2D-4FB3-B266-2A06E11527C2}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{8526777F-5C97-45EB-B218-E74C08FFAF4B}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{6E6507B5-5740-43BD-8E51-2A2A1E98C730}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{5CBF53F2-F35E-478F-BD24-FE622F19CFA3}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{A364B85E-B89E-433C-872B-7ADB0545A9CF}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{B2C1B832-5C66-4F09-BD15-B8194F862738}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{A7F9DEFA-F4B0-41A6-A1D0-B24E2F104DCC}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{B6B08496-3D0D-4AB3-8A43-C2817CD51F43}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{A8F50FCC-E2FC-4870-8A86-30F389CF94D0}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C

© 2009 Hewlett-Packard Company. All rights reserved.

&0&GPTPARTITION{F00692A7-975B-4773-A97D-97152C307015}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{C872E046-5FC8-4C72-A67A-0A806FD65EDD}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{AD56DEA2-B89E-466A-A653-DD7F534C3C34}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{188A23B6-9130-4B73-806D-20F3485CF2CA}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{225AEB5F-9774-41B2-AA48-A1AAE3D82D02}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{6DE40243-39FD-4597-ABDD-77620CC012BD}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{A268F563-EA25-4A2E-8CEE-9B6ECD4C7F94}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{06F0A360-74EA-4E3A-96AB-707C1D61CC95}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{7A8012E2-02CD-4BEE-842B-BCAB6764846D}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{310F7FA9-7073-4459-A1FF-236B9F00BF09}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{27ED1E7E-72E6-40A1-961B-D55282E85BE9}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{6E911989-54BF-44FA-BAEB-1B1EEC735D27}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{108EE89B-3E59-4CB6-8D88-2C08C0BA6E70}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{7733D911-9DCD-4BB2-B9E6-5EF17AE8C0B7}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{5AC93FD0-9DC4-499B-8F9E-BDFD4CA567BA}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{B2FF5CD8-BCCB-4F9B-BB4D-2A4E16FC3519}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{648210E4-CC9C-472A-A11F-6B3AD658812E}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{CE2B2A2B-60C9-4C55-A7BA-2007A7CAE844}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{4AF3BDB4-813D-4A40-B53F-67A17D6209CB}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{BCCC3B5C-C99E-4047-8AE7-9A25A93CC491}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{32B7B6CB-598F-4D32-BA4B-668D4575B37B}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{2463304A-10DE-44B8-86CF-A96C30D576D7}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{25D6E0B2-F8E6-4576-914E-33943257603F}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{6AECC633-669A-456F-96D2-D965C67969AF}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{9FC12391-1F10-40C0-8F29-0513CC7D9A67}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{6857BE3D-B031-4FC6-BEEC-651C92924B77}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{579CA2B4-975E-462A-A882-760930D41195}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{6E2AA3C2-4DA5-4178-8871-6147C3E7C174}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{68D88B5B-5983-4DCF-88C5-9F157B51E3B3}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{201C973C-7539-49BC-B7D3-FEE90905EA90}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{36BF22A5-D46E-4853-884B-235A3639A168}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{441A23EB-0A9F-4CF8-B6BF-FAD3833C3302}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C

© 2009 Hewlett-Packard Company. All rights reserved.

&0&GPTPARTITION{D0B4366D-2A8A-4199-8723-9480C2CF7771}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{AFBEFDD4-7EBC-4672-9A16-7D06A421A5C8}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{280E4EFB-836A-47A6-BA11-2364F45C03C6}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{8944E0EC-53ED-40B9-A81A-D892FF0AAAAB}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{93089393-7D82-4E0A-B50B-CE1A1657AC76}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{ED3B25DA-8DC5-4904-8548-301EE3F75A94}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{9C298928-73FC-49B9-BB32-95841EFFDE89}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{DE9753F4-37E6-44E6-831B-E3F886190D6A}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{586056B6-5B1F-475D-A2A6-5E59463AB7AA}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{34C7F75D-852C-4EDA-BD11-3C383E25E725}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{053DFE4E-E88A-4EA4-8251-DFDE67B28228}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{AD089DEB-1CCC-4B72-996F-E23506FFCCB9}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{1EA66C2A-0396-40C7-9FCC-077470457E7E}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{E0F6A31C-21C3-4EE3-AF92-D56374626CA8}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{FADDA5B2-5641-4BDB-A9BF-033712D80733}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{9C344574-A0B9-430C-9099-B5B7310493E4}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{33B68B03-25D6-4AEF-98F4-171AE19392A2}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{71AACF7A-EA67-4B9D-AE75-BE4151CF275A}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{F9450BFB-6689-40FE-AA6A-355CB515AA82}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{C68A0209-7143-483E-A1DE-2B27A228D9D6}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{1DF9757F-3B1A-4FB2-B474-AEDB34BE0292}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{7BEE41E7-7308-4B30-95DF-39668C8F9AE7}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{8479A2CC-B24A-40B4-ADFC-BCF0BA78607D}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{561FF517-19F2-4A9E-A7D3-AA6CF39F81CB}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{43F6885C-CDF0-4676-876E-1114F8C2CDC5}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{F614632F-4200-462F-8BAA-BFE74EDA1801}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{BDF0657D-6B6E-4873-BC9F-393FF54AF69E}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{D6C10FC8-E792-4766-BC6F-B29E719B1D79}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{2C86DFFF-B156-4DFE-8865-75282F3E9A93}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{58375FD4-9115-4935-BBCB-084E2FD47E06}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{31823C1D-52EB-43B6-BBF4-B2B510B71A53}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{D2307346-6AAB-4739-A4C5-820C33459C73}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C

© 2009 Hewlett-Packard Company. All rights reserved.

&0&GPTPARTITION{F63DACD5-FF91-44B6-B67E-3A176B054BCB}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{DE2F94FB-0DE7-44D4-AAEA-CB0BF1DB930B}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{6592E45E-BB73-4AE4-8582-08ADF51802F3}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{8A5F406F-A49C-4455-80C1-D59B056772A4}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{5ADD550A-59CE-4520-B089-F025C8F84992}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{72BFE2A8-FBC1-4D37-9CDB-EFB44E18F3E3}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{A08C4893-DF62-424A-9696-B773917B1A90}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{60316783-6525-4181-913E-8E9D73C89F4B}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{E85C5ABB-1249-4D88-8922-0AF4E438E0A3}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{07884335-D7DF-43DE-9769-B14A0A510823}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{11EBD1CA-5EBF-45DC-8653-B8348B6D4AA7}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{C0003552-1299-4F3E-8A52-C2A1D2723741}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{CC6C1838-E2B2-4A1F-96EF-CF950779E344}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{D97CD8B2-E30A-43E2-8D0C-CAB85BB2C25F}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{D016D95D-553E-41E1-B7EA-4FE9CE4A93D8}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{DD7ED067-E480-4204-A8B4-26C2B26ADF5C}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{8E6F3636-3B21-4503-9AF3-C1243D878F25}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{40228CE5-AECB-447C-8439-C17DDD9AF4F0}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{F4659F24-CAB3-4412-A985-5918C55B57AA}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{A61E3945-3651-443F-88A5-0DB9E3B061E7}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{53440FF8-A74E-4C07-BD66-4603ACAA033E}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{FD111149-643B-4EE1-AB0B-CBDC75FF33BC}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{82A3BC5E-AD25-487D-B3EE-45D620CF5D55}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{F4846BBE-8E3A-4AA0-9B3E-60D14E68CC86}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{485D4597-9B05-43CC-87D3-C97BD44E55F7}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{4A9B0C93-86C2-4D3E-8CEB-5128CD20496A}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{D4626431-D202-460B-BA5F-5BC99CA78034}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{77F8BA73-B897-4D9F-9BC1-D9482C5900A8}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{F9CD2A69-775B-4928-85E6-92E4DDCE9E77}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{B7217DAE-7C8B-41D7-8D49-D5AFCB08B26E}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{2CB73643-F08E-4A9C-A632-7E0EBD527F24}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{95D0675B-EFA9-48BB-908E-C55D645FF4AA}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C

© 2009 Hewlett-Packard Company. All rights reserved.

&0&GPTPARTITION{219C1767-8501-4A15-A89B-EFAC76BB2ADD}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{C680BBBA-D43F-48B7-B741-C11E0862CC4C}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&SIGNATURE61BAA311OFFSET105B900000LENGTHB9A00000
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&SIGNATURE61BAA311OFFSETA41100000LENGTH61A800000
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&SIGNATURE61BAA311OFFSET100000LENGTHA41000000
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&SIGNATURE6B5401A7OFFSET100000LENGTH878900000
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&SIGNATUREB4BCB4BCOFFSET7F8000LENGTH877FF8000
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&SIGNATUREB4BCB4BCOFFSET7E00LENGTH7D0400
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{B8D48A96-8C86-4733-A121-BFBDD21B2791}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{EE5E8FF2-B121-42B7-92AF-A97758041581}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{286FB685-CFF6-48C2-9C8C-DDCB046A9D4D}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{0E225241-2373-4684-82BC-7BFDCF8568D7}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{C87DF76C-B068-439F-8B62-74B3959F064F}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{B5DB5C3B-87AC-434F-90D9-E501CC911DC2}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{D827589B-8A98-497F-AF60-3C2A8B6A33EA}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{CCAF99DA-ACC7-490F-80B0-3DE56C62DE45}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{56C50E89-9F8D-4222-AE7A-99201D9DCB34}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{35E5DBD7-CE7A-433A-A547-DBE554E70F10}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{819D756D-9B6D-482F-B3C7-AC28D5989CF4}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{1B184699-1B86-4FA1-9712-1DA5DC7A8757}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{8E497B1E-C64E-44A1-8538-7FDDC1D387B3}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{EF69DDE7-93C3-47FE-BE5C-8D1E7B2434C9}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{65249633-EDAB-4CD2-9E11-A381BEB47B0E}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{EBC9F370-8C83-415B-A628-430AA9C6761E}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{1E09EB86-EF06-4D33-9B02-97F07F09ADA2}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{DD8A5E1D-0FC6-46C6-8E32-C523023735BA}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{9B6385F1-89C6-46B2-AC4B-33F18BA7DE20}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{C3F83F39-A16A-40F5-913C-A1A7E510022F}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{72DD5128-64FA-4EFE-91D6-0DB22F906DA8}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{6727462E-A587-4613-813B-9D9ED70176BC}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{10FEBC8A-73FA-4E52-9B61-0AE80039767B}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{5D959D1A-B98C-456E-BA9C-B633122D83A6}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C

© 2009 Hewlett-Packard Company. All rights reserved.

&0&GPTPARTITION{F00A53C7-A36B-46D5-BA64-F6C650568285}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{359C996B-DC52-4991-AD88-7E883AFC345E}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{018CC7AF-D3D4-4933-AD6B-029812814064}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{74E6BBFB-D218-497A-A9D3-A5A2CA83F6F6}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{5AB42361-3E38-4951-80C9-C3D086AF8982}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{031F036F-3BAF-4843-9D4A-4BFD4E49A29E}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{F1A497A9-31EF-4548-A865-84669324F8CB}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{5B1D70A5-DD3E-4BB9-B97C-FD815130BB21}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{73551EDB-01EF-4C3C-9E5E-FC98D064CAB6}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{B042D8AD-2CED-47F7-88C8-660381E1A6AA}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{94F6CD66-5349-459D-8DAA-C53B8928074E}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{4E934B62-3E1C-4D07-904F-DDFBE4C32B7D}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{16ABD90D-9248-4C23-B399-68457B92CCD3}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{7A21FB45-A808-4E07-B1C0-CD1637E994A2}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{A5409E1D-DB59-46FD-8ADB-5411024C8DE4}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{462141CE-7A0F-4E6C-BF08-7C153A94B93E}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{2D9BDE9D-5C95-4041-BFD4-478693D33569}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{5BE509D4-DE5C-4EA5-AB06-86174BF3D112}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{B01CAD0F-F9DF-4AA7-8610-0BEF59FB7985}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{027AD50F-F043-4B78-BFAA-7231F394288E}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{A75F48B0-0470-4063-B10A-C13D54E40905}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{4896C760-23AD-47D6-A3AE-99B722535A53}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{0F19734F-C18F-4356-BACD-9B7C821ADB5C}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{735DEBBD-457E-446D-8DB0-E95B1D6052F9}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{4A54F609-31D2-4082-B6B3-A3BF90082EE4}
Generic volume No VOLUME
6.0.6001.18000 6/21/2006
Microsoft volume.inf Not Available
STORAGE\VOLUME\1&19F7E59C&0&GPTPARTITION{B49F3761-980C-4B7E-9975-A881101C86CD}
Volume Manager No SYSTEM
6.0.6001.18000 6/21/2006
(Standard system devices)
machine.inf Not Available
ROOT\VOLMGR\0000
UMBus Enumerator No SYSTEM
6.0.6001.18000 6/21/2006
Microsoft umbus.inf Not Available
UMB\UMB\1&841921D&0&TSBUS
UMBus Root Bus Enumerator No
SYSTEM 6.0.6001.18000
6/21/2006 Microsoft umbus.inf
Not Available
ROOT\UMBUS\0000
Microsoft System Management BIOS Driver No SYSTEM 6.0.6001.18000
6/21/2006 (Standard system devices) machine.inf Not Available
ROOT\SYSTEM\0002
AMD Special Tools Driver No
SYSTEM 1.7.11.173 1/26/2009
Advanced Micro Devices
oem6.inf Not Available
ROOT\SYSTEM\0001
RAS Async Adapter No NET
6.0.6001.18000 6/21/2006
Microsoft netrasa.inf Not Available SW\{EEAB7790-C514-11D1-B42B-00805FC1270E}\ASYNCMAC
Plug and Play Software Device Enumerator No SYSTEM
6.0.6001.18000 6/21/2006
(Standard system devices)
machine.inf Not Available
ROOT\SYSTEM\0000
Terminal Server Mouse Driver No
SYSTEM 6.0.6001.18000
6/21/2006 (Standard system devices) machine.inf Not Available
ROOT\RDP_MOU\0000
Terminal Server Keyboard Driver No
SYSTEM 6.0.6001.18000
6/21/2006 (Standard system devices) machine.inf Not Available
ROOT\RDP_KBD\0000

© 2009 Hewlett-Packard Company. All rights reserved.

Terminal Server Device Redirector No
SYSTEM 6.0.6001.18000
6/21/2006 (Standard system
devices) machine.inf Not Available
ROOT\RDPDR\0000
WAN Miniport (SSTP) No NET
6.0.6001.18000 6/21/2006
Microsoft netsstpa.inf Not
Available
ROOT\MS_SSTPMINIPORT\0000
WAN Miniport (PPTP) No NET
6.0.6001.18000 6/21/2006
Microsoft netrasa.inf Not
Available
ROOT\MS_PPTPMINIPORT\0000
WAN Miniport (PPPOE) No NET
6.0.6001.18000 6/21/2006
Microsoft netrasa.inf Not
Available
ROOT\MS_PPPOEMINIPORT\0000
WAN Miniport (IPv6) No NET
6.0.6001.18000 6/21/2006
Microsoft netrasa.inf Not
Available
ROOT\MS_NDISWANIPV6\0000
WAN Miniport (IP) No NET
6.0.6001.18000 6/21/2006
Microsoft netrasa.inf Not
Available
ROOT\MS_NDISWANIP\0000
WAN Miniport (Network Monitor)No
NET 6.0.6001.18000 6/21/2006
Microsoft netrasa.inf Not
Available
ROOT\MS_NDISWANBH\0000
WAN Miniport (L2TP) No NET
6.0.6001.18000 6/21/2006
Microsoft netrasa.inf Not
Available
ROOT\MS_L2TPMINIPORT\0000
Kernel Mode Driver Frameworks service
Not Available LEGACYDRIVER
Not Available Not Available Not
Available Not Available Not
Available
ROOT\LEGACY_WDF01000\0000
Remote Access IPv6 ARP Driver Not
Available LEGACYDRIVER Not
Available Not Available Not
Available Not Available Not
Available
ROOT\LEGACY_WANARPV6\0000
Storage volumes Not Available
LEGACYDRIVER Not Available
Not Available Not Available Not
Available Not Available
ROOT\LEGACY_VOLSNAP\0000
Dynamic Volume Manager Not
Available LEGACYDRIVER Not
Available Not Available Not
Available Not Available Not
Available
ROOT\LEGACY_VOLMGRX\0000
VgaSave Not Available
LEGACYDRIVER Not Available
Not Available Not Available Not
Available Not Available
ROOT\LEGACY_VGASAVE\0000
Terminal Services Security Filter Driver
Not Available LEGACYDRIVER
Not Available Not Available Not
Available Not Available Not
Available
ROOT\LEGACY_TSSECSRV\0000
NetIO Legacy TDI Support Driver Not
Available LEGACYDRIVER Not
Available Not Available Not
Available Not Available Not
Available ROOT\LEGACY_TDX\0000
TDTCP Not Available
LEGACYDRIVER Not Available
Not Available Not Available Not
Available Not Available
ROOT\LEGACY_TDTCP\0000
TCP/IP Registry Compatibility Not
Available LEGACYDRIVER Not
Available Not Available Not
Available Not Available Not
Available
ROOT\LEGACY_TCPIPREG\0000
TCP/IP Protocol Driver Not Available
LEGACYDRIVER Not Available
Not Available Not Available Not
Available Not Available
ROOT\LEGACY_TCPIP\0000
Disk VMBUS Acceleration Filter Driver
Not Available LEGACYDRIVER
Not Available Not Available Not
Available Not Available Not
Available
ROOT\LEGACY_STORFLT\0000
Security Processor Loader Driver Not
Available LEGACYDRIVER Not
Available Not Available Not
Available Not Available Not
Available
ROOT\LEGACY_SPLDR\0000
Message-oriented TCP/IP and
TCP/IPv6 Protocol (SMB session) Not
Available LEGACYDRIVER Not
Available Not Available Not
Available Not Available Not
Available ROOT\LEGACY_SMB\0000
Security Driver Not Available
LEGACYDRIVER Not Available
Not Available Not Available Not
Available Not Available
ROOT\LEGACY_SECDRV\0000
Link-Layer Topology Discovery
Responder Not Available
LEGACYDRIVER Not Available
Not Available Not Available Not
Available Not Available
ROOT\LEGACY_RSPNDR\0000
RDP Winstation Driver Not Available
LEGACYDRIVER Not Available
Not Available Not Available Not
Available Not Available
ROOT\LEGACY_RDPWD\0000
RDP Encoder Mirror Driver Not
Available LEGACYDRIVER Not
Available Not Available Not
Available Not Available Not
Available
ROOT\LEGACY_RDPENCDD\0000
RDPCDD Not Available
LEGACYDRIVER Not Available
Not Available Not Available Not
Available Not Available
ROOT\LEGACY_RDPCDD\0000
Remote Access Auto Connection Driver
Not Available LEGACYDRIVER
Not Available Not Available Not
Available Not Available Not
Available
ROOT\LEGACY_RASACD\0000
QoS Packet Scheduler Not Available
LEGACYDRIVER Not Available
Not Available Not Available Not
Available Not Available
ROOT\LEGACY_PSCHED\0000
PEAUTH Not Available
LEGACYDRIVER Not Available
Not Available Not Available Not
Available Not Available
ROOT\LEGACY_PEAUTH\0000
Null Not Available LEGACYDRIVER
Not Available Not Available Not
Available Not Available Not
Available
ROOT\LEGACY_NULL\0000
NSI proxy service Not Available
LEGACYDRIVER Not Available
Not Available Not Available Not
Available Not Available
ROOT\LEGACY_NSIPROXY\0000
NETBT Not Available
LEGACYDRIVER Not Available
Not Available Not Available Not
Available Not Available
ROOT\LEGACY_NETBT\0000
NDProxy Not Available
LEGACYDRIVER Not Available
Not Available Not Available Not
Available Not Available
ROOT\LEGACY_NDPROXY\0000
NDIS System Driver Not Available
LEGACYDRIVER Not Available
Not Available Not Available Not
Available Not Available
ROOT\LEGACY_NDIS\0000
MultEvent Driver Not Available
LEGACYDRIVER Not Available
Not Available Not Available Not
Available Not Available
ROOT\LEGACY_MULTEVENT\000
0
ISA/EISA Class Driver Not Available
LEGACYDRIVER Not Available
Not Available Not Available Not
Available Not Available
ROOT\LEGACY_MSISADRV\0000
Windows Firewall Authorization Driver
Not Available LEGACYDRIVER
Not Available Not Available Not
Available Not Available Not
Available
ROOT\LEGACY_MPSDRV\0000
Mount Point Manager Not Available
LEGACYDRIVER Not Available
Not Available Not Available Not
Available Not Available
ROOT\LEGACY_MOUNTMGR\000
0
Link-Layer Topology Discovery Mapper
I/O Driver Not Available
LEGACYDRIVER Not Available
Not Available Not Available Not
Available Not Available
ROOT\LEGACY_LLTDIO\0000
KSecDD Not Available
LEGACYDRIVER Not Available
Not Available Not Available Not
Available Not Available
ROOT\LEGACY_KSECDD\0000
HTTP Not Available
LEGACYDRIVER Not Available
Not Available Not Available Not
Available Not Available
ROOT\LEGACY_HTTP\0000
HpCISSs Not Available
LEGACYDRIVER Not Available
Not Available Not Available Not

© 2009 Hewlett-Packard Company. All rights reserved.

Available Not Available
ROOT\LEGACY_HPCISSS\0000
Fusion ioDrive Device Driver Not Available LEGACYDRIVER Not Available Not Available Not Available Not Available
ROOT\LEGACY_FIODRIVE\0000
Crcdisk Filter Driver Not Available
LEGACYDRIVER Not Available
Not Available Not Available Not Available Not Available
ROOT\LEGACY_CRCDISK\0000
CpuSpy3 Driver Not Available
LEGACYDRIVER Not Available
Not Available Not Available Not Available Not Available
ROOT\LEGACY_CPUSPY3\0000
Common Log (CLFS) Not Available
LEGACYDRIVER Not Available
Not Available Not Available Not Available Not Available
ROOT\LEGACY_CLFS\0000
Ancilliary Function Driver for Winsock
Not Available LEGACYDRIVER
Not Available Not Available Not Available Not Available Not Available ROOT\LEGACY_AFD\0000
Microsoft iSCSI Initiator No
SCSIADAPTER 6.0.6001.18000
6/21/2006 Microsoft iscsi.inf Not Available ROOT\ISCSIPRT\0000
ACPI Fixed Feature Button No
SYSTEM 6.0.6001.18000
6/21/2006 (Standard system devices) machine.inf Not Available
ACPI\FIXEDBUTTON\2&DABA3FF&1
ServerWorks (Broadcom) HT-2100 HT-PCI-E Bridge 2&4 No SYSTEM
6.0.6001.18000 6/21/2006
ServerWorks (RCC) machine.inf Not Available
PCI\VEN_1166&DEV_0144&SUBSYS_3117103C&REV_A2\3&8970C6B&0&B8
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&3C92DC8&0&000F00
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&3C92DC8&0&000E00
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&3C92DC8&0&000D00
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&3C92DC8&0&000C00
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&3C92DC8&0&000B00
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&3C92DC8&0&000A00
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&3C92DC8&0&000900
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&3C92DC8&0&000800
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&3C92DC8&0&000700
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&3C92DC8&0&000600
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&3C92DC8&0&000500
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&3C92DC8&0&000400
HP Virtual LUN No SYSTEM
6.0.6001.18000 6/21/2006
Compaq scsidev.inf Not Available
SCSI\OTHER&VEN_COMPAQ&PROD_SCSI_COMMUNICATE\5&3C92DC8&0&000000
Smart Array P800 Controller Yes
SCSIADAPTER 6.13.0.64
8/29/2008 Hewlett-Packard Company oem5.inf Not Available
PCI\VEN_103C&DEV_3230&SUBSYS_3223103C&REV_03\4&30576D44&0&00B0
ServerWorks (Broadcom) HT-2100 HT-PCI-E Bridge 1&3 No SYSTEM
6.0.6001.18000 6/21/2006
ServerWorks (RCC) machine.inf Not Available
PCI\VEN_1166&DEV_0142&SUBSYS_3116103C&REV_A2\3&8970C6B&0&B0
ServerWorks (Broadcom) HT-2100 HT-PCI-E Bridge 2&4 No SYSTEM
6.0.6001.18000 6/21/2006
ServerWorks (RCC) machine.inf Not Available
PCI\VEN_1166&DEV_0144&SUBSYS_3115103C&REV_A2\3&8970C6B&0&A8
ServerWorks (Broadcom) HT-2100 HT-PCI-E Bridge 1&3 No SYSTEM
6.0.6001.18000 6/21/2006
ServerWorks (RCC) machine.inf Not Available
PCI\VEN_1166&DEV_0142&SUBSYS_3114103C&REV_A2\3&8970C6B&0&A0
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&41EDE1&0&001300
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&41EDE1&0&001200
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&41EDE1&0&001100
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&41EDE1&0&001000
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&41EDE1&0&000F00
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&41EDE1&0&000E00
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&41EDE1&0&000D00
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&41EDE1&0&000C00
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&41EDE1&0&000B00
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&41EDE1&0&000A00
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&41EDE1&0&000900

© 2009 Hewlett-Packard Company. All rights reserved.

Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&41EDE1&0&000800
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&41EDE1&0&000700
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&41EDE1&0&000600
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&41EDE1&0&000500
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&41EDE1&0&000400
HP Virtual LUN No SYSTEM
6.0.6001.18000 6/21/2006
Compaq scsidev.inf Not Available
SCSI\OTHER&VEN_COMPAQ&PROD_SCSI_COMMUNICATE\5&41EDE1&0&000000
Smart Array P800 Controller Yes
SCSIADAPTER 6.13.0.64
8/29/2008 Hewlett-Packard Company oem5.inf Not Available
PCI\VEN_103C&DEV_3230&SUBSYS_3223103C&REV_03\4&E9161B2&0&0098
ServerWorks (Broadcom) HT-2100 HT-PCI-E Bridge 0 No SYSTEM
6.0.6001.18000 6/21/2006
ServerWorks (RCC) machine.inf Not Available
PCI\VEN_1166&DEV_0140&SUBSYS_3113103C&REV_A2\3&8970C6B&0&98
PCI bus No SYSTEM 6.0.6001.18000
6/21/2006 (Standard system devices) machine.inf Not Available
ACPI\PNP0A03\73
ServerWorks (Broadcom) HT-2100 HT-PCI-E Bridge 2&4 No SYSTEM
6.0.6001.18000 6/21/2006
ServerWorks (RCC) machine.inf Not Available
PCI\VEN_1166&DEV_0144&SUBSYS_3117103C&REV_A2\3&317825CD&0&B8
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&19F7889D&0&000F00
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&19F7889D&0&000E00
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&19F7889D&0&000D00
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&19F7889D&0&000C00
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&19F7889D&0&000B00
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&19F7889D&0&000A00
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&19F7889D&0&000900
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&19F7889D&0&000800
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&19F7889D&0&000700
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&19F7889D&0&000600
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&19F7889D&0&000500
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&19F7889D&0&000400
HP Virtual LUN No SYSTEM
6.0.6001.18000 6/21/2006
Compaq scsidev.inf Not Available
SCSI\OTHER&VEN_COMPAQ&PROD_SCSI_COMMUNICATE\5&19F7889D&0&000000
Smart Array P800 Controller Yes
SCSIADAPTER 6.13.0.64
8/29/2008 Hewlett-Packard Company oem5.inf Not Available
PCI\VEN_103C&DEV_3230&SUBSYS_3223103C&REV_03\4&16D9858&0&00B0
ServerWorks (Broadcom) HT-2100 HT-PCI-E Bridge 1&3 No SYSTEM
6.0.6001.18000 6/21/2006
ServerWorks (RCC) machine.inf Not Available
PCI\VEN_1166&DEV_0142&SUBSYS_3116103C&REV_A2\3&317825CD&0&B0
ServerWorks (Broadcom) HT-2100 HT-PCI-E Bridge 2&4 No SYSTEM
6.0.6001.18000 6/21/2006
ServerWorks (RCC) machine.inf Not Available
PCI\VEN_1166&DEV_0144&SUBSYS_3115103C&REV_A2\3&317825CD&0&A8
ServerWorks (Broadcom) HT-2100 HT-PCI-E Bridge 1&3 No SYSTEM
6.0.6001.18000 6/21/2006
ServerWorks (RCC) machine.inf Not Available
PCI\VEN_1166&DEV_0142&SUBSYS_3114103C&REV_A2\3&317825CD&0&A0
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&32BF63B0&0&001700
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&32BF63B0&0&001600
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&32BF63B0&0&001500
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&32BF63B0&0&001400
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&32BF63B0&0&001300
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available

© 2009 Hewlett-Packard Company. All rights reserved.

SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&32BF63B0&0&001200
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&32BF63B0&0&001100
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&32BF63B0&0&001000
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&32BF63B0&0&000F00
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&32BF63B0&0&000E00
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&32BF63B0&0&000D00
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&32BF63B0&0&000C00
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&32BF63B0&0&000B00
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&32BF63B0&0&000A00
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&32BF63B0&0&000900
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&32BF63B0&0&000800
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&32BF63B0&0&000700
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&32BF63B0&0&000600
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&32BF63B0&0&000500
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&32BF63B0&0&000400
HP Virtual LUN No SYSTEM
6.0.6001.18000 6/21/2006
Compaq scsidev.inf Not Available
SCSI\OTHER&VEN_COMPAQ&PROD_SCSI_COMMUNICATE\5&32BF63B0&0&000000
Smart Array P800 Controller Yes
SCSIADAPTER 6.13.0.64
8/29/2008 Hewlett-Packard Company oem5.inf Not Available
PCI\VEN_103C&DEV_3230&SUBSYS_3223103C&REV_03\4&234B80F1&0&0098
ServerWorks (Broadcom) HT-2100 HT-PCI-E Bridge 0 No SYSTEM
6.0.6001.18000 6/21/2006
ServerWorks (RCC) machine.inf Not Available
PCI\VEN_1166&DEV_0140&SUBSYS_3113103C&REV_A2\3&317825CD&0&98
PCI bus No SYSTEM 6.0.6001.18000
6/21/2006 (Standard system devices) machine.inf Not Available
ACPI\PNP0A03\72
ServerWorks (Broadcom) HT-2100 HT-PCI-E Bridge 2&4 No SYSTEM
6.0.6001.18000 6/21/2006
ServerWorks (RCC) machine.inf Not Available
PCI\VEN_1166&DEV_0144&SUBSYS_3117103C&REV_A2\3&1EBE7528&0&B8
ServerWorks (Broadcom) HT-2100 HT-PCI-E Bridge 1&3 No SYSTEM
6.0.6001.18000 6/21/2006
ServerWorks (RCC) machine.inf Not Available
PCI\VEN_1166&DEV_0142&SUBSYS_3116103C&REV_A2\3&1EBE7528&0&B0
ServerWorks (Broadcom) HT-2100 HT-PCI-E Bridge 2&4 No SYSTEM
6.0.6001.18000 6/21/2006
ServerWorks (RCC) machine.inf Not Available
PCI\VEN_1166&DEV_0144&SUBSYS_3115103C&REV_A2\3&1EBE7528&0&A8
ServerWorks (Broadcom) HT-2100 HT-PCI-E Bridge 1&3 No SYSTEM
6.0.6001.18000 6/21/2006
ServerWorks (RCC) machine.inf Not Available
PCI\VEN_1166&DEV_0142&SUBSYS_3114103C&REV_A2\3&1EBE7528&0&A0
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&15B00B1C&0&001500
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&15B00B1C&0&001400
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&15B00B1C&0&001300
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&15B00B1C&0&001200
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&15B00B1C&0&001100
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&15B00B1C&0&001000
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&15B00B1C&0&000F00
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&15B00B1C&0&000E00
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&15B00B1C&0&000D00
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006

© 2009 Hewlett-Packard Company. All rights reserved.

(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&15B00B1C&0&000C00
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&15B00B1C&0&000B00
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&15B00B1C&0&000A00
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&15B00B1C&0&000900
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&15B00B1C&0&000800
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&15B00B1C&0&000700
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&15B00B1C&0&000600
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&15B00B1C&0&000500
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&15B00B1C&0&000400
HP Virtual LUN No SYSTEM
6.0.6001.18000 6/21/2006
Compaq scsidev.inf Not Available
SCSI\OTHER&VEN_COMPAQ&PROD_SCSI_COMMUNICATE\5&15B00B1C&0&000000
Smart Array P800 Controller Yes
SCSIADAPTER 6.13.0.64
8/29/2008 Hewlett-Packard Company oem5.inf Not Available
PCI\VEN_103C&DEV_3230&SUBSYS_3223103C&REV_03\4&31E04343&0&0098
ServerWorks (Broadcom) HT-2100 HT-PCI-E Bridge 0 No SYSTEM
6.0.6001.18000 6/21/2006
ServerWorks (RCC) machine.inf Not Available
PCI\VEN_1166&DEV_0140&SUBSYS_3113103C&REV_A2\3&1EBE7528&0&98
PCI bus No SYSTEM 6.0.6001.18000
6/21/2006 (Standard system devices) machine.inf Not Available
ACPI\PNP0A03\71
Not Available Not Available Not Available Not Available Not Available Not Available Not Available
ACPI\PNP0C0F\1
PCI standard host CPU bridge No SYSTEM 6.0.6001.18000
6/21/2006 (Standard system devices) machine.inf Not Available
PCI\VEN_1022&DEV_1204&SUBSYS_00000000&REV_00\3&C04C483&0&FC
PCI standard host CPU bridge No SYSTEM 6.0.6001.18000
6/21/2006 (Standard system devices) machine.inf Not Available
PCI\VEN_1022&DEV_1203&SUBSYS_00000000&REV_00\3&C04C483&0&FB
PCI standard host CPU bridge No SYSTEM 6.0.6001.18000
6/21/2006 (Standard system devices) machine.inf Not Available
PCI\VEN_1022&DEV_1202&SUBSYS_00000000&REV_00\3&C04C483&0&FA
PCI standard host CPU bridge No SYSTEM 6.0.6001.18000
6/21/2006 (Standard system devices) machine.inf Not Available
PCI\VEN_1022&DEV_1201&SUBSYS_00000000&REV_00\3&C04C483&0&F9
PCI standard host CPU bridge No SYSTEM 6.0.6001.18000
6/21/2006 (Standard system devices) machine.inf Not Available
PCI\VEN_1022&DEV_1200&SUBSYS_00000000&REV_00\3&C04C483&0&F8
PCI standard host CPU bridge No SYSTEM 6.0.6001.18000
6/21/2006 (Standard system devices) machine.inf Not Available
PCI\VEN_1022&DEV_1204&SUBSYS_00000000&REV_00\3&C04C483&0&F4
PCI standard host CPU bridge No SYSTEM 6.0.6001.18000
6/21/2006 (Standard system devices) machine.inf Not Available
PCI\VEN_1022&DEV_1203&SUBSYS_00000000&REV_00\3&C04C483&0&F3
PCI standard host CPU bridge No SYSTEM 6.0.6001.18000
6/21/2006 (Standard system devices) machine.inf Not Available
PCI\VEN_1022&DEV_1202&SUBSYS_00000000&REV_00\3&C04C483&0&F2
PCI standard host CPU bridge No SYSTEM 6.0.6001.18000
6/21/2006 (Standard system devices) machine.inf Not Available
PCI\VEN_1022&DEV_1201&SUBSYS_00000000&REV_00\3&C04C483&0&F1
PCI standard host CPU bridge No SYSTEM 6.0.6001.18000
6/21/2006 (Standard system devices) machine.inf Not Available
PCI\VEN_1022&DEV_1200&SUBSYS_00000000&REV_00\3&C04C483&0&F0
PCI standard host CPU bridge No SYSTEM 6.0.6001.18000
6/21/2006 (Standard system devices) machine.inf Not Available
PCI\VEN_1022&DEV_1204&SUBSYS_00000000&REV_00\3&C04C483&0&EC
PCI standard host CPU bridge No SYSTEM 6.0.6001.18000
6/21/2006 (Standard system devices) machine.inf Not Available
PCI\VEN_1022&DEV_1203&SUBSYS_00000000&REV_00\3&C04C483&0&EB
PCI standard host CPU bridge No SYSTEM 6.0.6001.18000
6/21/2006 (Standard system devices) machine.inf Not Available
PCI\VEN_1022&DEV_1202&SUBSYS_00000000&REV_00\3&C04C483&0&EA
PCI standard host CPU bridge No SYSTEM 6.0.6001.18000
6/21/2006 (Standard system devices) machine.inf Not Available
PCI\VEN_1022&DEV_1201&SUBSYS_00000000&REV_00\3&C04C483&0&E9
PCI standard host CPU bridge No SYSTEM 6.0.6001.18000
6/21/2006 (Standard system devices) machine.inf Not Available
PCI\VEN_1022&DEV_1200&SUBSYS_00000000&REV_00\3&C04C483&0&E8
PCI standard host CPU bridge No SYSTEM 6.0.6001.18000
6/21/2006 (Standard system devices) machine.inf Not Available
PCI\VEN_1022&DEV_1204&SUBSYS_00000000&REV_00\3&C04C483&0&E4
PCI standard host CPU bridge No SYSTEM 6.0.6001.18000
6/21/2006 (Standard system devices) machine.inf Not Available
PCI\VEN_1022&DEV_1203&SUBSYS_00000000&REV_00\3&C04C483&0&E3
PCI standard host CPU bridge No SYSTEM 6.0.6001.18000
6/21/2006 (Standard system devices) machine.inf Not Available
PCI\VEN_1022&DEV_1202&SUBSYS_00000000&REV_00\3&C04C483&0&E2
PCI standard host CPU bridge No SYSTEM 6.0.6001.18000
6/21/2006 (Standard system devices) machine.inf Not Available
PCI\VEN_1022&DEV_1201&SUBSYS_00000000&REV_00\3&C04C483&0&E1

© 2009 Hewlett-Packard Company. All rights reserved.

PCI standard host CPU bridge No
SYSTEM 6.0.6001.18000
6/21/2006 (Standard system
devices) machine.inf Not Available
PCI\VEN_1022&DEV_1200&SUBS
YS_00000000&REV_00\3&C04C483&0
&E0
PCI standard host CPU bridge No
SYSTEM 6.0.6001.18000
6/21/2006 (Standard system
devices) machine.inf Not Available
PCI\VEN_1022&DEV_1204&SUBS
YS_00000000&REV_00\3&C04C483&0
&DC
PCI standard host CPU bridge No
SYSTEM 6.0.6001.18000
6/21/2006 (Standard system
devices) machine.inf Not Available
PCI\VEN_1022&DEV_1203&SUBS
YS_00000000&REV_00\3&C04C483&0
&DB
PCI standard host CPU bridge No
SYSTEM 6.0.6001.18000
6/21/2006 (Standard system
devices) machine.inf Not Available
PCI\VEN_1022&DEV_1202&SUBS
YS_00000000&REV_00\3&C04C483&0
&DA
PCI standard host CPU bridge No
SYSTEM 6.0.6001.18000
6/21/2006 (Standard system
devices) machine.inf Not Available
PCI\VEN_1022&DEV_1201&SUBS
YS_00000000&REV_00\3&C04C483&0
&D9
PCI standard host CPU bridge No
SYSTEM 6.0.6001.18000
6/21/2006 (Standard system
devices) machine.inf Not Available
PCI\VEN_1022&DEV_1200&SUBS
YS_00000000&REV_00\3&C04C483&0
&D8
PCI standard host CPU bridge No
SYSTEM 6.0.6001.18000
6/21/2006 (Standard system
devices) machine.inf Not Available
PCI\VEN_1022&DEV_1204&SUBS
YS_00000000&REV_00\3&C04C483&0
&D4
PCI standard host CPU bridge No
SYSTEM 6.0.6001.18000
6/21/2006 (Standard system
devices) machine.inf Not Available
PCI\VEN_1022&DEV_1203&SUBS
YS_00000000&REV_00\3&C04C483&0
&D3
PCI standard host CPU bridge No
SYSTEM 6.0.6001.18000
6/21/2006 (Standard system
devices) machine.inf Not Available
PCI\VEN_1022&DEV_1202&SUBS
YS_00000000&REV_00\3&C04C483&0
&D2
PCI standard host CPU bridge No
SYSTEM 6.0.6001.18000
6/21/2006 (Standard system
devices) machine.inf Not Available
PCI\VEN_1022&DEV_1201&SUBS
YS_00000000&REV_00\3&C04C483&0
&D1
PCI standard host CPU bridge No
SYSTEM 6.0.6001.18000
6/21/2006 (Standard system
devices) machine.inf Not Available
PCI\VEN_1022&DEV_1200&SUBS
YS_00000000&REV_00\3&C04C483&0
&D0
PCI standard host CPU bridge No
SYSTEM 6.0.6001.18000
6/21/2006 (Standard system
devices) machine.inf Not Available
PCI\VEN_1022&DEV_1204&SUBS
YS_00000000&REV_00\3&C04C483&0
&CC
PCI standard host CPU bridge No
SYSTEM 6.0.6001.18000
6/21/2006 (Standard system
devices) machine.inf Not Available
PCI\VEN_1022&DEV_1203&SUBS
YS_00000000&REV_00\3&C04C483&0
&CB
PCI standard host CPU bridge No
SYSTEM 6.0.6001.18000
6/21/2006 (Standard system
devices) machine.inf Not Available
PCI\VEN_1022&DEV_1202&SUBS
YS_00000000&REV_00\3&C04C483&0
&CA
PCI standard host CPU bridge No
SYSTEM 6.0.6001.18000
6/21/2006 (Standard system
devices) machine.inf Not Available
PCI\VEN_1022&DEV_1201&SUBS
YS_00000000&REV_00\3&C04C483&0
&C9
PCI standard host CPU bridge No
SYSTEM 6.0.6001.18000
6/21/2006 (Standard system
devices) machine.inf Not Available
PCI\VEN_1022&DEV_1200&SUBS
YS_00000000&REV_00\3&C04C483&0
&C8
PCI standard host CPU bridge No
SYSTEM 6.0.6001.18000
6/21/2006 (Standard system
devices) machine.inf Not Available
PCI\VEN_1022&DEV_1204&SUBS
YS_00000000&REV_00\3&C04C483&0
&C4
PCI standard host CPU bridge No
SYSTEM 6.0.6001.18000
6/21/2006 (Standard system
devices) machine.inf Not Available
PCI\VEN_1022&DEV_1203&SUBS
YS_00000000&REV_00\3&C04C483&0
&C3
PCI standard host CPU bridge No
SYSTEM 6.0.6001.18000
6/21/2006 (Standard system
devices) machine.inf Not Available
PCI\VEN_1022&DEV_1202&SUBS
YS_00000000&REV_00\3&C04C483&0
&C2
PCI standard host CPU bridge No
SYSTEM 6.0.6001.18000
6/21/2006 (Standard system
devices) machine.inf Not Available
PCI\VEN_1022&DEV_1201&SUBS
YS_00000000&REV_00\3&C04C483&0
&C1
PCI standard host CPU bridge No
SYSTEM 6.0.6001.18000
6/21/2006 (Standard system
devices) machine.inf Not Available
PCI\VEN_1022&DEV_1200&SUBS
YS_00000000&REV_00\3&C04C483&0
&C0
ServerWorks (Broadcom) HT-2100 HT-
PCI-E Bridge 2&4 No SYSTEM
6.0.6001.18000 6/21/2006
ServerWorks (RCC) machine.inf Not
Available
PCI\VEN_1166&DEV_0144&SUBS
YS_3117103C&REV_A2\3&C04C483&0
&B8
ServerWorks (Broadcom) HT-2100 HT-
PCI-E Bridge 1&3 No SYSTEM
6.0.6001.18000 6/21/2006
ServerWorks (RCC) machine.inf Not
Available
PCI\VEN_1166&DEV_0142&SUBS
YS_3116103C&REV_A2\3&C04C483&0
&B0
ServerWorks (Broadcom) HT-2100 HT-
PCI-E Bridge 2&4 No SYSTEM
6.0.6001.18000 6/21/2006
ServerWorks (RCC) machine.inf Not
Available
PCI\VEN_1166&DEV_0144&SUBS
YS_3115103C&REV_A2\3&C04C483&0
&A8
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not
Available
SCSI\DISK&VEN_HP&PROD_LOGI
CAL_VOLUME\5&33FE751F&0&00060
0
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not
Available
SCSI\DISK&VEN_HP&PROD_LOGI
CAL_VOLUME\5&33FE751F&0&00050
0
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not
Available
SCSI\DISK&VEN_HP&PROD_LOGI
CAL_VOLUME\5&33FE751F&0&00040
0
HP Virtual LUN No SYSTEM
6.0.6001.18000 6/21/2006
Compaq scsidev.inf Not
Available
SCSI\OTHER&VEN_COMPAQ&PR
OD_SCSI_COMMUNICATE\5&33FE75
1F&0&000000
Smart Array P400 Controller Yes
SCSIADAPTER 6.13.0.64
8/29/2008 Hewlett-Packard
Company oem5.inf Not Available
PCI\VEN_103C&DEV_3230&SUBS
YS_3234103C&REV_03\4&322FE8E2&
0&00A0
ServerWorks (Broadcom) HT-2100 HT-
PCI-E Bridge 1&3 No SYSTEM
6.0.6001.18000 6/21/2006
ServerWorks (RCC) machine.inf Not
Available
PCI\VEN_1166&DEV_0142&SUBS
YS_3114103C&REV_A2\3&C04C483&0
&A0
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not
Available
SCSI\DISK&VEN_HP&PROD_LOGI
CAL_VOLUME\5&22B433C4&0&00040
0
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not
Available

© 2009 Hewlett-Packard Company. All rights reserved.

SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&22B433C4&0&001500
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&22B433C4&0&001400
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&22B433C4&0&001300
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&22B433C4&0&001200
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&22B433C4&0&001100
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&22B433C4&0&001000
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&22B433C4&0&000F00
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&22B433C4&0&000E00
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&22B433C4&0&000D00
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&22B433C4&0&000C00
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&22B433C4&0&000B00
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&22B433C4&0&000A00
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&22B433C4&0&000900
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&22B433C4&0&000800
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&22B433C4&0&000700
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&22B433C4&0&000600
Disk drive No DISKDRIVE
6.0.6001.18000 6/21/2006
(Standard disk drives) disk.inf Not Available
SCSI\DISK&VEN_HP&PROD_LOGICAL_VOLUME\5&22B433C4&0&000500
HP Virtual LUN No SYSTEM
6.0.6001.18000 6/21/2006
Compaq scsidev.inf Not Available
SCSI\OTHER&VEN_COMPAQ&PROD_SCSI_COMMUNICATE\5&22B433C4&0&000000
Smart Array P800 Controller Yes
SCSIADAPTER 6.13.0.64
8/29/2008 Hewlett-Packard Company oem5.inf Not Available
PCI\VEN_103C&DEV_3230&SUBSYS_3223103C&REV_03\4&359FAD9&0&0098
ServerWorks (Broadcom) HT-2100 HT-PCI-E Bridge 0 No SYSTEM
6.0.6001.18000 6/21/2006
ServerWorks (RCC) machine.inf Not Available
PCI\VEN_1166&DEV_0140&SUBSYS_3113103C&REV_A2\3&C04C483&0&98
USB Root Hub No USB
6.0.6001.18000 6/21/2006
(Standard USB Host Controller) usbport.inf Not Available
USB\ROOT_HUB20\4&2B0CDB6&0
Standard Enhanced PCI to USB Host Controller No USB 6.0.6001.18000
6/21/2006 (Standard USB Host Controller) usbport.inf Not Available
PCI\VEN_1166&DEV_0223&SUBSYS_320D103C&REV_01\3&C04C483&0&3A
Generic USB Hub No USB
6.0.6001.18000 6/21/2006
(Generic USB Hub) usb.inf Not Available
USB\VID_04B4&PID_6560\5&3E7D475&0&1
USB Root Hub No USB
6.0.6001.18000 6/21/2006
(Standard USB Host Controller) usbport.inf Not Available
USB\ROOT_HUB\4&30992D62&0
Standard OpenHCD USB Host Controller No USB 6.0.6001.18000
6/21/2006 (Standard USB Host Controller) usbport.inf Not Available
PCI\VEN_1166&DEV_0223&SUBSYS_320C103C&REV_01\3&C04C483&0&39
USB Root Hub No USB
6.0.6001.18000 6/21/2006
(Standard USB Host Controller) usbport.inf Not Available
USB\ROOT_HUB\4&7B81400&0
Standard OpenHCD USB Host Controller No USB 6.0.6001.18000
6/21/2006 (Standard USB Host Controller) usbport.inf Not Available
PCI\VEN_1166&DEV_0223&SUBSYS_320C103C&REV_01\3&C04C483&0&38
Communications Port No PORTS
6.0.6001.18000 6/21/2006
(Standard port types) msports.inf Not Available ACPI\PNP0501\0
Extended IO Bus No SYSTEM
6.0.6001.18000 6/21/2006
(Standard system devices) machine.inf Not Available
ACPI\PNP0A06\4&63ACF31&0
PS/2 Compatible Mouse No MOUSE
6.0.6001.18000 6/21/2006
Microsoft msmouse.inf Not Available
ACPI\PNP0F13\4&63ACF31&0
Standard PS/2 Keyboard No KEYBOARD 6.0.6001.18000
6/21/2006 (Standard keyboards) keyboard.inf Not Available
ACPI\PNP0303\4&63ACF31&0
Programmable interrupt controller No SYSTEM 6.0.6001.18000
6/21/2006 (Standard system devices) machine.inf Not Available
ACPI\PNP0000\4&63ACF31&0
System speaker No SYSTEM
6.0.6001.18000 6/21/2006
(Standard system devices) machine.inf Not Available
ACPI\PNP0800\4&63ACF31&0
Direct memory access controller No SYSTEM 6.0.6001.18000
6/21/2006 (Standard system devices) machine.inf Not Available
ACPI\PNP0200\4&63ACF31&0
High precision event timer No SYSTEM 6.0.6001.18000
6/21/2006 (Standard system devices) machine.inf Not Available
ACPI\PNP0103\0
System timer No SYSTEM
6.0.6001.18000 6/21/2006
(Standard system devices) machine.inf Not Available
ACPI\PNP0100\4&63ACF31&0

© 2009 Hewlett-Packard Company. All rights reserved.

Microsoft Generic IPMI Compliant Device No SYSTEM 6.0.6001.18000 6/21/2006 Microsoft ipmidrv.inf Not Available ACPI\IPI0001\0
Motherboard resources No SYSTEM 6.0.6001.18000 6/21/2006 (Standard system devices) machine.inf Not Available ACPI\PNP0C02\78
ServerWorks (Broadcom) HT-1000 HT-PCI-X to ISA Bridge No SYSTEM 6.0.6001.18000 6/21/2006 ServerWorks (RCC) machine.inf Not Available PCI\VEN_1166&DEV_0234&SUBSYS_02301166&REV_00\3&C04C483&0&32
IDE Channel No HDC 6.0.6001.18000 6/21/2006 (Standard IDE ATA/ATAPI controllers) mshdc.inf Not Available PCIIDE\IDECHANNEL\4&148384DD&0&1
CD-ROM Drive No CDROM 6.0.6001.18000 6/21/2006 (Standard CD-ROM drives) cdrom.inf Not Available IDE\CDROMTSSTCORP_DVD-ROM_TS-L332A_____HG00____\5&2274845F&0&0.0.0
IDE Channel No HDC 6.0.6001.18000 6/21/2006 (Standard IDE ATA/ATAPI controllers) mshdc.inf Not Available PCIIDE\IDECHANNEL\4&148384DD&0&0
Standard Dual Channel PCI IDE Controller No HDC 6.0.6001.18000 6/21/2006 (Standard IDE ATA/ATAPI controllers) mshdc.inf Not Available PCI\VEN_1166&DEV_0214&SUBSYS_320B103C&REV_00\3&C04C483&0&31
ServerWorks (Broadcom) HT-1000 HT-PCI-X Bridge 1 No SYSTEM 6.0.6001.18000 6/21/2006 ServerWorks (RCC) machine.inf Not Available PCI\VEN_1166&DEV_0205&SUBSYS_02011166&REV_00\3&C04C483&0&30
Broadcom BCM5706C NetXtreme II GigE (NDIS VBD Client) No NET 3.7.19.0 6/21/2006 Broadcom Corporation netbxnda.inf Not Available B06BDRV\L2ND&PCI_164A14E4&SUBSYS_1709103C&REV_02\6&23BF5E65&0&20050204
Broadcom BCM5706C NetXtreme II GigE No SYSTEM 3.7.20.0 6/21/2006 Broadcom Corporation netbvbda.inf Not Available PCI\VEN_14E4&DEV_164A&SUBSYS_1709103C&REV_02\5&3609893E&0&206828
Broadcom BCM5706C NetXtreme II GigE (NDIS VBD Client) No NET 3.7.19.0 6/21/2006 Broadcom Corporation netbxnda.inf Not Available B06BDRV\L2ND&PCI_164A14E4&SUBSYS_1709103C&REV_02\6&2FFF06&0&20050203
Broadcom BCM5706C NetXtreme II GigE No SYSTEM 3.7.20.0 6/21/2006 Broadcom Corporation netbvbda.inf Not Available PCI\VEN_14E4&DEV_164A&SUBSYS_1709103C&REV_02\5&3609893E&0&186828
ServerWorks (Broadcom) HT-1000 HT-PCI-X Bridge No SYSTEM 6.0.6001.18000 6/21/2006 ServerWorks (RCC) machine.inf Not Available PCI\VEN_1166&DEV_0104&SUBSYS_3111103C&REV_C0\4&C527D2B&0&6828
ServerWorks (Broadcom) HT-1000 HT-PCI-X Bridge 0 No SYSTEM 6.0.6001.18000 6/21/2006 ServerWorks (RCC) machine.inf Not Available PCI\VEN_1166&DEV_0036&SUBSYS_3110103C&REV_00\3&C04C483&0&28
HP ProLiant iLO 2 Management Controller Driver Yes SYSTEM 1.7.0.0 1/11/2008 Hewlett-Packard Company oem2.inf Not Available PCI\VEN_103C&DEV_3302&SUBSYS_3305103C&REV_00\3&C04C483&0&26
HID-compliant mouse No MOUSE 6.0.6001.18000 6/21/2006 Microsoft msmouse.inf Not Available HID\VID_03F0&PID_1027&MI_01\7&2B47AC2D&0&0000
USB Human Interface Device No HIDCLASS 6.0.6001.18000 6/21/2006 (Standard system devices) input.inf Not Available USB\VID_03F0&PID_1027&MI_01\6&174A8EC&0&0001
HID Keyboard Device No KEYBOARD 6.0.6001.18000 6/21/2006 (Standard keyboards) keyboard.inf Not Available HID\VID_03F0&PID_1027&MI_00\7&2DCFE64&0&0000
USB Human Interface Device No HIDCLASS 6.0.6001.18000 6/21/2006 (Standard system devices) input.inf Not Available USB\VID_03F0&PID_1027&MI_00\6&174A8EC&0&0000
USB Composite Device No USB 6.0.6001.18000 6/21/2006 (Standard USB Host Controller) usb.inf Not Available USB\VID_03F0&PID_1027\5&1845D16A&0&1
Generic USB Hub No USB 6.0.6001.18000 6/21/2006 (Generic USB Hub) usb.inf Not Available USB\VID_03F0&PID_1327\5&1845D16A&0&2
USB Root Hub No USB 6.0.6001.18000 6/21/2006 (Standard USB Host Controller) usbport.inf Not Available USB\ROOT_HUB\4&1AAB332F&0
Standard Universal PCI to USB Host Controller No USB 6.0.6001.18000 6/21/2006 (Standard USB Host Controller) usbport.inf Not Available PCI\VEN_103C&DEV_3300&SUBSYS_3305103C&REV_00\3&C04C483&0&24
HP iLO Management Channel Interface Driver Yes MULTIFUNCTION 1.13.0.0 1/1/2008 Hewlett-Packard Company oem3.inf Not Available PCI\VEN_0E11&DEV_B204&SUBSYS_3305103C&REV_03\3&C04C483&0&22
HP ProLiant iLO 2 Legacy Support Function Yes SYSTEM 1.7.0.0 1/11/2008 Hewlett-Packard Company oem2.inf Not Available PCI\VEN_0E11&DEV_B203&SUBSYS_3305103C&REV_03\3&C04C483&0&20
Generic PnP Monitor No MONITOR 6.0.6001.18000 6/21/2006 (Standard monitor types) monitor.inf Not Available DISPLAY\AVO0402\4&37029B4A&0&10000080&00&03
Generic Non-PnP Monitor No MONITOR 6.0.6001.18000 6/21/2006 (Standard monitor types) monitor.inf Not Available DISPLAY\DEFAULT_MONITOR\4&37029B4A&0&10000001&00&03
ATI ES1000 Yes DISPLAY 8.240.50.2000 10/25/2007 ATI Technologies Inc. oem1.inf Not Available PCI\VEN_1002&DEV_515E&SUBSYS_31FB103C&REV_02\3&C04C483&0&18
PCI bus No SYSTEM 6.0.6001.18000 6/21/2006 (Standard system devices) machine.inf Not Available ACPI\PNP0A03\70
Processor No PROCESSOR 6.0.6001.18000 6/21/2006 (Standard processor types) cpu.inf Not Available ACPI\AUTHENTICAMD_-_AMD64_FAMILY_16_MODEL_8\47
Processor No PROCESSOR 6.0.6001.18000 6/21/2006 (Standard processor types) cpu.inf Not Available ACPI\AUTHENTICAMD_-_AMD64_FAMILY_16_MODEL_8\46
Processor No PROCESSOR 6.0.6001.18000 6/21/2006 (Standard processor types) cpu.inf Not Available ACPI\AUTHENTICAMD_-_AMD64_FAMILY_16_MODEL_8\45
Processor No PROCESSOR 6.0.6001.18000 6/21/2006 (Standard processor types) cpu.inf Not Available ACPI\AUTHENTICAMD_-_AMD64_FAMILY_16_MODEL_8\44
Processor No PROCESSOR 6.0.6001.18000 6/21/2006 (Standard processor types) cpu.inf Not Available ACPI\AUTHENTICAMD_-_AMD64_FAMILY_16_MODEL_8\43

© 2009 Hewlett-Packard Company. All rights reserved.

Processor No PROCESSOR
6.0.6001.18000 6/21/2006
(Standard processor types) cpu.inf
Not Available
ACPI\AUTHENTICAMD_-
_AMD64_FAMILY_16_MODEL_8\42
Processor No PROCESSOR
6.0.6001.18000 6/21/2006
(Standard processor types) cpu.inf
Not Available
ACPI\AUTHENTICAMD_-
_AMD64_FAMILY_16_MODEL_8\41
Processor No PROCESSOR
6.0.6001.18000 6/21/2006
(Standard processor types) cpu.inf
Not Available
ACPI\AUTHENTICAMD_-
_AMD64_FAMILY_16_MODEL_8\40
Processor No PROCESSOR
6.0.6001.18000 6/21/2006
(Standard processor types) cpu.inf
Not Available
ACPI\AUTHENTICAMD_-
_AMD64_FAMILY_16_MODEL_8\39
Processor No PROCESSOR
6.0.6001.18000 6/21/2006
(Standard processor types) cpu.inf
Not Available
ACPI\AUTHENTICAMD_-
_AMD64_FAMILY_16_MODEL_8\38
Processor No PROCESSOR
6.0.6001.18000 6/21/2006
(Standard processor types) cpu.inf
Not Available
ACPI\AUTHENTICAMD_-
_AMD64_FAMILY_16_MODEL_8\37
Processor No PROCESSOR
6.0.6001.18000 6/21/2006
(Standard processor types) cpu.inf
Not Available
ACPI\AUTHENTICAMD_-
_AMD64_FAMILY_16_MODEL_8\36
Processor No PROCESSOR
6.0.6001.18000 6/21/2006
(Standard processor types) cpu.inf
Not Available
ACPI\AUTHENTICAMD_-
_AMD64_FAMILY_16_MODEL_8\35
Processor No PROCESSOR
6.0.6001.18000 6/21/2006
(Standard processor types) cpu.inf
Not Available
ACPI\AUTHENTICAMD_-
_AMD64_FAMILY_16_MODEL_8\34
Processor No PROCESSOR
6.0.6001.18000 6/21/2006
(Standard processor types) cpu.inf
Not Available
ACPI\AUTHENTICAMD_-
_AMD64_FAMILY_16_MODEL_8\33
Processor No PROCESSOR
6.0.6001.18000 6/21/2006
(Standard processor types) cpu.inf
Not Available
ACPI\AUTHENTICAMD_-
_AMD64_FAMILY_16_MODEL_8\32
Processor No PROCESSOR
6.0.6001.18000 6/21/2006
(Standard processor types) cpu.inf
Not Available
ACPI\AUTHENTICAMD_-
_AMD64_FAMILY_16_MODEL_8\31
Processor No PROCESSOR
6.0.6001.18000 6/21/2006
(Standard processor types) cpu.inf
Not Available
ACPI\AUTHENTICAMD_-
_AMD64_FAMILY_16_MODEL_8\30
Processor No PROCESSOR
6.0.6001.18000 6/21/2006
(Standard processor types) cpu.inf
Not Available
ACPI\AUTHENTICAMD_-
_AMD64_FAMILY_16_MODEL_8\29
Processor No PROCESSOR
6.0.6001.18000 6/21/2006
(Standard processor types) cpu.inf
Not Available
ACPI\AUTHENTICAMD_-
_AMD64_FAMILY_16_MODEL_8\28
Processor No PROCESSOR
6.0.6001.18000 6/21/2006
(Standard processor types) cpu.inf
Not Available
ACPI\AUTHENTICAMD_-
_AMD64_FAMILY_16_MODEL_8\27
Processor No PROCESSOR
6.0.6001.18000 6/21/2006
(Standard processor types) cpu.inf
Not Available
ACPI\AUTHENTICAMD_-
_AMD64_FAMILY_16_MODEL_8\26
Processor No PROCESSOR
6.0.6001.18000 6/21/2006
(Standard processor types) cpu.inf
Not Available
ACPI\AUTHENTICAMD_-
_AMD64_FAMILY_16_MODEL_8\25
Processor No PROCESSOR
6.0.6001.18000 6/21/2006
(Standard processor types) cpu.inf
Not Available
ACPI\AUTHENTICAMD_-
_AMD64_FAMILY_16_MODEL_8\24
Processor No PROCESSOR
6.0.6001.18000 6/21/2006
(Standard processor types) cpu.inf
Not Available
ACPI\AUTHENTICAMD_-
_AMD64_FAMILY_16_MODEL_8\23
Processor No PROCESSOR
6.0.6001.18000 6/21/2006
(Standard processor types) cpu.inf
Not Available
ACPI\AUTHENTICAMD_-
_AMD64_FAMILY_16_MODEL_8\22
Processor No PROCESSOR
6.0.6001.18000 6/21/2006
(Standard processor types) cpu.inf
Not Available
ACPI\AUTHENTICAMD_-
_AMD64_FAMILY_16_MODEL_8\21
Processor No PROCESSOR
6.0.6001.18000 6/21/2006
(Standard processor types) cpu.inf
Not Available
ACPI\AUTHENTICAMD_-
_AMD64_FAMILY_16_MODEL_8\20
Processor No PROCESSOR
6.0.6001.18000 6/21/2006
(Standard processor types) cpu.inf
Not Available
ACPI\AUTHENTICAMD_-
_AMD64_FAMILY_16_MODEL_8\19
Processor No PROCESSOR
6.0.6001.18000 6/21/2006
(Standard processor types) cpu.inf
Not Available
ACPI\AUTHENTICAMD_-
_AMD64_FAMILY_16_MODEL_8\18
Processor No PROCESSOR
6.0.6001.18000 6/21/2006
(Standard processor types) cpu.inf
Not Available
ACPI\AUTHENTICAMD_-
_AMD64_FAMILY_16_MODEL_8\17
Processor No PROCESSOR
6.0.6001.18000 6/21/2006
(Standard processor types) cpu.inf
Not Available
ACPI\AUTHENTICAMD_-
_AMD64_FAMILY_16_MODEL_8\16
Processor No PROCESSOR
6.0.6001.18000 6/21/2006
(Standard processor types) cpu.inf
Not Available
ACPI\AUTHENTICAMD_-
_AMD64_FAMILY_16_MODEL_8\15
Processor No PROCESSOR
6.0.6001.18000 6/21/2006
(Standard processor types) cpu.inf
Not Available
ACPI\AUTHENTICAMD_-
_AMD64_FAMILY_16_MODEL_8\14
Processor No PROCESSOR
6.0.6001.18000 6/21/2006
(Standard processor types) cpu.inf
Not Available
ACPI\AUTHENTICAMD_-
_AMD64_FAMILY_16_MODEL_8\13
Processor No PROCESSOR
6.0.6001.18000 6/21/2006
(Standard processor types) cpu.inf
Not Available
ACPI\AUTHENTICAMD_-
_AMD64_FAMILY_16_MODEL_8\12
Processor No PROCESSOR
6.0.6001.18000 6/21/2006
(Standard processor types) cpu.inf
Not Available
ACPI\AUTHENTICAMD_-
_AMD64_FAMILY_16_MODEL_8\11
Processor No PROCESSOR
6.0.6001.18000 6/21/2006
(Standard processor types) cpu.inf
Not Available
ACPI\AUTHENTICAMD_-
_AMD64_FAMILY_16_MODEL_8\10
Processor No PROCESSOR
6.0.6001.18000 6/21/2006
(Standard processor types) cpu.inf
Not Available
ACPI\AUTHENTICAMD_-
_AMD64_FAMILY_16_MODEL_8_9
Processor No PROCESSOR
6.0.6001.18000 6/21/2006
(Standard processor types) cpu.inf
Not Available
ACPI\AUTHENTICAMD_-
_AMD64_FAMILY_16_MODEL_8_8
Processor No PROCESSOR
6.0.6001.18000 6/21/2006
(Standard processor types) cpu.inf
Not Available
ACPI\AUTHENTICAMD_-
_AMD64_FAMILY_16_MODEL_8_7
Processor No PROCESSOR
6.0.6001.18000 6/21/2006
(Standard processor types) cpu.inf
Not Available
ACPI\AUTHENTICAMD_-
_AMD64_FAMILY_16_MODEL_8_6
Processor No PROCESSOR
6.0.6001.18000 6/21/2006
(Standard processor types) cpu.inf

© 2009 Hewlett-Packard Company. All rights reserved.

Not Available
ACPI\AUTHENTICAMD_-
_AMD64_FAMILY_16_MODEL_8_5
Processor No PROCESSOR
6.0.6001.18000 6/21/2006
(Standard processor types) cpu.inf
Not Available
ACPI\AUTHENTICAMD_-
_AMD64_FAMILY_16_MODEL_8_4
Processor No PROCESSOR
6.0.6001.18000 6/21/2006
(Standard processor types) cpu.inf
Not Available
ACPI\AUTHENTICAMD_-
_AMD64_FAMILY_16_MODEL_8_3
Processor No PROCESSOR
6.0.6001.18000 6/21/2006
(Standard processor types) cpu.inf
Not Available
ACPI\AUTHENTICAMD_-
_AMD64_FAMILY_16_MODEL_8_2
Processor No PROCESSOR
6.0.6001.18000 6/21/2006
(Standard processor types) cpu.inf
Not Available
ACPI\AUTHENTICAMD_-
_AMD64_FAMILY_16_MODEL_8_1
Processor No PROCESSOR
6.0.6001.18000 6/21/2006
(Standard processor types) cpu.inf
Not Available
ACPI\AUTHENTICAMD_-
_AMD64_FAMILY_16_MODEL_8_0
Microsoft ACPI-Compliant System No
SYSTEM 6.0.6001.18000
6/21/2006 Microsoft acpi.inf Not
Available ACPI_HAL\PNP0C08\0
ACPI x64-based PC No COMPUTER
6.0.6001.18000 6/21/2006
(Standard computers) hal.inf Not
Available ROOT\ACPI_HAL\0000
Microsoft ISATAP Adapter No NET
6.0.6001.18000 6/21/2006
Microsoft nettun.inf Not
Available ROOT*ISATAP\0002
Microsoft ISATAP Adapter No NET
6.0.6001.18000 6/21/2006
Microsoft nettun.inf Not
Available ROOT*ISATAP\0000
Microsoft 6to4 Adapter No NET
6.0.6001.18000 6/21/2006
Microsoft nettun.inf Not
Available ROOT*6TO4MP\0000
Not Available Not Available Not
Available Not Available Not
Available Not Available Not
Available Not Available
HTREE\ROOT\0

[Environment Variables]

Variable Value User Name
SQL <SYSTEM>
#This controls <SYSTEM>
ComSpec
%SystemRoot%\system32\cmd.exe
<SYSTEM>
FP_NO_HOST_CHECK NO <SYSTEM>
NUMBER_OF_PROCESSORS 48
<SYSTEM>
OS Windows_NT <SYSTEM>
Path E:\Program Files
(x86)\AMD\syscat;E:\Program
Files\AMD\MemBlock;E:\Program
Files\AMD\mreport;E:\Program
Files\AMD\multevent;E:\Program
Files\AMD\CpuSpy;%SystemRoot%\sys
tem32;%SystemRoot%;%SystemRoot%
\System32\Wbem;%SYSTEMROOT%\S
ystem32\WindowsPowerShell\v1.0\;%Sy
stemRoot%\System32\Windows System
Resource Manager\bin;E:\Program Files
(x86)\Microsoft SQL
Server\100\Tools\Binn\;E:\Program
Files\Microsoft SQL
Server\100\Tools\Binn\;E:\Program
Files\Microsoft SQL
Server\100\DTS\Binn\;E:\Program Files
(x86)\Microsoft SQL
Server\100\Tools\Binn\VSShell\Commo
n7\IDE\;E:\Program Files (x86)\Microsoft
SQL Server\100\DTS\Binn\;E:\Program
Files (x86)\Microsoft Visual Studio
9.0\Common7\IDE\PrivateAssemblies\
<SYSTEM>
PATHEXT
.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.
JS;.JSE;.WSF;.WSH;.MSC <SYSTEM>
PROCESSOR_ARCHITECTURE
AMD64 <SYSTEM>
PROCESSOR_IDENTIFIER AMD64
Family 16 Model 8 Stepping 0,
AuthenticAMD <SYSTEM>
PROCESSOR_LEVEL 16 <SYSTEM>
PROCESSOR_REVISION 0800
<SYSTEM>
TEMP %SystemRoot%\TEMP
<SYSTEM>
TMP %SystemRoot%\TEMP
<SYSTEM>
USERNAME SYSTEM <SYSTEM>
windir %SystemRoot% <SYSTEM>
TEMP
%USERPROFILE%\AppData\Local\
Temp NT AUTHORITY\SYSTEM
TMP
%USERPROFILE%\AppData\Local\
Temp NT AUTHORITY\SYSTEM
TEMP
%USERPROFILE%\AppData\Local\
Temp NT AUTHORITY\LOCAL
SERVICE
TMP
%USERPROFILE%\AppData\Local\
Temp NT AUTHORITY\LOCAL
SERVICE
TEMP
%USERPROFILE%\AppData\Local\
Temp NT AUTHORITY\NETWORK
SERVICE
TMP
%USERPROFILE%\AppData\Local\
Temp NT AUTHORITY\NETWORK
SERVICE
SQL OCTANE1\Administrator
#This controls
OCTANE1\Administrator
TEMP
%USERPROFILE%\AppData\Local\
Temp OCTANE1\Administrator
TMP
%USERPROFILE%\AppData\Local\
Temp OCTANE1\Administrator

[Print Jobs]

Can't Collect Information

[Network Connections]

Local Name Remote Name Type
Status User Name

[Running Tasks]

Name Path Process ID Priority Min
Working Set Max Working Set
Start Time Version Size File
Date
system idle process Not Available 0
0 Not Available Not Available
Not Available Not Available Not
Available Not Available
system Not Available 4 8 Not
Available Not Available 7/13/2009
2:58 PM Not Available Not
Available Not Available
smss.exe Not Available 680 11 200
1380 7/13/2009 2:58 PM Not
Available Not Available Not
Available
csrss.exe
e:\windows\system32\csrss.exe 756
13 200 1380 7/13/2009 2:58 PM
6.0.6001.18000 7.50 KB (7,680
bytes) 1/18/2008 9:59 PM
wininit.exe
e:\windows\system32\wininit.exe
812 13 200 1380 7/13/2009 2:58
PM 6.0.6001.18000 121.00 KB (123,904
bytes) 1/18/2008 10:17 PM
services.exe
e:\windows\system32\services.exe
848 9 200 1380 7/13/2009 2:58
PM 6.0.6001.18000 375.50 KB (384,512
bytes) 1/18/2008 10:03 PM
lsass.exe
e:\windows\system32\lsass.exe 860
9 200 1380 7/13/2009 2:58 PM
6.0.6001.18000 11.00 KB (11,264
bytes) 1/18/2008 10:16 PM
lsm.exe e:\windows\system32\lsm.exe
868 8 200 1380 7/13/2009 2:58
PM 6.0.6001.18000 258.50 KB (264,704
bytes) 1/18/2008 10:43 PM
svchost.exe
e:\windows\system32\svchost.exe
468 8 200 1380 7/13/2009 2:58
PM 6.0.6001.18000 27.00 KB (27,648
bytes) 1/18/2008 10:02 PM
svchost.exe
e:\windows\system32\svchost.exe
604 8 200 1380 7/13/2009 2:58
PM 6.0.6001.18000 27.00 KB (27,648
bytes) 1/18/2008 10:02 PM
svchost.exe
e:\windows\system32\svchost.exe
764 8 200 1380 7/13/2009 2:58
PM 6.0.6001.18000 27.00 KB (27,648
bytes) 1/18/2008 10:02 PM
svchost.exe
e:\windows\system32\svchost.exe
840 8 200 1380 7/13/2009 2:58
PM 6.0.6001.18000 27.00 KB (27,648
bytes) 1/18/2008 10:02 PM
svchost.exe
e:\windows\system32\svchost.exe
864 8 200 1380 7/13/2009 2:58
PM 6.0.6001.18000 27.00 KB (27,648
bytes) 1/18/2008 10:02 PM
slsvc.exe
e:\windows\system32\slsvc.exe 592
8 200 1380 7/13/2009 2:58 PM

© 2009 Hewlett-Packard Company. All rights reserved.

6.0.6001.18000 2.06 MB (2,161,664 bytes) 1/18/2008 11:33 PM
svchost.exe
e:\windows\system32\svchost.exe
1052 8 200 1380 7/13/2009 2:58 PM 6.0.6001.18000 27.00 KB (27,648 bytes) 1/18/2008 10:02 PM
svchost.exe
e:\windows\system32\svchost.exe
1140 8 200 1380 7/13/2009 2:58 PM 6.0.6001.18000 27.00 KB (27,648 bytes) 1/18/2008 10:02 PM
svchost.exe
e:\windows\system32\svchost.exe
1220 8 200 1380 7/13/2009 2:59 PM 6.0.6001.18000 27.00 KB (27,648 bytes) 1/18/2008 10:02 PM
cissesrv.exe e:\program files\hp\cissesrv\cissesrv.exe 1484
8 200 1380 7/13/2009 2:59 PM
6.8.0.64 147.00 KB (150,528 bytes) 11/6/2007 1:01 PM
svchost.exe
e:\windows\system32\svchost.exe
1504 8 200 1380 7/13/2009 2:59 PM 6.0.6001.18000 27.00 KB (27,648 bytes) 1/18/2008 10:02 PM
svchost.exe
e:\windows\system32\svchost.exe
1524 8 200 1380 7/13/2009 2:59 PM 6.0.6001.18000 27.00 KB (27,648 bytes) 1/18/2008 10:02 PM
sqlwriter.exe e:\program files\microsoft sql server\90\shared\sqlwriter.exe 1624
8 200 1380 7/13/2009 2:59 PM
2007.100.1600.22 154.02 KB (157,720 bytes) 7/10/2008 5:31 AM
taskeng.exe
e:\windows\system32\taskeng.exe
2620 6 200 1380 7/13/2009 3:00 PM 6.0.6001.18000 259.00 KB (265,216 bytes) 1/18/2008 10:13 PM
csrss.exe
e:\windows\system32\csrss.exe
2676 13 200 1380 7/13/2009 3:00 PM 6.0.6001.18000 7.50 KB (7,680 bytes) 1/18/2008 9:59 PM
winlogon.exe
e:\windows\system32\winlogon.exe
2700 13 200 1380 7/13/2009 3:00 PM 6.0.6001.18000 396.50 KB (406,016 bytes) 1/18/2008 10:18 PM
taskeng.exe
e:\windows\system32\taskeng.exe
2912 8 200 1380 7/13/2009 3:00 PM 6.0.6001.18000 259.00 KB (265,216 bytes) 1/18/2008 10:13 PM
dwm.exe
e:\windows\system32\dwm.exe
3040 8 200 1380 7/13/2009 3:00 PM 6.0.6001.18000 96.50 KB (98,816 bytes) 1/18/2008 10:10 PM
explorer.exe e:\windows\explorer.exe
564 8 200 1380 7/13/2009 3:00 PM 6.0.6001.18000 2.94 MB (3,080,704 bytes) 1/18/2008 10:22 PM
sqlservr.exe e:\program files\microsoft sql server\mssql10.mssqlserver\mssql\binn\sqlservr.exe 2584 8 200 1380
7/13/2009 3:01 PM
2007.100.2531.0 54.95 MB (57,617,752 bytes) 3/30/2009 4:02 AM
msdtc.exe
e:\windows\system32\msdtc.exe
2864 8 200 1380 7/13/2009 3:01 PM 2001.12.6931.18000
104.00 KB (106,496 bytes)
1/18/2008 10:27 PM
svchost.exe
e:\windows\system32\svchost.exe
2752 8 200 1380 7/13/2009 3:46 PM 6.0.6001.18000 27.00 KB (27,648 bytes) 1/18/2008 10:02 PM
csrss.exe
e:\windows\system32\csrss.exe
4444 13 200 1380 7/13/2009 4:34 PM 6.0.6001.18000 7.50 KB (7,680 bytes) 1/18/2008 9:59 PM
rdpclip.exe
e:\windows\system32\rdpclip.exe
4288 8 200 1380 7/13/2009 4:34 PM 6.0.6001.18000 187.00 KB (191,488 bytes) 1/19/2008 5:51 AM
winlogon.exe
e:\windows\system32\winlogon.exe
4300 13 200 1380 7/13/2009 4:34 PM 6.0.6001.18000 396.50 KB (406,016 bytes) 1/18/2008 10:18 PM
logonui.exe
e:\windows\system32\logonui.exe
1180 13 200 1380 7/13/2009 4:34 PM 6.0.6001.18000 20.50 KB (20,992 bytes) 1/18/2008 10:17 PM
msinfo32.exe
e:\windows\system32\msinfo32.exe
4056 8 200 1380 7/14/2009 11:18 AM 6.0.6001.18000 477.50 KB (488,960 bytes) 1/18/2008 10:03 PM
wmiprvse.exe
e:\windows\system32\wbem\wmiprvse.exe 4828 8 200 1380
7/14/2009 11:18 AM
6.0.6001.18000 340.50 KB (348,672 bytes) 1/18/2008 10:13 PM
wmiprvse.exe
e:\windows\system32\wbem\wmiprvse.exe 5056 8 200 1380
7/14/2009 11:18 AM
6.0.6001.18000 340.50 KB (348,672 bytes) 1/18/2008 10:13 PM

[Loaded Modules]

Name Version Size File Date
Manufacturer Path
csrss 6.0.6001.18000 7.50 KB (7,680 bytes) 1/18/2008 9:59 PM Microsoft Corporation
e:\windows\system32\csrss.exe
ntdll 6.0.6001.18000 1.49 MB (1,559,696 bytes) 1/19/2008 1:50 AM
Microsoft Corporation
e:\windows\system32\ntdll.dll
csrsrv 6.0.6001.18000 83.50 KB (85,504 bytes) 1/18/2008 9:59 PM
Microsoft Corporation
e:\windows\system32\csrsrv.dll
basesrv 6.0.6001.18000 78.50 KB (80,384 bytes) 1/18/2008 9:59 PM
Microsoft Corporation
e:\windows\system32\basesrv.dll
winsrv 6.0.6001.18000 439.50 KB (450,048 bytes) 1/18/2008 10:08 PM
Microsoft Corporation
e:\windows\system32\winsrv.dll
user32 6.0.6001.18000 801.00 KB (820,224 bytes) 1/18/2008 10:08 PM
Microsoft Corporation
e:\windows\system32\user32.dll
kernel32 6.0.6001.18000 1.16 MB (1,213,952 bytes) 1/18/2008 10:01 PM Microsoft Corporation
e:\windows\system32\kernel32.dll
gdi32 6.0.6001.18023 379.00 KB (388,096 bytes) 10/6/2008 11:11 AM
Microsoft Corporation
e:\windows\system32\gdi32.dll
advapi32 6.0.6001.18000 1.01 MB (1,062,400 bytes) 1/18/2008 11:13 PM Microsoft Corporation
e:\windows\system32\advapi32.dll
rpcrt4 6.0.6001.18000 1.22 MB (1,281,024 bytes) 1/18/2008 10:28 PM Microsoft Corporation
e:\windows\system32\rpcrt4.dll
lpk 6.0.6001.18000 32.00 KB (32,768 bytes) 1/18/2008 10:08 PM
Microsoft Corporation
e:\windows\system32\lpk.dll
usp10 1.626.6001.18000 607.50 KB (622,080 bytes) 1/18/2008 10:08 PM
Microsoft Corporation
e:\windows\system32\usp10.dll
msvcrt 7.0.6001.18000 606.50 KB (621,056 bytes) 1/18/2008 9:52 PM
Microsoft Corporation
e:\windows\system32\msvcrt.dll
sxs 6.0.6001.18000 560.50 KB (573,952 bytes) 1/18/2008 10:00 PM
Microsoft Corporation
e:\windows\system32\sxs.dll
wininit 6.0.6001.18000 121.00 KB (123,904 bytes) 1/18/2008 10:17 PM
Microsoft Corporation
e:\windows\system32\wininit.exe
userenv 6.0.6001.18000 134.00 KB (137,216 bytes) 1/18/2008 10:16 PM
Microsoft Corporation
e:\windows\system32\userenv.dll
secur32 6.0.6001.18000 92.00 KB (94,208 bytes) 1/18/2008 10:16 PM
Microsoft Corporation
e:\windows\system32\secur32.dll
imm32 6.0.6001.18000 160.00 KB (163,840 bytes) 1/18/2008 10:07 PM
Microsoft Corporation
e:\windows\system32\imm32.dll
msctf 6.0.6001.18000 1,016.00 KB (1,040,384 bytes) 1/18/2008 10:09 PM Microsoft Corporation
e:\windows\system32\msctf.dll
apphelp 6.0.6001.18000 195.50 KB (200,192 bytes) 1/18/2008 9:52 PM
Microsoft Corporation
e:\windows\system32\apphelp.dll
ws2_32 6.0.6001.18000 259.00 KB (265,216 bytes) 1/18/2008 10:38 PM
Microsoft Corporation
e:\windows\system32\ws2_32.dll
nsi 6.0.6001.18000 11.00 KB (11,264 bytes) 1/18/2008 10:36 PM
Microsoft Corporation
e:\windows\system32\nsi.dll
mswsock 6.0.6001.18000 297.00 KB (304,128 bytes) 1/18/2008 10:38 PM
Microsoft Corporation
e:\windows\system32\mswsock.dll
wshtcpip 6.0.6001.18000 12.50 KB (12,800 bytes) 1/18/2008 10:36 PM
Microsoft Corporation
e:\windows\system32\wshtcpip.dll

© 2009 Hewlett-Packard Company. All rights reserved.

wship6 6.0.6001.18000 11.00 KB (11,264 bytes) 1/18/2008 10:36 PM Microsoft Corporation e:\windows\system32\wship6.dll
credssp 6.0.6001.18000 18.00 KB (18,432 bytes) 1/18/2008 10:16 PM Microsoft Corporation e:\windows\system32\credssp.dll
crypt32 6.0.6001.18000 1.20 MB (1,254,400 bytes) 1/18/2008 10:15 PM Microsoft Corporation e:\windows\system32\crypt32.dll
msasn1 6.0.6001.18000 79.00 KB (80,896 bytes) 1/18/2008 10:57 PM Microsoft Corporation e:\windows\system32\msasn1.dll
schannel 6.0.6001.18000 326.50 KB (334,336 bytes) 1/18/2008 10:16 PM Microsoft Corporation e:\windows\system32\schannel.dll
netapi32 6.0.6001.18000 633.50 KB (648,704 bytes) 1/18/2008 10:19 PM Microsoft Corporation e:\windows\system32\netapi32.dll
psapi 6.0.6001.18000 16.50 KB (16,896 bytes) 1/18/2008 10:40 PM Microsoft Corporation e:\windows\system32\psapi.dll
services 6.0.6001.18000 375.50 KB (384,512 bytes) 1/18/2008 10:03 PM Microsoft Corporation e:\windows\system32\services.exe
scesrv 6.0.6001.18000 390.00 KB (399,360 bytes) 1/18/2008 10:15 PM Microsoft Corporation e:\windows\system32\scesrv.dll
authz 6.0.6001.18000 139.50 KB (142,848 bytes) 1/18/2008 10:16 PM Microsoft Corporation e:\windows\system32\authz.dll
ncobjapi 6.0.6001.18000 68.50 KB (70,144 bytes) 1/18/2008 10:13 PM Microsoft Corporation e:\windows\system32\ncobjapi.dll
ntmarta 6.0.6001.18000 155.50 KB (159,232 bytes) 1/18/2008 10:16 PM Microsoft Corporation e:\windows\system32\ntmarta.dll
wldap32 6.0.6001.18000 321.00 KB (328,704 bytes) 1/18/2008 10:20 PM Microsoft Corporation e:\windows\system32\wldap32.dll
samlib 6.0.6001.18000 97.00 KB (99,328 bytes) 1/18/2008 10:19 PM Microsoft Corporation e:\windows\system32\samlib.dll
ole32 6.0.6001.18000 1.83 MB (1,923,072 bytes) 1/18/2008 10:30 PM Microsoft Corporation e:\windows\system32\ole32.dll
lsass 6.0.6001.18000 11.00 KB (11,264 bytes) 1/18/2008 10:16 PM Microsoft Corporation e:\windows\system32\lsass.exe
lsasrv 6.0.6001.18000 1.61 MB (1,692,160 bytes) 1/18/2008 11:21 PM Microsoft Corporation e:\windows\system32\lsasrv.dll
samsrv 6.0.6001.18000 651.00 KB (666,624 bytes) 1/18/2008 10:19 PM Microsoft Corporation e:\windows\system32\samsrv.dll
cryptdll 6.0.6001.18000 63.50 KB (65,024 bytes) 1/18/2008 10:15 PM Microsoft Corporation e:\windows\system32\cryptdll.dll
dnsapi 6.0.6001.18000 214.50 KB (219,648 bytes) 1/18/2008 10:20 PM Microsoft Corporation e:\windows\system32\dnsapi.dll
ntdsapi 6.0.6001.18000 143.00 KB (146,432 bytes) 1/18/2008 10:20 PM Microsoft Corporation e:\windows\system32\ntdsapi.dll
feclient 6.0.6001.18000 67.00 KB (68,608 bytes) 1/18/2008 10:18 PM Microsoft Corporation e:\windows\system32\feclient.dll
mpr 6.0.6001.18000 83.50 KB (85,504 bytes) 1/18/2008 10:38 PM Microsoft Corporation e:\windows\system32\mpr.dll
slc 6.0.6001.18000 146.50 KB (150,016 bytes) 1/18/2008 11:32 PM Microsoft Corporation e:\windows\system32\slc.dll
sysntfy 6.0.6000.16386 21.00 KB (21,504 bytes) 1/18/2008 10:17 PM Microsoft Corporation e:\windows\system32\sysntfy.dll
wevtapi 6.0.6001.18000 384.50 KB (393,728 bytes) 1/18/2008 10:12 PM Microsoft Corporation e:\windows\system32\wevtapi.dll
iphlpapi 6.0.6001.18000 124.00 KB (126,976 bytes) 1/18/2008 10:36 PM Microsoft Corporation e:\windows\system32\iphlpapi.dll
dhcpcsvc 6.0.6001.18000 262.00 KB (268,288 bytes) 1/18/2008 10:35 PM Microsoft Corporation e:\windows\system32\dhcpcsvc.dll
winnsi 6.0.6001.18000 21.50 KB (22,016 bytes) 1/18/2008 10:36 PM Microsoft Corporation e:\windows\system32\winnsi.dll
dhcpcsvc6 6.0.6001.18000 155.00 KB (158,720 bytes) 1/18/2008 10:35 PM Microsoft Corporation e:\windows\system32\dhcpcsvc6.dll
cngaudit 6.0.6000.16386 14.50 KB (14,848 bytes) 1/18/2008 10:15 PM Microsoft Corporation e:\windows\system32\cngaudit.dll
ncrypt 6.0.6001.18000 247.50 KB (253,440 bytes) 1/18/2008 10:15 PM Microsoft Corporation e:\windows\system32\ncrypt.dll
bcrypt 6.0.6001.18000 299.50 KB (306,688 bytes) 1/18/2008 10:15 PM Microsoft Corporation e:\windows\system32\bcrypt.dll
msprivs 6.0.6000.16386 2.00 KB (2,048 bytes) 1/18/2008 10:16 PM Microsoft Corporation e:\windows\system32\msprivs.dll
kerberos 6.0.6001.18000 639.50 KB (654,848 bytes) 1/18/2008 10:17 PM Microsoft Corporation e:\windows\system32\kerberos.dll
msv1_0 6.0.6001.18000 259.50 KB (265,728 bytes) 1/18/2008 10:16 PM Microsoft Corporation e:\windows\system32\msv1_0.dll
netlogon 6.0.6001.18000 700.00 KB (716,800 bytes) 1/18/2008 10:19 PM Microsoft Corporation e:\windows\system32\netlogon.dll
winbrand 6.0.6001.18000 851.00 KB (871,424 bytes) 1/18/2008 10:02 PM Microsoft Corporation e:\windows\system32\winbrand.dll
wdigest 6.0.6001.18000 193.00 KB (197,632 bytes) 1/18/2008 10:16 PM Microsoft Corporation e:\windows\system32\wdigest.dll
rsaenh 6.0.6001.18000 283.05 KB (289,848 bytes) 1/18/2008 10:18 PM Microsoft Corporation e:\windows\system32\rsaenh.dll
tspkg 6.0.6001.18000 77.00 KB (78,848 bytes) 1/18/2008 10:16 PM Microsoft Corporation e:\windows\system32\tspkg.dll
gpapi 6.0.6001.18000 82.50 KB (84,480 bytes) 1/18/2008 10:20 PM Microsoft Corporation e:\windows\system32\gpapi.dll
setupapi 6.0.6001.18000 1.83 MB (1,921,536 bytes) 1/18/2008 10:00 PM Microsoft Corporation e:\windows\system32\setupapi.dll
oleaut32 6.0.6001.18000 828.00 KB (847,872 bytes) 1/18/2008 10:27 PM Microsoft Corporation e:\windows\system32\oleaut32.dll
scecli 6.0.6001.18000 230.00 KB (235,520 bytes) 1/18/2008 10:15 PM Microsoft Corporation e:\windows\system32\scecli.dll
rassfm 6.0.6001.18000 25.50 KB (26,112 bytes) 1/19/2008 5:51 AM Microsoft Corporation e:\windows\system32\rassfm.dll
dssenh 6.0.6001.18000 197.55 KB (202,296 bytes) 1/18/2008 10:18 PM Microsoft Corporation e:\windows\system32\dssenh.dll
winsta 6.0.6001.18000 200.50 KB (205,312 bytes) 1/18/2008 10:43 PM Microsoft Corporation e:\windows\system32\winsta.dll
cryptnet 6.0.6001.18000 127.00 KB (130,048 bytes) 1/18/2008 10:15 PM Microsoft Corporation e:\windows\system32\cryptnet.dll
sensapi 6.0.6001.18000 12.50 KB (12,800 bytes) 1/18/2008 10:27 PM Microsoft Corporation e:\windows\system32\sensapi.dll
shlwapi 6.0.6001.18000 443.50 KB (454,144 bytes) 1/18/2008 10:21 PM Microsoft Corporation e:\windows\system32\shlwapi.dll
comctl32 6.10.6001.18000 1.95 MB (2,049,024 bytes) 1/19/2008 1:51 AM Microsoft Corporation e:\windows\winsxs\amd64_microsoft.windows.common-controls_6595b64144ccf1df_6.0.6001.18000_none_152e7382f3bd50c6\comctl32.dll
lsm 6.0.6001.18000 258.50 KB (264,704 bytes) 1/18/2008 10:43 PM Microsoft Corporation e:\windows\system32\lsm.exe
wmsgapi 6.0.6000.16386 14.00 KB (14,336 bytes) 1/18/2008 10:17 PM Microsoft Corporation e:\windows\system32\wmsgapi.dll
clbcatq 2001.12.6931.18000 597.00 KB (611,328 bytes) 1/18/2008 10:28

© 2009 Hewlett-Packard Company. All rights reserved.

PM Microsoft Corporation
e:\windows\system32\clbcatq.dll
lsmproxy 6.0.6001.18000 43.50 KB
(44,544 bytes) 1/18/2008 10:42 PM
Microsoft Corporation
e:\windows\system32\lsmproxy.dll
svchost 6.0.6001.18000 27.00 KB
(27,648 bytes) 1/18/2008 10:02 PM
Microsoft Corporation
e:\windows\system32\svchost.exe
umpnpmgr 6.0.6001.18000 304.50 KB
(311,808 bytes) 1/18/2008 9:59 PM
Microsoft Corporation
e:\windows\system32\umpnpmgr.dll
powrprof 6.0.6001.18000 118.50 KB
(121,344 bytes) 1/18/2008 10:23 PM
Microsoft Corporation
e:\windows\system32\powrprof.dll
rpcss 6.0.6001.18000 697.00 KB
(713,728 bytes) 1/18/2008 10:27 PM
Microsoft Corporation
e:\windows\system32\rpcss.dll
FirewallAPI 6.0.6001.18000 685.00 KB
(701,440 bytes) 1/18/2008 10:35 PM
Microsoft Corporation
e:\windows\system32\firewallapi.dll
version 6.0.6001.18000 26.50 KB
(27,136 bytes) 1/18/2008 10:24 PM
Microsoft Corporation
e:\windows\system32\version.dll
wtsapi32 6.0.6001.18000 30.50 KB
(31,232 bytes) 1/18/2008 10:42 PM
Microsoft Corporation
e:\windows\system32\wtsapi32.dll
cabinet 6.0.6001.18000 91.00 KB
(93,184 bytes) 1/18/2008 10:23 PM
Microsoft Corporation
e:\windows\system32\cabinet.dll
wevtsvc 6.0.6001.18000 1.42 MB
(1,486,336 bytes) 1/18/2008 10:13
PM Microsoft Corporation
e:\windows\system32\wevtsvc.dll
lmhsvc 6.0.6001.18000 23.50 KB
(24,064 bytes) 1/18/2008 10:36 PM
Microsoft Corporation
e:\windows\system32\lmhsvc.dll
gpsvc 6.0.6001.18000 701.50 KB
(718,336 bytes) 1/18/2008 10:20 PM
Microsoft Corporation
e:\windows\system32\gpsvc.dll
nlaapi 6.0.6001.18000 60.00 KB
(61,440 bytes) 1/18/2008 10:36 PM
Microsoft Corporation
e:\windows\system32\nlaapi.dll
wbemprox 6.0.6001.18000 42.50 KB
(43,520 bytes) 1/18/2008 10:12 PM
Microsoft Corporation
e:\windows\system32\wbem\wbemp
rox.dll
wbemcomn 6.0.6001.18000 516.00 KB
(528,384 bytes) 1/18/2008 10:13 PM
Microsoft Corporation
e:\windows\system32\wbemcomn.dll
wbemsvc 6.0.6001.18000 121.00 KB
(123,904 bytes) 1/18/2008 10:12 PM
Microsoft Corporation
e:\windows\system32\wbem\wbems
vc.dll
fastprox 6.0.6001.18000 869.50 KB
(890,368 bytes) 1/18/2008 10:13 PM
Microsoft Corporation
e:\windows\system32\wbem\fastpro
x.dll
profsvc 6.0.6001.18000 174.50 KB
(178,688 bytes) 1/18/2008 10:16 PM
Microsoft Corporation
e:\windows\system32\profsvc.dll
atl 3.5.2284.0 85.50 KB (87,552 bytes)
1/18/2008 11:09 PM Microsoft
Corporation e:\windows\system32\atl.dll
sens 6.0.6001.18000 60.50 KB
(61,952 bytes) 1/18/2008 10:27 PM
Microsoft Corporation
e:\windows\system32\sens.dll
schedsvc 6.0.6001.18000 824.00 KB
(843,776 bytes) 1/18/2008 10:13 PM
Microsoft Corporation
e:\windows\system32\schedsvc.dll
ktmw32 6.0.6001.18000 14.50 KB
(14,848 bytes) 1/18/2008 9:52 PM
Microsoft Corporation
e:\windows\system32\ktmw32.dll
taskcomp 6.0.6001.18000 400.00 KB
(409,600 bytes) 1/18/2008 10:13 PM
Microsoft Corporation
e:\windows\system32\taskcomp.dll
seclogon 6.0.6001.18000 28.00 KB
(28,672 bytes) 1/18/2008 10:18 PM
Microsoft Corporation
e:\windows\system32\seclogon.dll
wmisvc 6.0.6001.18000 216.50 KB
(221,696 bytes) 1/18/2008 10:13 PM
Microsoft Corporation
e:\windows\system32\wbem\wmisvc
.dll
iphlpsvc 6.0.6001.18000 218.00 KB
(223,232 bytes) 1/18/2008 10:36 PM
Microsoft Corporation
e:\windows\system32\iphlpsvc.dll
fwpuclnt 6.0.6001.18000 761.50 KB
(779,776 bytes) 1/18/2008 10:36 PM
Microsoft Corporation
e:\windows\system32\fwpuclnt.dll
rtutils 6.0.6001.18000 49.50 KB
(50,688 bytes) 1/18/2008 10:37 PM
Microsoft Corporation
e:\windows\system32\rtutils.dll
sqmapi 6.0.6001.18000 172.00 KB
(176,128 bytes) 1/18/2008 10:11 PM
Microsoft Corporation
e:\windows\system32\sqmapi.dll
srvsvc 6.0.6001.18000 172.50 KB
(176,640 bytes) 1/18/2008 10:18 PM
Microsoft Corporation
e:\windows\system32\srvsvc.dll
sscore 6.0.6000.16386 12.00 KB
(12,288 bytes) 1/18/2008 10:18 PM
Microsoft Corporation
e:\windows\system32\sscore.dll
clusapi 6.0.6001.18000 237.50 KB
(243,200 bytes) 1/18/2008 10:05 PM
Microsoft Corporation
e:\windows\system32\clusapi.dll
activeds 6.0.6001.18000 259.50 KB
(265,728 bytes) 1/18/2008 10:19 PM
Microsoft Corporation
e:\windows\system32\activeds.dll
adsldpc 6.0.6001.18000 224.00 KB
(229,376 bytes) 1/18/2008 10:19 PM
Microsoft Corporation
e:\windows\system32\adsldpc.dll
credui 6.0.6001.18000 186.50 KB
(190,976 bytes) 1/18/2008 10:18 PM
Microsoft Corporation
e:\windows\system32\credui.dll
shell32 6.0.6001.18062 12.30 MB
(12,897,280 bytes) 10/6/2008 11:12
AM Microsoft Corporation
e:\windows\system32\shell32.dll
resutils 6.0.6001.18000 76.00 KB
(77,824 bytes) 1/18/2008 10:04 PM
Microsoft Corporation
e:\windows\system32\resutils.dll
wintrust 6.0.6001.18000 213.00 KB
(218,112 bytes) 1/18/2008 10:15 PM
Microsoft Corporation
e:\windows\system32\wintrust.dll
imagehlp 6.0.6001.18000 72.50 KB
(74,240 bytes) 1/18/2008 10:40 PM
Microsoft Corporation
e:\windows\system32\imagehlp.dll
vssapi 6.0.6001.18000 1.43 MB
(1,494,528 bytes) 1/18/2008 10:30
PM Microsoft Corporation
e:\windows\system32\vssapi.dll
vsstrace 6.0.6001.18000 90.00 KB
(92,160 bytes) 1/18/2008 10:29 PM
Microsoft Corporation
e:\windows\system32\vsstrace.dll
xmllite 1.2.1009.0 176.00 KB (180,224
bytes) 1/18/2008 11:13 PM
Microsoft Corporation
e:\windows\system32\xmllite.dll
propsys 6.0.6001.18000 895.50 KB
(916,992 bytes) 1/18/2008 10:21 PM
Microsoft Corporation
e:\windows\system32\propsys.dll
wbemcore 6.0.6001.18000 1.12 MB
(1,171,456 bytes) 1/18/2008 10:14
PM Microsoft Corporation
e:\windows\system32\wbem\wbemc
ore.dll
esscli 6.0.6001.18000 418.00 KB
(428,032 bytes) 1/18/2008 10:12 PM
Microsoft Corporation
e:\windows\system32\wbem\esscli.d
ll
wmiutils 6.0.6001.18000 128.50 KB
(131,584 bytes) 1/18/2008 10:12 PM
Microsoft Corporation
e:\windows\system32\wbem\wmiutil
s.dll
repdrvfs 6.0.6001.18000 372.50 KB
(381,440 bytes) 1/18/2008 10:13 PM
Microsoft Corporation
e:\windows\system32\wbem\repdrvf
s.dll
wmiprvsd 6.0.6001.18000 671.50 KB
(687,616 bytes) 1/18/2008 10:13 PM
Microsoft Corporation
e:\windows\system32\wbem\wmiprv
sd.dll
wbemess 6.0.6001.18000 501.00 KB
(513,024 bytes) 1/18/2008 10:13 PM
Microsoft Corporation
e:\windows\system32\wbem\wbeme
ss.dll
winrnr 6.0.6001.18000 27.00 KB
(27,648 bytes) 1/18/2008 10:19 PM
Microsoft Corporation
e:\windows\system32\winrnr.dll
napinsp 6.0.6001.18000 61.50 KB
(62,976 bytes) 1/18/2008 10:37 PM
Microsoft Corporation
e:\windows\system32\napinsp.dll
ncprov 6.0.6001.18000 77.50 KB
(79,360 bytes) 1/18/2008 10:13 PM
Microsoft Corporation
e:\windows\system32\wbem\ncprov.
dll
sessenv 6.0.6001.18000 73.00 KB
(74,752 bytes) 1/18/2008 10:43 PM
Microsoft Corporation
e:\windows\system32\sessenv.dll

© 2009 Hewlett-Packard Company. All rights reserved.

rasadhlp 6.0.6001.18000 13.00 KB (13,312 bytes) 1/18/2008 10:37 PM Microsoft Corporation e:\windows\system32\rasadhlp.dll
TSChannel 6.0.6000.16386 18.50 KB (18,944 bytes) 1/18/2008 10:12 PM Microsoft Corporation e:\windows\system32\tschannel.dll
qmgr 7.0.6001.18000 1.03 MB (1,082,368 bytes) 1/18/2008 10:12 PM Microsoft Corporation e:\windows\system32\qmgr.dll
shfolder 6.0.6001.18000 10.00 KB (10,240 bytes) 1/18/2008 10:22 PM Microsoft Corporation e:\windows\system32\shfolder.dll
winhttp 6.0.6001.18000 429.50 KB (439,808 bytes) 1/18/2008 10:26 PM Microsoft Corporation e:\windows\system32\winhttp.dll
bitsperf 7.0.6000.16386 22.50 KB (23,040 bytes) 1/18/2008 10:11 PM Microsoft Corporation e:\windows\system32\bitsperf.dll
bitsigd 7.0.6001.18000 45.50 KB (46,592 bytes) 1/18/2008 10:11 PM Microsoft Corporation e:\windows\system32\bitsigd.dll
wuaueng 7.2.6001.784 2.19 MB (2,291,912 bytes) 10/6/2008 11:10 AM Microsoft Corporation e:\windows\system32\wuaueng.dll
esent 6.0.6001.18000 2.41 MB (2,522,624 bytes) 1/18/2008 10:17 PM Microsoft Corporation e:\windows\system32\esent.dll
winspool 6.0.6001.18000 333.50 KB (341,504 bytes) 1/18/2008 11:11 PM Microsoft Corporation e:\windows\system32\winspool.drv
mspatcha 6.0.6001.18000 45.50 KB (46,592 bytes) 1/18/2008 10:05 PM Microsoft Corporation e:\windows\system32\mspatcha.dll
rasapi32 6.0.6001.18000 329.50 KB (337,408 bytes) 1/18/2008 10:37 PM Microsoft Corporation e:\windows\system32\rasapi32.dll
rasman 6.0.6001.18000 90.50 KB (92,672 bytes) 1/18/2008 10:37 PM Microsoft Corporation e:\windows\system32\rasman.dll
tapi32 6.0.6000.16386 238.00 KB (243,712 bytes) 1/18/2008 11:13 PM Microsoft Corporation e:\windows\system32\tapi32.dll
winmm 6.0.6001.18000 207.00 KB (211,968 bytes) 1/18/2008 10:44 PM Microsoft Corporation e:\windows\system32\winmm.dll
oleacc 4.2.5406.0 300.50 KB (307,712 bytes) 1/18/2008 10:08 PM Microsoft Corporation e:\windows\system32\oleacc.dll
rasmans 6.0.6001.18000 301.00 KB (308,224 bytes) 1/18/2008 10:37 PM Microsoft Corporation e:\windows\system32\rasmans.dll
rastapi 6.0.6001.18000 79.50 KB (81,408 bytes) 1/18/2008 10:37 PM Microsoft Corporation e:\windows\system32\rastapi.dll
rasppp 6.0.6001.18000 299.00 KB (306,176 bytes) 1/18/2008 10:37 PM Microsoft Corporation e:\windows\system32\rasppp.dll
mprapi 6.0.6001.18000 126.50 KB (129,536 bytes) 1/18/2008 10:37 PM Microsoft Corporation e:\windows\system32\mprapi.dll
rasqec 6.0.6001.18000 72.00 KB (73,728 bytes) 1/18/2008 10:37 PM Microsoft Corporation e:\windows\system32\rasqec.dll
qutil 6.0.6001.18000 97.00 KB (99,328 bytes) 1/18/2008 10:34 PM Microsoft Corporation e:\windows\system32\qutil.dll
raschap 6.0.6001.18000 289.50 KB (296,448 bytes) 1/18/2008 10:37 PM Microsoft Corporation e:\windows\system32\raschap.dll
rastls 6.0.6001.18000 274.00 KB (280,576 bytes) 1/18/2008 10:37 PM Microsoft Corporation e:\windows\system32\rastls.dll
cryptui 6.0.6001.18000 1,010.50 KB (1,034,752 bytes) 1/18/2008 10:15 PM Microsoft Corporation e:\windows\system32\cryptui.dll
msimg32 6.0.6001.18000 8.00 KB (8,192 bytes) 1/18/2008 10:07 PM Microsoft Corporation e:\windows\system32\msimg32.dll
winscard 6.0.6001.18000 186.00 KB (190,464 bytes) 1/18/2008 10:15 PM Microsoft Corporation e:\windows\system32\winscard.dll
comctl32 5.82.6001.18000 619.00 KB (633,856 bytes) 1/19/2008 1:48 AM Microsoft Corporation e:\windows\winsxs\amd64_microsoft.windows.common-controls_6595b64144ccf1df_5.82.6001.18000_none_40ba501d3c2b20ff\comctl32.dll
es 2001.12.6931.18057 353.50 KB (361,984 bytes) 10/6/2008 11:12 AM Microsoft Corporation e:\windows\system32\es.dll
npmproxy 6.0.6000.16386 31.50 KB (32,256 bytes) 1/18/2008 10:38 PM Microsoft Corporation e:\windows\system32\npmproxy.dll
slsvc 6.0.6001.18000 2.06 MB (2,161,664 bytes) 1/18/2008 11:33 PM Microsoft Corporation e:\windows\system32\slsvc.exe
nsisvc 6.0.6001.18000 24.00 KB (24,576 bytes) 1/18/2008 10:36 PM Microsoft Corporation e:\windows\system32\nsisvc.dll
wkssvc 6.0.6001.18000 198.00 KB (202,752 bytes) 1/18/2008 10:18 PM Microsoft Corporation e:\windows\system32\wkssvc.dll
netprofm 6.0.6001.18000 297.00 KB (304,128 bytes) 1/18/2008 10:38 PM Microsoft Corporation e:\windows\system32\netprofm.dll
sstpsvc 6.0.6001.18000 138.00 KB (141,312 bytes) 1/18/2008 10:37 PM Microsoft Corporation e:\windows\system32\sstpsvc.dll
httpapi 6.0.6001.18000 32.50 KB (33,280 bytes) 1/18/2008 10:35 PM Microsoft Corporation e:\windows\system32\httpapi.dll
normaliz 6.0.6000.16386 3.00 KB (3,072 bytes) 1/18/2008 9:59 PM Microsoft Corporation e:\windows\system32\normaliz.dll
uxsms 6.0.6001.18000 32.00 KB (32,768 bytes) 1/18/2008 10:10 PM Microsoft Corporation e:\windows\system32\uxsms.dll
trkwks 6.0.6001.18000 114.50 KB (117,248 bytes) 1/18/2008 10:27 PM Microsoft Corporation e:\windows\system32\trkwks.dll
umrdp 6.0.6001.18000 247.00 KB (252,928 bytes) 1/19/2008 5:52 AM Microsoft Corporation e:\windows\system32\umrdp.dll
umb 6.0.6001.18000 58.50 KB (59,904 bytes) 1/18/2008 10:06 PM Microsoft Corporation e:\windows\system32\umb.dll
wdi 6.0.6001.18000 80.00 KB (81,920 bytes) 1/18/2008 10:03 PM Microsoft Corporation e:\windows\system32\wdi.dll
radardt 6.0.6000.16386 77.50 KB (79,360 bytes) 1/19/2008 5:52 AM Microsoft Corporation e:\windows\system32\radardt.dll
netman 6.0.6001.18000 340.00 KB (348,160 bytes) 1/18/2008 10:35 PM Microsoft Corporation e:\windows\system32\netman.dll
netshell 6.0.6001.18000 3.19 MB (3,341,312 bytes) 1/18/2008 10:35 PM Microsoft Corporation e:\windows\system32\netshell.dll
rasdlg 6.0.6001.18000 890.00 KB (911,360 bytes) 1/18/2008 10:37 PM Microsoft Corporation e:\windows\system32\rasdlg.dll
hnetcfg 6.0.6001.18000 423.50 KB (433,664 bytes) 1/18/2008 10:35 PM Microsoft Corporation e:\windows\system32\hnetcfg.dll
netcfgx 6.0.6001.18000 492.00 KB (503,808 bytes) 1/18/2008 10:35 PM Microsoft Corporation e:\windows\system32\netcfgx.dll
dnsrslvr 6.0.6001.18000 115.00 KB (117,760 bytes) 1/18/2008 10:20 PM Microsoft Corporation e:\windows\system32\dnsrslvr.dll
nlasvc 6.0.6001.18000 201.50 KB (206,336 bytes) 1/18/2008 10:36 PM Microsoft Corporation e:\windows\system32\nlasvc.dll
ncsi 6.0.6001.18000 106.50 KB (109,056 bytes) 1/18/2008 10:35 PM Microsoft Corporation e:\windows\system32\ncsi.dll
cfgmgr32 6.0.6001.18000 17.50 KB (17,920 bytes) 1/18/2008 9:59 PM Microsoft Corporation e:\windows\system32\cfgmgr32.dll
ssdpapi 6.0.6000.16386 49.00 KB (50,176 bytes) 1/18/2008 10:38 PM Microsoft Corporation e:\windows\system32\ssdpapi.dll
termsrv 6.0.6001.18000 534.00 KB (546,816 bytes) 1/18/2008 10:43 PM Microsoft Corporation e:\windows\system32\termsrv.dll
icaapi 6.0.6000.16386 20.00 KB (20,480 bytes) 1/18/2008 10:42 PM

© 2009 Hewlett-Packard Company. All rights reserved.

Microsoft Corporation
e:\windows\system32\icaapi.dll
regapi 6.0.6001.18000 87.00 KB (89,088 bytes) 1/18/2008 10:42 PM
Microsoft Corporation
e:\windows\system32\regapi.dll
rdpwsx 6.0.6001.18000 115.00 KB (117,760 bytes) 1/18/2008 10:42 PM
Microsoft Corporation
e:\windows\system32\rdpwsx.dll
mstlsapi 6.0.6001.18000 135.00 KB (138,240 bytes) 1/18/2008 10:42 PM
Microsoft Corporation
e:\windows\system32\mstlsapi.dll
msdtckrm 2001.12.6931.18000 386.00 KB (395,264 bytes)
1/18/2008 10:27 PM Microsoft Corporation
e:\windows\system32\msdtckrm.dll
wsmsvc 6.0.6001.18000 1.04 MB (1,091,072 bytes) 1/18/2008 10:14 PM Microsoft Corporation
e:\windows\system32\wsmsvc.dll
wsmprov 6.0.6001.18000 71.50 KB (73,216 bytes) 1/18/2008 10:13 PM
Microsoft Corporation
e:\windows\system32\wsmprov.dll
winrsmgr 6.0.6001.18000 294.00 KB (301,056 bytes) 1/18/2008 10:14 PM
Microsoft Corporation
e:\windows\system32\winrsmgr.dll
wsmres 6.0.6001.18000 13.00 KB (13,312 bytes) 1/18/2008 10:13 PM
Microsoft Corporation
e:\windows\system32\wsmres.dll
wevtfwd 6.0.6001.18000 104.50 KB (107,008 bytes) 1/18/2008 10:12 PM
Microsoft Corporation
e:\windows\system32\wevtfwd.dll
cissesrv 6.8.0.64 147.00 KB (150,528 bytes) 11/6/2007 1:01 PM
Hewlett-Packard Company
e:\program files\hp\cissesrv\cissesrv.exe
dps 6.0.6001.18000 136.00 KB (139,264 bytes) 1/18/2008 10:03 PM
Microsoft Corporation
e:\windows\system32\dps.dll
taskschd 6.0.6001.18000 640.50 KB (655,872 bytes) 1/18/2008 10:13 PM
Microsoft Corporation
e:\windows\system32\taskschd.dll
regsvc 6.0.6001.18000 201.50 KB (206,336 bytes) 1/18/2008 10:03 PM
Microsoft Corporation
e:\windows\system32\regsvc.dll
sqlwriter 2007.100.1600.22 154.02 KB (157,720 bytes) 7/10/2008 5:31 AM
Microsoft Corporation e:\program files\microsoft sql server\90\shared\sqlwriter.exe
msvcr80 8.0.50727.3053 787.00 KB (805,888 bytes) 10/4/2008 9:26 PM
Microsoft Corporation
e:\windows\winsxs\amd64_microsoft.vc80.crt_1fc8b3b9a1e18e3b_8.0.50727.3053_none_88e044e32fae7230\msvcr80.dll
sqlwvss 2007.100.1600.22 384.02 KB (393,240 bytes) 7/10/2008 5:31 AM
Microsoft Corporation e:\program files\microsoft sql server\90\shared\sqlwvss.dll
msvcp80 8.0.50727.3053 1.02 MB (1,071,616 bytes) 10/4/2008 9:26 PM
Microsoft Corporation
e:\windows\winsxs\amd64_microsoft.vc80.crt_1fc8b3b9a1e18e3b_8.0.50727.3053_none_88e044e32fae7230\msvcp80.dll
taskeng 6.0.6001.18000 259.00 KB (265,216 bytes) 1/18/2008 10:13 PM
Microsoft Corporation
e:\windows\system32\taskeng.exe
dimsjob 6.0.6001.18000 43.00 KB (44,032 bytes) 1/18/2008 10:18 PM
Microsoft Corporation
e:\windows\system32\dimsjob.dll
pautoenr 6.0.6000.16386 46.00 KB (47,104 bytes) 1/18/2008 10:18 PM
Microsoft Corporation
e:\windows\system32\pautoenr.dll
certcli 6.0.6001.18000 434.00 KB (444,416 bytes) 1/18/2008 10:16 PM
Microsoft Corporation
e:\windows\system32\certcli.dll
CertEnroll 6.0.6001.18000 1.58 MB (1,658,368 bytes) 1/18/2008 10:19 PM Microsoft Corporation
e:\windows\system32\certenroll.dll
wininet 7.0.6001.18099 989.50 KB (1,013,248 bytes) 10/6/2008 11:12 AM Microsoft Corporation
e:\windows\system32\wininet.dll
iertutil 7.0.6001.18000 366.00 KB (374,784 bytes) 1/18/2008 10:25 PM
Microsoft Corporation
e:\windows\system32\iertutil.dll
winlogon 6.0.6001.18000 396.50 KB (406,016 bytes) 1/18/2008 10:18 PM
Microsoft Corporation
e:\windows\system32\winlogon.exe
shsvcs 6.0.6001.18000 294.50 KB (301,568 bytes) 1/18/2008 10:21 PM
Microsoft Corporation
e:\windows\system32\shsvcs.dll
uxtheme 6.0.6001.18000 310.00 KB (317,440 bytes) 1/18/2008 10:21 PM
Microsoft Corporation
e:\windows\system32\uxtheme.dll
MsCtfMonitor 6.0.6000.16386 25.50 KB (26,112 bytes) 1/18/2008 10:08 PM Microsoft Corporation
e:\windows\system32\msctfmonitor.dll
msutb 6.0.6001.18000 222.50 KB (227,840 bytes) 1/18/2008 10:08 PM
Microsoft Corporation
e:\windows\system32\msutb.dll
dwmapi 6.0.6001.18000 38.50 KB (39,424 bytes) 1/18/2008 10:10 PM
Microsoft Corporation
e:\windows\system32\dwmapi.dll
PlaySndSrv 6.0.6000.16386 74.50 KB (76,288 bytes) 1/18/2008 10:43 PM
Microsoft Corporation
e:\windows\system32\playsndsrv.dll
mmdevapi 6.0.6001.18000 197.50 KB (202,240 bytes) 1/18/2008 10:44 PM
Microsoft Corporation
e:\windows\system32\mmdevapi.dll
qagent 6.0.6001.18000 245.00 KB (250,880 bytes) 1/18/2008 10:35 PM
Microsoft Corporation
e:\windows\system32\qagent.dll
dwm 6.0.6001.18000 96.50 KB (98,816 bytes) 1/18/2008 10:10 PM
Microsoft Corporation
e:\windows\system32\dwm.exe
dwmredir 6.0.6001.18000 100.00 KB (102,400 bytes) 1/18/2008 10:10 PM
Microsoft Corporation
e:\windows\system32\dwmredir.dll
slwga 6.0.6001.18000 14.00 KB (14,336 bytes) 1/18/2008 10:17 PM
Microsoft Corporation
e:\windows\system32\slwga.dll
urlmon 7.0.6001.18099 1.35 MB (1,418,240 bytes) 10/6/2008 11:12 AM Microsoft Corporation
e:\windows\system32\urlmon.dll
milcore 6.0.6001.18000 2.45 MB (2,570,240 bytes) 1/18/2008 10:12 PM Microsoft Corporation
e:\windows\system32\milcore.dll
explorer 6.0.6001.18000 2.94 MB (3,080,704 bytes) 1/18/2008 10:22 PM Microsoft Corporation
e:\windows\explorer.exe
shdocvw 6.0.6001.18000 1.14 MB (1,195,008 bytes) 1/18/2008 10:22 PM Microsoft Corporation
e:\windows\system32\shdocvw.dll
gdiplus 5.2.6001.18065 2.09 MB (2,190,848 bytes) 10/6/2008 11:12 AM Microsoft Corporation
e:\windows\winsxs\amd64_microsoft.windows.gdiplus_6595b64144ccf1df_1.0.6001.18065_none_56cd8757b545091c\gdiplus.dll
browseui 6.0.6001.18000 1.58 MB (1,654,784 bytes) 1/18/2008 10:22 PM Microsoft Corporation
e:\windows\system32\browseui.dll
duser 6.0.6001.18000 244.50 KB (250,368 bytes) 1/18/2008 10:09 PM
Microsoft Corporation
e:\windows\system32\duser.dll
WindowsCodecs 6.0.6001.18000 821.50 KB (841,216 bytes)
1/18/2008 10:11 PM Microsoft Corporation
e:\windows\system32\windowscodecs.dll
iconcodecservice 6.0.6000.16386 12.50 KB (12,800 bytes) 1/19/2008 5:51 AM Microsoft Corporation
e:\windows\system32\iconcodecservice.dll
timedate 6.0.6001.18000 860.50 KB (881,152 bytes) 1/18/2008 10:23 PM
Microsoft Corporation
e:\windows\system32\timedate.cpl
ieframe 7.0.6001.18099 6.68 MB (7,003,648 bytes) 10/6/2008 11:12 AM Microsoft Corporation
e:\windows\system32\ieframe.dll
networkexplorer 6.0.6001.18000 2.14 MB (2,247,168 bytes) 1/18/2008 10:35 PM Microsoft Corporation
e:\windows\system32\networkexplorer.dll
cscapi 6.0.6001.18000 38.00 KB (38,912 bytes) 1/18/2008 9:55 PM
Microsoft Corporation
e:\windows\system32\cscapi.dll
ExplorerFrame 6.0.6001.18000 39.00 KB (39,936 bytes) 1/18/2008 10:21 PM Microsoft Corporation
e:\windows\system32\explorerframe.dll
stobject 6.0.6001.18000 731.00 KB (748,544 bytes) 1/18/2008 10:23 PM

© 2009 Hewlett-Packard Company. All rights reserved.

Microsoft Corporation
e:\windows\system32\stobject.dll
batmeter 6.0.6001.18000 727.50 KB (744,960 bytes) 1/18/2008 10:23 PM
Microsoft Corporation
e:\windows\system32\batmeter.dll
SndVolSSO 6.0.6000.16386 173.50 KB (177,664 bytes) 1/18/2008 10:44 PM
Microsoft Corporation
e:\windows\system32\sndvolsso.dll
pnidui 6.0.6001.18000 1.93 MB (2,024,960 bytes) 1/18/2008 10:35 PM Microsoft Corporation
e:\windows\system32\pnidui.dll
wlanutil 6.0.6000.16386 10.00 KB (10,240 bytes) 1/18/2008 10:34 PM
Microsoft Corporation
e:\windows\system32\wlanutil.dll
cscui 6.0.6001.18000 656.50 KB (672,256 bytes) 1/19/2008 5:51 AM
Microsoft Corporation
e:\windows\system32\cscui.dll
cscdll 6.0.6001.18000 28.00 KB (28,672 bytes) 1/18/2008 9:55 PM
Microsoft Corporation
e:\windows\system32\cscdll.dll
srchadmin 6.0.6001.18000 292.00 KB (299,008 bytes) 1/18/2008 10:23 PM
Microsoft Corporation
e:\windows\system32\srchadmin.dll
webcheck 7.0.6001.18000 284.00 KB (290,816 bytes) 1/18/2008 10:26 PM
Microsoft Corporation
e:\windows\system32\webcheck.dll
imapi2 6.0.6001.18000 396.00 KB (405,504 bytes) 1/18/2008 10:29 PM
Microsoft Corporation
e:\windows\system32\imapi2.dll
bthprops 6.0.6001.18000 993.50 KB (1,017,344 bytes) 1/18/2008 10:23 PM Microsoft Corporation
e:\windows\system32\bthprops.cpl
actxprxy 6.0.6001.18000 979.00 KB (1,002,496 bytes) 1/18/2008 11:13 PM Microsoft Corporation
e:\windows\system32\actxprxy.dll
drprov 6.0.6001.18000 23.50 KB (24,064 bytes) 1/18/2008 10:43 PM
Microsoft Corporation
e:\windows\system32\drprov.dll
ntlanman 6.0.6001.18000 116.00 KB (118,784 bytes) 1/18/2008 10:14 PM
Microsoft Corporation
e:\windows\system32\ntlanman.dll
ntshrui 6.0.6001.18000 347.00 KB (355,328 bytes) 1/18/2008 10:24 PM
Microsoft Corporation
e:\windows\system32\ntshrui.dll
thumbcache 6.0.6001.18000 102.00 KB (104,448 bytes) 1/18/2008 10:22 PM Microsoft Corporation
e:\windows\system32\thumbcache.dll
zipfldr 6.0.6001.18000 377.50 KB (386,560 bytes) 1/18/2008 10:23 PM
Microsoft Corporation
e:\windows\system32\zipfldr.dll
sqlservr 2007.100.2531.0 54.95 MB (57,617,752 bytes) 3/30/2009 4:02 AM
Microsoft Corporation e:\program files\microsoft sql server\mssql10.mssqlserver\mssql\binn\sqlservr.exe
sqlos 2007.100.2531.0 14.85 KB (15,208 bytes) 3/30/2009 4:01 AM
Microsoft Corporation e:\program files\microsoft sql server\mssql10.mssqlserver\mssql\binn\sqlos.dll
pdh 6.0.6001.18000 301.50 KB (308,736 bytes) 1/18/2008 10:03 PM
Microsoft Corporation
e:\windows\system32\pdh.dll
opends60 2007.100.1600.22 23.52 KB (24,088 bytes) 7/10/2008 4:46 AM
Microsoft Corporation e:\program files\microsoft sql server\mssql10.mssqlserver\mssql\binn\opends60.dll
batchparser 2007.100.1600.22 170.52 KB (174,616 bytes) 7/10/2008 4:38 AM
Microsoft Corporation e:\program files\microsoft sql server\mssql10.mssqlserver\mssql\binn\batchparser.dll
instapi10 2007.100.1600.22 42.52 KB (43,544 bytes) 7/10/2008 4:39 AM
Microsoft Corporation e:\program files\microsoft sql server\100\shared\instapi10.dll
sqlevn70 2007.100.2531.0 1.99 MB (2,086,760 bytes) 3/30/2009 3:27 AM Microsoft Corporation e:\program files\microsoft sql server\mssql10.mssqlserver\mssql\binn\resources\1033\sqlevn70.rll
mscoree 2.0.50727.3053 397.00 KB (406,528 bytes) 10/4/2008 9:26 PM
Microsoft Corporation
e:\windows\system32\mscoree.dll
security 6.0.6000.16386 5.50 KB (5,632 bytes) 1/18/2008 10:16 PM
Microsoft Corporation
e:\windows\system32\security.dll
msdtc 2001.12.6931.18000 104.00 KB (106,496 bytes) 1/18/2008 10:27 PM Microsoft Corporation
e:\windows\system32\msdtc.exe
msdtctm 2001.12.6931.18000
1.43 MB (1,497,088 bytes)
1/18/2008 10:28 PM Microsoft Corporation
e:\windows\system32\msdtctm.dll
msdtcprx 2001.12.6931.18000
708.50 KB (725,504 bytes)
1/18/2008 10:27 PM Microsoft Corporation
e:\windows\system32\msdtcprx.dll
mtxclu 2001.12.6931.18000 350.50 KB (358,912 bytes) 1/18/2008 10:27 PM Microsoft Corporation
e:\windows\system32\mtxclu.dll
msdtclog 2001.12.6931.18000
113.00 KB (115,712 bytes)
1/18/2008 10:27 PM Microsoft Corporation
e:\windows\system32\msdtclog.dll
xolehlp 2001.12.6931.18000 47.00 KB (48,128 bytes) 1/18/2008 10:27 PM Microsoft Corporation
e:\windows\system32\xolehlp.dll
comres 2001.12.6931.18000 1.23 MB (1,291,264 bytes) 1/18/2008 10:27 PM Microsoft Corporation
e:\windows\system32\comres.dll
msdtcVSp1res 2001.12.6931.18000
20.50 KB (20,992 bytes) 1/18/2008 10:27 PM Microsoft Corporation
e:\windows\system32\msdtcvsp1res.dll
mtxoci 2001.12.6931.18000 148.00 KB (151,552 bytes) 1/18/2008 10:27 PM Microsoft Corporation
e:\windows\system32\mtxoci.dll
tapisrv 6.0.6001.18000 311.00 KB (318,464 bytes) 1/18/2008 11:13 PM
Microsoft Corporation
e:\windows\system32\tapisrv.dll
unimdm 6.0.6001.18000 312.00 KB (319,488 bytes) 1/18/2008 10:38 PM
Microsoft Corporation
e:\windows\system32\unimdm.tsp
uniplat 6.0.6001.18000 21.00 KB (21,504 bytes) 1/18/2008 10:38 PM
Microsoft Corporation
e:\windows\system32\uniplat.dll
kmddsp 6.0.6000.16386 45.50 KB (46,592 bytes) 1/18/2008 10:37 PM
Microsoft Corporation
e:\windows\system32\kmddsp.tsp
ndptsp 6.0.6000.16386 58.00 KB (59,392 bytes) 1/18/2008 10:37 PM
Microsoft Corporation
e:\windows\system32\ndptsp.tsp
hidphone 6.0.6000.16386 38.50 KB (39,424 bytes) 1/18/2008 11:13 PM
Microsoft Corporation
e:\windows\system32\hidphone.tsp
hid 6.0.6001.18000 28.50 KB (29,184 bytes) 1/18/2008 10:33 PM
Microsoft Corporation
e:\windows\system32\hid.dll
rdpclip 6.0.6001.18000 187.00 KB (191,488 bytes) 1/19/2008 5:51 AM
Microsoft Corporation
e:\windows\system32\rdpclip.exe
logonui 6.0.6001.18000 20.50 KB (20,992 bytes) 1/18/2008 10:17 PM
Microsoft Corporation
e:\windows\system32\logonui.exe
authui 6.0.6001.18000 2.17 MB (2,271,744 bytes) 1/18/2008 10:25 PM Microsoft Corporation
e:\windows\system32\authui.dll
SmartcardCredentialProvider
6.0.6001.18000 157.00 KB (160,768 bytes) 1/18/2008 10:15 PM
Microsoft Corporation
e:\windows\system32\smartcardcredentialprovider.dll
rasplap 6.0.6001.18000 380.50 KB (389,632 bytes) 1/18/2008 10:37 PM
Microsoft Corporation
e:\windows\system32\rasplap.dll
msinfo32 6.0.6001.18000 477.50 KB (488,960 bytes) 1/18/2008 10:03 PM
Microsoft Corporation
e:\windows\system32\msinfo32.exe
mfc42u 6.6.8063.0 1.29 MB (1,357,824 bytes) 1/18/2008 11:09 PM
Microsoft Corporation
e:\windows\system32\mfc42u.dll
odbc32 6.0.6001.18000 448.00 KB (458,752 bytes) 1/18/2008 10:56 PM
Microsoft Corporation
e:\windows\system32\odbc32.dll
comdlg32 6.0.6001.18000 536.50 KB (549,376 bytes) 1/18/2008 10:21 PM
Microsoft Corporation
e:\windows\system32\comdlg32.dll
odbcint 6.0.6000.16386 224.00 KB (229,376 bytes) 1/18/2008 10:56 PM
Microsoft Corporation
e:\windows\system32\odbcint.dll

© 2009 Hewlett-Packard Company. All rights reserved.

msshsq 6.0.6001.18000 336.50 KB (344,576 bytes) 1/18/2008 10:58 PM Microsoft Corporation e:\windows\system32\msshsq.dll
wmiprvse 6.0.6001.18000 340.50 KB (348,672 bytes) 1/18/2008 10:13 PM Microsoft Corporation e:\windows\system32\wbem\wmiprvse.exe
cimwin32 6.0.6001.18000 1.99 MB (2,082,304 bytes) 1/18/2008 10:14 PM Microsoft Corporation e:\windows\system32\wbem\cimwin32.dll
framedynos 6.0.6001.18000 275.00 KB (281,600 bytes) 1/18/2008 10:13 PM Microsoft Corporation e:\windows\system32\framedynos.dll
wmi 6.0.6001.18000 5.50 KB (5,632 bytes) 1/18/2008 11:13 PM Microsoft Corporation e:\windows\system32\wmi.dll
ntevt 6.0.6001.18000 250.00 KB (256,000 bytes) 1/18/2008 10:13 PM Microsoft Corporation e:\windows\system32\wbem\ntevt.dll
provthrd 6.0.6001.18000 327.50 KB (335,360 bytes) 1/18/2008 10:13 PM Microsoft Corporation e:\windows\system32\provthrd.dll
msvcirt 7.0.6000.16386 78.50 KB (80,384 bytes) 1/18/2008 9:52 PM Microsoft Corporation e:\windows\system32\msvcirt.dll
wsock32 6.0.6001.18000 18.00 KB (18,432 bytes) 1/18/2008 10:37 PM Microsoft Corporation e:\windows\system32\wsock32.dll
signdrv 6.0.6000.16386 53.50 KB (54,784 bytes) 1/18/2008 10:03 PM Microsoft Corporation e:\windows\system32\signdrv.dll
riched32 6.0.6001.18000 10.50 KB (10,752 bytes) 1/18/2008 10:08 PM Microsoft Corporation e:\windows\system32\riched32.dll
riched20 5.31.23.1228 592.00 KB (606,208 bytes) 1/18/2008 10:08 PM Microsoft Corporation e:\windows\system32\riched20.dll
WmiPerfClass 6.0.6001.18000 134.00 KB (137,216 bytes) 1/18/2008 10:03 PM Microsoft Corporation e:\windows\system32\wbem\wmiperfclass.dll

[Services]

Display Name Name State Start Mode Service Type Path Error Control Start Name Tag ID
Application Experience AeLookupSvc Stopped Manual Share Process e:\windows\system32\svchost.exe -k netsvcs Normal localSystem0
Application Layer Gateway Service ALG Stopped Manual Own Process e:\windows\system32\alg.exe Normal NT AUTHORITY\LocalService 0
Application Information Appinfo Stopped Manual Share Process e:\windows\system32\svchost.exe -k netsvcs Normal LocalSystem 0
Application Management AppMgmt Stopped Manual Share Process e:\windows\system32\svchost.exe -k netsvcs Normal LocalSystem 0
ASTrace ASTrace Stopped Manual Own Process "c:\g-drive\astrace\tim\astrace.exe" Normal LocalSystem 0
Windows Audio Endpoint Builder AudioEndpointBuilder Stopped Manual Share Process e:\windows\system32\svchost.exe -k localsystemnetworkrestricted Normal LocalSystem 0
Windows Audio AudioSrv Stopped Manual Share Process e:\windows\system32\svchost.exe -k localservicenetworkrestricted Normal NT AUTHORITY\LocalService 0
Base Filtering Engine BFE Stopped Manual Share Process e:\windows\system32\svchost.exe -k localservicenonetwork Normal NT AUTHORITY\LocalService 0
Background Intelligent Transfer Service BITS Running Auto Share Process e:\windows\system32\svchost.exe -k netsvcs Normal LocalSystem 0
Computer Browser Browser Stopped Disabled Share Process e:\windows\system32\svchost.exe -k netsvcs Normal LocalSystem 0
Certificate Propagation CertPropSvc Stopped Disabled Share Process e:\windows\system32\svchost.exe -k netsvcs Normal LocalSystem 0
HP Smart Array SAS/SATA Event Notification Service Cissesrv Running Auto Own Process "e:\program files\hp\cissesrv\cissesrv.exe" Normal LocalSystem 0
Microsoft .NET Framework NGEN v2.0.50727_X86 clr_optimization_v2.0.50727_32 Stopped Manual Own Process e:\windows\microsoft.net\framework\v2.0.50727\mscorsvw.exe Ignore LocalSystem 0
Microsoft .NET Framework NGEN v2.0.50727_X64 clr_optimization_v2.0.50727_64 Stopped Manual Own Process e:\windows\microsoft.net\framework64\v2.0.50727\mscorsvw.exe Ignore LocalSystem 0
COM+ System Application COMSysApp Stopped Manual Own Process e:\windows\system32\dllhost.exe /processid:{02d4b3f1-fd88-11d1-960d-00805fc79235} Normal LocalSystem 0
Cryptographic Services CryptSvc Stopped Manual Share Process e:\windows\system32\svchost.exe -k networkservice Normal NT Authority\NetworkService 0
Offline Files CscService Stopped Disabled Share Process e:\windows\system32\svchost.exe -k localsystemnetworkrestricted Normal LocalSystem 0
DCOM Server Process Launcher DcomLaunch Running Auto Share Process e:\windows\system32\svchost.exe -k dcomlaunch Normal LocalSystem 0
DHCP Client Dhcp Running Auto Share Process e:\windows\system32\svchost.exe -k localservicenetworkrestricted Normal NT Authority\LocalService 0
DNS Client Dnscache Running Auto Share Process e:\windows\system32\svchost.exe -k networkservice Normal NT AUTHORITY\NetworkService 0
Wired AutoConfig dot3svc Stopped Manual Share Process e:\windows\system32\svchost.exe -k localsystemnetworkrestricted Normal localSystem0
Diagnostic Policy Service DPS Running Auto Share Process e:\windows\system32\svchost.exe -k localservicenonetwork Normal NT AUTHORITY\LocalService 0
Extensible Authentication Protocol EapHost Stopped Manual Share Process e:\windows\system32\svchost.exe -k netsvcs Normal localSystem0
Windows Event Log EventLog Running Auto Share Process e:\windows\system32\svchost.exe -k localservicenetworkrestricted Normal NT AUTHORITY\LocalService 0
COM+ Event System EventSystem Running Auto Share Process e:\windows\system32\svchost.exe -k localservice Normal NT AUTHORITY\LocalService 0
Microsoft Fibre Channel Platform Registration Service FCRegSvc Stopped Manual Share Process e:\windows\system32\svchost.exe -k localservicenetworkrestricted Normal NT AUTHORITY\LocalService 0
Function Discovery Provider Host fdPHost Stopped Manual Share Process e:\windows\system32\svchost.exe -k localservice Normal NT AUTHORITY\LocalService 0
Function Discovery Resource Publication FDResPub Stopped Manual Share Process e:\windows\system32\svchost.exe -k localservice Normal NT AUTHORITY\LocalService 0
Windows Presentation Foundation Font Cache 3.0.0.0 FontCache3.0.0.0 Stopped Manual Own Process e:\windows\microsoft.net\framework64\v3.0\wpf\presentationfontcache.exe Normal NT Authority\LocalService 0
Group Policy Client gpsvc Running Auto Own Process e:\windows\system32\svchost.exe -k gpsvcgroup Normal LocalSystem 0
Human Interface Device Access hidserv Stopped Manual Share Process e:\windows\system32\svchost.exe -k

© 2009 Hewlett-Packard Company. All rights reserved.

localsystemnetworkrestricted Normal
LocalSystem 0
Health Key and Certificate Management
hkmsvc Stopped Manual Share
Process
e:\windows\system32\svchost.exe -k
netsvcs Normal localSystem0
Windows CardSpace idsvc
Stopped Manual Share Process
"e:\windows\microsoft.net\framewor
k64\v3.0\windows communication
foundation\infocard.exe" Normal
LocalSystem 0
IKE and AuthIP IPsec Keying Modules
IKEEXTStopped Manual Share
Process
e:\windows\system32\svchost.exe -k
netsvcs Normal LocalSystem 0
PnP-X IP Bus Enumerator IPBusEnum
Stopped Disabled Share
Process
e:\windows\system32\svchost.exe -k
localsystemnetworkrestricted Normal
LocalSystem 0
IP Helper iphlpsvc Running
Auto Share Process
e:\windows\system32\svchost.exe -k
netsvcs Normal LocalSystem 0
CNG Key Isolation KeyIso Stopped
Manual Share Process
e:\windows\system32\lsass.exe
Normal LocalSystem 0
KtmRm for Distributed Transaction
Coordinator KtmRm Running Auto
Share Process
e:\windows\system32\svchost.exe -k
networkservice Normal NT
AUTHORITY\NetworkService 0
Server LanmanServer Running
Auto Share Process
e:\windows\system32\svchost.exe -k
netsvcs Normal LocalSystem 0
Workstation LanmanWorkstation
Running Auto Share Process
e:\windows\system32\svchost.exe -k
localservice Normal NT
AUTHORITY\LocalService 0
Link-Layer Topology Discovery Mapper
lltdsvc Stopped Manual Share
Process
e:\windows\system32\svchost.exe -k
localservice Normal NT
AUTHORITY\LocalService 0
TCP/IP NetBIOS Helper lmhosts
Running Auto Share Process
e:\windows\system32\svchost.exe -k
localservicenetworkrestricted Normal
NT AUTHORITY\LocalService 0
Multimedia Class Scheduler MMCSS
Stopped Manual Share Process
e:\windows\system32\svchost.exe -k
netsvcs Normal LocalSystem 0
Windows Firewall MpsSvc Stopped
Manual Share Process
e:\windows\system32\svchost.exe -k
localservicenonetwork Normal NT
Authority\LocalService 0
Distributed Transaction Coordinator
MSDTC Running Auto Own
Process
e:\windows\system32\msdtc.exe
Normal NT
AUTHORITY\NetworkService 0
Microsoft iSCSI Initiator Service
MSiSCSI Stopped Manual
Share Process
e:\windows\system32\svchost.exe -k
netsvcs Normal LocalSystem 0
Windows Installer msiserver
Stopped Manual Own Process
e:\windows\system32\msiexec /v
Normal LocalSystem 0
SQL Server Analysis Services (TEST)
MSOLAP$TEST Stopped
Manual Own Process "e:\program
files\microsoft sql
server\msas10.test\olap\bin\msmdsrv.ex
e" -s "e:\program files\microsoft sql
server\msas10.test\olap\config" Normal
LocalSystem 0
Windows Internal Database
(MICROSOFT##SSEE)
MSSQL$MICROSOFT##SSEE
Stopped Manual Own Process
e:\windows\sysmsi\ssee\mssql.2005
\mssql\binn\sqlservr.exe -
smicrosoft##ssee Normal NT
AUTHORITY\NetworkService 0
SQL Server (MSSQLSERVER)
MSSQLSERVER Stopped
Manual Own Process "e:\program
files\microsoft sql
server\mssql10.mssqlserver\mssql\binn\
sqlservr.exe" -smssqlserver Normal
LocalSystem 0
SQL Active Directory Helper Service
MSSQLServerADHelper100
Stopped Disabled Own
Process "e:\program files\microsoft
sql server\100\shared\sqladhlp.exe"
Normal NT
AUTHORITY\NETWORK SERVICE 0
SQL Server Analysis Services
(MSSQLSERVER)
MSSQLServerOLAPService
Stopped Manual Own Process
"e:\program files\microsoft sql
server\msas10.mssqlserver\olap\bin\ms
mdsrv.exe" -s "e:\program files\microsoft
sql
server\msas10.mssqlserver\olap\config"
Normal LocalSystem 0
Network Access Protection Agent
napagent Stopped Manual
Share Process
e:\windows\system32\svchost.exe -k
networkservice Normal NT
AUTHORITY\NetworkService 0
Netlogon Netlogon Stopped
Manual Share Process
e:\windows\system32\lsass.exe
Normal LocalSystem 0
Network Connections Netman
Running Manual Share Process
e:\windows\system32\svchost.exe -k
localsystemnetworkrestricted Normal
LocalSystem 0
Network List Service netprofm
Running Auto Share Process
e:\windows\system32\svchost.exe -k
localservice Normal NT
AUTHORITY\LocalService 0
Net.Tcp Port Sharing Service
NetTcpPortSharing Stopped
Disabled Share Process
"e:\windows\microsoft.net\framewor
k64\v3.0\windows communication
foundation\smsvchost.exe" Normal NT
AUTHORITY\LocalService 0
Network Location Awareness NlaSvc
Running Auto Share Process
e:\windows\system32\svchost.exe -k
networkservice Normal NT
AUTHORITY\NetworkService 0
Network Store Interface Service nsi
Running Auto Share Process
e:\windows\system32\svchost.exe -k
localservice Normal NT
Authority\LocalService 0
Performance Counter DLL Host
PerfHost Stopped Manual
Own Process
e:\windows\syswow64\perfhost.exe
Normal NT
AUTHORITY\LocalService 0
Performance Logs & Alerts pla
Stopped Manual Share Process
e:\windows\system32\svchost.exe -k
localservicenonetwork Normal NT
AUTHORITY\LocalService 0
Plug and Play PlugPlay Running
Auto Share Process
e:\windows\system32\svchost.exe -k
dcomlaunch Normal LocalSystem 0
IPsec Policy Agent PolicyAgent
Stopped Manual Share Process
e:\windows\system32\svchost.exe -k
networkservicenetworkrestricted Normal
NT Authority\NetworkService 0
User Profile Service ProfSvc Running
Auto Share Process
e:\windows\system32\svchost.exe -k
netsvcs Normal LocalSystem 0
Protected Storage ProtectedStorage
Stopped Manual Share Process
e:\windows\system32\lsass.exe
Normal LocalSystem 0
Remote Access Auto Connection
Manager RasAuto Stopped
Manual Share Process
e:\windows\system32\svchost.exe -k
netsvcs Normal localSystem0
Remote Access Connection Manager
RasMan Running Manual
Share Process
e:\windows\system32\svchost.exe -k
netsvcs Normal localSystem0
Routing and Remote Access
RemoteAccess Stopped
Disabled Share Process
e:\windows\system32\svchost.exe -k
netsvcs Normal localSystem0
Remote Registry RemoteRegistry
Running Auto Share Process
e:\windows\system32\svchost.exe -k
regsvc Normal NT
AUTHORITY\LocalService 0
Remote Procedure Call (RPC) Locator
RpcLocator Stopped Manual
Own Process
e:\windows\system32\locator.exe
Normal NT
AUTHORITY\NetworkService 0
Remote Procedure Call (RPC) RpcSs
Running Auto Share Process
e:\windows\system32\svchost.exe -k
rpcss Normal NT
AUTHORITY\NetworkService 0
Resultant Set of Policy Provider
RSoPProv Stopped Manual
Share Process
e:\windows\system32\rsopprov.exe
Normal LocalSystem 0

© 2009 Hewlett-Packard Company. All rights reserved.

Special Administration Console Helper
sacsvr Stopped Manual Share
Process
e:\windows\system32\svchost.exe -k
netsvcs Normal LocalSystem 0
Security Accounts Manager SamSs
Running Auto Share Process
e:\windows\system32\lsass.exe
Normal LocalSystem 0
Smart Card SCardSvr Stopped
Manual Share Process
e:\windows\system32\svchost.exe -k
localservice Normal NT
AUTHORITY\LocalService 0
Task Scheduler Schedule Running
Auto Share Process
e:\windows\system32\svchost.exe -k
netsvcs Normal LocalSystem 0
Smart Card Removal Policy
SCPolicySvc Stopped Manual
Share Process
e:\windows\system32\svchost.exe -k
netsvcs Normal LocalSystem 0
Secondary Logon seclogon
Running Auto Share Process
e:\windows\system32\svchost.exe -k
netsvcs Normal LocalSystem 0
System Event Notification Service
SENS Running Auto Share
Process
e:\windows\system32\svchost.exe -k
netsvcs Normal LocalSystem 0
Terminal Services Configuration
SessionEnv Running Manual
Share Process
e:\windows\system32\svchost.exe -k
netsvcs Normal localSystem0
Internet Connection Sharing (ICS)
SharedAccess Stopped
Disabled Share Process
e:\windows\system32\svchost.exe -k
netsvcs Normal LocalSystem 0
Shell Hardware Detection
ShellHWDetection Stopped
Disabled Share Process
e:\windows\system32\svchost.exe -k
netsvcs Ignore LocalSystem 0
Software Licensing slsvc Running
Auto Own Process
e:\windows\system32\slsvc.exe
Normal NT
AUTHORITY\NetworkService 0
SL UI Notification Service SLUINotify
Stopped Manual Share Process
e:\windows\system32\svchost.exe -k
localservice Normal NT
AUTHORITY\LocalService 0
SNMP Trap SNMPTRAP Stopped
Manual Own Process
e:\windows\system32\snmptrap.exe
Normal NT
AUTHORITY\LocalService 0
Print Spooler Spooler Stopped
Manual Own Process
e:\windows\system32\spoolsv.exe
Normal LocalSystem 0
SQL Server Browser SQLBrowser
Stopped Manual Own Process
"e:\program files (x86)\microsoft sql
server\90\shared\sqlbrowser.exe"
Normal NT AUTHORITY\LOCAL
SERVICE 0
SQL Server Agent (MSSQLSERVER)
SQLSERVERAGENT Stopped
Manual Own Process "e:\program
files\microsoft sql
server\mssql10.mssqlserver\mssql\binn\
sqlagent.exe" -i mssqlserver Normal
LocalSystem 0
SQL Server VSS Writer SQLWriter
Running Auto Own Process
"e:\program files\microsoft sql
server\90\shared\sqlwriter.exe" Normal
LocalSystem 0
SSDP Discovery SSDPSRV
Stopped Disabled Share
Process
e:\windows\system32\svchost.exe -k
localservice Normal NT
AUTHORITY\LocalService 0
Secure Socket Tunneling Protocol
Service SstpSvc Running Manual
Share Process
e:\windows\system32\svchost.exe -k
localservice Normal NT
Authority\LocalService 0
Microsoft Software Shadow Copy
Provider swprv Stopped Manual
Own Process
e:\windows\system32\svchost.exe -k
swprv Normal LocalSystem 0
HP ProLiant System Shutdown Service
sysdown Stopped Manual
Own Process
e:\windows\system32\sysdown.exe
Normal LocalSystem 0
Superfetch SysMain Stopped
Disabled Share Process
e:\windows\system32\svchost.exe -k
localsystemnetworkrestricted Ignore
LocalSystem 0
Telephony TapiSrv Running Manual
Own Process
e:\windows\system32\svchost.exe -k
tapisrv Normal NT
AUTHORITY\NetworkService 0
TPM Base Services TBS Stopped
Auto Share Process
e:\windows\system32\svchost.exe -k
localservice Normal NT
AUTHORITY\LocalService 0
Terminal Services TermService
Running Auto Share Process
e:\windows\system32\svchost.exe -k
networkservice Normal NT
Authority\NetworkService 0
Themes Themes Stopped
Disabled Share Process
e:\windows\system32\svchost.exe -k
netsvcs Normal LocalSystem 0
Thread Ordering Server
THREADORDER Stopped
Manual Share Process
e:\windows\system32\svchost.exe -k
localservice Normal NT
AUTHORITY\LocalService 0
Distributed Link Tracking Client TrkWks
Running Auto Share Process
e:\windows\system32\svchost.exe -k
localsystemnetworkrestricted Normal
LocalSystem 0
Windows Modules Installer
TrustedInstaller Stopped Manual
Own Process
e:\windows\servicing\trustedinstaller
.exe Normal localSystem0
Interactive Services Detection
UI0Detect Stopped Manual
Own Process
e:\windows\system32\ui0detect.exe
Normal LocalSystem 0
Terminal Services UserMode Port
Redirector UmRdpService Running
Manual Share Process
e:\windows\system32\svchost.exe -k
localsystemnetworkrestricted Normal
localSystem0
UPnP Device Host upnphost
Stopped Disabled Share
Process
e:\windows\system32\svchost.exe -k
localservice Normal NT
AUTHORITY\LocalService 0
Desktop Window Manager Session
Manager UxSms Running Auto
Share Process
e:\windows\system32\svchost.exe -k
localsystemnetworkrestricted Normal
localSystem0
Virtual Disk vds Stopped Manual
Own Process
e:\windows\system32\vds.exe
Normal LocalSystem 0
Volume Shadow Copy VSS
Stopped Manual Own Process
e:\windows\system32\vssvc.exe
Normal LocalSystem 0
Windows Time W32Time Stopped
Manual Share Process
e:\windows\system32\svchost.exe -k
localservice Normal NT
AUTHORITY\LocalService 0
Windows Color System
WcsPlugInService Stopped
Manual Share Process
e:\windows\system32\svchost.exe -k
wcssvc Normal NT
AUTHORITY\LocalService 0
Diagnostic Service Host WdiServiceHost
Stopped Manual Share Process
e:\windows\system32\svchost.exe -k
wdisvc Normal NT
AUTHORITY\LocalService 0
Diagnostic System Host WdiSystemHost
Running Manual Share Process
e:\windows\system32\svchost.exe -k
localsystemnetworkrestricted Normal
LocalSystem 0
Windows Event Collector Wecsvc
Stopped Manual Share Process
e:\windows\system32\svchost.exe -k
networkservice Normal NT
AUTHORITY\NetworkService 0
Problem Reports and Solutions Control
Panel Support wercplsupport
Stopped Manual Share Process
e:\windows\system32\svchost.exe -k
netsvcs Normal localSystem0
Windows Error Reporting Service
WerSvc Stopped Manual Share
Process
e:\windows\system32\svchost.exe -k
wersvcgroup Ignore localSystem0
WinHTTP Web Proxy Auto-Discovery
Service WinHttpAutoProxySvc
Stopped Manual Share Process
e:\windows\system32\svchost.exe -k
localservice Normal NT
AUTHORITY\LocalService 0
Windows Management Instrumentation
Winmgmt Running Auto
Share Process
e:\windows\system32\svchost.exe -k
netsvcs Ignore localSystem0

© 2009 Hewlett-Packard Company. All rights reserved.

Windows Remote Management (WS-Management) WinRM Running
Auto Share Process
e:\windows\system32\svchost.exe -k networkservice Normal NT AUTHORITY\NetworkService 0
WMI Performance Adapter wmiApSrv
Stopped Manual Own Process
e:\windows\system32\wbem\wmiapsrv.exe Normal localSystem0
Portable Device Enumerator Service
WPDBusEnum Stopped Manual
Share Process
e:\windows\system32\svchost.exe -k localsystemnetworkrestricted Normal
LocalSystem 0
Windows System Resource Manager
WSRM Stopped Disabled
Own Process
e:\windows\system32\wsrm.exe
Normal LocalSystem 0
Windows Update wuauserv
Running Auto Share Process
e:\windows\system32\svchost.exe -k netsvcs Normal LocalSystem 0
Windows Driver Foundation - User-mode Driver Framework wudfsvc
Stopped Manual Share Process
e:\windows\system32\svchost.exe -k localsystemnetworkrestricted Normal
LocalSystem 0

[Program Groups]

Group Name Name User Name
Start Menu Default:Start Menu Default
Start Menu\Programs Default:Start Menu\Programs Default
Start Menu\Programs\Accessories
Default:Start Menu\Programs\Accessories Default
Start Menu\Programs\Accessories\Accessibility Default:Start Menu\Programs\Accessories\Accessibility Default
Start Menu\Programs\Accessories\System Tools Default:Start Menu\Programs\Accessories\System Tools Default
Start Menu\Programs\Maintenance
Default:Start Menu\Programs\Maintenance Default
Start Menu Public:Start Menu Public
Start Menu\Programs Public:Start Menu\Programs Public
Start Menu\Programs\Accessories
Public:Start Menu\Programs\Accessories Public
Start Menu\Programs\Accessories\Accessibility Public:Start Menu\Programs\Accessories\Accessibility Public
Start Menu\Programs\Accessories\System Tools Public:Start Menu\Programs\Accessories\System Tools Public
Start Menu\Programs\Administrative Tools Public:Start Menu\Programs\Administrative Tools
Public
Start Menu\Programs\Administrative Tools\Terminal Services Public:Start Menu\Programs\Administrative Tools\Terminal Services Public
Start Menu\Programs\Debugging Tools for Windows (x64) Public:Start Menu\Programs\Debugging Tools for Windows (x64) Public
Start Menu\Programs\Extras and Upgrades Public:Start Menu\Programs\Extras and Upgrades
Public
Start Menu\Programs\HP System Tools
Public:Start Menu\Programs\HP System Tools Public
Start Menu\Programs\HP System Tools\HP Array Configuration Utility
Public:Start Menu\Programs\HP System Tools\HP Array Configuration Utility Public
Start Menu\Programs\HP System Tools\HP Array Diagnostic Utility
Public:Start Menu\Programs\HP System Tools\HP Array Diagnostic Utility Public
Start Menu\Programs\Maintenance
Public:Start Menu\Programs\Maintenance Public
Start Menu\Programs\Microsoft SQL Server 2008 Public:Start Menu\Programs\Microsoft SQL Server 2008 Public
Start Menu\Programs\Microsoft SQL Server 2008\Analysis Services
Public:Start Menu\Programs\Microsoft SQL Server 2008\Analysis Services Public
Start Menu\Programs\Microsoft SQL Server 2008\Configuration Tools
Public:Start Menu\Programs\Microsoft SQL Server 2008\Configuration Tools Public
Start Menu\Programs\Microsoft SQL Server 2008\Integration Services
Public:Start Menu\Programs\Microsoft SQL Server 2008\Integration Services Public
Start Menu\Programs\Microsoft SQL Server 2008\Performance Tools
Public:Start Menu\Programs\Microsoft SQL Server 2008\Performance Tools Public
Start Menu\Programs\Microsoft Visual Studio 2008 Public:Start Menu\Programs\Microsoft Visual Studio 2008 Public
Start Menu\Programs\Microsoft Visual Studio 2008\Visual Studio Tools
Public:Start Menu\Programs\Microsoft Visual Studio 2008\Visual Studio Tools Public
Start Menu\Programs\Startup
Public:Start Menu\Programs\Startup
Public
Start Menu\Programs\Windows PowerShell 1.0 Public:Start Menu\Programs\Windows PowerShell 1.0 Public
Start Menu
OCTANE1\Administrator:Start Menu
OCTANE1\Administrator
Start Menu\Programs
OCTANE1\Administrator:Start Menu\Programs
OCTANE1\Administrator
Start Menu\Programs\Accessories
OCTANE1\Administrator:Start Menu\Programs\Accessories
OCTANE1\Administrator
Start Menu\Programs\Accessories\Accessibility OCTANE1\Administrator:Start Menu\Programs\Accessories\Accessibility OCTANE1\Administrator
Start Menu\Programs\Accessories\System Tools OCTANE1\Administrator:Start Menu\Programs\Accessories\System Tools OCTANE1\Administrator
Start Menu\Programs\Administrative Tools OCTANE1\Administrator:Start Menu\Programs\Administrative Tools
OCTANE1\Administrator
Start Menu\Programs\AMD System Analysis Tools
OCTANE1\Administrator:Start Menu\Programs\AMD System Analysis Tools OCTANE1\Administrator
Start Menu\Programs\AMD System Analysis Tools\CpuSpy
OCTANE1\Administrator:Start Menu\Programs\AMD System Analysis Tools\CpuSpy
OCTANE1\Administrator
Start Menu\Programs\AMD System Analysis Tools\mreport
OCTANE1\Administrator:Start Menu\Programs\AMD System Analysis Tools\mreport
OCTANE1\Administrator
Start Menu\Programs\AMD System Analysis Tools\MultEvent
OCTANE1\Administrator:Start Menu\Programs\AMD System Analysis Tools\MultEvent
OCTANE1\Administrator
Start Menu\Programs\Maintenance
OCTANE1\Administrator:Start Menu\Programs\Maintenance
OCTANE1\Administrator
Start Menu\Programs\Startup
OCTANE1\Administrator:Start Menu\Programs\Startup
OCTANE1\Administrator
Start Menu\Programs\WattsUp USB
OCTANE1\Administrator:Start Menu\Programs\WattsUp USB
OCTANE1\Administrator
Start Menu\Programs\XML Notepad 2007 OCTANE1\Administrator:Start Menu\Programs\XML Notepad 2007
OCTANE1\Administrator

[Startup Programs]

Program Command User Name
Location

[OLE Registration]

Object Local Server
WordPad Document
"%programfiles%\windows nt\accessories\wordpad.exe"
Package Not Available

[Windows Error Reporting]

Time Type Details

© 2009 Hewlett-Packard Company. All rights reserved.

© 2009 Hewlett-Packard Company. All rights reserved.

SQL Server

Engine Version: Microsoft SQL Server 2008 (SP1) - 10.0.2531.0 (X64)

SQL install: The installation followed the default options. For the sort order Latin1_General_binary was chosen. Mixed mode authentication was used.

SQL Startup parameters:

-x Disable the keeping of CPU time and cache-hit ratio statistics

-c Start SQL Server independently of Windows NT Service Control Manager

-E Increase the number of consecutive extents allocated per file to 4

-T2301 Trace flag to enable more accurate query run-time behavior modeling in the SQL Server query optimizer typically only needed for large data set decision support processing.

-T834 On systems with 8GB or more, this trace flag causes the buffer pool to use large pages. These are allocated at startup and are kept throughout the lifetime of the process. This trace flag can only be set on 64-bit installations of SQL Server.

The following parameters were changed from their default values:

name	minimum	maximum	config_value	run_value
affinity mask	-2147483648	2147483648	-1	-1
affintiy64 mask	-2147483648	2147483648	65535	65535
allow updates	0	1	1	1
lightweight pooling	0	1	1	1
max worker threads	128	32767	4096	4096
max server memory (MB)	16	2147483647	248000	248000
min server memory (MB)	0	2147483647	240000	240000
network packet size (B)	512	32767	32767	32767
recovery interval (min)	0	32767	32767	32767
show advanced options	0	1	1	1
query wait	-1	2147483647	2147483647	2147483647

© 2009 Hewlett-Packard Company. All rights reserved.

Appendix B: Database Build Scripts

Create Database

```
if exists (select name from sysdatabases where name = 'tpch300g') drop database tpch300g

CREATE DATABASE tpch300g ON PRIMARY
(NAME = tpch300g_root,
 FILENAME = "G:\TPCH\tpch300g.mdf",
 SIZE = 10MB,
 FILEGROWTH = 10MB),

FILEGROUP Load_FG
(NAME = tpch300g_load1,  FILENAME = 'G:\mnt\ntfs\1\load\load.mdf',  SIZE = 10GB),
(NAME = tpch300g_load2,  FILENAME = 'G:\mnt\ntfs\2\load\load.mdf',  SIZE = 10GB),
(NAME = tpch300g_load3,  FILENAME = 'G:\mnt\ntfs\3\load\load.mdf',  SIZE = 10GB),
(NAME = tpch300g_load4,  FILENAME = 'G:\mnt\ntfs\4\load\load.mdf',  SIZE = 10GB),
(NAME = tpch300g_load5,  FILENAME = 'G:\mnt\ntfs\5\load\load.mdf',  SIZE = 10GB),
(NAME = tpch300g_load6,  FILENAME = 'G:\mnt\ntfs\6\load\load.mdf',  SIZE = 10GB),
(NAME = tpch300g_load7,  FILENAME = 'G:\mnt\ntfs\7\load\load.mdf',  SIZE = 10GB),
(NAME = tpch300g_load8,  FILENAME = 'G:\mnt\ntfs\8\load\load.mdf',  SIZE = 10GB),
(NAME = tpch300g_load9,  FILENAME = 'G:\mnt\ntfs\9\load\load.mdf',  SIZE = 10GB),
(NAME = tpch300g_load10, FILENAME = 'G:\mnt\ntfs\10\load\load.mdf', SIZE = 10GB),
(NAME = tpch300g_load11, FILENAME = 'G:\mnt\ntfs\11\load\load.mdf', SIZE = 10GB),
(NAME = tpch300g_load12, FILENAME = 'G:\mnt\ntfs\12\load\load.mdf', SIZE = 10GB),
(NAME = tpch300g_load13, FILENAME = 'G:\mnt\ntfs\13\load\load.mdf', SIZE = 10GB),
(NAME = tpch300g_load14, FILENAME = 'G:\mnt\ntfs\14\load\load.mdf', SIZE = 10GB),
(NAME = tpch300g_load15, FILENAME = 'G:\mnt\ntfs\15\load\load.mdf', SIZE = 10GB),
(NAME = tpch300g_load16, FILENAME = 'G:\mnt\ntfs\16\load\load.mdf', SIZE = 10GB),
(NAME = tpch300g_load17, FILENAME = 'G:\mnt\ntfs\17\load\load.mdf', SIZE = 10GB),
(NAME = tpch300g_load18, FILENAME = 'G:\mnt\ntfs\18\load\load.mdf', SIZE = 10GB),
(NAME = tpch300g_load19, FILENAME = 'G:\mnt\ntfs\19\load\load.mdf', SIZE = 10GB),
(NAME = tpch300g_load20, FILENAME = 'G:\mnt\ntfs\20\load\load.mdf', SIZE = 10GB),
(NAME = tpch300g_load21, FILENAME = 'G:\mnt\ntfs\21\load\load.mdf', SIZE = 10GB),
(NAME = tpch300g_load22, FILENAME = 'G:\mnt\ntfs\22\load\load.mdf', SIZE = 10GB),
(NAME = tpch300g_load23, FILENAME = 'G:\mnt\ntfs\23\load\load.mdf', SIZE = 10GB),
(NAME = tpch300g_load24, FILENAME = 'G:\mnt\ntfs\24\load\load.mdf', SIZE = 10GB),
(NAME = tpch300g_load25, FILENAME = 'G:\mnt\ntfs\25\load\load.mdf', SIZE = 10GB),
(NAME = tpch300g_load26, FILENAME = 'G:\mnt\ntfs\26\load\load.mdf', SIZE = 10GB),
(NAME = tpch300g_load27, FILENAME = 'G:\mnt\ntfs\27\load\load.mdf', SIZE = 10GB),
(NAME = tpch300g_load28, FILENAME = 'G:\mnt\ntfs\28\load\load.mdf', SIZE = 10GB),
(NAME = tpch300g_load29, FILENAME = 'G:\mnt\ntfs\29\load\load.mdf', SIZE = 10GB),
(NAME = tpch300g_load30, FILENAME = 'G:\mnt\ntfs\30\load\load.mdf', SIZE = 10GB),
(NAME = tpch300g_load31, FILENAME = 'G:\mnt\ntfs\31\load\load.mdf', SIZE = 10GB),
(NAME = tpch300g_load32, FILENAME = 'G:\mnt\ntfs\32\load\load.mdf', SIZE = 10GB),
(NAME = tpch300g_load33, FILENAME = 'G:\mnt\ntfs\33\load\load.mdf', SIZE = 10GB),
(NAME = tpch300g_load34, FILENAME = 'G:\mnt\ntfs\34\load\load.mdf', SIZE = 10GB),
(NAME = tpch300g_load35, FILENAME = 'G:\mnt\ntfs\35\load\load.mdf', SIZE = 10GB),
(NAME = tpch300g_load36, FILENAME = 'G:\mnt\ntfs\36\load\load.mdf', SIZE = 10GB),
(NAME = tpch300g_load37, FILENAME = 'G:\mnt\ntfs\37\load\load.mdf', SIZE = 10GB),
(NAME = tpch300g_load38, FILENAME = 'G:\mnt\ntfs\38\load\load.mdf', SIZE = 10GB),
(NAME = tpch300g_load39, FILENAME = 'G:\mnt\ntfs\39\load\load.mdf', SIZE = 10GB),
(NAME = tpch300g_load40, FILENAME = 'G:\mnt\ntfs\40\load\load.mdf', SIZE = 10GB),
(NAME = tpch300g_load41, FILENAME = 'G:\mnt\ntfs\41\load\load.mdf', SIZE = 10GB),
(NAME = tpch300g_load42, FILENAME = 'G:\mnt\ntfs\42\load\load.mdf', SIZE = 10GB),
(NAME = tpch300g_load43, FILENAME = 'G:\mnt\ntfs\43\load\load.mdf', SIZE = 10GB),
(NAME = tpch300g_load44, FILENAME = 'G:\mnt\ntfs\44\load\load.mdf', SIZE = 10GB),
(NAME = tpch300g_load45, FILENAME = 'G:\mnt\ntfs\45\load\load.mdf', SIZE = 10GB),
(NAME = tpch300g_load46, FILENAME = 'G:\mnt\ntfs\46\load\load.mdf', SIZE = 10GB),
(NAME = tpch300g_load47, FILENAME = 'G:\mnt\ntfs\47\load\load.mdf', SIZE = 10GB),
(NAME = tpch300g_load48, FILENAME = 'G:\mnt\ntfs\48\load\load.mdf', SIZE = 10GB),

FILEGROUP LINEITEM_FG

(NAME = tpch300g_li1,  FILENAME = 'G:\mnt\li\1\',  SIZE = 4850MB),
(NAME = tpch300g_li2,  FILENAME = 'G:\mnt\li\2\',  SIZE = 4850MB),
(NAME = tpch300g_li3,  FILENAME = 'G:\mnt\li\3\',  SIZE = 4850MB),
(NAME = tpch300g_li4,  FILENAME = 'G:\mnt\li\4\',  SIZE = 4850MB),
(NAME = tpch300g_li5,  FILENAME = 'G:\mnt\li\5\',  SIZE = 4850MB),
(NAME = tpch300g_li6,  FILENAME = 'G:\mnt\li\6\',  SIZE = 4850MB),
(NAME = tpch300g_li7,  FILENAME = 'G:\mnt\li\7\',  SIZE = 4850MB),
(NAME = tpch300g_li8,  FILENAME = 'G:\mnt\li\8\',  SIZE = 4850MB),
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
(NAME = tpch300g_li9,  FILENAME = 'G:\mnt\li\9\',  SIZE = 4850MB),
(NAME = tpch300g_li10, FILENAME = 'G:\mnt\li\10\', SIZE = 4850MB),
(NAME = tpch300g_li11, FILENAME = 'G:\mnt\li\11\', SIZE = 4850MB),
(NAME = tpch300g_li12, FILENAME = 'G:\mnt\li\12\', SIZE = 4850MB),
(NAME = tpch300g_li13, FILENAME = 'G:\mnt\li\13\', SIZE = 4850MB),
(NAME = tpch300g_li14, FILENAME = 'G:\mnt\li\14\', SIZE = 4850MB),
(NAME = tpch300g_li15, FILENAME = 'G:\mnt\li\15\', SIZE = 4850MB),
(NAME = tpch300g_li16, FILENAME = 'G:\mnt\li\16\', SIZE = 4850MB),
(NAME = tpch300g_li17, FILENAME = 'G:\mnt\li\17\', SIZE = 4850MB),
(NAME = tpch300g_li18, FILENAME = 'G:\mnt\li\18\', SIZE = 4850MB),
(NAME = tpch300g_li19, FILENAME = 'G:\mnt\li\19\', SIZE = 4850MB),
(NAME = tpch300g_li20, FILENAME = 'G:\mnt\li\20\', SIZE = 4850MB),
(NAME = tpch300g_li21, FILENAME = 'G:\mnt\li\21\', SIZE = 4850MB),
(NAME = tpch300g_li22, FILENAME = 'G:\mnt\li\22\', SIZE = 4850MB),
(NAME = tpch300g_li23, FILENAME = 'G:\mnt\li\23\', SIZE = 4850MB),
(NAME = tpch300g_li24, FILENAME = 'G:\mnt\li\24\', SIZE = 4850MB),
(NAME = tpch300g_li25, FILENAME = 'G:\mnt\li\25\', SIZE = 4850MB),
(NAME = tpch300g_li26, FILENAME = 'G:\mnt\li\26\', SIZE = 4850MB),
(NAME = tpch300g_li27, FILENAME = 'G:\mnt\li\27\', SIZE = 4850MB),
(NAME = tpch300g_li28, FILENAME = 'G:\mnt\li\28\', SIZE = 4850MB),
(NAME = tpch300g_li29, FILENAME = 'G:\mnt\li\29\', SIZE = 4850MB),
(NAME = tpch300g_li30, FILENAME = 'G:\mnt\li\30\', SIZE = 4850MB),
(NAME = tpch300g_li31, FILENAME = 'G:\mnt\li\31\', SIZE = 4850MB),
(NAME = tpch300g_li32, FILENAME = 'G:\mnt\li\32\', SIZE = 4850MB),
(NAME = tpch300g_li33, FILENAME = 'G:\mnt\li\33\', SIZE = 4850MB),
(NAME = tpch300g_li34, FILENAME = 'G:\mnt\li\34\', SIZE = 4850MB),
(NAME = tpch300g_li35, FILENAME = 'G:\mnt\li\35\', SIZE = 4850MB),
(NAME = tpch300g_li36, FILENAME = 'G:\mnt\li\36\', SIZE = 4850MB),
(NAME = tpch300g_li37, FILENAME = 'G:\mnt\li\37\', SIZE = 4850MB),
(NAME = tpch300g_li38, FILENAME = 'G:\mnt\li\38\', SIZE = 4850MB),
(NAME = tpch300g_li39, FILENAME = 'G:\mnt\li\39\', SIZE = 4850MB),
(NAME = tpch300g_li40, FILENAME = 'G:\mnt\li\40\', SIZE = 4850MB),
(NAME = tpch300g_li41, FILENAME = 'G:\mnt\li\41\', SIZE = 4850MB),
(NAME = tpch300g_li42, FILENAME = 'G:\mnt\li\42\', SIZE = 4850MB),
(NAME = tpch300g_li43, FILENAME = 'G:\mnt\li\43\', SIZE = 4850MB),
(NAME = tpch300g_li44, FILENAME = 'G:\mnt\li\44\', SIZE = 4850MB),
(NAME = tpch300g_li45, FILENAME = 'G:\mnt\li\45\', SIZE = 4850MB),
(NAME = tpch300g_li46, FILENAME = 'G:\mnt\li\46\', SIZE = 4850MB),
(NAME = tpch300g_li47, FILENAME = 'G:\mnt\li\47\', SIZE = 4850MB),
(NAME = tpch300g_li48, FILENAME = 'G:\mnt\li\48\', SIZE = 4850MB),

FILEGROUP     GENERAL_FG
(NAME = tpch300g_gen1,  FILENAME = 'G:\mnt\gen\1\',  SIZE = 2400MB),
(NAME = tpch300g_gen2,  FILENAME = 'G:\mnt\gen\2\',  SIZE = 2400MB),
(NAME = tpch300g_gen3,  FILENAME = 'G:\mnt\gen\3\',  SIZE = 2400MB),
(NAME = tpch300g_gen4,  FILENAME = 'G:\mnt\gen\4\',  SIZE = 2400MB),
(NAME = tpch300g_gen5,  FILENAME = 'G:\mnt\gen\5\',  SIZE = 2400MB),
(NAME = tpch300g_gen6,  FILENAME = 'G:\mnt\gen\6\',  SIZE = 2400MB),
(NAME = tpch300g_gen7,  FILENAME = 'G:\mnt\gen\7\',  SIZE = 2400MB),
(NAME = tpch300g_gen8,  FILENAME = 'G:\mnt\gen\8\',  SIZE = 2400MB),
(NAME = tpch300g_gen9,  FILENAME = 'G:\mnt\gen\9\',  SIZE = 2400MB),
(NAME = tpch300g_gen10, FILENAME = 'G:\mnt\gen\10\', SIZE = 2400MB),
(NAME = tpch300g_gen11, FILENAME = 'G:\mnt\gen\11\', SIZE = 2400MB),
(NAME = tpch300g_gen12, FILENAME = 'G:\mnt\gen\12\', SIZE = 2400MB),
(NAME = tpch300g_gen13, FILENAME = 'G:\mnt\gen\13\', SIZE = 2400MB),
(NAME = tpch300g_gen14, FILENAME = 'G:\mnt\gen\14\', SIZE = 2400MB),
(NAME = tpch300g_gen15, FILENAME = 'G:\mnt\gen\15\', SIZE = 2400MB),
(NAME = tpch300g_gen16, FILENAME = 'G:\mnt\gen\16\', SIZE = 2400MB),
(NAME = tpch300g_gen17, FILENAME = 'G:\mnt\gen\17\', SIZE = 2400MB),
(NAME = tpch300g_gen18, FILENAME = 'G:\mnt\gen\18\', SIZE = 2400MB),
(NAME = tpch300g_gen19, FILENAME = 'G:\mnt\gen\19\', SIZE = 2400MB),
(NAME = tpch300g_gen20, FILENAME = 'G:\mnt\gen\20\', SIZE = 2400MB),
(NAME = tpch300g_gen21, FILENAME = 'G:\mnt\gen\21\', SIZE = 2400MB),
(NAME = tpch300g_gen22, FILENAME = 'G:\mnt\gen\22\', SIZE = 2400MB),
(NAME = tpch300g_gen23, FILENAME = 'G:\mnt\gen\23\', SIZE = 2400MB),
(NAME = tpch300g_gen24, FILENAME = 'G:\mnt\gen\24\', SIZE = 2400MB),
(NAME = tpch300g_gen25, FILENAME = 'G:\mnt\gen\25\', SIZE = 2400MB),
(NAME = tpch300g_gen26, FILENAME = 'G:\mnt\gen\26\', SIZE = 2400MB),
(NAME = tpch300g_gen27, FILENAME = 'G:\mnt\gen\27\', SIZE = 2400MB),
(NAME = tpch300g_gen28, FILENAME = 'G:\mnt\gen\28\', SIZE = 2400MB),
(NAME = tpch300g_gen29, FILENAME = 'G:\mnt\gen\29\', SIZE = 2400MB),
(NAME = tpch300g_gen30, FILENAME = 'G:\mnt\gen\30\', SIZE = 2400MB),
(NAME = tpch300g_gen31, FILENAME = 'G:\mnt\gen\31\', SIZE = 2400MB),
(NAME = tpch300g_gen32, FILENAME = 'G:\mnt\gen\32\', SIZE = 2400MB),
(NAME = tpch300g_gen33, FILENAME = 'G:\mnt\gen\33\', SIZE = 2400MB),
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
(NAME = tpch300g_gen34, FILENAME = 'G:\mnt\gen\34\', SIZE = 2400MB),
(NAME = tpch300g_gen35, FILENAME = 'G:\mnt\gen\35\', SIZE = 2400MB),
(NAME = tpch300g_gen36, FILENAME = 'G:\mnt\gen\36\', SIZE = 2400MB),
(NAME = tpch300g_gen37, FILENAME = 'G:\mnt\gen\37\', SIZE = 2400MB),
(NAME = tpch300g_gen38, FILENAME = 'G:\mnt\gen\38\', SIZE = 2400MB),
(NAME = tpch300g_gen39, FILENAME = 'G:\mnt\gen\39\', SIZE = 2400MB),
(NAME = tpch300g_gen40, FILENAME = 'G:\mnt\gen\40\', SIZE = 2400MB),
(NAME = tpch300g_gen41, FILENAME = 'G:\mnt\gen\41\', SIZE = 2400MB),
(NAME = tpch300g_gen42, FILENAME = 'G:\mnt\gen\42\', SIZE = 2400MB),
(NAME = tpch300g_gen43, FILENAME = 'G:\mnt\gen\43\', SIZE = 2400MB),
(NAME = tpch300g_gen44, FILENAME = 'G:\mnt\gen\44\', SIZE = 2400MB),
(NAME = tpch300g_gen45, FILENAME = 'G:\mnt\gen\45\', SIZE = 2400MB),
(NAME = tpch300g_gen46, FILENAME = 'G:\mnt\gen\46\', SIZE = 2400MB),
(NAME = tpch300g_gen47, FILENAME = 'G:\mnt\gen\47\', SIZE = 2400MB),
(NAME = tpch300g_gen48, FILENAME = 'G:\mnt\gen\48\', SIZE = 2400MB)

LOG ON
(NAME = tpch300g_log1, FILENAME = "L:\\", SIZE = 40GB)

COLLATE Latin1_General_BIN
```

Create Base Tables

```
create table LINEITEM
    (L_ORDERKEY      int          not null,
     L_PARTKEY       int          not null,
     L_SUPPKEY       int          not null,
     L_LINENUMBER    int          not null,
     L_QUANTITY      float        not null,
     L_EXTENDEDPRICE float        not null,
     L_DISCOUNT      float        not null,
     L_TAX           float        not null,
     L_RETURNFLAG    char(1)      not null,
     L_LINESTATUS    char(1)      not null,
     L_SHIPDATE      date         not null,
     L_COMMITDATE    date         not null,
     L_RECEIPTDATE   date         not null,
     L_SHIPINSTRUCT  char(25)     not null,
     L_SHIPMODE      char(10)     not null,
     L_COMMENT       varchar(44) not null)
on STAGING_FG
with (DATA_COMPRESSION = PAGE);

create table ORDERS
    (O_ORDERKEY      int          not null,
     O_CUSTKEY       int          not null,
     O_ORDERSTATUS   char(1)      not null,
     O_TOTALPRICE    float        not null,
     O_ORDERDATE     date         not null,
     O_ORDERPRIORITY char(15)     not null,
     O_CLERK         char(15)     not null,
     O_SHIPPRIORITY  int          not null,
     O_COMMENT       varchar(79) not null)
on STAGING_FG
with (DATA_COMPRESSION = PAGE);

create table PART
    (P_PARTKEY       int          not null,
     P_NAME          varchar(55) not null,
     P_MFGR          char(25)     not null,
     P_BRAND         char(10)     not null,
     P_TYPE          varchar(25) not null,
     P_SIZE          int          not null,
     P_CONTAINER     char(10)     not null,
     P_RETAILPRICE   float        not null,
     P_COMMENT       varchar(23) not null)
on STAGING_FG
with (DATA_COMPRESSION = PAGE);
```

```
create table CUSTOMER
    (C_CUSTKEY       int          not null,
     C_NAME          varchar(25)  not null,
     C_ADDRESS       varchar(40)  not null,
     C_NATIONKEY     int          not null,
     C_PHONE         char(15)     not null,
     C_ACCTBAL       float        not null,
     C_MKTSEGMENT    char(10)     not null,
     C_COMMENT       varchar(117) not null)
on STAGING_FG
with (DATA_COMPRESSION = PAGE);

create table SUPPLIER
    (S_SUPPKEY       int          not null,
     S_NAME          char(25)     not null,
     S_ADDRESS       varchar(40)  not null,
     S_NATIONKEY     int          not null,
     S_PHONE         char(15)     not null,
     S_ACCTBAL       float        not null,
     S_COMMENT       varchar(101) not null)
on STAGING_FG
with (DATA_COMPRESSION = PAGE);

create table PARTSUPP
    (PS_PARTKEY      int          not null,
     PS_SUPPKEY      int          not null,
     PS_AVAILQTY     int          not null,
     PS_SUPPLYCOST   float        not null,
     PS_COMMENT      varchar(199) not null)
on STAGING_FG
with (DATA_COMPRESSION = PAGE);

create table NATION
    (N_NATIONKEY     int          not null,
     N_NAME          char(25)     not null,
     N_REGIONKEY     int          not null,
     N_COMMENT       varchar(152) not null)
on STAGING_FG
with (DATA_COMPRESSION = PAGE);

create table REGION
    (R_REGIONKEY     int          not null,
     R_NAME          char(25)     not null,
     R_COMMENT       varchar(152) not null)
on STAGING_FG
with (DATA_COMPRESSION = PAGE);
```


© 2009 Hewlett-Packard Company. All rights reserved.

Indexes

```
create clustered index L_SHIPDATE_CLUIDX
on LINEITEM(L_SHIPDATE)
with (FILLFACTOR=95, SORT_IN_TEMPDB = ON, MAXDOP = 1)
on LINEITEM_FG

create clustered index O_ORDERDATE_CLUIDX
on ORDERS(O_ORDERDATE)
with (FILLFACTOR=95, SORT_IN_TEMPDB = ON, MAXDOP = 1)
on GENERAL_FG

create index L_ORDERKEY_IDX
on LINEITEM(L_ORDERKEY)
with (FILLFACTOR=95,SORT_IN_TEMPDB = ON,
DATA_COMPRESSION = PAGE) on LINEITEM_FG

create index L_PARTKEY_IDX
on LINEITEM(L_PARTKEY)
with (FILLFACTOR=95, SORT_IN_TEMPDB = ON,
DATA_COMPRESSION = PAGE) on LINEITEM_FG

create index PS_SUPPKEY_IDX
on PARTSUPP(PS_SUPPKEY)
with (fillfactor=100, SORT_IN_TEMPDB = ON,
DATA_COMPRESSION = PAGE) on GENERAL_FG

create index N_REGIONKEY_IDX
on NATION(N_REGIONKEY)
with (fillfactor=100, SORT_IN_TEMPDB = ON,
DATA_COMPRESSION = PAGE) on GENERAL_FG

create index S_NATIONKEY_IDX
on SUPPLIER(S_NATIONKEY)
with (fillfactor=100, SORT_IN_TEMPDB = ON,
DATA_COMPRESSION = PAGE) on GENERAL_FG
```

Foreign Keys

```
alter table SUPPLIER add constraint FK_S_NATIONKEY foreign key (S_NATIONKEY) references NATION(N_NATIONKEY)
alter table PARTSUPP add constraint FK_PS_PARTKEY foreign key (PS_PARTKEY) references PART(P_PARTKEY)
alter table PARTSUPP add constraint FK_PS_SUPPKEY foreign key (PS_SUPPKEY) references SUPPLIER(S_SUPPKEY)
alter table CUSTOMER add constraint FK_C_NATIONKEY foreign key (C_NATIONKEY) references NATION (N_NATIONKEY)
alter table ORDERS add constraint FK_O_CUSTKEY foreign key (O_CUSTKEY) references CUSTOMER (C_CUSTKEY)
alter table LINEITEM add constraint FK_L_ORDERKEY foreign key (L_ORDERKEY)  references ORDERS (O_ORDERKEY)
alter table LINEITEM add constraint FK_L_PARTKEY foreign key (L_PARTKEY) references PART (P_PARTKEY)
alter table LINEITEM add constraint FK_L_SUPPKEY foreign key (L_SUPPKEY) references SUPPLIER (S_SUPPKEY)
alter table LINEITEM add constraint FK_L_PARTKEY_SUPPKEY foreign key (L_PARTKEY,L_SUPPKEY) references
PARTSUPP(PS_PARTKEY,PS_SUPPKEY)
alter table NATION add constraint FK_N_REGIONKEY foreign key (N_REGIONKEY)  references REGION (R_REGIONKEY)
```

Primary Keys

```
ALTER table NATION ADD constraint PK_N_NATIONKEY primary key (N_NATIONKEY) ON GENERAL_FG
ALTER table REGION ADD constraint PK_R_REGIONKEY primary key (R_REGIONKEY) ON GENERAL_FG
ALTER table SUPPLIER ADD constraint PK_S_SUPPKEY primary key (S_SUPPKEY) ON GENERAL_FG
ALTER table CUSTOMER ADD constraint PK_C_CUSTKEY primary key (C_CUSTKEY) ON GENERAL_FG
ALTER table ORDERS ADD constraint PK_O_ORDERKEY primary key (O_ORDERKEY) WITH (FILLFACTOR = 95) ON GENERAL_FG
ALTER table PARTSUPP ADD constraint PK_PS_PARTKEY_PS_SUPPKEY primary key (PS_PARTKEY, PS_SUPPKEY) ON
GENERAL_FG
ALTER table PART ADD constraint PK_P_PARTKEY primary key (P_PARTKEY) ON GENERAL_FG
```

Backup

```
backup database tpch300g to
DISK = 'G:\mnt\ntfs\1\backup\tpch300g_audit.bak',DISK = 'G:\mnt\ntfs\2\backup\tpch300g_audit.bak',
DISK = 'G:\mnt\ntfs\3\backup\tpch300g_audit.bak',DISK = 'G:\mnt\ntfs\4\backup\tpch300g_audit.bak',
DISK = 'G:\mnt\ntfs\5\backup\tpch300g_audit.bak',DISK = 'G:\mnt\ntfs\6\backup\tpch300g_audit.bak',
DISK = 'G:\mnt\ntfs\7\backup\tpch300g_audit.bak',DISK = 'G:\mnt\ntfs\8\backup\tpch300g_audit.bak',
DISK = 'G:\mnt\ntfs\9\backup\tpch300g_audit.bak',DISK = 'G:\mnt\ntfs\10\backup\tpch300g_audit.bak,
DISK = 'G:\mnt\ntfs\11\backup\tpch300g_audit.bak,DISK = 'G:\mnt\ntfs\12\backup\tpch300g_audit.bak,
DISK = 'G:\mnt\ntfs\13\backup\tpch300g_audit.bak',DISK = 'G:\mnt\ntfs\14\backup\tpch300g_audit.bak',
DISK = 'G:\mnt\ntfs\15\backup\tpch300g_audit.bak',DISK = 'G:\mnt\ntfs\16\backup\tpch300g_audit.bak',
DISK = 'G:\mnt\ntfs\17\backup\tpch300g_audit.bak',DISK = 'G:\mnt\ntfs\18\backup\tpch300g_audit.bak',
DISK = 'G:\mnt\ntfs\19\backup\tpch300g_audit.bak',DISK = 'G:\mnt\ntfs\20\backup\tpch300g_audit.bak',
DISK = 'G:\mnt\ntfs\21\backup\tpch300g_audit.bak',DISK = 'G:\mnt\ntfs\22\backup\tpch300g_audit.bak',
DISK = 'G:\mnt\ntfs\23\backup\tpch300g_audit.bak',DISK = 'G:\mnt\ntfs\24\backup\tpch300g_audit.bak',
DISK = 'G:\mnt\ntfs\25\backup\tpch300g_audit.bak',DISK = 'G:\mnt\ntfs\26\backup\tpch300g_audit.bak',
DISK = 'G:\mnt\ntfs\27\backup\tpch300g_audit.bak',DISK = 'G:\mnt\ntfs\28\backup\tpch300g_audit.bak',
DISK = 'G:\mnt\ntfs\29\backup\tpch300g_audit.bak',DISK = 'G:\mnt\ntfs\30\backup\tpch300g_audit.bak',
DISK = 'G:\mnt\ntfs\31\backup\tpch300g_audit.bak',DISK = 'G:\mnt\ntfs\32\backup\tpch300g_audit.bak',
DISK = 'G:\mnt\ntfs\33\backup\tpch300g_audit.bak',DISK = 'G:\mnt\ntfs\34\backup\tpch300g_audit.bak',
DISK = 'G:\mnt\ntfs\35\backup\tpch300g_audit.bak',DISK = 'G:\mnt\ntfs\36\backup\tpch300g_audit.bak',
DISK = 'G:\mnt\ntfs\37\backup\tpch300g_audit.bak',DISK = 'G:\mnt\ntfs\38\backup\tpch300g_audit.bak',
DISK = 'G:\mnt\ntfs\39\backup\tpch300g_audit.bak',DISK = 'G:\mnt\ntfs\40\backup\tpch300g_audit.bak',
DISK = 'G:\mnt\ntfs\41\backup\tpch300g_audit.bak',DISK = 'G:\mnt\ntfs\42\backup\tpch300g_audit.bak',
DISK = 'G:\mnt\ntfs\43\backup\tpch300g_audit.bak',DISK = 'G:\mnt\ntfs\44\backup\tpch300g_audit.bak',
DISK = 'G:\mnt\ntfs\45\backup\tpch300g_audit.bak',DISK = 'G:\mnt\ntfs\46\backup\tpch300g_audit.bak',
DISK = 'G:\mnt\ntfs\47\backup\tpch300g_audit.bak',DISK = 'G:\mnt\ntfs\48\backup\tpch300g_audit.bak'
with init, compression, maxtransfersize=1048576, BUFFERCOUNT=1000, stats = 1
```


© 2009 Hewlett-Packard Company. All rights reserved.

Appendix C: Query Text and Output

Using Standard Substitutions

/* TPC_H Query 1 - Pricing Summary Report */

```
SELECT L_RETURNFLAG, L_LINESTATUS, SUM(L_QUANTITY) AS SUM_QTY, SUM(L_EXTENDEDPRICE)
    AS SUM_BASE_PRICE, SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT)) AS SUM_DISC_PRICE,
    SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT)*(1+L_TAX)) AS SUM_CHARGE, AVG(L_QUANTITY) AS AVG_QTY,
    AVG(L_EXTENDEDPRICE) AS AVG_PRICE, AVG(L_DISCOUNT) AS AVG_DISC, COUNT(*) AS COUNT_ORDER
FROM LINEITEM
WHERE  L_SHIPDATE <= dateadd(dd, -90, cast('1998-12-01' as date))
GROUP  BY L_RETURNFLAG, L_LINESTATUS
ORDER  BY L_RETURNFLAG, L_LINESTATUS

L_RETURNFLAG L_LINESTATUS SUM_QTY              SUM_BASE_PRICE         SUM_DISC_PRICE         SUM_CHARGE
AVG_QTY              AVG_PRICE              AVG_DISC               COUNT_ORDER
------------ ------------ ---------------------- ---------------------- ---------------------- ---------------------- ---------------------- ---------------
---------- -----------
A            F            37734107.000000        56586554400.729401     53758257134.869392     55909065222.827774     25.522006
38273.129735           0.049985               1478493
N            F            991417.000000          1487504710.379998      1413082168.054106      1469649223.194383      25.516472
38284.467761           0.050093               38854
N            O            74476040.000000        111701729697.741550    106118230307.607930    110367043872.496800    25.502227
38249.117989           0.049997               2920374
R            F            37719753.000000        56568041380.899246     53741292684.604187     55889619119.831680     25.505794
38250.854626           0.050009               1478870
(4 row(s) affected)
```

/* TPC_H Query 2 - Minimum Cost Supplier */

```
SELECT TOP 100 S_ACCTBAL, S_NAME, N_NAME, P_PARTKEY, P_MFGR, S_ADDRESS, S_PHONE, S_COMMENT
FROM PART, SUPPLIER, PARTSUPP,NATION, REGION
WHERE P_PARTKEY = PS_PARTKEY AND S_SUPPKEY = PS_SUPPKEY AND P_SIZE = 15 AND
    P_TYPE LIKE '%%BRASS' AND S_NATIONKEY = N_NATIONKEY AND N_REGIONKEY = R_REGIONKEY AND
    R_NAME = 'EUROPE' AND
    PS_SUPPLYCOST = (SELECT MIN(PS_SUPPLYCOST) FROM PARTSUPP, SUPPLIER, NATION, REGION
    WHERE P_PARTKEY = PS_PARTKEY AND S_SUPPKEY = PS_SUPPKEY
    AND S_NATIONKEY= N_NATIONKEY AND N_REGIONKEY = R_REGIONKEY AND R_NAME = 'EUROPE')
ORDER BY S_ACCTBAL DESC, N_NAME, S_NAME, P_PARTKEY

S_ACCTBAL              S_NAME                 N_NAME                 P_PARTKEY   P_MFGR                 S_ADDRESS
S_PHONE        S_COMMENT
------------------------ ------------------------ ------------------------ ----------- ------------------------ ---------------------------------------- ----------------
 ----------------------------------------------------------------------------------------------------------------

9938.530000            Supplier#000005359     UNITED KINGDOM         185358      Manufacturer#4         QKuHYh,vZGiwu2FWEJoLDx04
33-429-790-6131 uriously regular requests hag
9937.840000            Supplier#000005969     ROMANIA                108438      Manufacturer#1
ANDENSOSmk,miq23Xfb5RWt6dvUcvt6Qa      29-520-692-3537 efully express instructions. regular requests against the slyly fin
9936.220000            Supplier#000005250     UNITED KINGDOM         249         Manufacturer#4         B3rqp0xbSEim4Mpy2RH J
33-320-228-2957 etect about the furiously final accounts. slyly ironic pinto beans sleep inside the furiously
9923.770000            Supplier#000002324     GERMANY                29821       Manufacturer#4         y3OD9UywSTOk
17-779-299-1839 ackages boost blithely. blithely regular deposits c
.
.
.
7850.660000            Supplier#000001518     UNITED KINGDOM         86501       Manufacturer#1         ONda3YJiHKJOC
33-730-383-3892 ifts haggle fluffily pending pai
7843.520000            Supplier#000006683     FRANCE                 11680       Manufacturer#4         2Z0JGkiv01Y00oCFwUGfviIbhzCdy
16-464-517-8943  express, final pinto beans x-ray slyly asymptotes. unusual, unusual


(100 row(s) affected)
```


© 2009 Hewlett-Packard Company. All rights reserved.

/* TPC_H Query 3 - Shipping Priority */

```
SELECT TOP 10 L_ORDERKEY, SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT))AS REVENUE, O_ORDERDATE, O_SHIPPRIORITY
FROM CUSTOMER, ORDERS, LINEITEM
WHERE C_MKTSEGMENT = 'BUILDING' AND C_CUSTKEY = O_CUSTKEY AND L_ORDERKEY = O_ORDERKEY AND
    O_ORDERDATE < '1995-03-15' AND L_SHIPDATE > '1995-03-15'
GROUP BY L_ORDERKEY, O_ORDERDATE, O_SHIPPRIORITY
ORDER BY REVENUE DESC, O_ORDERDATE
```

L_ORDERKEY	REVENUE	O_ORDERDATE	O_SHIPPRIORITY
2456423	406181.011100	1995-03-05 00:00:00.000	0
3459808	405838.698900	1995-03-04 00:00:00.000	0
492164	390324.061000	1995-02-19 00:00:00.000	0
1188320	384537.935900	1995-03-09 00:00:00.000	0
2435712	378673.055800	1995-02-26 00:00:00.000	0
4878020	378376.795200	1995-03-12 00:00:00.000	0
5521732	375153.921500	1995-03-13 00:00:00.000	0
2628192	373133.309400	1995-02-22 00:00:00.000	0
993600	371407.459500	1995-03-05 00:00:00.000	0
2300070	367371.145200	1995-03-13 00:00:00.000	0

(10 row(s) affected)

/* TPC_H Query 4 - Order Priority Checking */

```
SELECT    O_ORDERPRIORITY, COUNT(*) AS ORDER_COUNT  FROM ORDERS
    WHERE O_ORDERDATE >= '1993-07-01' AND O_ORDERDATE < dateadd (mm, 3, cast ('1993-07-01' as date))
    AND EXISTS (SELECT * FROMLINEITEM WHERE L_ORDERKEY = O_ORDERKEY
    AND L_COMMITDATE < L_RECEIPTDATE)
GROUP BY O_ORDERPRIORITY
ORDER BY O_ORDERPRIORITY
```

O_ORDERPRIORITY	ORDER_COUNT
1-URGENT	10594
2-HIGH	10476
3-MEDIUM	10410
4-NOT SPECIFIED	10556
5-LOW	10487

(5 row(s) affected)

/* TPC_H Query 5 - Local Supplier Volume */

```
SELECT    N_NAME, SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT)) AS REVENUE
    FROM CUSTOMER, ORDERS, LINEITEM, SUPPLIER, NATION, REGION
WHERE C_CUSTKEY = O_CUSTKEY AND L_ORDERKEY = O_ORDERKEY AND L_SUPPKEY = S_SUPPKEY
 AND C_NATIONKEY = S_NATIONKEY AND S_NATIONKEY = N_NATIONKEY AND N_REGIONKEY = R_REGIONKEY
 AND R_NAME = 'ASIA' AND O_ORDERDATE >= '1994-01-01' AND O_ORDERDATE < DATEADD(YY, 1, cast ('1994-01-01'as date))
GROUP BY N_NAME
ORDER BY REVENUE DESC
```

N_NAME	REVENUE
INDONESIA	55502041.169700
VIETNAM	55295086.996700
CHINA	53724494.256600
INDIA	52035512.000200
JAPAN	45410175.695400

(5 row(s) affected)

/* TPC_H Query 6 - Forecasting Revenue Change */

```
SELECT    SUM(L_EXTENDEDPRICE*L_DISCOUNT) AS REVENUE
FROM LINEITEM
WHERE L_SHIPDATE >= '1994-01-01' AND L_SHIPDATE < dateadd (yy, 1, cast('1994-01-01' as date))
AND L_DISCOUNT BETWEEN .06 - 0.01
AND .06 + 0.01 AND L_QUANTITY < 24
```

REVENUE
123141078.228300

(1 row(s) affected)

© 2009 Hewlett-Packard Company. All rights reserved.

/* TPC_H Query 7 - Volume Shipping */

```
SELECT     SUPP_NATION, CUST_NATION, L_YEAR, SUM(VOLUME) AS REVENUE
FROM ( SELECT N1.N_NAME AS SUPP_NATION, N2.N_NAME AS CUST_NATION, datepart(yy, L_SHIPDATE) AS
L_YEAR,
     L_EXTENDEDPRICE*(1-L_DISCOUNT) AS VOLUME
     FROM SUPPLIER, LINEITEM, ORDERS, CUSTOMER, NATION N1, NATION N2
     WHERE S_SUPPKEY = L_SUPPKEY AND O_ORDERKEY = L_ORDERKEY AND C_CUSTKEY = O_CUSTKEY
     AND S_NATIONKEY = N1.N_NATIONKEY AND C_NATIONKEY = N2.N_NATIONKEY AND
          ((N1.N_NAME = 'FRANCE' AND N2.N_NAME = 'GERMANY') OR
          (N1.N_NAME = 'GERMANY' AND N2.N_NAME = 'FRANCE')) AND
          L_SHIPDATE BETWEEN '1995-01-01' AND '1996-12-31' )
     AS SHIPPING
GROUP BY SUPP_NATION, CUST_NATION, L_YEAR
ORDER BY SUPP_NATION, CUST_NATION, L_YEAR

SUPP_NATION      CUST_NATION      L_YEAR     REVENUE
-------------------------- -------------------------- -------------- -------------------------
FRANCE           GERMANY          1995       54639732.733600
FRANCE           GERMANY          1996       54633083.307600
GERMANY          FRANCE           1995       52531746.669700
GERMANY          FRANCE           1996       52520549.022400

(4 row(s) affected)
```

/* TPC_H Query 8 - National Market Share */

```
SELECT O_YEAR, SUM(CASE WHEN NATION = 'BRAZIL' THEN VOLUME ELSE 0 END) / SUM(VOLUME) AS MKT_SHARE
FROM (SELECT datepart(yy, O_ORDERDATE) AS O_YEAR, L_EXTENDEDPRICE * (1-L_DISCOUNT) AS VOLUME,
     N2.N_NAME AS NATION
     FROM PART, SUPPLIER,   LINEITEM, ORDERS, CUSTOMER, NATION N1, NATION N2, REGION
     WHERE P_PARTKEY = L_PARTKEY AND S_SUPPKEY = L_SUPPKEY AND L_ORDERKEY = O_ORDERKEY
      AND O_CUSTKEY = C_CUSTKEY AND C_NATIONKEY = N1.N_NATIONKEY AND
     N1.N_REGIONKEY = R_REGIONKEY AND R_NAME = 'AMERICA' AND
     S_NATIONKEY = N2.N_NATIONKEY AND O_ORDERDATE  BETWEEN '1995-01-01' AND
     '1996-12-31' AND P_TYPE= 'ECONOMY ANODIZED STEEL') AS ALL_NATIONS
GROUP BY O_YEAR
ORDER BY O_YEAR

O_YEAR       MKT_SHARE
-----------  --------------------
1995         0.034436
1996         0.041486

(2 row(s) affected)
```

/* TPC_H Query 9 - Product Type Profit Measure */

```
SELECT     NATION, O_YEAR, SUM(AMOUNT) AS SUM_PROFIT
FROM (SELECT N_NAME AS NATION, datepart(yy, O_ORDERDATE)   AS O_YEAR,
     L_EXTENDEDPRICE*(1-L_DISCOUNT)-PS_SUPPLYCOST*L_QUANTITY AS AMOUNT
     FROM PART, SUPPLIER,   LINEITEM, PARTSUPP, ORDERS,  NATION
     WHERE S_SUPPKEY = L_SUPPKEY AND PS_SUPPKEY= L_SUPPKEY AND PS_PARTKEY = L_PARTKEY AND
     P_PARTKEY= L_PARTKEY AND O_ORDERKEY = L_ORDERKEY AND S_NATIONKEY      = N_NATIONKEY AND
     P_NAME LIKE '%%green%%') AS PROFIT
GROUP BY NATION, O_YEAR
ORDER BY NATION, O_YEAR DESC

NATION           O_YEAR     SUM_PROFIT
------------------ -------------- ------------------------
ALGERIA          1998       31342867.234500
ALGERIA          1997       57138193.023300
ALGERIA          1996       56140140.133000
ALGERIA          1995       53051469.653400
ALGERIA          1994        53867582.128600
ALGERIA          1993        54942718.132400
ALGERIA          1992       54628034.712700
ARGENTINA        1998       30211185.708100
.
.
VIETNAM          1992        49613838.315100

(175 row(s) affected)
```


© 2009 Hewlett-Packard Company. All rights reserved.

/* TPC_H Query 10 - Returned Item Reporting */

```
SELECT TOP 20 C_CUSTKEY, C_NAME, SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT))     AS REVENUE, C_ACCTBAL,
          N_NAME, C_ADDRESS, C_PHONE, C_COMMENT
FROM CUSTOMER, ORDERS, LINEITEM, NATION
WHERE C_CUSTKEY = O_CUSTKEY AND L_ORDERKEY = O_ORDERKEY AND O_ORDERDATE>= '1993-10-01 AND
     O_ORDERDATE < dateadd(mm, 3, cast('1993-10-01' as date)) AND
     L_RETURNFLAG = 'R' AND C_NATIONKEY= N_NATIONKEY
GROUP BY C_CUSTKEY, C_NAME, C_ACCTBAL, C_PHONE, N_NAME, C_ADDRESS, C_COMMENT
ORDER BY REVENUE DESC

C_CUSTKEY   C_NAME                 REVENUE                C_ACCTBAL              N_NAME                 C_ADDRESS
                        C_PHONE         C_COMMENT
----------- ------------------------ ------------------------ ------------------------ ------------------------------------------
--------------- ---------------------------------------------------------------------------------------------------------------
57040       Customer#000057040       734235.245500          632.870000             JAPAN                  Eioyzjf4pp                    22-895-
641-3466 sits. slyly regular requests sleep alongside of the regular inst
143347      Customer#000143347       721002.694800          2557.470000            EGYPT                  1aReFYv,Kw4                   14-
742-935-3718 ggle carefully enticing requests. final deposits use bold, bold pinto beans. ironic, idle re
60838       Customer#000060838       679127.307700          2454.770000            BRAZIL                 64EaJ5vMAHWJlBOxJklpNc2RJiWE
12-913-494-9813  need to boost against the slyly regular account                                                    .
.
.
.
23431       Customer#000023431       554269.536000          3381.860000            ROMANIA                HgiV0phqhaIa9aydNoIlb
29-915-458-2654 nusual, even instructions: furiously stealthy n

(20 row(s) affected)
```

/* TPC_H Query 11 - Important Stock Identification */

```
SELECT PS_PARTKEY, SUM(PS_SUPPLYCOST*PS_AVAILQTY) AS VALUE
FROM PARTSUPP, SUPPLIER, NATION
WHERE PS_SUPPKEY = S_SUPPKEY AND S_NATIONKEY = N_NATIONKEY AND N_NAME = 'GERMANY'
GROUP BY PS_PARTKEY
HAVING SUM(PS_SUPPLYCOST*PS_AVAILQTY) > (SELECT SUM(PS_SUPPLYCOST*PS_AVAILQTY) * 0.0001000000
     FROM PARTSUPP, SUPPLIER, NATION
WHERE PS_SUPPKEY = S_SUPPKEY AND S_NATIONKEY = N_NATIONKEY AND N_NAME = 'GERMANY')
ORDER BY VALUE DESC

PS_PARTKEY          VALUE
----------------------  ------------------------
129760              17538456.860000
166726              16503353.920000
191287              16474801.970000
161758              16101755.540000
.
.
72073               7877736.110000
5182                7874521.730000

(1048 row(s) affected)
```

/* TPC_H Query 12 - Shipping Modes and Order Priority */

```
SELECT L_SHIPMODE,
     SUM(CASE WHEN O_ORDERPRIORITY  = '1-URGENT' OR  O_ORDERPRIORITY  = '2-HIGH' THEN 1 ELSE 0 END)
     AS HIGH_LINE_COUNT, SUM(CASE WHEN O_ORDERPRIORITY <> '1-URGENT' AND O_ORDERPRIORITY <> '2-HIGH'
     THEN 1 ELSE 0 END) AS LOW_LINE_COUNT
FROM ORDERS, LINEITEM
WHERE O_ORDERKEY = L_ORDERKEY AND L_SHIPMODE IN ('MAIL','SHIP')
     AND L_COMMITDATE < L_RECEIPTDATE
     AND L_SHIPDATE < L_COMMITDATE AND L_RECEIPTDATE >= '1994-01-01'
     AND L_RECEIPTDATE < dateadd(mm, 1, cast('1995-09-01' as date))
GROUP BY L_SHIPMODE
ORDER BY L_SHIPMODE

L_SHIPMODE          HIGH_LINE_COUNT     LOW_LINE_COUNT
--------------------  ----------------------------  ----------------------------
MAIL                6202                9324
SHIP                6200                9262

(2 row(s) affected)
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
/* TPC_H  Query 13 - Customer Distribution */
SELECT     C_COUNT, COUNT(*) AS CUSTDIST
FROM (SELECT C_CUSTKEY, COUNT(O_ORDERKEY)
      FROM CUSTOMER left outer join ORDERS on C_CUSTKEY = O_CUSTKEY
      AND O_COMMENT not like '%%special%%requests%%'
GROUP BY C_CUSTKEY) AS C_ORDERS (C_CUSTKEY, C_COUNT)
GROUP BY C_COUNT
ORDER BY CUSTDIST DESC, C_COUNT DESC

C_COUNT  CUSTDIST
--------------- -----------
0            50005
9            6641
10           6532
11           6014
8            5937
12           5639
13           5024
19           4793
7            4687
.
.
41           2
39           1
(42 row(s) affected)

/* TPC_H  Query 14 - Promotion Effect */
SELECT 100.00 * SUM (CASE WHEN P_TYPE LIKE 'PROMO%% THEN L_EXTENDEDPRICE*(1-L_DISCOUNT)
      ELSE 0 END) / SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT)) AS PROMO_REVENUE
FROM LINEITEM, PART
WHERE L_PARTKEY = P_PARTKEY AND L_SHIPDATE >= '1995-09-01' AND L_SHIPDATE < dateadd(mm, 1, '1995-09-01')

PROMO_REVENUE
-------------------------
16.380779

(1 row(s) affected)

/* TPC_H  Query 15 - Create View for Top Supplier Query */
CREATE VIEW REVENUE0 (SUPPLIER_NO, TOTAL_REVENUE) AS
      SELECT L_SUPPKEY, SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT))  FROM LINEITEM
      WHERE L_SHIPDATE >= '1996-01-01' AND L_SHIPDATE < dateadd(mm, 3, cast( '1996-01-01' as date))
GROUP BY L_SUPPKEY
GO

/* TPC_H  Query 15 - Top Supplier */

SELECT     S_SUPPKEY,
      S_NAME,
      S_ADDRESS,
      S_PHONE,
      TOTAL_REVENUE
FROM  SUPPLIER, REVENUE0
WHERE S_SUPPKEY = SUPPLIER_NO AND TOTAL_REVENUE = ( SELECT MAX(TOTAL_REVENUE) FROM REVENUE0)
ORDER BY S_SUPPKEY

DROP VIEW REVENUE0

S_SUPPKEY    S_NAME                S_ADDRESS                S_PHONE          TOTAL_REVENUE
-------------------- ----------------------------- --------------------------------- --------------- ---------------------------
8449         Supplier#000008449    Wp34zim9qYFbVctdW        20-469-856-8873  1772627.208700

(1 row(s) affected)
```


© 2009 Hewlett-Packard Company. All rights reserved.

/* TPC_H Query 16 - Parts/Supplier Relationship */

```
SELECT     P_BRAND, P_TYPE, P_SIZE, COUNT(DISTINCT PS_SUPPKEY) AS SUPPLIER_CNT
     FROM PARTSUPP, PART
WHERE P_PARTKEY = PS_PARTKEY AND P_BRAND <> 'Brand#45' AND P_TYPE NOT LIKE 'MEDIUM POLISHED%%'
     AND P_SIZE IN (49, 14, 23, 45, 19, 3, 36, 9) AND PS_SUPPKEY NOT IN (SELECT S_SUPPKEY FROM SUPPLIER
          WHERE S_COMMENT LIKE '%%Customer%%Complaints%%')
GROUP BY P_BRAND, P_TYPE, P_SIZE
ORDER BY SUPPLIER_CNT DESC, P_BRAND, P_TYPE, P_SIZE

P_BRAND  P_TYPE                          P_SIZE     SUPPLIER_CNT
--------------  ---------------------------------------------- -----------  -----------------------
Brand#41   MEDIUM BRUSHED TIN            3          28
Brand#54   STANDARD BRUSHED COPPER       14         27
Brand#11   STANDARD BRUSHED TIN          23         24
Brand#11   STANDARD BURNISHED BRASS      36         24
Brand#15   MEDIUM ANODIZED NICKEL        3          24
.
.
Brand#55   STANDARD PLATED TIN           49         3

(18314 row(s) affected)
```

/* TPC_H Query 17 - Small-Quantity-Order Revenue */

```
SELECT SUM(L_EXTENDEDPRICE)/7.0 AS AVG_YEARLY FROM LINEITEM, PART
WHERE P_PARTKEY = L_PARTKEY AND P_BRAND = 'Brand#23' AND P_CONTAINER = 'MED BOX'
AND L_QUANTITY<(SELECT 0.2 * AVG(L_QUANTITY) FROM LINEITEM WHERE L_PARTKEY = P_PARTKEY)

AVG_YEARLY
------------------------
348406.054286

(1 row(s) affected)
```

/* TPC_H Query 18 - Large Volume Customer */

```
SELECT TOP 100 C_NAME, C_CUSTKEY, O_ORDERKEY, O_ORDERDATE, O_TOTALPRICE,
     SUM(L_QUANTITY)
FROM CUSTOMER, ORDERS, LINEITEM
WHERE O_ORDERKEY IN (SELECT L_ORDERKEY FROM LINEITEM GROUP BY L_ORDERKEY HAVING
     SUM(L_QUANTITY) > 300) AND C_CUSTKEY = O_CUSTKEY AND O_ORDERKEY = L_ORDERKEY
GROUP BY C_NAME, C_CUSTKEY, O_ORDERKEY, O_ORDERDATE, O_TOTALPRICE
ORDER BY O_TOTALPRICE DESC, O_ORDERDATE

C_NAME                 C_CUSTKEY   O_ORDERKEY   O_ORDERDATE            O_TOTALPRICE
-------------------------  ------------------  ------------------------  ------------------------------  ------------------------  ------------------------
Customer#000128120  128120      4722021      1994-04-07 00:00:00.000  544089.090000    323.000000
Customer#000144617  144617      3043270      1997-02-12 00:00:00.000  530604.440000    317.000000
Customer#000013940  13940       2232932      1997-04-13 00:00:00.000  522720.610000    304.000000
Customer#000066790  66790       2199712      1996-09-30 00:00:00.000  515531.820000    327.000000
.
.
Customer#000088703  88703       2995076      1994-01-30 00:00:00.000  363812.120000    302.000000

(57 row(s) affected)
```

/* TPC_H Query 19 - Discounted Revenue */

```
SELECT SUM(L_EXTENDEDPRICE* (1 - L_DISCOUNT)) AS REVENUE
FROM LINEITEM, PART
WHERE (P_PARTKEY = L_PARTKEY AND P_BRAND = 'Brand#12' AND P_CONTAINER IN ( 'SM CASE', 'SM BOX',
      'SM PACK', 'SM PKG') AND L_QUANTITY >= 1 AND L_QUANTITY <= 1 + 10 AND P_SIZE BETWEEN 1 AND 5
     AND L_SHIPMODE IN ('AIR', 'AIR REG') AND L_SHIPINSTRUCT = 'DELIVER IN PERSON')
     OR (P_PARTKEY = L_PARTKEY AND P_BRAND = 'Brand#23' AND P_CONTAINER IN ( 'MED BAG', 'MED BOX',
     'MED PKG', 'MED PACK') AND L_QUANTITY >= 10 AND L_QUANTITY <= 10 + 10 AND P_SIZE BETWEEN 1
     AND 10 AND L_SHIPMODE IN ('AIR', 'AIR REG') AND L_SHIPINSTRUCT = 'DELIVER IN PERSON') OR
     (P_PARTKEY = L_PARTKEY AND P_BRAND = 'Brand#34'  AND P_CONTAINER IN ( 'LG CASE', 'LG BOX',
     'LG PACK', 'LG PKG') AND L_QUANTITY >= 20 AND L_QUANTITY <= 20 + 10 AND P_SIZE BETWEEN 1 AND 15
     AND L_SHIPMODE IN ('AIR', 'AIR REG') AND L_SHIPINSTRUCT = 'DELIVER IN PERSON')

REVENUE
------------------------
3083843.057800
(1 row(s) affected)
```


© 2009 Hewlett-Packard Company. All rights reserved.

/* TPC_H Query 20 - Potential Part Promotion */

```
SELECT S_NAME, S_ADDRESS FROM SUPPLIER, NATION
WHERE S_SUPPKEY IN ( SELECT PS_SUPPKEY FROM PARTSUPP
    WHERE PS_PARTKEY in (SELECT P_PARTKEY FROM PART WHERE P_NAME like 'forest%%') AND
    PS_AVAILQTY (SELECT 0.5 * sum(L_QUANTITY) FROM LINEITEM WHERE L_PARTKEY = PS_PARTKEY AND
    L_SUPPKEY  = PS_SUPPKEY AND L_SHIPDATE >= '1994-01-01' AND
    L_SHIPDATE < dateadd(yy,1,'1994-01-01')) AND S_NATIONKEY = N_NATIONKEY AND N_NAME = 'CANADA'
ORDER BY S_NAME
```

```
S_NAME                    S_ADDRESS
----------------------------  ----------------------------------------------------------------------
Supplier#000000020        iybAE,RmTymrZVYaFZva2SH,j
Supplier#000000091        YV45D7TkfdQanOOZ7q9QxkyGUapU1oOWU6q3
Supplier#000000197        YC2Acon6kjY3zj3Fbxs2k4Vdf7X0cd2F
Supplier#000000226        83qOdU2EYRdPQAQhEtn GRZEd
Supplier#000000285        Br7e1nnt1yxrw6ImgpJ7YdhFDjuBf
Supplier#000000378        FfbhyCxWvcPrO8ltp9
.
.
Supplier#000009899        7XdpAHrzr1t,UQFZE
Supplier#000009974        7wJ,J5DKcxSU4Kp1cQLpbcAvB5AsvKT

(204 row(s) affected)
```

/* TPC_H Query 21 - Suppliers Who Kept Orders Waiting */

```
SELECT TOP 100 S_NAME, COUNT(*) AS NUMWAIT
FROM SUPPLIER, LINEITEM L1, ORDERS, NATION WHERE S_SUPPKEY = L1.L_SUPPKEY AND
    O_ORDERKEY = L1.L_ORDERKEY AND O_ORDERSTATUS = 'F' AND L1.L_RECEIPTDATE> L1.L_COMMITDATE
    AND EXISTS (SELECT * FROM LINEITEM L2 WHERE L2.L_ORDERKEY = L1.L_ORDERKEY
    AND L2.L_SUPPKEY <> L1.L_SUPPKEY) AND
    NOT EXISTS (SELECT * FROM LINEITEM L3 WHERE L3.L_ORDERKEY = L1.L_ORDERKEY AND
    L3.L_SUPPKEY <> L1.L_SUPPKEY AND L3.L_RECEIPTDATE > L3.L_COMMITDATE) AND
    S_NATIONKEY = N_NATIONKEY AND N_NAME = 'SAUDI ARABIA'
GROUP BY S_NAME
ORDER BY NUMWAIT DESC, S_NAME
```

```
S_NAME                    NUMWAIT
----------------------------  ---------------
Supplier#000002829        20
Supplier#000005808        18
Supplier#000000262        17
Supplier#000000496        17
Supplier#000002160        17
.
.
Supplier#000002357        12
Supplier#000002483        12

(100 row(s) affected)
```

/* TPC_H Query 22 - Global Sales Opportunity */

```
SELECT CNTRYCODE, COUNT(*) AS NUMCUST, SUM(C_ACCTBAL) AS TOTACCTBAL
FROM (SELECT SUBSTRING(C_PHONE,1,2) AS CNTRYCODE, C_ACCTBAL
    FROM CUSTOMER WHERE SUBSTRING(C_PHONE,1,2) IN ('13', '31', '23', '29', '30', '18', '17') AND
        C_ACCTBAL > (SELECT AVG(C_ACCTBAL) FROM CUSTOMER WHERE C_ACCTBAL > 0.00 AND
        SUBSTRING(C_PHONE,1,2) IN ('13', '31', '23', '29', '30', '18', '17')) AND
        NOT EXISTS ( SELECT * FROM ORDERS  WHERE O_CUSTKEY = C_CUSTKEY)) AS CUSTSALE
GROUP BY CNTRYCODE
ORDER BY CNTRYCODE
```

```
CNTRYCODE   NUMCUST       TOTACCTBAL
-------------------  ----------------  ---------------------
13                  888               6737713.990000
17                  861               6460573.720000
18                  964               7236687.400000
23                  892               6701457.950000
29                  948               7158866.630000
30                  909               6808436.130000
31                  922               6806670.180000

(7 row(s) affected)
```


© 2009 Hewlett-Packard Company. All rights reserved.

Appendix D: Seeds and Query Substitution Parameters

Stream 0 Substitution Parameters Seed 712035837	Stream 1 Substitution Parameters 712035838
1 64	1 72
2 2 NICKEL MIDDLE EAST	2 40 COPPER AMERICA
3 FURNITURE 1995-03-26	3 AUTOMOBILE 1995-03-11
4 1997-01-01	4 1994-10-01
5 ASIA 1996-01-01	5 EUROPE 1996-01-01
6 1996-01-01 0.09 25	6 1996-01-01 0.06 25
7 RUSSIA JAPAN	7 KENYA IRAN
8 RUSSIA EUROPE MEDIUM BRUSHED COPPER	8 KENYA AFRICA MEDIUM POLISHED TIN
9 lime	9 khaki
10 1994-07-01	10 1993-04-01
11 ETHIOPIA 0.0001000000	11 CHINA 0.0001000000
12 FOB MAIL 1993-01-01	12 TRUCK MAIL 1993-01-01
13 unusual packages	13 unusual packages
14 1993-04-01	14 1993-07-01
15 1995-07-01	15 1993-03-01
16 Brand#33 ECONOMY BRUSHED 14 5 34 44 15 7 26 9	16 Brand#13 SMALL ANODIZED 31 13 24 23 12 49 9 43
17 Brand#35 MED PKG	17 Brand#31 JUMBO CASE
18 315	18 312
19 Brand#32 Brand#15 Brand#21 2 20 27	19 Brand#44 Brand#43 Brand#25 8 10 23
20 dim 1993-01-01 VIETNAM	20 peach 1996-01-01 IRAQ
21 ARGENTINA	21 ROMANIA
22 20 17 13 19 12 24 16	22 21 24 34 27 15 25 31

Stream 2 Substitution Parameters 712035839	Stream 3 Substitution Parameters 712035840
1 80	1 88
2 28 STEEL MIDDLE EAST	2 15 BRASS ASIA
3 FURNITURE 1995-03-28	3 MACHINERY 1995-03-13
4 1997-05-01	4 1995-02-01
5 MIDDLE EAST 1996-01-01	5 AFRICA 1997-01-01
6 1996-01-01 0.03 24	6 1997-01-01 0.09 25
7 FRANCE INDONESIA	7 UNITED KINGDOM GERMANY
8 FRANCE EUROPE MEDIUM BURNISHED TIN	8 UNITED KINGDOM EUROPE SMALL BRUSHED TIN
9 green	9 floral
10 1994-02-01	10 1994-11-01
11 FRANCE 0.0001000000	11 ROMANIA 0.0001000000
12 RAIL FOB 1993-01-01	12 AIR FOB 1994-01-01
13 unusual packages	13 express packages
14 1993-10-01	14 1994-02-01
15 1995-10-01	15 1993-07-01
16 Brand#43 LARGE PLATED 38 46 4 10 8 27 18 37	16 Brand#33 PROMO POLISHED 17 49 7 28 5 18 39 43
17 Brand#33 JUMBO BAG	17 Brand#35 JUMBO PKG
18 314	18 315
19 Brand#41 Brand#31 Brand#15 3 11 30	19 Brand#43 Brand#14 Brand#14 8 12 26
20 blue 1995-01-01 ARGENTINA	20 magenta 1993-01-01 MOROCCO
21 IRAQ	21 CANADA
22 21 10 30 14 28 17 22	22 25 14 24 29 16 18 22

© 2009 Hewlett-Packard Company. All rights reserved.

Stream 4 Substitution Parameters 712035841	Stream 5 Substitution Parameters 712035842
1 96 2 3 NICKEL MIDDLE EAST 3 FURNITURE 1995-03-30 4 1997-08-01 5 AMERICA 1997-01-01 6 1997-01-01 0.06 25 7 MOROCCO ETHIOPIA 8 MOROCCO AFRICA SMALL PLATED TIN 9 dark 10 1993-08-01 11 GERMANY 0.0001000000 12 REG AIR FOB 1994-01-01 13 express requests 14 1994-05-01 15 1996-02-01 16 Brand#13 MEDIUM ANODIZED 41 48 42 21 47 38 15 17 17 Brand#32 WRAP CASE 18 313 19 Brand#55 Brand#52 Brand#13 3 13 22 20 thistle 1997-01-01 ETHIOPIA 21 SAUDI ARABIA 22 11 20 14 17 19 30 26	1 104 2 41 TIN ASIA 3 MACHINERY 1995-03-15 4 1995-05-01 5 EUROPE 1997-01-01 6 1997-01-01 0.04 24 7 FRANCE GERMANY 8 GERMANY EUROPE SMALL ANODIZED TIN 9 chocolate 10 1994-05-01 11 SAUDI ARABIA 0.0001000000 12 SHIP FOB 1994-01-01 13 express requests 14 1994-08-01 15 1993-10-01 16 Brand#43 ECONOMY BURNISHED 44 21 27 41 34 2 42 22 17 Brand#34 WRAP BAG 18 314 19 Brand#52 Brand#35 Brand#52 9 14 30 20 ghost 1995-01-01 SAUDI ARABIA 21 JAPAN 22 27 33 18 10 20 11 26

Stream 6 Substitution Parameters 712035843
1 112 2 29 STEEL AFRICA 3 BUILDING 1995-03-01 4 1993-02-01 5 MIDDLE EAST 1997-01-01 6 1997-01-01 0.09 25 7 UNITED KINGDOM UNITED STATES 8 UNITED STATES AMERICA STANDARD POLISHED NICKEL 9 blush 10 1993-02-01 11 INDIA 0.0001000000 12 FOB SHIP 1994-01-01 13 express requests 14 1994-11-01 15 1996-05-01 16 Brand#33 STANDARD POLISHED 19 13 25 14 45 35 18 26 17 Brand#31 WRAP PKG 18 312 19 Brand#54 Brand#13 Brand#52 4 15 26 20 rose 1993-01-01 IRAN 21 EGYPT 22 25 31 15 18 19 14 26

© 2009 Hewlett-Packard Company. All rights reserved.

Appendix E: Refresh Function Source Code

RF1

```
IF exists (SELECT name FROM sysobjects WHERE name = 'RF1')
     DROP PROCEDURE RF1
GO
--
--  Create a stored RefreshInsert procedure which will catch the deadlock
--  victim abort and restart the insert transaction.
--
CREATE PROCEDURE RF1
      @current_execution INTEGER, @insert_sets INTEGER, @parallel_executions INTEGER, @total_executions INTEGER
AS
BEGIN

DECLARE @startdate DATE
DECLARE @enddate DATE
DECLARE @edate DATE
DECLARE @rangeStart INTEGER
DECLARE @rangeSize INTEGER
DECLARE @range INTEGER

DECLARE @success INTEGER
DECLARE @index INTEGER
DECLARE @div INTEGER
DECLARE @mod INTEGER
DECLARE @skip INTEGER
DECLARE @i INTEGER
DECLARE @rangeSum INTEGER
DECLARE @totRangeSize INTEGER
DECLARE @stmt NCHAR(1000)
DECLARE @orderSql NCHAR(1000)
DECLARE @liSql NCHAR(1000)

DECLARE @ErrorMessage    NVARCHAR(4000)
DECLARE @ErrorNumber     INT
DECLARE @ErrorSeverity   INT
DECLARE @ErrorState      INT
DECLARE @ErrorLine       INT
DECLARE @ErrorProcedure  NVARCHAR(200)

SET @skip = @total_executions/@parallel_executions
SET @div = (@current_execution - 1)/@parallel_executions
SET @mod = (@current_execution - 1) - @div * @parallel_executions
SET @index = @mod*@skip + @div + 1

--
-- Get the range for this execution
--
SET @stmt = N'SELECT @sdate = dateadd(day,-1,cast(min(O_ORDERDATE) as date)), @edate = max(O_ORDERDATE)
          FROM NEWORDERS'
EXEC sp_executesql @stmt,N'@sdate date output, @edate date output',@startdate output, @enddate output

IF (@total_executions > @parallel_executions)
     BEGIN
     SET @div = (@index-1)/@skip
     SET @mod = (@index-1) - @div * @skip

     --SET @rangeSize = datediff(day, @startdate, cast(@enddate as date))/@parallel_executions + 1
     SET @rangeSize = ((@div+1) * datediff(day, @startdate, cast(@enddate as date)))/@parallel_executions - (@div * datediff(day,
@startdate, cast(@enddate as date)))/@parallel_executions

     SET @totRangeSize = @rangeSize
     SET @rangeSum = 0

     --SET @rangeStart = @div * @rangeSize
     SET @rangeStart = (@div * datediff(day, @startdate, cast(@enddate as date)))/@parallel_executions

     SET @i = @mod
     while (@i > 0)
          BEGIN
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
            SET @rangeSize = (@totRangeSize - @rangeSum)/2
            SET @rangeSum = @rangeSum + @rangeSize
            SET @rangeStart = @rangeStart + @rangeSize
            SET @insert_sets = @insert_sets/2
            SET @i = @i - 1
            end

        IF (@mod + 1 = @skip) --  last allocation
            SET @rangeSize = @totRangeSize - @rangeSum
        ELSE
            SET @rangeSize = (@totRangeSize - @rangeSum)/2
        IF (@rangeSize < 0)
            SET @rangeSize = 0
        IF (@insert_sets <= 0)
            SET @insert_sets = 1
        end
    ELSE
        BEGIN
        SET @rangeSize = (@current_execution * datediff(day, @startdate, cast(@enddate as date)))/@total_executions - ((@current_execution-
1) * datediff(day, @startdate, cast(@enddate as date)))/@total_executions
        SET @rangeStart = ((@current_execution-1) * datediff(day, @startdate, cast(@enddate as date)))/@total_executions
        end

    SET @startdate = dateadd(day, @rangeStart, cast(@startdate as date))
    IF (@index < @total_executions)
        SET @enddate = dateadd(day, @rangeSize, cast(@startdate as date))

    SET @range = datediff(day, @startdate, cast(@enddate as date)) / @insert_sets

    --
    --  This handles the case when the max-min/insert_sets is less that 1
    --
    IF @range = 0
        SET @range = 1

    --
    -- Generate the two insert statements
    --

    SET @edate = dateadd(day, @range, cast(@startdate as date))
    SET @orderSql = N'INSERT INTO ORDERS (O_ORDERKEY, O_CUSTKEY, O_ORDERSTATUS, O_TOTALPRICE,
                        O_ORDERDATE, O_ORDERPRIORITY, O_CLERK, O_SHIPPRIORITY, O_COMMENT)
                (SELECT O_ORDERKEY, O_CUSTKEY, O_ORDERSTATUS, O_TOTALPRICE,
                        O_ORDERDATE, O_ORDERPRIORITY, O_CLERK, O_SHIPPRIORITY, O_COMMENT
                 FROM NEWORDERS
                 WHERE O_ORDERDATE > @startdate AND O_ORDERDATE <= @edate)
                option (loop join,MaxDop 1)'
    SET @liSql = N'INSERT INTO LINEITEM (L_ORDERKEY,L_PARTKEY,L_SUPPKEY,L_LINENUMBER,L_QUANTITY,
                    L_EXTENDEDPRICE, L_DISCOUNT, L_TAX, L_RETURNFLAG, L_LINESTATUS,
                    L_SHIPDATE, L_COMMITDATE, L_RECEIPTDATE, L_SHIPINSTRUCT, L_SHIPMODE, L_COMMENT)
                (SELECT L_ORDERKEY,L_PARTKEY,L_SUPPKEY,L_LINENUMBER,L_QUANTITY,
                    L_EXTENDEDPRICE, L_DISCOUNT, L_TAX, L_RETURNFLAG, L_LINESTATUS,
                    L_SHIPDATE, L_COMMITDATE, L_RECEIPTDATE, L_SHIPINSTRUCT, L_SHIPMODE, L_COMMENT
                 FROM NEWLINEITEM, NEWORDERS
                 WHERE L_ORDERKEY = O_ORDERKEY AND O_ORDERDATE > @startdate AND
                        O_ORDERDATE <= @edate)
                option (loop join,MaxDop 1)'

    --
    --  Loop through the order keys inserting sets into the
    --  ORDERS and LINTEITEM tables
    --
    WHILE @startdate < @enddate
        BEGIN
        --
        --  Insert into ORDERS and LINEITEM tables
        --
        INSERT_TRANS:
        SET @success = 1
        BEGIN TRANSACTION

        BEGIN TRY
            EXEC sp_executesql @orderSql, N'@startdate date, @edate date', @startdate, @edate
            EXEC sp_executesql @liSql, N'@startdate date, @edate date', @startdate, @edate
        END TRY
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
    BEGIN CATCH
        SET @success = 0
        IF (error_number() = 1205)  -- deadlock victim
            BEGIN
            PRINT 'Insert deadlock - restarting RF1'
            IF (@@trancount>0)
                ROLLBACK TRANSACTION
            END
        ELSE
            BEGIN             -- not a deadlock
            PRINT ERROR_MESSAGE()
            SELECT     @ErrorNumber = ERROR_NUMBER(),
                @ErrorSeverity = ERROR_SEVERITY(),
                @ErrorState = ERROR_STATE(),
                @ErrorLine = ERROR_LINE(),
                @ErrorProcedure = ISNULL(ERROR_PROCEDURE(), '-');
            SELECT @ErrorMessage = N'Error %%d, Level %%d, State %%d, Procedure %%s, Line %%d, ' +
                                    'Message: '+ ERROR_MESSAGE();
            IF (@@trancount>0)
                ROLLBACK TRANSACTION
            RAISERROR
                (
                @ErrorMessage,
                @ErrorSeverity,
                1,
                @ErrorNumber,    -- parameter: original error number.
                @ErrorSeverity,  -- parameter: original error severity.
                @ErrorState,     -- parameter: original error state.
                @ErrorProcedure, -- parameter: original error procedure name.
                @ErrorLine       -- parameter: original error line number.
                );
            END
    END CATCH

    IF (@success = 0)              -- deadlock - redo the inserts
        GOTO INSERT_TRANS

    COMMIT TRANSACTION

    SET @startdate = @edate
    SET @edate = dateadd(day, @range, cast(@edate as date))

    IF (@edate > @enddate)
        SET @edate = @enddate

    END
END

GO
```


© 2009 Hewlett-Packard Company. All rights reserved.

RF2

```
IF exists (SELECT name FROM sysobjects WHERE name = 'RF2')
     DROP PROCEDURE RF2
GO

--
--  Create a stored Refresh Delete procedure which will catch the deadlock
--  victim abort and restart the delete transaction.
--
CREATE PROCEDURE RF2
      @current_execution INTEGER, @delete_sets INTEGER, @parallel_executions INTEGER, @total_executions INTEGER
AS
BEGIN

DECLARE @startdate DATE
DECLARE @enddate DATE
DECLARE @edate DATE
DECLARE @rangeStart INTEGER
DECLARE @rangeSize INTEGER
DECLARE @range INTEGER

declare @success INTEGER
declare @index INTEGER
declare @div INTEGER
declare @mod INTEGER
declare @skip INTEGER
declare @i INTEGER
declare @rangeSum INTEGER
declare @totRangeSize INTEGER
declare @sql NCHAR(1000)
declare @orderSql NCHAR(1000)
declare @liSql NCHAR(1000)

DECLARE @ErrorMessage    NVARCHAR(4000)
DECLARE @ErrorNumber     INT
DECLARE @ErrorSeverity   INT
DECLARE @ErrorState      INT
DECLARE @ErrorLine       INT
DECLARE @ErrorProcedure  NVARCHAR(200)

SET @skip = @total_executions/@parallel_executions
SET @div = floor((@current_execution-1)/@parallel_executions)
SET @mod = (@current_execution - 1) - @div * @parallel_executions
SET @index = @mod*@skip + @div + 1

SET @sql = N'SELECT @sdate = dateadd(day,-1,cast(min(O_ORDERDATE) as date)), @edate = max(O_ORDERDATE)
          FROM MOD_OLDORDERS'
EXEC sp_executesql @sql,N'@sdate date output, @edate date output',@startdate output, @enddate output

IF (@total_executions > @parallel_executions)
     BEGIN
     SET @div = (@index-1)/@skip
     SET @mod = (@index-1) - @div * @skip
     --SET @rangeSize = datediff(day, @startdate, cast(@enddate as date))/@parallel_executions + 1
     SET @rangeSize = ((@div+1) * datediff(day, @startdate, cast(@enddate as date)))/@parallel_executions - (@div * datediff(day,
@startdate, cast(@enddate as date)))/@parallel_executions

     SET @totRangeSize = @rangeSize
     SET @rangeSum = 0

     --SET @rangeStart = @div * @rangeSize
     SET @rangeStart = (@div * datediff(day, @startdate, cast(@enddate as date)))/@parallel_executions

     SET @i = @mod
     WHILE (@i > 0)
          BEGIN
          SET @rangeSize = (@totRangeSize - @rangeSum)/2
          SET @rangeSum = @rangeSum + @rangeSize
          SET @rangeStart = @rangeStart + @rangeSize
          SET @delete_sets = @delete_sets/2
          SET @i = @i - 1
          END

     IF (@mod + 1 = @skip) --  last allocation
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
        SET @rangeSize = @totRangeSize - @rangeSum
    ELSE
        SET @rangeSize = (@totRangeSize - @rangeSum)/2
    IF (@rangeSize < 0)
        SET @rangeSize = 0
    IF (@delete_sets <= 0)
        SET @delete_sets = 1
    END
ELSE
    BEGIN
    SET @rangeSize = (@current_execution * datediff(day, @startdate, cast(@enddate as date)))/@total_executions - ((@current_execution-
1) * datediff(day, @startdate, cast(@enddate as date)))/@total_executions
    SET @rangeStart = ((@current_execution-1) * datediff(day, @startdate,cast(@enddate as date)))/@total_executions
    END

SET @startdate = dateadd(day, @rangeStart,cast(@startdate as date))
IF (@index < @total_executions)
    SET @enddate = dateadd(day, @rangeSize, cast(@startdate as date))

SET @range = datediff(day, @startdate, cast(@enddate as date)) / @delete_sets

--
--  This handles the case when the max-min/delete_sets is less that 1
--
IF @range = 0
    SET @range = 1

--
--  Loop through the order keys deleting sets from orders
--  and lineitem tables
--
SET @edate = dateadd(day, @range, cast(@startdate as date))
SET @liSql = N'DELETE FROM LINEITEM WHERE L_ORDERKEY in
            (SELECT O_ORDERKEY FROM MOD_OLDORDERS
             WHERE O_ORDERDATE > @startdate AND O_ORDERDATE <= @edate)
            option (loop join,MaxDop 1)'
SET @orderSql = N'DELETE FROM ORDERS WHERE O_ORDERKEY in
            (SELECT O_ORDERKEY FROM MOD_OLDORDERS
             WHERE O_ORDERDATE > @startdate AND O_ORDERDATE <= @edate)
            option (loop join,MaxDop 1)'

WHILE @startdate < @enddate
    BEGIN

    DELETE_TRANS:
    SET @success = 1
    BEGIN TRANSACTION

    BEGIN TRY
        EXEC sp_executesql @liSql, N'@startdate date, @edate date', @startdate, @edate
        EXEC sp_executesql @orderSql, N'@startdate date, @edate date', @startdate, @edate
    END TRY
    BEGIN CATCH
        SET @success = 0
        IF (error_number() = 1205)  -- deadlock victim
            BEGIN
            PRINT 'Insert deadlock - restarting RF2'
            IF (@@trancount>0)
                ROLLBACK TRANSACTION
            END
        ELSE
            BEGIN           -- not a deadlock
            PRINT ERROR_MESSAGE()
            SELECT     @ErrorNumber = ERROR_NUMBER(),
                @ErrorSeverity = ERROR_SEVERITY(),
                @ErrorState = ERROR_STATE(),
                @ErrorLine = ERROR_LINE(),
                @ErrorProcedure = ISNULL(ERROR_PROCEDURE(), '-');
            SELECT @ErrorMessage = N'Error %%d, Level %%d, State %%d, Procedure %%s, Line %%d, ' +
                                    'Message: '+ ERROR_MESSAGE();
            IF (@@trancount>0)
                ROLLBACK TRANSACTION
            RAISERROR
                (
                @ErrorMessage,
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
                @ErrorSeverity,
                1,
                @ErrorNumber,    -- parameter: original error number.
                @ErrorSeverity,  -- parameter: original error severity.
                @ErrorState,     -- parameter: original error state.
                @ErrorProcedure, -- parameter: original error procedure name.
                @ErrorLine       -- parameter: original error line number.
                );
            END
    END CATCH

    IF (@success = 0)              -- deadlock - redo the inserts
        GOTO DELETE_TRANS

    COMMIT TRANSACTION

    SET @startdate = @edate
    SET @edate = dateadd(day, @range, cast(@edate as date))

    IF (@edate > @enddate)
        SET @edate = @enddate

    END
END

GO
```

RF1_Whole.CMD

```
setlocal
set TPCH_NUM_LOAD_STREAMS=%1
set TPCH_NUM_LOAD_FILES=%2
set TPCH_NUM_EXEC_STREAMS=%3
set TPCH_NUM_EXEC_TOTAL=%4
set TPCH_EXEC_BATCH_SIZE=%5
set TPCH_FF_LOCN=%6
set TPCH_RF_STREAM_NUM=%7
set TPCH_RUN_NUM=%8
set TPCH_COMMENT_STR=%9
set
TPCH_OUTPUT_DIR=%TPCH_OUTPUT_DIR_PREFIX%\%TPC
H_RUN_NUM%
set
TPCH_MASTERLOG_FILE=%TPCH_OUTPUT_DIR%\%TPCH_M
ASTERLOG_NAME%
if not exist %TPCH_OUTPUT_DIR% md %TPCH_OUTPUT_DIR%
pushd %TPCH_OUTPUT_DIR%
echo !DATE! !TIME! Start: Executing RF1
%TPCH_COMMENT_STR% >> %TPCH_MASTERLOG_FILE%
%TPCH_SQL_SHELL% -d %TPCH_DATABASE% -i
create_RF1_tables.sql >> create_RF1_tables.out
set
TPCH_SEMAPHORE_NAME=Load.%TPCH_RF_STREAM_NUM
%.%TPCH_RUN_NUM%
start
%TPCH_AUTOMATION_CMD_PATH%\WaitForSemaphore.cmd
%TPCH_SEMAPHORE_NAME%
%TPCH_NUM_LOAD_STREAMS%
%TPCH_SEMAPHORE_NAME%_all
set /A LOCAL_CURR_STREAM = 1
:outerLoopLoadRun
start
%TPCH_AUTOMATION_CMD_PATH%\execSQLFileListSemapho
re.cmd %TPCH_SEMAPHORE_NAME%
RF1_Load_List_%TPCH_RF_STREAM_NUM%.%LOCAL_CURR
_STREAM%.txt %TPCH_RUN_NUM% no no no
set /A LOCAL_CURR_STREAM = %LOCAL_CURR_STREAM% +
1
if %LOCAL_CURR_STREAM% LEQ
%TPCH_NUM_LOAD_STREAMS% goto outerLoopLoadRun
%TPCH_AUTOMATION_EXEC%\semaphore -wait
%TPCH_SEMAPHORE_NAME%_all -count 1
%TPCH_SQL_SHELL% -d %TPCH_DATABASE% -i
create_RF1_indices.sql >> create_indices_RF1.out
set
TPCH_SEMAPHORE_NAME=Exec.%TPCH_RF_STREAM_NUM
```

RF2_Whole.CMD

```
setlocal
set TPCH_NUM_LOAD_STREAMS=%1
set TPCH_NUM_LOAD_FILES=%2
set TPCH_NUM_EXEC_STREAMS=%3
set TPCH_NUM_EXEC_TOTAL=%4
set TPCH_EXEC_BATCH_SIZE=%5
set TPCH_FF_LOCN=%6
set TPCH_RF_STREAM_NUM=%7
set TPCH_RUN_NUM=%8
set TPCH_COMMENT_STR=%9
set
TPCH_OUTPUT_DIR=%TPCH_OUTPUT_DIR_PREFIX%\%TPCH_RU
N_NUM%
set
TPCH_MASTERLOG_FILE=%TPCH_OUTPUT_DIR%\%TPCH_MAST
ERLOG_NAME%
if not exist %TPCH_OUTPUT_DIR% md %TPCH_OUTPUT_DIR%
pushd %TPCH_OUTPUT_DIR%
echo !DATE! !TIME! Start: Executing RF2 %TPCH_COMMENT_STR%
>> %TPCH_MASTERLOG_FILE%
%TPCH_SQL_SHELL% -d %TPCH_DATABASE% -i
create_RF2_tables.sql >> create_RF2_tables.out
set
TPCH_SEMAPHORE_NAME=Load.%TPCH_RF_STREAM_NUM%.%T
PCH_RUN_NUM%
start %TPCH_AUTOMATION_CMD_PATH%\WaitForSemaphore.cmd
%TPCH_SEMAPHORE_NAME% %TPCH_NUM_LOAD_STREAMS%
%TPCH_SEMAPHORE_NAME%_all
set /A LOCAL_CURR_STREAM = 1
:outerLoopLoadRun
start
%TPCH_AUTOMATION_CMD_PATH%\execSQLFileListSemaphore.c
md %TPCH_SEMAPHORE_NAME%
RF2_Load_List_%TPCH_RF_STREAM_NUM%.%LOCAL_CURR_STR
EAM%.txt %TPCH_RUN_NUM% no no no

set /A LOCAL_CURR_STREAM = %LOCAL_CURR_STREAM% + 1
if %LOCAL_CURR_STREAM% LEQ
%TPCH_NUM_LOAD_STREAMS% goto outerLoopLoadRun
%TPCH_AUTOMATION_EXEC%\semaphore -wait
%TPCH_SEMAPHORE_NAME%_all -count 1
create_RF2_indices.sql >> create_indices_RF2.out
set
TPCH_SEMAPHORE_NAME=Exec.%TPCH_RF_STREAM_NUM%.%T
PCH_RUN_NUM%
start %TPCH_AUTOMATION_CMD_PATH%\WaitForSemaphore.cmd
%TPCH_SEMAPHORE_NAME% %TPCH_NUM_EXEC_STREAMS%
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
%.%TPCH_RUN_NUM%
start
%TPCH_AUTOMATION_CMD_PATH%\WaitForSemaphore.cmd
%TPCH_SEMAPHORE_NAME%
%TPCH_NUM_EXEC_STREAMS%
%TPCH_SEMAPHORE_NAME%_all
set /A LOCAL_CURR_STREAM = 1
:outerLoopExecRun
start
%TPCH_AUTOMATION_CMD_PATH%\execSQLFileListSemapho
re.cmd %TPCH_SEMAPHORE_NAME%
RF1_Execute_List_%TPCH_RF_STREAM_NUM%.%LOCAL_CU
RR_STREAM%.txt %TPCH_RUN_NUM% no no no
set /A LOCAL_CURR_STREAM = %LOCAL_CURR_STREAM% +
1
if %LOCAL_CURR_STREAM% LEQ
%TPCH_NUM_EXEC_STREAMS% goto outerLoopExecRun
%TPCH_AUTOMATION_EXEC%\semaphore -wait
%TPCH_SEMAPHORE_NAME%_all -count 1
echo !DATE! !TIME! End  : Executing RF1
%TPCH_COMMENT_STR% >> %TPCH_MASTERLOG_FILE%
%TPCH_SQL_SHELL% -d %TPCH_DATABASE% -i
%TPCH_AUTOMATION_SQL%\Dump_Row_Count.sql >>
Post_RF1_Row_Count.out
endlocal
```

```
%TPCH_SEMAPHORE_NAME%_all
set /A LOCAL_CURR_STREAM = 1
:outerLoopExecRun
start
%TPCH_AUTOMATION_CMD_PATH%\execSQLFileListSemaphore.c
md %TPCH_SEMAPHORE_NAME%
RF2_Execute_List_%TPCH_RF_STREAM_NUM%.%LOCAL_CURR_S
TREAM%.txt %TPCH_RUN_NUM% no no no
set /A LOCAL_CURR_STREAM = %LOCAL_CURR_STREAM% + 1
if %LOCAL_CURR_STREAM% LEQ
%TPCH_NUM_EXEC_STREAMS% goto outerLoopExecRun
%TPCH_AUTOMATION_EXEC%\semaphore -wait
%TPCH_SEMAPHORE_NAME%_all -count 1
%TPCH_COMMENT_STR%_Exec >> %TPCH_MASTERLOG_FILE%
echo !DATE! !TIME! End  : Executing RF2 %TPCH_COMMENT_STR%
>> %TPCH_MASTERLOG_FILE%
%TPCH_SQL_SHELL% -d %TPCH_DATABASE% -i
%TPCH_AUTOMATION_SQL%\Dump_Row_Count.sql >>
%TPCH_SQL_SHELL% -d %TPCH_DATABASE% -i
%TPCH_AUTOMATION_SQL%\Increment_Update_Set.sql >>
Increment_Update_Set.out
endlocal
```

Global Variables

```
TPCH_AUTOMATION_PATH "e:\rfs"
TPCH_AUTOMATION_CMD_PATH "%TPCH_AUTOMATION_PATH%\cmd"
TPCH_AUTOMATION_SQL "%TPCH_AUTOMATION_PATH%\sql"
TPCH_AUTOMATION_CONFIG "%TPCH_AUTOMATION_PATH%\config"
TPCH_AUTOMATION_EXEC "%TPCH_AUTOMATION_PATH%\execs"
TPCH_AUTOMATION_VBS "%TPCH_AUTOMATION_PATH%\vbs"
TPCH_MASTERLOG_NAME "MasterLog.txt"
TPCH_OUTPUT_DIR_PREFIX "e:\tpchRuns"
TPCH_EXEC_WINDOW_COLOR "9F"
TPCH_DATABASE "tpch300g"
TPCH_SCALE_FACTOR "300g"
TPCH_SCALE_FACTOR_SIMPLE "300"
TPCH_SQL_SHELL "sqlcmd –SOCTANE1 -Usa -P**** -b -w 65535 -l 60 -y 0
```

Create_RF1_Indices.sql

```
create clustered index NEWORDERS_CLUIDX on NEWORDERS (O_ORDERDATE) on GENERAL_FG
create clustered index NEWLINEITEM_CLUIDX on NEWLINEITEM (L_ORDERKEY) on GENERAL_FG
```

Create_RF1_Tables.sql

```
if exists (select name from sysobjects where name = 'NEWORDERS') drop table NEWORDERS
create table NEWORDERS (O_ORDERKEY bigint not null,
    O_CUSTKEY int not null,
    O_ORDERSTATUS char(1)  not null,
    O_TOTALPRICE  float  not null,
    O_ORDERDATE datetime not null,
    O_ORDERPRIORITY char(15)  not null,
    O_CLERK  char(15) not null,
    O_SHIPPRIORITY  int not null,
    O_COMMENT varchar(79) not null) on GENERAL_FG

if exists (select name from sysobjects  where name = 'NEWLINEITEM') drop table NEWLINEITEM
create table NEWLINEITEM (L_ORDERKEY  bigint  not null,
    L_PARTKEY int not null,
    L_SUPPKEY int not null,
    L_LINENUMBER int not null,
    L_QUANTITY float not null,
    L_EXTENDEDPRICE float not null,
    L_DISCOUNT float not null,
    L_TAX float not null,
    L_RETURNFLAG char(1) not null,
    L_LINESTATUS char(1) not null,
    L_SHIPDATE datetime not null,
    L_COMMITDATE datetime not null,
    L_RECEIPTDATE datetime not null,
    L_SHIPINSTRUCT char(25)  not null,
    L_SHIPMODE char(10 ) not null,
    L_COMMENT varchar(44) not null) on GENERAL_FG
```


© 2009 Hewlett-Packard Company. All rights reserved.

Create_RF2_Indices.sql

```
create clustered index MOD_OLDORDERS_CLUIDX on MOD_OLDORDERS (O_ORDERDATE) on GENERAL_FG
create index MOD_OLDORDERS_IDX on MOD_OLDORDERS (O_ORDERKEY) on GENERAL_FG
```

Create_RF2_Tables.sql

```
declare @segment integer
declare @sql nchar(1000)
if exists (select name from sysobjects  where name = 'MOD_OLDORDERS')  drop table MOD_OLDORDERS
create table MOD_OLDORDERS (O_ORDERDATE datetime, O_ORDERKEY bigint)
set @segment = 1
while @segment <= $(totalSegments)
    begin
    set @sql = 'if exists (select name from sysobjects  where name = "OLDORDERS_'
                    + RTRIM(CONVERT(varchar(30),@segment))+ '")
            drop table OLDORDERS_' + RTRIM(CONVERT(varchar(30),@segment))
    exec sp_executesql @sql
    set @sql = 'create table OLDORDERS_' + RTRIM(CONVERT(varchar(30),@segment))
            + '(O_ORDERKEY int)'
    exec sp_executesql @sql
    set @segment = @segment + 1
    end
```

Execute_RF1_tmpl.sql

```
EXEC  RF1 $(currExec), $(batchSize), $(numParallelExecs), $(totalExecs)
```

Execute_RF2_tmpl.sql

```
EXEC  RF2 $(currExec), $(batchSize), $(numParallelExecs), $(totalExecs)
```

Increment_Update_Set.sql

```
UPDATE TPCH_AUX_TABLE SET updateset=updateset+1
go
select * from TPCH_AUX_TABLE
go
```

Load_RF1_tmpl.sql

```
print 'RF1 Load'
declare @sql nchar(1000)
declare @set integer
select @set = updateset from TPCH_AUX_TABLE
set @sql = 'bulk insert NEWLINEITEM from "$(ffLoc)\Lineitem.tbl.u' + RTRIM(CONVERT(varchar(30),@set))+ '.$(partNo)" with
(FieldTerminator = "|", RowTerminator ="\n",tablock)'
exec sp_executesql @sql

set @sql = 'bulk insert NEWORDERS from "$(ffLoc)\Orders.tbl.u' + RTRIM(CONVERT(varchar(30), @set)) + '.$(partNo)" with
(FieldTerminator = "|", RowTerminator ="\n",tablock)'
exec sp_executesql @sql
```

Load_RF2_tmpl.sql

```
print 'RF2 Load'
declare @set integer
declare @sql nchar(1000)
select @set = updateset from TPCH_AUX_TABLE
set @sql = 'bulk insert OLDORDERS_$(partNo)  from "$(ffLoc)\Delete.u' + RTRIM(CONVERT(varchar(30),@set)) + '.$(partNo)" with
(RowTerminator ="|",tablock)'
exec sp_executesql @sql

insert into MOD_OLDORDERS (O_ORDERDATE, O_ORDERKEY) (select B.O_ORDERDATE, B.O_ORDERKEY from
OLDORDERS_$(partNo) A, ORDERS B where A.O_ORDERKEY = B.O_ORDERKEY)
```

Get_Row_Count.sql

```
select name,  rowcnt from sysindexes where id = object_id('LINEITEM')
select name,  rowcnt from sysindexes where id = object_id('ORDERS')
```

Script calls

```
RF1_Whole 48 48 96 96 40 G:\mnt\ntfs 0 Run1 Run1_Power
RF1_Whole 48 48 96 96 40 G:\mnt\ntfs 7 Run2 Run2_Power
RF2_Whole 48 48 48 96 40 G:\mnt\ntfs 0 Run1 Run1_Power
RF2_Whole 48 48 48 96 40 G:\mnt\ntfs 7 Run2 Run2_Power
RFStreams_Run1 48 48 96 96 40 g:\mnt\ntfs 6 Run1
RFStreams_Run2 48 48 96 96 40 g:\mnt\ntfs 13 Run2
```

execSQLFileList.cmd

```
echo on
set TPCH_INPUT_LIST=%1
set TPCH_INPUT_LIST_DIR=%~dp1
set TPCH_INPUT_LIST_NAME=%~n1
set TPCH_RUN_NUM=%2
set TPCH_OUTPUT_DIR=%TPCH_OUTPUT_DIR_PREFIX%\%TPCH_RUN_NUM%
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
set TPCH_MASTERLOG_FILE=%TPCH_OUTPUT_DIR%\%TPCH_MASTERLOG_NAME%
set TPCH_COLLECT_SHOWPLAN=%3
set TPCH_COLLECT_PERFMON=%4
set TPCH_WRITE_TO_MASTER=%5
if not exist %TPCH_OUTPUT_DIR% md %TPCH_OUTPUT_DIR%
COLOR %TPCH_EXEC_WINDOW_COLOR%
for /F "tokens=1" %%i in (%TPCH_INPUT_LIST%) do (
    set TPCH_CURRENT_FILE_NAME=%%~nxi
    set TPCH_CURRENT_FILE=%TPCH_INPUT_LIST_DIR%\%%~nxi
    set TPCH_CURRENT_OUTPUT_FILE=%TPCH_OUTPUT_DIR%\%%~ni.out
    set TPCH_EXEC_FILE=!TPCH_CURRENT_FILE!
    if not exist !TPCH_CURRENT_FILE! goto :missingfile
    title %%~ni
    if /i '%TPCH_WRITE_TO_MASTER%'=='yes' (
        echo !DATE! !TIME! Start: Executing !TPCH_CURRENT_FILE! >> %TPCH_MASTERLOG_FILE%
    )
    set ERRORLEVEL=0
    %TPCH_SQL_SHELL% -d %TPCH_DATABASE% -i !TPCH_EXEC_FILE! -o !TPCH_CURRENT_OUTPUT_FILE!
    echo %ERRORLEVEL%
    if not errorlevel 0 goto :errorexit
    if /i '%TPCH_WRITE_TO_MASTER%'=='yes' (
        echo !DATE! !TIME! End : Executing !TPCH_CURRENT_FILE! >> %TPCH_MASTERLOG_FILE%
)
)
goto finish
:errorexit
echo Program returned with error when executing %TPCH_CURRENT_FILE%
goto finish
:missingfile
echo File %TPCH_CURRENT_FILE% missing
goto finish
:finish
endlocal
```

execSQLFileListSemaphore

```
setlocal
set TPCH_SEMAPHORE_NAME=%1
shift
call %TPCH_AUTOMATION_CMD_PATH%\execSQLFileList %1 %2 %3 %4 %5 %6 %7 %8 %9
%TPCH_AUTOMATION_EXEC%\semaphore -release %TPCH_SEMAPHORE_NAME%
exit
endlocal
```

RFStreams_Run1.cmd

```
setlocal
set TPCH_NUM_LOAD_STREAMS=%1
set TPCH_NUM_LOAD_FILES=%2
set TPCH_NUM_EXEC_STREAMS=%3
set TPCH_NUM_EXEC_TOTAL=%4
set TPCH_EXEC_BATCH_SIZE=%5
set TPCH_FF_LOCN=%6
set TPCH_NUM_RF_STREAMS=%7
set TPCH_RUN_NUM=%8
set TPCH_OUTPUT_DIR=%TPCH_OUTPUT_DIR_PREFIX%\%TPCH_RUN_NUM%
set TPCH_MASTERLOG_FILE=%TPCH_OUTPUT_DIR%\%TPCH_MASTERLOG_NAME%
if not exist %TPCH_OUTPUT_DIR% md %TPCH_OUTPUT_DIR%
pushd %TPCH_OUTPUT_DIR%
set /A LOCAL_CURR_STREAM = 1
:outerRFLoop
call %TPCH_AUTOMATION_CMD_PATH%\RF1_Whole.cmd %TPCH_NUM_LOAD_STREAMS% %TPCH_NUM_LOAD_FILES%
%TPCH_NUM_EXEC_STREAMS% %TPCH_NUM_EXEC_TOTAL% %TPCH_EXEC_BATCH_SIZE% %TPCH_FF_LOCN%
%LOCAL_CURR_STREAM% %TPCH_RUN_NUM% Throughput_RF1_Stream%LOCAL_CURR_STREAM%
call %TPCH_AUTOMATION_CMD_PATH%\RF2_Whole.cmd %TPCH_NUM_LOAD_STREAMS% %TPCH_NUM_LOAD_FILES%
%TPCH_NUM_EXEC_STREAMS% %TPCH_NUM_EXEC_TOTAL% %TPCH_EXEC_BATCH_SIZE% %TPCH_FF_LOCN%
%LOCAL_CURR_STREAM% %TPCH_RUN_NUM% Throughput_RF2_Stream%LOCAL_CURR_STREAM%
set /A LOCAL_CURR_STREAM = %LOCAL_CURR_STREAM% + 1
if %LOCAL_CURR_STREAM% LEQ %TPCH_NUM_RF_STREAMS% goto outerRFLoop
endlocal
```

RFStreams_Run2

```
setlocal
set TPCH_NUM_LOAD_STREAMS=%1
set TPCH_NUM_LOAD_FILES=%2
set TPCH_NUM_EXEC_STREAMS=%3
set TPCH_NUM_EXEC_TOTAL=%4
set TPCH_EXEC_BATCH_SIZE=%5
set TPCH_FF_LOCN=%6
set TPCH_NUM_RF_STREAMS=%7
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
set TPCH_RUN_NUM=%8
set TPCH_OUTPUT_DIR=%TPCH_OUTPUT_DIR_PREFIX%\%TPCH_RUN_NUM%
set TPCH_MASTERLOG_FILE=%TPCH_OUTPUT_DIR%\%TPCH_MASTERLOG_NAME%
if not exist %TPCH_OUTPUT_DIR% md %TPCH_OUTPUT_DIR%
pushd %TPCH_OUTPUT_DIR%
set /A LOCAL_CURR_STREAM = 9
:outerRFLoop
call %TPCH_AUTOMATION_CMD_PATH%\RF1_Whole.cmd %TPCH_NUM_LOAD_STREAMS% %TPCH_NUM_LOAD_FILES%
%TPCH_NUM_EXEC_STREAMS% %TPCH_NUM_EXEC_TOTAL% %TPCH_EXEC_BATCH_SIZE% %TPCH_FF_LOCN%
%LOCAL_CURR_STREAM% %TPCH_RUN_NUM% Throughput_RF1_Stream%LOCAL_CURR_STREAM%
call %TPCH_AUTOMATION_CMD_PATH%\RF2_Whole.cmd %TPCH_NUM_LOAD_STREAMS% %TPCH_NUM_LOAD_FILES%
%TPCH_NUM_EXEC_STREAMS% %TPCH_NUM_EXEC_TOTAL% %TPCH_EXEC_BATCH_SIZE% %TPCH_FF_LOCN%
%LOCAL_CURR_STREAM% %TPCH_RUN_NUM% Throughput_RF2_Stream%LOCAL_CURR_STREAM%
set /A LOCAL_CURR_STREAM = %LOCAL_CURR_STREAM% + 1
if %LOCAL_CURR_STREAM% LEQ %TPCH_NUM_RF_STREAMS% goto outerRFLoop
endlocal
```

WaitForSemaphore.cmd

```
setlocal
set TPCH_Group=%1
set waitcount=%2
set TPCH_Sentinel=%3
%TPCH_AUTOMATION_EXEC%\semaphore -wait %TPCH_Group% -count %waitcount%
%TPCH_AUTOMATION_EXEC%\semaphore -release %TPCH_Sentinel%
exit
endlocal
```

Create-RF1_Whole

```
setlocal
set TPCH_NUM_LOAD_STREAMS=%1
set TPCH_NUM_LOAD_FILES=%2
set TPCH_NUM_EXEC_STREAMS=%3
set TPCH_NUM_EXEC_TOTAL=%4
set TPCH_EXEC_BATCH_SIZE=%5
set TPCH_FF_LOCN=%6
set TPCH_RF_STREAM_NUM=%7
set /a TPCH_UPDATESET=%TPCH_RF_STREAM_NUM% + 1
set TPCH_RUN_NUM=%8
set TPCH_COMMENT_STR=%9
set TPCH_FF_LOCN=%TPCH_FF_LOCN%\%TPCH_UPDATESET%\rf128
set TPCH_OUTPUT_DIR=%TPCH_OUTPUT_DIR_PREFIX%\%TPCH_RUN_NUM%
set TPCH_MASTERLOG_FILE=%TPCH_OUTPUT_DIR%\%TPCH_MASTERLOG_NAME%
if not exist %TPCH_OUTPUT_DIR% md %TPCH_OUTPUT_DIR%
pushd %TPCH_OUTPUT_DIR%
if not exist Create_RF1_Tables.sql copy %TPCH_AUTOMATION_SQL%\RF_templ\Create_RF1_Tables.sql
if not exist Create_RF1_Indices.sql copy %TPCH_AUTOMATION_SQL%\RF_templ\Create_RF1_Indices.sql
set /A LOCAL_CURR_STREAM = 1
:outerLoopLoadCreate
set /A LOCAL_CURR_LOAD = %LOCAL_CURR_STREAM%
if exist (RF1_Load_List_%TPCH_RF_STREAM_NUM%.%LOCAL_CURR_STREAM%.txt) del
RF1_Load_List_%TPCH_RF_STREAM_NUM%.%LOCAL_CURR_STREAM%.txt
:innerLoopLoadCreate
echo :setvar ffLoc %TPCH_FF_LOCN%  > load_RF1_input.%TPCH_RF_STREAM_NUM%.%LOCAL_CURR_LOAD%.sql
echo :setvar partNo %LOCAL_CURR_LOAD% >> load_RF1_input.%TPCH_RF_STREAM_NUM%.%LOCAL_CURR_LOAD%.sql
type %TPCH_AUTOMATION_SQL%\RF_templ\Load_RF1_tmpl.sql  >>
load_RF1_input.%TPCH_RF_STREAM_NUM%.%LOCAL_CURR_LOAD%.sql
echo load_RF1_input.%TPCH_RF_STREAM_NUM%.%LOCAL_CURR_LOAD%.sql >>
RF1_Load_List_%TPCH_RF_STREAM_NUM%.%LOCAL_CURR_STREAM%.txt
set /A LOCAL_CURR_LOAD = %LOCAL_CURR_LOAD% + %TPCH_NUM_LOAD_STREAMS%
if %LOCAL_CURR_LOAD% LEQ %TPCH_NUM_LOAD_FILES% goto innerLoopLoadCreate
set /A LOCAL_CURR_STREAM = %LOCAL_CURR_STREAM% + 1
if %LOCAL_CURR_STREAM% LEQ %TPCH_NUM_LOAD_STREAMS% goto outerLoopLoadCreate
set /A LOCAL_CURR_STREAM = 1
:outerLoopExecCreate
set /A LOCAL_CURR_EXEC = %LOCAL_CURR_STREAM%
if exist (RF1_Execute_List_%TPCH_RF_STREAM_NUM%.%LOCAL_CURR_STREAM%.txt) del
RF1_Execute_List_%TPCH_RF_STREAM_NUM%.%LOCAL_CURR_STREAM%.txt
:innerLoopExecCreate
echo :setvar currExec %LOCAL_CURR_EXEC% > exec_RF1_input.%TPCH_RF_STREAM_NUM%.%LOCAL_CURR_EXEC%.sql
echo :setvar batchSize %TPCH_EXEC_BATCH_SIZE% >> exec_RF1_input.%TPCH_RF_STREAM_NUM%.%LOCAL_CURR_EXEC%.sql
echo :setvar numParallelExecs %TPCH_NUM_EXEC_STREAMS% >>
exec_RF1_input.%TPCH_RF_STREAM_NUM%.%LOCAL_CURR_EXEC%.sql
echo :setvar totalExecs %TPCH_NUM_EXEC_TOTAL%  >> exec_RF1_input.%TPCH_RF_STREAM_NUM%.%LOCAL_CURR_EXEC%.sql
type %TPCH_AUTOMATION_SQL%\RF_templ\Execute_RF1_tmpl.sql>>
exec_RF1_input.%TPCH_RF_STREAM_NUM%.%LOCAL_CURR_EXEC%.sql
echo exec_RF1_input.%TPCH_RF_STREAM_NUM%.%LOCAL_CURR_EXEC%.sql >>
RF1_Execute_List_%TPCH_RF_STREAM_NUM%.%LOCAL_CURR_STREAM%.txt
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
set /A LOCAL_CURR_EXEC = %LOCAL_CURR_EXEC% + %TPCH_NUM_EXEC_STREAMS%
if %LOCAL_CURR_EXEC% LEQ %TPCH_NUM_EXEC_TOTAL% goto innerLoopExecCreate
set /A LOCAL_CURR_STREAM = %LOCAL_CURR_STREAM% + 1
if %LOCAL_CURR_STREAM% LEQ %TPCH_NUM_EXEC_STREAMS% goto outerLoopExecCreate
endlocal
```

Create-RF2_Whole.cmd

```
setlocal
set TPCH_NUM_LOAD_STREAMS=%1
set TPCH_NUM_LOAD_FILES=%2
set TPCH_NUM_EXEC_STREAMS=%3
set TPCH_NUM_EXEC_TOTAL=%4
set TPCH_EXEC_BATCH_SIZE=%5
set TPCH_FF_LOCN=%6
set TPCH_RF_STREAM_NUM=%7
set /a TPCH_UPDATESET=%7+1
set TPCH_RUN_NUM=%8
set TPCH_COMMENT_STR=%9
set TPCH_FF_LOCN=%TPCH_FF_LOCN%\%TPCH_UPDATESET%\rf128
set TPCH_OUTPUT_DIR=%TPCH_OUTPUT_DIR_PREFIX%\%TPCH_RUN_NUM%
set TPCH_MASTERLOG_FILE=%TPCH_OUTPUT_DIR%\%TPCH_MASTERLOG_NAME%
if not exist %TPCH_OUTPUT_DIR% md %TPCH_OUTPUT_DIR%
pushd %TPCH_OUTPUT_DIR%
echo :setvar totalSegments %TPCH_NUM_LOAD_FILES% > Create_RF2_Tables.sql
type %TPCH_AUTOMATION_SQL%\RF_templ\Create_RF2_Tables.sql  >> Create_RF2_Tables.sql
if not exist Increment_Update_Set.sql copy %TPCH_AUTOMATION_SQL%\RF_templ\Increment_Update_Set.sql
if not exist Create_RF2_Indices.sql copy %TPCH_AUTOMATION_SQL%\RF_templ\Create_RF2_Indices.sql
set /A LOCAL_CURR_STREAM = 1
:outerLoopLoadCreate
set /A LOCAL_CURR_LOAD = %LOCAL_CURR_STREAM%
if exist (RF2_Load_List_%TPCH_RF_STREAM_NUM%.%LOCAL_CURR_STREAM%.txt) del
RF2_Load_List_%TPCH_RF_STREAM_NUM%.%LOCAL_CURR_STREAM%.txt
echo :setvar ffLoc %TPCH_FF_LOCN%  > load_RF2_input.%TPCH_RF_STREAM_NUM%.%LOCAL_CURR_LOAD%.sql
echo :setvar partNo %LOCAL_CURR_LOAD% >> load_RF2_input.%TPCH_RF_STREAM_NUM%.%LOCAL_CURR_LOAD%.sql
type %TPCH_AUTOMATION_SQL%\RF_templ\Load_RF2_tmpl.sql  >>
load_RF2_input.%TPCH_RF_STREAM_NUM%.%LOCAL_CURR_LOAD%.sql
echo load_RF2_input.%TPCH_RF_STREAM_NUM%.%LOCAL_CURR_LOAD%.sql >>
RF2_Load_List_%TPCH_RF_STREAM_NUM%.%LOCAL_CURR_STREAM%.txt
set /A LOCAL_CURR_LOAD = %LOCAL_CURR_LOAD% + %TPCH_NUM_LOAD_STREAMS%
if %LOCAL_CURR_LOAD% LEQ %TPCH_NUM_LOAD_FILES% goto innerLoopLoadCreate
set /A LOCAL_CURR_STREAM = %LOCAL_CURR_STREAM% + 1
if %LOCAL_CURR_STREAM% LEQ %TPCH_NUM_LOAD_STREAMS% goto outerLoopLoadCreate
set /A LOCAL_CURR_STREAM = 1
:outerLoopExecCreate
set /A LOCAL_CURR_EXEC = %LOCAL_CURR_STREAM%
if exist (RF2_Execute_List_%TPCH_RF_STREAM_NUM%.%LOCAL_CURR_STREAM%.txt) del
RF2_Execute_List_%TPCH_RF_STREAM_NUM%.%LOCAL_CURR_STREAM%.txt
:innerLoopExecCreate
echo :setvar currExec %LOCAL_CURR_EXEC%                    >
exec_RF2_input.%TPCH_RF_STREAM_NUM%.%LOCAL_CURR_EXEC%.sql
echo :setvar batchSize %TPCH_EXEC_BATCH_SIZE%       >>
exec_RF2_input.%TPCH_RF_STREAM_NUM%.%LOCAL_CURR_EXEC%.sql
echo :setvar numParallelExecs %TPCH_NUM_EXEC_STREAMS%  >>
exec_RF2_input.%TPCH_RF_STREAM_NUM%.%LOCAL_CURR_EXEC%.sql
echo :setvar totalExecs %TPCH_NUM_EXEC_TOTAL%       >>
exec_RF2_input.%TPCH_RF_STREAM_NUM%.%LOCAL_CURR_EXEC%.sql
type %TPCH_AUTOMATION_SQL%\RF_templ\Execute_RF2_tmpl.sql>>
exec_RF2_input.%TPCH_RF_STREAM_NUM%.%LOCAL_CURR_EXEC%.sql
echo exec_RF2_input.%TPCH_RF_STREAM_NUM%.%LOCAL_CURR_EXEC%.sql >>
RF2_Execute_List_%TPCH_RF_STREAM_NUM%.%LOCAL_CURR_STREAM%.txt
set /A LOCAL_CURR_EXEC = %LOCAL_CURR_EXEC% + %TPCH_NUM_EXEC_STREAMS%
if %LOCAL_CURR_EXEC% LEQ %TPCH_NUM_EXEC_TOTAL% goto innerLoopExecCreate
set /A LOCAL_CURR_STREAM = %LOCAL_CURR_STREAM% + 1
if %LOCAL_CURR_STREAM% LEQ %TPCH_NUM_EXEC_STREAMS% goto outerLoopExecCreate
endlocal
```

Create-RF_Streams.cmd

```
setlocal
set TPCH_NUM_LOAD_STREAMS=%1
set TPCH_NUM_LOAD_FILES=%2
set TPCH_NUM_EXEC_STREAMS=%3
set TPCH_NUM_EXEC_TOTAL=%4
set TPCH_EXEC_BATCH_SIZE=%5
set TPCH_FF_LOCN=%6
set TPCH_NUM_RF_STREAMS=%7
set TPCH_RUN_NUM=%8
set TPCH_SWITCH_TO_RUN2=%9
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
set TPCH_OUTPUT_DIR=%TPCH_OUTPUT_DIR_PREFIX%\%TPCH_RUN_NUM%
set TPCH_MASTERLOG_FILE=%TPCH_OUTPUT_DIR%\%TPCH_MASTERLOG_NAME%
if not exist %TPCH_OUTPUT_DIR% md %TPCH_OUTPUT_DIR%
pushd %TPCH_OUTPUT_DIR%
set /A LOCAL_CURR_STREAM = 0
:outerRFLoop
if %LOCAL_CURR_STREAM% GTR %TPCH_SWITCH_TO_RUN2% set TPCH_RUN_NUM=Run2
call %TPCH_AUTOMATION_PATH%\CreateFiles\Create-RF1_Whole.cmd %TPCH_NUM_LOAD_STREAMS%
%TPCH_NUM_LOAD_FILES% %TPCH_NUM_EXEC_STREAMS% %TPCH_NUM_EXEC_TOTAL% %TPCH_EXEC_BATCH_SIZE%
%TPCH_FF_LOCN% %LOCAL_CURR_STREAM% %TPCH_RUN_NUM% Throughput_RF1_Stream%LOCAL_CURR_STREAM%
call %TPCH_AUTOMATION_PATH%\CreateFiles\Create-RF2_Whole.cmd %TPCH_NUM_LOAD_STREAMS%
%TPCH_NUM_LOAD_FILES% %TPCH_NUM_EXEC_STREAMS% %TPCH_NUM_EXEC_TOTAL% %TPCH_EXEC_BATCH_SIZE%
%TPCH_FF_LOCN% %LOCAL_CURR_STREAM% %TPCH_RUN_NUM% Throughput_RF2_Stream%LOCAL_CURR_STREAM%
set /A LOCAL_CURR_STREAM = %LOCAL_CURR_STREAM% + 1
if %LOCAL_CURR_STREAM% LEQ %TPCH_NUM_RF_STREAMS% goto outerRFLoop
endlocal
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cArrConstraints"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:      cArrConstraints.cls
'             Microsoft TPC-H Kit Ver. 1.00
'             Copyright Microsoft, 1999
'             All Rights Reserved
'
'
'  PURPOSE:    Implements an array of
cConstraint objects.
'             Type-safe wrapper around
cNodeCollections.
'             Also contains additional functions
that determine all the
'             constraints for a step, all
constraints in a workspace,
'             validation functions, etc.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

Private mcarrConstraints As
cNodeCollections

' Used to indicate the source module name
when errors
' are raised by this class
Private mstrSource As String
Private Const mstrModuleName As String =
"cArrConstraints."
Public Sub SaveWspConstraints(ByVal
lngWorkspace As Long)
   ' Calls a procedure to commit all changes
to the constraints
   ' in the passed in workspace.

   Call
mcarrConstraints.Save(lngWorkspace)

End Sub
Public Property Set ConstraintDB(vdata As
Database)

   Set mcarrConstraints.NodeDB = vdata

End Property
Public Property Get ConstraintDB() As
Database

   Set ConstraintDB =
mcarrConstraints.NodeDB

End Property

Public Sub Modify(cConsToUpdate As
cConstraint)

   ' Modify the constraint record
   Call
mcarrConstraints.Modify(cConsToUpdate)

End Sub
Public Sub
CreateNewConstraintVersion(ByVal lngStepId
As Long, _
      ByVal strNewVersion As String, _
      ByVal strOldVersion As String, _
      ByVal intStepType As Integer)

   ' Does all the processing needed to create
new versions of
   ' all the constraints for a given step
   ' It inserts new constraint records in the
database with
   ' the new version numbers on them
   ' It also updates the version number on all
constraints
   ' for the step in the array to the new version
passed in
   ' Since it handles both global and
manager/worker steps,
   ' it checks for the step_id or global_step_id
fields,
   ' depending on the type of step

   Dim lngIndex As Long
   Dim cUpdateConstraint As cConstraint

   On Error GoTo
CreateNewConstraintVersionErr
   mstrSource = mstrModuleName &
"CreateNewConstraintVersion"

   ' Update the version/global version on
Constraint with the
   ' passed in step/global step id
   For lngIndex = 0 To mcarrConstraints.Count
- 1
      Set cUpdateConstraint =
mcarrConstraints(lngIndex)
      If intStepType = gintGlobalStep Then
         If cUpdateConstraint.GlobalStepId =
lngStepId And _
               cUpdateConstraint.IndOperation
<> DeleteOp Then
            cUpdateConstraint.GlobalVersionNo
= strNewVersion

            ' Set the operation to indicate an
insert
            cUpdateConstraint.IndOperation =
InsertOp
         End If
      Else
         If cUpdateConstraint.StepId =
lngStepId And _
               cUpdateConstraint.IndOperation
<> DeleteOp Then
            cUpdateConstraint.VersionNo =
strNewVersion

            ' Set the operation to indicate an
insert
            cUpdateConstraint.IndOperation =
InsertOp
         End If
      End If
   Next lngIndex

   Exit Sub
CreateNewConstraintVersionErr:
   LogErrors Errors
   gstrSource = mstrModuleName &
"CreateNewConstraintVersion"
   On Error GoTo 0
   Err.Raise vbObjectError +
errCreateNewConstraintVersionFailed, _
         mstrSource, _

LoadResString(errCreateNewConstraintVersionFail
ed)

End Sub
Private Sub Class_Initialize()

   Set mcarrConstraints = New cNodeCollections
   BugMessage "cArrConstraints: Initialize event -
setting Constraint count to 0"

End Sub

Private Sub Class_Terminate()

   Set mcarrConstraints = Nothing
   BugMessage "cArrConstraints: Terminate event
triggered"

End Sub

Public Sub Add(ByVal cConstraintToAdd As
cConstraint)

   Set cConstraintToAdd.NodeDB =
mcarrConstraints.NodeDB

   ' Retrieve a unique constraint identifier
   cConstraintToAdd.ConstraintId =
cConstraintToAdd.NextIdentifier

   ' Call a procedure to load the constraint record in
the array
   Call mcarrConstraints.Add(cConstraintToAdd)

End Sub
Public Sub Delete(ByVal cOldConstraint As
cConstraint)

   Dim lngDeleteElement As Long
   Dim cConsToDelete As cConstraint

   lngDeleteElement =
QueryConstraintIndex(cOldConstraint.ConstraintId)
   Set cConsToDelete =
mcarrConstraints(lngDeleteElement)

   Call
mcarrConstraints.Delete(cConsToDelete.Position)

   Set cConsToDelete = Nothing

End Sub
Private Function
QueryConstraintIndex(lngConstraintId As Long) _
      As Long

   Dim lngIndex As Integer

   ' Find the element in the array to be deleted
   For lngIndex = 0 To mcarrConstraints.Count - 1

      ' Note: The constraint id is not a primary key
field in
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
        ' the database - there can be multiple
records with the
        ' same constraint_id but for different
versions of a step
        ' However, since we'll always load the
constraint information
        ' for the latest version of a step, we'll
have just one
        ' constraint record with a given
constraint_id
        If
mcarrConstraints(lngIndex).ConstraintId =
lngConstraintId Then
            QueryConstraintIndex = lngIndex
            Exit Function
        End If

    Next lngIndex

    ' Raise error that Constraint has not been
found
    ShowError errConstraintNotFound
    On Error GoTo 0
    Err.Raise vbObjectError +
errConstraintNotFound, mstrSource, _
        LoadResString(errConstraintNotFound)

End Function

Public Function QueryConstraint(ByVal
lngConstraintId As Long) _
    As cConstraint

    ' Returns a cConstraint object with the
property values
    ' corresponding to the Constraint
Identifier, lngConstraintId

    Dim lngQueryElement As Long

    lngQueryElement =
QueryConstraintIndex(lngConstraintId)

    ' Set the return value to the queried
Constraint
    Set QueryConstraint =
mcarrConstraints(lngQueryElement)

End Function

Public Sub LoadConstraints(ByVal
lngWorkspaceId As Long, rstStepsInWsp
As Recordset)

    ' Loads the constraints array with all the
constraints
    ' for the workspace
    Dim recConstraints As Recordset
    Dim qyCons As DAO.QueryDef
    Dim strSql As String
    Dim dtStart As Date

    On Error GoTo LoadConstraintsErr
    mstrSource = mstrModuleName &
"LoadConstraints"

    If rstStepsInWsp.RecordCount = 0 Then
        Exit Sub
    End If

    ' First check if the database object has
been set
    If mcarrConstraints.NodeDB Is Nothing
Then
        On Error GoTo 0
        Err.Raise vbObjectError +
errSetDBBeforeLoad, _
                mstrSource, _

LoadResString(errSetDBBeforeLoad)
    End If

    dtStart = Now

    ' Select based on the global step id since
there might
    ' be constraints for a global step that run are
executed
    ' for the workspace
    ' This method has the advantage that if the
steps are queried right, everything else follows
    strSql = "Select a.constraint_id, a.step_id,
a.version_no, " & _
        " a.constraint_type, a.global_step_id,
a.global_version_no, " & _
        " a.sequence_no, b.workspace_id " & _
        " from step_constraints a, att_steps b " &
_
        " where a.global_step_id = b.step_id " &
_
        " and a.global_version_no = b.version_no
" & _
        " and a.global_step_id = [g_s_id] " & _
        " and a.global_version_no = [g_ver_no] "
& _
        " and b.archived_flag = [archived] "

    ' Find the highest X-component of the
version number
    strSql = strSql & " AND ( a.step_id = 0 or (
cint( mid( a.version_no, 1, instr( a.version_no,
" & gstrDQ & gstrVerSeparator & gstrDQ & "
) - 1 ) ) = " & _
        " ( select max( cint( mid( version_no, 1,
instr( version_no, " & gstrDQ &
gstrVerSeparator & gstrDQ & " ) - 1 ) ) ) " & _
        " from att_steps AS d " & _
        " WHERE a.step_id = d.step_id " & _
        " and d.archived_flag = [archived] ) "

    ' Find the highest Y-component of the
version number for the highest X-component
    strSql = strSql & " AND cint( mid(
a.version_no, instr( a.version_no, " & gstrDQ
& gstrVerSeparator & gstrDQ & " ) + 1 ) ) = "
& _
        " ( select max( cint( mid( version_no,
instr( version_no, " & gstrDQ &
gstrVerSeparator & gstrDQ & " ) + 1 ) ) ) " &
_
        " from att_steps AS y " & _
        " Where a.step_id = y.step_id " & _
        " AND cint( mid( version_no, 1, instr(
version_no, " & gstrDQ & gstrVerSeparator &
gstrDQ & " ) - 1 ) ) = " & _
        " ( select max( cint( mid( version_no, 1,
instr( version_no, " & gstrDQ &
gstrVerSeparator & gstrDQ & " ) - 1 ) ) ) " & _
        " from att_steps AS c " & _
        " WHERE y.step_id = c.step_id " & _
        " and c.archived_flag = [archived] ) ) ) ) "

    ' Order the constraints by sequence within a
given step
    strSql = strSql & " order by a.sequence_no "

    Set qyCons =
mcarrConstraints.NodeDB.CreateQueryDef(gst
rEmptyString, strSql)
    qyCons.Parameters("archived").Value = False

    rstStepsInWsp.MoveFirst

    While Not rstStepsInWsp.EOF

        If Not (rstStepsInWsp!global_flag) Then
            qyCons.Close
            BugMessage "Query constraints Read +
load took: " & CStr(DateDiff("s", dtStart, Now))
            Exit Sub
        End If

        qyCons.Parameters("g_s_id").Value =
rstStepsInWsp!step_id
        qyCons.Parameters("g_ver_no").Value =
rstStepsInWsp!version_no

        Set recConstraints =
qyCons.OpenRecordset(dbOpenSnapshot)

        Call
LoadRecordsetInConstraintArray(recConstraints)
        recConstraints.Close

        rstStepsInWsp.MoveNext
    Wend

    qyCons.Close
    BugMessage "Query constraints Read + load
took: " & CStr(DateDiff("s", dtStart, Now))

    Exit Sub

LoadConstraintsErr:
    LogErrors Errors
    gstrSource = mstrModuleName &
"LoadConstraints"
    On Error GoTo 0
    Err.Raise vbObjectError + errLoadDataFailed, _
            mstrSource, _
            LoadResString(errLoadDataFailed)

End Sub
Public Sub UnloadStepConstraints(ByVal lngStepId
As Long)

    ' Unloads all the constraints for the workspace
from
    ' the constraints array

    Dim lngIndex As Long

    ' Find all constraints in the array with a matching
step id
    ' It is important to step in reverse order through
the array,
    ' since we delete constraint records!
    For lngIndex = mcarrConstraints.Count - 1 To 0
Step -1
        If mcarrConstraints(lngIndex).GlobalStepId =
lngStepId Then

            ' Unload the constraint from the array
            Call mcarrConstraints.Unload(lngIndex)

        End If
    Next lngIndex

End Sub
Public Sub UnloadConstraint(cOldConstraint As
cConstraint)
    ' Unloads the constraint from the constraints array

    Dim lngDeleteElement As Long
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   lngDeleteElement =
QueryConstraintIndex(cOldConstraint.Const
raintId)

   Call
mcarrConstraints.Unload(lngDeleteElement)

End Sub
Private Sub
LoadRecordsetInConstraintArray(ByVal
recConstraints As Recordset)
   ' Loads all the constraint records in the
passed in
   ' recordset into the array

   Dim cNewConstraint As cConstraint

   On Error GoTo
LoadRecordsetInConsArrayErr
   mstrSource = mstrModuleName &
"LoadRecordsetInConstraintArray"

   If recConstraints.RecordCount = 0 Then
      Exit Sub
   End If

   recConstraints.MoveFirst
   While Not recConstraints.EOF
      Set cNewConstraint = New cConstraint

      ' Initialize Constraint values
      cNewConstraint.ConstraintId =
CLng(ErrorOnNullField(recConstraints,
"Constraint_id"))
      cNewConstraint.StepId =
CLng(ErrorOnNullField(recConstraints,
"step_id"))
      cNewConstraint.VersionNo =
CStr(ErrorOnNullField(recConstraints,
"version_no"))

      cNewConstraint.GlobalStepId =
CLng(ErrorOnNullField(recConstraints,
"global_step_id"))
      cNewConstraint.GlobalVersionNo =
CStr(ErrorOnNullField(recConstraints,
"global_version_no"))
      cNewConstraint.SequenceNo =
CInt(ErrorOnNullField(recConstraints,
"sequence_no"))

      cNewConstraint.WorkspaceId =
CLng(ErrorOnNullField(recConstraints,
FLD_ID_WORKSPACE))
      cNewConstraint.ConstraintType =
CInt(ErrorOnNullField(recConstraints,
"constraint_type"))

      ' Add this record to the array of
Constraints
      mcarrConstraints.Load cNewConstraint

      Set cNewConstraint = Nothing
      recConstraints.MoveNext
   Wend

   Exit Sub

LoadRecordsetInConsArrayErr:
   LogErrors Errors
   gstrSource = mstrModuleName &
"LoadRecordsetInConstraintArray"
   On Error GoTo 0
   Err.Raise vbObjectError +
errLoadRsInArrayFailed, _
         mstrSource, _

LoadResString(errLoadRsInArrayFailed)

End Sub

Public Function ConstraintsForStep( _
      ByVal lngStepId As Long, _
      ByVal strVersionNo As String, _
      Optional ByVal intConstraintType As
ConstraintType = 0, _
      Optional ByVal blnSort As Boolean =
True, _
      Optional ByVal blnGlobal As Boolean =
False, _
      Optional ByVal
blnGlobalConstraintsOnly As Boolean = False)
_
       As Variant

   ' Returns a variant containing an array of
cConstraint objects,
   ' containing all the constraints that have been
defined for the
   ' given step. If the Global flag is set to true,
the
   ' search will be made for all the constraints
that have
   ' a matching global_step_id

   Dim lngIndex As Long
   Dim cStepConstraint() As cConstraint
   Dim lngConstraintCount As Long
   Dim cTempConstraint As cConstraint

   On Error GoTo ConstraintsForStepErr
   mstrSource = mstrModuleName &
"ConstraintsForStep"

   lngConstraintCount = 0

   ' Find each element in the constraints array
   For lngIndex = 0 To mcarrConstraints.Count
- 1
      ' If a constraint type has been specified
then check
      ' if the constraint type for the record
matches the
      ' passed in type
      Set cTempConstraint =
mcarrConstraints(lngIndex)
      If Not blnGlobal Then
         If cTempConstraint.StepId = lngStepId
And _
            cTempConstraint.VersionNo =
strVersionNo And _
            cTempConstraint.IndOperation <>
DeleteOp And _
            (intConstraintType = 0 Or _
            cTempConstraint.ConstraintType
= intConstraintType) Then
            ' We have a matching constraint for
the given step
            AddArrayElement cStepConstraint, _
                cTempConstraint,
lngConstraintCount
         End If
      Else
         If cTempConstraint.GlobalStepId =
lngStepId And _

cTempConstraint.GlobalVersionNo =
strVersionNo And _
            cTempConstraint.IndOperation <>
DeleteOp Then
            If blnGlobalConstraintsOnly = False Or _
                (blnGlobalConstraintsOnly And _
                cTempConstraint.StepId = 0 And _
                cTempConstraint.VersionNo =
gstrMinVersion) Then

            ' We have a matching constraint for the
global step
            AddArrayElement cStepConstraint, _
                cTempConstraint,
lngConstraintCount
         End If
      End If
   End If

   Next lngIndex

   ' Set the return value of the function to the array
of
   ' constraints that has been built above
   If lngConstraintCount = 0 Then
      ConstraintsForStep = Empty
   Else
      ConstraintsForStep = cStepConstraint()
   End If

   ' Sort the constraints
   If blnSort Then
      Call QuickSort(ConstraintsForStep)
   End If

   Exit Function

ConstraintsForStepErr:
   LogErrors Errors
   On Error GoTo 0
   Err.Raise vbObjectError +
errConstraintsForStepFailed, _
         mstrSource, _

LoadResString(errConstraintsForStepFailed)

End Function
Private Sub AddArrayElement(ByRef arrNodes()
As cConstraint, _
      ByVal objToAdd As cConstraint, _
      ByRef lngCount As Long)
   ' Adds the passed in object to the array

   ' Increase the array dimension and add the object
to it
   ReDim Preserve arrNodes(lngCount)
   Set arrNodes(lngCount) = objToAdd
   lngCount = lngCount + 1

End Sub

Public Function ConstraintsForWsp( _
      ByVal lngWorkspaceId As Long, _
      Optional ByVal intConstraintType As Integer
= 0, _
      Optional ByVal blnSort As Boolean = True, _
      Optional ByVal blnGlobalConstraintsOnly As
Boolean = False) _
       As Variant

   ' Returns a variant containing an array of
cConstraint objects,
   ' containing all the constraints that have been
defined for the
   ' given workspace.

   Dim lngIndex As Long
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   Dim cWspConstraint() As cConstraint
   Dim lngConstraintCount As Long
   Dim cTempConstraint As cConstraint

   On Error GoTo ConstraintsForWspErr
   mstrSource = mstrModuleName &
"ConstraintsForWsp"

   lngConstraintCount = 0

   ' Find each element in the constraints
array
   For lngIndex = 0 To
mcarrConstraints.Count - 1
      ' If a constraint type has been specified
then check
      ' if the constraint type for the record
matches the
      ' passed in type
      Set cTempConstraint =
mcarrConstraints(lngIndex)
      If cTempConstraint.WorkspaceId =
lngWorkspaceId And _
            cTempConstraint.IndOperation <>
DeleteOp And _
            (intConstraintType = 0 Or _
            cTempConstraint.ConstraintType
= intConstraintType) Then

         If blnGlobalConstraintsOnly = False
Or _
               (blnGlobalConstraintsOnly And
_
               cTempConstraint.StepId = 0
And _
               cTempConstraint.VersionNo =
gstrMinVersion) Then

            ' We have a matching constraint
for the workspace
            AddArrayElement
cWspConstraint, _
                  cTempConstraint,
lngConstraintCount
         End If
      End If
   Next lngIndex

   ' Set the return value of the function to the
array of
   ' constraints that has been built above
   If lngConstraintCount = 0 Then
      ConstraintsForWsp = Empty
   Else
      ConstraintsForWsp = cWspConstraint()
   End If

   ' Sort the constraints
   If blnSort Then
      Call QuickSort(ConstraintsForWsp)
   End If

   Exit Function

ConstraintsForWspErr:
   LogErrors Errors
   On Error GoTo 0
   Err.Raise vbObjectError +
errConstraintsForWspFailed, _
         mstrSource, _

LoadResString(errConstraintsForWspFailed
)

End Function
Public Function PreConstraintsForStep( _
      ByVal lngStepId As Long, _
      ByVal strVersionNo As String, _
      Optional ByVal blnSort As Boolean) As
Variant

   ' Returns a variant containing an array of
cConstraint objects,
   ' containing all the pre-execution constraints
that have
   ' been defined for the given step_id and
version

   ' Call a function that will return a variant
containing
   ' all the constraints of the passed in type
   PreConstraintsForStep =
ConstraintsForStep(lngStepId, _
         strVersionNo, gintPreStep, blnSort)

End Function
Public Function PostConstraintsForStep( _
      ByVal lngStepId As Long, _
      ByVal strVersionNo As String, _
      Optional ByVal blnSort As Boolean) As
Variant

   ' Returns a variant containing an array of
cConstraint objects,
   ' containing all the Post-execution
constraints that have
   ' been defined for the given step_id and
version

   ' Call a function that will return a variant
containing
   ' all the constraints of the passed in type
   PostConstraintsForStep =
ConstraintsForStep(lngStepId, _
         strVersionNo, gintPostStep, blnSort)

End Function
Public Function PostConstraintsForWsp( _
      ByVal lngWorkspaceId As Long, _
      Optional ByVal blnSort As Boolean) As
Variant

   ' Returns a variant containing an array of
cConstraint objects,
   ' containing all the Post-execution globals
that have
   ' been defined for the workspace

   ' Call a function that will return a variant
containing
   ' all the constraints of the passed in type
   PostConstraintsForWsp =
ConstraintsForWsp(lngWorkspaceId, _
         gintPostStep, blnSort, True)

End Function
Public Function PreConstraintsForWsp( _
      ByVal lngWorkspaceId As Long, _
      Optional ByVal blnSort As Boolean) As
Variant

   ' Returns a variant containing an array of
cConstraint objects,
   ' containing all the Pre-execution globals that
have
   ' been defined for the workspace

   ' Call a function that will return a variant
containing
   ' all the constraints of the passed in type
   PreConstraintsForWsp =
ConstraintsForWsp(lngWorkspaceId, _
         gintPreStep, blnSort, True)

End Function
Public Property Get ConstraintCount() As Long

   ConstraintCount = mcarrConstraints.Count

End Property

VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cArrParameters"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:      cArrParameters.cls
'            Microsoft TPC-H Kit Ver. 1.00
'            Copyright Microsoft, 1999
'            All Rights Reserved
'
'
'  PURPOSE:    Implements an array of cParameter
objects.
'            Type-safe wrapper around
cNodeCollections.
'            Also contains additional functions to
determine parameter
'            values, validation functions, etc.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

Private mcarrParameters As cNodeCollections

' Used to indicate the source module name when
errors
' are raised by this class
Private mstrSource As String
Private Const mstrModuleName As String =
"cArrParameters."

Public Sub InitBuiltInsForRun(lWspId As Long,
lRunId As Long)

   Dim cParamRec As cParameter

   ' Initialize the values of the run_id and output_dir
built-in parameters and save them
   ' to the database
   Set cParamRec = GetParameterValue(lWspId,
PARAM_RUN_ID)
   cParamRec.ParameterValue = CStr(lRunId)
   Call Modify(cParamRec)

   Set cParamRec = GetParameterValue(lWspId,
PARAM_OUTPUT_DIR)
   cParamRec.ParameterValue =
GetDefaultDir(lWspId, Me)
   cParamRec.ParameterValue =
cParamRec.ParameterValue & gstrFileSeparator &
CStr(lRunId)
   Call Modify(cParamRec)

   Call SaveParametersInWsp(lWspId)

End Sub
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
Public Property Set ParamDatabase(vdata
As Database)

   Set mcarrParameters.NodeDB = vdata

End Property
Public Sub Modify(cModifiedParam As
cParameter)

   ' First check if the parameter record is
valid
   Call
CheckDupParamName(cModifiedParam)

   Call
mcarrParameters.Modify(cModifiedParam)

End Sub
Public Sub Load(ByRef cParamToAdd As
cParameter)

   Call
mcarrParameters.Load(cParamToAdd)

End Sub
Public Sub Add(ByRef cParamToAdd As
cParameter)

   Set cParamToAdd.NodeDB =
mcarrParameters.NodeDB

   ' First check if the parameter record is
valid
   Call Validate(cParamToAdd)

   ' Retrieve a unique parameter identifier
   cParamToAdd.ParameterId =
cParamToAdd.NextIdentifier

   Call
mcarrParameters.Add(cParamToAdd)

End Sub

Public Sub Unload(lngParamToDelete As
Long)

   Dim lngDeleteElement As Long

   lngDeleteElement =
QueryIndex(lngParamToDelete)

   Call
mcarrParameters.Unload(lngDeleteElement)

End Sub

Public Sub SaveParametersInWsp(ByVal
lngWorkspace As Long)
   ' Calls a procedure to commit all changes
to the parameters
   ' for the passed in workspace.

   ' Call a procedure to save all parameter
records for the
   ' workspace
   Call
mcarrParameters.Save(lngWorkspace)

End Sub
Public Function GetParameterValue(ByVal
lngWorkspace As Long, _
      ByVal strParamName As String) As
cParameter

   ' Returns the value for the passed in
workspace parameter

   Dim cParamRec As cParameter
   Dim lngIndex As Long

   On Error GoTo GetParameterValueErr

   ' Find all parameters in the array with a
matching workspace id
   For lngIndex = 0 To mcarrParameters.Count
- 1
      Set cParamRec =
mcarrParameters(lngIndex)
      If cParamRec.WorkspaceId =
lngWorkspace And _
            cParamRec.ParameterName =
strParamName Then

         Set GetParameterValue = cParamRec
         Exit For
      End If
   Next lngIndex

   If lngIndex > mcarrParameters.Count - 1
Then
      ' The parameter has not been defined for
the workspace
      ' Raise an error
      On Error GoTo 0
      Err.Raise vbObjectError +
errParamNameInvalid, _
            mstrModuleName &
"GetParameterValue", _

LoadResString(errParamNameInvalid)
   End If

   Exit Function

GetParameterValueErr:
   ' Log the error code raised by Visual Basic
   Call LogErrors(Errors)
   gstrSource = mstrModuleName &
"GetParameterValue"
   On Error GoTo 0
   Err.Raise vbObjectError +
errGetParamValueFailed, _
         gstrSource, _

LoadResString(errGetParamValueFailed)

End Function
Public Sub Delete(lngParamToDelete As
Long)
   ' Delete the passed in parameter

   Dim lngDeleteElement As Long

   lngDeleteElement =
QueryIndex(lngParamToDelete)
   Call
mcarrParameters.Delete(lngDeleteElement)

End Sub
Private Function QueryIndex(lngParameterId
As Long) As Long

   Dim lngIndex As Long

   ' Find the matching parameter record in the
array
   For lngIndex = 0 To mcarrParameters.Count
- 1
      If mcarrParameters(lngIndex).ParameterId =
lngParameterId And _
            mcarrParameters(lngIndex).IndOperation
<> DeleteOp Then
         QueryIndex = lngIndex
         Exit Function
      End If
   Next lngIndex

   ' Raise error that parameter has not been found
   On Error GoTo 0
   Err.Raise vbObjectError + errParamNotFound,
"cArrParameters.QueryIndex", _
      LoadResString(errParamNotFound)

End Function

Public Function QueryParameter(lngParameterId
As Long) _
      As cParameter

   Dim lngQueryElement As Long

   lngQueryElement = QueryIndex(lngParameterId)

   ' Return the queried parameter object
   Set QueryParameter =
mcarrParameters(lngQueryElement)

End Function
Public Property Get ParameterCount() As Long

   ParameterCount = mcarrParameters.Count

End Property
Public Property Get Item(lngIndex As Long) As
cParameter
Attribute Item.VB_UserMemId = 0

   Set Item = mcarrParameters(lngIndex)

End Property

Public Sub Validate(ByVal cParamToValidate As
cParameter)
   ' This procedure is necessary since the class
cannot validate
   ' all the parameter properties on it's own. This is
'coz we
   ' might have created new parameters in the
workspace, but not
   ' saved them to the database yet - hence the
duplicate check
   ' has to be repeated in the array

   Dim lngIndex As Long
   Dim cTempParam As cParameter

   On Error GoTo ValidateErr

   ' Check if the parameter name already exists in
the workspace
   For lngIndex = 0 To mcarrParameters.Count - 1
      Set cTempParam = mcarrParameters(lngIndex)
      If cTempParam.WorkspaceId =
cParamToValidate.WorkspaceId And _
            cTempParam.ParameterName =
cParamToValidate.ParameterName And _
            cTempParam.IndOperation <> DeleteOp
Then
         On Error GoTo 0
         Err.Raise vbObjectError +
errDuplicateParameterName, _
            mstrSource,
LoadResString(errDuplicateParameterName)
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
      End If
   Next lngIndex

   Exit Sub

ValidateErr:
   LogErrors Errors
   mstrSource = mstrModuleName &
"Validate"
   On Error GoTo 0
   Err.Raise vbObjectError +
errValidateFailed, _
      mstrSource,
LoadResString(errValidateFailed)

End Sub
Public Sub CheckDupParamName(ByVal
cParamToValidate As cParameter)

   Dim lngIndex As Long
   Dim cTempParam As cParameter

   ' Check if the parameter name already
exists in the workspace
   For lngIndex = 0 To
mcarrParameters.Count - 1
      Set cTempParam =
mcarrParameters(lngIndex)
      If cTempParam.WorkspaceId =
cParamToValidate.WorkspaceId And _
            cTempParam.ParameterName =
cParamToValidate.ParameterName And _
            cTempParam.ParameterId <>
cParamToValidate.ParameterId And _
            cTempParam.IndOperation <>
DeleteOp Then
         ShowError
errDuplicateParameterName
         On Error GoTo 0
         Err.Raise vbObjectError +
errDuplicateParameterName, _
            mstrSource,
LoadResString(errDuplicateParameterName
)
      End If
   Next lngIndex

End Sub

Private Sub Class_Initialize()

   'bugmessage "cArrParameters: Initialize
event - setting parameter count to 0"
   Set mcarrParameters = New
cNodeCollections

End Sub


Private Sub Class_Terminate()

   Set mcarrParameters = Nothing

End Sub
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cArrSteps"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:      cArrSteps.cls
'            Microsoft TPC-H Kit Ver. 1.00
'            Copyright Microsoft, 1999
'            All Rights Reserved
'
'
'  PURPOSE:    Implements an array of cStep
objects.
'            Type-safe wrapper around
cNodeCollections.
'            Also contains additional functions to
update parent version
'            on substeps, validation functions, etc.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

Private mcarrSteps As cNodeCollections

' Used to indicate the source module name
when errors
' are raised by this class
Private mstrSource As String
Private Const mstrModuleName As String =
"cArrSteps."

Public Sub Unload(lngStepToDelete As Long)

   Dim lngDeleteElement As Long
   Dim cUnloadStep As cStep

   lngDeleteElement =
QueryStepIndex(lngStepToDelete)
   Set cUnloadStep =
QueryStep(lngStepToDelete)

   ' First unload all iterators for the step
   Call cUnloadStep.UnloadIterators

   ' Unload the step from the collection
   Call mcarrSteps.Unload(lngDeleteElement)

End Sub
Public Sub Modify(cModifiedStep As cStep)

   Dim iAppend As Integer
   Dim sLabel As String

   On Error GoTo ModifyErr

   iAppend = 0
   sLabel = cModifiedStep.StepLabel

   Validate cModifiedStep

   Call mcarrSteps.Modify(cModifiedStep)

   Exit Sub

ModifyErr:
   ' If the error raised by the add function is due
to a duplication
   ' of the step label, then try to generate a
unique label
   If Err.Number - vbObjectError =
errStepLabelUnique Then
      iAppend = iAppend + 1
      cModifiedStep.StepLabel = sLabel &
CStr(iAppend)
      ' Try to insert the step record again
      Resume
   End If

   ' Log the error code raised by Visual Basic
   Call LogErrors(Errors)
   On Error GoTo 0
   Err.Raise vbObjectError + errModifyStepFailed,
_
         gstrSource, _
         LoadResString(errModifyStepFailed)

End Sub
Public Sub UpdateParentVersion(ByVal lngStepId
As Long, _
      ByVal strNewVersion As String, _
      ByVal strOldVersion As String, _
      ByVal intStepType As Integer)

   ' Does all the processing needed to update the
parent version
   ' number on all the sub-steps for a given step
   ' It updates the parent version no in the database
for all
   ' sub-steps of the passed in step id
   ' It also updates the parent version number on all
sub-steps
   ' in the array to the new version passed in

   Dim lngIndex As Long
   Dim cUpdateStep As cStep

   On Error GoTo UpdateParentVersionErr

   If intStepType <> gintManagerStep Then
      ' Only a manager can have sub-steps - if the
passed
      ' in step is not a manager, exit
      Exit Sub
   End If

   ' For all steps in the array
   For lngIndex = 0 To mcarrSteps.Count - 1

      Set cUpdateStep = mcarrSteps(lngIndex)

      ' If the current step is a sub-step of the passed
in step
      If cUpdateStep.ParentStepId = lngStepId And
_
            cUpdateStep.ParentVersionNo =
strOldVersion And _
            Not cUpdateStep.ArchivedFlag Then

         ' Update the parent version number for the
sub-step
         ' in the array
         cUpdateStep.ParentVersionNo =
strNewVersion

         ' Update the parent version number for the
sub-step
         ' in the array
         Call Modify(cUpdateStep)

      End If
   Next lngIndex

   Exit Sub

UpdateParentVersionErr:
   LogErrors Errors
   mstrSource = mstrModuleName &
"UpdateParentVersion"
   On Error GoTo 0
   Err.Raise vbObjectError +
errUpdateParentVersionFailed, _
         mstrSource, _

LoadResString(errUpdateParentVersionFailed)

End Sub
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
Private Sub Validate(cCheckStep As cStep)

   ' Step validations that depend on other
steps in the collection

   Dim lngIndex As Long

   ' Ensure that the step label is unique in the
workspace
   For lngIndex = 0 To mcarrSteps.Count - 1

      ' If the current step is a sub-step of the
passed in step
      If mcarrSteps(lngIndex).WorkspaceId
= cCheckStep.WorkspaceId And _
            mcarrSteps(lngIndex).StepLabel =
cCheckStep.StepLabel And _
            mcarrSteps(lngIndex).StepId <>
cCheckStep.StepId And _

mcarrSteps(lngIndex).IndOperation <>
DeleteOp Then
         On Error GoTo 0
         Err.Raise vbObjectError +
errStepLabelUnique, _
               mstrModuleName &
"Validate", _

LoadResString(errStepLabelUnique)
      End If
   Next lngIndex

End Sub

Public Sub ValidateStep(cCheckStep As
cStep)

   On Error GoTo ValidateStepErr

   'Public wrapper for Validate function (2
many Validates)
   Call Validate(cCheckStep)

   Exit Sub

ValidateStepErr:
   ShowError errStepLabelUnique
   On Error GoTo 0
   Err.Raise vbObjectError +
errValidateFailed, _
         gstrSource, _
         LoadResString(errValidateFailed)
End Sub

Private Sub Class_Initialize()

   BugMessage "cArrSteps: Initialize event -
setting step count to 0"
   Set mcarrSteps = New cNodeCollections

End Sub

Private Sub Class_Terminate()

   BugMessage "cArrSteps: Terminate event
triggered"
   Set mcarrSteps = Nothing

End Sub

Public Sub Add(ByVal cStepToAdd As
cStep)

   Dim iAppend As Integer
   Dim sLabel As String

   On Error GoTo AddErr

   iAppend = 0
   sLabel = cStepToAdd.StepLabel

   Set cStepToAdd.NodeDB =
mcarrSteps.NodeDB

   ' Retrieve a unique step identifier
   cStepToAdd.StepId =
cStepToAdd.NextStepId

   Validate cStepToAdd

   ' Call a procedure to add the step record
   Call mcarrSteps.Add(cStepToAdd)

   ' Call a procedure to add all iterators for the
step
   cStepToAdd.AddAllIterators

   Exit Sub

AddErr:
   ' If the error raised by the add function is due
to a duplication
   ' of the step label, then try to generate a
unique label
   If Err.Number - vbObjectError =
errStepLabelUnique Then
      iAppend = iAppend + 1
      cStepToAdd.StepLabel = sLabel &
CStr(iAppend)
      ' Try to insert the step record again
      Resume
   End If

   ' Log the error code raised by Visual Basic
   Call LogErrors(Errors)
   On Error GoTo 0
   Err.Raise vbObjectError +
errInsertStepFailed, _
         gstrSource, _
         LoadResString(errInsertStepFailed)

End Sub
Public Sub Load(cStepToLoad As cStep)

   Call mcarrSteps.Load(cStepToLoad)

End Sub
Public Sub SaveStepsInWsp(ByVal
lngWorkspace As Long)
   ' Calls a procedure to commit all changes to
the steps
   ' in the passed in workspace.

   Dim lngIndex As Integer

   ' Find all steps in the array with a matching
workspace id
   ' It is important to step in reverse order
through the array,
   ' since we delete step records sometimes!
   For lngIndex = mcarrSteps.Count - 1 To 0
Step -1
      If mcarrSteps(lngIndex).WorkspaceId =
lngWorkspace Then

         ' Call a procedure to commit all
changes to the
         ' Step record, if any
         Call
CommitStep(mcarrSteps(lngIndex), lngIndex)
      End If
   Next lngIndex

End Sub
Private Sub CommitStep(ByVal cCommitStep As
cStep, _
      ByVal intIndex As Integer)
   ' This procedure checks if any changes have been
made to the
   ' passed in Step. If so, it calls the step methods to
commit
   ' the changes.

   ' First commit all changes to the iterator records
for
   ' the step
   cCommitStep.SaveIterators

   Call mcarrSteps.Commit(cCommitStep, intIndex)

End Sub
Public Sub Delete(lngStepToDelete As Long)

   Dim lngDeleteElement As Long

   lngDeleteElement =
QueryStepIndex(lngStepToDelete)
   Call mcarrSteps.Delete(lngDeleteElement)

End Sub
Public Function QueryStepIndex(lngStepId As
Long) As Long

   Dim lngIndex As Long

   ' Find the element in the array that corresponds to
the
   ' passed in step id - note that while there will be
multiple
   ' versions of a step in the database, only one
version will
   ' be currently loaded in the array - meaning that
the stepid
   ' is enough to uniquely identify a step
   For lngIndex = 0 To mcarrSteps.Count - 1
      If mcarrSteps(lngIndex).StepId = lngStepId
Then
         QueryStepIndex = lngIndex
         Exit Function
      End If
   Next lngIndex

   ' Raise error that step has not been found
   On Error GoTo 0
   Err.Raise vbObjectError + errStepNotFound,
mstrSource, _
      LoadResString(errStepNotFound)

End Function

Public Function QueryStep(ByVal lngStepId As
Long) As cStep

   ' Populates the passed in cStep object with the
property
   ' values corresponding to the Step Identifier,
lngStepId

   Dim lngQueryElement As Integer

   lngQueryElement = QueryStepIndex(lngStepId)

   ' Initialize the passed in step object to the queried
step
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   Set QueryStep =
mcarrSteps(lngQueryElement)

End Function
Public Property Get Item(ByVal Position As
Long) As cStep
Attribute Item.VB_UserMemId = 0

   ' Returns the element at the passed in
position in the array
   If Position >= 0 And Position <
mcarrSteps.Count Then
      Set Item = mcarrSteps(Position)
   Else
      On Error GoTo 0
      Err.Raise vbObjectError +
errItemDoesNotExist, mstrSource, _

LoadResString(errItemDoesNotExist)
   End If

End Property
Public Property Set Item(ByVal Position As
Long, _
      ByVal cStepRec As cStep)

   ' Returns the element at the passed in
position in the array
   If Position >= 0 And Position <
mcarrSteps.Count Then
      Set mcarrSteps(Position) = cStepRec
   Else
      On Error GoTo 0
      Err.Raise vbObjectError +
errItemDoesNotExist, mstrSource, _

LoadResString(errItemDoesNotExist)
   End If

End Property
Public Property Set StepDB(vdata As
Database)

   Set mcarrSteps.NodeDB = vdata

End Property
Public Function SubSteps(ByVal lngStepId
As Long, _
      ByVal strVersionNo As String) As
Variant

   ' Returns a variant containing an array of
all the substeps
   ' for the passed in step

   Dim intIndex As Integer
   Dim cSubSteps() As cStep
   Dim lngStepCount As Long
   Dim cQueryStep As cStep

   On Error GoTo SubStepsErr

   lngStepCount = 0

   Set cQueryStep = QueryStep(lngStepId)

   ' Only a manager can have sub-steps
   If cQueryStep.StepType =
gintManagerStep Then

      ' For each element in the Steps array
      For intIndex = 0 To mcarrSteps.Count -
1
         ' Check if the parent step id and
parent version number
         ' match the passed in step
         If mcarrSteps(intIndex).ParentStepId =
lngStepId And _

mcarrSteps(intIndex).ParentVersionNo =
strVersionNo And _

mcarrSteps(intIndex).IndOperation <>
DeleteOp Then

            ' Increase the array dimension and
add the step
            ' to it
            ReDim Preserve
cSubSteps(lngStepCount)
            Set cSubSteps(lngStepCount) =
mcarrSteps(intIndex)
            lngStepCount = lngStepCount + 1

         End If
      Next intIndex

   End If

   ' Set the return value of the function to the
array of
   ' Steps that has been built above
   If lngStepCount = 0 Then
      SubSteps = Empty
   Else
      SubSteps = cSubSteps()
   End If

   Exit Function

SubStepsErr:
   LogErrors Errors
   mstrSource = mstrModuleName &
"SubSteps"
   On Error GoTo 0
   Err.Raise vbObjectError +
errSubStepsFailed, _
         mstrSource, _
         LoadResString(errSubStepsFailed)

End Function

Public Property Get StepCount() As Integer

   StepCount = mcarrSteps.Count

End Property

VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cAsyncShell"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'-----------------------------------------------------------
-----------------------
' Copyright © 1997 Microsoft Corporation. All
rights reserved.
'
' You have a royalty-free right to use, modify,
reproduce and distribute the
' Sample Application Files (and/or any
modified version) in any way you find
' useful, provided that you agree that Microsoft
has no warranty, obligations or
' liability for any Sample Application Files.
'-----------------------------------------------------------
-----------------

Option Explicit

' Used to indicate the source module name when
errors
' are raised by this class
Private mstrSource As String
Private Const mstrModuleName As String =
"cAsyncShell."

Public Event Terminated()

Private WithEvents moTimer As cTimerSM
Attribute moTimer.VB_VarHelpID = -1
Private proc As PROCESS_INFORMATION
Private mfShelling As Boolean


'-----------------------------------------------------------
-----------------
'Initialization and cleanup:

Private Sub Class_Initialize()
   Set moTimer = New cTimerSM
End Sub

Private Sub Class_Terminate()
   If mfShelling Then CloseHandle proc.hProcess
End Sub

'-----------------------------------------------------------
-----------------
'Shelling:

Public Sub Shell(CommandLine As String,
Optional PollingInterval As Long = 1000)
   Dim Start As STARTUPINFO

   If mfShelling Then
      On Error GoTo 0
      Err.Raise vbObjectError + errInstanceInUse, _
            mstrSource, _
            LoadResString(errInstanceInUse)
   End If
   mfShelling = True

   ' Initialize the STARTUPINFO structure:
   Start.cb = Len(Start)
   Start.dwFlags =
STARTF_USESHOWWINDOW
   Start.wShowWindow =
SW_SHOWMINNOACTIVE

   ' Start the shelled application:
   CreateProcessA 0&, CommandLine, 0&, 0&,
1&, _
      NORMAL_PRIORITY_CLASS, 0&, 0&,
Start, proc

   With moTimer
      If PollingInterval > 0 Then
         .Interval = PollingInterval
      Else
         .Interval = 1000
      End If
      .Enabled = True
   End With
End Sub
'-----------------------------------------------------------
-----------------
'Aborting:
Public Sub Abort()
   Dim nCode As Long
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
'    Dim X As Integer
'    Dim ReturnVal As Integer

    On Error GoTo AbortErr

    If Not mfShelling Then
        Call WriteError(errProgramError,
mstrSource)
    Else
'       If IsWindow(proc.hProcess) = False
Then Exit Sub
'
'       If (GetWindowLong(proc.hProcess,
GWL_STYLE) And WS_DISABLED)
Then Exit Sub
'
'       If IsWindow(proc.hProcess) Then
'           If Not
(GetWindowLong(proc.hProcess,
GWL_STYLE) And WS_DISABLED)
Then
'               X = PostMessage(proc.hProcess,
WM_CANCELMODE, 0, 0&)
'               X = PostMessage(proc.hProcess,
WM_CLOSE, 0, 0&)
'           End If
'       End If

        If TerminateProcess(proc.hProcess,
0&) = 0 Then
            Debug.Print "Unable to terminate
process: " & proc.hProcess
            Call
WriteError(errTerminateProcessFailed,
mstrSource, _
                ApiError(GetLastError()))
        Else
            ' Should always come here!
            GetExitCodeProcess proc.hProcess,
nCode
            If nCode = STILL_ACTIVE Then
                ' Write an error and close the
handles to the
                ' process anyway
                Call
WriteError(errTerminateProcessFailed,
mstrSource)
            End If
        End If

        ' Close all open handles to the shelled
process, even
        ' if any of the above calls error out
        CloseHandle proc.hProcess
        moTimer.Enabled = False
        mfShelling = False
        RaiseEvent Terminated

    End If

    Exit Sub

AbortErr:
    Call LogErrors(Errors)
    mstrSource = mstrModuleName &
"Abort"
    On Error GoTo 0
    Err.Raise vbObjectError +
errProgramError, _
        mstrSource, _
        LoadResString(errProgramError)

End Sub
Private Sub moTimer_Timer()
    Dim nCode As Long

    GetExitCodeProcess proc.hProcess, nCode
    If nCode <> STILL_ACTIVE Then
        CloseHandle proc.hProcess
        moTimer.Enabled = False
        mfShelling = False
        RaiseEvent Terminated
    End If
End Sub
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cConnDtl"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:       cConnDtl.cls
'              Microsoft TPC-H Kit Ver. 1.00
'              Copyright Microsoft, 1999
'              All Rights Reserved
'
'
'  PURPOSE:     Encapsulates the properties
and methods of a connection.
'              Contains functions to insert, update
and delete
'              connection_dtls records from the
database.
'  Contact:     Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit
Option Base 0

' Local variable(s) to hold property value(s)
Public WorkspaceId As Long
Public ConnNameId As Long
Public ConnName As String
Public ConnectionString As String
Public ConnType As ConnectionType
Public Position As Long
Public NodeDB As Database

Private mintOperation As Operation

' Used to indicate the source module name
when errors
' are raised by this class
Private mstrSource As String
Private Const mstrModuleName As String =
"cConnDtl."

' The cSequence class is used to generate
unique Connection identifiers
Private mConnectionSeq As cSequence

' The StringSM class is used to carry out string
operations
Private mFieldValue As cStringSM

Private Sub AssignParameters(qyExec As
DAO.QueryDef)
    ' Assigns values to the parameters in the
querydef object
    ' The parameter names are cryptic to
differentiate them from the field names.
    ' When the parameter names are the same as
the field names, parameters in the where
    ' clause do not get created.

    Dim prmParam As DAO.Parameter

    On Error GoTo AssignParametersErr

    For Each prmParam In qyExec.Parameters
        Select Case prmParam.Name
            Case "[w_id]"
                prmParam.Value = WorkspaceId

            Case "[c_id]"
                prmParam.Value = ConnNameId

            Case "[c_name]"
                prmParam.Value = ConnName

            Case "[c_str]"
                prmParam.Value = ConnectionString

            Case "[c_type]"
                prmParam.Value = ConnType

            Case Else
                ' Write the parameter name that is faulty
                WriteError errInvalidParameter,
mstrSource, prmParam.Name
                On Error GoTo 0
                Err.Raise errInvalidParameter,
mstrModuleName & "AssignParameters", _
                    LoadResString(errInvalidParameter)
        End Select
    Next prmParam

    Exit Sub

AssignParametersErr:

    Call LogErrors(Errors)
    On Error GoTo 0
    Err.Raise vbObjectError +
errAssignParametersFailed, _
        mstrModuleName & "AssignParameters",
LoadResString(errAssignParametersFailed)

End Sub

Public Function Clone() As cConnDtl

    ' Creates a copy of a given Connection

    Dim cCloneConn As cConnDtl

    On Error GoTo CloneErr

    Set cCloneConn = New cConnDtl

    ' Copy all the Connection properties to the newly
created Connection
    cCloneConn.WorkspaceId = WorkspaceId
    cCloneConn.ConnNameId = ConnNameId
    cCloneConn.ConnName = ConnName
    cCloneConn.ConnectionString =
ConnectionString
    cCloneConn.ConnType = ConnType
    cCloneConn.IndOperation = mintOperation
    cCloneConn.Position = Position

    ' And set the return value to the newly created
Connection
    Set Clone = cCloneConn
    Set cCloneConn = Nothing

    Exit Function

CloneErr:
    LogErrors Errors
    mstrSource = mstrModuleName & "Clone"
    On Error GoTo 0
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   Err.Raise vbObjectError +
errCloneFailed, mstrSource,
LoadResString(errCloneFailed)

End Function
Private Sub CheckDupConnectionName()
   ' Check if the Connection name already
exists in the workspace

   Dim rstConnection As Recordset
   Dim strSql As String
   Dim qy As DAO.QueryDef

   On Error GoTo
CheckDupConnectionNameErr
   mstrSource = mstrModuleName &
"CheckDupConnectionName"

   ' Create a recordset object to retrieve the
count of all Connections
   ' for the workspace with the same name
   strSql = "Select count(*) as
Connection_count " & _
       " from " &
TBL_CONNECTION_DTLS & _
       " where " & FLD_ID_WORKSPACE
& " = [w_id]" & _
       " and " &
FLD_CONN_DTL_CONNECTION_NAM
E & " = [c_name]" & _
       " and " & FLD_ID_CONN_NAME & "
<> [c_id]"

   Set qy =
dbsAttTool.CreateQueryDef(gstrEmptyStrin
g, strSql)
   Call AssignParameters(qy)

   Set rstConnection =
qy.OpenRecordset(dbOpenForwardOnly)

   If rstConnection![Connection_count] > 0
Then
      rstConnection.Close
      qy.Close
      ShowError errDupConnDtlName
      On Error GoTo 0
      Err.Raise vbObjectError +
errDupConnDtlName, _
         mstrSource,
LoadResString(errDupConnDtlName)
   End If

   rstConnection.Close
   qy.Close

   Exit Sub

CheckDupConnectionNameErr:
   LogErrors Errors
   mstrSource = mstrModuleName &
"CheckDupConnectionName"
   On Error GoTo 0
   Err.Raise vbObjectError +
errProgramError, _
      mstrSource,
LoadResString(errProgramError)

End Sub
Public Property Let IndOperation(ByVal
vdata As Operation)

   ' The valid operations are define in the
cOperations
   ' class. Check if the operation is valid
   Select Case vdata
      Case QueryOp, InsertOp, UpdateOp,
DeleteOp
         mintOperation = vdata

      Case Else
         BugAssert True
   End Select

End Property
Public Sub Validate()
   ' Each distinct object will have a Validate
method which
   ' will check if the class properties are valid.
This method
   ' will be used to check interdependant
properties that
   ' cannot be validated by the let procedures.
   ' It should be called by the add and modify
methods of the class

   If ConnName = gstrEmptyString Then

      ShowError
errConnectionNameMandatory
      On Error GoTo 0
      ' Propogate this error back to the caller
      Err.Raise vbObjectError +
errConnectionNameMandatory, _
         mstrSource,
LoadResString(errConnectionNameMandatory
)
   End If

   ' Raise an error if the Connection name
already exists in the workspace
   Call CheckDupConnectionName

End Sub
Public Sub Add()

   Dim strInsert As String
   Dim qy As DAO.QueryDef

   On Error GoTo AddErr

   ' Validate the record before trying to insert
the record
   Call Validate

   ' Create a temporary querydef object
   strInsert = "insert into " &
TBL_CONNECTION_DTLS & _
         "( " & FLD_ID_WORKSPACE & _
         ", " & FLD_ID_CONN_NAME & _
         ", " &
FLD_CONN_DTL_CONNECTION_NAME
& _
         ", " &
FLD_CONN_DTL_CONNECTION_STRING
& _
         ", " &
FLD_CONN_DTL_CONNECTION_TYPE &
" ) " & _
         " values ( [w_id], [c_id], " & _
         " [c_name], [c_str], [c_type] ) "

   Set qy =
dbsAttTool.CreateQueryDef(gstrEmptyString,
strInsert)

   ' Call a procedure to assign the Connection
values
   Call AssignParameters(qy)
   qy.Execute dbFailOnError
   qy.Close

   Exit Sub

AddErr:

   Call LogErrors(Errors)
   On Error GoTo 0
   Err.Raise vbObjectError + errInsertFailed, _
      mstrModuleName & "Add",
LoadResString(errInsertFailed)

End Sub
Public Sub Delete()

   Dim strDelete As String
   Dim qy As DAO.QueryDef

   On Error GoTo DeleteErr

   strDelete = "delete from " &
TBL_CONNECTION_DTLS & _
         " where " & FLD_ID_CONN_NAME & " =
[c_id]"
   Set qy =
dbsAttTool.CreateQueryDef(gstrEmptyString,
strDelete)

   Call AssignParameters(qy)
   qy.Execute dbFailOnError

   qy.Close

   Exit Sub

DeleteErr:
   LogErrors Errors
   On Error GoTo 0
   Err.Raise vbObjectError + errDeleteFailed, _
         mstrModuleName & "Delete",
LoadResString(errDeleteFailed)

End Sub

Public Sub Modify()

   Dim strUpdate As String
   Dim qy As QueryDef

   On Error GoTo ModifyErr

   ' Validate the updated values before trying to
modify the db
   Call Validate

   ' Create a temporary querydef object with the
modify string
   strUpdate = "update " &
TBL_CONNECTION_DTLS & _
         " set " & FLD_ID_WORKSPACE & " =
[w_id], " & _

FLD_CONN_DTL_CONNECTION_NAME & " =
[c_name], " & _

FLD_CONN_DTL_CONNECTION_STRING & "
= [c_str], " & _
         FLD_CONN_DTL_CONNECTION_TYPE
& " = [c_type] " & _
         " where " & FLD_ID_CONN_NAME & " =
[c_id]"
   Set qy =
dbsAttTool.CreateQueryDef(gstrEmptyString,
strUpdate)
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   ' Call a procedure to assign the
Connection values to the
   ' querydef object
   Call AssignParameters(qy)
   qy.Execute dbFailOnError

   qy.Close

   Exit Sub

ModifyErr:

   Call LogErrors(Errors)
   On Error GoTo 0
   Err.Raise vbObjectError +
errModifyFailed, _
       mstrModuleName & "Modify",
LoadResString(errModifyFailed)

End Sub
Public Property Get NextIdentifier() As
Long

   Dim lngNextId As Long

   On Error GoTo NextIdentifierErr

   ' Retrieve the next identifier using the
sequence class
   Set mConnectionSeq = New cSequence
   Set mConnectionSeq.IdDatabase =
dbsAttTool
   mConnectionSeq.IdentifierColumn =
FLD_ID_CONN_NAME
   lngNextId = mConnectionSeq.Identifier
   Set mConnectionSeq = Nothing

   NextIdentifier = lngNextId
   Exit Property

NextIdentifierErr:
   LogErrors Errors
   On Error GoTo 0
   Err.Raise vbObjectError + errIdGetFailed,
_
       mstrModuleName & "NextIdentifier",
LoadResString(errIdGetFailed)

End Property
Public Property Get IndOperation() As
Operation

   IndOperation = mintOperation

End Property

Private Sub Class_Initialize()

   Set mFieldValue = New cStringSM

   ' Initialize the operation indicator variable
to Query
   ' It will be modified later by the collection
class when
   ' inserts, updates or deletes are performed
   mintOperation = QueryOp

   ConnType = giDefaultConnType

End Sub

Private Sub Class_Terminate()

   Set mFieldValue = Nothing

End Sub
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cConnDtls"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:      cConnDtls.cls
'             Microsoft TPC-H Kit Ver. 1.00
'             Copyright Microsoft, 1999
'             All Rights Reserved
'
'
'  PURPOSE:    Implements an array of
cConnDtl objects.
'             Type-safe wrapper around
cNodeCollections.
'             Also contains additional functions to
determine the connection
'             string value, validation functions, etc.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

Private mcarrConnDtls As cNodeCollections

' Used to indicate the source module name
when errors
' are raised by this class
Private mstrSource As String
Private Const mstrModuleName As String =
"cConnDtls."

Public Property Set ConnDb(vdata As
Database)

   Set mcarrConnDtls.NodeDB = vdata

End Property
Public Sub Modify(cModifiedConn As
cConnDtl)

   ' First check if the parameter record is valid
   Call CheckDupConnName(cModifiedConn)

   Call
mcarrConnDtls.Modify(cModifiedConn)

End Sub
Public Sub Load(ByRef cConnToAdd As
cConnDtl)

   Call mcarrConnDtls.Load(cConnToAdd)

End Sub
Public Sub Add(ByRef cConnToAdd As
cConnDtl)

   ' First check if the record is valid
   Call Validate(cConnToAdd)

   ' Retrieve a unique identifier
   cConnToAdd.ConnNameId =
cConnToAdd.NextIdentifier

   Call mcarrConnDtls.Add(cConnToAdd)

End Sub

Public Sub Unload(lConnNameId As Long)

   Dim lngDeleteElement As Long

   lngDeleteElement = QueryIndex(lConnNameId)

   Call mcarrConnDtls.Unload(lngDeleteElement)

End Sub

Public Sub SaveConnDtlsInWsp(ByVal
lngWorkspace As Long)
   ' Call a procedure to save all connection details
records for the workspace
   Call mcarrConnDtls.Save(lngWorkspace)

End Sub
Public Function GetConnectionDtl(ByVal
lngWorkspace As Long, _
       ByVal strConnectionName As String) As
cConnDtl
   ' Returns the connection dtl for the passed in
connection name

   Dim lngIndex As Long

   ' Find all parameters in the array with a matching
workspace id
   For lngIndex = 0 To mcarrConnDtls.Count - 1
       If mcarrConnDtls(lngIndex).WorkspaceId =
lngWorkspace And _
               mcarrConnDtls(lngIndex).ConnName =
strConnectionName Then

           Set GetConnectionDtl =
mcarrConnDtls(lngIndex)
           Exit For
       End If
   Next lngIndex

   If lngIndex > mcarrConnDtls.Count - 1 Then
       ' The parameter has not been defined for the
workspace
       ' Raise an error
       On Error GoTo 0
       Err.Raise vbObjectError +
errConnNameInvalid, mstrModuleName &
"GetConnection", _
               LoadResString(errConnNameInvalid)
   End If

End Function
Public Sub Delete(lConnNameId As Long)
   ' Delete the passed in parameter

   Dim lngDeleteElement As Long

   lngDeleteElement = QueryIndex(lConnNameId)
   Call mcarrConnDtls.Delete(lngDeleteElement)

End Sub
Private Function QueryIndex(lConnNameId As
Long) As Long

   Dim lngIndex As Long

   ' Find the matching parameter record in the array
   For lngIndex = 0 To mcarrConnDtls.Count - 1
       If mcarrConnDtls(lngIndex).ConnNameId =
lConnNameId And _
               mcarrConnDtls(lngIndex).IndOperation
<> DeleteOp Then
           QueryIndex = lngIndex
           Exit Function
       End If
   Next lngIndex
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   ' Raise error that parameter has not been
found
   On Error GoTo 0
   Err.Raise vbObjectError +
errQueryIndexFailed,
"cArrParameters.QueryIndex", _
       LoadResString(errQueryIndexFailed)

End Function

Public Function
QueryConnDtl(lConnNameId As Long) As
cConnDtl

   Dim lngQueryElement As Long

   lngQueryElement =
QueryIndex(lConnNameId)

   ' Return the queried connection object
   Set QueryConnDtl =
mcarrConnDtls(lngQueryElement)

End Function
Public Property Get Count() As Long

   Count = mcarrConnDtls.Count

End Property
Public Property Get Item(lngIndex As Long)
As cConnDtl
Attribute Item.VB_UserMemId = 0

   Set Item = mcarrConnDtls(lngIndex)

End Property

Private Sub Validate(ByVal
cConnToValidate As cConnDtl)
   ' This procedure is necessary since the
class cannot validate
   ' all the connection_dtl properties on it's
own. This is 'coz we
   ' might have created new connections in
the workspace, but not
   ' saved them to the database yet - hence
the duplicate check
   ' has to be repeated in the array

   Dim lngIndex As Long
   Dim cTempParam As cConnDtl

   ' Check if the parameter name already
exists in the workspace
   For lngIndex = 0 To
mcarrConnDtls.Count - 1
       Set cTempParam =
mcarrConnDtls(lngIndex)
       If cTempParam.WorkspaceId =
cConnToValidate.WorkspaceId And _
           cTempParam.ConnName =
cConnToValidate.ConnName And _
           cTempParam.IndOperation <>
DeleteOp Then
           On Error GoTo 0
           Err.Raise vbObjectError +
errDupConnDtlName, _
           mstrSource,
LoadResString(errDupConnDtlName)
       End If
   Next lngIndex

End Sub

Private Sub CheckDupConnName(ByVal
cConnToValidate As cConnDtl)

   Dim lngIndex As Long
   Dim cTempParam As cConnDtl

   ' Check if the parameter name already exists
in the workspace
   For lngIndex = 0 To mcarrConnDtls.Count -
1
       Set cTempParam =
mcarrConnDtls(lngIndex)
       If cTempParam.WorkspaceId =
cConnToValidate.WorkspaceId And _
           cTempParam.ConnName =
cConnToValidate.ConnName And _
           cTempParam.ConnNameId <>
cConnToValidate.ConnNameId And _
           cTempParam.IndOperation <>
DeleteOp Then
           ShowError errDupConnDtlName
           On Error GoTo 0
           Err.Raise vbObjectError +
errDupConnDtlName, _
           mstrSource,
LoadResString(errDupConnDtlName)
       End If
   Next lngIndex

End Sub

Private Sub Class_Initialize()

   Set mcarrConnDtls = New cNodeCollections

End Sub

Private Sub Class_Terminate()

   Set mcarrConnDtls = Nothing

End Sub
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cConnection"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:       cConnection.cls
'              Microsoft TPC-H Kit Ver. 1.00
'              Copyright Microsoft, 1999
'              All Rights Reserved
'
'
'  PURPOSE:    Encapsulates the properties
and methods of a connection string.
'              Contains functions to insert, update
and delete
'              workspace_connections records from
the database.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit
Option Base 0

' Local variable(s) to hold property value(s)
Public WorkspaceId As Long
Public ConnectionId As Long
Public ConnectionValue As String
Public Description As String
Public NodeDB As Database
Public Position As Long
Public NoCountDisplay As Boolean
Public NoExecute As Boolean
Public ParseQueryOnly As Boolean
Public QuotedIdentifiers As Boolean
Public AnsiNulls As Boolean
Public ShowQueryPlan As Boolean
Public ShowStatsTime As Boolean
Public ShowStatsIO As Boolean
Public ParseOdbcMsg As Boolean
Public RowCount As Long
Public TsqlBatchSeparator As String
Public QueryTimeOut As Long
Public ServerLanguage As String
Public CharacterTranslation As Boolean
Public RegionalSettings As Boolean

Private mstrConnectionName As String
Private mintOperation As Operation

' Used to indicate the source module name when
errors
' are raised by this class
Private mstrSource As String
Private Const mstrModuleName As String =
"cConnection."

' The cSequence class is used to generate unique
Connection identifiers
Private mConnectionSeq As cSequence

' The StringSM class is used to carry out string
operations
Private mFieldValue As cStringSM

Private Sub AssignParameters(qyExec As
DAO.QueryDef)
   ' Assigns values to the parameters in the querydef
object
   ' The parameter names are cryptic to differentiate
them from the field names.
   ' When the parameter names are the same as the
field names, parameters in the where
   ' clause do not get created.

   Dim prmParam As DAO.Parameter

   On Error GoTo AssignParametersErr

   For Each prmParam In qyExec.Parameters
      Select Case prmParam.Name
         Case "[w_id]"
            prmParam.Value = WorkspaceId

         Case "[c_id]"
            prmParam.Value = ConnectionId

         Case "[c_name]"
            prmParam.Value = mstrConnectionName

         Case "[c_value]"
            prmParam.Value = ConnectionValue

         Case "[desc]"
            prmParam.Value = Description

         Case "[no_count]"
            prmParam.Value = NoCountDisplay

         Case "[no_exec]"
            prmParam.Value = NoExecute

         Case "[parse_only]"
            prmParam.Value = ParseQueryOnly
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
        Case "[quoted_id]"
            prmParam.Value =
QuotedIdentifiers

        Case "[a_nulls]"
            prmParam.Value = AnsiNulls

        Case "[show_qp]"
            prmParam.Value =
ShowQueryPlan

        Case "[stats_tm]"
            prmParam.Value =
ShowStatsTime

        Case "[stats_io]"
            prmParam.Value = ShowStatsIO

        Case "[parse_odbc]"
            prmParam.Value = ParseOdbcMsg

        Case "[row_cnt]"
            prmParam.Value = RowCount

        Case "[batch_sep]"
            prmParam.Value =
TsqlBatchSeparator

        Case "[qry_tmout]"
            prmParam.Value = QueryTimeOut

        Case "[lang]"
            prmParam.Value =
ServerLanguage

        Case "[char_trans]"
            prmParam.Value =
CharacterTranslation

        Case "[reg_settings]"
            prmParam.Value =
RegionalSettings

        Case Else
            ' Write the parameter name that is
faulty
            WriteError errInvalidParameter,
mstrSource, prmParam.Name
            On Error GoTo 0
            Err.Raise errInvalidParameter,
mstrModuleName & "AssignParameters", _

LoadResString(errInvalidParameter)
        End Select
    Next prmParam

    Exit Sub

AssignParametersErr:

    Call LogErrors(Errors)
    On Error GoTo 0
    Err.Raise vbObjectError +
errAssignParametersFailed, _
        mstrModuleName &
"AssignParameters",
LoadResString(errAssignParametersFailed)

End Sub

Public Function Clone() As cConnection

    ' Creates a copy of a given Connection

    Dim cCloneConn As cConnection

    On Error GoTo CloneErr

    Set cCloneConn = New cConnection

    ' Copy all the Connection properties to the
newly
    ' created Connection
    Set cCloneConn.NodeDB = NodeDB
    cCloneConn.WorkspaceId = WorkspaceId
    cCloneConn.ConnectionId = ConnectionId
    cCloneConn.ConnectionName =
mstrConnectionName
    cCloneConn.ConnectionValue =
ConnectionValue
    cCloneConn.Description = Description
    cCloneConn.IndOperation = mintOperation
    cCloneConn.Position = Position
    cCloneConn.NoCountDisplay =
NoCountDisplay
    cCloneConn.NoExecute = NoExecute
    cCloneConn.ParseQueryOnly =
ParseQueryOnly
    cCloneConn.QuotedIdentifiers =
QuotedIdentifiers
    cCloneConn.AnsiNulls = AnsiNulls
    cCloneConn.ShowQueryPlan =
ShowQueryPlan
    cCloneConn.ShowStatsTime =
ShowStatsTime
    cCloneConn.ShowStatsIO = ShowStatsIO
    cCloneConn.ParseOdbcMsg =
ParseOdbcMsg
    cCloneConn.RowCount = RowCount
    cCloneConn.TsqlBatchSeparator =
TsqlBatchSeparator
    cCloneConn.QueryTimeOut =
QueryTimeOut
    cCloneConn.ServerLanguage =
ServerLanguage
    cCloneConn.CharacterTranslation =
CharacterTranslation
    cCloneConn.RegionalSettings =
RegionalSettings

    ' And set the return value to the newly
created Connection
    Set Clone = cCloneConn
    Set cCloneConn = Nothing

    Exit Function

CloneErr:
    LogErrors Errors
    mstrSource = mstrModuleName & "Clone"
    On Error GoTo 0
    Err.Raise vbObjectError + errCloneFailed,
mstrSource, LoadResString(errCloneFailed)

End Function
Private Sub CheckDupConnectionName()
    ' Check if the Connection name already
exists in the workspace

    Dim rstConnection As Recordset
    Dim strSql As String
    Dim qy As DAO.QueryDef

    On Error GoTo
CheckDupConnectionNameErr
    mstrSource = mstrModuleName &
"CheckDupConnectionName"

    ' Create a recordset object to retrieve the
count of all Connections
    ' for the workspace with the same name
    strSql = "Select count(*) as Connection_count "
& _
        " from workspace_connections " & _
        " where workspace_id = [w_id]" & _
        " and connection_name = [c_name]" & _
        " and connection_id <> [c_id]"

    Set qy =
NodeDB.CreateQueryDef(gstrEmptyString, strSql)
    Call AssignParameters(qy)

    Set rstConnection =
qy.OpenRecordset(dbOpenForwardOnly)

    If rstConnection![Connection_count] > 0 Then
        rstConnection.Close
        qy.Close
        ShowError errDuplicateConnectionName
        On Error GoTo 0
        Err.Raise vbObjectError +
errDuplicateConnectionName, _
            mstrSource,
LoadResString(errDuplicateConnectionName)
    End If

    rstConnection.Close
    qy.Close

    Exit Sub

CheckDupConnectionNameErr:
    LogErrors Errors
    mstrSource = mstrModuleName &
"CheckDupConnectionName"
    On Error GoTo 0
    Err.Raise vbObjectError + errProgramError, _
        mstrSource, LoadResString(errProgramError)

End Sub
Private Sub CheckDB()
    ' Check if the database object has been initialized

    If NodeDB Is Nothing Then
        On Error GoTo 0
        Err.Raise vbObjectError + errInvalidDB, _
            mstrModuleName & "CheckDB",
LoadResString(errInvalidDB)
    End If

End Sub
Public Property Let ConnectionName(vdata As
String)

    If vdata = gstrEmptyString Then

        ShowError errConnectionNameMandatory
        On Error GoTo 0
        ' Propogate this error back to the caller
        Err.Raise vbObjectError +
errConnectionNameMandatory, _
            mstrSource,
LoadResString(errConnectionNameMandatory)
    Else
        mstrConnectionName = vdata
    End If

End Property

Public Property Let IndOperation(ByVal vdata As
Operation)

    ' The valid operations are define in the
cOperations
    ' class. Check if the operation is valid
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   Select Case vdata
      Case QueryOp, InsertOp, UpdateOp,
DeleteOp
         mintOperation = vdata

      Case Else
         BugAssert True
   End Select

End Property
Public Sub Validate()
   ' Each distinct object will have a Validate
method which
   ' will check if the class properties are
valid. This method
   ' will be used to check interdependant
properties that
   ' cannot be validated by the let
procedures.
   ' It should be called by the add and
modify methods of the class

   ' Check if the db object is valid
   Call CheckDB

   ' Raise an error if the Connection name
already exists in the workspace
   Call CheckDupConnectionName

End Sub
Public Sub Add()

   Dim strInsert As String
   Dim qy As DAO.QueryDef

   On Error GoTo AddErr

   ' Validate the record before trying to
insert the record
   Call Validate

   ' Create a temporary querydef object
   strInsert = "insert into
workspace_connections " & _
            "( workspace_id, connection_id, " &
_
            "connection_name,
connection_value, " & _
            "description, no_count_display, " &
_
            "no_execute, parse_query_only, " &
_
            "ANSI_quoted_identifiers,
ANSI_nulls, " & _
            "show_query_plan,
show_stats_time, " & _
            "show_stats_io,
parse_odbc_msg_prefixes, " & _
            "row_count, tsql_batch_separator, "
& _
            "query_time_out, server_language, "
& _
            "character_translation,
regional_settings ) " & _
           " values ( [w_id], [c_id], [c_name],
[c_value], " & _
            " [desc], [no_count], [no_exec],
[parse_only], " & _
            " [quoted_id], [a_nulls], [show_qp],
[stats_tm], " & _
            " [stats_io], [parse_odbc], [row_cnt],
[batch_sep], " & _
            " [qry_tmout], [lang], [char_trans],
[reg_settings] ) "
   Set qy =
NodeDB.CreateQueryDef(gstrEmptyString,
strInsert)

   ' Call a procedure to assign the Connection
values
   Call AssignParameters(qy)

   qy.Execute dbFailOnError
   qy.Close

   Exit Sub

AddErr:

   Call LogErrors(Errors)
   On Error GoTo 0
   Err.Raise vbObjectError + errInsertFailed, _
      mstrModuleName & "Add",
LoadResString(errInsertFailed)

End Sub
Public Sub Delete()

   Dim strDelete As String
   Dim qy As DAO.QueryDef

   On Error GoTo DeleteErr

   ' Check if the db object is valid
   Call CheckDB

   strDelete = "delete from
workspace_connections " & _
            " where connection_id = [c_id]"
   Set qy =
NodeDB.CreateQueryDef(gstrEmptyString,
strDelete)

   Call AssignParameters(qy)
   qy.Execute dbFailOnError

   qy.Close

   Exit Sub

DeleteErr:
   LogErrors Errors
   On Error GoTo 0
   Err.Raise vbObjectError + errDeleteFailed, _
         mstrModuleName & "Delete",
LoadResString(errDeleteFailed)

End Sub

Public Sub Modify()

   Dim strUpdate As String
   Dim qy As QueryDef

   On Error GoTo ModifyErr

   ' Validate the updated values before trying to
modify the db
   Call Validate

   ' Create a temporary querydef object with
the modify string
   strUpdate = "update workspace_connections
" & _
            " set workspace_id = [w_id], " & _
            "connection_name = [c_name], " & _
            "connection_value = [c_value], " & _
            "description = [desc], " & _
            "no_count_display = [no_count], " & _
            "no_execute = [no_exec], " & _
            "parse_query_only = [parse_only], " & _
            "ANSI_quoted_identifiers = [quoted_id], "
& _
            "ANSI_nulls = [a_nulls], " & _
            "show_query_plan = [show_qp], " & _
            "show_stats_time = [stats_tm], " & _
            "show_stats_io = [stats_io], " & _
            "parse_odbc_msg_prefixes = [parse_odbc],
" & _
            "row_count = [row_cnt], " & _
            "tsql_batch_separator = [batch_sep], " & _
            "query_time_out = [qry_tmout], " & _
            "server_language = [lang], " & _
            "character_translation = [char_trans], " & _
            "regional_settings = [reg_settings] " & _
            " where connection_id = [c_id]"
   Set qy =
NodeDB.CreateQueryDef(gstrEmptyString,
strUpdate)

   ' Call a procedure to assign the Connection values
to the
   ' querydef object
   Call AssignParameters(qy)
   qy.Execute dbFailOnError

   qy.Close

   Exit Sub

ModifyErr:

   Call LogErrors(Errors)
   On Error GoTo 0
   Err.Raise vbObjectError + errModifyFailed, _
      mstrModuleName & "Modify",
LoadResString(errModifyFailed)

End Sub
Public Property Get ConnectionName() As String

   ConnectionName = mstrConnectionName

End Property

Public Property Get NextIdentifier() As Long

   Dim lngNextId As Long

   On Error GoTo NextIdentifierErr

   ' First check if the database object is valid
   Call CheckDB

   ' Retrieve the next identifier using the sequence
class
   Set mConnectionSeq = New cSequence
   Set mConnectionSeq.IdDatabase = NodeDB
   mConnectionSeq.IdentifierColumn =
"connection_id"
   lngNextId = mConnectionSeq.Identifier
   Set mConnectionSeq = Nothing

   NextIdentifier = lngNextId
   Exit Property

NextIdentifierErr:
   LogErrors Errors
   On Error GoTo 0
   Err.Raise vbObjectError + errIdGetFailed, _
      mstrModuleName & "NextIdentifier",
LoadResString(errIdGetFailed)

End Property
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
Public Property Get IndOperation() As
Operation

   IndOperation = mintOperation

End Property

Private Sub Class_Initialize()

   Set mFieldValue = New cStringSM

   ' Initialize the operation indicator variable
to Query
   ' It will be modified later by the collection
class when
   ' inserts, updates or deletes are performed
   mintOperation = QueryOp

   ' Initialize connection properties to their
default values
   NoCountDisplay =
DEF_NO_COUNT_DISPLAY
   NoExecute = DEF_NO_EXECUTE
   ParseQueryOnly =
DEF_PARSE_QUERY_ONLY
   QuotedIdentifiers =
DEF_ANSI_QUOTED_IDENTIFIERS
    AnsiNulls = DEF_ANSI_NULLS
   ShowQueryPlan =
DEF_SHOW_QUERY_PLAN
   ShowStatsTime =
DEF_SHOW_STATS_TIME
   ShowStatsIO = DEF_SHOW_STATS_IO
   ParseOdbcMsg =
DEF_PARSE_ODBC_MSG_PREFIXES
   RowCount = DEF_ROW_COUNT
    TsqlBatchSeparator =
DEF_TSQL_BATCH_SEPARATOR
   QueryTimeOut =
DEF_QUERY_TIME_OUT
   ServerLanguage =
DEF_SERVER_LANGUAGE
   CharacterTranslation =
DEF_CHARACTER_TRANSLATION
   RegionalSettings =
DEF_REGIONAL_SETTINGS

End Sub

Private Sub Class_Terminate()

   Set NodeDB = Nothing
   Set mFieldValue = Nothing

End Sub
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cConnections"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:      cConnections.cls
'            Microsoft TPC-H Kit Ver. 1.00
'            Copyright Microsoft, 1999
'            All Rights Reserved
'
'
'  PURPOSE:    Implements an array of
cConnection objects.
'            Type-safe wrapper around
cNodeCollections.
'            Also contains validation functions,
etc.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

Private mcarrConnections As
cNodeCollections

' Used to indicate the source module name
when errors
' are raised by this class
Private mstrSource As String
Private Const mstrModuleName As String =
"cConnections."

Public Property Set ConnDb(vdata As
Database)

   Set mcarrConnections.NodeDB = vdata

End Property
Public Sub Modify(cModifiedConn As
cConnection)

   ' First check if the parameter record is valid
   Call CheckDupConnName(cModifiedConn)

   Call
mcarrConnections.Modify(cModifiedConn)

End Sub
Public Sub Load(ByRef cConnToAdd As
cConnection)

   Call mcarrConnections.Load(cConnToAdd)

End Sub
Public Sub Add(ByRef cConnToAdd As
cConnection)

   Set cConnToAdd.NodeDB =
mcarrConnections.NodeDB

   ' First check if the record is valid
   Call Validate(cConnToAdd)

   ' Retrieve a unique identifier
   cConnToAdd.ConnectionId =
cConnToAdd.NextIdentifier

   Call mcarrConnections.Add(cConnToAdd)

End Sub

Public Sub Unload(lngConnId As Long)

   Dim lngDeleteElement As Long

   lngDeleteElement = QueryIndex(lngConnId)

   Call
mcarrConnections.Unload(lngDeleteElement)

End Sub

Public Sub SaveConnectionsInWsp(ByVal
lngWorkspace As Long)
   ' Call a procedure to save all connection
records for the workspace
   Call mcarrConnections.Save(lngWorkspace)

End Sub
Public Function GetConnection(ByVal
lngWorkspace As Long, _
      ByVal strConnectionName As String) As
cConnection
   ' Returns the connection string for the passed in
connection name

   Dim lngIndex As Long

   ' Find all parameters in the array with a matching
workspace id
   For lngIndex = 0 To mcarrConnections.Count - 1
      If mcarrConnections(lngIndex).WorkspaceId =
lngWorkspace And _

mcarrConnections(lngIndex).ConnectionName =
strConnectionName Then

         Set GetConnection =
mcarrConnections(lngIndex)
         Exit For
      End If
   Next lngIndex

   If lngIndex > mcarrConnections.Count - 1 Then
      '  The parameter has not been defined for the
workspace
      ' Raise an error
      On Error GoTo 0
      Err.Raise vbObjectError +
errConnNameInvalid, mstrModuleName &
"GetConnection", _
            LoadResString(errConnNameInvalid)
   End If

End Function
Public Sub Delete(lngConnId As Long)
   ' Delete the passed in parameter

   Dim lngDeleteElement As Long

   lngDeleteElement = QueryIndex(lngConnId)
   Call
mcarrConnections.Delete(lngDeleteElement)

End Sub
Private Function QueryIndex(lngConnId As Long)
As Long

   Dim lngIndex As Long

   ' Find the matching parameter record in the array
   For lngIndex = 0 To mcarrConnections.Count - 1
      If mcarrConnections(lngIndex).ConnectionId =
lngConnId And _

mcarrConnections(lngIndex).IndOperation <>
DeleteOp Then
         QueryIndex = lngIndex
         Exit Function
      End If
   Next lngIndex

   ' Raise error that parameter has not been found
   On Error GoTo 0
   Err.Raise vbObjectError + errQueryIndexFailed,
"cArrParameters.QueryIndex", _
      LoadResString(errQueryIndexFailed)

End Function

Public Function QueryConnection(lngConnId As
Long) As cConnection

   Dim lngQueryElement As Long
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   lngQueryElement =
QueryIndex(lngConnId)

   ' Return the queried connection object
   Set QueryConnection =
mcarrConnections(lngQueryElement)

End Function
Public Property Get Count() As Long

   Count = mcarrConnections.Count

End Property
Public Property Get Item(lngIndex As Long)
As cConnection
Attribute Item.VB_UserMemId = 0

   Set Item = mcarrConnections(lngIndex)

End Property

Public Sub Validate(ByVal
cConnToValidate As cConnection)
   ' This procedure is necessary since the
class cannot validate
   ' all the parameter properties on it's own.
This is 'coz we
   ' might have created new parameters in
the workspace, but not
   ' saved them to the database yet - hence
the duplicate check
   ' has to be repeated in the array

   Dim lngIndex As Long
   Dim cTempParam As cConnection

   ' Check if the parameter name already
exists in the workspace
   For lngIndex = 0 To
mcarrConnections.Count - 1
      Set cTempParam =
mcarrConnections(lngIndex)
      If cTempParam.WorkspaceId =
cConnToValidate.WorkspaceId And _
            cTempParam.ConnectionName =
cConnToValidate.ConnectionName And _
            cTempParam.IndOperation <>
DeleteOp Then
         On Error GoTo 0
         Err.Raise vbObjectError +
errDuplicateConnectionName, _
            mstrSource,
LoadResString(errDuplicateConnectionNam
e)
      End If
   Next lngIndex

End Sub
Public Sub CheckDupConnName(ByVal
cConnToValidate As cConnection)

   Dim lngIndex As Long
   Dim cTempParam As cConnection

   ' Check if the parameter name already
exists in the workspace
   For lngIndex = 0 To
mcarrConnections.Count - 1
      Set cTempParam =
mcarrConnections(lngIndex)
      If cTempParam.WorkspaceId =
cConnToValidate.WorkspaceId And _
            cTempParam.ConnectionName =
cConnToValidate.ConnectionName And _
            cTempParam.ConnectionId <>
cConnToValidate.ConnectionId And _
            cTempParam.IndOperation <>
DeleteOp Then
         ShowError
errDuplicateConnectionName
         On Error GoTo 0
         Err.Raise vbObjectError +
errDuplicateConnectionName, _
            mstrSource,
LoadResString(errDuplicateConnectionName)
      End If
   Next lngIndex

End Sub

Private Sub Class_Initialize()

   Set mcarrConnections = New
cNodeCollections

End Sub

Private Sub Class_Terminate()

   Set mcarrConnections = Nothing

End Sub
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cConstraint"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:       cConstraint.cls
'              Microsoft TPC-H Kit Ver. 1.00
'              Copyright Microsoft, 1999
'              All Rights Reserved
'
'
'  PURPOSE:    Encapsulates the properties
and methods of a constraint.
'              Contains functions to insert, update
and delete
'              step_constraints records from the
database.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

' Module level variables to store the property
values
Private mlngConstraintId As Long
Private mlngStepId As Long
Private mstrVersionNo As String
Private mintConstraintType As Integer
Private mlngGlobalStepId As Long
Private mstrGlobalVersionNo As String
Private mintSequenceNo As Integer
Private mdbsConstraintDB As Database
Private mlngWorkspaceId As Integer
Private mintOperation As Operation
Private mlngPosition As Long

' The cSequence class is used to generate
unique step identifiers
Private mConstraintSeq As cSequence

Private Const mstrModuleName As String =
".cConstraint."
Private mstrSource As String

Public Enum ConstraintType
   gintPreStep = 1
   gintPostStep = 2
End Enum

Private Const mstrSQ As String = """
Public Property Get WorkspaceId() As Long
   WorkspaceId = mlngWorkspaceId
End Property
Public Property Let WorkspaceId(ByVal vdata As
Long)
   mlngWorkspaceId = vdata
End Property

Public Property Get IndOperation() As Operation

   IndOperation = mintOperation

End Property
Public Property Let IndOperation(ByVal vdata As
Operation)

   On Error GoTo IndOperationErr
   mstrSource = mstrModuleName &
"IndOperation"

   ' The valid operations are define in the
cOperations
   ' class. Check if the operation is valid
   Select Case vdata
      Case QueryOp, InsertOp, UpdateOp, DeleteOp
         mintOperation = vdata

      Case Else
         On Error GoTo 0
         Err.Raise vbObjectError +
errInvalidOperation, _
            mstrSource,
LoadResString(errInvalidOperation)
   End Select

   Exit Property

IndOperationErr:
   LogErrors Errors
   mstrSource = mstrModuleName &
"IndOperation"
   On Error GoTo 0
   Err.Raise vbObjectError +
errLetOperationFailed, _
      mstrSource,
LoadResString(errLetOperationFailed)

End Property

Public Function Clone() As cConstraint

   ' Creates a copy of a given constraint

   Dim cConsClone As cConstraint

   On Error GoTo CloneErr
   mstrSource = mstrModuleName & "Clone"

   Set cConsClone = New cConstraint

   ' Copy all the workspace properties to the newly
   ' created workspace
   cConsClone.ConstraintId = mlngConstraintId
   cConsClone.StepId = mlngStepId
   cConsClone.VersionNo = mstrVersionNo
   cConsClone.ConstraintType =
mintConstraintType
   cConsClone.GlobalStepId = mlngGlobalStepId
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   cConsClone.GlobalVersionNo =
mstrGlobalVersionNo
   cConsClone.SequenceNo =
mintSequenceNo
   cConsClone.WorkspaceId =
mlngWorkspaceId
   cConsClone.IndOperation =
mintOperation

   ' And set the return value to the newly
created constraint
   Set Clone = cConsClone

   Exit Function

CloneErr:
   LogErrors Errors
   mstrSource = mstrModuleName &
"Clone"
   On Error GoTo 0
   Err.Raise vbObjectError +
errCloneFailed, _
         mstrSource,
LoadResString(errCloneFailed)

End Function

Public Property Get SequenceNo() As
Integer

   SequenceNo = mintSequenceNo

End Property

Public Property Let SequenceNo(ByVal
vdata As Integer)
   mintSequenceNo = vdata
End Property

Public Sub Add()
   ' Inserts a new step constraint into the
database

   Dim strInsert As String
   Dim qy As DAO.QueryDef

   On Error GoTo AddErr

   ' First check if the database object is valid
   Call CheckDB

   ' Any record validations
   Call Validate

   ' Create a temporary querydef object
   strInsert = "insert into step_constraints "
& _
         "( constraint_id, step_id, version_no,
" & _
         " constraint_type, global_step_id,
global_version_no, sequence_no ) " & _
         " values ( [cons_id], [s_id], [ver_no],
" & _
         " [cons_type], [g_step_id],
[g_ver_no], " & _
         " [seq_no] )"
   Set qy =
mdbsConstraintDB.CreateQueryDef(gstrEm
ptyString, strInsert)

   ' Call a procedure to execute the Querydef
object
   Call AssignParameters(qy)

   qy.Execute dbFailOnError
   qy.Close

'   strInsert = "insert into step_constraints " &
_
'      "( constraint_id, step_id, version_no, " &
_
'      " constraint_type, global_step_id,
global_version_no, sequence_no ) " & _
'      " values ( " & _
'      Str(mlngConstraintId) & ", " &
Str(mlngStepId) & ", " & _
'      mstrSQ & mstrVersionNo & mstrSQ & ",
" & Str(mintConstraintType) & ", " & _
'      Str(mlngGlobalStepId) & ", " & mstrSQ
& mstrGlobalVersionNo & mstrSQ & ", " & _
'      Str(mintSequenceNo) & " ) "
'
'   BugMessage strInsert
'   mdbsConstraintDB.Execute strInsert,
dbFailOnError
   Exit Sub

AddErr:
   LogErrors Errors
   mstrSource = mstrModuleName & "Add"
   On Error GoTo 0
   Err.Raise vbObjectError +
errAddConstraintFailed, _
         mstrSource, _

LoadResString(errAddConstraintFailed)
End Sub
Private Sub AssignParameters(qyExec As
DAO.QueryDef)
   ' Assigns values to the parameters in the
querydef object
   ' The parameter names are cryptic to make
them different
   ' from the field names. When the parameter
names are
   ' the same as the field names, parameters in
the where
   ' clause do not get created.

   Dim prmParam As DAO.Parameter

   On Error GoTo AssignParametersErr
   mstrSource = mstrModuleName &
"AssignParameters"

   For Each prmParam In qyExec.Parameters
      Select Case prmParam.Name
         Case "[cons_id]"
            prmParam.Value = mlngConstraintId

         Case "[s_id]"
            prmParam.Value = mlngStepId

         Case "[ver_no]"
            prmParam.Value = mstrVersionNo

         Case "[cons_type]"
            prmParam.Value =
mintConstraintType

         Case "[g_step_id]"
            prmParam.Value =
mlngGlobalStepId

         Case "[g_ver_no]"
            prmParam.Value =
mstrGlobalVersionNo

         Case "[seq_no]"
            prmParam.Value = mintSequenceNo

         Case Else
            ' Write the parameter name that is faulty
            WriteError errInvalidParameter,
mstrSource, _
                  prmParam.Name
            On Error GoTo 0
            Err.Raise errInvalidParameter,
mstrSource, _
                  LoadResString(errInvalidParameter)
      End Select
   Next prmParam

   Exit Sub

AssignParametersErr:

   mstrSource = mstrModuleName &
"AssignParameters"
   Call LogErrors(Errors)
   On Error GoTo 0
   Err.Raise vbObjectError +
errAssignParametersFailed, _
      mstrSource,
LoadResString(errAssignParametersFailed)

End Sub
Public Property Get NextIdentifier() As Long

   Dim lngNextId As Long

   On Error GoTo NextIdentifierErr

   ' First check if the database object is valid
   Call CheckDB

   ' Retrieve the next constraint identifier using the
   ' sequence class
   Set mConstraintSeq = New cSequence
   Set mConstraintSeq.IdDatabase =
mdbsConstraintDB
   mConstraintSeq.IdentifierColumn =
"constraint_id"
   lngNextId = mConstraintSeq.Identifier
   Set mConstraintSeq = Nothing

   NextIdentifier = lngNextId
   Exit Property

NextIdentifierErr:
   LogErrors Errors
   mstrSource = mstrModuleName &
"NextIdentifier"
   On Error GoTo 0
   Err.Raise vbObjectError + errStepIdGetFailed, _
      mstrSource,
LoadResString(errStepIdGetFailed)

End Property

Private Sub CheckDB()
   ' Check if the database object has been initialized

   If mdbsConstraintDB Is Nothing Then
      ShowError errInvalidDB
      On Error GoTo 0
      Err.Raise vbObjectError + errInvalidDB, _
         mstrModuleName,
LoadResString(errInvalidDB)
   End If

End Sub

Public Sub Delete()
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   ' Deletes the step constraint record from
the database

   Dim strDelete As String
   Dim qy As DAO.QueryDef

   On Error GoTo DeleteErr
   mstrSource = mstrModuleName &
"Delete"

   ' There can be multiple constraints for a
step,
   ' meaning that there can be multiple
constraint records
   ' with the same constraint_id. Only a
combination
   ' of the step_id, version and constraint_id
will be
   ' unique
   strDelete = "delete from step_constraints
" & _
           " where constraint_id = [cons_id]" &
_
           " and step_id = [s_id] " & _
           " and version_no = [ver_no] "
   Set qy =
mdbsConstraintDB.CreateQueryDef(gstrEm
ptyString, strDelete)

   Call AssignParameters(qy)
   qy.Execute dbFailOnError

   qy.Close

'   strDelete = "Delete from step_constraints
" & _
'           " where constraint_id = " &
Str(mlngConstraintId) & _
'           " and step_id = " & Str(mlngStepId)
& _
'           " and version_no = " & mstrSQ &
mstrVersionNo & mstrSQ
'
'   'BugMessage strDelete
'   mdbsConstraintDB.Execute strDelete,
dbFailOnError

   Exit Sub

DeleteErr:
   LogErrors Errors
   mstrSource = mstrModuleName &
"Delete"
   On Error GoTo 0
   Err.Raise vbObjectError +
errDeleteConstraintFailed, _
           mstrSource, _

LoadResString(errDeleteConstraintFailed)
End Sub
Public Sub Modify()
   ' Updates the sequence no of the step
constraint record
   ' in the database

   Dim strUpdate As String
   Dim qy As QueryDef

   On Error GoTo Modify

   ' First check if the database object is valid
   Call CheckDB

   ' Any record validations
   Call Validate

   ' There can be multiple constraints for a step,
   ' meaning that there can be multiple
constraint records
   ' with the same constraint_id. Only a
combination
   ' of the step_id, version and constraint_id
will be
   ' unique
   ' Create a temporary querydef object with
the modify string
   strUpdate = "Update step_constraints " & _
           " set sequence_no = [seq_no] " & _
           " where constraint_id = [cons_id] " & _
           " and step_id = [s_id] " & _
           " and version_no = [ver_no] "
   Set qy =
mdbsConstraintDB.CreateQueryDef(gstrEmpt
yString, strUpdate)

   ' Call a procedure to assign the parameter
values to the
   ' querydef object
   Call AssignParameters(qy)
   qy.Execute dbFailOnError

   qy.Close

'   strUpdate = "Update step_constraints " & _
'           " set sequence_no = " &
Str(mintSequenceNo) & _
'           " where constraint_id = " &
Str(mlngConstraintId) & _
'           " and step_id = " & Str(mlngStepId) &
_
'           " and version_no = " & mstrSQ &
mstrVersionNo & mstrSQ
'
'   'BugMessage strUpdate
'   mdbsConstraintDB.Execute strUpdate,
dbFailOnError
   Exit Sub

Modify:
   LogErrors Errors
   mstrSource = mstrModuleName & "Modify"
   On Error GoTo 0
   Err.Raise vbObjectError +
errUpdateConstraintFailed, _
           mstrSource, _

LoadResString(errUpdateConstraintFailed)
End Sub
Public Property Get Position() As Long

   Position = mlngPosition

End Property
Public Property Let Position(ByVal RHS As
Long)

   mlngPosition = RHS

End Property

Public Sub Validate()
   ' Each distinct object will have a Validate
method which
   ' will check if the class properties are valid.
This method
   ' will be used to check interdependant
properties that
   ' cannot be validated by the let procedures.
   ' It should be called by the add and modify
methods of the class

   ' No validations are necessary for the constraint
object

End Sub

Public Property Set NodeDB(vdata As Database)

   Set mdbsConstraintDB = vdata

End Property

Public Property Get NodeDB() As Database

   Set NodeDB = mdbsConstraintDB

End Property

Public Property Get GlobalVersionNo() As String

   GlobalVersionNo = mstrGlobalVersionNo

End Property

Public Property Let GlobalVersionNo(ByVal vdata
As String)

   mstrGlobalVersionNo = vdata

End Property

Public Property Get GlobalStepId() As Long

   GlobalStepId = mlngGlobalStepId

End Property

Public Property Get ConstraintId() As Long

   ConstraintId = mlngConstraintId

End Property

Public Property Get VersionNo() As String

   VersionNo = mstrVersionNo

End Property

Public Property Get StepId() As Long

   StepId = mlngStepId

End Property

Public Property Let VersionNo(ByVal vdata As
String)

   mstrVersionNo = vdata

End Property

Public Property Let StepId(ByVal vdata As Long)

   mlngStepId = vdata

End Property

Public Property Let ConstraintId(ByVal vdata As
Long)
   On Error GoTo ConstraintIdErr
   mstrSource = mstrModuleName &
"ConstraintId"

   If (vdata > 0) Then
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
      mlngConstraintId = vdata
   Else
      ' Propogate this error back to the caller
      On Error GoTo 0
      Err.Raise vbObjectError +
errConstraintIdInvalid, _
         mstrSource,
LoadResString(errConstraintIdInvalid)
   End If

   Exit Property

ConstraintIdErr:
   LogErrors Errors
   mstrSource = mstrModuleName &
"ConstraintId"
   On Error GoTo 0
   Err.Raise vbObjectError +
errConstraintIdSetFailed, _
      mstrSource,
LoadResString(errConstraintIdSetFailed)

End Property

Public Property Let GlobalStepId(ByVal
vdata As Long)

   On Error GoTo GlobalStepIdErr
   mstrSource = mstrModuleName &
"GlobalStepId"

   If (vdata > 0) Then
      mlngGlobalStepId = vdata
   Else
      ' Propogate this error back to the caller
      On Error GoTo 0
      Err.Raise vbObjectError +
errGlobalStepIdInvalid, _
         mstrSource,
LoadResString(errGlobalStepIdInvalid)
   End If

   Exit Property

GlobalStepIdErr:
   LogErrors Errors
   mstrSource = mstrModuleName &
"GlobalStepId"
   On Error GoTo 0
   Err.Raise vbObjectError +
errGlobalStepIdSetFailed, _
      mstrSource,
LoadResString(errGlobalStepIdSetFailed)

End Property

Public Property Let ConstraintType(ByVal
vdata As ConstraintType)

   On Error GoTo ConstraintTypeErr

   ' A global step can be either a pre- or a
post-execution step.
   ' These constants have been defined in the
enumeration,
   ' ConstraintType, which is exposed
   Select Case vdata
      Case gintPreStep, gintPostStep
         mintConstraintType = vdata

      Case Else
         On Error GoTo 0
         Err.Raise vbObjectError +
errConstraintTypeInvalid, _
         mstrSource,
LoadResString(errConstraintTypeInvalid)
   End Select

   Exit Property

ConstraintTypeErr:
   LogErrors Errors
   mstrSource = mstrModuleName &
"ConstraintType"
   On Error GoTo 0
   Err.Raise vbObjectError +
errConstraintTypeLetFailed, _
      mstrSource,
LoadResString(errConstraintTypeLetFailed)

End Property

Public Property Get ConstraintType() As
ConstraintType

   ConstraintType = mintConstraintType

End Property

Private Sub Class_Initialize()

   ' Initialize the operation indicator variable to
Query
   ' It will be modified later by the collection
class when
   ' inserts, updates or deletes are performed
   mintOperation = QueryOp

End Sub


VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cFailedStep"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:       cFailedStep.cls
'             Microsoft TPC-H Kit Ver. 1.00
'             Copyright Microsoft, 1999
'             All Rights Reserved
'
'
'  PURPOSE:    Properties of a step execution
failure.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

Public InstanceId As Long
Public StepId As Long
Public ParentStepId As Long
Public ContCriteria As ContinuationCriteria
Public EndTime As Currency
Public AskResponse As Long
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cFailedSteps"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:       cFailedSteps.cls
'             Microsoft TPC-H Kit Ver. 1.00
'             Copyright Microsoft, 1999
'             All Rights Reserved
'
'
'  PURPOSE:    This module encapsulates a
collection of failed steps. It
'             also determines whether sub-steps of a
passed in step need
'             to be skipped due to a failure.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

Private mcFailedSteps As cVector
Public Function ExecuteSubStep(lParentStepId As
Long) As Boolean
   ' Returns False if there is any condition that
prevents sub-steps of the passed
   ' in instance from being executed
   Dim lIndex As Long

   ExecuteSubStep = True

   For lIndex = 0 To Count() - 1
      If mcFailedSteps(lIndex).ContCriteria =
gintOnFailureCompleteSiblings And _
            lParentStepId <>
mcFailedSteps(lIndex).ParentStepId Then
         ExecuteSubStep = False
         Exit For
      End If

      If mcFailedSteps(lIndex).ContCriteria =
gintOnFailureAbortSiblings And _
            lParentStepId =
mcFailedSteps(lIndex).ParentStepId Then
         ExecuteSubStep = False
         Exit For
      End If

      If mcFailedSteps(lIndex).ContCriteria =
gintOnFailureSkipSiblings And _
            lParentStepId =
mcFailedSteps(lIndex).ParentStepId Then
         ExecuteSubStep = False
         Exit For
      End If

      If mcFailedSteps(lIndex).ContCriteria =
gintOnFailureAbort Then
         ExecuteSubStep = False
         Exit For
      End If

   Next lIndex

End Function
Public Sub Add(ByVal objItem As cFailedStep)

   mcFailedSteps.Add objItem

End Sub
Public Function Delete(ByVal lPosition As Long)
As cFailedStep

   Set Delete = mcFailedSteps.Delete(lPosition)

End Function

Public Sub Clear()

   mcFailedSteps.Clear
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
End Sub
Public Function Count() As Long

   Count = mcFailedSteps.Count

End Function
Public Property Get Item(ByVal Position As
Long) As cFailedStep
Attribute Item.VB_UserMemId = 0

   Set Item = mcFailedSteps.Item(Position)

End Property

Public Function StepFailed(lStepId As
Long) As Boolean

   ' Returns True if a failure record already
exists for the passed in step
   Dim lIndex As Long

   StepFailed = False

   For lIndex = 0 To Count() - 1
      If mcFailedSteps(lIndex).StepId =
lStepId Then
         StepFailed = True
         Exit For
      End If
   Next lIndex

End Function

Private Sub Class_Initialize()

   Set mcFailedSteps = New cVector

End Sub

Private Sub Class_Terminate()

   Set mcFailedSteps = Nothing

End Sub
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cFileInfo"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:       cFileInfo.cls
'             Microsoft TPC-H Kit Ver. 1.00
'             Copyright Microsoft, 1999
'             All Rights Reserved
'
'
'  PURPOSE:    File Properties viz. name,
handle, etc.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

Private mstrFileName As String
Private mintFileHandle As Integer
Private mdbsNodeDb As Database ' Since it
is used to form a cNodeCollection
Private mlngPosition As Long ' Since it is
used to form a cNodeCollection
Public Property Get FileName() As String

   FileName = mstrFileName

End Property
Public Property Let FileName(ByVal vdata As
String)

   mstrFileName = vdata

End Property
Public Property Let FileHandle(ByVal vdata
As Integer)

   mintFileHandle = vdata

End Property
Public Property Set NodeDB(vdata As
Database)

   Set mdbsNodeDb = vdata

End Property

Public Property Get NodeDB() As Database

   Set NodeDB = mdbsNodeDb

End Property
Public Property Get Position() As Long

   Position = mlngPosition

End Property
Public Property Let Position(ByVal vdata As
Long)

   mlngPosition = vdata

End Property

Public Property Get FileHandle() As Integer

   FileHandle = mintFileHandle

End Property

VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cFileSM"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
Attribute VB_Ext_KEY =
"SavedWithClassBuilder" ,"Yes"
Attribute VB_Ext_KEY = "Top_Level" ,"Yes"
'  FILE:       cFileSM.cls
'             Microsoft TPC-H Kit Ver. 1.00
'             Copyright Microsoft, 1999
'             All Rights Reserved
'
'
'  PURPOSE:    Encapsulates functions to open
a file and write to it.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

' Used to indicate the source module name
when errors
' are raised by this class
Private Const mstrModuleName As String =
"cFileSM."
Private mstrSource As String
Private mstrFileName As String
Private mintHFile As Integer
Private mstrFileHeader As String
Private mstrProjectName As String

Public Sub CloseFile()

   ' Close the file
   If mintHFile > 0 Then
      Call CloseFileSM(mstrFileName)
      mintHFile = 0
   End If

End Sub

Public Property Let ProjectName(ByVal vdata As
String)
   ' An optional field - will be appended to the file
   ' header string if specified

   Const strProjectHdr As String = "Project Name:"

   mstrProjectName = vdata
   mstrFileHeader = mstrFileHeader & _
         Space$(1) & strProjectHdr & Space$(1) & _
         gstrSQ & vdata & gstrSQ

End Property
Public Property Get ProjectName() As String

   ProjectName = mstrProjectName

End Property
Public Property Get FileName() As String

   FileName = mstrFileName

End Property
Public Property Let FileName(ByVal vdata As
String)

   mstrFileName = vdata

End Property
Public Sub WriteLine(strMsg As String)

   ' Writes the passed in string to the file
   Call WriteToFile(strMsg, False)

End Sub

Public Sub WriteField(strMsg As String)

   ' Writes the passed in string to the file
   Call WriteToFile(strMsg, True)

End Sub

Private Sub WriteToFile(strMsg As String, _
      blnContinue As Boolean)
   ' Writes the passed in string to the file - the
   ' Continue flag indicates whether the next line
will
   ' be continued on the same line or printed on a
new one

   On Error GoTo WriteToFileErr

   ' Open the file if it hasn't been already
   If mintHFile = 0 Then

      ' If the filename has not been initialized, do not
      ' attempt to open it
      If mstrFileName <> gstrEmptyString Then
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
        mintHFile =
OpenFileSM(mstrFileName)

        If mintHFile = 0 Then
          ' The Open File command failed
for some reason
          ' No point in trying to write the
file header
        Else
          ' Print a file header, if a header
string has been
          ' initialized
          If mstrFileHeader <>
gstrEmptyString Then
            Print #mintHFile,
            Print #mintHFile,
mstrFileHeader
            Print #mintHFile,
          End If
        End If
      End If
    End If

    If mintHFile <> 0 Then
      If strMsg = gstrEmptyString Then
        Print #mintHFile,
      Else
        If blnContinue Then
          ' Write the message to the file -
continue
          ' all subsequent characters on the
same line
          Print #mintHFile, strMsg;
        Else
          ' Write the message to the file
          Print #mintHFile, strMsg
        End If
      End If
    Else
      ' Display the string to the user instead
of
      ' trying to write it to the file
      ' This could be the project error log that
we were
      ' trying to open! Play it safe and display
errors - do
      ' not try to log them.
      MsgBox strMsg, vbOKOnly
    End If

    Exit Sub

WriteToFileErr:
    ' Log the error code raised by Visual
Basic
    Call DisplayErrors(Errors)

    ' Display the string to the user instead of
    ' trying to write it to the file
    MsgBox strMsg, vbOKOnly

End Sub
Public Property Let FileHeader(ByVal vdata
As String)

    mstrFileHeader = vdata

End Property
Public Property Get FileHeader() As String

    FileHeader = mstrFileHeader

End Property
Private Sub Class_Terminate()

    ' Close the file opened by this instance
    Call CloseFile

End Sub


VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cGlobalStep"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:       cGlobalStep.cls
'             Microsoft TPC-H Kit Ver. 1.00
'             Copyright Microsoft, 1999
'             All Rights Reserved
'
'
'  PURPOSE:     Encapsulates the properties
and methods of a global step.
'             Implements the cStep class - carries
out initializations
'             and validations that are specific to
global steps.
'  Contact:     Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

Implements cStep

' Object variable to keep the reference in
Private mcStep As cStep

' Used to indicate the source module name
when errors
' are raised by this class
Private mstrSource As String
Private Const mstrModuleName As String =
"cGlobalStep."

Private Sub cStep_AddAllIterators()

    Call mcStep.AddAllIterators

End Sub

Private Sub cStep_AddIterator(cItRecord As
cIterator)

    Call mcStep.AddIterator(cItRecord)

End Sub

Private Property Let
cStep_ArchivedFlag(ByVal RHS As Boolean)

    mcStep.ArchivedFlag = RHS

End Property

Private Property Get cStep_ArchivedFlag() As
Boolean

    cStep_ArchivedFlag = mcStep.ArchivedFlag

End Property

Private Sub Class_Initialize()

    ' Create the object
    Set mcStep = New cStep

    ' Initialize the object with valid values for a
global step
    ' The global flag should be the first field to be
initialized
    ' since subsequent validations might try to check
if the
    ' step being created is global
    mcStep.GlobalFlag = True
    mcStep.StepType = gintGlobalStep

    ' A global step cannot have any sub-steps
associated with it
    ' Hence, it will always be at Step Level 0
    mcStep.ParentStepId = 0
    mcStep.ParentVersionNo = gstrMinVersion
    mcStep.StepLevel = 0

    ' The enabled flag must be False for all global
steps
    ' Global steps can be of two types
    ' a. Those that are run globally within a
workspace either
    '    before every step, after every step or during
the entire
    '    run, depending on the global run method
    ' b. Those that are not run globally, but qualify to
be either
    '    pre or post-execution steps for other steps in
the workspace.
    '    Whether or not such a step will be executed
depends on
    '    whether the step for which it is defined as a
pre/post
    '    step will be executed
    mcStep.EnabledFlag = False

    mcStep.ContinuationCriteria = gintNoOption
    mcStep.DegreeParallelism =
gstrGlobalParallelism

End Sub
Private Sub Class_Terminate()

    ' Remove the step object
    Set mcStep = Nothing

End Sub

Private Sub cStep_Add()

    ' Call a private procedure to see if the step text
has been
    ' entered - since a global step actually executes a
step, entry
    ' of the text is mandatory
    Call StepTextOrFileEntered

    ' Call the Add method of the step class to carry
out the insert
    mcStep.Add

End Sub

Private Function cStep_Clone(Optional cCloneStep
As cStep) As cStep

    Dim cNewGlobal As cGlobalStep

    Set cNewGlobal = New cGlobalStep
    Set cStep_Clone = mcStep.Clone(cNewGlobal)

End Function
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
Private Property Get
cStep_ContinuationCriteria() As
ContinuationCriteria

   cStep_ContinuationCriteria =
mcStep.ContinuationCriteria

End Property

Private Property Let
cStep_ContinuationCriteria(ByVal RHS As
ContinuationCriteria)

   ' The continuation criteria field will
always be empty for a
   ' global step
   mcStep.ContinuationCriteria = 0

End Property

Private Property Let
cStep_DegreeParallelism(ByVal RHS As
String)

   ' Will always be zero for a global step
   mcStep.DegreeParallelism =
gstrGlobalParallelism

End Property

Private Property Get
cStep_DegreeParallelism() As String

   cStep_DegreeParallelism =
mcStep.DegreeParallelism

End Property

Private Sub cStep_DeleteIterator(cItRecord
As cIterator)

   Call mcStep.DeleteIterator(cItRecord)

End Sub

Private Sub cStep_Delete()

   mcStep.Delete

End Sub

Private Property Get cStep_EnabledFlag()
As Boolean

   cStep_EnabledFlag =
mcStep.EnabledFlag

End Property

Private Property Let
cStep_EnabledFlag(ByVal RHS As
Boolean)

   ' The enabled flag must be False for all
global steps
   ' Global steps can be of two types
   ' a. Those that are run globally within a
workspace either
   '    before every step, after every step or
during the entire
   '    run, depending on the global run
method
   ' b. Those that are not run globally, but
qualify to be either
   '    pre or post-execution steps for other steps
in the workspace.
   '    Whether or not such a step will be
executed depends on
   '    whether the step for which it is defined as
a pre/post
   '    step will be executed
   mcStep.EnabledFlag = False

End Property

Private Property Let cStep_ErrorFile(ByVal
RHS As String)

   mcStep.ErrorFile = RHS

End Property

Private Property Get cStep_ErrorFile() As
String

   cStep_ErrorFile = mcStep.ErrorFile

End Property

Private Property Let
cStep_ExecutionMechanism(ByVal RHS As
ExecutionMethod)

   ' Whether or not the Execution Mechanism
is valid will be
   ' checked by the Step class
   mcStep.ExecutionMechanism = RHS

End Property

Private Property Get
cStep_ExecutionMechanism() As
ExecutionMethod

   cStep_ExecutionMechanism =
mcStep.ExecutionMechanism

End Property

Private Property Let
cStep_FailureDetails(ByVal RHS As String)

   ' Whether or not the Failure Details are valid
for the
   ' selected failure criteria will be checked by
the Step class
   mcStep.FailureDetails = RHS

End Property

Private Property Get cStep_FailureDetails() As
String

   cStep_FailureDetails =
mcStep.FailureDetails

End Property

Private Property Get cStep_GlobalFlag() As
Boolean

   cStep_GlobalFlag = mcStep.GlobalFlag

End Property

Private Property Let cStep_GlobalFlag(ByVal
RHS As Boolean)

   ' Set the global flag to true
   mcStep.GlobalFlag = True

End Property

Private Function cStep_IncVersionX() As String

   cStep_IncVersionX = mcStep.IncVersionX

End Function

Private Function cStep_IncVersionY() As String

   cStep_IncVersionY = mcStep.IncVersionY

End Function


'Private Property Let
cStep_GlobalRunMethod(ByVal RHS As Integer)
'
'    ' Whether or not the Global Run Method is valid
for the step
'    ' will be checked by the Step class
'    mcStep.GlobalRunMethod = RHS
'
'End Property
'
'Private Property Get cStep_GlobalRunMethod() As
Integer
'
'    cStep_GlobalRunMethod =
mcStep.GlobalRunMethod
'
'End Property
'
Private Property Get cStep_IndOperation() As
Operation

   cStep_IndOperation = mcStep.IndOperation

End Property

Private Property Let cStep_IndOperation(ByVal
RHS As Operation)

   mcStep.IndOperation = RHS

End Property

Private Sub cStep_InsertIterator(cItRecord As
cIterator)

   Call mcStep.InsertIterator(cItRecord)

End Sub

Private Function cStep_IsNewVersion() As
Boolean
   cStep_IsNewVersion = mcStep.IsNewVersion
End Function

Private Function cStep_IteratorCount() As Long

   cStep_IteratorCount = mcStep.IteratorCount

End Function

Private Property Let cStep_IteratorName(ByVal
RHS As String)

   mcStep.IteratorName = RHS

End Property
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
Private Property Get cStep_IteratorName() As String

   cStep_IteratorName = mcStep.IteratorName

End Property

Private Function cStep_Iterators() As Variant

   cStep_Iterators = mcStep.Iterators

End Function

Private Sub cStep_LoadIterator(cItRecord As cIterator)

   Call mcStep.LoadIterator(cItRecord)

End Sub

'Private Property Let cStep_LogFile(ByVal RHS As String)
'
'   mcStep.LogFile = RHS
'
'End Property
'
'Private Property Get cStep_LogFile() As String
'
'   cStep_LogFile = mcStep.LogFile
'
'End Property

Private Sub cStep_ModifyIterator(cItRecord As cIterator)

   Call mcStep.ModifyIterator(cItRecord)

End Sub

Private Sub cStep_Modify()

   ' Call a private procedure to see if the step text has been
   ' entered - since a global step actually executes a step,
   ' entry of the text is mandatory
   Call StepTextOrFileEntered

   ' Call the Modify method of the step class to carry out the update
   mcStep.Modify

End Sub

Private Property Get cStep_NextStepId() As Long

   cStep_NextStepId = mcStep.NextStepId

End Property

Private Property Set cStep_NodeDB(RHS As DAO.Database)

   Set mcStep.NodeDB = RHS

End Property

Private Property Get cStep_NodeDB() As DAO.Database

   Set cStep_NodeDB = mcStep.NodeDB

End Property

Private Function cStep_OldVersionNo() As String
   cStep_OldVersionNo = mcStep.OldVersionNo
End Function

Private Property Let cStep_OutputFile(ByVal RHS As String)

   mcStep.OutputFile = RHS

End Property

Private Property Get cStep_OutputFile() As String

   cStep_OutputFile = mcStep.OutputFile

End Property

Private Property Let cStep_ParentStepId(ByVal RHS As Long)

   ' A global step cannot have any sub-steps associated with it
   ' Hence, the parent step id and parent version number will be zero
   mcStep.ParentStepId = 0

End Property

Private Property Get cStep_ParentStepId() As Long

   cStep_ParentStepId = mcStep.ParentStepId

End Property

Private Property Let cStep_ParentVersionNo(ByVal RHS As String)

   ' A global step cannot have any sub-steps associated with it
   ' Hence, the parent step id and parent version number will be zero
   mcStep.ParentVersionNo = gstrMinVersion

End Property

Private Property Get cStep_ParentVersionNo() As String

   cStep_ParentVersionNo = mcStep.ParentVersionNo

End Property

Private Property Let cStep_Position(ByVal RHS As Long)

   mcStep.Position = RHS

End Property

Private Property Get cStep_Position() As Long

   cStep_Position = mcStep.Position

End Property

Private Sub cStep_RemoveIterator(cItRecord As cIterator)

   Call mcStep.RemoveIterator(cItRecord)

End Sub

Private Sub cStep_SaveIterators()

   Call mcStep.SaveIterators

End Sub

Private Property Let cStep_SequenceNo(ByVal RHS As Integer)

   mcStep.SequenceNo = RHS

End Property

Private Property Get cStep_SequenceNo() As Integer

   cStep_SequenceNo = mcStep.SequenceNo

End Property

Private Property Let cStep_StepId(ByVal RHS As Long)

   mcStep.StepId = RHS

End Property
Private Property Get cStep_StepId() As Long

   cStep_StepId = mcStep.StepId

End Property


Private Property Let cStep_StepLabel(ByVal RHS As String)

   mcStep.StepLabel = RHS

End Property

Private Property Get cStep_StepLabel() As String

   cStep_StepLabel = mcStep.StepLabel

End Property

Private Property Let cStep_StartDir(ByVal RHS As String)

   mcStep.StartDir = RHS

End Property

Private Property Get cStep_StartDir() As String

   cStep_StartDir = mcStep.StartDir

End Property

Private Property Let cStep_StepLevel(ByVal RHS As Integer)

   ' A global step cannot have any sub-steps associated with it
   ' Hence, it will always be at step level 0
   mcStep.StepLevel = 0
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
End Property

Private Property Get cStep_StepLevel() As
Integer

   cStep_StepLevel = mcStep.StepLevel

End Property

Private Property Let cStep_StepText(ByVal
RHS As String)

   mcStep.StepText = RHS

End Property

Private Property Get cStep_StepText() As
String

   cStep_StepText = mcStep.StepText

End Property

Private Property Let
cStep_StepTextFile(ByVal RHS As String)

   mcStep.StepTextFile = RHS

End Property

Private Property Get cStep_StepTextFile()
As String

   cStep_StepTextFile =
mcStep.StepTextFile

End Property

Private Property Let cStep_StepType(RHS
As gintStepType)

   mcStep.StepType = gintGlobalStep

End Property

Private Property Get cStep_StepType() As
gintStepType

   cStep_StepType = mcStep.StepType

End Property

Private Sub cStep_UnloadIterators()

   Call mcStep.UnloadIterators

End Sub

Private Sub cStep_UpdateIterator(cItRecord
As cIterator)

   Call mcStep.UpdateIterator(cItRecord)

End Sub

Private Sub cStep_UpdateIteratorVersion()

   Call mcStep.UpdateIteratorVersion

End Sub

Private Sub cStep_Validate()
   ' The validate routines for each of the
steps will
   ' carry out the specific validations for the
type and
   ' call the generic validation routine

   On Error GoTo cStep_ValidateErr
   mstrSource = mstrModuleName &
"cStep_Validate"

   ' Validations specific to global steps

   ' Check if the step text or a file name has
been
   ' specified
   Call StepTextOrFileEntered

   ' The step level must be zero for all globals
   If mcStep.StepLevel <> 0 Then
      ShowError errStepLevelZeroForGlobal
      On Error GoTo 0
      Err.Raise vbObjectError +
errValidateFailed, _
            gstrSource, _
            LoadResString(errValidateFailed)
   End If

   If mcStep.EnabledFlag Then
      ShowError errEnabledFlagFalseForGlobal
      On Error GoTo 0
      Err.Raise vbObjectError +
errValidateFailed, _
            gstrSource, _
            LoadResString(errValidateFailed)
   End If

   If mcStep.DegreeParallelism > 0 Then
      ShowError
errDegParallelismNullForGlobal
      On Error GoTo 0
      Err.Raise vbObjectError +
errValidateFailed, _
            gstrSource, _
            LoadResString(errValidateFailed)
   End If

   If mcStep.ContinuationCriteria > 0 Then
      ShowError errContCriteriaNullForGlobal
      On Error GoTo 0
      Err.Raise vbObjectError +
errValidateFailed, _
            gstrSource, _
            LoadResString(errValidateFailed)
   End If

   mcStep.Validate

   Exit Sub

cStep_ValidateErr:
   LogErrors Errors
   mstrSource = mstrModuleName &
"cStep_Validate"
   On Error GoTo 0
   Err.Raise vbObjectError +
errValidateFailed, _
         mstrSource, _
         LoadResString(errValidateFailed)
End Sub
Private Sub StepTextOrFileEntered()
   ' Checks if either the step text or the name of
the file containing
   ' the text has been entered
   ' If both of them are null or both of them are
not null,
   ' the global step is invalid and an error is
raised

   If StringEmpty(mcStep.StepText) And
StringEmpty(mcStep.StepTextFile) Then
      ShowError errStepTextAndFileNull
      On Error GoTo 0
      Err.Raise vbObjectError +
errStepTextAndFileNull, _
         mstrSource,
LoadResString(errStepTextAndFileNull)
   ElseIf Not StringEmpty(mcStep.StepText) And
Not StringEmpty(mcStep.StepTextFile) Then
      ShowError errStepTextOrFile
      On Error GoTo 0
      Err.Raise vbObjectError + errStepTextOrFile,
_
         mstrSource,
LoadResString(errStepTextOrFile)
   End If

End Sub

Private Property Let cStep_VersionNo(ByVal RHS
As String)

   mcStep.VersionNo = RHS

End Property

Private Property Get cStep_VersionNo() As String

   cStep_VersionNo = mcStep.VersionNo

End Property

Private Property Let cStep_WorkspaceId(ByVal
RHS As Long)

   mcStep.WorkspaceId = RHS

End Property

Private Property Get cStep_WorkspaceId() As Long

   cStep_WorkspaceId = mcStep.WorkspaceId

End Property
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cInstance"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:       cInstance.cls
'             Microsoft TPC-H Kit Ver. 1.00
'             Copyright Microsoft, 1999
'             All Rights Reserved
'
'
'  PURPOSE:    Encapsulates the properties and
methods of an instance.
'             An instance is created when a step is
executed for a
'             particular iterator value (if applicable) at
'run' time.
'             Contains functions to determine if an
instance is running,
'             complete, and so on.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
' Used to indicate the source module name
when errors
' are raised by this class
Private Const mstrModuleName As String =
"cInstance."
Private mstrSource As String

Private mcStep As cStep
Public Key As String ' Node key for the step
being executed
Public InstanceId As Long
Public ParentInstanceId As Long ' The
parent instance
Private mblnNoMoreToStart As Boolean
Private mblnComplete As Boolean
Public StartTime As Currency
Public EndTime As Currency
Public ElapsedTime As Currency
Private mintStatus As InstanceStatus
Public DegreeParallelism As Integer
Private mcIterators As cRunColIt

' A collection of all the sub-steps for this
step
Private mcSubSteps As cSubSteps
Public Sub UpdateStartTime(lStepId As
Long, Optional ByVal StartTm As Currency
= gdtmEmpty, _
      Optional ByVal EndTm As Currency =
gdtmEmpty, _
      Optional ByVal Elapsed As Currency =
0)
   ' We do not maintain start and end
timestamps for the constraint
   ' of a step. Hence we check if the process
that just started/
   ' terminated is the worker step that is
being executed. If so,
   ' we update the start/end time and status
on the instance record.

   BugAssert (StartTm <> gdtmEmpty) Or
(EndTm <> gdtmEmpty), "Mandatory
parameter missing."

   ' Make sure that we are executing the
actual step and not
   ' a pre or post-execution constraint
   If mcStep.StepId = lStepId Then
      If StartTm <> 0 Then
         StartTime = StartTm
         mintStatus = gintRunning
      Else
         EndTime = EndTm
         ElapsedTime = Elapsed
         mintStatus = gintComplete
      End If
   End If

End Sub
Public Function
ValidForIteration(cParentInstance As
cInstance, _
      ByVal intConsType As
ConstraintType) As Boolean
   ' Returns true if the instance passed in is
the first or
   ' last iteration for the step, depending on
the constraint type

   Dim cSubStepRec As cSubStep
   Dim vntIterators As Variant

   On Error GoTo ValidForIterationErr

   If cParentInstance Is Nothing Then
      ' This will only be true for the dummy
instance, which
      ' cannot have any iterators defined for it
      ValidForIteration = True
      Exit Function
   End If

   vntIterators = mcStep.Iterators

   If Not StringEmpty(mcStep.IteratorName)
And Not IsEmpty(vntIterators) Then

      Set cSubStepRec =
cParentInstance.QuerySubStep(mcStep.StepId)

      If intConsType = gintPreStep Then
         ' Pre-execution constraints will only be
executed
         ' before the first iteration
         If
cSubStepRec.LastIterator.IteratorType =
gintValue Then
            ValidForIteration =
(cSubStepRec.LastIterator.Sequence = _
                  gintMinIteratorSequence)
         Else
            ValidForIteration =
(cSubStepRec.LastIterator.Value = _

cSubStepRec.LastIterator.RangeFrom)
         End If
      Else
         ' Post-execution constraints will only be
executed
         ' after the last iteration - check if there
are any
         ' pending iterations
         ValidForIteration =
cSubStepRec.NextIteration(mcStep) Is
Nothing
      End If
   Else
      ValidForIteration = True
   End If

   Exit Function

ValidForIterationErr:
   ' Log the error code raised by Visual Basic
   Call LogErrors(Errors)
   On Error GoTo 0
   mstrSource = mstrModuleName &
"ValidForIteration"
   Err.Raise vbObjectError +
errExecInstanceFailed, _
         mstrSource,
LoadResString(errExecInstanceFailed)

End Function

Public Sub CreateSubStep(cSubStepDtls As
cStep, RunParams As cArrParameters)

   Dim cNewSubStep As cSubStep

   On Error GoTo CreateSubStepErr

   Set cNewSubStep = New cSubStep

   cNewSubStep.StepId = cSubStepDtls.StepId
   cNewSubStep.TasksComplete = 0
   cNewSubStep.TasksRunning = 0

   ' Initialize the iterator for the instance

   Set cNewSubStep.LastIterator = New
cRunItDetails
   Call cNewSubStep.InitializeIt(cSubStepDtls,
RunParams)

   ' Add add the substep to the collection
   mcSubSteps.Add cNewSubStep

   Set cNewSubStep = Nothing

   Exit Sub

CreateSubStepErr:
   ' Log the error code raised by Visual Basic
   Call LogErrors(Errors)
   On Error GoTo 0
   mstrSource = mstrModuleName &
"CreateSubStep"
   Err.Raise vbObjectError + errProgramError,
mstrSource, _
      LoadResString(errProgramError)

End Sub

Public Function QuerySubStep(ByVal SubStepId
As Long) As cSubStep
   ' Retrieves the sub-step record for the passed in
sub-step id

   Dim lngIndex As Long

   On Error GoTo QuerySubStepErr

   ' Find the sub-step node with the matching step id
   For lngIndex = 0 To mcSubSteps.Count - 1
      If mcSubSteps(lngIndex).StepId = SubStepId
Then
         Set QuerySubStep = mcSubSteps(lngIndex)
         Exit For
      End If
   Next lngIndex

   Exit Function

QuerySubStepErr:
   ' Log the error code raised by Visual Basic
   Call LogErrors(Errors)
   On Error GoTo 0
   mstrSource = mstrModuleName &
"QuerySubStep"
   Err.Raise vbObjectError +
errNavInstancesFailed, _
         mstrSource,
LoadResString(errNavInstancesFailed)

End Function
Public Property Let AllStarted(ByVal vdata As
Boolean)

   'bugmessage "Set All Started to " & vData & "
for : " & _
      mstrKey

   mblnNoMoreToStart = vdata

End Property
Public Property Get AllStarted() As Boolean

   AllStarted = mblnNoMoreToStart

End Property
Public Property Let AllComplete(ByVal vdata As
Boolean)
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   'bugmessage "Set All Complete to " &
vData & " for : " & _
      mstrKey

   mblnComplete = vdata

End Property

Public Property Get AllComplete() As
Boolean

   AllComplete = mblnComplete

End Property

Public Sub ChildExecuted(mlngStepId As
Long)
   ' This procedure is called when a sub-step
executes.

   Dim lngIndex As Long

   On Error GoTo ChildExecutedErr

   BugAssert mcStep.StepType =
gintManagerStep

   For lngIndex = 0 To mcSubSteps.Count -
1
      If mcSubSteps(lngIndex).StepId =
mlngStepId Then

mcSubSteps(lngIndex).TasksRunning = _

mcSubSteps(lngIndex).TasksRunning + 1
'          BugMessage "Tasks Running for
Step Id : " & _
'
CStr(mcSubSteps(lngIndex).StepId) & _
'                  " Instance Id: " & InstanceId &
_
'                  " = " &
mcSubSteps(lngIndex).TasksRunning
         Exit For
      End If
   Next lngIndex

   If lngIndex > mcSubSteps.Count - 1 Then
      ' The child step wasn't found - raise an
error
      On Error GoTo 0
      Err.Raise vbObjectError +
errInvalidChild, mstrModuleName, _
            LoadResString(errInvalidChild)
   End If

   Exit Sub

ChildExecutedErr:
   ' Log the error code raised by Visual
Basic
   Call LogErrors(Errors)
   On Error GoTo 0
   Err.Raise vbObjectError +
errInstanceOpFailed, mstrModuleName &
"ChildExecuted", _
         LoadResString(errInstanceOpFailed)

End Sub
Public Sub ChildTerminated(mlngStepId As
Long)
   ' This procedure is called when any sub-
step process
   ' terminates. Note: The TasksComplete
field will be
   ' updated only when all the instances for a
sub-step
   ' complete execution.
   Dim lngIndex As Long

   On Error GoTo ChildTerminatedErr

   BugAssert mcStep.StepType =
gintManagerStep

   For lngIndex = 0 To mcSubSteps.Count - 1

      If mcSubSteps(lngIndex).StepId =
mlngStepId Then
         mcSubSteps(lngIndex).TasksRunning =
_

mcSubSteps(lngIndex).TasksRunning - 1
'          BugMessage "Tasks Running for Step
Id : " & _
'
CStr(mcSubSteps(lngIndex).StepId) & _
'                  " Instance Id: " & InstanceId & _
'                  " = " &
mcSubSteps(lngIndex).TasksRunning

         BugAssert
mcSubSteps(lngIndex).TasksRunning >= 0, _
               "Tasks running for " &
CStr(mlngStepId) & _
               " Instance Id " & InstanceId & " is
less than 0."
         Exit For
      End If
   Next lngIndex

   If lngIndex > mcSubSteps.Count - 1 Then
      ' The child step wasn't found - raise an
error
      On Error GoTo 0
      Err.Raise errInvalidChild,
mstrModuleName & "ChildTerminated", _
            LoadResString(errInvalidChild)
   End If

   Exit Sub

ChildTerminatedErr:
   ' Log the error code raised by Visual Basic
   Call LogErrors(Errors)
   On Error GoTo 0
   mstrSource = mstrModuleName &
"ChildTerminated"
   Err.Raise vbObjectError +
errInstanceOpFailed, mstrSource, _
         LoadResString(errInstanceOpFailed)

End Sub
Public Sub ChildCompleted(mlngStepId As
Long)
   ' This procedure is called when any a sub-
step completes
   ' execution. Note: The TasksComplete field
will be
   ' incremented.
   Dim lngIndex As Long

   On Error GoTo ChildCompletedErr

   BugAssert mcStep.StepType =
gintManagerStep

   For lngIndex = 0 To mcSubSteps.Count - 1
      BugAssert
mcSubSteps(lngIndex).TasksComplete >= 0, _
            "Tasks complete for " &
CStr(mcSubSteps(lngIndex).StepId) & _
            " Instance Id " & InstanceId & " is less
than 0."

      If mcSubSteps(lngIndex).StepId = mlngStepId
Then
         mcSubSteps(lngIndex).TasksComplete = _
               mcSubSteps(lngIndex).TasksComplete
+ 1
'          BugMessage "Tasks Complete for Step Id :
" & _
'                  CStr(mcSubSteps(lngIndex).StepId) &
_
'                  " Instance Id: " & InstanceId & _
'                  " = " &
mcSubSteps(lngIndex).TasksComplete
         Exit For
      End If
   Next lngIndex

   If lngIndex > mcSubSteps.Count - 1 Then
      ' The child step wasn't found - raise an error
      On Error GoTo 0
      Err.Raise errInvalidChild, mstrModuleName, _
            LoadResString(errInvalidChild)
   End If

   Exit Sub

ChildCompletedErr:
   ' Log the error code raised by Visual Basic
   Call LogErrors(Errors)
   On Error GoTo 0
   Err.Raise vbObjectError + errInstanceOpFailed,
mstrModuleName & "ChildCompleted", _
         LoadResString(errInstanceOpFailed)

End Sub
Public Sub ChildDeleted(mlngStepId As Long)
   ' This procedure is called when a sub-step needs
to be re-executed
   ' Note: The TasksComplete field is decremented.
We needn't worry about
   ' the TasksRunning field since no steps are
currently running.
   Dim lngIndex As Long

   On Error GoTo ChildDeletedErr

   BugAssert mcStep.StepType = gintManagerStep

   For lngIndex = 0 To mcSubSteps.Count - 1

      If mcSubSteps(lngIndex).StepId = mlngStepId
Then
         mcSubSteps(lngIndex).TasksRunning = _
               mcSubSteps(lngIndex).TasksRunning -
1

         BugAssert
mcSubSteps(lngIndex).TasksRunning >= 0, _
               "Tasks running for " &
CStr(mcSubSteps(lngIndex).StepId) & _
               " Instance Id " & InstanceId & " is less
than 0."
         Exit For
      End If
   Next lngIndex

   If lngIndex > mcSubSteps.Count - 1 Then
      ' The child step wasn't found - raise an error
      On Error GoTo 0
      Err.Raise errInvalidChild, mstrModuleName, _
            LoadResString(errInvalidChild)
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   End If

   Exit Sub

ChildDeletedErr:
   ' Log the error code raised by Visual
Basic
   Call LogErrors(Errors)
   On Error GoTo 0
   Err.Raise vbObjectError +
errInstanceOpFailed, mstrModuleName &
"ChildDeleted", _
         LoadResString(errInstanceOpFailed)

End Sub
Private Sub RaiseErrForWorker()

   If mcStep.StepType <> gintManagerStep
Then
      On Error GoTo 0
      mstrSource = mstrModuleName &
"RaiseErrForWorker"
      Err.Raise vbObjectError +
errInvalidForWorker, _
            mstrSource, _

LoadResString(errInvalidForWorker)
   End If

End Sub

Public Property Get Step() As cStep

   Set Step = mcStep

End Property
Public Property Get Iterators() As cRunColIt

   Set Iterators = mcIterators

End Property
Public Property Get SubSteps() As
cSubSteps

   Call RaiseErrForWorker

   Set SubSteps = mcSubSteps

End Property
Public Property Set Step(cRunStep As
cStep)

   Set mcStep = cRunStep

End Property

Public Property Set Iterators(cIts As
cRunColIt)

   Set mcIterators = cIts

End Property
Public Property Get IsPending() As Boolean
   ' Returns true if the step has any substeps
that need
   ' execution
   Dim lngIndex As Long
   Dim lngRunning As Long

   Call RaiseErrForWorker

   If Not mblnComplete And Not
mblnNoMoreToStart Then
      ' Get a count of all the substeps that are
already being
      ' executed
      lngRunning = 0
      For lngIndex = 0 To mcSubSteps.Count -
1
         lngRunning = lngRunning +
mcSubSteps(lngIndex).TasksRunning
      Next lngIndex

      IsPending = (lngRunning <
DegreeParallelism)
   Else
      ' This should be sufficient to prove that
there r no
      ' more sub-steps to be executed.
      ' mblnComplete: Handles the case where
all steps have
      ' been executed
      ' mblnNoMoreToStart: Handles the case
where the step
      ' has a degree of parallelism greater than
the total
      ' number of sub-steps available to execute
      IsPending = False
   End If

End Property
Public Property Get IsRunning() As Boolean
   ' Returns true if the any one of the substeps
is still
   ' executing
   Dim lngIndex As Long

   Call RaiseErrForWorker

   IsRunning = False

   ' If a substep has no currently executing
tasks and
   ' the tasks completed is greater than zero,
then we can
   ' assume that it has completed execution
(otherwise we
   ' would've run a new task the moment one
completed!)
   For lngIndex = 0 To mcSubSteps.Count - 1
      If mcSubSteps(lngIndex).TasksRunning >
0 Then
         IsRunning = True
         Exit For
      End If
   Next lngIndex

End Property
Public Property Get TotalRunning() As Long
   ' Returns the total number of substeps that
are executing
   Dim lngTotalProcesses As Long
   Dim lngIndex As Long

   Call RaiseErrForWorker

   lngTotalProcesses = 0
   For lngIndex = 0 To mcSubSteps.Count - 1
      BugAssert
mcSubSteps(lngIndex).TasksRunning >= 0, _
            "Tasks running for " &
CStr(mcSubSteps(lngIndex).StepId) & _
            " is less than 0."
      lngTotalProcesses = lngTotalProcesses +
mcSubSteps(lngIndex).TasksRunning
   Next lngIndex

   TotalRunning = lngTotalProcesses
End Property
Public Property Get RunningForStep(lngSubStepId
As Long) As Long
   ' Returns the total number of instances of the
substep
   ' that are executing
   Dim lngIndex As Long

   Call RaiseErrForWorker

   For lngIndex = 0 To mcSubSteps.Count - 1
      BugAssert
mcSubSteps(lngIndex).TasksRunning >= 0, _
            "Tasks running for " &
CStr(mcSubSteps(lngIndex).StepId) & _
            " is less than 0."

      If mcSubSteps(lngIndex).StepId =
lngSubStepId Then
         RunningForStep =
mcSubSteps(lngIndex).TasksRunning
         Exit For
      End If
   Next lngIndex

   If lngIndex > mcSubSteps.Count - 1 Then
      ' The child step wasn't found - raise an error
      On Error GoTo 0
      Err.Raise errInvalidChild, mstrSource, _
            LoadResString(errInvalidChild)
   End If

End Property

Public Property Let Status(ByVal vdata As
InstanceStatus)

   mintStatus = vdata

End Property

Public Property Get Status() As InstanceStatus

   Status = mintStatus

End Property
Private Sub Class_Initialize()

   Set mcSubSteps = New cSubSteps

   mblnNoMoreToStart = False
   mblnComplete = False
   StartTime = gdtmEmpty
   EndTime = gdtmEmpty

End Sub

Private Sub Class_Terminate()

   mcSubSteps.Clear
   Set mcSubSteps = Nothing

End Sub
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cInstances"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:      cInstances.cls
'            Microsoft TPC-H Kit Ver. 1.00
'            Copyright Microsoft, 1999
'            All Rights Reserved
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
'
'
'  PURPOSE:    Implements a collection of
cInstance objects.
'            Type-safe wrapper around cVector.
'            Also contains additional functions
to query an instance, etc.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

' Used to indicate the source module name
when errors
' are raised by this class
Private Const mstrModuleName As String =
"cInstance."
Private mstrSource As String

Private mcInstances As cVector

Public Function QueryInstance(ByVal
InstanceId As Long) As cInstance
   ' Retrieves the record for the passed in
instance from
   ' the collection

   Dim lngIndex As Long

   On Error GoTo QueryInstanceErr

   ' Check for valid values of the instance id
   If InstanceId > 0 Then
      ' Find the run node with the matching
step id
      For lngIndex = 0 To Count() - 1
         If mcInstances(lngIndex).InstanceId
= InstanceId Then
            Set QueryInstance =
mcInstances(lngIndex)
            Exit For
         End If
      Next lngIndex

      If lngIndex > mcInstances.Count - 1
Then
         On Error GoTo 0
         Err.Raise vbObjectError +
errQueryFailed, mstrSource, _
               LoadResString(errQueryFailed)
      End If
   Else
      On Error GoTo 0
      Err.Raise vbObjectError +
errQueryFailed, mstrSource, _
            LoadResString(errQueryFailed)
   End If

   Exit Function

QueryInstanceErr:
   ' Log the error code raised by Visual
Basic
   Call LogErrors(Errors)
   On Error GoTo 0
   mstrSource = mstrModuleName &
"QueryInstance"
   Err.Raise vbObjectError +
errQueryFailed, _
         mstrSource,
LoadResString(errQueryFailed)

End Function
Public Function QueryPendingInstance(ByVal
ParentInstanceId As Long, _
      ByVal lngSubStepId As Long) As
cInstance
   ' Retrieves a pending instance for the passed
in substep
   ' and the given parent instance id.

   Dim lngIndex As Long

   On Error GoTo QueryPendingInstanceErr

   ' Find the run node with the matching step id
   For lngIndex = 0 To Count() - 1
      If mcInstances(lngIndex).ParentInstanceId
= ParentInstanceId And _
            mcInstances(lngIndex).Step.StepId =
lngSubStepId Then
         ' Put in a separate if condition since the
IsPending
         ' property is valid only for manager
steps. If the
         ' calling procedure does not pass a
manager step
         ' identifier, the procedure will error out.
         If mcInstances(lngIndex).IsPending
Then
            Set QueryPendingInstance =
mcInstances(lngIndex)
            Exit For
         End If
      End If
   Next lngIndex

   Exit Function

QueryPendingInstanceErr:
   ' Log the error code raised by Visual Basic
   Call LogErrors(Errors)
   On Error GoTo 0
   mstrSource = mstrModuleName &
"QueryPendingInstance"
   Err.Raise vbObjectError + errQueryFailed, _
         mstrSource,
LoadResString(errQueryFailed)

End Function
Public Function InstanceAborted(cSubStepRec
As cSubStep) As Boolean

   Dim lIndex As Long

   InstanceAborted = False

   For lIndex = 0 To Count() - 1
      If mcInstances(lIndex).Step.StepId =
cSubStepRec.StepId And _
            mcInstances(lIndex).Status =
gintAborted Then
         InstanceAborted = True
         Exit For
      End If
   Next lIndex

End Function
Public Function
CompletedInstanceExists(lParentInstance As
Long, _
      cSubStepDtls As cStep) As Boolean
   ' Checks if there is a completed instance of
the passed in step

   Dim lngIndex As Long

   CompletedInstanceExists = False

   If cSubStepDtls.StepType = gintManagerStep
Then
      ' Find the run node with the matching step id
      For lngIndex = 0 To Count() - 1
         If mcInstances(lngIndex).ParentInstanceId =
lParentInstance And _
               mcInstances(lngIndex).Step.StepId =
cSubStepDtls.StepId Then
            ' Put in a separate if condition since the
IsPending
            ' property is valid only for manager steps.
            BugAssert (Not
mcInstances(lngIndex).IsPending), "Pending
instance exists!"

            CompletedInstanceExists = True
            Exit Function
         End If
      Next lngIndex
   End If

End Function
Public Sub Add(ByVal objItem As cInstance)

   mcInstances.Add objItem

End Sub


Public Sub Clear()

   mcInstances.Clear

End Sub


Public Function Count() As Long

   Count = mcInstances.Count

End Function


Public Function Delete(ByVal lngDelete As Long)
As cInstance

   Set Delete = mcInstances.Delete(lngDelete)

End Function


Public Property Set Item(Optional ByVal Position
As Long, _
      RHS As cInstance)

   If Position = -1 Then
      Position = 0
   End If
   Set mcInstances(Position) = RHS

End Property

Public Property Get Item(Optional ByVal Position
As Long = -1) _
      As cInstance
Attribute Item.VB_UserMemId = 0

   If Position = -1 Then
      Position = 0
   End If
   Set Item = mcInstances.Item(Position)

End Property
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
Private Sub Class_Initialize()

   Set mcInstances = New cVector

End Sub


Private Sub Class_Terminate()

   Set mcInstances = Nothing

End Sub


VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cIterator"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:       cIterator.cls
'              Microsoft TPC-H Kit Ver. 1.00
'              Copyright Microsoft, 1999
'              All Rights Reserved
'
'
'  PURPOSE:    Encapsulates the properties
and methods of an iterator.
'              Contains functions to insert,
update and delete
'              iterator_values records from the
database.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

Implements cNode

' Module level variables to store the property
values
Private mintType As Integer
Private mintSequenceNo As Integer
Private mstrValue As String
Private mdbsIteratorDB As Database
Private mintOperation As Integer
Private mlngPosition As Long

Private Const mstrModuleName As String =
"cIterator."
Private mstrSource As String

Public Enum ValueType
   gintFrom = 1
   gintTo
   gintStep
   gintValue
End Enum
Public Property Get Value() As String

   Value = mstrValue

End Property
Public Property Let Value(ByVal vdata As
String)

   mstrValue = vdata

End Property

Public Property Get IndOperation() As
Operation

   IndOperation = mintOperation

End Property
Public Property Let IndOperation(ByVal vdata
As Operation)

   On Error GoTo IndOperationErr
   mstrSource = mstrModuleName &
"IndOperation"

   ' The valid operations are define in the
cOperations
   ' class. Check if the operation is valid
   Select Case vdata
      Case QueryOp, InsertOp, UpdateOp,
DeleteOp
         mintOperation = vdata

      Case Else
         On Error GoTo 0
         Err.Raise vbObjectError +
errInvalidOperation, _
               mstrSource,
LoadResString(errInvalidOperation)
   End Select

   Exit Property

IndOperationErr:
   LogErrors Errors
   mstrSource = mstrModuleName &
"IndOperation"
   On Error GoTo 0
   Err.Raise vbObjectError +
errLetOperationFailed, _
      mstrSource,
LoadResString(errLetOperationFailed)

End Property

Public Function Clone() As cIterator

   ' Creates a copy of a given Iterator

   Dim cItClone As cIterator

   On Error GoTo CloneErr

   Set cItClone = New cIterator

   ' Copy all the iterator properties to the newly
   ' created object
   cItClone.IteratorType = mintType
   cItClone.SequenceNo = mintSequenceNo
   cItClone.IndOperation = mintOperation
   cItClone.Value = mstrValue

   ' And set the return value to the newly
created Iterator
   Set Clone = cItClone

   Exit Function

CloneErr:
   LogErrors Errors
   mstrSource = mstrModuleName & "Clone"
   On Error GoTo 0
   Err.Raise vbObjectError + errCloneFailed, _
         mstrSource,
LoadResString(errCloneFailed)

End Function
Public Property Get SequenceNo() As Integer

   SequenceNo = mintSequenceNo

End Property

Public Property Let SequenceNo(ByVal vdata As
Integer)
   mintSequenceNo = vdata
End Property

Public Sub Add(ByVal lngStepId As Long, _
      strVersion As String)
   ' Inserts a new iterator values record into the
database

   Dim strInsert As String
   Dim qy As DAO.QueryDef

   On Error GoTo AddIteratorErr

   ' First check if the database object is valid
   Call CheckDB

   ' Create a temporary querydef object
   strInsert = "insert into iterator_values " & _
         "( step_id, version_no, type, " & _
         " iterator_value, sequence_no ) " & _
         " values ( [st_id], [ver_no], [it_typ], " & _
         " [it_val], [seq_no] )"
   Set qy =
mdbsIteratorDB.CreateQueryDef(gstrEmptyString,
strInsert)

   ' Call a procedure to execute the Querydef object
   Call AssignParameters(qy, lngStepId, strVersion)

   qy.Execute dbFailOnError
   qy.Close

   Exit Sub

AddIteratorErr:
   LogErrors Errors
   mstrSource = mstrModuleName & "AddIterator"
   On Error GoTo 0
   Err.Raise vbObjectError + errInsertIteratorFailed,
_
         mstrSource, _
         LoadResString(errInsertIteratorFailed)
End Sub

Private Sub AssignParameters(qyExec As
DAO.QueryDef, _
      ByVal lngStepId As Long, _
      strVersion As String)
   ' Assigns values to the parameters in the querydef
object
   ' The parameter names are cryptic to make them
different
   ' from the field names. When the parameter
names are
   ' the same as the field names, parameters in the
where
   ' clause do not get created.

   Dim prmParam As DAO.Parameter

   On Error GoTo AssignParametersErr
   mstrSource = mstrModuleName &
"AssignParameters"

   For Each prmParam In qyExec.Parameters
      Select Case prmParam.Name
         Case "[st_id]"
            prmParam.Value = lngStepId
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
        Case "[ver_no]"
            prmParam.Value = strVersion

        Case "[it_typ]"
            prmParam.Value = mintType

        Case "[it_val]"
            prmParam.Value = mstrValue

        Case "[seq_no]"
            prmParam.Value =
mintSequenceNo

        Case Else
            ' Write the parameter name that is
faulty
            WriteError errInvalidParameter,
mstrSource, _
                prmParam.Name
            On Error GoTo 0
            Err.Raise errInvalidParameter,
mstrSource, _

LoadResString(errInvalidParameter)
    End Select
  Next prmParam

  Exit Sub

AssignParametersErr:

  mstrSource = mstrModuleName &
"AssignParameters"
  Call LogErrors(Errors)
  On Error GoTo 0
  Err.Raise vbObjectError +
errAssignParametersFailed, _
    mstrSource,
LoadResString(errAssignParametersFailed)

End Sub
Private Sub CheckDB()
  ' Check if the database object has been
initialized

  If mdbsIteratorDB Is Nothing Then
    ShowError errInvalidDB
    On Error GoTo 0
    Err.Raise vbObjectError +
errInvalidDB, _
      mstrModuleName,
LoadResString(errInvalidDB)
  End If

End Sub

Public Sub Delete(ByVal lngStepId As
Long, _
    strVersion As String)
  ' Deletes the step iterator record from the
database

  Dim strDelete As String
  Dim qy As DAO.QueryDef

  On Error GoTo DeleteIteratorErr
  mstrSource = mstrModuleName &
"DeleteIterator"

  ' There can be multiple iterators for a step.
  ' However the values that an iterator for a
step can
  ' assume will be unique, meaning that a
combination of
  ' the iterator_id and value will be unique.
  strDelete = "delete from iterator_values " &
_
        " where step_id = [st_id]" & _
        " and version_no = [ver_no]" & _
        " and iterator_value = [it_val] "
  Set qy =
mdbsIteratorDB.CreateQueryDef(gstrEmptyStr
ing, strDelete)

  Call AssignParameters(qy, lngStepId,
strVersion)
  qy.Execute dbFailOnError

  qy.Close

  Exit Sub

DeleteIteratorErr:
  LogErrors Errors
  mstrSource = mstrModuleName &
"DeleteIterator"
  On Error GoTo 0
  Err.Raise vbObjectError +
errDeleteIteratorFailed, _
      mstrSource, _
      LoadResString(errDeleteIteratorFailed)
End Sub
Public Sub Update(ByVal lngStepId As Long,
strVersion As String)
  ' Updates the sequence no of the step iterator
record
  ' in the database

  Dim strUpdate As String
  Dim qy As QueryDef

  On Error GoTo UpdateErr

  ' First check if the database object is valid
  Call CheckDB

  If mintType = gintValue Then
    ' If the iterator is of type value, only the
sequence of the values can get updated
    strUpdate = "Update iterator_values " & _
        " set sequence_no = [seq_no] " & _
        " where step_id = [st_id]" & _
        " and version_no = [ver_no]" & _
        " and iterator_value = [it_val] "
  Else
    ' If the iterator is of type range, only the
values can get updated
    strUpdate = "Update iterator_values " & _
        " set iterator_value = [it_val] " & _
        " where step_id = [st_id]" & _
        " and version_no = [ver_no]" & _
        " and type = [it_typ] "
  End If

  Set qy =
mdbsIteratorDB.CreateQueryDef(gstrEmptyStr
ing, strUpdate)

  ' Call a procedure to assign the parameter
values to the
  ' querydef object
  Call AssignParameters(qy, lngStepId,
strVersion)
  qy.Execute dbFailOnError

  qy.Close

  Exit Sub

UpdateErr:
  LogErrors Errors
  mstrSource = mstrModuleName & "Update"
  On Error GoTo 0
  Err.Raise vbObjectError +
errUpdateConstraintFailed, _
      mstrSource, _
      LoadResString(errUpdateConstraintFailed)

End Sub
Public Property Set NodeDB(vdata As Database)

  Set mdbsIteratorDB = vdata

End Property

Public Property Get NodeDB() As Database

  Set NodeDB = mdbsIteratorDB

End Property
Public Property Get Position() As Long

  Position = mlngPosition

End Property
Public Property Let Position(ByVal vdata As Long)

  mlngPosition = vdata

End Property

Public Property Let IteratorType(ByVal vdata As
ValueType)

  On Error GoTo TypeErr
  mstrSource = mstrModuleName & "Type"

  ' These constants have been defined in the
enumeration,
  ' Type, which is exposed
  Select Case vdata
    Case gintFrom, gintTo, gintStep, gintValue
      mintType = vdata

    Case Else
      On Error GoTo 0
      Err.Raise vbObjectError + errTypeInvalid, _
        mstrSource,
LoadResString(errTypeInvalid)
  End Select

  Exit Property

TypeErr:
  LogErrors Errors
  mstrSource = mstrModuleName & "Type"
  On Error GoTo 0
  Err.Raise vbObjectError + errTypeInvalid, _
    mstrSource, LoadResString(errTypeInvalid)

End Property

Public Property Get IteratorType() As ValueType

  IteratorType = mintType

End Property
Public Sub Validate()

  ' No validations necessary for the iterator class

End Sub

Private Sub Class_Initialize()
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
    ' Initialize the operation indicator variable
to Query
    ' It will be modified later by the collection
class when
    ' inserts, updates or deletes are performed
    mintOperation = QueryOp

End Sub


Private Property Let
cNode_IndOperation(ByVal vdata As
Operation)

    On Error GoTo IndOperationErr
    mstrSource = mstrModuleName &
"IndOperation"

    ' The valid operations are define in the
cOperations
    ' class. Check if the operation is valid
    Select Case vdata
        Case QueryOp, InsertOp, UpdateOp,
DeleteOp
            mintOperation = vdata

        Case Else
            On Error GoTo 0
            Err.Raise vbObjectError +
errInvalidOperation, _
                mstrSource,
LoadResString(errInvalidOperation)
    End Select

    Exit Property

IndOperationErr:
    LogErrors Errors
    mstrSource = mstrModuleName &
"IndOperation"
    On Error GoTo 0
    Err.Raise vbObjectError +
errLetOperationFailed, _
        mstrSource,
LoadResString(errLetOperationFailed)

End Property

Private Property Get cNode_IndOperation()
As Operation

    IndOperation = mintOperation

End Property

Private Property Set cNode_NodeDB(RHS
As DAO.Database)

    Set mdbsIteratorDB = RHS

End Property

Private Property Get cNode_NodeDB() As
DAO.Database

    Set cNode_NodeDB = mdbsIteratorDB

End Property


Private Property Let cNode_Position(ByVal
vdata As Long)

    mlngPosition = vdata

End Property


Private Property Get cNode_Position() As
Long

    cNode_Position = mlngPosition

End Property


Private Sub cNode_Validate()

    ' No validations necessary for the iterator
class

End Sub


Private Property Let cNode_Value(ByVal
vdata As String)

    mstrValue = vdata

End Property

Private Property Get cNode_Value() As String

    Value = mstrValue

End Property
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cManager"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:       cManager.cls
'              Microsoft TPC-H Kit Ver. 1.00
'              Copyright Microsoft, 1999
'              All Rights Reserved
'
'
'  PURPOSE:     Encapsulates the properties
and methods of a manager step.
'              Implements the cStep class - carries
out initializations
'              and validations that are specific to
manager steps.
'  Contact:     Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

Implements cStep

' Object variable to keep the step reference in
Private mcStep As cStep

' Used to indicate the source module name
when errors
' are raised by this class
Private mstrSource As String
Private Const mstrModuleName As String =
"cManager."
Private Sub cStep_AddAllIterators()

    Call mcStep.AddAllIterators

End Sub

Private Property Let cStep_StartDir(ByVal RHS As
String)

    mcStep.StartDir = RHS

End Property

Private Property Get cStep_StartDir() As String

    cStep_StartDir = mcStep.StartDir

End Property
Private Sub cStep_Delete()

    Call mcStep.Delete

End Sub

Private Property Set cStep_NodeDB(RHS As
DAO.Database)

    Set mcStep.NodeDB = RHS

End Property

Private Function cStep_IncVersionY() As String

    cStep_IncVersionY = mcStep.IncVersionY

End Function
Private Function cStep_IsNewVersion() As
Boolean
    cStep_IsNewVersion = mcStep.IsNewVersion
End Function
Private Function cStep_OldVersionNo() As String
    cStep_OldVersionNo = mcStep.OldVersionNo
End Function

Private Function cStep_IncVersionX() As String

    cStep_IncVersionX = mcStep.IncVersionX

End Function
Private Sub cStep_UpdateIteratorVersion()

    Call mcStep.UpdateIteratorVersion

End Sub

Private Function cStep_IteratorCount() As Long

    cStep_IteratorCount = mcStep.IteratorCount

End Function

Private Sub cStep_UnloadIterators()

    Call mcStep.UnloadIterators

End Sub

Private Sub cStep_DeleteIterator(cItRecord As
cIterator)

    Call mcStep.DeleteIterator(cItRecord)

End Sub
Private Property Get cStep_IteratorName() As
String

    cStep_IteratorName = mcStep.IteratorName

End Property
Private Property Let cStep_IteratorName(ByVal
RHS As String)
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
    mcStep.IteratorName = RHS

End Property

Private Sub cStep_SaveIterators()

    Call mcStep.SaveIterators

End Sub
Private Sub cStep_LoadIterator(cItRecord
As cIterator)

    Call mcStep.LoadIterator(cItRecord)

End Sub

Private Property Let cStep_Position(ByVal
RHS As Long)

    mcStep.Position = RHS

End Property
Private Sub cStep_InsertIterator(cItRecord
As cIterator)

    Call mcStep.InsertIterator(cItRecord)

End Sub
Private Function cStep_Iterators() As
Variant

    cStep_Iterators = mcStep.Iterators

End Function
Private Sub cStep_ModifyIterator(cItRecord
As cIterator)

    Call mcStep.ModifyIterator(cItRecord)

End Sub
Private Sub
cStep_RemoveIterator(cItRecord As
cIterator)

    Call mcStep.RemoveIterator(cItRecord)

End Sub
Private Sub cStep_UpdateIterator(cItRecord
As cIterator)

    Call mcStep.UpdateIterator(cItRecord)

End Sub
Private Sub cStep_AddIterator(cItRecord As
cIterator)

    Call mcStep.AddIterator(cItRecord)

End Sub

Private Property Get cStep_Position() As
Long

    cStep_Position = mcStep.Position

End Property

Private Function cStep_Clone(Optional
cCloneStep As cStep) As cStep

    Dim cNewManager As cManager

    Set cNewManager = New cManager

    Set cStep_Clone =
mcStep.Clone(cNewManager)

End Function

Private Property Get cStep_IndOperation() As
Operation

    cStep_IndOperation = mcStep.IndOperation

End Property

Private Property Let
cStep_IndOperation(ByVal RHS As
Operation)

    mcStep.IndOperation = RHS

End Property

Private Property Get cStep_NextStepId() As
Long

    cStep_NextStepId = mcStep.NextStepId

End Property

Private Property Let cStep_OutputFile(ByVal
RHS As String)

    mcStep.OutputFile = RHS

End Property

Private Property Get cStep_OutputFile() As
String

    cStep_OutputFile = mcStep.OutputFile

End Property

Private Property Let cStep_ErrorFile(ByVal
RHS As String)

    mcStep.ErrorFile = RHS

End Property

Private Property Get cStep_ErrorFile() As
String

    cStep_ErrorFile = mcStep.ErrorFile

End Property
'Private Property Let cStep_LogFile(ByVal
RHS As String)
'
'    mcStep.LogFile = RHS
'
'End Property
'
'Private Property Get cStep_LogFile() As
String
'
'    cStep_LogFile = mcStep.LogFile
'
'End Property

Private Property Let
cStep_ArchivedFlag(ByVal RHS As Boolean)

    mcStep.ArchivedFlag = RHS

End Property

Private Property Get cStep_ArchivedFlag() As
Boolean

    cStep_ArchivedFlag = mcStep.ArchivedFlag

End Property

Private Property Get cStep_NodeDB() As
DAO.Database

    Set cStep_NodeDB = mcStep.NodeDB

End Property

Private Sub Class_Initialize()

    ' Create the object
    Set mcStep = New cStep

    ' Initialize the object with valid values for a
manager step
    ' The global flag should be the first field to be
initialized
    ' since subsequent validations might try to check
if the
    ' step being created is global
    mcStep.GlobalFlag = False
'    mcStep.GlobalRunMethod = gintNoOption
    mcStep.StepType = gintManagerStep

    ' Since the manager step does not take any action,
the step
    ' text and file name will always be empty
    mcStep.StepText = gstrEmptyString
    mcStep.StepTextFile = gstrEmptyString

    ' Since the manager step does not take any action,
execution
    ' properties for the step will be empty
    mcStep.ExecutionMechanism = gintNoOption
    mcStep.FailureDetails = gstrEmptyString
    mcStep.ContinuationCriteria = gintNoOption

End Sub
Private Sub Class_Terminate()

    ' Remove the step object
    Set mcStep = Nothing

End Sub
Private Sub cStep_Add()

    ' Call the Add method of the step class to carry
out the insert
    mcStep.Add

End Sub
Private Property Get cStep_ContinuationCriteria()
As ContinuationCriteria

    cStep_ContinuationCriteria =
mcStep.ContinuationCriteria

End Property

Private Property Let
cStep_ContinuationCriteria(ByVal RHS As
ContinuationCriteria)

    ' Since a manager step cannot take any action, the
continuation
    ' criteria property does not apply to it
    mcStep.ContinuationCriteria = gintNoOption

End Property
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
Private Property Let
cStep_DegreeParallelism(ByVal RHS As
String)

   mcStep.DegreeParallelism = RHS

End Property

Private Property Get
cStep_DegreeParallelism() As String

   cStep_DegreeParallelism =
mcStep.DegreeParallelism

End Property

Private Sub cStep_DeleteStep()

   On Error GoTo cStep_DeleteStepErr
   mstrSource = mstrModuleName &
"cStep_DeleteStep"

   mcStep.Delete
   Exit Sub

cStep_DeleteStepErr:
   LogErrors Errors
   mstrSource = mstrModuleName &
"cStep_DeleteStep"
   On Error GoTo 0
   Err.Raise vbObjectError +
errDeleteStepFailed, _
         mstrSource, _
         LoadResString(errDeleteStepFailed)

End Sub

Private Property Get cStep_EnabledFlag()
As Boolean

   cStep_EnabledFlag =
mcStep.EnabledFlag

End Property

Private Property Let
cStep_EnabledFlag(ByVal RHS As
Boolean)

   mcStep.EnabledFlag = RHS

End Property

Private Property Let
cStep_ExecutionMechanism(ByVal RHS As
ExecutionMethod)

   ' Since a manager step cannot take any
action, the Execution
   ' Mechanism property does not apply to it
   mcStep.ExecutionMechanism =
gintNoOption

End Property

Private Property Get
cStep_ExecutionMechanism() As
ExecutionMethod

   cStep_ExecutionMechanism =
mcStep.ExecutionMechanism

End Property

Private Property Let
cStep_FailureDetails(ByVal RHS As String)

   ' Since a manager step cannot take any
action, the Failure
   ' Details property does not apply to it
   mcStep.FailureDetails = gstrEmptyString

End Property

Private Property Get cStep_FailureDetails() As
String

   cStep_FailureDetails =
mcStep.FailureDetails

End Property

Private Property Get cStep_GlobalFlag() As
Boolean

   cStep_GlobalFlag = mcStep.GlobalFlag

End Property

Private Property Let cStep_GlobalFlag(ByVal
RHS As Boolean)

   ' Set the global flag to false - this flag is
initialized when
   ' an instance of the class is created. Just
making sure that
   ' nobody changes the value inadvertently
   mcStep.GlobalFlag = False

End Property
Private Sub cStep_Modify()

   ' Call the Modify method of the step class to
carry out the update
   mcStep.Modify

End Sub

Private Property Let
cStep_ParentStepId(ByVal RHS As Long)

   mcStep.ParentStepId = RHS

End Property

Private Property Get cStep_ParentStepId() As
Long

   cStep_ParentStepId = mcStep.ParentStepId

End Property

Private Property Let
cStep_ParentVersionNo(ByVal RHS As
String)

   mcStep.ParentVersionNo = RHS

End Property

Private Property Get cStep_ParentVersionNo()
As String

   cStep_ParentVersionNo =
mcStep.ParentVersionNo

End Property

Private Property Let cStep_SequenceNo(ByVal
RHS As Integer)

   mcStep.SequenceNo = RHS

End Property

Private Property Get cStep_SequenceNo() As
Integer

   cStep_SequenceNo = mcStep.SequenceNo

End Property

Private Property Let cStep_StepId(ByVal RHS As
Long)

   mcStep.StepId = RHS

End Property

Private Property Get cStep_StepId() As Long

   cStep_StepId = mcStep.StepId

End Property


Private Property Let cStep_StepLabel(ByVal RHS
As String)

   mcStep.StepLabel = RHS

End Property

Private Property Get cStep_StepLabel() As String

   cStep_StepLabel = mcStep.StepLabel

End Property


Private Property Let cStep_StepLevel(ByVal RHS
As Integer)

   mcStep.StepLevel = RHS

End Property

Private Property Get cStep_StepLevel() As Integer

   cStep_StepLevel = mcStep.StepLevel

End Property

Private Property Let cStep_StepText(ByVal RHS
As String)

   ' Since the manager step does not take any action,
the step
   ' text and file name will always be empty
   mcStep.StepText = gstrEmptyString

End Property

Private Property Get cStep_StepText() As String

   cStep_StepText = mcStep.StepText

End Property

Private Property Let cStep_StepTextFile(ByVal
RHS As String)
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   ' Since the manager step does not take any
action, the step
   ' text and file name will always be empty
   mcStep.StepTextFile = gstrEmptyString

End Property

Private Property Get cStep_StepTextFile()
As String

   cStep_StepTextFile =
mcStep.StepTextFile

End Property

Private Property Let cStep_StepType(RHS
As gintStepType)

   mcStep.StepType = gintManagerStep

End Property

Private Property Get cStep_StepType() As
gintStepType

   cStep_StepType = mcStep.StepType

End Property

Private Sub cStep_Validate()
   ' The validate routines for each of the
steps will
   ' carry out the specific validations for the
type and
   ' call the generic validation routine

   On Error GoTo cStep_ValidateErr
   mstrSource = mstrModuleName &
"cStep_Validate"

   ' Validations specific to manager steps

   ' Check if the step text or a file name has
been
   ' specified
   If Not StringEmpty(mcStep.StepText) Or
Not StringEmpty(mcStep.StepTextFile)
Then
      ShowError
errTextAndFileNullForManager
      On Error GoTo 0
      Err.Raise vbObjectError +
errValidateFailed, _
            gstrSource, _
            LoadResString(errValidateFailed)
   End If

   If mcStep.ExecutionMechanism <>
gintNoOption Then
      ShowError
errExecutionMechanismInvalid
      On Error GoTo 0
      Err.Raise vbObjectError +
errValidateFailed, _
            gstrSource, _
            LoadResString(errValidateFailed)
   End If

   If mcStep.FailureDetails <>
gstrEmptyString Then
      ShowError
errFailureDetailsNullForMgr
      On Error GoTo 0
      Err.Raise vbObjectError +
errValidateFailed, _
            gstrSource, _
            LoadResString(errValidateFailed)
   End If

   If mcStep.ContinuationCriteria <>
gintNoOption Then
      ShowError errContCriteriaInvalid
      On Error GoTo 0
      Err.Raise vbObjectError +
errValidateFailed, _
            gstrSource, _
            LoadResString(errValidateFailed)
   End If

   mcStep.Validate

   Exit Sub

cStep_ValidateErr:
   LogErrors Errors
   mstrSource = mstrModuleName &
"cStep_Validate"
   On Error GoTo 0
   Err.Raise vbObjectError +
errValidateFailed, _
         mstrSource, _
         LoadResString(errValidateFailed)
End Sub

Private Property Let cStep_VersionNo(ByVal
RHS As String)

   mcStep.VersionNo = RHS

End Property

Private Property Get cStep_VersionNo() As
String

   cStep_VersionNo = mcStep.VersionNo

End Property

Private Property Let
cStep_WorkspaceId(ByVal RHS As Long)

   mcStep.WorkspaceId = RHS

End Property

Private Property Get cStep_WorkspaceId() As
Long

   cStep_WorkspaceId = mcStep.WorkspaceId

End Property
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cNode"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:       cNode.cls
'             Microsoft TPC-H Kit Ver. 1.00
'             Copyright Microsoft, 1999
'             All Rights Reserved
'
'
'  PURPOSE:    Defines the properties that an
object has to implement.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

Public Property Get IndOperation() As Operation
End Property
Public Property Let IndOperation(ByVal vdata As
Operation)
End Property
Public Sub Validate()
End Sub
Public Property Get Value() As String
End Property
Public Property Let Value(ByVal vdata As String)
End Property

Public Property Get NodeDB() As Database
End Property
Public Property Set NodeDB(vdata As Database)
End Property

Public Property Get Position() As Long
End Property
Public Property Let Position(ByVal vdata As Long)
End Property

VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cNodeCollections"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:       cNodeCollections.cls
'             Microsoft TPC-H Kit Ver. 1.00
'             Copyright Microsoft, 1999
'             All Rights Reserved
'
'
'  PURPOSE:    Implements an array of objects.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

' Node counter
Private mlngNodeCount As Long
Private mdbsNodeDb As Database
Private mcarrNodes() As Object

' Used to indicate the source module name when
errors
' are raised by this class
Private mstrSource As String
Private Const mstrModuleName As String =
"cNodeCollections."

Public Property Set Item(ByVal Position As Long,
_
      ByVal objNode As Object)

   ' Returns the element at the passed in position in
the array
   If Position >= 0 And Position < mlngNodeCount
Then
      Set mcarrNodes(Position) = objNode
   Else
      On Error GoTo 0
      Err.Raise vbObjectError +
errItemDoesNotExist, mstrSource, _
         LoadResString(errItemDoesNotExist)
   End If

End Property
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
Public Property Get Item(ByVal Position As
Long) As Object
Attribute Item.VB_UserMemId = 0

   ' Returns the element at the passed in
position in the array
   If Position >= 0 And Position <
mlngNodeCount Then
      Set Item = mcarrNodes(Position)
   Else
      On Error GoTo 0
      Err.Raise vbObjectError +
errItemDoesNotExist, mstrSource, _

LoadResString(errItemDoesNotExist)
   End If

End Property

Public Sub Commit(ByVal cSaveObj As
Object, _
      ByVal lngIndex As Long)
   ' This procedure checks if any changes
have been made to the
   ' passed in object. If so, it calls the
corresponding method
   ' to commit the changes.

   On Error GoTo CommitErr
   mstrSource = mstrModuleName &
"Commit"

   Select Case cSaveObj.IndOperation
      Case QueryOp
         ' No changes were made to the
queried parameter.
         ' Do nothing

      Case InsertOp
         cSaveObj.Add
         cSaveObj.IndOperation = QueryOp

      Case UpdateOp
         cSaveObj.Modify
         cSaveObj.IndOperation = QueryOp

      Case DeleteOp
         cSaveObj.Delete
         ' Now we can remove the record
from the array
         Call Unload(lngIndex)

   End Select

   Exit Sub

CommitErr:
   LogErrors Errors
   mstrSource = mstrModuleName &
"Commit"
   On Error GoTo 0
   Err.Raise vbObjectError +
errCommitFailed, _
         mstrSource, _
         LoadResString(errCommitFailed)

End Sub
Public Sub Save(ByVal lngWorkspace As
Long)
   ' Calls a procedure to commit all changes
for the passed
   ' in workspace.

   Dim lngIndex As Long

   On Error GoTo SaveErr

   ' Find all parameters in the array with a
matching workspace id
   ' It is important to step backwards through
the array, since
   ' we delete parameter records as we go
along!
   For lngIndex = mlngNodeCount - 1 To 0
Step -1
      If mcarrNodes(lngIndex).WorkspaceId =
lngWorkspace Then

         ' Call a procedure to commit all
changes to the
         ' parameter record, if any
         Call Commit(mcarrNodes(lngIndex),
lngIndex)

      End If
   Next lngIndex

   Exit Sub

SaveErr:
   LogErrors Errors
   mstrSource = mstrModuleName & "Save"
   On Error GoTo 0
   Err.Raise vbObjectError + errSaveFailed, _
         mstrSource, _
         LoadResString(errSaveFailed)

End Sub
Public Property Get Count() As Long

   Count = mlngNodeCount

End Property

Public Property Get NodeDB() As Database

   Set NodeDB = mdbsNodeDb

End Property
Public Property Set NodeDB(vdata As
Database)

   Set mdbsNodeDb = vdata

End Property

Public Sub Load(cNodeToLoad As Object)
   ' Adds the passed in object to the array

   On Error GoTo LoadErr

   ' If this procedure is called by the add to
array procedure,
   ' the database object has already been
initialized
   If cNodeToLoad.NodeDB Is Nothing Then

      ' All the Nodes will be initialized with the
database
      ' objects before being added to the array
      Set cNodeToLoad.NodeDB =
mdbsNodeDb

   End If

   ReDim Preserve
mcarrNodes(mlngNodeCount)

   ' Set the newly added element in the array to
the passed in Node
   cNodeToLoad.Position = mlngNodeCount
   Set mcarrNodes(mlngNodeCount) =
cNodeToLoad

   mlngNodeCount = mlngNodeCount + 1

   Exit Sub

LoadErr:
   LogErrors Errors
   On Error GoTo 0
   Err.Raise vbObjectError + errLoadFailed,
mstrModuleName & "Load", _
         LoadResString(errLoadFailed)

End Sub
Public Sub Unload(lngDeletePosition As Long)
   ' Unloads the passed in object from the array

   On Error GoTo UnloadErr

   If lngDeletePosition < (mlngNodeCount - 1)
Then

      ' Set the Node at the position being deleted to
      ' the last Node in the Node array
      Set mcarrNodes(lngDeletePosition) =
mcarrNodes(mlngNodeCount - 1)
      mcarrNodes(lngDeletePosition).Position =
lngDeletePosition
   End If

   ' Delete the last Node from the array
   mlngNodeCount = mlngNodeCount - 1
   If mlngNodeCount > 0 Then
      ReDim Preserve mcarrNodes(0 To
mlngNodeCount - 1)
   Else
      ReDim mcarrNodes(0)
   End If

   Exit Sub

UnloadErr:
   LogErrors Errors
   mstrSource = mstrModuleName & "Unload"
   On Error GoTo 0
   Err.Raise vbObjectError + errUnloadFailed, _
         mstrSource, _
         LoadResString(errUnloadFailed)

End Sub
Public Sub Delete(lngDeletePosition As Long)
   ' Deletes the object at the specified position in the
   ' array

   Dim cDeleteObj As Object

   On Error GoTo DeleteErr
   mstrSource = mstrModuleName & "Delete"

   Set cDeleteObj = mcarrNodes(lngDeletePosition)

   If cDeleteObj.IndOperation = InsertOp Then
      ' If we are deleting a record that has just been
inserted,
      ' blow it away
      Call Unload(lngDeletePosition)
   Else
      ' Set the operation for the deleted object to
indicate a
      ' delete - we actually delete the element only at
the time
      ' of a save operation
      cDeleteObj.IndOperation = DeleteOp
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   End If

   Exit Sub

DeleteErr:
   LogErrors Errors
   mstrSource = mstrModuleName &
"Delete"
   On Error GoTo 0
   Err.Raise vbObjectError +
errDeleteFailed, _
         mstrSource, _
         LoadResString(errDeleteFailed)

End Sub
Public Sub Modify(cModifiedNode As
Object)
   ' Sets the object at the passed in position
to the
   ' modified object passed in

   On Error GoTo ModifyErr

   ' First check if the record is valid - all
objects that
   ' use this collection class must have a
Validate routine
   cModifiedNode.Validate

   ' If we are updating a record that hasn't yet
been inserted,
   ' do not change the operation indicator -
or we try to update
   ' a non-existant record
   If cModifiedNode.IndOperation <>
InsertOp Then
      ' Set the operations to indicate an
update
      cModifiedNode.IndOperation =
UpdateOp
   End If

   ' Modify the object at the queried position
- the Position
   ' will be maintained by this class
   Set mcarrNodes(cModifiedNode.Position)
= cModifiedNode

   Exit Sub

ModifyErr:
   LogErrors Errors
   mstrSource = mstrModuleName &
"Modify"
   On Error GoTo 0
   Err.Raise vbObjectError +
errModifyFailed, _
         mstrSource, _
         LoadResString(errModifyFailed)

End Sub
Public Sub Add(cNodeToAdd As Object)

   On Error GoTo AddErr

   Set cNodeToAdd.NodeDB =
mdbsNodeDb

   ' First check if the record is valid
   cNodeToAdd.Validate

   ' Set the operation to indicate an insert
   cNodeToAdd.IndOperation = InsertOp

   ' Call a procedure to load the record in the
array
   Call Load(cNodeToAdd)

   Exit Sub

AddErr:
   LogErrors Errors
   mstrSource = mstrModuleName & "Add"
   On Error GoTo 0
   Err.Raise vbObjectError + errAddFailed, _
         mstrSource, _
         LoadResString(errAddFailed)

End Sub

Private Sub Class_Terminate()

   ReDim mcarrNodes(0)
   mlngNodeCount = 0

End Sub


Attribute VB_Name = "Common"
'  FILE:      Common.bas
'             Microsoft TPC-H Kit Ver. 1.00
'             Copyright Microsoft, 1999
'             All Rights Reserved
'
'
'  PURPOSE:    Module containing common
functionality throughout
'             StepMaster
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

Private Const mstrModuleName As String =
"Common."

' Used to separate the variable data from the
constant error
' message being raised when a context-sensitive
error is displayed
Private Const mintDelimiter As String = " : "
Private Const mstrFormatString = "mmddyy"

' Identifiers for the different labels that need to
be loaded
' into the tree view for each workspace
Public Const mstrWorkspacePrefix = "W"
Public Const mstrParameterPrefix = "P"
Public Const mstrParamConnectionPrefix =
"C"
Public Const mstrConnectionDtlPrefix = "N"
Public Const mstrParamExtensionPrefix = "E"
Public Const mstrParamBuiltInPrefix = "B"
Public Const gstrGlobalStepPrefix = "G"
Public Const gstrManagerStepPrefix = "M"
Public Const gstrWorkerStepPrefix = "S"
Public Const gstrDummyPrefix = "D"
Public Const mstrLabelPrefix = "L"
Public Const mstrInstancePrefix = "I"
Public Function
LabelStep(lngWorkspaceIdentifier As Long)
As String
   ' Returns the step label for the workspace
identifier passed in
   ' Basically this is a wrapper around the
MakeKeyValid function

   LabelStep = MakeKeyValid(gintStepLabel,
gintStepLabel, lngWorkspaceIdentifier)

End Function


Public Function JulianDateToString(dt64Bit As
Currency) As String

   Dim lYear As Long
   Dim lMonth As Long
   Dim lDay As Long
   Dim lHour As Long
   Dim lMin As Long
   Dim lSec As Long
   Dim lMs As Long

   Call JulianToTime(dt64Bit, lYear, lMonth, lDay,
lHour, lMin, lSec, lMs)
   JulianDateToString = Format$(lYear,
gsYearFormat) & gsDateSeparator & _
         Format$(lMonth, gsDtFormat) &
gsDateSeparator & _
         Format$(lDay, gsDtFormat) & gstrBlank &
_
         Format$(lHour, gsTmFormat) &
gsTimeSeparator & _
         Format$(lMin, gsTmFormat) &
gsTimeSeparator & _
         Format$(lSec, gsTmFormat) &
gsMsSeparator & _
         Format$(lMs, gsMSecondFormat)

End Function
Public Sub DeleteFile(strFile As String, Optional
ByVal bCheckIfEmpty As Boolean = False)

   ' Ensure that there is only a single file of the
name before delete, since
   ' Kill supports wildcards and can potentially
delete a number of files
   Dim strTemp As String

   If CheckFileExists(strFile) Then
      If bCheckIfEmpty Then
         If FileLen(strFile) = 0 Then
            Kill strFile
         End If
      Else
         Kill strFile
      End If
   End If

End Sub
Public Function CheckFileExists(strFile As String)
As Boolean

   ' Returns true if the passed in file exists
   ' Raises an error if multiple files are found
(filename contains a wildcard)
   CheckFileExists = False

   If Not StringEmpty(Dir(strFile)) Then
      If Not StringEmpty(Dir()) Then
         On Error GoTo 0
         Err.Raise vbObjectError +
errDeleteSingleFile, _
            mstrModuleName & "DeleteFile",
LoadResString(errDeleteSingleFile)
      End If

      CheckFileExists = True
   End If

End Function

Public Function GetVersionString() As String
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
GetVersionString = "Version " & gsVersion
End Function

Function IsLabel(strKey As String) As
Boolean

   ' The tree view control on frmMain can
contain two types of
   ' nodes -
   ' 1. Nodes that contain data for the
workspace - this could
   ' be data for the different types of steps or
parameters
   ' 2. Nodes that display static data - these
kind of nodes
   ' are referred to as label nodes e.g.
"Global Steps" is a
   ' label node
   ' This function returns True if the passed
in key corresponds
   ' to a label node

   IsLabel = InStr(strKey, mstrLabelPrefix)
> 0

End Function

Function MakeKeyValid(lngIdentifier As
Long, _
   intTypeOfNode As Integer, _
   Optional ByVal WorkspaceId As Long =
0, _
   Optional ByVal InstanceId As Long = 0)
As String

   ' We use a numbering scheme while
loading the tree view with
   ' all node data, since it needs a unique key
and we want to
   ' use the key to identify the data it
contains.
   ' Moreover, add a character to the
beginning of the identifier
   ' so that the tree view control accepts it as
a valid string,
   ' viz. "456" doesn't work, so change it to
"W456"
   ' The general scheme is to concatenate a
Label with the Identifier
   ' e.g A Global Step Node will have the
Label, G and the Step Id
   ' concatenated to form the unique key
   ' The list of all such node types is given
below
   ' 1. "W" + Workspace_Id for Workspace
nodes
   ' 2. "P" + Parameter_Id for Parameter
nodes
   ' 3. "M" + Step_Id for Manager Step
nodes
   ' 4. "S" + Step_Id for Worker Step nodes
   ' 5. "G" + Step_Id for Global Step nodes
   ' 6. Instance_id + "I" + Step_Id for
Instance nodes
   ' 7. Workspace_id + "L" + the label
identifier = node type for all Label nodes
   ' Since the manager, worker and global
steps are stored in the
   ' same table and the step identifiers will
always be unique, we
   ' can use the same character as the prefix,
but this is a
   ' convenient way to know the type of step
being processed.
   ' The workspace id is appended to the label
identifier to make
   ' it unique, since multiple workspaces may
be open during a session
   ' Strip the prefix characters off while saving
the Ids to the db

   Dim strPrefixChar As String

   On Error GoTo MakeKeyValidErr
   gstrSource = mstrModuleName &
"MakeKeyValid"

   Select Case intTypeOfNode
      Case gintWorkspace
         strPrefixChar = mstrWorkspacePrefix
      Case gintGlobalStep
         strPrefixChar = gstrGlobalStepPrefix
      Case gintManagerStep
         strPrefixChar = gstrManagerStepPrefix
      Case gintWorkerStep
         strPrefixChar = gstrWorkerStepPrefix
      Case gintRunManager, gintRunWorker
         If InstanceId = 0 Then
            On Error GoTo 0
            Err.Raise vbObjectError +
errMandatoryParameterMissing, _
               gstrSource, _

LoadResString(errMandatoryParameterMissin
g)
         End If
         ' Concatenate the instance identifier and
the step
         ' identifier to form a unique key
         strPrefixChar =
Trim$(Str$(InstanceId)) & mstrInstancePrefix
      Case gintParameter
         strPrefixChar = mstrParameterPrefix
      Case gintNodeParamConnection
         strPrefixChar =
mstrParamConnectionPrefix
      Case gintConnectionDtl
         strPrefixChar =
mstrConnectionDtlPrefix
      Case gintNodeParamExtension
         strPrefixChar =
mstrParamExtensionPrefix
      Case gintNodeParamBuiltIn
         strPrefixChar = mstrParamBuiltInPrefix
      Case gintGlobalsLabel,
gintParameterLabel,
gintParamConnectionLabel, _
            gintConnDtlLabel, _
            gintParamExtensionLabel,
gintParamBuiltInLabel, gintGlobalStepLabel, _
            gintStepLabel
         If WorkspaceId = 0 Then
            ' The Workspace Id has to be
specified for a label node
            ' Otherwise it will not be possible to
generate unique label
            ' identifiers if multiple workspaces
are open
            On Error GoTo 0
            Err.Raise vbObjectError +
errWorkspaceIdMandatory, _
               gstrSource, _

LoadResString(errWorkspaceIdMandatory)
         End If
         ' For all labels, the workspace identifier
and the
         ' label prefix are concatenated to form
the key
         strPrefixChar = Trim$(Str$(WorkspaceId))
& mstrLabelPrefix
      Case Else
            On Error GoTo 0
            Err.Raise vbObjectError +
errInvalidNodeType, _
               gstrSource, _
               LoadResString(errInvalidNodeType)
   End Select

   MakeKeyValid = strPrefixChar &
Trim$(Str$(lngIdentifier))

   Exit Function

MakeKeyValidErr:
   Call LogErrors(Errors)
   On Error GoTo 0
   Err.Raise vbObjectError +
errMakeKeyValidFailed, _
         gstrSource, _
         LoadResString(errMakeKeyValidFailed)

End Function

Function MakeIdentifierValid(strKey As String) As
Long

   ' Returns the Identifier corresponding to the
passed in key
   ' (Reverse of what was done in MakeKeyValid)

   On Error GoTo MakeIdentifierValidErr

   If IsLabel(strKey) Then
      ' If the key corresponds to a label node, the
identifier
      ' appears to the right of the label prefix
      MakeIdentifierValid = Val(Mid(strKey,
InStr(strKey, mstrLabelPrefix) + 1))
   ElseIf InStr(strKey, mstrInstancePrefix) = 0 Then
      ' For all other nodes, stripping the first
character off
      ' returns a valid Id
      MakeIdentifierValid = Val(Mid(strKey, 2))
   Else
      ' Instance node - strip of all characters till the
      ' instance prefix
      MakeIdentifierValid = Val(Mid(strKey,
InStr(strKey, mstrInstancePrefix) + 1))
   End If

   Exit Function

MakeIdentifierValidErr:
   Call LogErrors(Errors)
   On Error GoTo 0
   Err.Raise vbObjectError +
errMakeIdentifierValidFailed, _
         mstrModuleName & "MakeIdentifierValid",
_

LoadResString(errMakeIdentifierValidFailed)

End Function
Public Function IsInstanceNode(strNodeKey As
String) As Boolean

   ' Returns true if the passed in node key
corresponds to a step instance
   IsInstanceNode = InStr(strNodeKey,
mstrInstancePrefix) > 0

End Function
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
Public Function IsBuiltInLabel(strNodeKey
As String) As Boolean

   ' Returns true if the passed in node key
corresponds to a step instance
   IsBuiltInLabel = (IsLabel(strNodeKey)
And _
       (MakeIdentifierValid(strNodeKey) =
gintParamBuiltInLabel))

End Function

Public Sub ShowBusy()
   ' Modifies the mousepointer to indicate
that the
   ' application is busy

   On Error Resume Next

   Screen.MousePointer = vbHourglass

End Sub
Public Sub ShowFree()
   ' Modifies the mousepointer to indicate
that the
   ' application has finished processing and
is ready
   ' to accept user input

   On Error Resume Next

   Screen.MousePointer = vbDefault

End Sub

Public Function InstrR(strMain As String, _
       strSearch As String) As Integer
   ' Finds the last occurrence of the passed in
string

   Dim intPos As Integer
   Dim intPrev As Integer

   On Error GoTo InstrRErr

   intPrev = intPos
   intPos = InStr(1, strMain, strSearch)

   Do While intPos > 0
      intPrev = intPos
      intPos = InStr(intPos + 1, strMain,
strSearch)
   Loop
   InstrR = intPrev

   Exit Function

InstrRErr:
   Call LogErrors(Errors)
   gstrSource = mstrModuleName &
"InstrR"
   On Error GoTo 0
   Err.Raise vbObjectError +
errInstrRFailed, _
       gstrSource, _
       LoadResString(errInstrRFailed)

End Function

Public Function GetDefaultDir(lWspId As
Long, WspParameters As cArrParameters)
As String

   Dim sDir As String
   sDir = SubstituteParameters( _
          gstrEnvVarSeparator &
PARAM_DEFAULT_DIR &
gstrEnvVarSeparator, _
          lWspId,
WspParameters:=WspParameters)
   MakePathValid (sDir & gstrFileSeparator &
"a.txt")
   GetDefaultDir = GetShortName(sDir)
   If StringEmpty(GetDefaultDir) Then
      GetDefaultDir = App.Path
   End If

End Function

Public Sub AddArrayElement(ByRef
arrNodes() As Object, _
       ByVal objToAdd As Object, _
       ByRef lngCount As Long)
   ' Adds the passed in object to the array

   On Error GoTo AddArrayElementErr

   ' Increase the array dimension and add the
object to it
   ReDim Preserve arrNodes(lngCount)
   Set arrNodes(lngCount) = objToAdd
   lngCount = lngCount + 1

   Exit Sub

AddArrayElementErr:
   LogErrors Errors
   gstrSource = mstrModuleName &
"AddArrayElement"
   On Error GoTo 0
   Err.Raise vbObjectError +
errAddArrayElementFailed, _
          gstrSource, _

LoadResString(errAddArrayElementFailed)

End Sub


Public Function CheckForNullField(rstRecords
As Recordset, strFieldName As String) As
String

   ' Returns an empty string if a given field is
null
   On Error GoTo CheckForNullFieldErr

   If IsNull(rstRecords.Fields(strFieldName))
Then
      CheckForNullField = gstrEmptyString
   Else
      CheckForNullField =
rstRecords.Fields(strFieldName)
   End If
   Exit Function

CheckForNullFieldErr:
   Call LogErrors(Errors)
   On Error GoTo 0
   Err.Raise vbObjectError +
errCheckForNullFieldFailed, _
          mstrModuleName &
"CheckForNullField", _

LoadResString(errCheckForNullFieldFailed)

End Function

Public Function ErrorOnNullField(rstRecords As
Recordset, strFieldName As String) As Variant

   ' If a given field is null, raises an error
   ' Else, returns the field value in a variant
   ' The calling function must convert the return
value to the
   ' appropriate type
   On Error GoTo ErrorOnNullFieldErr
   gstrSource = mstrModuleName &
"ErrorOnNullField"

   If IsNull(rstRecords.Fields(strFieldName)) Then
      On Error GoTo 0
      Err.Raise vbObjectError +
errMandatoryFieldNull, _
          gstrSource, _
          strFieldName & mintDelimiter &
LoadResString(errMandatoryFieldNull)
   Else
      ErrorOnNullField =
rstRecords.Fields(strFieldName)
   End If
   Exit Function

ErrorOnNullFieldErr:
   Call LogErrors(Errors)
   On Error GoTo 0
   Err.Raise vbObjectError +
errUnableToCheckNull, _
       gstrSource, _
       strFieldName & mintDelimiter &
LoadResString(errUnableToCheckNull)

End Function
Public Function StringEmpty(strCheckString As
String) As Boolean

   StringEmpty = (strCheckString =
gstrEmptyString)

End Function
Public Function GetIteratorValue(cStepIterators As
cRunColIt, _
       ByVal strItName As String)

   Dim lngIndex As Long
   Dim strValue As String

   On Error GoTo GetIteratorValueErr
   gstrSource = mstrModuleName &
"GetIteratorValue"

   ' Find the iterator in the Iterators collection
   For lngIndex = 0 To cStepIterators.Count - 1
      If cStepIterators(lngIndex).IteratorName =
strItName Then
          strValue = cStepIterators(lngIndex).Value
          Exit For
      End If
   Next lngIndex

   If lngIndex > cStepIterators.Count - 1 Then
      ' The iterator has not been defined for the
branch
      ' Raise an error
      On Error GoTo 0
      Err.Raise vbObjectError +
errParamNameInvalid, _
             gstrSource, _
             LoadResString(errParamNameInvalid)
   End If

   GetIteratorValue = strValue
   Exit Function
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
GetIteratorValueErr:
   ' Log the error code raised by Visual
Basic
   Call LogErrors(Errors)
   gstrSource = mstrModuleName &
"GetIteratorValue"
   On Error GoTo 0
   Err.Raise vbObjectError +
errGetParamValueFailed, _
         gstrSource, _

LoadResString(errGetParamValueFailed)

End Function
Public Function SubstituteParameters(ByVal
strComString As String, _
      ByVal lngWorkspaceId As Long, _
      Optional StepIterators As cRunColIt =
Nothing, _
      Optional WspParameters As
cArrParameters = Nothing) As String
   ' This function substitutes all parameter
names and
   ' environment variables in the passed in
string with
   ' their values. It also substitutes the value
for the
   ' iterators, if any.
   ' Since the syntax is to enclose parameter
names and
   ' environment variables in "%", we check
if a given
   ' variable is a parameter - if so, we
substitute the
   ' parameter value - else we try to get the
value from
   ' the environment

   Dim intPos As Integer
   Dim intEndPos As Integer
   Dim strEnvVariable As String
   Dim strValue As String
   Dim strCommand As String
   Dim cTempStr As cStringSM

   ' Initialize the return value of the function
to the
   ' passed in command
   strCommand = strComString

   If WspParameters Is Nothing Then Set
WspParameters = gcParameters

   Set cTempStr = New cStringSM

   intPos = InStr(strCommand,
gstrEnvVarSeparator)
   Do While intPos <> 0
      If Mid(strCommand, intPos + 1, 1) =
gstrEnvVarSeparator Then
         ' Wildcard character - to be
substituted by a single % - later!
         intPos = intPos + 2
         If intPos > Len(strCommand) Then
Exit Do
      Else
         ' Extract the environment variable
from the passed
         ' in string
         intEndPos = InStr(intPos + 1,
strCommand, gstrEnvVarSeparator)

         If intEndPos > 0 Then
            strEnvVariable = Mid(strCommand,
intPos + 1, intEndPos - intPos - 1)
         Else
            On Error GoTo 0
            Err.Raise vbObjectError +
errParamSeparatorMissing, _
                  gstrSource, _

LoadResString(errParamSeparatorMissing)
         End If
         strValue = gstrEmptyString

         ' Get the value of the variable and call a
function
         ' to replace the variable with it's value
         strValue = GetValue(strEnvVariable,
lngWorkspaceId, StepIterators,
WspParameters)
         ' The function raises an error if the
variable is
         ' not found
         strCommand =
cTempStr.ReplaceSubString(strCommand, _
               gstrEnvVarSeparator &
strEnvVariable & gstrEnvVarSeparator, _
               strValue)
      End If

      intPos = InStr(intPos, strCommand,
gstrEnvVarSeparator)
   Loop

   strCommand =
cTempStr.ReplaceSubString(strCommand, _
         gstrEnvVarSeparator &
gstrEnvVarSeparator, gstrEnvVarSeparator)

   Set cTempStr = Nothing
   SubstituteParameters = strCommand

End Function
Private Function GetValue(ByVal strParameter
As String, _
      ByVal lngWorkspaceId As Long, _
      cStepIterators As cRunColIt, _
      WspParameters As cArrParameters) As
String
   ' This function returns the value for the
passed in
   ' parameter - it may be a workspace
parameter, an
   ' environment variable or an iterator

   Dim intPos As Integer
   Dim intEndPos As Integer
   Dim strVariable As String
   Dim strValue As String
   Dim cParamRec As cParameter

   On Error GoTo GetValueErr

   ' Initialize the return value of the function to
the
   ' empty
   strValue = gstrEmptyString

   intPos = InStr(strParameter,
gstrEnvVarSeparator)
   If intPos > 0 Then
      ' Extract the variable from the passed in
string
      intEndPos = InStr(intPos + 1,
strParameter, gstrEnvVarSeparator)
      If intEndPos = 0 Then
         intEndPos = Len(strParameter)
      End If

      strVariable = Mid(strParameter, intPos + 1,
intEndPos - intPos - 1)
   Else
      ' The separator charactor has not been passed
in -
      ' try to find the value of the passed in
parameter
      strVariable = strParameter
   End If

   If Not StringEmpty(strVariable) Then
      ' Check if this is the timestamp parameter first
      If strVariable = gstrTimeStamp Then
         strValue = Format$(Now, mstrFormatString,
_
               vbUseSystemDayOfWeek,
vbUseSystem)
      Else
         ' Try to find a parameter for the workspace
with
         ' the same name
         Set cParamRec =
WspParameters.GetParameterValue(lngWorkspaceI
d, _
               strVariable)
         If cParamRec Is Nothing Then
            If Not cStepIterators Is Nothing Then
               ' If the string is not a parameter, then
check
               ' if it is an iterator
               strValue =
GetIteratorValue(cStepIterators, strVariable)
            End If

            If StringEmpty(strValue) Then
               ' Neither - Check if it is an environment
variable
               strValue = Environ$(strVariable)
               If StringEmpty(strValue) Then
                  On Error GoTo 0
                  WriteError errSubValuesFailed, _
                        OptArgs:="Invalid parameter: "
& gstrSQ & strVariable & gstrSQ
                  Err.Raise vbObjectError +
errSubValuesFailed, _
                        mstrModuleName &
"GetValue", _

LoadResString(errSubValuesFailed) & "Invalid
parameter: " & gstrSQ & strVariable & gstrSQ
               End If
            End If
         Else
            strValue = cParamRec.ParameterValue
         End If
      End If
   End If

   GetValue = strValue

   Exit Function

GetValueErr:
   If Err.Number = vbObjectError +
errParamNameInvalid Then
      ' If the parameter has not been defined for the
      ' workspace then check if it is an environment
      ' variable
      Resume Next
   End If

   ' Log the error code raised by Visual Basic
   Call LogErrors(Errors)
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   gstrSource = mstrModuleName &
"GetValue"
   WriteError errSubValuesFailed,
gstrSource, "Parameter: " & gstrSQ &
strVariable & gstrSQ
   On Error GoTo 0
   Err.Raise vbObjectError +
errSubValuesFailed, _
         gstrSource, _
         LoadResString(errSubValuesFailed)
& "Parameter: " & gstrSQ & strVariable &
gstrSQ

End Function
Public Function SQLFixup(strField As
String) As String
   ' Returns a string that can be executed by
SQL Server

   Dim cMyStr As New cStringSM
   Dim strTemp As String

   On Error GoTo SQLFixupErr

   strTemp = strField
   SQLFixup = strTemp

   ' Single-quotes have to be replaced by two
single-quotes,
   ' since a single-quote is the identifier
delimiter
   ' character - call a procedure to do the
replace
'    SQLFixup =
cMyStr.ReplaceSubString(strTemp, gstrDQ,
"\" & gstrDQ)

   ' Replace pipe characters with the
corresponding chr function
   ' SQLFixup =
cMyStr.ReplaceSubString(strTemp, gstrDQ,
gstrDQ & gstrDQ)

   Exit Function

SQLFixupErr:
   gstrSource = mstrModuleName &
"SQLFixup"
   LogErrors Errors
   On Error GoTo 0
   Err.Raise vbObjectError +
errMakeFieldValidFailed, _
         gstrSource,
LoadResString(errMakeFieldValidFailed)

End Function
Public Function TranslateStepLabel(sLabel
As String) As String
   ' Translates the passed in step label to a
valid file name
   ' All characters in the label that are invalid
for filenames (viz. \ / : * ? " < > |)
   ' and spaces are substituted with
underscores - also ensure that the resulting
filename
   ' is not greater than 255 characters
   Dim cTempStr As New cStringSM
   TranslateStepLabel =
cTempStr.ReplaceSubString(sLabel,
gstrFileSeparator, "_")
   TranslateStepLabel =
cTempStr.ReplaceSubString(TranslateStepL
abel, "/", gstrUnderscore)
   TranslateStepLabel =
cTempStr.ReplaceSubString(TranslateStepLab
el, ":", gstrUnderscore)
   TranslateStepLabel =
cTempStr.ReplaceSubString(TranslateStepLab
el, "*", gstrUnderscore)
   TranslateStepLabel =
cTempStr.ReplaceSubString(TranslateStepLab
el, "?", gstrUnderscore)
   TranslateStepLabel =
cTempStr.ReplaceSubString(TranslateStepLab
el, gstrDQ, gstrUnderscore)
   TranslateStepLabel =
cTempStr.ReplaceSubString(TranslateStepLab
el, "<", gstrUnderscore)
   TranslateStepLabel =
cTempStr.ReplaceSubString(TranslateStepLab
el, ">", gstrUnderscore)
   TranslateStepLabel =
cTempStr.ReplaceSubString(TranslateStepLab
el, "|", gstrUnderscore)
   TranslateStepLabel =
cTempStr.ReplaceSubString(TranslateStepLab
el, gstrBlank, gstrUnderscore)

   ' Commas are substituted with underscores
since the command shell uses a comma to
   ' delimit commands
   TranslateStepLabel =
cTempStr.ReplaceSubString(TranslateStepLab
el, ",", gstrUnderscore)

   If Len(TranslateStepLabel) > MAX_PATH
Then
      TranslateStepLabel =
Mid(TranslateStepLabel, 1, MAX_PATH)
   End If

End Function

Public Function TypeOfObject(ByVal objNode
As Object) As Integer
   ' Determines the type of object that is passed
in

   On Error GoTo TypeOfObjectErr
   gstrSource = mstrModuleName &
"TypeOfObject"

   Select Case TypeName(objNode)
      Case "cWorkspace"
         TypeOfObject = gintWorkspace

      Case "cParameter"
         TypeOfObject = gintParameter

      Case "cConnection"
         TypeOfObject = gintParameterConnect

      Case "cConnDtl"
         TypeOfObject = gintConnectionDtl

      Case "cGlobalStep"
         TypeOfObject = gintGlobalStep

      Case "cManager"
         TypeOfObject = gintManagerStep

      Case "cWorker"
         TypeOfObject = gintWorkerStep

      Case "cStep"
         ' If a step record is passed in, call a
function
         ' to determine the type of step
         TypeOfObject =
TypeOfStep(StepClass:=objNode)

      Case Else
         WriteError errTypeOfObjectFailed,
gstrSource, _
            TypeName(objNode)
         On Error GoTo 0
         Err.Raise vbObjectError +
errTypeOfObjectFailed, _
            gstrSource, _

LoadResString(errTypeOfObjectFailed)
   End Select

   Exit Function

TypeOfObjectErr:
   ' Log the error code raised by Visual Basic
   Call LogErrors(Errors)
   On Error GoTo 0
   Err.Raise vbObjectError +
errTypeOfObjectFailed, _
         gstrSource, _
         LoadResString(errTypeOfObjectFailed)

End Function
Attribute VB_Name = "ConnDtlCommon"
'  FILE:      ConnDtlCommon.bas
'            Microsoft TPC-H Kit Ver. 1.00
'            Copyright Microsoft, 1999
'            All Rights Reserved
'
'
'  PURPOSE:    Contains functionality common
across StepMaster and
'            SMRunOnly, pertaining to connections
'            Specifically, functions to load connections
in an array
'            and so on.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

' Used to indicate the source module name when
errors
' are raised by this module
Private Const mstrModuleName As String =
"ConnDtlCommon."

Public Sub LoadRSInConnDtlArray(rstConns As
Recordset, cConns As cConnDtls)

   Dim cNewConnDtl As cConnDtl

   On Error GoTo LoadRSInConnDtlArrayErr

   If rstConns.RecordCount = 0 Then
      Exit Sub
   End If

   rstConns.MoveFirst
   While Not rstConns.EOF

      Set cNewConnDtl = New cConnDtl

      ' Initialize ConnDtl values
      ' Call a procedure to raise an error if mandatory
fields are null.
      cNewConnDtl.ConnNameId =
ErrorOnNullField(rstConns,
FLD_ID_CONN_NAME)
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
      cNewConnDtl.WorkspaceId =
ErrorOnNullField(rstConns,
FLD_ID_WORKSPACE)
      cNewConnDtl.ConnName =
CStr(ErrorOnNullField(rstConns,
FLD_CONN_DTL_CONNECTION_NAM
E))
      cNewConnDtl.ConnectionString =
CheckForNullField(rstConns,
FLD_CONN_DTL_CONNECTION_STRI
NG)
      cNewConnDtl.ConnType =
CheckForNullField(rstConns,
FLD_CONN_DTL_CONNECTION_TYPE)

      cConns.Load cNewConnDtl

      Set cNewConnDtl = Nothing
      rstConns.MoveNext
   Wend

   Exit Sub

LoadRSInConnDtlArrayErr:
   LogErrors Errors
   gstrSource = mstrModuleName &
"LoadRSInConnDtlArray"
   On Error GoTo 0
   Err.Raise vbObjectError +
errLoadRsInArrayFailed, gstrSource, _

LoadResString(errLoadRsInArrayFailed)
End Sub
Attribute VB_Name =
"ConnectionCommon"
'  FILE:      ConnnectionCommon.bas
'            Microsoft TPC-H Kit Ver. 1.00
'            Copyright Microsoft, 1999
'            All Rights Reserved
'
'
'  PURPOSE:    Contains functionality
common across StepMaster and
'            SMRunOnly, pertaining to
connection strings
'            Specifically, functions to load
connections strings
'            in an array and so on.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

' Used to indicate the source module name
when errors
' are raised by this module
Private Const mstrModuleName As String =
"ConnectionCommon."

Public Sub
LoadRecordsetInConnectionArray(rstConns
As Recordset, cConns As cConnections)

   Dim cNewConnection As cConnection

   On Error GoTo
LoadRecordsetInConnectionArrayErr

   If rstConns.RecordCount = 0 Then
      Exit Sub
   End If

   rstConns.MoveFirst
   While Not rstConns.EOF
      Set cNewConnection = New cConnection

      ' Initialize Connection values
      ' Call a procedure to raise an error if
mandatory fields are null.
      cNewConnection.ConnectionId =
ErrorOnNullField(rstConns, "connection_id")
      cNewConnection.WorkspaceId =
CStr(ErrorOnNullField(rstConns,
FLD_ID_WORKSPACE))
      cNewConnection.ConnectionName =
CStr(ErrorOnNullField(rstConns,
"connection_name"))
      cNewConnection.ConnectionValue =
CheckForNullField(rstConns,
"connection_value")
      cNewConnection.Description =
CheckForNullField(rstConns, "description")

      cNewConnection.NoCountDisplay =
CheckForNullField(rstConns,
"no_count_display")
      cNewConnection.NoExecute =
CheckForNullField(rstConns, "no_execute")
      cNewConnection.ParseQueryOnly =
CheckForNullField(rstConns,
"parse_query_only")
      cNewConnection.QuotedIdentifiers =
CheckForNullField(rstConns,
"ANSI_quoted_identifiers")
      cNewConnection.AnsiNulls =
CheckForNullField(rstConns, "ANSI_nulls")
      cNewConnection.ShowQueryPlan =
CheckForNullField(rstConns,
"show_query_plan")
      cNewConnection.ShowStatsTime =
CheckForNullField(rstConns,
"show_stats_time")
      cNewConnection.ShowStatsIO =
CheckForNullField(rstConns, "show_stats_io")
      cNewConnection.ParseOdbcMsg =
CheckForNullField(rstConns,
"parse_odbc_msg_prefixes")
      cNewConnection.RowCount =
CheckForNullField(rstConns, "row_count")
      cNewConnection.TsqlBatchSeparator =
CheckForNullField(rstConns,
"tsql_batch_separator")
      cNewConnection.QueryTimeOut =
CheckForNullField(rstConns,
"query_time_out")
      cNewConnection.ServerLanguage =
CheckForNullField(rstConns,
"server_language")
      cNewConnection.CharacterTranslation =
CheckForNullField(rstConns,
"character_translation")
      cNewConnection.RegionalSettings =
CheckForNullField(rstConns,
"regional_settings")

      cConns.Load cNewConnection

      Set cNewConnection = Nothing
      rstConns.MoveNext
   Wend

   Exit Sub

LoadRecordsetInConnectionArrayErr:
   LogErrors Errors
   gstrSource = mstrModuleName &
"LoadRecordsetInConnectionArray"
   On Error GoTo 0
   Err.Raise vbObjectError +
errLoadRsInArrayFailed, gstrSource, _
         LoadResString(errLoadRsInArrayFailed)
End Sub
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cParameter"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:       cParameter.cls
'            Microsoft TPC-H Kit Ver. 1.00
'            Copyright Microsoft, 1999
'            All Rights Reserved
'
'
'  PURPOSE:     Encapsulates the properties and
methods of a parameter.
'            Contains functions to insert, update and
delete
'            workspace_parameters records from the
database.
'  Contact:     Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit
Option Base 0

' Local variable(s) to hold property value(s)
Private mlngWorkspaceId As Long
Private mlngParameterId As Long
Private mstrParameterName As String
Private mstrParameterValue As String
Private mstrDescription As String
Private mintParameterType As Integer
Private mdbsStepMaster As Database
Private mintOperation As Operation
Private mlngPosition As Long

' Used to indicate the source module name when
errors
' are raised by this class
Private mstrSource As String
Private Const mstrModuleName As String =
"cParameter."

' The cSequence class is used to generate unique
parameter identifiers
Private mParameterSeq As cSequence

' The StringSM class is used to carry out string
operations
Private mFieldValue As cStringSM

' Parameter types
Public Enum ParameterType
   gintParameterGeneric = 0
   gintParameterConnect
   gintParameterApplication
   gintParameterBuiltIn
End Enum


Private Sub AssignParameters(qyExec As
DAO.QueryDef)
   ' Assigns values to the parameters in the querydef
object
   ' The parameter names are cryptic to make them
different
   ' from the field names. When the parameter
names are
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   ' the same as the field names, parameters
in the where
   ' clause do not get created.

   Dim prmParam As DAO.Parameter

   On Error GoTo AssignParametersErr

   For Each prmParam In
qyExec.Parameters
      Select Case prmParam.Name
         Case "[w_id]"
            prmParam.Value =
mlngWorkspaceId

         Case "[p_id]"
            prmParam.Value =
mlngParameterId

         Case "[p_name]"
            prmParam.Value =
mstrParameterName

         Case "[p_value]"
            prmParam.Value =
mstrParameterValue

         Case "[desc]"
            prmParam.Value =
mstrDescription

         Case "[p_type]"
            prmParam.Value =
mintParameterType

         Case Else
            ' Write the parameter name that is
faulty
            WriteError errInvalidParameter,
mstrSource, _
                  prmParam.Name
            On Error GoTo 0
            Err.Raise errInvalidParameter,
mstrModuleName & "AssignParameters", _

LoadResString(errInvalidParameter)
      End Select
   Next prmParam

'    qyExec.Parameters("w_id").Value =
mlngWorkspaceId
'    qyExec.Parameters("p_id").Value =
mlngParameterId
'    qyExec.Parameters("p_name").Value =
mstrParameterName
'    qyExec.Parameters("p_value").Value =
mstrParameterValue
'

   Exit Sub

AssignParametersErr:

   mstrSource = mstrModuleName &
"AssignParameters"
   Call LogErrors(Errors)
   On Error GoTo 0
   Err.Raise vbObjectError +
errAssignParametersFailed, _
      mstrSource,
LoadResString(errAssignParametersFailed)

End Sub
Public Property Let Position(ByVal RHS As
Long)

   mlngPosition = RHS

End Property

Public Property Get Position() As Long

   Position = mlngPosition

End Property

Public Function Clone() As cParameter

   ' Creates a copy of a given parameter

   Dim cCloneParam As cParameter

   On Error GoTo CloneErr
   mstrSource = mstrModuleName & "Clone"

   Set cCloneParam = New cParameter

   ' Copy all the parameter properties to the
newly
   ' created parameter
   Set cCloneParam.NodeDB =
mdbsStepMaster
   cCloneParam.WorkspaceId =
mlngWorkspaceId
   cCloneParam.ParameterId =
mlngParameterId
   cCloneParam.ParameterName =
mstrParameterName
   cCloneParam.ParameterValue =
mstrParameterValue
   cCloneParam.Description = mstrDescription
   cCloneParam.ParameterType =
mintParameterType
   cCloneParam.IndOperation = mintOperation
   cCloneParam.Position = mlngPosition

   ' And set the return value to the newly
created parameter
   Set Clone = cCloneParam
   Set cCloneParam = Nothing

   Exit Function

CloneErr:
   LogErrors Errors
   mstrSource = mstrModuleName & "Clone"
   On Error GoTo 0
   Err.Raise vbObjectError + errCloneFailed, _
         mstrSource,
LoadResString(errCloneFailed)

End Function
Public Property Set NodeDB(vdata As
Database)

   Set mdbsStepMaster = vdata

End Property
Public Property Get NodeDB() As Database

   Set NodeDB = mdbsStepMaster

End Property

Private Sub CheckDupParameterName()
   ' Check if the parameter name already exists
in the workspace
   Dim rstParameter As Recordset
   Dim strSql As String
   Dim qy As DAO.QueryDef

   On Error GoTo CheckDupParameterNameErr
   mstrSource = mstrModuleName &
"CheckDupParameterName"

   ' Create a recordset object to retrieve the count of
all parameters
   ' for the workspace with the same name
   strSql = "Select count(*) as parameter_count " &
_
      " from workspace_parameters " & _
      " where workspace_id = [w_id]" & _
      " and parameter_name = [p_name]" & _
      " and parameter_id <> [p_id]"

   Set qy =
mdbsStepMaster.CreateQueryDef(gstrEmptyString,
strSql)
   Call AssignParameters(qy)

   Set rstParameter =
qy.OpenRecordset(dbOpenForwardOnly)

   If rstParameter![parameter_count] > 0 Then
      rstParameter.Close
      qy.Close
      ShowError errDuplicateParameterName
      On Error GoTo 0
      Err.Raise vbObjectError +
errDuplicateParameterName, _
         mstrSource,
LoadResString(errDuplicateParameterName)
   End If

   rstParameter.Close
   qy.Close

   Exit Sub

CheckDupParameterNameErr:
   LogErrors Errors
   mstrSource = mstrModuleName &
"CheckDupParameterName"
   On Error GoTo 0
   Err.Raise vbObjectError +
errCheckDupParameterNameFailed, _
      mstrSource,
LoadResString(errCheckDupParameterNameFailed
)

End Sub
Private Sub CheckDB()
   ' Check if the database object has been initialized

   If mdbsStepMaster Is Nothing Then
      On Error GoTo 0
      Err.Raise vbObjectError + errInvalidDB, _
         mstrModuleName & "CheckDB",
LoadResString(errInvalidDB)
   End If

End Sub
Public Property Let ParameterValue(vdata As
String)

   mstrParameterValue = vdata

End Property
Public Property Let Description(vdata As String)

   mstrDescription = vdata
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
End Property
Public Property Let ParameterType(vdata
As ParameterType)

   mintParameterType = vdata

End Property

Public Property Let ParameterName(vdata
As String)

   If vdata = gstrEmptyString Then

      ShowError
errParameterNameMandatory
      On Error GoTo 0
      ' Propogate this error back to the caller
      Err.Raise vbObjectError +
errParameterNameMandatory, _
         mstrSource,
LoadResString(errParameterNameMandator
y)
   Else
      mstrParameterName = vdata
   End If

End Property

Public Property Let ParameterId(vdata As
Long)
   mlngParameterId = vdata
End Property

Public Property Let IndOperation(ByVal
vdata As Operation)

   ' The valid operations are define in the
cOperations
   ' class. Check if the operation is valid
   Select Case vdata
      Case QueryOp, InsertOp, UpdateOp,
DeleteOp
         mintOperation = vdata

      Case Else
         On Error GoTo 0
         Err.Raise vbObjectError +
errInvalidOperation, _
               mstrSource,
LoadResString(errInvalidOperation)
   End Select

End Property
Public Sub Validate()
   ' Each distinct object will have a Validate
method which
   ' will check if the class properties are
valid. This method
   ' will be used to check interdependant
properties that
   ' cannot be validated by the let
procedures.
   ' It should be called by the add and
modify methods of the class

   On Error GoTo ValidateErr

   ' Check if the db object is valid
   Call CheckDB

   ' Call procedure to raise an error if the
parameter name
   ' already exists in the workspace -
   ' if there are duplicates, we don't know
what value for the
   ' parameter to use at runtime
   Call CheckDupParameterName

   Exit Sub

ValidateErr:

   mstrSource = mstrModuleName &
"Validate"
   Call LogErrors(Errors)
   On Error GoTo 0
   Err.Raise vbObjectError +
errValidateFailed, _
      mstrSource,
LoadResString(errValidateFailed)

End Sub
Public Property Let WorkspaceId(vdata As
Long)

   mlngWorkspaceId = vdata

End Property

Public Sub Add()

   Dim strInsert As String
   Dim qy As DAO.QueryDef

   On Error GoTo AddErr

   ' Validate the record before trying to insert
the record
   Call Validate

   ' Create a temporary querydef object
   strInsert = "insert into
workspace_parameters " & _
         "( workspace_id, parameter_id, " & _
         " parameter_name, parameter_value, "
& _
         " description, parameter_type ) " & _
         " values ( [w_id], [p_id], [p_name],
[p_value], [desc], [p_type] ) "
   Set qy =
mdbsStepMaster.CreateQueryDef(gstrEmptySt
ring, strInsert)

   ' Call a procedure to assign the parameter
values
   Call AssignParameters(qy)

   qy.Execute dbFailOnError
   qy.Close

'   strInsert = "insert into
workspace_parameters " & _
'         "( workspace_id, parameter_id, " & _
'         " parameter_name, parameter_value ) "
& _
'         " values ( " & _
'         Str(mlngWorkspaceId) & ", " &
Str(mlngParameterId) & _
'         ", " &
mFieldValue.MakeStringFieldValid(mstrPara
meterName) & _
'         ", " &
mFieldValue.MakeStringFieldValid(mstrPara
meterValue) & " ) "
'   mdbsStepMaster.Execute strInsert,
dbFailOnError + dbSQLPassThrough

   Exit Sub

AddErr:

   mstrSource = mstrModuleName & "Add"
   Call LogErrors(Errors)
   On Error GoTo 0
   Err.Raise vbObjectError +
errParameterInsertFailed, _
      mstrSource,
LoadResString(errParameterInsertFailed)

End Sub
Public Sub Delete()

   Dim strDelete As String
   Dim qy As DAO.QueryDef

   On Error GoTo DeleteErr

   ' Check if the db object is valid
   Call CheckDB

   strDelete = "delete from workspace_parameters "
& _
         " where parameter_id = [p_id]"
   Set qy =
mdbsStepMaster.CreateQueryDef(gstrEmptyString,
strDelete)

   Call AssignParameters(qy)
   qy.Execute dbFailOnError

   qy.Close

   Exit Sub

DeleteErr:
   LogErrors Errors
   mstrSource = mstrModuleName & "Delete"
   On Error GoTo 0
   Err.Raise vbObjectError +
errDeleteParameterFailed, _
         mstrSource, _
         LoadResString(errDeleteParameterFailed)

End Sub

Public Sub Modify()

   Dim strUpdate As String
   Dim qy As QueryDef

   On Error GoTo ModifyErr

   ' Validate the updated values before trying to
modify the db
   Call Validate

   ' Create a temporary querydef object with the
modify string
   strUpdate = "update workspace_parameters " & _
         " set workspace_id = [w_id], " & _
         "parameter_name = [p_name], " & _
         "parameter_value = [p_value], " & _
         "description = [desc], " & _
         "parameter_type = [p_type] " & _
         " where parameter_id = [p_id]"
   Set qy =
mdbsStepMaster.CreateQueryDef(gstrEmptyString,
strUpdate)

   ' Call a procedure to assign the parameter values
to the
   ' querydef object
   Call AssignParameters(qy)
   qy.Execute dbFailOnError
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   qy.Close

'   mdbsStepMaster.Execute strUpdate,
dbFailOnError
'
   Exit Sub

ModifyErr:

   mstrSource = mstrModuleName &
"Modify"
   Call LogErrors(Errors)
   On Error GoTo 0
   Err.Raise vbObjectError +
errParameterUpdateFailed, _
       mstrSource,
LoadResString(errParameterUpdateFailed)

End Sub
Public Property Get ParameterName() As
String

   ParameterName = mstrParameterName

End Property

Public Property Get ParameterId() As Long

   ParameterId = mlngParameterId

End Property
Public Property Get NextIdentifier() As
Long

   Dim lngNextId As Long

   On Error GoTo NextIdentifierErr

   ' First check if the database object is valid
   Call CheckDB

   ' Retrieve the next identifier using the
sequence class
   Set mParameterSeq = New cSequence
   Set mParameterSeq.IdDatabase =
mdbsStepMaster
   mParameterSeq.IdentifierColumn =
FLD_ID_PARAMETER
   lngNextId = mParameterSeq.Identifier
   Set mParameterSeq = Nothing

   NextIdentifier = lngNextId
   Exit Property

NextIdentifierErr:
   LogErrors Errors
   mstrSource = mstrModuleName &
"NextIdentifier"
   On Error GoTo 0
   Err.Raise vbObjectError + errIdGetFailed,
_
       mstrSource,
LoadResString(errIdGetFailed)

End Property
Public Property Get IndOperation() As
Operation

   IndOperation = mintOperation

End Property

Public Property Get WorkspaceId() As Long

   WorkspaceId = mlngWorkspaceId

End Property

Public Property Get ParameterValue() As
String

   ParameterValue = mstrParameterValue

End Property
Public Property Get Description() As String

   Description = mstrDescription

End Property
Public Property Get ParameterType() As
ParameterType

   ParameterType = mintParameterType

End Property


Private Sub Class_Initialize()

   Set mFieldValue = New cStringSM

   ' Initialize the operation indicator variable to
Query
   ' It will be modified later by the collection
class when
   ' inserts, updates or deletes are performed
   mintOperation = QueryOp

End Sub

Private Sub Class_Terminate()

   Set mdbsStepMaster = Nothing
   Set mFieldValue = Nothing

End Sub
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cRunColIt"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:       cRunColIt.cls
'              Microsoft TPC-H Kit Ver. 1.00
'              Copyright Microsoft, 1999
'              All Rights Reserved
'
'
'  PURPOSE:     This module implements a
stack of Iterator nodes.
'              Ensures that only cRunItNode
objects are stored in the stack.
'  Contact:     Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

' Used to indicate the source module name
when errors
' are raised by this class
Private Const mstrModuleName As String =
"cRunColIt."
Private mstrSource As String

Private mcIterators As cStack
Public Sub Clear()

   mcIterators.Clear

End Sub

Private Sub Class_Initialize()

   Set mcIterators = New cStack

End Sub

Private Sub Class_Terminate()

   Set mcIterators = Nothing

End Sub


Public Function Value(strItName As String) As
String

   Dim lngIndex As Long

   For lngIndex = 0 To mcIterators.Count - 1
       If mcIterators(lngIndex).IteratorName =
strItName Then
           Value = mcIterators(lngIndex).Value
           Exit For
       End If
   Next lngIndex

End Function

Public Property Get Item(ByVal Position As Long)
As cRunItNode
Attribute Item.VB_UserMemId = 0

   Set Item = mcIterators(Position)

End Property


Public Function Count() As Long

   Count = mcIterators.Count

End Function

Public Function Pop() As cRunItNode

   Set Pop = mcIterators.Pop

End Function

Public Sub Push(objToPush As cRunItNode)

   Call mcIterators.Push(objToPush)

End Sub


VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
  Persistable = 0  'NotPersistable
  DataBindingBehavior = 0  'vbNone
  DataSourceBehavior  = 0  'vbNone
  MTSTransactionMode  = 0  'NotAnMTSObject
END
Attribute VB_Name = "cRunInst"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:       cRunColIt.cls
'              Microsoft TPC-H Kit Ver. 1.00
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
'           Copyright Microsoft, 1999
'           All Rights Reserved
'
'
'  PURPOSE:    This module controls the
run processing. It runs a branch
'           at a time and raises events when
each step completes execution.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

' Used to indicate the source module name
when errors
' are raised by this class
Private Const mstrModuleName As String =
"cRunInst."
Private mstrSource As String

' Local variable(s) to hold property value(s)
Private mstrRootKey As String
Public WspId As Long
Private mcParameters As cArrParameters
Private mcRunSteps As cArrSteps
Private mcRunConstraints As
cArrConstraints
Public RunConnections As cConnections
Public RunConnDtls As cConnDtls
Private mcvntWspPreCons As Variant
Private mcvntWspPostCons As Variant
Private mcNavSteps As cStepTree

Private mcInstances As cInstances
Private mcFreeSteps As cVectorLng
Private mcFailures As cFailedSteps
Private mblnAsk As Boolean   ' Set to True
when the a step with continuation
criteria=Ask fails
Private mblnAbort As Boolean ' Set to True
when the run is aborted
Private msAbortDtls As String
Private mbarrFree() As Byte
Private WithEvents mcTermSteps As
cTermSteps
Attribute mcTermSteps.VB_VarHelpID = -1
Public RunId As Long
Public CreateInputFiles As Boolean
Private Enum WspLogEvents
   mintRunStart
   mintRunComplete
   mintStepStart
   mintStepComplete
End Enum

Private mcWspLog As cFileSM
Private mstrCurBranchRoot As String
Private mcDummyRootInstance As
cInstance
' Key for the dummy root instance - Should
be a key that is invalid for an actual step
record
Private Const mstrDummyRootKey As
String = "D"
' Public events to notify the calling function
of the
' start and end time for each step
Public Event RunStart(dtmStartTime As
Currency, strWspLog As String)
Public Event RunComplete(dtmEndTime As
Currency)
Public Event StepStart(cStepRecord As
cStep, dtmStartTime As Currency, _
      lngInstanceId As Long, lParentInstanceId
As Long, sPath As String, _
      sIts As String, sItValue As String)
Public Event StepComplete(cStepRecord As
cStep, dtmEndTime As Currency,
lngInstanceId As Long, lElapsed As Long)
Public Event ProcessStart(cStepRecord As
cStep, strCommand As String, _
      dtmStartTime As Currency, lngInstanceId
As Long, lParentInstanceId As Long, _
      sItValue As String)
Public Event ProcessComplete(cStepRecord
As cStep, dtmEndTime As Currency,
lngInstanceId As Long, lElapsed As Long)
' The class that will execute each step - we trap
the events
' that are raised by it when a step
starts/completes
' execution
Private WithEvents cExecStep1 As cRunStep
Attribute cExecStep1.VB_VarHelpID = -1
Private WithEvents cExecStep2 As cRunStep
Attribute cExecStep2.VB_VarHelpID = -1
Private WithEvents cExecStep3 As cRunStep
Attribute cExecStep3.VB_VarHelpID = -1
Private WithEvents cExecStep4 As cRunStep
Attribute cExecStep4.VB_VarHelpID = -1
Private WithEvents cExecStep5 As cRunStep
Attribute cExecStep5.VB_VarHelpID = -1
Private WithEvents cExecStep6 As cRunStep
Attribute cExecStep6.VB_VarHelpID = -1
Private WithEvents cExecStep7 As cRunStep
Attribute cExecStep7.VB_VarHelpID = -1
Private WithEvents cExecStep8 As cRunStep
Attribute cExecStep8.VB_VarHelpID = -1
Private WithEvents cExecStep9 As cRunStep
Attribute cExecStep9.VB_VarHelpID = -1

Private WithEvents cExecStep10 As cRunStep
Attribute cExecStep10.VB_VarHelpID = -1
Private WithEvents cExecStep11 As cRunStep
Attribute cExecStep11.VB_VarHelpID = -1
Private WithEvents cExecStep12 As cRunStep
Attribute cExecStep12.VB_VarHelpID = -1
Private WithEvents cExecStep13 As cRunStep
Attribute cExecStep13.VB_VarHelpID = -1
Private WithEvents cExecStep14 As cRunStep
Attribute cExecStep14.VB_VarHelpID = -1
Private WithEvents cExecStep15 As cRunStep
Attribute cExecStep15.VB_VarHelpID = -1
Private WithEvents cExecStep16 As cRunStep
Attribute cExecStep16.VB_VarHelpID = -1
Private WithEvents cExecStep17 As cRunStep
Attribute cExecStep17.VB_VarHelpID = -1
Private WithEvents cExecStep18 As cRunStep
Attribute cExecStep18.VB_VarHelpID = -1
Private WithEvents cExecStep19 As cRunStep
Attribute cExecStep19.VB_VarHelpID = -1

Private WithEvents cExecStep20 As cRunStep
Attribute cExecStep20.VB_VarHelpID = -1
Private WithEvents cExecStep21 As cRunStep
Attribute cExecStep21.VB_VarHelpID = -1
Private WithEvents cExecStep22 As cRunStep
Attribute cExecStep22.VB_VarHelpID = -1
Private WithEvents cExecStep23 As cRunStep
Attribute cExecStep23.VB_VarHelpID = -1
Private WithEvents cExecStep24 As cRunStep
Attribute cExecStep24.VB_VarHelpID = -1
Private WithEvents cExecStep25 As cRunStep
Attribute cExecStep25.VB_VarHelpID = -1
Private WithEvents cExecStep26 As cRunStep
Attribute cExecStep26.VB_VarHelpID = -1
Private WithEvents cExecStep27 As cRunStep
Attribute cExecStep27.VB_VarHelpID = -1
Private WithEvents cExecStep28 As cRunStep
Attribute cExecStep28.VB_VarHelpID = -1
Private WithEvents cExecStep29 As cRunStep
Attribute cExecStep29.VB_VarHelpID = -1

Private WithEvents cExecStep30 As cRunStep
Attribute cExecStep30.VB_VarHelpID = -1
Private WithEvents cExecStep31 As cRunStep
Attribute cExecStep31.VB_VarHelpID = -1
Private WithEvents cExecStep32 As cRunStep
Attribute cExecStep32.VB_VarHelpID = -1
Private WithEvents cExecStep33 As cRunStep
Attribute cExecStep33.VB_VarHelpID = -1
Private WithEvents cExecStep34 As cRunStep
Attribute cExecStep34.VB_VarHelpID = -1
Private WithEvents cExecStep35 As cRunStep
Attribute cExecStep35.VB_VarHelpID = -1
Private WithEvents cExecStep36 As cRunStep
Attribute cExecStep36.VB_VarHelpID = -1
Private WithEvents cExecStep37 As cRunStep
Attribute cExecStep37.VB_VarHelpID = -1
Private WithEvents cExecStep38 As cRunStep
Attribute cExecStep38.VB_VarHelpID = -1
Private WithEvents cExecStep39 As cRunStep
Attribute cExecStep39.VB_VarHelpID = -1

Private WithEvents cExecStep40 As cRunStep
Attribute cExecStep40.VB_VarHelpID = -1
Private WithEvents cExecStep41 As cRunStep
Attribute cExecStep41.VB_VarHelpID = -1
Private WithEvents cExecStep42 As cRunStep
Attribute cExecStep42.VB_VarHelpID = -1
Private WithEvents cExecStep43 As cRunStep
Attribute cExecStep43.VB_VarHelpID = -1
Private WithEvents cExecStep44 As cRunStep
Attribute cExecStep44.VB_VarHelpID = -1
Private WithEvents cExecStep45 As cRunStep
Attribute cExecStep45.VB_VarHelpID = -1
Private WithEvents cExecStep46 As cRunStep
Attribute cExecStep46.VB_VarHelpID = -1
Private WithEvents cExecStep47 As cRunStep
Attribute cExecStep47.VB_VarHelpID = -1
Private WithEvents cExecStep48 As cRunStep
Attribute cExecStep48.VB_VarHelpID = -1
Private WithEvents cExecStep49 As cRunStep
Attribute cExecStep49.VB_VarHelpID = -1

Private WithEvents cExecStep50 As cRunStep
Attribute cExecStep50.VB_VarHelpID = -1
Private WithEvents cExecStep51 As cRunStep
Attribute cExecStep51.VB_VarHelpID = -1
Private WithEvents cExecStep52 As cRunStep
Attribute cExecStep52.VB_VarHelpID = -1
Private WithEvents cExecStep53 As cRunStep
Attribute cExecStep53.VB_VarHelpID = -1
Private WithEvents cExecStep54 As cRunStep
Attribute cExecStep54.VB_VarHelpID = -1
Private WithEvents cExecStep55 As cRunStep
Attribute cExecStep55.VB_VarHelpID = -1
Private WithEvents cExecStep56 As cRunStep
Attribute cExecStep56.VB_VarHelpID = -1
Private WithEvents cExecStep57 As cRunStep
Attribute cExecStep57.VB_VarHelpID = -1
Private WithEvents cExecStep58 As cRunStep
Attribute cExecStep58.VB_VarHelpID = -1
Private WithEvents cExecStep59 As cRunStep
Attribute cExecStep59.VB_VarHelpID = -1

Private WithEvents cExecStep60 As cRunStep
Attribute cExecStep60.VB_VarHelpID = -1
Private WithEvents cExecStep61 As cRunStep
Attribute cExecStep61.VB_VarHelpID = -1
Private WithEvents cExecStep62 As cRunStep
Attribute cExecStep62.VB_VarHelpID = -1
Private WithEvents cExecStep63 As cRunStep
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
Attribute cExecStep63.VB_VarHelpID = -1
Private WithEvents cExecStep64 As
cRunStep
Attribute cExecStep64.VB_VarHelpID = -1
Private WithEvents cExecStep65 As
cRunStep
Attribute cExecStep65.VB_VarHelpID = -1
Private WithEvents cExecStep66 As
cRunStep
Attribute cExecStep66.VB_VarHelpID = -1
Private WithEvents cExecStep67 As
cRunStep
Attribute cExecStep67.VB_VarHelpID = -1
Private WithEvents cExecStep68 As
cRunStep
Attribute cExecStep68.VB_VarHelpID = -1
Private WithEvents cExecStep69 As
cRunStep
Attribute cExecStep69.VB_VarHelpID = -1

Private WithEvents cExecStep70 As
cRunStep
Attribute cExecStep70.VB_VarHelpID = -1
Private WithEvents cExecStep71 As
cRunStep
Attribute cExecStep71.VB_VarHelpID = -1
Private WithEvents cExecStep72 As
cRunStep
Attribute cExecStep72.VB_VarHelpID = -1
Private WithEvents cExecStep73 As
cRunStep
Attribute cExecStep73.VB_VarHelpID = -1
Private WithEvents cExecStep74 As
cRunStep
Attribute cExecStep74.VB_VarHelpID = -1
Private WithEvents cExecStep75 As
cRunStep
Attribute cExecStep75.VB_VarHelpID = -1
Private WithEvents cExecStep76 As
cRunStep
Attribute cExecStep76.VB_VarHelpID = -1
Private WithEvents cExecStep77 As
cRunStep
Attribute cExecStep77.VB_VarHelpID = -1
Private WithEvents cExecStep78 As
cRunStep
Attribute cExecStep78.VB_VarHelpID = -1
Private WithEvents cExecStep79 As
cRunStep
Attribute cExecStep79.VB_VarHelpID = -1

Private WithEvents cExecStep80 As
cRunStep
Attribute cExecStep80.VB_VarHelpID = -1
Private WithEvents cExecStep81 As
cRunStep
Attribute cExecStep81.VB_VarHelpID = -1
Private WithEvents cExecStep82 As
cRunStep
Attribute cExecStep82.VB_VarHelpID = -1
Private WithEvents cExecStep83 As
cRunStep
Attribute cExecStep83.VB_VarHelpID = -1
Private WithEvents cExecStep84 As
cRunStep
Attribute cExecStep84.VB_VarHelpID = -1
Private WithEvents cExecStep85 As
cRunStep
Attribute cExecStep85.VB_VarHelpID = -1
Private WithEvents cExecStep86 As
cRunStep
Attribute cExecStep86.VB_VarHelpID = -1
Private WithEvents cExecStep87 As
cRunStep
Attribute cExecStep87.VB_VarHelpID = -1
Private WithEvents cExecStep88 As cRunStep
Attribute cExecStep88.VB_VarHelpID = -1
Private WithEvents cExecStep89 As cRunStep
Attribute cExecStep89.VB_VarHelpID = -1

Private WithEvents cExecStep90 As cRunStep
Attribute cExecStep90.VB_VarHelpID = -1
Private WithEvents cExecStep91 As cRunStep
Attribute cExecStep91.VB_VarHelpID = -1
Private WithEvents cExecStep92 As cRunStep
Attribute cExecStep92.VB_VarHelpID = -1
Private WithEvents cExecStep93 As cRunStep
Attribute cExecStep93.VB_VarHelpID = -1
Private WithEvents cExecStep94 As cRunStep
Attribute cExecStep94.VB_VarHelpID = -1
Private WithEvents cExecStep95 As cRunStep
Attribute cExecStep95.VB_VarHelpID = -1
Private WithEvents cExecStep96 As cRunStep
Attribute cExecStep96.VB_VarHelpID = -1
Private WithEvents cExecStep97 As cRunStep
Attribute cExecStep97.VB_VarHelpID = -1
Private WithEvents cExecStep98 As cRunStep
Attribute cExecStep98.VB_VarHelpID = -1
Private WithEvents cExecStep99 As cRunStep
Attribute cExecStep99.VB_VarHelpID = -1

Private Const msIt As String = " Iterator: "
Private Const msItValue As String = " Value: "
Public Sub Abort()

   On Error GoTo AbortErr

   ' Make sure that we don't execute any more
steps
   Call StopRun

   If cExecStep1 Is Nothing And cExecStep2 Is
Nothing And cExecStep3 Is Nothing And
cExecStep4 Is Nothing And cExecStep5 Is
Nothing And cExecStep6 Is Nothing And
cExecStep7 Is Nothing And cExecStep8 Is
Nothing And cExecStep9 Is Nothing And _
      cExecStep10 Is Nothing And cExecStep11
Is Nothing And cExecStep12 Is Nothing And
cExecStep13 Is Nothing And cExecStep14 Is
Nothing And cExecStep15 Is Nothing And
cExecStep16 Is Nothing And cExecStep17 Is
Nothing And cExecStep18 Is Nothing And
cExecStep19 Is Nothing And _
      cExecStep20 Is Nothing And cExecStep21
Is Nothing And cExecStep22 Is Nothing And
cExecStep23 Is Nothing And cExecStep24 Is
Nothing And cExecStep25 Is Nothing And
cExecStep26 Is Nothing And cExecStep27 Is
Nothing And cExecStep28 Is Nothing And
cExecStep29 Is Nothing And _
      cExecStep30 Is Nothing And cExecStep31
Is Nothing And cExecStep32 Is Nothing And
cExecStep33 Is Nothing And cExecStep34 Is
Nothing And cExecStep35 Is Nothing And
cExecStep36 Is Nothing And cExecStep37 Is
Nothing And cExecStep38 Is Nothing And
cExecStep39 Is Nothing And _
      cExecStep40 Is Nothing And cExecStep41
Is Nothing And cExecStep42 Is Nothing And
cExecStep43 Is Nothing And cExecStep44 Is
Nothing And cExecStep45 Is Nothing And
cExecStep46 Is Nothing And cExecStep47 Is
Nothing And cExecStep48 Is Nothing And
cExecStep49 Is Nothing And _
      cExecStep50 Is Nothing And cExecStep51
Is Nothing And cExecStep52 Is Nothing And
cExecStep53 Is Nothing And cExecStep54 Is
Nothing And cExecStep55 Is Nothing And
cExecStep56 Is Nothing And cExecStep57 Is
Nothing And cExecStep58 Is Nothing And
cExecStep59 Is Nothing And _
      cExecStep60 Is Nothing And cExecStep61 Is
Nothing And cExecStep62 Is Nothing And
cExecStep63 Is Nothing And cExecStep64 Is
Nothing And cExecStep65 Is Nothing And
cExecStep66 Is Nothing And cExecStep67 Is
Nothing And cExecStep68 Is Nothing And
cExecStep69 Is Nothing And _
      cExecStep70 Is Nothing And cExecStep71 Is
Nothing And cExecStep72 Is Nothing And
cExecStep73 Is Nothing And cExecStep74 Is
Nothing And cExecStep75 Is Nothing And
cExecStep76 Is Nothing And cExecStep77 Is
Nothing And cExecStep78 Is Nothing And
cExecStep79 Is Nothing And _
      cExecStep80 Is Nothing And cExecStep81 Is
Nothing And cExecStep82 Is Nothing And
cExecStep83 Is Nothing And cExecStep84 Is
Nothing And cExecStep85 Is Nothing And
cExecStep86 Is Nothing And cExecStep87 Is
Nothing And cExecStep88 Is Nothing And
cExecStep89 Is Nothing And _
      cExecStep90 Is Nothing And cExecStep91 Is
Nothing And cExecStep92 Is Nothing And
cExecStep93 Is Nothing And cExecStep94 Is
Nothing And cExecStep95 Is Nothing And
cExecStep96 Is Nothing And cExecStep97 Is
Nothing And cExecStep98 Is Nothing And
cExecStep99 Is Nothing Then
      ' Then...
      WriteToWspLog (mintRunComplete)
      RaiseEvent
RunComplete(Determine64BitTime())
   Else
      ' Abort each of the steps that is currently
executing.
      If Not cExecStep1 Is Nothing Then
         cExecStep1.Abort
      End If
      If Not cExecStep2 Is Nothing Then
         cExecStep2.Abort
      End If
      If Not cExecStep3 Is Nothing Then
         cExecStep3.Abort
      End If
      If Not cExecStep4 Is Nothing Then
         cExecStep4.Abort
      End If
      If Not cExecStep5 Is Nothing Then
         cExecStep5.Abort
      End If
      If Not cExecStep6 Is Nothing Then
         cExecStep6.Abort
      End If
      If Not cExecStep7 Is Nothing Then
         cExecStep7.Abort
      End If
      If Not cExecStep8 Is Nothing Then
         cExecStep8.Abort
      End If
      If Not cExecStep9 Is Nothing Then
         cExecStep9.Abort
      End If
      If Not cExecStep10 Is Nothing Then
         cExecStep10.Abort
      End If
      If Not cExecStep11 Is Nothing Then
         cExecStep11.Abort
      End If
      If Not cExecStep12 Is Nothing Then
         cExecStep12.Abort
      End If
      If Not cExecStep13 Is Nothing Then
         cExecStep13.Abort
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
    End If

    If Not cExecStep14 Is Nothing Then
        cExecStep14.Abort
    End If

    If Not cExecStep15 Is Nothing Then
        cExecStep15.Abort
    End If

    If Not cExecStep16 Is Nothing Then
        cExecStep16.Abort
    End If

    If Not cExecStep17 Is Nothing Then
        cExecStep17.Abort
    End If

    If Not cExecStep18 Is Nothing Then
        cExecStep18.Abort
    End If

    If Not cExecStep19 Is Nothing Then
        cExecStep19.Abort
    End If

    If Not cExecStep20 Is Nothing Then
        cExecStep20.Abort
    End If

    If Not cExecStep21 Is Nothing Then
        cExecStep21.Abort
    End If

    If Not cExecStep22 Is Nothing Then
        cExecStep22.Abort
    End If

    If Not cExecStep23 Is Nothing Then
        cExecStep23.Abort
    End If

    If Not cExecStep24 Is Nothing Then
        cExecStep24.Abort
    End If

    If Not cExecStep25 Is Nothing Then
        cExecStep25.Abort
    End If

    If Not cExecStep26 Is Nothing Then
        cExecStep26.Abort
    End If

    If Not cExecStep27 Is Nothing Then
        cExecStep27.Abort
    End If

    If Not cExecStep28 Is Nothing Then
        cExecStep28.Abort
    End If

    If Not cExecStep29 Is Nothing Then
        cExecStep29.Abort
    End If

    ' ============== 30 - 39
==============
    If Not cExecStep30 Is Nothing Then
        cExecStep30.Abort
    End If

    If Not cExecStep31 Is Nothing Then
        cExecStep31.Abort
    End If

    If Not cExecStep32 Is Nothing Then
        cExecStep32.Abort
    End If

    If Not cExecStep33 Is Nothing Then
        cExecStep33.Abort
    End If

    If Not cExecStep34 Is Nothing Then
        cExecStep34.Abort
    End If

    If Not cExecStep35 Is Nothing Then
        cExecStep35.Abort
    End If

    If Not cExecStep36 Is Nothing Then
        cExecStep36.Abort
    End If

    If Not cExecStep37 Is Nothing Then
        cExecStep37.Abort
    End If

    If Not cExecStep38 Is Nothing Then
        cExecStep38.Abort
    End If

    If Not cExecStep39 Is Nothing Then
        cExecStep39.Abort
    End If

    ' ============== 40 - 49
==============
    If Not cExecStep40 Is Nothing Then
        cExecStep40.Abort
    End If

    If Not cExecStep41 Is Nothing Then
        cExecStep41.Abort
    End If

    If Not cExecStep42 Is Nothing Then
        cExecStep42.Abort
    End If

    If Not cExecStep43 Is Nothing Then
        cExecStep43.Abort
    End If

    If Not cExecStep44 Is Nothing Then
        cExecStep44.Abort
    End If

    If Not cExecStep45 Is Nothing Then
        cExecStep45.Abort
    End If

    If Not cExecStep46 Is Nothing Then
        cExecStep46.Abort
    End If

    If Not cExecStep47 Is Nothing Then
        cExecStep47.Abort
    End If

    If Not cExecStep48 Is Nothing Then
        cExecStep48.Abort
    End If

    If Not cExecStep49 Is Nothing Then
        cExecStep49.Abort
    End If

    ' ============== 50 - 59
==============
    If Not cExecStep50 Is Nothing Then
        cExecStep50.Abort
    End If

    If Not cExecStep51 Is Nothing Then
        cExecStep51.Abort
    End If

    If Not cExecStep52 Is Nothing Then
        cExecStep52.Abort
    End If

    If Not cExecStep53 Is Nothing Then
        cExecStep53.Abort
    End If

    If Not cExecStep54 Is Nothing Then
        cExecStep54.Abort
    End If

    If Not cExecStep55 Is Nothing Then
        cExecStep55.Abort
    End If

    If Not cExecStep56 Is Nothing Then
        cExecStep56.Abort
    End If

    If Not cExecStep57 Is Nothing Then
        cExecStep57.Abort
    End If

    If Not cExecStep58 Is Nothing Then
        cExecStep58.Abort
    End If

    If Not cExecStep59 Is Nothing Then
        cExecStep59.Abort
    End If

    ' ============== 60 - 69
==============
    If Not cExecStep60 Is Nothing Then
        cExecStep60.Abort
    End If

    If Not cExecStep61 Is Nothing Then
        cExecStep61.Abort
    End If

    If Not cExecStep62 Is Nothing Then
        cExecStep62.Abort
    End If

    If Not cExecStep63 Is Nothing Then
        cExecStep63.Abort
    End If

    If Not cExecStep64 Is Nothing Then
        cExecStep64.Abort
    End If

    If Not cExecStep65 Is Nothing Then
        cExecStep65.Abort
    End If

    If Not cExecStep66 Is Nothing Then
        cExecStep66.Abort
    End If

    If Not cExecStep67 Is Nothing Then
        cExecStep67.Abort
    End If
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
    If Not cExecStep68 Is Nothing Then
        cExecStep68.Abort
    End If

    If Not cExecStep69 Is Nothing Then
        cExecStep69.Abort
    End If

    ' ============== 70 - 79
==============
    If Not cExecStep70 Is Nothing Then
        cExecStep70.Abort
    End If

    If Not cExecStep71 Is Nothing Then
        cExecStep71.Abort
    End If

    If Not cExecStep72 Is Nothing Then
        cExecStep72.Abort
    End If

    If Not cExecStep73 Is Nothing Then
        cExecStep73.Abort
    End If

    If Not cExecStep74 Is Nothing Then
        cExecStep74.Abort
    End If

    If Not cExecStep75 Is Nothing Then
        cExecStep75.Abort
    End If

    If Not cExecStep76 Is Nothing Then
        cExecStep76.Abort
    End If

    If Not cExecStep77 Is Nothing Then
        cExecStep77.Abort
    End If

    If Not cExecStep78 Is Nothing Then
        cExecStep78.Abort
    End If

    If Not cExecStep79 Is Nothing Then
        cExecStep79.Abort
    End If

    ' ============== 80 - 89
==============
    If Not cExecStep80 Is Nothing Then
        cExecStep80.Abort
    End If

    If Not cExecStep81 Is Nothing Then
        cExecStep81.Abort
    End If

    If Not cExecStep82 Is Nothing Then
        cExecStep82.Abort
    End If

    If Not cExecStep83 Is Nothing Then
        cExecStep83.Abort
    End If

    If Not cExecStep84 Is Nothing Then
        cExecStep84.Abort
    End If

    If Not cExecStep85 Is Nothing Then
        cExecStep85.Abort
    End If

    If Not cExecStep86 Is Nothing Then
        cExecStep86.Abort
    End If

    If Not cExecStep87 Is Nothing Then
        cExecStep87.Abort
    End If

    If Not cExecStep88 Is Nothing Then
        cExecStep88.Abort
    End If

    If Not cExecStep89 Is Nothing Then
        cExecStep89.Abort
    End If

    ' ============== 90 - 99
==============
    If Not cExecStep90 Is Nothing Then
        cExecStep90.Abort
    End If

    If Not cExecStep91 Is Nothing Then
        cExecStep91.Abort
    End If

    If Not cExecStep92 Is Nothing Then
        cExecStep92.Abort
    End If

    If Not cExecStep93 Is Nothing Then
        cExecStep93.Abort
    End If

    If Not cExecStep94 Is Nothing Then
        cExecStep94.Abort
    End If

    If Not cExecStep95 Is Nothing Then
        cExecStep95.Abort
    End If

    If Not cExecStep96 Is Nothing Then
        cExecStep96.Abort
    End If

    If Not cExecStep97 Is Nothing Then
        cExecStep97.Abort
    End If

    If Not cExecStep98 Is Nothing Then
        cExecStep98.Abort
    End If

    If Not cExecStep99 Is Nothing Then
        cExecStep99.Abort
    End If

  End If

  Exit Sub

AbortErr:
  Call LogErrors(Errors)
  On Error GoTo 0
  ShowError errAbortFailed
  ' Try to abort the remaining steps, if any
  Resume Next

End Sub
Public Sub AbortSiblings(cTermInstance As
cInstance)
  On Error GoTo AbortSiblingsErr

  ' Abort each of the steps that is currently
executing.
  If Not cExecStep1 Is Nothing Then
    If cExecStep1.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
      cExecStep1.Abort
    End If
  End If

  If Not cExecStep2 Is Nothing Then
    If cExecStep2.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
      cExecStep2.Abort
    End If
  End If

  If Not cExecStep3 Is Nothing Then
    If cExecStep3.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
      cExecStep3.Abort
    End If
  End If

  If Not cExecStep4 Is Nothing Then
    If cExecStep4.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
      cExecStep4.Abort
    End If
  End If

  If Not cExecStep5 Is Nothing Then
    If cExecStep5.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
      cExecStep5.Abort
    End If
  End If

  If Not cExecStep6 Is Nothing Then
    If cExecStep6.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
      cExecStep6.Abort
    End If
  End If

  If Not cExecStep7 Is Nothing Then
    If cExecStep7.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
      cExecStep7.Abort
    End If
  End If

  If Not cExecStep8 Is Nothing Then
    If cExecStep8.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
      cExecStep8.Abort
    End If
  End If

  If Not cExecStep9 Is Nothing Then
    If cExecStep9.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
      cExecStep9.Abort
    End If
  End If

  If Not cExecStep10 Is Nothing Then
    If cExecStep10.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
      cExecStep10.Abort
    End If
  End If

  If Not cExecStep11 Is Nothing Then
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
      If
cExecStep11.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
         cExecStep11.Abort
      End If
   End If

   If Not cExecStep12 Is Nothing Then
      If
cExecStep12.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
         cExecStep12.Abort
      End If
   End If

   If Not cExecStep13 Is Nothing Then
      If
cExecStep13.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
         cExecStep13.Abort
      End If
   End If

   If Not cExecStep14 Is Nothing Then
      If
cExecStep14.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
         cExecStep14.Abort
      End If
   End If

   If Not cExecStep15 Is Nothing Then
      If
cExecStep15.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
         cExecStep15.Abort
      End If
   End If

   If Not cExecStep16 Is Nothing Then
      If
cExecStep16.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
         cExecStep16.Abort
      End If
   End If

   If Not cExecStep17 Is Nothing Then
      If
cExecStep17.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
         cExecStep17.Abort
      End If
   End If

   If Not cExecStep18 Is Nothing Then
      If
cExecStep18.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
         cExecStep18.Abort
      End If
   End If

   If Not cExecStep19 Is Nothing Then
      If
cExecStep19.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
         cExecStep19.Abort
      End If
   End If

   If Not cExecStep20 Is Nothing Then
      If
cExecStep20.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
         cExecStep20.Abort
      End If
   End If

   If Not cExecStep21 Is Nothing Then
      If cExecStep21.ExecuteStep.ParentStepId
= cTermInstance.Step.ParentStepId Then
         cExecStep21.Abort
      End If
   End If

   If Not cExecStep22 Is Nothing Then
      If cExecStep22.ExecuteStep.ParentStepId
= cTermInstance.Step.ParentStepId Then
         cExecStep22.Abort
      End If
   End If

   If Not cExecStep23 Is Nothing Then
      If cExecStep23.ExecuteStep.ParentStepId
= cTermInstance.Step.ParentStepId Then
         cExecStep23.Abort
      End If
   End If

   If Not cExecStep24 Is Nothing Then
      If cExecStep24.ExecuteStep.ParentStepId
= cTermInstance.Step.ParentStepId Then
         cExecStep24.Abort
      End If
   End If

   If Not cExecStep25 Is Nothing Then
      If cExecStep25.ExecuteStep.ParentStepId
= cTermInstance.Step.ParentStepId Then
         cExecStep25.Abort
      End If
   End If

   If Not cExecStep26 Is Nothing Then
      If cExecStep26.ExecuteStep.ParentStepId
= cTermInstance.Step.ParentStepId Then
         cExecStep26.Abort
      End If
   End If

   If Not cExecStep27 Is Nothing Then
      If cExecStep27.ExecuteStep.ParentStepId
= cTermInstance.Step.ParentStepId Then
         cExecStep27.Abort
      End If
   End If

   If Not cExecStep28 Is Nothing Then
      If cExecStep28.ExecuteStep.ParentStepId
= cTermInstance.Step.ParentStepId Then
         cExecStep28.Abort
      End If
   End If

   If Not cExecStep29 Is Nothing Then
      If cExecStep29.ExecuteStep.ParentStepId
= cTermInstance.Step.ParentStepId Then
         cExecStep29.Abort
      End If
   End If

   ' ============ 30
================
   If Not cExecStep30 Is Nothing Then
      If cExecStep30.ExecuteStep.ParentStepId
= cTermInstance.Step.ParentStepId Then
         cExecStep30.Abort
      End If
   End If

   If Not cExecStep31 Is Nothing Then
      If cExecStep31.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
         cExecStep31.Abort
      End If
   End If

   If Not cExecStep32 Is Nothing Then
      If cExecStep32.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
         cExecStep32.Abort
      End If
   End If

   If Not cExecStep33 Is Nothing Then
      If cExecStep33.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
         cExecStep33.Abort
      End If
   End If

   If Not cExecStep34 Is Nothing Then
      If cExecStep34.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
         cExecStep34.Abort
      End If
   End If

   If Not cExecStep35 Is Nothing Then
      If cExecStep35.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
         cExecStep35.Abort
      End If
   End If

   If Not cExecStep36 Is Nothing Then
      If cExecStep36.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
         cExecStep36.Abort
      End If
   End If

   If Not cExecStep37 Is Nothing Then
      If cExecStep37.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
         cExecStep37.Abort
      End If
   End If

   If Not cExecStep38 Is Nothing Then
      If cExecStep38.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
         cExecStep38.Abort
      End If
   End If

   If Not cExecStep39 Is Nothing Then
      If cExecStep39.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
         cExecStep39.Abort
      End If
   End If

   ' ============ 40 ================
   If Not cExecStep40 Is Nothing Then
      If cExecStep40.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
         cExecStep40.Abort
      End If
   End If

   If Not cExecStep41 Is Nothing Then
      If cExecStep41.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
        cExecStep41.Abort
      End If
    End If

    If Not cExecStep42 Is Nothing Then
      If
cExecStep42.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
        cExecStep42.Abort
      End If
    End If

    If Not cExecStep43 Is Nothing Then
      If
cExecStep43.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
        cExecStep43.Abort
      End If
    End If

    If Not cExecStep44 Is Nothing Then
      If
cExecStep44.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
        cExecStep44.Abort
      End If
    End If

    If Not cExecStep45 Is Nothing Then
      If
cExecStep45.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
        cExecStep45.Abort
      End If
    End If

    If Not cExecStep46 Is Nothing Then
      If
cExecStep46.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
        cExecStep46.Abort
      End If
    End If

    If Not cExecStep47 Is Nothing Then
      If
cExecStep47.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
        cExecStep47.Abort
      End If
    End If

    If Not cExecStep48 Is Nothing Then
      If
cExecStep48.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
        cExecStep48.Abort
      End If
    End If

    If Not cExecStep49 Is Nothing Then
      If
cExecStep49.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
        cExecStep49.Abort
      End If
    End If

    ' ============ 50
=================
    If Not cExecStep50 Is Nothing Then
      If
cExecStep50.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
        cExecStep50.Abort
      End If
    End If

    If Not cExecStep51 Is Nothing Then
      If cExecStep51.ExecuteStep.ParentStepId
= cTermInstance.Step.ParentStepId Then
        cExecStep51.Abort
      End If
    End If

    If Not cExecStep52 Is Nothing Then
      If cExecStep52.ExecuteStep.ParentStepId
= cTermInstance.Step.ParentStepId Then
        cExecStep52.Abort
      End If
    End If

    If Not cExecStep53 Is Nothing Then
      If cExecStep53.ExecuteStep.ParentStepId
= cTermInstance.Step.ParentStepId Then
        cExecStep53.Abort
      End If
    End If

    If Not cExecStep54 Is Nothing Then
      If cExecStep54.ExecuteStep.ParentStepId
= cTermInstance.Step.ParentStepId Then
        cExecStep54.Abort
      End If
    End If

    If Not cExecStep55 Is Nothing Then
      If cExecStep55.ExecuteStep.ParentStepId
= cTermInstance.Step.ParentStepId Then
        cExecStep55.Abort
      End If
    End If

    If Not cExecStep56 Is Nothing Then
      If cExecStep56.ExecuteStep.ParentStepId
= cTermInstance.Step.ParentStepId Then
        cExecStep56.Abort
      End If
    End If

    If Not cExecStep57 Is Nothing Then
      If cExecStep57.ExecuteStep.ParentStepId
= cTermInstance.Step.ParentStepId Then
        cExecStep57.Abort
      End If
    End If

    If Not cExecStep58 Is Nothing Then
      If cExecStep58.ExecuteStep.ParentStepId
= cTermInstance.Step.ParentStepId Then
        cExecStep58.Abort
      End If
    End If

    If Not cExecStep59 Is Nothing Then
      If cExecStep59.ExecuteStep.ParentStepId
= cTermInstance.Step.ParentStepId Then
        cExecStep59.Abort
      End If
    End If

    ' ============ 60
=================
    If Not cExecStep60 Is Nothing Then
      If cExecStep60.ExecuteStep.ParentStepId
= cTermInstance.Step.ParentStepId Then
        cExecStep60.Abort
      End If
    End If

    If Not cExecStep61 Is Nothing Then
      If cExecStep61.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
        cExecStep61.Abort
      End If
    End If

    If Not cExecStep62 Is Nothing Then
      If cExecStep62.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
        cExecStep62.Abort
      End If
    End If

    If Not cExecStep63 Is Nothing Then
      If cExecStep63.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
        cExecStep63.Abort
      End If
    End If

    If Not cExecStep64 Is Nothing Then
      If cExecStep64.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
        cExecStep64.Abort
      End If
    End If

    If Not cExecStep65 Is Nothing Then
      If cExecStep65.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
        cExecStep65.Abort
      End If
    End If

    If Not cExecStep66 Is Nothing Then
      If cExecStep66.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
        cExecStep66.Abort
      End If
    End If

    If Not cExecStep67 Is Nothing Then
      If cExecStep67.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
        cExecStep67.Abort
      End If
    End If

    If Not cExecStep68 Is Nothing Then
      If cExecStep68.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
        cExecStep68.Abort
      End If
    End If

    If Not cExecStep69 Is Nothing Then
      If cExecStep69.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
        cExecStep69.Abort
      End If
    End If

    ' ============ 70 =================
    If Not cExecStep70 Is Nothing Then
      If cExecStep70.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
        cExecStep70.Abort
      End If
    End If

    If Not cExecStep71 Is Nothing Then
      If cExecStep71.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
        cExecStep71.Abort
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
        End If
    End If

    If Not cExecStep72 Is Nothing Then
        If
cExecStep72.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
            cExecStep72.Abort
        End If
    End If

    If Not cExecStep73 Is Nothing Then
        If
cExecStep73.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
            cExecStep73.Abort
        End If
    End If

    If Not cExecStep74 Is Nothing Then
        If
cExecStep74.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
            cExecStep74.Abort
        End If
    End If

    If Not cExecStep75 Is Nothing Then
        If
cExecStep75.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
            cExecStep75.Abort
        End If
    End If

    If Not cExecStep76 Is Nothing Then
        If
cExecStep76.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
            cExecStep76.Abort
        End If
    End If

    If Not cExecStep77 Is Nothing Then
        If
cExecStep77.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
            cExecStep77.Abort
        End If
    End If

    If Not cExecStep78 Is Nothing Then
        If
cExecStep78.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
            cExecStep78.Abort
        End If
    End If

    If Not cExecStep79 Is Nothing Then
        If
cExecStep79.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
            cExecStep79.Abort
        End If
    End If

    ' ============ 80
=================
    If Not cExecStep80 Is Nothing Then
        If
cExecStep80.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
            cExecStep80.Abort
        End If
    End If

    If Not cExecStep81 Is Nothing Then
        If cExecStep81.ExecuteStep.ParentStepId
= cTermInstance.Step.ParentStepId Then
            cExecStep81.Abort
        End If
    End If

    If Not cExecStep82 Is Nothing Then
        If cExecStep82.ExecuteStep.ParentStepId
= cTermInstance.Step.ParentStepId Then
            cExecStep82.Abort
        End If
    End If

    If Not cExecStep83 Is Nothing Then
        If cExecStep83.ExecuteStep.ParentStepId
= cTermInstance.Step.ParentStepId Then
            cExecStep83.Abort
        End If
    End If

    If Not cExecStep84 Is Nothing Then
        If cExecStep84.ExecuteStep.ParentStepId
= cTermInstance.Step.ParentStepId Then
            cExecStep84.Abort
        End If
    End If

    If Not cExecStep85 Is Nothing Then
        If cExecStep85.ExecuteStep.ParentStepId
= cTermInstance.Step.ParentStepId Then
            cExecStep85.Abort
        End If
    End If

    If Not cExecStep86 Is Nothing Then
        If cExecStep86.ExecuteStep.ParentStepId
= cTermInstance.Step.ParentStepId Then
            cExecStep86.Abort
        End If
    End If

    If Not cExecStep87 Is Nothing Then
        If cExecStep87.ExecuteStep.ParentStepId
= cTermInstance.Step.ParentStepId Then
            cExecStep87.Abort
        End If
    End If

    If Not cExecStep88 Is Nothing Then
        If cExecStep88.ExecuteStep.ParentStepId
= cTermInstance.Step.ParentStepId Then
            cExecStep88.Abort
        End If
    End If

    If Not cExecStep89 Is Nothing Then
        If cExecStep89.ExecuteStep.ParentStepId
= cTermInstance.Step.ParentStepId Then
            cExecStep89.Abort
        End If
    End If

    ' ============ 90
=================
    If Not cExecStep90 Is Nothing Then
        If cExecStep90.ExecuteStep.ParentStepId
= cTermInstance.Step.ParentStepId Then
            cExecStep90.Abort
        End If
    End If

    If Not cExecStep91 Is Nothing Then
        If cExecStep91.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
            cExecStep91.Abort
        End If
    End If

    If Not cExecStep92 Is Nothing Then
        If cExecStep92.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
            cExecStep92.Abort
        End If
    End If

    If Not cExecStep93 Is Nothing Then
        If cExecStep93.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
            cExecStep93.Abort
        End If
    End If

    If Not cExecStep94 Is Nothing Then
        If cExecStep94.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
            cExecStep94.Abort
        End If
    End If

    If Not cExecStep95 Is Nothing Then
        If cExecStep95.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
            cExecStep95.Abort
        End If
    End If

    If Not cExecStep96 Is Nothing Then
        If cExecStep96.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
            cExecStep96.Abort
        End If
    End If

    If Not cExecStep97 Is Nothing Then
        If cExecStep97.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
            cExecStep97.Abort
        End If
    End If

    If Not cExecStep98 Is Nothing Then
        If cExecStep98.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
            cExecStep98.Abort
        End If
    End If

    If Not cExecStep99 Is Nothing Then
        If cExecStep99.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
            cExecStep99.Abort
        End If
    End If

    Exit Sub

AbortSiblingsErr:
    Call LogErrors(Errors)
    On Error GoTo 0
    ShowError errAbortFailed
    ' Try to abort the remaining steps, if any
    Resume Next

End Sub
Private Sub ExecutionFailed(cTermStep As
cRunStep)
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
    ' Called when execution of a step fails for
any reason - ensure that execution
    ' continues

    On Error GoTo ExecutionFailedErr

    Call AddFreeProcess(cTermStep.Index)

    Call RunBranch(mstrCurBranchRoot)

    Exit Sub

ExecutionFailedErr:
    ' Log the error code raised by Visual
Basic - do not raise an error here!
    Call LogErrors(Errors)

End Sub
Private Sub FreeExecStep(lngIndex As
Long)
    ' Frees an instance of a cExecuteSM
object depending on the index
    On Error GoTo FreeExecStepErr

    Select Case lngIndex + 1
        Case 1
            Set cExecStep1 = Nothing
        Case 2
            Set cExecStep2 = Nothing
        Case 3
            Set cExecStep3 = Nothing
        Case 4
            Set cExecStep4 = Nothing
        Case 5
            Set cExecStep5 = Nothing
        Case 6
            Set cExecStep6 = Nothing
        Case 7
            Set cExecStep7 = Nothing
        Case 8
            Set cExecStep8 = Nothing
        Case 9
            Set cExecStep9 = Nothing
        Case 10
            Set cExecStep10 = Nothing
        Case 11
            Set cExecStep11 = Nothing
        Case 12
            Set cExecStep12 = Nothing
        Case 13
            Set cExecStep13 = Nothing
        Case 14
            Set cExecStep14 = Nothing
        Case 15
            Set cExecStep15 = Nothing
        Case 16
            Set cExecStep16 = Nothing
        Case 17
            Set cExecStep17 = Nothing
        Case 18
            Set cExecStep18 = Nothing
        Case 19
            Set cExecStep19 = Nothing
        Case 20
            Set cExecStep20 = Nothing
        Case 21
            Set cExecStep21 = Nothing
        Case 22
            Set cExecStep22 = Nothing
        Case 23
            Set cExecStep23 = Nothing
        Case 24
            Set cExecStep24 = Nothing
        Case 25
            Set cExecStep25 = Nothing
        Case 26
            Set cExecStep26 = Nothing
        Case 27
            Set cExecStep27 = Nothing
        Case 28
            Set cExecStep28 = Nothing
        Case 29
            Set cExecStep29 = Nothing
        Case 30
            Set cExecStep30 = Nothing
        Case 31
            Set cExecStep31 = Nothing
        Case 32
            Set cExecStep32 = Nothing
        Case 33
            Set cExecStep33 = Nothing
        Case 34
            Set cExecStep34 = Nothing
        Case 35
            Set cExecStep35 = Nothing
        Case 36
            Set cExecStep36 = Nothing
        Case 37
            Set cExecStep37 = Nothing
        Case 38
            Set cExecStep38 = Nothing
        Case 39
            Set cExecStep39 = Nothing
        Case 40
            Set cExecStep40 = Nothing
        Case 41
            Set cExecStep41 = Nothing
        Case 42
            Set cExecStep42 = Nothing
        Case 43
            Set cExecStep43 = Nothing
        Case 44
            Set cExecStep44 = Nothing
        Case 45
            Set cExecStep45 = Nothing
        Case 46
            Set cExecStep46 = Nothing
        Case 47
            Set cExecStep47 = Nothing
        Case 48
            Set cExecStep48 = Nothing
        Case 49
            Set cExecStep49 = Nothing
        Case 50
            Set cExecStep50 = Nothing
        Case 51
            Set cExecStep51 = Nothing
        Case 52
            Set cExecStep52 = Nothing
        Case 53
            Set cExecStep53 = Nothing
        Case 54
            Set cExecStep54 = Nothing
        Case 55
            Set cExecStep55 = Nothing
        Case 56
            Set cExecStep56 = Nothing
        Case 57
            Set cExecStep57 = Nothing
        Case 58
            Set cExecStep58 = Nothing
        Case 59
            Set cExecStep59 = Nothing
        Case 60
            Set cExecStep60 = Nothing
        Case 61
            Set cExecStep61 = Nothing
        Case 62
            Set cExecStep62 = Nothing
        Case 63
            Set cExecStep63 = Nothing
        Case 64
            Set cExecStep64 = Nothing
        Case 65
            Set cExecStep65 = Nothing
        Case 66
            Set cExecStep66 = Nothing
        Case 67
            Set cExecStep67 = Nothing
        Case 68
            Set cExecStep68 = Nothing
        Case 69
            Set cExecStep69 = Nothing
        Case 70
            Set cExecStep70 = Nothing
        Case 71
            Set cExecStep71 = Nothing
        Case 72
            Set cExecStep72 = Nothing
        Case 73
            Set cExecStep73 = Nothing
        Case 74
            Set cExecStep74 = Nothing
        Case 75
            Set cExecStep75 = Nothing
        Case 76
            Set cExecStep76 = Nothing
        Case 77
            Set cExecStep77 = Nothing
        Case 78
            Set cExecStep78 = Nothing
        Case 79
            Set cExecStep79 = Nothing
        Case 80
            Set cExecStep80 = Nothing
        Case 81
            Set cExecStep81 = Nothing
        Case 82
            Set cExecStep82 = Nothing
        Case 83
            Set cExecStep83 = Nothing
        Case 84
            Set cExecStep84 = Nothing
        Case 85
            Set cExecStep85 = Nothing
        Case 86
            Set cExecStep86 = Nothing
        Case 87
            Set cExecStep87 = Nothing
        Case 88
            Set cExecStep88 = Nothing
        Case 89
            Set cExecStep89 = Nothing
        Case 90
            Set cExecStep90 = Nothing
        Case 91
            Set cExecStep91 = Nothing
        Case 92
            Set cExecStep92 = Nothing
        Case 93
            Set cExecStep93 = Nothing
        Case 94
            Set cExecStep94 = Nothing
        Case 95
            Set cExecStep95 = Nothing
        Case 96
            Set cExecStep96 = Nothing
        Case 97
            Set cExecStep97 = Nothing
        Case 98
            Set cExecStep98 = Nothing
        Case 99
            Set cExecStep99 = Nothing
        Case Else
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
        BugAssert False, "FreeExecStep:
Invalid index value!"
    End Select

    Exit Sub

FreeExecStepErr:
    ' Log the error code raised by Visual
Basic
    Call LogErrors(Errors)

End Sub
Private Sub ProcessAskFailures()
    ' This procedure is called when a step
with a continuation criteria = Ask has failed.
    ' Wait for all running processes to
complete before displaying an
Abort/Retry/Fail
    ' message to the user. We process every
Ask step that has failed and use a simple
    ' algorithm to determine what to do next.
    ' 1. An abort response to any failure
results in an immediate abort of the run
    ' 2. A continue means the run continues -
this failure is popped off the failure list.
    ' 3. A retry means that the execution
details for the instance are cleared and the
    '    step is re-executed.
    Dim lIndex As Long
    Dim cStepRec As cStep
    Dim cNextInst As cInstance
    Dim cFailureRec As cFailedStep

    On Error GoTo ProcessAskFailuresErr

    ' Display a popup message for all steps
that have failed with a continuation
    ' criteria of Ask
    For lIndex = mcFailures.Count - 1 To 0
Step -1

        Set cFailureRec = mcFailures(lIndex)

        If cFailureRec.ContCriteria =
gintOnFailureAsk Then
            Set cStepRec =
mcRunSteps.QueryStep(cFailureRec.StepId)
            ' Ask the user whether to
abort/retry/continue
            #If RUN_ONLY Then
                cFailureRec.AskResponse =
ShowMessageBox(0, _
                    "Step '" &
GetStepNodeText(cStepRec) & "' failed. " &
_
                    "Select Abort to abort run
and Ignore to continue. " & _
                    "Select Retry to re-execute
the failed step.", _
                    "Step Failure", _

MB_ABORTRETRYIGNORE +
MB_APPLMODAL +
MB_ICONEXCLAMATION)
            #Else
                cFailureRec.AskResponse =
ShowMessageBox(frmRunning.hWnd, _
                    "Step '" &
GetStepNodeText(cStepRec) & "' failed. " &
_
                    "Select Abort to abort run
and Ignore to continue. " & _
                    "Select Retry to re-execute
the failed step.", _
                    "Step Failure", _
                    MB_ABORTRETRYIGNORE
+ MB_APPLMODAL +
MB_ICONEXCLAMATION)
            #End If

            ' Process an abort response immediately
            If cFailureRec.AskResponse =
IDABORT Then
                mblnAbort = True
                Set cNextInst =
mcInstances.QueryInstance(cFailureRec.Instan
ceId)
                Call RunPendingSiblings(cNextInst,
cFailureRec.EndTime)
                Exit For
            End If
        End If

    Next lIndex

    ' Process all failed steps for which we have
Ignore and Retry responses.
    If Not mblnAbort Then
        ' Navigate in reverse order since we'll be
deleting items from the collection
        For lIndex = mcFailures.Count - 1 To 0
Step -1
            If mcFailures(lIndex).ContCriteria =
gintOnFailureAsk Then
                mblnAsk = False
                Set cFailureRec =
mcFailures.Delete(lIndex)

                Select Case
cFailureRec.AskResponse
                    Case IDABORT
                        BugAssert True

                    Case IDRETRY
                        ' Delete all instances for the
failed step and re-try
                        ' Returns a parent instance
reference
                        Set cNextInst =
ProcessRetryStep(cFailureRec)
                        Call
RunPendingStepInBranch(mstrCurBranchRoot
, cNextInst)

                    Case IDIGNORE
                        Set cNextInst =
mcInstances.QueryInstance(cFailureRec.Instan
ceId)
                        Call
RunPendingSiblings(cNextInst,
cFailureRec.EndTime)

                End Select
            End If
        Next lIndex
    End If

    Exit Sub

ProcessAskFailuresErr:
    ' Log the error code raised by Visual Basic
    Call LogErrors(Errors)
    Err.Raise vbObjectError +
errExecuteBranchFailed, mstrModuleName, _

LoadResString(errExecuteBranchFailed)

End Sub
Private Function ProcessRetryStep(cFailureRec As
cFailedStep) As cInstance
    ' This procedure is called when a step with a
continuation criteria = Ask has failed
    ' and the user wants to re-execute the step.
    ' We delete all existing instances for the step and
reset the iterator, if
    ' any on the parent instance - this way we ensure
that the step will be executed
    ' in the next pass.
    Dim lIndex As Long
    Dim cParentInstance As cInstance
    Dim cSubStepRec As cSubStep
    Dim cStepRec As cStep

    On Error GoTo ProcessRetryStepErr

    ' Navigate in reverse order since we'll be deleting
items from the collection
    For lIndex = mcInstances.Count - 1 To 0 Step -1

        If mcInstances(lIndex).Step.StepId =
cFailureRec.StepId Then
            Set cParentInstance =
mcInstances.QueryInstance(mcInstances(lIndex).Pa
rentInstanceId)
            Set cSubStepRec =
cParentInstance.QuerySubStep(cFailureRec.StepId)
            Set cStepRec =
mcRunSteps.QueryStep(cFailureRec.StepId)

            ' Decrement the child count on the parent
instance and reset the
            ' step iterators on the sub-step record, if any
-
            ' all the iterations of the step will be re-
executed.
            cParentInstance.ChildDeleted
cFailureRec.StepId
            cParentInstance.AllComplete = False
            cParentInstance.AllStarted = False

            cSubStepRec.InitializeIt cStepRec,
mcParameters

            ' Now delete the current instance
            Set ProcessRetryStep =
mcInstances.Delete(lIndex)
        End If
    Next lIndex

    Exit Function

ProcessRetryStepErr:
    ' Log the error code raised by Visual Basic
    Call LogErrors(Errors)
    Err.Raise vbObjectError +
errExecuteBranchFailed, mstrModuleName, _
            LoadResString(errExecuteBranchFailed)

End Function


Private Sub RunNextStep(ByVal
dtmCompleteTime As Currency, ByVal lngIndex
As Long, _
        ByVal InstanceId As Long, ByVal
ExecutionStatus As InstanceStatus)
    ' Checks if there are any steps remaining to be
    ' executed in the current branch. If so, it executes
    ' the step.
    Dim cTermInstance As cInstance
    Dim cFailure As cFailedStep

    On Error GoTo RunNextStepErr
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   BugMessage "RunNextStep: cExecStep"
& CStr(lngIndex + 1) & " has completed."

   Call mcTermSteps.Delete
   Call FreeExecStep(lngIndex)

   ' Call a procedure to add the freed up
object to the list
   Call AddFreeProcess(lngIndex)

   Set cTermInstance =
mcInstances.QueryInstance(InstanceId)
   cTermInstance.Status = ExecutionStatus

   If ExecutionStatus = gintFailed Then
      If
cTermInstance.Step.ContinuationCriteria =
gintOnFailureAbortSiblings Then
         Call AbortSiblings(cTermInstance)
      End If

      If Not
mcFailures.StepFailed(cTermInstance.Step.
StepId) Then
         Set cFailure = New cFailedStep
         cFailure.InstanceId =
cTermInstance.InstanceId
         cFailure.StepId =
cTermInstance.Step.StepId
         cFailure.ParentStepId =
cTermInstance.Step.ParentStepId
         cFailure.ContCriteria =
cTermInstance.Step.ContinuationCriteria
         cFailure.EndTime =
dtmCompleteTime
         mcFailures.Add cFailure
         Set cFailure = Nothing
      End If
   End If

   If ExecutionStatus = gintFailed And
cTermInstance.Step.ContinuationCriteria =
gintOnFailureAbort Then
      If StringEmpty(msAbortDtls) Then
         ' Initialize the abort message
         msAbortDtls = "Step '" &
GetStepNodeText(cTermInstance.Step) & "'
failed. " & _
               "Aborting execution. Please
check the error file for details."
      End If
      Call Abort
   ElseIf ExecutionStatus = gintFailed And
cTermInstance.Step.ContinuationCriteria =
gintOnFailureAsk Then
      mblnAsk = True

      ' If the step failed due to a Cancel
operation (Abort), abort the run
      If mblnAbort Then
         Call
RunPendingSiblings(cTermInstance,
dtmCompleteTime)
      End If
   Else
      Call
RunPendingSiblings(cTermInstance,
dtmCompleteTime)
   End If

   If mblnAbort Then
      If Not AnyStepRunning(mcFreeSteps,
mbarrFree) And Not
StringEmpty(msAbortDtls) Then
         ' Display an error only if the abort is
due to a failure
         ' We had to abort since a step failed -
since no other steps are currently
         ' running, we can display a message to
the user saying that we had to abort
         #If RUN_ONLY Then
            Call ShowMessageBox(0,
msAbortDtls, "Run Aborted", _
                  MB_APPLMODAL + MB_OK
+ MB_ICONEXCLAMATION)
         #Else
            Call
ShowMessageBox(frmRunning.hWnd,
msAbortDtls, "Run Aborted", _
                  MB_APPLMODAL + MB_OK
+ MB_ICONEXCLAMATION)
         #End If
'          MsgBox msAbortDtls, vbOKOnly,
"Run Aborted"
      End If
   ElseIf mblnAsk Then
      If Not AnyStepRunning(mcFreeSteps,
mbarrFree) Then
         ' Ask the user whether to
abort/retry/ignore failed steps
         Call ProcessAskFailures
      End If
   End If

   Exit Sub

RunNextStepErr:
   ' Log the error code raised by Visual Basic
   Call LogErrors(Errors)
   WriteError errExecuteBranchFailed,
mstrSource
   Call ResetForm(lngIndex)

End Sub
Public Sub StopRun()

   ' Setting the Abort flag to True will ensure
that we
   ' don't execute any more steps
   mblnAbort = True

End Sub

Private Sub CreateDummyInstance(strRootKey
As String)

   Dim cNewInstance As cInstance
   Dim cSubStepDtls  As cStep
   Dim lngSubStepId As Long

   On Error GoTo CreateDummyInstanceErr

   ' Create a new instance of the step
   ' initialize substeps for the step
   Set cNewInstance = New cInstance

   ' There can be multiple iterations of the top
level nodes
   ' running at the same time, but only one
branch at any
   ' time - so enforce a degree of parallelism of
1 on this
   ' node!
   Set cNewInstance.Step = New cStep
   cNewInstance.DegreeParallelism = 1
   cNewInstance.Key = mstrDummyRootKey

   cNewInstance.InstanceId = NewInstanceId
   cNewInstance.ParentInstanceId = 0
   lngSubStepId =
MakeIdentifierValid(strRootKey)

   Set cSubStepDtls =
mcRunSteps.QueryStep(lngSubStepId)
   If cSubStepDtls.EnabledFlag Then
      ' Create a child node for the step corresponding
to
      ' the root node of the branch being currently
executed,
      ' only if it has been enabled
      Call
cNewInstance.CreateSubStep(cSubStepDtls,
mcParameters)
   End If

   mcInstances.Add cNewInstance
   Set cNewInstance.Iterators =
DetermineIterators(cNewInstance)

   ' Set a reference to the newly created dummy
instance
   Set mcDummyRootInstance = cNewInstance

   Set cNewInstance = Nothing

   Exit Sub

CreateDummyInstanceErr:
   ' Log the error code raised by Visual Basic
   Call LogErrors(Errors)
   On Error GoTo 0
   mstrSource = mstrModuleName &
"CreateDummyInstance"
   Err.Raise vbObjectError +
errCreateInstanceFailed, _
         mstrSource,
LoadResString(errCreateInstanceFailed)

End Sub
Private Function CreateInstance(cExecStep As
cStep, _
      cParentInstance As cInstance) As cInstance
   ' Creates a new instance of the passed in step.
Returns
   ' a reference to the newly created instance object.

   Dim cNewInstance As cInstance
   Dim nodChild As cStep
   Dim lngSubStepId As Long

   On Error GoTo CreateInstanceErr

   ' Create a new instance of the step
   ' initialize substeps for the step
   Set cNewInstance = New cInstance
   Set cNewInstance.Step = cExecStep
   cNewInstance.Key =
MakeKeyValid(cExecStep.StepId,
cExecStep.StepType)
   cNewInstance.ParentInstanceId =
cParentInstance.InstanceId
   cNewInstance.InstanceId = NewInstanceId
   ' Validate the degree of parallelism field before
assigning it to the instance -
   ' (the parameter value might have been set to an
invalid value at runtime)
   Call
ValidateParallelism(cExecStep.DegreeParallelism,
_
         cExecStep.WorkspaceId,
ParamsInWsp:=mcParameters)
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   cNewInstance.DegreeParallelism =
SubstituteParameters(cExecStep.DegreePara
llelism, _
         cExecStep.WorkspaceId,
WspParameters:=mcParameters)

   If
mcNavSteps.HasChild(StepKey:=cNewInsta
nce.Key) Then
      Set nodChild =
mcNavSteps.ChildStep(StepKey:=cNewInst
ance.Key)
      Do
         If nodChild.EnabledFlag Then
            ' Create nodes for all it's substeps
only
            ' if the substeps have been enabled
            Call
cNewInstance.CreateSubStep(nodChild,
mcParameters)
         End If

         Set nodChild =
mcNavSteps.NextStep(StepId:=nodChild.St
epId)
      Loop While (Not nodChild Is Nothing)
   End If

   mcInstances.Add cNewInstance
   Set cNewInstance.Iterators =
DetermineIterators(cNewInstance)

   ' Increment the number of executing steps
on the parent
   cParentInstance.ChildExecuted
(cExecStep.StepId)

   Set CreateInstance = cNewInstance

   Exit Function

CreateInstanceErr:
   ' Log the error code raised by Visual
Basic
   Call LogErrors(Errors)
   On Error GoTo 0
   mstrSource = mstrModuleName &
"CreateInstance"
   Err.Raise vbObjectError +
errCreateInstanceFailed, _
         mstrSource,
LoadResString(errCreateInstanceFailed)

End Function
Private Function
DetermineIterators(cInstanceRec As
cInstance) As cRunColIt
   ' Returns a collection of all the iterator
values for this
   ' instance - since an iterator that is defined
at a
   ' particular level can be used in all it's
substeps, we
   ' need to navigate the step tree all the way
to the root

   Dim cRunIts As cRunColIt
   Dim cRunIt As cRunItNode
   Dim cStepIt As cIterator
   Dim cParentInst As cInstance
   Dim cSubStepRec As cSubStep
   Dim cSubStepDtls As cStep
   Dim lngSubStepId As Long
   Dim lngIndex As Long

   On Error GoTo DetermineIteratorsErr

   Set cRunIts = New cRunColIt

   If cInstanceRec.ParentInstanceId > 0 Then
      ' The last iterator for an instance of a step
is stored
      ' on it's parent! So navigate up before
beginning the
      ' search for iterator values.
      Set cParentInst =
mcInstances.QueryInstance(cInstanceRec.Pare
ntInstanceId)

      ' Get the sub-step record for the current
step
      ' on it's parent's instance!
      lngSubStepId = cInstanceRec.Step.StepId
      Set cSubStepRec =
cParentInst.QuerySubStep(lngSubStepId)
      Set cSubStepDtls =
mcRunSteps.QueryStep(lngSubStepId)

      ' And determine the next iteration value
for the
      ' substep in this instance
      Set cStepIt =
cSubStepRec.NewIteration(cSubStepDtls)

      If Not cStepIt Is Nothing Then
         ' Add the iterator details to the
collection since
         ' an iterator has been defined for the
step
         Set cRunIt = New cRunItNode
         cRunIt.IteratorName =
cSubStepDtls.IteratorName
         cRunIt.Value =
SubstituteParameters(cStepIt.Value,
cSubStepDtls.WorkspaceId,
WspParameters:=mcParameters)
         cRunIt.StepId = cSubStepRec.StepId
         cRunIts.Push cRunIt
      End If

      ' Since the parent instance has all the
iterators upto
      ' that level, read them and push them on to
the stack for
      ' this instance
      For lngIndex = 0 To
cParentInst.Iterators.Count - 1
         Set cRunIt =
cParentInst.Iterators(lngIndex)
         cRunIts.Push cRunIt
      Next lngIndex
   End If

   Set DetermineIterators = cRunIts

   Exit Function

DetermineIteratorsErr:
   ' Log the error code raised by Visual Basic
   Call LogErrors(Errors)
   On Error GoTo 0
   mstrSource = mstrModuleName &
"DetermineIterators"
   Err.Raise vbObjectError +
errExecInstanceFailed, _
         mstrSource,
LoadResString(errExecInstanceFailed)

End Function
Private Function
DetermineConstraints(cInstanceRec As cInstance, _
      intConsType As ConstraintType) As Variant
   ' Returns a collection of all the constraints for this
   ' instance of the passed in type - all the
constraints defined
   ' for the manager are executed first, followed by
those defined
   ' for the step. If a step has an iterator defined for
it, each
   ' constraint is executed only once.

   Dim cParentInst As cInstance
   Dim cTempInst As cInstance
   Dim vntConstraints As Variant
   Dim vntTempCons As Variant
   Dim cColConstraints() As Variant
   Dim lngConsCount As Long

   On Error GoTo DetermineConstraintsErr

   Set cTempInst = cInstanceRec
   lngConsCount = 0

   ' Go all the way to the root
   Do
      If cTempInst.ParentInstanceId > 0 Then
         Set cParentInst =
mcInstances.QueryInstance(cTempInst.ParentInstan
ceId)
      Else
         Set cParentInst = Nothing
      End If

      ' Check if the step has an iterator defined for it
      If cTempInst.ValidForIteration(cParentInst,
intConsType) Then
         vntTempCons =
mcRunConstraints.ConstraintsForStep( _
               cTempInst.Step.StepId,
cTempInst.Step.VersionNo, _
               intConsType, blnSort:=True, _
               blnGlobal:=False,
blnGlobalConstraintsOnly:=False)

         If Not IsEmpty(vntTempCons) Then
            ReDim Preserve
cColConstraints(lngConsCount)
            cColConstraints(lngConsCount) =
vntTempCons
            lngConsCount = lngConsCount + 1
         End If
      End If

      Set cTempInst = cParentInst

   Loop While Not cTempInst Is Nothing

   If lngConsCount > 0 Then
      vntTempCons =
OrderConstraints(cColConstraints, intConsType)
   End If

   DetermineConstraints = vntTempCons

   Exit Function

DetermineConstraintsErr:
   ' Log the error code raised by Visual Basic
   Call LogErrors(Errors)
   On Error GoTo 0
   mstrSource = mstrModuleName &
"DetermineConstraints"
   Err.Raise vbObjectError +
errExecInstanceFailed, _
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
        mstrSource,
LoadResString(errExecInstanceFailed)

End Function
Private Function
GetInstanceToExecute(cParentNode As
cInstance, _
    cSubStepRec As cSubStep, _
    cSubStepDtls As cStep) As cInstance

  Dim cSubStepInst As cInstance

  On Error GoTo GetInstanceToExecuteErr

  BugAssert Not (cParentNode Is Nothing
Or _
        cSubStepRec Is Nothing Or _
        cSubStepDtls Is Nothing), _
        "GetInstanceToExecute: Input
invalid"

  ' Check if it has iterators
  If cSubStepDtls.IteratorCount = 0 Then
    ' Check if the step has been executed
    If cSubStepRec.TasksRunning = 0 And
cSubStepRec.TasksComplete = 0 And _
          Not
mcInstances.CompletedInstanceExists(cPare
ntNode.InstanceId, cSubStepDtls) Then
        ' The sub-step hasn't been executed
yet.
        ' Create an instance for it and exit
        Set cSubStepInst =
CreateInstance(cSubStepDtls, cParentNode)
    Else
        Set cSubStepInst = Nothing
    End If
  Else
    ' Check if there are pending iterations
for the sub-step
    If Not
cSubStepRec.NextIteration(cSubStepDtls) Is
Nothing Then
        ' Pending iterations exist - create an
instance for the sub-step and exit
        Set cSubStepInst =
CreateInstance(cSubStepDtls, cParentNode)
    Else
        ' No more iterations - continue with
the next substep
        Set cSubStepInst = Nothing
    End If
  End If

  Set GetInstanceToExecute = cSubStepInst
  Exit Function

GetInstanceToExecuteErr:
  ' Log the error code raised by Visual
Basic
  Call LogErrors(Errors)
  On Error GoTo 0
  mstrSource = mstrModuleName &
"GetInstanceToExecute"
  Err.Raise vbObjectError +
errNavInstancesFailed, _
        mstrSource,
LoadResString(errNavInstancesFailed)

End Function

Public Function InstancesForStep(lngStepId
As Long, ByRef StepStatus As
InstanceStatus) As cInstances
  ' Returns an array of all the instances for a
step
  Dim lngIndex As Long
  Dim cTempInst As cInstance
  Dim cStepInstances As cInstances
  Dim cStepRec As cStep

  On Error GoTo InstancesForStepErr

  Set cStepInstances = New cInstances

  For lngIndex = 0 To mcInstances.Count - 1
    Set cTempInst = mcInstances(lngIndex)

    If cTempInst.Step.StepId = lngStepId
Then
        cStepInstances.Add cTempInst
    End If
  Next lngIndex

  If cStepInstances.Count = 0 Then
    Set cStepRec =
mcRunSteps.QueryStep(lngStepId)
    If Not
mcFailures.ExecuteSubStep(cStepRec.ParentSt
epId) Then
        StepStatus = gintAborted
    End If
    Set cStepRec = Nothing
  End If

  ' Set the return value of the function to the
array of
  ' constraints that has been built above
  Set InstancesForStep = cStepInstances

  Set cStepInstances = Nothing
  Exit Function

InstancesForStepErr:
  ' Log the error code raised by Visual Basic
  Call LogErrors(Errors)
  On Error GoTo 0
  mstrSource = mstrModuleName &
"InstancesForStep"
  Err.Raise vbObjectError +
errNavInstancesFailed, mstrSource, _
        LoadResString(errNavInstancesFailed)

End Function
Private Sub
RemoveFreeProcess(lngRunningProcess As
Long)
  ' Removes the passed in element from the
collection of
  ' free objects

  ' Confirm that the last element in the array is
the one
  ' we need to delete
  If mcFreeSteps(mcFreeSteps.Count - 1) =
lngRunningProcess Then
    mcFreeSteps.Delete
Position:=mcFreeSteps.Count - 1
  Else
    ' Ask the class to find the element and
delete it
    mcFreeSteps.Delete
Item:=lngRunningProcess
  End If

End Sub
Private Sub
AddFreeProcess(lngTerminatedProcess As
Long)
  ' Adds the passed in element to the collection of
  ' free objects

  mcFreeSteps.Add lngTerminatedProcess

End Sub

Private Sub ResetForm(Optional ByVal lngIndex
As Long)

  Dim lngTemp As Long

  On Error GoTo ResetFormErr

  ' Check if there are any running instances to wait
for
  If mcFreeSteps.Count <>
glngNumConcurrentProcesses Then

    For lngTemp = 0 To mcFreeSteps.Count - 1
      If mcFreeSteps(lngTemp) = lngIndex Then
        Exit For
      End If
    Next lngTemp

    If lngTemp <= mcFreeSteps.Count - 1 Then
      ' This process that just completed did not
exist in the list of
      ' free processes
      Call AddFreeProcess(lngIndex)
    End If

    If Not AnyStepRunning(mcFreeSteps,
mbarrFree) Then
      WriteToWspLog (mintRunComplete)
      ' All steps are complete
      RaiseEvent
RunComplete(Determine64BitTime())
    End If
  Else
    WriteToWspLog (mintRunComplete)
    RaiseEvent
RunComplete(Determine64BitTime())
  End If

  Exit Sub

ResetFormErr:

End Sub
Private Function NewInstanceId() As Long
  ' Will return new instance id's - uses a static
counter
  ' that it increments each time
  Static lngInstance As Long

  lngInstance = lngInstance + 1
  NewInstanceId = lngInstance

End Function

Private Function
RunPendingStepInBranch(strCurBranchRoot As
String, _
    Optional cExecInstance As cInstance =
Nothing) As cInstance
  ' Runs a worker step in the branch being
executed, if
  ' there are any pending execution
  ' This function is also called when a step has just
completed
  ' execution - in which case the terminated
instance is
  ' passed in as the optional parameter. When that
happens,
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
    ' we first try to execute the siblings of the
terminated
    ' step if any are pending execution.
    ' If the terminated instance has not been
passed in, we
    ' start with the dummy root instance and
navigate down,
    ' trying to find a pending worker step.

    Dim cExecSubStep As cStep
    Dim cParentInstance As cInstance
    Dim cNextInst As cInstance

    On Error GoTo
RunPendingStepInBranchErr

    If Not cExecInstance Is Nothing Then
        ' Called when an instance has
terminated
        ' When a worker step terminates, then
we need to
        ' decremement the number of running
steps on it's
        ' manager
        Set cParentInstance = _

mcInstances.QueryInstance(cExecInstance.P
arentInstanceId)

    Else
        If StringEmpty(strCurBranchRoot) Or
mcDummyRootInstance Is Nothing Then
            ' Run complete - event raised by Run
method
            Set RunPendingStepInBranch =
Nothing
            Exit Function
        End If

        ' If there are no pending steps on the
root instance,
        ' then there are no steps within the
branch that need
        ' to be executed
        If
mcDummyRootInstance.AllComplete Or
mcDummyRootInstance.AllStarted Then
            Set RunPendingStepInBranch =
Nothing
            Exit Function
        End If

        Set cParentInstance =
mcDummyRootInstance
    End If

    Do
        Set cNextInst =
GetSubStepToExecute(cParentInstance)
        If cNextInst Is Nothing Then
            ' There are no steps within the branch
that can
            ' be executed - If we are at the
dummy instance,
            ' this branch has completed executing
            If cParentInstance.Key =
mstrDummyRootKey Then
                Set cNextInst = Nothing
                Exit Do
            Else
                ' Go to the parent instance and try
to find
                ' some other sibling is pending
execution
                Set cNextInst =
mcInstances.QueryInstance(cParentInstance.Pa
rentInstanceId)

                If cParentInstance.SubSteps.Count =
0 Then
                    cNextInst.ChildTerminated
cParentInstance.Step.StepId
                End If
            End If
        End If

        BugAssert Not cNextInst Is Nothing
        Set cParentInstance = cNextInst

    Loop While cNextInst.Step.StepType <>
gintWorkerStep

    If Not cNextInst Is Nothing Then
        Call ExecuteStep(cNextInst)
    End If

    Set RunPendingStepInBranch = cNextInst

    Exit Function

RunPendingStepInBranchErr:
    ' Log the error code raised by Visual Basic
    Call LogErrors(Errors)
    On Error GoTo 0
    Err.Raise vbObjectError +
errNavInstancesFailed, _
        mstrModuleName &
"RunPendingStepInBranch",
LoadResString(errNavInstancesFailed)

End Function
Private Function
RunPendingSibling(cTermInstance As
cInstance, _
        dtmCompleteTime As Currency) As
cInstance
    ' This process is called when a step
terminates. Tries to
    ' run a sibling of the terminated step, if one
is pending
    ' execution.

    Dim cParentInstance As cInstance
    Dim cNextInst As cInstance

    On Error GoTo RunPendingSiblingErr

    If StringEmpty(mstrCurBranchRoot) Or
mcDummyRootInstance Is Nothing Then
        ' Run complete - event raised by Run
method
        Set RunPendingSibling = Nothing
        Exit Function
    End If

    BugAssert cTermInstance.ParentInstanceId
> 0, "Orphaned instance in array!"

    ' When a worker step terminates, then we
need to
    ' decremement the number of running steps
on it's
    ' manager
    Set cParentInstance =
mcInstances.QueryInstance(cTermInstance.Par
entInstanceId)

    ' Decrement the number of running
processes on the
    ' parent by 1
    Call
cParentInstance.ChildTerminated(cTermInstance.St
ep.StepId)

    ' The first step that terminates has to be a worker
    ' If it is complete, update the completed steps on
the
    ' parent by 1.
    Call
cParentInstance.ChildCompleted(cTermInstance.St
ep.StepId)
    cParentInstance.AllStarted = False

    Do
        Set cNextInst =
GetSubStepToExecute(cParentInstance,
dtmCompleteTime)
        If cNextInst Is Nothing Then
            If cParentInstance.Key =
mstrDummyRootKey Then
                Set cNextInst = Nothing
                Exit Do
            Else
                ' Go to the parent instance and try to find
                ' some other sibling is pending execution
                Set cNextInst =
mcInstances.QueryInstance(cParentInstance.ParentI
nstanceId)
                If cParentInstance.IsRunning Then
                    cNextInst.AllStarted = True
                Else
                    ' No more sub-steps to execute
                    Call
cNextInst.ChildCompleted(cParentInstance.Step.Ste
pId)
                    Call
cNextInst.ChildTerminated(cParentInstance.Step.St
epId)
                    cNextInst.AllStarted = False
                End If
            End If
        End If

        BugAssert Not cNextInst Is Nothing
        Set cParentInstance = cNextInst

    Loop While cNextInst.Step.StepType <>
gintWorkerStep

    If Not cNextInst Is Nothing Then
        Call ExecuteStep(cNextInst)
    End If

    Set RunPendingSibling = cNextInst

    Exit Function

RunPendingSiblingErr:
    ' Log the error code raised by Visual Basic
    Call LogErrors(Errors)
    On Error GoTo 0
    mstrSource = mstrModuleName &
"RunPendingSibling"
    Err.Raise vbObjectError +
errNavInstancesFailed, mstrSource, _
        LoadResString(errNavInstancesFailed)

End Function
Private Sub RunPendingSiblings(cTermInstance As
cInstance, _
        dtmCompleteTime As Currency)
    ' This process is called when a step terminates.
Tries to
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
    ' run siblings of the terminated step, if
they are pending
    ' execution.

    Dim cExecInst As cInstance

    On Error GoTo RunPendingSiblingsErr
    BugMessage "In RunPendingSiblings"

    ' Call a procedure to run the sibling of the
terminated
    ' step, if any. This procedure will also
update the
    ' number of complete/running tasks on the
manager steps.
    Set cExecInst =
RunPendingSibling(cTermInstance,
dtmCompleteTime)

    If Not cExecInst Is Nothing Then
        Do
            ' Execute any other pending steps in
the branch.
            ' The step that has just terminated
might be
            ' the last one that was executing in a
sub-branch.
            ' That would mean that we can
execute another
            ' sub-branch that might involve more
than 1 step.
            ' Pass the just executed step as a
parameter.
            Set cExecInst =
RunPendingStepInBranch(mstrCurBranchR
oot, cExecInst)
        Loop While Not cExecInst Is Nothing
    Else
        If Not
mcDummyRootInstance.IsRunning Then
            ' All steps have been executed in the
branch - run
            ' a new branch
            Call RunNewBranch
        Else
            ' There are no more steps to execute
in the current
            ' branch but we have running
processes.
        End If
    End If

    Exit Sub

RunPendingSiblingsErr:
    ' Log the error code raised by Visual
Basic
    Call LogErrors(Errors)
    On Error GoTo 0
    mstrSource = mstrModuleName &
"RunPendingSiblings"
    Err.Raise vbObjectError +
errNavInstancesFailed, _
            mstrSource,
LoadResString(errNavInstancesFailed)

End Sub

Private Sub
NoSubStepsToExecute(cMgrInstance As
cInstance, Optional dtmCompleteTime As
Currency = gdtmEmpty)
    ' Called when we cannot find any more
substeps to run for
    ' manager step - set the allcomplete or
allstarted
    ' properties to true

    If cMgrInstance.IsRunning() Then
        cMgrInstance.AllStarted = True
    Else
        cMgrInstance.AllComplete = True
        If dtmCompleteTime <> gdtmEmpty
Then
            ' Update the end time on the manager
step
            Call
TimeCompleteUpdateForStep(cMgrInstance,
dtmCompleteTime)
        End If
    End If

End Sub

Private Function
GetSubStepToExecute(cParentNode As
cInstance, _
        Optional dtmCompleteTime As Currency
= 0) As cInstance
    ' Returns the child of the passed in node that
is to be
    ' executed next. Checks if we are in the
middle of an instance
    ' being executed in which case it returns the
pending
    ' instance. Creates a new instance if there are
pending
    ' instances for a sub-step.

    Dim lngIndex As Long
    Dim cSubStepRec As cSubStep
    Dim cSubStepDtls As cStep
    Dim cSubStepInst As cInstance

    On Error GoTo GetSubStepToExecuteErr

    ' There are a number of cases that need to be
accounted
    ' for here.
    ' 1. While traversing through all enabled
nodes for the
    ' first time - instance records may not exist
for the
    ' substeps.
    ' 2. Instance records exist, and there are
processes
    ' that need to be executed for a sub-step
    ' 3. There are no more processes that need to
be currently
    ' executed (till a process completes)
    ' 4. There are no more processes that need to
be executed
    ' (All substeps have completed execution)

    ' This is the only point where we check the
Abort flag -
    ' since this is the heart of the navigation
routine that
    ' selects processes to execute. Also, when a
step terminates
    ' selection of the next process goes through
here.
    If mblnAbort Then
        Set GetSubStepToExecute = Nothing
        cParentNode.Status = gintAborted
        Exit Function
    End If

    If mblnAsk Then
        Set GetSubStepToExecute = Nothing
        Exit Function
    End If

    If Not
mcFailures.ExecuteSubStep(cParentNode.Step.Step
Id) Then
        Set GetSubStepToExecute = Nothing
        cParentNode.Status = gintAborted
        Exit Function
    End If

    ' First check if there are pending steps for the
parent!
    If cParentNode.IsPending Then
        ' Loop through all the sub-steps for the parent
node
        For lngIndex = 0 To
cParentNode.SubSteps.Count - 1
            Set cSubStepRec =
cParentNode.SubSteps(lngIndex)
            Set cSubStepDtls =
mcRunSteps.QueryStep(cSubStepRec.StepId)
            If Not
mcInstances.InstanceAborted(cSubStepRec) Then
                ' Check if the sub-step is a worker
                If cSubStepDtls.StepType =
gintWorkerStep Then
                    ' Find/create an instance to execute
                    Set cSubStepInst =
GetInstanceToExecute( _
                        cParentNode, cSubStepRec,
cSubStepDtls)
                    If Not cSubStepInst Is Nothing Then
                        Exit For
                    Else
                        ' Continue w/ the next sub-step
                    End If
                Else
                    ' The sub-step is a manager step
                    ' Check if there are any pending
instances for
                    ' the manager
                    Set cSubStepInst =
mcInstances.QueryPendingInstance( _
                        cParentNode.InstanceId,
cSubStepRec.StepId)
                    If cSubStepInst Is Nothing Then
                        ' Find/create an instance to execute
                        Set cSubStepInst =
GetInstanceToExecute( _
                            cParentNode, cSubStepRec,
cSubStepDtls)
                        If Not cSubStepInst Is Nothing Then
                            Exit For
                        Else
                            ' Continue w/ the next sub-step
                        End If
                    Else
                        ' We have found a pending instance
for the
                        ' sub-step (manager) - exit the loop
                        Exit For
                    End If
                End If
            End If
        Next lngIndex

        If lngIndex > cParentNode.SubSteps.Count - 1
Or cParentNode.SubSteps.Count = 0 Then
            ' If we could not find any sub-steps to
execute,
            ' mark the parent node as complete/all
started
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
        Call
NoSubStepsToExecute(cParentNode,
dtmCompleteTime)
        Set cSubStepInst = Nothing
      End If
   End If

   Set GetSubStepToExecute = cSubStepInst
   Exit Function

GetSubStepToExecuteErr:
   ' Log the error code raised by Visual
Basic
   Call LogErrors(Errors)
   On Error GoTo 0
   mstrSource = mstrModuleName &
"GetSubStepToExecute"
   Err.Raise vbObjectError +
errNavInstancesFailed, mstrSource, _
      LoadResString(errNavInstancesFailed)

End Function

Private Sub
TimeCompleteUpdateForStep(cMgrInstance
As cInstance, ByVal EndTime As Currency)

   ' Called when there are no more sub-steps
to execute for
   ' the manager step. It updates the end time
and status on
   ' the manager.
   Dim lElapsed As Long

   On Error GoTo
TimeCompleteUpdateForStepErr

   If cMgrInstance.Key <>
mstrDummyRootKey Then
      cMgrInstance.EndTime = EndTime
      cMgrInstance.Status = gintComplete
      lElapsed = (EndTime -
cMgrInstance.StartTime) * 10000
      cMgrInstance.ElapsedTime = lElapsed
      RaiseEvent
StepComplete(cMgrInstance.Step, EndTime,
cMgrInstance.InstanceId, lElapsed)
   End If

   Exit Sub

TimeCompleteUpdateForStepErr:
   ' Log the error code raised by Visual
Basic
   Call LogErrors(Errors)
   WriteError errUpdateDisplayFailed,
mstrModuleName &
"TimeCompleteUpdateForStep"

End Sub

Private Function GetFreeObject() As Long

   ' Check the array of free objects and
retrieve the first one
   If mcFreeSteps.Count > 0 Then
      GetFreeObject =
mcFreeSteps(mcFreeSteps.Count - 1)
   Else
      mstrSource = mstrModuleName &
"GetFreeObject"
      ShowError errMaxProcessesExceeded
      On Error GoTo 0
      Err.Raise vbObjectError +
errMaxProcessesExceeded, _
         mstrSource, _

LoadResString(errMaxProcessesExceeded)
   End If

End Function
Private Function
StepTerminated(cCompleteStep As cStep,
ByVal dtmCompleteTime As Currency, _
      ByVal lngIndex As Long, ByVal
InstanceId As Long, ByVal ExecutionStatus
As InstanceStatus) As cStep
   ' This procedure is called whenever a step
terminates.
   Dim cTermRec As cTermStep
   Dim cInstRec As cInstance
   Dim cStartInst As cInstance
   Dim lElapsed As Long
   Dim sLogLabel As String
   Dim LogLabels As New cVectorStr
   Dim iItIndex As Long

   On Error GoTo StepTerminatedErr

   Set cInstRec =
mcInstances.QueryInstance(InstanceId)
   If dtmCompleteTime <> 0 And
cInstRec.StartTime <> 0 Then
      ' Convert to milliseconds since that is the
default precision
      lElapsed = (dtmCompleteTime -
cInstRec.StartTime) * 10000
   Else
      lElapsed = 0
   End If

   Set cStartInst = cInstRec
   iItIndex = 0
   Do While cInstRec.Key <>
mstrDummyRootKey
      sLogLabel = gstrSQ &
cInstRec.Step.StepLabel & gstrSQ

      If iItIndex < cInstRec.Iterators.Count
Then
         If cStartInst.Iterators(iItIndex).StepId =
cInstRec.Step.StepId Then
            sLogLabel = sLogLabel & msIt &
gstrSQ &
cStartInst.Iterators(iItIndex).IteratorName &
gstrSQ & _
                msItValue & gstrSQ &
cStartInst.Iterators(iItIndex).Value & gstrSQ
            iItIndex = iItIndex + 1
         End If
      End If

      If cInstRec.Key = cStartInst.Key Then
         ' Append the execution status
         sLogLabel = sLogLabel & " Status: " &
gstrSQ & gsExecutionStatus(ExecutionStatus)
& gstrSQ
         If ExecutionStatus = gintFailed Then
            ' Append the continuation criteria for
the step since it failed
            sLogLabel = sLogLabel & "
Continuation Criteria: " & gstrSQ &
gsContCriteria(cInstRec.Step.ContinuationCrit
eria) & gstrSQ
         End If
      End If
      LogLabels.Add sLogLabel
      Set cInstRec =
mcInstances.QueryInstance(cInstRec.ParentInstanc
eId)
   Loop

   Call WriteToWspLog(mintStepComplete,
LogLabels, dtmCompleteTime)
   Set LogLabels = Nothing

   ' Adds the terminated step details to a queue.
   Set cTermRec = New cTermStep
   cTermRec.ExecutionStatus = ExecutionStatus
   cTermRec.Index = lngIndex
   cTermRec.InstanceId = InstanceId
   cTermRec.TimeComplete = dtmCompleteTime
   Call mcTermSteps.Add(cTermRec)
   Set cTermRec = Nothing

   RaiseEvent StepComplete(cCompleteStep,
dtmCompleteTime, InstanceId, lElapsed)

   Exit Function

StepTerminatedErr:
   ' Log the error code raised by Visual Basic
   Call LogErrors(Errors)
   WriteError errExecuteBranchFailed, mstrSource
   Call ResetForm(lngIndex)

End Function
Public Property Let RootKey(ByVal vdata As
String)

   mstrRootKey = vdata

End Property

Public Property Get RootKey() As String
   RootKey = mstrRootKey
End Property

Private Function InitExecStep() As cRunStep
   ' Since arrays of objects cannot be declared as
WithEvents,
   ' we use a limited number of objects and set a
maximum
   ' on the number of steps that can run in parallel
   ' This is a wrapper that will create an instance of
   ' a cExecuteSM object depending on the index
   Dim lngIndex As Long

   On Error GoTo InitExecStepErr

   lngIndex = GetFreeObject

   Select Case lngIndex + 1
      Case 1
         Set cExecStep1 = New cRunStep
         Set InitExecStep = cExecStep1
      Case 2
         Set cExecStep2 = New cRunStep
         Set InitExecStep = cExecStep2
      Case 3
         Set cExecStep3 = New cRunStep
         Set InitExecStep = cExecStep3
      Case 4
         Set cExecStep4 = New cRunStep
         Set InitExecStep = cExecStep4
      Case 5
         Set cExecStep5 = New cRunStep
         Set InitExecStep = cExecStep5
      Case 6
         Set cExecStep6 = New cRunStep
         Set InitExecStep = cExecStep6
      Case 7
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
    Set cExecStep7 = New cRunStep
    Set InitExecStep = cExecStep7
Case 8
    Set cExecStep8 = New cRunStep
    Set InitExecStep = cExecStep8
Case 9
    Set cExecStep9 = New cRunStep
    Set InitExecStep = cExecStep9
Case 10
    Set cExecStep10 = New cRunStep
    Set InitExecStep = cExecStep10
Case 11
    Set cExecStep11 = New cRunStep
    Set InitExecStep = cExecStep11
Case 12
    Set cExecStep12 = New cRunStep
    Set InitExecStep = cExecStep12
Case 13
    Set cExecStep13 = New cRunStep
    Set InitExecStep = cExecStep13
Case 14
    Set cExecStep14 = New cRunStep
    Set InitExecStep = cExecStep14
Case 15
    Set cExecStep15 = New cRunStep
    Set InitExecStep = cExecStep15
Case 16
    Set cExecStep16 = New cRunStep
    Set InitExecStep = cExecStep16
Case 17
    Set cExecStep17 = New cRunStep
    Set InitExecStep = cExecStep17
Case 18
    Set cExecStep18 = New cRunStep
    Set InitExecStep = cExecStep18
Case 19
    Set cExecStep19 = New cRunStep
    Set InitExecStep = cExecStep19
Case 20
    Set cExecStep20 = New cRunStep
    Set InitExecStep = cExecStep20
Case 21
    Set cExecStep21 = New cRunStep
    Set InitExecStep = cExecStep21
Case 22
    Set cExecStep22 = New cRunStep
    Set InitExecStep = cExecStep22
Case 23
    Set cExecStep23 = New cRunStep
    Set InitExecStep = cExecStep23
Case 24
    Set cExecStep24 = New cRunStep
    Set InitExecStep = cExecStep24
Case 25
    Set cExecStep25 = New cRunStep
    Set InitExecStep = cExecStep25
Case 26
    Set cExecStep26 = New cRunStep
    Set InitExecStep = cExecStep26
Case 27
    Set cExecStep27 = New cRunStep
    Set InitExecStep = cExecStep27
Case 28
    Set cExecStep28 = New cRunStep
    Set InitExecStep = cExecStep28
Case 29
    Set cExecStep29 = New cRunStep
    Set InitExecStep = cExecStep29
Case 30
    Set cExecStep30 = New cRunStep
    Set InitExecStep = cExecStep30
Case 31
    Set cExecStep31 = New cRunStep
    Set InitExecStep = cExecStep31
Case 32
    Set cExecStep32 = New cRunStep
    Set InitExecStep = cExecStep32
Case 33
    Set cExecStep33 = New cRunStep
    Set InitExecStep = cExecStep33
Case 34
    Set cExecStep34 = New cRunStep
    Set InitExecStep = cExecStep34
Case 35
    Set cExecStep35 = New cRunStep
    Set InitExecStep = cExecStep35
Case 36
    Set cExecStep36 = New cRunStep
    Set InitExecStep = cExecStep36
Case 37
    Set cExecStep37 = New cRunStep
    Set InitExecStep = cExecStep37
Case 38
    Set cExecStep38 = New cRunStep
    Set InitExecStep = cExecStep38
Case 39
    Set cExecStep39 = New cRunStep
    Set InitExecStep = cExecStep39
Case 40
    Set cExecStep40 = New cRunStep
    Set InitExecStep = cExecStep40
Case 41
    Set cExecStep41 = New cRunStep
    Set InitExecStep = cExecStep41
Case 42
    Set cExecStep42 = New cRunStep
    Set InitExecStep = cExecStep42
Case 43
    Set cExecStep43 = New cRunStep
    Set InitExecStep = cExecStep43
Case 44
    Set cExecStep44 = New cRunStep
    Set InitExecStep = cExecStep44
Case 45
    Set cExecStep45 = New cRunStep
    Set InitExecStep = cExecStep45
Case 46
    Set cExecStep46 = New cRunStep
    Set InitExecStep = cExecStep46
Case 47
    Set cExecStep47 = New cRunStep
    Set InitExecStep = cExecStep47
Case 48
    Set cExecStep48 = New cRunStep
    Set InitExecStep = cExecStep48
Case 49
    Set cExecStep49 = New cRunStep
    Set InitExecStep = cExecStep49
Case 50
    Set cExecStep50 = New cRunStep
    Set InitExecStep = cExecStep50
Case 51
    Set cExecStep51 = New cRunStep
    Set InitExecStep = cExecStep51
Case 52
    Set cExecStep52 = New cRunStep
    Set InitExecStep = cExecStep52
Case 53
    Set cExecStep53 = New cRunStep
    Set InitExecStep = cExecStep53
Case 54
    Set cExecStep54 = New cRunStep
    Set InitExecStep = cExecStep54
Case 55
    Set cExecStep55 = New cRunStep
    Set InitExecStep = cExecStep55
Case 56
    Set cExecStep56 = New cRunStep
    Set InitExecStep = cExecStep56
Case 57
    Set cExecStep57 = New cRunStep
    Set InitExecStep = cExecStep57
Case 58
    Set cExecStep58 = New cRunStep
    Set InitExecStep = cExecStep58
Case 59
    Set cExecStep59 = New cRunStep
    Set InitExecStep = cExecStep59
Case 60
    Set cExecStep60 = New cRunStep
    Set InitExecStep = cExecStep60
Case 61
    Set cExecStep61 = New cRunStep
    Set InitExecStep = cExecStep61
Case 62
    Set cExecStep62 = New cRunStep
    Set InitExecStep = cExecStep62
Case 63
    Set cExecStep63 = New cRunStep
    Set InitExecStep = cExecStep63
Case 64
    Set cExecStep64 = New cRunStep
    Set InitExecStep = cExecStep64
Case 65
    Set cExecStep65 = New cRunStep
    Set InitExecStep = cExecStep65
Case 66
    Set cExecStep66 = New cRunStep
    Set InitExecStep = cExecStep66
Case 67
    Set cExecStep67 = New cRunStep
    Set InitExecStep = cExecStep67
Case 68
    Set cExecStep68 = New cRunStep
    Set InitExecStep = cExecStep68
Case 69
    Set cExecStep69 = New cRunStep
    Set InitExecStep = cExecStep69
Case 70
    Set cExecStep70 = New cRunStep
    Set InitExecStep = cExecStep70
Case 71
    Set cExecStep71 = New cRunStep
    Set InitExecStep = cExecStep71
Case 72
    Set cExecStep72 = New cRunStep
    Set InitExecStep = cExecStep72
Case 73
    Set cExecStep73 = New cRunStep
    Set InitExecStep = cExecStep73
Case 74
    Set cExecStep74 = New cRunStep
    Set InitExecStep = cExecStep74
Case 75
    Set cExecStep75 = New cRunStep
    Set InitExecStep = cExecStep75
Case 76
    Set cExecStep76 = New cRunStep
    Set InitExecStep = cExecStep76
Case 77
    Set cExecStep77 = New cRunStep
    Set InitExecStep = cExecStep77
Case 78
    Set cExecStep78 = New cRunStep
    Set InitExecStep = cExecStep78
Case 79
    Set cExecStep79 = New cRunStep
    Set InitExecStep = cExecStep79
Case 80
    Set cExecStep80 = New cRunStep
    Set InitExecStep = cExecStep80
Case 81
    Set cExecStep81 = New cRunStep
    Set InitExecStep = cExecStep81
Case 82
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
        Set cExecStep82 = New cRunStep
        Set InitExecStep = cExecStep82
      Case 83
        Set cExecStep83 = New cRunStep
        Set InitExecStep = cExecStep83
      Case 84
        Set cExecStep84 = New cRunStep
        Set InitExecStep = cExecStep84
      Case 85
        Set cExecStep85 = New cRunStep
        Set InitExecStep = cExecStep85
      Case 86
        Set cExecStep86 = New cRunStep
        Set InitExecStep = cExecStep86
      Case 87
        Set cExecStep87 = New cRunStep
        Set InitExecStep = cExecStep87
      Case 88
        Set cExecStep88 = New cRunStep
        Set InitExecStep = cExecStep88
      Case 89
        Set cExecStep89 = New cRunStep
        Set InitExecStep = cExecStep89
      Case 90
        Set cExecStep90 = New cRunStep
        Set InitExecStep = cExecStep90
      Case 91
        Set cExecStep91 = New cRunStep
        Set InitExecStep = cExecStep91
      Case 92
        Set cExecStep92 = New cRunStep
        Set InitExecStep = cExecStep92
      Case 93
        Set cExecStep93 = New cRunStep
        Set InitExecStep = cExecStep93
      Case 94
        Set cExecStep94 = New cRunStep
        Set InitExecStep = cExecStep94
      Case 95
        Set cExecStep95 = New cRunStep
        Set InitExecStep = cExecStep95
      Case 96
        Set cExecStep96 = New cRunStep
        Set InitExecStep = cExecStep96
      Case 97
        Set cExecStep97 = New cRunStep
        Set InitExecStep = cExecStep97
      Case 98
        Set cExecStep98 = New cRunStep
        Set InitExecStep = cExecStep98
      Case 99
        Set cExecStep99 = New cRunStep
        Set InitExecStep = cExecStep99
     Case Else
        Set InitExecStep = Nothing
   End Select

   BugMessage "Sending cExecStep" &
(lngIndex + 1) & "!"

   If Not InitExecStep Is Nothing Then
      InitExecStep.Index = lngIndex

      ' Remove this element from the
collection of free objects
      Call RemoveFreeProcess(lngIndex)
   End If

   Exit Function

InitExecStepErr:
   ' Log the error code raised by Visual
Basic
   Call LogErrors(Errors)
   Set InitExecStep = Nothing

End Function
Public Sub Run()
   ' Calls procedures to build a list of all the
steps that
   ' need to be executed and to execute them
   ' Determines whether the run has
started/terminated and
   ' raises the Run Start and Complete events.
   Dim cTempStep As cStep

   On Error GoTo RunErr

   If StringEmpty(mstrRootKey) Then
      Call ShowError(errExecuteBranchFailed)
      On Error GoTo 0
      Err.Raise vbObjectError +
errExecuteBranchFailed, mstrModuleName &
"Run", _

LoadResString(errExecuteBranchFailed)
   Else
      ' Execute the first branch
      WriteToWspLog (mintRunStart)
      RaiseEvent
RunStart(Determine64BitTime(),
mcWspLog.FileName)

      If
mcNavSteps.HasChild(StepKey:=mstrRootKe
y) Then
         Set cTempStep =
mcNavSteps.ChildStep(StepKey:=mstrRootKe
y)
         mstrCurBranchRoot =
MakeKeyValid(cTempStep.StepId,
cTempStep.StepType)

         Call
CreateDummyInstance(mstrCurBranchRoot)

         ' Run all pending steps in the branch
         If Not RunBranch(mstrCurBranchRoot)
Then
            ' Execute a new branch if there aren't
any
            ' steps to run
            Call RunNewBranch
         End If
      Else
         WriteToWspLog (mintRunComplete)
         ' No children to execute - the run is
complete
         RaiseEvent
RunComplete(Determine64BitTime())
      End If
   End If

   Exit Sub

RunErr:
   ' Log the error code raised by Visual Basic
   Call LogErrors(Errors)
   Call ShowError(errExecuteBranchFailed,
OptArgs:=mstrCurBranchRoot)
   Call ResetForm

End Sub
Private Sub RunNewBranch()
   ' We will build a tree of all instances that
occur and
   ' the count of the sub-steps that are running
will be
   ' stored at each node in the tree (maintained
internally
   ' as an array). Since there can be multiple
iterations
   ' of the top level nodes running at the same time,
we
   ' create a dummy node at the root that keeps a
record of
   ' the instances of the top level node.

   ' Determines whether the run has
started/terminated and
   ' raises the Run Start and Complete events.
   Dim cNextStep As cStep
   Dim bRunComplete As Boolean

   On Error GoTo RunNewBranchErr

   bRunComplete = False

   Do
      If StringEmpty(mstrCurBranchRoot) Then
         Exit Do
'         On Error GoTo 0
'         Err.Raise vbObjectError +
errExecuteBranchFailed, mstrSource, _
'            LoadResString(errExecuteBranchFailed)
      Else
         Set cNextStep =
mcNavSteps.NextStep(StepKey:=mstrCurBranchRo
ot)
         If cNextStep Is Nothing Then
            mstrCurBranchRoot = gstrEmptyString
            bRunComplete = True
            Exit Do
         Else
            ' Starting execution of a new branch -
initialize the
            ' module-level variable
            mstrCurBranchRoot =
MakeKeyValid(cNextStep.StepId,
cNextStep.StepType)
            Call
CreateDummyInstance(mstrCurBranchRoot)
         End If
      End If
      Debug.Print "Running new branch: " &
mstrCurBranchRoot

   ' Loop until we find a branch that has steps to
execute
   Loop While Not
RunBranch(mstrCurBranchRoot)

   If bRunComplete Then
      WriteToWspLog (mintRunComplete)
      ' Run is complete
      RaiseEvent
RunComplete(Determine64BitTime())
   End If

   Exit Sub

RunNewBranchErr:
   ' Log the error code raised by Visual Basic
   Call LogErrors(Errors)
   Call ShowError(errExecuteBranchFailed,
OptArgs:=mstrCurBranchRoot)
   On Error GoTo 0
   mstrSource = mstrModuleName &
"RunNewBranch"
   Err.Raise vbObjectError +
errExecuteBranchFailed, mstrSource, _
      LoadResString(errExecuteBranchFailed)

End Sub
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
Private Function RunBranch(strRootNode
As String) As Boolean
   ' This procedure is called to run all the
necessary steps
   ' in a branch. It can also be called when a
step terminates,
   ' in which case the terminated step is
passed in as the
   ' optional parameter. When a step
terminates, we need to
   ' either wait for some other steps to
terminate before
   ' we execute more steps or run as many
steps as necessary
   ' Returns True if there are steps currently
executing
   ' in the branch, else returns False
   Dim cRunning As cInstance

   On Error GoTo RunBranchErr

   If Not StringEmpty(strRootNode) Then
      ' Call a procedure to execute all the
enabled steps
      ' in the branch - will return the step
node that is
      ' being executed - nothing means 'No
more steps to
      ' execute in the branch'.
      Do
         Set cRunning =
RunPendingStepInBranch(strRootNode,
cRunning)

      Loop While Not cRunning Is Nothing

      RunBranch =
mcDummyRootInstance.IsRunning
   End If

   Exit Function

RunBranchErr:
   ' Log the error code raised by Visual
Basic
   Call LogErrors(Errors)
   On Error GoTo 0
   mstrSource = mstrModuleName &
"RunBranch"
   Err.Raise vbObjectError +
errExecuteBranchFailed, _
         mstrSource,
LoadResString(errExecuteBranchFailed)

End Function
Private Sub
TimeUpdateForProcess(StepRecord As
cStep, _
      ByVal InstanceId As Long, _
      Optional ByVal StartTime As Currency
= 0, _
      Optional ByVal EndTime As Currency
= 0, _
      Optional ByVal ElapsedTime As Long
= 0, _
      Optional Command As String)
   ' We do not maintain start and end
timestamps for the constraint
   ' of a step. Hence we check if the process
that just started/
   ' terminated is the worker step that is
being executed. If so,
   ' we update the start/end time and status
on the instance record.
   Dim cInstanceRec As cInstance
   Dim sItVal As String

   On Error GoTo TimeUpdateForProcessErr

   Set cInstanceRec =
mcInstances.QueryInstance(InstanceId)

   If StartTime = 0 Then
      RaiseEvent ProcessComplete(StepRecord,
EndTime, InstanceId, ElapsedTime)
   Else
      sItVal =
GetInstanceItValue(cInstanceRec)
      RaiseEvent ProcessStart(StepRecord,
Command, StartTime, InstanceId, _
            cInstanceRec.ParentInstanceId,
sItVal)
   End If

   Call
cInstanceRec.UpdateStartTime(StepRecord.Ste
pId, StartTime, EndTime, ElapsedTime)

   Exit Sub

TimeUpdateForProcessErr:
   ' Log the error code raised by Visual Basic
   Call LogErrors(Errors)
   WriteError errUpdateDisplayFailed,
mstrModuleName & "TimeUpdateForProcess"

End Sub
Private Sub
TimeStartUpdateForStep(StepRecord As
cStep, _
      ByVal InstanceId As Long, _
      ByVal StartTime As Currency)

   ' Called when a step starts execution. Checks
if this is the
   ' first enabled child of the manager step. If
so, updates
   ' the start time and status on the manager.
   ' Also raises the Step Start event for the
completed step.

   Dim cStartInst As cInstance
   Dim cInstanceRec As cInstance
   Dim LogLabels As New cVectorStr
   Dim iItIndex As Long
   Dim sLogLabel As String
   Dim sPath As String
   Dim sIt As String
   Dim sItVal As String

   On Error GoTo TimeStartUpdateForStepErr

   Set cStartInst =
mcInstances.QueryInstance(InstanceId)

   ' Determine the step path and iterator values
for the step and raise a step start event
   Set cInstanceRec = cStartInst
   Do While cInstanceRec.Key <>
mstrDummyRootKey
      If Not StringEmpty(sPath) Then
         sPath = sPath & gstrFileSeparator
      End If
      sPath = sPath & gstrSQ &
cInstanceRec.Step.StepLabel & gstrSQ
      Set cInstanceRec =
mcInstances.QueryInstance(cInstanceRec.Pare
ntInstanceId)
   Loop

   For iItIndex = cStartInst.Iterators.Count - 1 To 0
Step -1
      If Not StringEmpty(sIt) Then
         sIt = sIt & gstrFileSeparator
      End If
      sIt = sIt & gstrSQ &
cStartInst.Iterators(iItIndex).Value & gstrSQ
   Next iItIndex

   sItVal = GetInstanceItValue(cStartInst)
   RaiseEvent StepStart(StepRecord, StartTime,
InstanceId, cStartInst.ParentInstanceId, _
         sPath, sIt, sItVal)

   iItIndex = 0
   Set cInstanceRec = cStartInst
   ' Raise a StepStart event for the manager step, if
this is it's first sub-step being executed
   Do While cInstanceRec.Key <>
mstrDummyRootKey

      sLogLabel = gstrSQ &
cInstanceRec.Step.StepLabel & gstrSQ
      If iItIndex < cStartInst.Iterators.Count Then
         If cStartInst.Iterators(iItIndex).StepId =
cInstanceRec.Step.StepId Then
            sLogLabel = sLogLabel & msIt & gstrSQ
& cStartInst.Iterators(iItIndex).IteratorName &
gstrSQ & _
                  msItValue & gstrSQ &
cStartInst.Iterators(iItIndex).Value & gstrSQ
            iItIndex = iItIndex + 1
         End If
      End If
      LogLabels.Add sLogLabel

      If cInstanceRec.Key <> cStartInst.Key And
cInstanceRec.StartTime = 0 Then
         cInstanceRec.StartTime = StartTime
         cInstanceRec.Status = gintRunning
         sItVal = GetInstanceItValue(cInstanceRec)
         ' The step path and iterator values are not
needed for manager steps, since
         ' they are primarily used by the run status
form
         RaiseEvent StepStart(cInstanceRec.Step,
StartTime, cInstanceRec.InstanceId, _
               cInstanceRec.ParentInstanceId,
gstrEmptyString, gstrEmptyString, _
               sItVal)
      End If

      Set cInstanceRec =
mcInstances.QueryInstance(cInstanceRec.ParentIns
tanceId)
   Loop

   Call WriteToWspLog(mintStepStart, LogLabels,
StartTime)
   Set LogLabels = Nothing

   Exit Sub

TimeStartUpdateForStepErr:
   ' Log the error code raised by Visual Basic
   Call LogErrors(Errors)
   WriteError errUpdateDisplayFailed,
mstrModuleName & "TimeStartUpdateForStep"

End Sub
Private Sub WriteToWspLog(iLogEvent As
WspLogEvents, Optional StepDtls As cVectorStr, _
      Optional dtStamp As Currency = gdtmEmpty)
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   ' Writes to the workspace log that is
generated for the run. The last three
   ' parameters are valid only for Step Start
and Step Complete events.
   Static bError As Boolean
   Dim sLabel As String
   Dim lIndex As Long
   Dim bHdr As Boolean
   Dim cTempConn As cConnection

   On Error GoTo WriteToWspLogErr

   Select Case iLogEvent
      Case mintRunStart
         Set mcWspLog = New cFileSM
         mcWspLog.FileName =
GetDefaultDir(WspId, mcParameters) &
gstrFileSeparator & _
               Trim(Str(RunId)) &
gstrFileSeparator & "SMLog-" &
Format(Now, FMT_WSP_LOG_FILE) &
gstrLogFileSuffix
         mcWspLog.WriteLine
(JulianDateToString(Determine64BitTime()
) & " Start Run: " & vbTab & gstrSQ &
GetWorkspaceDetails(WorkspaceId:=WspId
)) & gstrSQ

         ' Write all current parameter values
to the log
         bHdr = False
         For lIndex = 0 To
mcParameters.ParameterCount - 1
            If
mcParameters(lIndex).ParameterType <>
gintParameterApplication Then
               If Not bHdr Then
                  mcWspLog.WriteField
JulianDateToString(Determine64BitTime())
& " Parameters: "
                  bHdr = True
               Else
                  mcWspLog.WriteField
vbTab & vbTab & vbTab
               End If
               mcWspLog.WriteLine vbTab &
gstrSQ &
mcParameters(lIndex).ParameterName &
gstrSQ & vbTab & vbTab & gstrSQ &
mcParameters(lIndex).ParameterValue &
gstrSQ
            End If
         Next lIndex

         ' Write all connection properties to
the log
         For lIndex = 0 To
RunConnections.Count - 1
            Set cTempConn =
RunConnections(lIndex)
            If lIndex = 0 Then
               mcWspLog.WriteField
JulianDateToString(Determine64BitTime())
& " Connections: "
            Else
               mcWspLog.WriteField vbTab
& vbTab & vbTab
            End If
            mcWspLog.WriteLine vbTab &
gstrSQ & cTempConn.ConnectionName &
gstrSQ & _
                  vbTab & vbTab & gstrSQ &
cTempConn.ConnectionValue & gstrSQ &
_
                  vbTab & "No Count: " &
gstrSQ & cTempConn.NoCountDisplay &
gstrSQ & gstrBlank & _
                  "No Execute: " & gstrSQ &
cTempConn.NoExecute & gstrSQ & gstrBlank
& _
                  "Parse Query Only: " & gstrSQ
& cTempConn.ParseQueryOnly & gstrSQ &
gstrBlank & _
                  "Quoted Identifiers: " & gstrSQ
& cTempConn.QuotedIdentifiers & gstrSQ &
gstrBlank & _
                  "ANSI Nulls: " & gstrSQ &
cTempConn.AnsiNulls & gstrSQ & gstrBlank
& _
                  "Show Query Plan: " & gstrSQ
& cTempConn.ShowQueryPlan & gstrSQ &
gstrBlank & _
                  "Show Stats Time: " & gstrSQ
& cTempConn.ShowStatsTime & gstrSQ &
gstrBlank & _
                  "Show Stats IO: " & gstrSQ &
cTempConn.ShowStatsIO & gstrSQ &
gstrBlank & _
                  "Row Count" & gstrSQ &
cTempConn.RowCount & gstrSQ & gstrBlank
& _
                  "Query Timeout" & gstrSQ &
cTempConn.QueryTimeOut & gstrSQ
         Next lIndex

      Case mintRunComplete
         BugAssert Not mcWspLog Is Nothing
         mcWspLog.WriteLine
(JulianDateToString(Determine64BitTime())
& " Comp. Run: " & vbTab & gstrSQ &
GetWorkspaceDetails(WorkspaceId:=WspId))
& gstrSQ
         Set mcWspLog = Nothing

      Case mintStepStart
         For lIndex = StepDtls.Count - 1 To 0
Step -1
            sLabel = StepDtls(lIndex)
            If lIndex = StepDtls.Count - 1 Then
               mcWspLog.WriteLine
JulianDateToString(dtStamp) & " Start Step: "
& vbTab & sLabel
            Else
               mcWspLog.WriteLine vbTab &
vbTab & vbTab & vbTab & sLabel
            End If
         Next lIndex

      Case mintStepComplete
         For lIndex = StepDtls.Count - 1 To 0
Step -1
            sLabel = StepDtls(lIndex)
            If lIndex = StepDtls.Count - 1 Then
               mcWspLog.WriteLine
JulianDateToString(dtStamp) & " Comp. Step:
" & vbTab & sLabel
            Else
               mcWspLog.WriteLine vbTab &
vbTab & vbTab & vbTab & sLabel
            End If
         Next lIndex

   End Select

   Exit Sub

WriteToWspLogErr:
   If Not bError Then
      bError = True
   End If

End Sub
'Private Sub WriteToWspLog(iLogEvent As
WspLogEvents, Optional StepDtls As cVectorStr, _
'       Optional dtStamp As Date = gdtmEmpty)
'
'   This function uses the LogWriter dll - memory
corruption problems since the vb exe
'   and the vc Execute Dll both use the same dll to
write.
'    ' Writes to the workspace log that is generated
for the run. The last three
'    ' parameters are valid only for StepStart and
StepComplete events.
'    Static bError As Boolean
'    Static sFile As String
'    Dim sLabel As String
'    Dim lIndex As Long
'    Dim bHdr As Boolean
'
'    On Error GoTo WriteToWspLogErr
'
'    Select Case iLogEvent
'       Case mintRunStart
'          Set mcWspLog = New
LOGWRITERLib.SMLog
'          sFile = App.Path & "\" & "SMLog-" &
Format(Now, FMT_WSP_LOG_FILE) &
gstrLogFileSuffix
'          mcWspLog.FileName = sFile
'          mcWspLog.Init
'          mcWspLog.WriteLine (Format(Now,
FMT_WSP_LOG_DATE) & " Start Run: " &
vbTab & gstrSQ &
GetWorkspaceDetails(WorkspaceId:=WspId)) &
gstrSQ
'
'          ' Write all current parameter values to the
log
'          bHdr = False
'          For lIndex = 0 To
mcParameters.ParameterCount - 1
'             If mcParameters(lIndex).ParameterType
<> gintParameterApplication Then
'                If Not bHdr Then
'                   'mcWspLog.WriteLine
Format(Now, FMT_WSP_LOG_DATE) & "
Parameters: " & vbTab & gstrSQ &
mcParameters(lIndex).ParameterName & gstrSQ &
vbTab & vbTab & gstrSQ &
mcParameters(lIndex).ParameterValue & gstrSQ
'                   bHdr = True
'                Else
'                   'mcWspLog.WriteLine vbTab &
vbTab & vbTab & vbTab & gstrSQ &
mcParameters(lIndex).ParameterName & gstrSQ &
vbTab & vbTab & gstrSQ &
mcParameters(lIndex).ParameterValue & gstrSQ
'                End If
'             End If
'          Next lIndex
'
'       Case mintRunComplete
'          BugAssert Not mcWspLog Is Nothing
'          mcWspLog.WriteLine (Format(Now,
FMT_WSP_LOG_DATE) & " Comp. Run: " &
vbTab & gstrSQ &
GetWorkspaceDetails(WorkspaceId:=WspId)) &
gstrSQ
'          Set mcWspLog = Nothing
'
'       Case mintStepStart
'          For lIndex = StepDtls.Count - 1 To 0 Step -
1
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
'            sLabel = StepDtls(lIndex)
'            If lIndex = StepDtls.Count - 1
Then
'                mcWspLog.WriteLine
Format(dtStamp, FMT_WSP_LOG_DATE)
& " Start Step: " & vbTab & sLabel
'            Else
'                mcWspLog.WriteLine vbTab
& vbTab & vbTab & vbTab & sLabel
'            End If
'        Next lIndex
'
'      Case mintStepComplete
'          For lIndex = StepDtls.Count - 1 To
0 Step -1
'            sLabel = StepDtls(lIndex)
'            If lIndex = StepDtls.Count - 1
Then
'                mcWspLog.WriteLine
Format(dtStamp, FMT_WSP_LOG_DATE)
& " Comp. Step: " & vbTab & sLabel
'            Else
'                mcWspLog.WriteLine vbTab
& vbTab & vbTab & vbTab & sLabel
'            End If
'        Next lIndex
'
'   End Select
'
'   Exit Sub
'
'WriteToWspLogErr:
'   If Not bError Then
'       bError = True
'   End If
'
'End Sub
'
Public Property Get WspPreExecution() As
Variant
   WspPreExecution = mcvntWspPreCons
End Property
Public Property Let
WspPreExecution(ByVal vdata As Variant)
   mcvntWspPreCons = vdata
End Property

Public Property Get WspPostExecution() As
Variant
   WspPostExecution = mcvntWspPostCons
End Property
Public Property Let
WspPostExecution(ByVal vdata As Variant)
   mcvntWspPostCons = vdata
End Property

Private Sub ExecuteStep(cCurStep As
cInstance)
   ' Initializes a cRunStep object with all the
properties
   ' corresponding to the step to be executed
and calls it's
   ' execute method to execute the step

   Dim cExecStep As cRunStep

   On Error GoTo ExecuteStepErr
   mstrSource = mstrModuleName &
"ExecuteStep"

   ' Confirm that the step is a worker
   If cCurStep.Step.StepType <>
gintWorkerStep Then
      On Error GoTo 0
      Err.Raise vbObjectError +
errExecInstanceFailed, mstrSource, _

LoadResString(errExecInstanceFailed)
   End If

   Set cExecStep = InitExecStep()
   ' Exceeded the number of processes that we
can run simultaneously
   If cExecStep Is Nothing Then
      ' Raise an error
      On Error GoTo 0
      Err.Raise vbObjectError +
errProgramError, mstrSource, _
         LoadResString(errProgramError)
   End If
   ' Initialize the instance id - not needed for
step execution
   ' but necessary to identify later which
instance completed
   cExecStep.InstanceId = cCurStep.InstanceId

   Set cExecStep.ExecuteStep = cCurStep.Step
   Set cExecStep.Iterators = cCurStep.Iterators
   Set cExecStep.Globals = mcRunSteps
   Set cExecStep.WspParameters =
mcParameters
   Set cExecStep.WspConnections =
RunConnections
   Set cExecStep.WspConnDtls =
RunConnDtls

   ' Initialize all the pre and post-execution
constraints that
   ' have been defined globally for the
workspace
   cExecStep.WspPreCons =
mcvntWspPreCons
   cExecStep.WspPostCons =
mcvntWspPostCons

   ' Initialize all the pre and post-execution
constraints for
   ' the step being executed
   cExecStep.PreCons =
DetermineConstraints(cCurStep, gintPreStep)
   cExecStep.PostCons =
DetermineConstraints(cCurStep, gintPostStep)

   cExecStep.RunId = RunId
   cExecStep.CreateInputFiles =
CreateInputFiles

   ' Call the execute method to execute the step
   cExecStep.Execute

   Set cExecStep = Nothing

   Exit Sub

ExecuteStepErr:
   ' Log the error code raised by Visual Basic
   Call LogErrors(Errors)
   On Error GoTo 0
   Call ExecutionFailed(cExecStep)

End Sub

Public Property Set Steps(cRunSteps As
cArrSteps)

   Set mcRunSteps = cRunSteps
   Set mcNavSteps.StepRecords = cRunSteps

End Property
Public Property Set Parameters(cParameters As
cArrParameters)
   ' A reference to the parameter array - we use it to
   ' substitute parameter values in the step text

   Set mcParameters = cParameters

End Property
Public Property Get Steps() As cArrSteps

   Set Steps = mcRunSteps

End Property
Public Property Get Constraints() As
cArrConstraints

   Set Constraints = mcRunConstraints

End Property
Public Property Set Constraints(vdata As
cArrConstraints)

   Set mcRunConstraints = vdata

End Property


Private Sub
cExecStep1_ProcessComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, lngInstanceId As
Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub


Private Sub cExecStep1_ProcessStart(cStepRecord
As cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub


Private Sub
cExecStep1_StepComplete(cStepRecord As cStep,
_
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord, dtmEndTime,
cExecStep1.Index, InstanceId, Status)

End Sub

Private Sub cExecStep1_StepStart(cStepRecord As
cStep, _
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
Private Sub
cExecStep9_ProcessComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency,
lngInstanceId As Long, lElapsed As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub


Private Sub
cExecStep9_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime
As Currency, lngInstanceId As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub


Private Sub
cExecStep9_StepComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, InstanceId
As Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep9.Index, InstanceId,
Status)

End Sub

Private Sub
cExecStep9_StepStart(cStepRecord As
cStep, _
      dtmStartTime As Currency, InstanceId
As Long)

   Call
TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep10_ProcessComplete(cStepRecor
d As cStep, _
      dtmEndTime As Currency,
lngInstanceId As Long, lElapsed As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub


Private Sub
cExecStep10_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime
As Currency, lngInstanceId As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub


Private Sub
cExecStep10_StepComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep10.Index, InstanceId,
Status)

End Sub

Private Sub
cExecStep10_StepStart(cStepRecord As cStep,
_
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep11_ProcessComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, lngInstanceId
As Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub


Private Sub
cExecStep11_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub


Private Sub
cExecStep11_StepComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep11.Index, InstanceId,
Status)

End Sub

Private Sub
cExecStep11_StepStart(cStepRecord As cStep,
_
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub
Private Sub
cExecStep12_ProcessComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, lngInstanceId As
Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub


Private Sub
cExecStep12_ProcessStart(cStepRecord As cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub


Private Sub
cExecStep12_StepComplete(cStepRecord As cStep,
_
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord, dtmEndTime,
cExecStep12.Index, InstanceId, Status)

End Sub

Private Sub cExecStep12_StepStart(cStepRecord
As cStep, _
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub
Private Sub
cExecStep13_ProcessComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, lngInstanceId As
Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub


Private Sub
cExecStep13_ProcessStart(cStepRecord As cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub


Private Sub
cExecStep13_StepComplete(cStepRecord As cStep,
_
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
      dtmEndTime As Currency, InstanceId
As Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep13.Index,
InstanceId, Status)

End Sub

Private Sub
cExecStep13_StepStart(cStepRecord As
cStep, _
      dtmStartTime As Currency, InstanceId
As Long)

   Call
TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub
Private Sub
cExecStep14_ProcessComplete(cStepRecor
d As cStep, _
      dtmEndTime As Currency,
lngInstanceId As Long, lElapsed As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub


Private Sub
cExecStep14_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime
As Currency, lngInstanceId As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub


Private Sub
cExecStep14_StepComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, InstanceId
As Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep14.Index,
InstanceId, Status)

End Sub

Private Sub
cExecStep14_StepStart(cStepRecord As
cStep, _
      dtmStartTime As Currency, InstanceId
As Long)

   Call
TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub
Private Sub
cExecStep15_ProcessComplete(cStepRecor
d As cStep, _
      dtmEndTime As Currency, lngInstanceId
As Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub


Private Sub
cExecStep15_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub


Private Sub
cExecStep15_StepComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep15.Index, InstanceId,
Status)

End Sub

Private Sub
cExecStep15_StepStart(cStepRecord As cStep,
_
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub
Private Sub
cExecStep16_ProcessComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, lngInstanceId
As Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub


Private Sub
cExecStep16_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub


Private Sub
cExecStep16_StepComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord, dtmEndTime,
cExecStep16.Index, InstanceId, Status)

End Sub

Private Sub cExecStep16_StepStart(cStepRecord
As cStep, _
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub
Private Sub
cExecStep17_ProcessComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, lngInstanceId As
Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub


Private Sub
cExecStep17_ProcessStart(cStepRecord As cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub


Private Sub
cExecStep17_StepComplete(cStepRecord As cStep,
_
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord, dtmEndTime,
cExecStep17.Index, InstanceId, Status)

End Sub

Private Sub cExecStep17_StepStart(cStepRecord
As cStep, _
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep18_ProcessComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, lngInstanceId As
Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
Private Sub
cExecStep18_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime
As Currency, lngInstanceId As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub


Private Sub
cExecStep18_StepComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, InstanceId
As Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep18.Index,
InstanceId, Status)

End Sub

Private Sub
cExecStep18_StepStart(cStepRecord As
cStep, _
      dtmStartTime As Currency, InstanceId
As Long)

   Call
TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep19_ProcessComplete(cStepRecor
d As cStep, _
      dtmEndTime As Currency,
lngInstanceId As Long, lElapsed As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub


Private Sub
cExecStep19_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime
As Currency, lngInstanceId As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub


Private Sub
cExecStep19_StepComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, InstanceId
As Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep19.Index, InstanceId,
Status)

End Sub

Private Sub
cExecStep19_StepStart(cStepRecord As cStep,
_
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep20_ProcessComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, lngInstanceId
As Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep20_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep20_StepComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep20.Index, InstanceId,
Status)

End Sub

Private Sub
cExecStep20_StepStart(cStepRecord As cStep,
_
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep21_ProcessComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, lngInstanceId
As Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep21_ProcessStart(cStepRecord As cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep21_StepComplete(cStepRecord As cStep,
_
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord, dtmEndTime,
cExecStep21.Index, InstanceId, Status)

End Sub

Private Sub cExecStep21_StepStart(cStepRecord
As cStep, _
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep22_ProcessComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, lngInstanceId As
Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep22_ProcessStart(cStepRecord As cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep22_StepComplete(cStepRecord As cStep,
_
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord, dtmEndTime,
cExecStep22.Index, InstanceId, Status)

End Sub

Private Sub cExecStep22_StepStart(cStepRecord
As cStep, _
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
End Sub

Private Sub
cExecStep23_ProcessComplete(cStepRecor
d As cStep, _
      dtmEndTime As Currency,
lngInstanceId As Long, lElapsed As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep23_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime
As Currency, lngInstanceId As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep23_StepComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, InstanceId
As Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep23.Index,
InstanceId, Status)

End Sub

Private Sub
cExecStep23_StepStart(cStepRecord As
cStep, _
      dtmStartTime As Currency, InstanceId
As Long)

   Call
TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep24_ProcessComplete(cStepRecor
d As cStep, _
      dtmEndTime As Currency,
lngInstanceId As Long, lElapsed As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep24_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime
As Currency, lngInstanceId As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep24_StepComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep24.Index, InstanceId,
Status)

End Sub

Private Sub
cExecStep24_StepStart(cStepRecord As cStep,
_
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep25_ProcessComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, lngInstanceId
As Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep25_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep25_StepComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep25.Index, InstanceId,
Status)

End Sub

Private Sub
cExecStep25_StepStart(cStepRecord As cStep,
_
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep26_ProcessComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, lngInstanceId As
Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep26_ProcessStart(cStepRecord As cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep26_StepComplete(cStepRecord As cStep,
_
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord, dtmEndTime,
cExecStep26.Index, InstanceId, Status)

End Sub

Private Sub cExecStep26_StepStart(cStepRecord
As cStep, _
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep27_ProcessComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, lngInstanceId As
Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep27_ProcessStart(cStepRecord As cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep27_StepComplete(cStepRecord As cStep,
_
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord, dtmEndTime,
cExecStep27.Index, InstanceId, Status)
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
End Sub

Private Sub
cExecStep27_StepStart(cStepRecord As
cStep, _
      dtmStartTime As Currency, InstanceId
As Long)

   Call
TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep28_ProcessComplete(cStepRecor
d As cStep, _
      dtmEndTime As Currency,
lngInstanceId As Long, lElapsed As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep28_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime
As Currency, lngInstanceId As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep28_StepComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, InstanceId
As Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep28.Index,
InstanceId, Status)

End Sub

Private Sub
cExecStep28_StepStart(cStepRecord As
cStep, _
      dtmStartTime As Currency, InstanceId
As Long)

   Call
TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep29_ProcessComplete(cStepRecor
d As cStep, _
      dtmEndTime As Currency,
lngInstanceId As Long, lElapsed As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep29_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep29_StepComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep29.Index, InstanceId,
Status)

End Sub

Private Sub
cExecStep29_StepStart(cStepRecord As cStep,
_
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep30_ProcessComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, lngInstanceId
As Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep30_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep30_StepComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep30.Index, InstanceId,
Status)

End Sub

Private Sub cExecStep30_StepStart(cStepRecord
As cStep, _
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep31_ProcessComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, lngInstanceId As
Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep31_ProcessStart(cStepRecord As cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep31_StepComplete(cStepRecord As cStep,
_
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord, dtmEndTime,
cExecStep31.Index, InstanceId, Status)

End Sub

Private Sub cExecStep31_StepStart(cStepRecord
As cStep, _
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep32_ProcessComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, lngInstanceId As
Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep32_ProcessStart(cStepRecord As cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
End Sub

Private Sub
cExecStep32_StepComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, InstanceId
As Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep32.Index,
InstanceId, Status)

End Sub

Private Sub
cExecStep32_StepStart(cStepRecord As
cStep, _
      dtmStartTime As Currency, InstanceId
As Long)

   Call
TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep33_ProcessComplete(cStepRecor
d As cStep, _
      dtmEndTime As Currency,
lngInstanceId As Long, lElapsed As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep33_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime
As Currency, lngInstanceId As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep33_StepComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, InstanceId
As Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep33.Index,
InstanceId, Status)

End Sub

Private Sub
cExecStep33_StepStart(cStepRecord As
cStep, _
      dtmStartTime As Currency, InstanceId
As Long)

   Call
TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep34_ProcessComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, lngInstanceId
As Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep34_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep34_StepComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep34.Index, InstanceId,
Status)

End Sub

Private Sub
cExecStep34_StepStart(cStepRecord As cStep,
_
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep35_ProcessComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, lngInstanceId
As Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep35_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep35_StepComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord, dtmEndTime,
cExecStep35.Index, InstanceId, Status)

End Sub

Private Sub cExecStep35_StepStart(cStepRecord
As cStep, _
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep36_ProcessComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, lngInstanceId As
Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep36_ProcessStart(cStepRecord As cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep36_StepComplete(cStepRecord As cStep,
_
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord, dtmEndTime,
cExecStep36.Index, InstanceId, Status)

End Sub

Private Sub cExecStep36_StepStart(cStepRecord
As cStep, _
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep37_ProcessComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, lngInstanceId As
Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
Private Sub
cExecStep37_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime
As Currency, lngInstanceId As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep37_StepComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, InstanceId
As Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep37.Index,
InstanceId, Status)

End Sub

Private Sub
cExecStep37_StepStart(cStepRecord As
cStep, _
      dtmStartTime As Currency, InstanceId
As Long)

   Call
TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep38_ProcessComplete(cStepRecor
d As cStep, _
      dtmEndTime As Currency,
lngInstanceId As Long, lElapsed As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep38_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime
As Currency, lngInstanceId As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep38_StepComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, InstanceId
As Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep38.Index,
InstanceId, Status)

End Sub

Private Sub
cExecStep38_StepStart(cStepRecord As cStep,
_
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep39_ProcessComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, lngInstanceId
As Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep39_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep39_StepComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep39.Index, InstanceId,
Status)

End Sub

Private Sub
cExecStep39_StepStart(cStepRecord As cStep,
_
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep40_ProcessComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, lngInstanceId
As Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep40_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep40_StepComplete(cStepRecord As cStep,
_
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord, dtmEndTime,
cExecStep40.Index, InstanceId, Status)

End Sub

Private Sub cExecStep40_StepStart(cStepRecord
As cStep, _
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep41_ProcessComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, lngInstanceId As
Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep41_ProcessStart(cStepRecord As cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep41_StepComplete(cStepRecord As cStep,
_
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord, dtmEndTime,
cExecStep41.Index, InstanceId, Status)

End Sub

Private Sub cExecStep41_StepStart(cStepRecord
As cStep, _
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep42_ProcessComplete(cStepRecord As
cStep, _
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
        dtmEndTime As Currency,
lngInstanceId As Long, lElapsed As Long)

    Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep42_ProcessStart(cStepRecord As
cStep, _
        strCommand As String, dtmStartTime
As Currency, lngInstanceId As Long)

    Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep42_StepComplete(cStepRecord
As cStep, _
        dtmEndTime As Currency, InstanceId
As Long, Status As InstanceStatus)

    Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep42.Index,
InstanceId, Status)

End Sub

Private Sub
cExecStep42_StepStart(cStepRecord As
cStep, _
        dtmStartTime As Currency, InstanceId
As Long)

    Call
TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep43_ProcessComplete(cStepRecor
d As cStep, _
        dtmEndTime As Currency,
lngInstanceId As Long, lElapsed As Long)

    Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep43_ProcessStart(cStepRecord As
cStep, _
        strCommand As String, dtmStartTime
As Currency, lngInstanceId As Long)

    Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep43_StepComplete(cStepRecord As
cStep, _
        dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

    Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep43.Index, InstanceId,
Status)

End Sub

Private Sub
cExecStep43_StepStart(cStepRecord As cStep,
_
        dtmStartTime As Currency, InstanceId As
Long)

    Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep44_ProcessComplete(cStepRecord
As cStep, _
        dtmEndTime As Currency, lngInstanceId
As Long, lElapsed As Long)

    Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep44_ProcessStart(cStepRecord As
cStep, _
        strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

    Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep44_StepComplete(cStepRecord As
cStep, _
        dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

    Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep44.Index, InstanceId,
Status)

End Sub

Private Sub
cExecStep44_StepStart(cStepRecord As cStep,
_
        dtmStartTime As Currency, InstanceId As
Long)

    Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep45_ProcessComplete(cStepRecord
As cStep, _
        dtmEndTime As Currency, lngInstanceId
As Long, lElapsed As Long)

    Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep45_ProcessStart(cStepRecord As cStep, _
        strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

    Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep45_StepComplete(cStepRecord As cStep,
_
        dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

    Call StepTerminated(cStepRecord, dtmEndTime,
cExecStep45.Index, InstanceId, Status)

End Sub

Private Sub cExecStep45_StepStart(cStepRecord
As cStep, _
        dtmStartTime As Currency, InstanceId As
Long)

    Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep46_ProcessComplete(cStepRecord As
cStep, _
        dtmEndTime As Currency, lngInstanceId As
Long, lElapsed As Long)

    Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep46_ProcessStart(cStepRecord As cStep, _
        strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

    Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep46_StepComplete(cStepRecord As cStep,
_
        dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

    Call StepTerminated(cStepRecord, dtmEndTime,
cExecStep46.Index, InstanceId, Status)

End Sub

Private Sub cExecStep46_StepStart(cStepRecord
As cStep, _
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
      dtmStartTime As Currency, InstanceId
As Long)

   Call
TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep47_ProcessComplete(cStepRecor
d As cStep, _
      dtmEndTime As Currency,
lngInstanceId As Long, lElapsed As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep47_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime
As Currency, lngInstanceId As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep47_StepComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, InstanceId
As Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep47.Index,
InstanceId, Status)

End Sub

Private Sub
cExecStep47_StepStart(cStepRecord As
cStep, _
      dtmStartTime As Currency, InstanceId
As Long)

   Call
TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep48_ProcessComplete(cStepRecor
d As cStep, _
      dtmEndTime As Currency,
lngInstanceId As Long, lElapsed As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep48_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep48_StepComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep48.Index, InstanceId,
Status)

End Sub

Private Sub
cExecStep48_StepStart(cStepRecord As cStep,
_
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep49_ProcessComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, lngInstanceId
As Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep49_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep49_StepComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep49.Index, InstanceId,
Status)

End Sub

Private Sub
cExecStep49_StepStart(cStepRecord As cStep,
_
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep50_ProcessComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, lngInstanceId As
Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep50_ProcessStart(cStepRecord As cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep50_StepComplete(cStepRecord As cStep,
_
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord, dtmEndTime,
cExecStep50.Index, InstanceId, Status)

End Sub

Private Sub cExecStep50_StepStart(cStepRecord
As cStep, _
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep51_ProcessComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, lngInstanceId As
Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep51_ProcessStart(cStepRecord As cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep51_StepComplete(cStepRecord As cStep,
_
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
        dtmEndTime As Currency, InstanceId
As Long, Status As InstanceStatus)

    Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep51.Index,
InstanceId, Status)

End Sub

Private Sub
cExecStep51_StepStart(cStepRecord As
cStep, _
        dtmStartTime As Currency, InstanceId
As Long)

    Call
TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep52_ProcessComplete(cStepRecor
d As cStep, _
        dtmEndTime As Currency,
lngInstanceId As Long, lElapsed As Long)

    Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep52_ProcessStart(cStepRecord As
cStep, _
        strCommand As String, dtmStartTime
As Currency, lngInstanceId As Long)

    Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep52_StepComplete(cStepRecord
As cStep, _
        dtmEndTime As Currency, InstanceId
As Long, Status As InstanceStatus)

    Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep52.Index,
InstanceId, Status)

End Sub

Private Sub
cExecStep52_StepStart(cStepRecord As
cStep, _
        dtmStartTime As Currency, InstanceId
As Long)

    Call
TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep53_ProcessComplete(cStepRecor
d As cStep, _
        dtmEndTime As Currency, lngInstanceId
As Long, lElapsed As Long)

    Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep53_ProcessStart(cStepRecord As
cStep, _
        strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

    Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep53_StepComplete(cStepRecord As
cStep, _
        dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

    Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep53.Index, InstanceId,
Status)

End Sub

Private Sub
cExecStep53_StepStart(cStepRecord As cStep,
_
        dtmStartTime As Currency, InstanceId As
Long)

    Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep54_ProcessComplete(cStepRecord
As cStep, _
        dtmEndTime As Currency, lngInstanceId
As Long, lElapsed As Long)

    Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep54_ProcessStart(cStepRecord As
cStep, _
        strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

    Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep54_StepComplete(cStepRecord As
cStep, _
        dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

    Call StepTerminated(cStepRecord, dtmEndTime,
cExecStep54.Index, InstanceId, Status)

End Sub

Private Sub cExecStep54_StepStart(cStepRecord
As cStep, _
        dtmStartTime As Currency, InstanceId As
Long)

    Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep55_ProcessComplete(cStepRecord As
cStep, _
        dtmEndTime As Currency, lngInstanceId As
Long, lElapsed As Long)

    Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep55_ProcessStart(cStepRecord As cStep, _
        strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

    Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep55_StepComplete(cStepRecord As cStep,
_
        dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

    Call StepTerminated(cStepRecord, dtmEndTime,
cExecStep55.Index, InstanceId, Status)

End Sub

Private Sub cExecStep55_StepStart(cStepRecord
As cStep, _
        dtmStartTime As Currency, InstanceId As
Long)

    Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep56_ProcessComplete(cStepRecord As
cStep, _
        dtmEndTime As Currency, lngInstanceId As
Long, lElapsed As Long)

    Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep56_ProcessStart(cStepRecord As cStep, _
        strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
    Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep56_StepComplete(cStepRecord
As cStep, _
        dtmEndTime As Currency, InstanceId
As Long, Status As InstanceStatus)

    Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep56.Index,
InstanceId, Status)

End Sub

Private Sub
cExecStep56_StepStart(cStepRecord As
cStep, _
        dtmStartTime As Currency, InstanceId
As Long)

    Call
TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep57_ProcessComplete(cStepRecor
d As cStep, _
        dtmEndTime As Currency,
lngInstanceId As Long, lElapsed As Long)

    Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep57_ProcessStart(cStepRecord As
cStep, _
        strCommand As String, dtmStartTime
As Currency, lngInstanceId As Long)

    Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep57_StepComplete(cStepRecord
As cStep, _
        dtmEndTime As Currency, InstanceId
As Long, Status As InstanceStatus)

    Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep57.Index,
InstanceId, Status)

End Sub

Private Sub
cExecStep57_StepStart(cStepRecord As
cStep, _
        dtmStartTime As Currency, InstanceId
As Long)

    Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep58_ProcessComplete(cStepRecord
As cStep, _
        dtmEndTime As Currency, lngInstanceId
As Long, lElapsed As Long)

    Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep58_ProcessStart(cStepRecord As
cStep, _
        strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

    Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep58_StepComplete(cStepRecord As
cStep, _
        dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

    Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep58.Index, InstanceId,
Status)

End Sub

Private Sub
cExecStep58_StepStart(cStepRecord As cStep,
_
        dtmStartTime As Currency, InstanceId As
Long)

    Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep59_ProcessComplete(cStepRecord
As cStep, _
        dtmEndTime As Currency, lngInstanceId
As Long, lElapsed As Long)

    Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep59_ProcessStart(cStepRecord As
cStep, _
        strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

    Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep59_StepComplete(cStepRecord As cStep,
_
        dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

    Call StepTerminated(cStepRecord, dtmEndTime,
cExecStep59.Index, InstanceId, Status)

End Sub

Private Sub cExecStep59_StepStart(cStepRecord
As cStep, _
        dtmStartTime As Currency, InstanceId As
Long)

    Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep60_ProcessComplete(cStepRecord As
cStep, _
        dtmEndTime As Currency, lngInstanceId As
Long, lElapsed As Long)

    Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep60_ProcessStart(cStepRecord As cStep, _
        strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

    Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep60_StepComplete(cStepRecord As cStep,
_
        dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

    Call StepTerminated(cStepRecord, dtmEndTime,
cExecStep60.Index, InstanceId, Status)

End Sub

Private Sub cExecStep60_StepStart(cStepRecord
As cStep, _
        dtmStartTime As Currency, InstanceId As
Long)

    Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep61_ProcessComplete(cStepRecord As
cStep, _
        dtmEndTime As Currency, lngInstanceId As
Long, lElapsed As Long)
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep61_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime
As Currency, lngInstanceId As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep61_StepComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, InstanceId
As Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep61.Index,
InstanceId, Status)

End Sub

Private Sub
cExecStep61_StepStart(cStepRecord As
cStep, _
      dtmStartTime As Currency, InstanceId
As Long)

   Call
TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep62_ProcessComplete(cStepRecor
d As cStep, _
      dtmEndTime As Currency,
lngInstanceId As Long, lElapsed As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep62_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime
As Currency, lngInstanceId As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep62_StepComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, InstanceId
As Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep62.Index, InstanceId,
Status)

End Sub

Private Sub
cExecStep62_StepStart(cStepRecord As cStep,
_
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep63_ProcessComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, lngInstanceId
As Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep63_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep63_StepComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep63.Index, InstanceId,
Status)

End Sub

Private Sub
cExecStep63_StepStart(cStepRecord As cStep,
_
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep64_ProcessComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, lngInstanceId
As Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep64_ProcessStart(cStepRecord As cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep64_StepComplete(cStepRecord As cStep,
_
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord, dtmEndTime,
cExecStep64.Index, InstanceId, Status)

End Sub

Private Sub cExecStep64_StepStart(cStepRecord
As cStep, _
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep65_ProcessComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, lngInstanceId As
Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep65_ProcessStart(cStepRecord As cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep65_StepComplete(cStepRecord As cStep,
_
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord, dtmEndTime,
cExecStep65.Index, InstanceId, Status)

End Sub

Private Sub cExecStep65_StepStart(cStepRecord
As cStep, _
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
End Sub

Private Sub
cExecStep66_ProcessComplete(cStepRecor
d As cStep, _
      dtmEndTime As Currency,
lngInstanceId As Long, lElapsed As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep66_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime
As Currency, lngInstanceId As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep66_StepComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, InstanceId
As Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep66.Index,
InstanceId, Status)

End Sub

Private Sub
cExecStep66_StepStart(cStepRecord As
cStep, _
      dtmStartTime As Currency, InstanceId
As Long)

   Call
TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep67_ProcessComplete(cStepRecor
d As cStep, _
      dtmEndTime As Currency,
lngInstanceId As Long, lElapsed As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep67_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime
As Currency, lngInstanceId As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep67_StepComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep67.Index, InstanceId,
Status)

End Sub

Private Sub
cExecStep67_StepStart(cStepRecord As cStep,
_
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep68_ProcessComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, lngInstanceId
As Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep68_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep68_StepComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep68.Index, InstanceId,
Status)

End Sub

Private Sub
cExecStep68_StepStart(cStepRecord As cStep,
_
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep69_ProcessComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, lngInstanceId As
Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep69_ProcessStart(cStepRecord As cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep69_StepComplete(cStepRecord As cStep,
_
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord, dtmEndTime,
cExecStep69.Index, InstanceId, Status)

End Sub

Private Sub cExecStep69_StepStart(cStepRecord
As cStep, _
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep70_ProcessComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, lngInstanceId As
Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep70_ProcessStart(cStepRecord As cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep70_StepComplete(cStepRecord As cStep,
_
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord, dtmEndTime,
cExecStep70.Index, InstanceId, Status)
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
End Sub

Private Sub
cExecStep70_StepStart(cStepRecord As
cStep, _
      dtmStartTime As Currency, InstanceId
As Long)

   Call
TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep71_ProcessComplete(cStepRecor
d As cStep, _
      dtmEndTime As Currency,
lngInstanceId As Long, lElapsed As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep71_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime
As Currency, lngInstanceId As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep71_StepComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, InstanceId
As Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep71.Index,
InstanceId, Status)

End Sub

Private Sub
cExecStep71_StepStart(cStepRecord As
cStep, _
      dtmStartTime As Currency, InstanceId
As Long)

   Call
TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep72_ProcessComplete(cStepRecor
d As cStep, _
      dtmEndTime As Currency,
lngInstanceId As Long, lElapsed As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep72_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep72_StepComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep72.Index, InstanceId,
Status)

End Sub

Private Sub
cExecStep72_StepStart(cStepRecord As cStep,
_
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep73_ProcessComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, lngInstanceId
As Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep73_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep73_StepComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep73.Index, InstanceId,
Status)

End Sub

Private Sub cExecStep73_StepStart(cStepRecord
As cStep, _
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep74_ProcessComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, lngInstanceId As
Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep74_ProcessStart(cStepRecord As cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep74_StepComplete(cStepRecord As cStep,
_
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord, dtmEndTime,
cExecStep74.Index, InstanceId, Status)

End Sub

Private Sub cExecStep74_StepStart(cStepRecord
As cStep, _
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep75_ProcessComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, lngInstanceId As
Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep75_ProcessStart(cStepRecord As cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
End Sub

Private Sub
cExecStep75_StepComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, InstanceId
As Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep75.Index,
InstanceId, Status)

End Sub

Private Sub
cExecStep75_StepStart(cStepRecord As
cStep, _
      dtmStartTime As Currency, InstanceId
As Long)

   Call
TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep76_ProcessComplete(cStepRecor
d As cStep, _
      dtmEndTime As Currency,
lngInstanceId As Long, lElapsed As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep76_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime
As Currency, lngInstanceId As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep76_StepComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, InstanceId
As Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep76.Index,
InstanceId, Status)

End Sub

Private Sub
cExecStep76_StepStart(cStepRecord As
cStep, _
      dtmStartTime As Currency, InstanceId
As Long)

   Call
TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep77_ProcessComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, lngInstanceId
As Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep77_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep77_StepComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep77.Index, InstanceId,
Status)

End Sub

Private Sub
cExecStep77_StepStart(cStepRecord As cStep,
_
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep78_ProcessComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, lngInstanceId
As Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep78_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep78_StepComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord, dtmEndTime,
cExecStep78.Index, InstanceId, Status)

End Sub

Private Sub cExecStep78_StepStart(cStepRecord
As cStep, _
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep79_ProcessComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, lngInstanceId As
Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep79_ProcessStart(cStepRecord As cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep79_StepComplete(cStepRecord As cStep,
_
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord, dtmEndTime,
cExecStep79.Index, InstanceId, Status)

End Sub

Private Sub cExecStep79_StepStart(cStepRecord
As cStep, _
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep80_ProcessComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, lngInstanceId As
Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
Private Sub
cExecStep80_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime
As Currency, lngInstanceId As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep80_StepComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, InstanceId
As Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep80.Index,
InstanceId, Status)

End Sub

Private Sub
cExecStep80_StepStart(cStepRecord As
cStep, _
      dtmStartTime As Currency, InstanceId
As Long)

   Call
TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep81_ProcessComplete(cStepRecor
d As cStep, _
      dtmEndTime As Currency,
lngInstanceId As Long, lElapsed As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep81_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime
As Currency, lngInstanceId As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep81_StepComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, InstanceId
As Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep81.Index,
InstanceId, Status)

End Sub

Private Sub
cExecStep81_StepStart(cStepRecord As cStep,
_
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep82_ProcessComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, lngInstanceId
As Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep82_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep82_StepComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep82.Index, InstanceId,
Status)

End Sub

Private Sub
cExecStep82_StepStart(cStepRecord As cStep,
_
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep83_ProcessComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, lngInstanceId
As Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep83_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep83_StepComplete(cStepRecord As cStep,
_
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord, dtmEndTime,
cExecStep83.Index, InstanceId, Status)

End Sub

Private Sub cExecStep83_StepStart(cStepRecord
As cStep, _
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep84_ProcessComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, lngInstanceId As
Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep84_ProcessStart(cStepRecord As cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep84_StepComplete(cStepRecord As cStep,
_
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord, dtmEndTime,
cExecStep84.Index, InstanceId, Status)

End Sub

Private Sub cExecStep84_StepStart(cStepRecord
As cStep, _
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep85_ProcessComplete(cStepRecord As
cStep, _
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
        dtmEndTime As Currency,
lngInstanceId As Long, lElapsed As Long)

    Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep85_ProcessStart(cStepRecord As
cStep, _
        strCommand As String, dtmStartTime
As Currency, lngInstanceId As Long)

    Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep85_StepComplete(cStepRecord
As cStep, _
        dtmEndTime As Currency, InstanceId
As Long, Status As InstanceStatus)

    Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep85.Index,
InstanceId, Status)

End Sub

Private Sub
cExecStep85_StepStart(cStepRecord As
cStep, _
        dtmStartTime As Currency, InstanceId
As Long)

    Call
TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep86_ProcessComplete(cStepRecor
d As cStep, _
        dtmEndTime As Currency,
lngInstanceId As Long, lElapsed As Long)

    Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep86_ProcessStart(cStepRecord As
cStep, _
        strCommand As String, dtmStartTime
As Currency, lngInstanceId As Long)

    Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep86_StepComplete(cStepRecord As
cStep, _
        dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

    Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep86.Index, InstanceId,
Status)

End Sub

Private Sub
cExecStep86_StepStart(cStepRecord As cStep,
_
        dtmStartTime As Currency, InstanceId As
Long)

    Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep87_ProcessComplete(cStepRecord
As cStep, _
        dtmEndTime As Currency, lngInstanceId
As Long, lElapsed As Long)

    Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep87_ProcessStart(cStepRecord As
cStep, _
        strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

    Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep87_StepComplete(cStepRecord As
cStep, _
        dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

    Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep87.Index, InstanceId,
Status)

End Sub

Private Sub
cExecStep87_StepStart(cStepRecord As cStep,
_
        dtmStartTime As Currency, InstanceId As
Long)

    Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep88_ProcessComplete(cStepRecord
As cStep, _
        dtmEndTime As Currency, lngInstanceId
As Long, lElapsed As Long)

    Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep88_ProcessStart(cStepRecord As cStep, _
        strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

    Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep88_StepComplete(cStepRecord As cStep,
_
        dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

    Call StepTerminated(cStepRecord, dtmEndTime,
cExecStep88.Index, InstanceId, Status)

End Sub

Private Sub cExecStep88_StepStart(cStepRecord
As cStep, _
        dtmStartTime As Currency, InstanceId As
Long)

    Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep89_ProcessComplete(cStepRecord As
cStep, _
        dtmEndTime As Currency, lngInstanceId As
Long, lElapsed As Long)

    Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep89_ProcessStart(cStepRecord As cStep, _
        strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

    Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep89_StepComplete(cStepRecord As cStep,
_
        dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

    Call StepTerminated(cStepRecord, dtmEndTime,
cExecStep89.Index, InstanceId, Status)

End Sub

Private Sub cExecStep89_StepStart(cStepRecord
As cStep, _
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
    dtmStartTime As Currency, InstanceId
As Long)

   Call
TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep90_ProcessComplete(cStepRecor
d As cStep, _
    dtmEndTime As Currency,
lngInstanceId As Long, lElapsed As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep90_ProcessStart(cStepRecord As
cStep, _
    strCommand As String, dtmStartTime
As Currency, lngInstanceId As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep90_StepComplete(cStepRecord
As cStep, _
    dtmEndTime As Currency, InstanceId
As Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep90.Index,
InstanceId, Status)

End Sub

Private Sub
cExecStep90_StepStart(cStepRecord As
cStep, _
    dtmStartTime As Currency, InstanceId
As Long)

   Call
TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep91_ProcessComplete(cStepRecor
d As cStep, _
    dtmEndTime As Currency,
lngInstanceId As Long, lElapsed As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep91_ProcessStart(cStepRecord As
cStep, _
    strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep91_StepComplete(cStepRecord As
cStep, _
    dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep91.Index, InstanceId,
Status)

End Sub

Private Sub
cExecStep91_StepStart(cStepRecord As cStep,
_
    dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep92_ProcessComplete(cStepRecord
As cStep, _
    dtmEndTime As Currency, lngInstanceId
As Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep92_ProcessStart(cStepRecord As
cStep, _
    strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep92_StepComplete(cStepRecord As
cStep, _
    dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep92.Index, InstanceId,
Status)

End Sub

Private Sub
cExecStep92_StepStart(cStepRecord As cStep,
_
    dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep93_ProcessComplete(cStepRecord As
cStep, _
    dtmEndTime As Currency, lngInstanceId As
Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep93_ProcessStart(cStepRecord As cStep, _
    strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep93_StepComplete(cStepRecord As cStep,
_
    dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord, dtmEndTime,
cExecStep93.Index, InstanceId, Status)

End Sub

Private Sub cExecStep93_StepStart(cStepRecord
As cStep, _
    dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep94_ProcessComplete(cStepRecord As
cStep, _
    dtmEndTime As Currency, lngInstanceId As
Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep94_ProcessStart(cStepRecord As cStep, _
    strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep94_StepComplete(cStepRecord As cStep,
_
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
    dtmEndTime As Currency, InstanceId
As Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep94.Index,
InstanceId, Status)

End Sub

Private Sub
cExecStep94_StepStart(cStepRecord As
cStep, _
    dtmStartTime As Currency, InstanceId
As Long)

   Call
TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep95_ProcessComplete(cStepRecor
d As cStep, _
    dtmEndTime As Currency,
lngInstanceId As Long, lElapsed As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep95_ProcessStart(cStepRecord As
cStep, _
    strCommand As String, dtmStartTime
As Currency, lngInstanceId As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep95_StepComplete(cStepRecord
As cStep, _
    dtmEndTime As Currency, InstanceId
As Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep95.Index,
InstanceId, Status)

End Sub

Private Sub
cExecStep95_StepStart(cStepRecord As
cStep, _
    dtmStartTime As Currency, InstanceId
As Long)

   Call
TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep96_ProcessComplete(cStepRecor
d As cStep, _
    dtmEndTime As Currency, lngInstanceId
As Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep96_ProcessStart(cStepRecord As
cStep, _
    strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep96_StepComplete(cStepRecord As
cStep, _
    dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep96.Index, InstanceId,
Status)

End Sub

Private Sub
cExecStep96_StepStart(cStepRecord As cStep,
_
    dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep97_ProcessComplete(cStepRecord
As cStep, _
    dtmEndTime As Currency, lngInstanceId
As Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep97_ProcessStart(cStepRecord As
cStep, _
    strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep97_StepComplete(cStepRecord As
cStep, _
    dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord, dtmEndTime,
cExecStep97.Index, InstanceId, Status)

End Sub

Private Sub cExecStep97_StepStart(cStepRecord
As cStep, _
    dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep98_ProcessComplete(cStepRecord As
cStep, _
    dtmEndTime As Currency, lngInstanceId As
Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep98_ProcessStart(cStepRecord As cStep, _
    strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep98_StepComplete(cStepRecord As cStep,
_
    dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord, dtmEndTime,
cExecStep98.Index, InstanceId, Status)

End Sub

Private Sub cExecStep98_StepStart(cStepRecord
As cStep, _
    dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep99_ProcessComplete(cStepRecord As
cStep, _
    dtmEndTime As Currency, lngInstanceId As
Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep99_ProcessStart(cStepRecord As cStep, _
    strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
    Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep99_StepComplete(cStepRecord
As cStep, _
        dtmEndTime As Currency, InstanceId
As Long, Status As InstanceStatus)

    Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep99.Index,
InstanceId, Status)

End Sub

Private Sub
cExecStep99_StepStart(cStepRecord As
cStep, _
        dtmStartTime As Currency, InstanceId
As Long)

    Call
TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep2_ProcessComplete(cStepRecord
As cStep, _
        dtmEndTime As Currency,
lngInstanceId As Long, lElapsed As Long)

    Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep2_ProcessStart(cStepRecord As
cStep, _
        strCommand As String, dtmStartTime
As Currency, lngInstanceId As Long)

    Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub


Private Sub
cExecStep2_StepComplete(cStepRecord As
cStep, _
        dtmEndTime As Currency, InstanceId
As Long, Status As InstanceStatus)

    Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep2.Index, _
            InstanceId, Status)

End Sub

Private Sub
cExecStep2_StepStart(cStepRecord As
cStep, _
        dtmStartTime As Currency, InstanceId As
Long)

    Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub


Private Sub
cExecStep3_ProcessComplete(cStepRecord As
cStep, _
        dtmEndTime As Currency, lngInstanceId
As Long, lElapsed As Long)

    Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep3_ProcessStart(cStepRecord As
cStep, _
        strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

    Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub


Private Sub
cExecStep3_StepComplete(cStepRecord As
cStep, _
        dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

    Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep3.Index, _
            InstanceId, Status)

End Sub

Private Sub
cExecStep3_StepStart(cStepRecord As cStep,
_
        dtmStartTime As Currency, InstanceId As
Long)

    Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub


Private Sub
cExecStep4_ProcessComplete(cStepRecord As
cStep, _
        dtmEndTime As Currency, lngInstanceId
As Long, lElapsed As Long)

    Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep4_ProcessStart(cStepRecord As
cStep, _
        strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

    Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep4_StepComplete(cStepRecord As cStep,
_
        dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

    Call StepTerminated(cStepRecord, dtmEndTime,
cExecStep4.Index, _
            InstanceId, Status)

End Sub


Private Sub cExecStep4_StepStart(cStepRecord As
cStep, _
        dtmStartTime As Currency, InstanceId As
Long)

    Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep5_ProcessComplete(cStepRecord As
cStep, _
        dtmEndTime As Currency, lngInstanceId As
Long, lElapsed As Long)

    Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub cExecStep5_ProcessStart(cStepRecord
As cStep, _
        strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

    Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep5_StepComplete(cStepRecord As cStep,
_
        dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

    Call StepTerminated(cStepRecord, dtmEndTime,
cExecStep5.Index, _
            InstanceId, Status)

End Sub

Private Sub cExecStep5_StepStart(cStepRecord As
cStep, _
        dtmStartTime As Currency, InstanceId As
Long)

    Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
Private Sub
cExecStep6_ProcessComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency,
lngInstanceId As Long, lElapsed As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep6_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime
As Currency, lngInstanceId As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep6_StepComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, InstanceId
As Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep6.Index, _
         InstanceId, Status)

End Sub

Private Sub
cExecStep6_StepStart(cStepRecord As
cStep, _
      dtmStartTime As Currency, InstanceId
As Long)

   Call
TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep7_ProcessComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency,
lngInstanceId As Long, lElapsed As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep7_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime
As Currency, lngInstanceId As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep7_StepComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep7.Index, _
         InstanceId, Status)

End Sub

Private Sub
cExecStep7_StepStart(cStepRecord As cStep,
_
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep8_ProcessComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, lngInstanceId
As Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep8_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep8_StepComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep8.Index, _
         InstanceId, Status)

End Sub

Private Sub
cExecStep8_StepStart(cStepRecord As cStep,
_
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub Class_Initialize()

   Dim lngCount As Long
   Dim lngTemp As Long

   On Error GoTo InitializeErr

   Set mcFreeSteps = New cVectorLng
   ' Initialize the array of free objects with all
elements
   ' for now
   For lngCount = 0 To
glngNumConcurrentProcesses - 1 Step 1
      mcFreeSteps.Add lngCount
   Next lngCount

   ' Initialize a byte array with the number of free
processes. It will
   ' be used later to determine if any step is running
   ' Each element in the array can represent 8 steps,
1 for each bit
   ReDim mbarrFree(glngNumConcurrentProcesses
\ gintBitsPerByte)

   ' Initialize each element in the byte array w/ all
1's
   ' (upto glngNumConcurrentProcesses)
   For lngCount = LBound(mbarrFree) To
UBound(mbarrFree) Step 1
      lngTemp = IIf( _
            glngNumConcurrentProcesses -
(gintBitsPerByte * lngCount) > gintBitsPerByte, _
            gintBitsPerByte, _
            glngNumConcurrentProcesses -
(gintBitsPerByte * lngCount))

      mbarrFree(lngCount) = (2 ^ lngTemp) - 1
   Next lngCount

   Set mcInstances = New cInstances
   Set mcFailures = New cFailedSteps
   Set mcNavSteps = New cStepTree
   Set mcTermSteps = New cTermSteps

   ' Initialize the Abort flag to False
   mblnAbort = False
   mblnAsk = False

   Exit Sub

InitializeErr:
   ' Log the error code raised by Visual Basic
   Call LogErrors(Errors)
   On Error GoTo 0
   Err.Raise vbObjectError + errInitializeFailed,
mstrModuleName & "Initialize", _
         LoadResString(errInitializeFailed)

End Sub
Private Sub Class_Terminate()

   On Error GoTo Class_TerminateErr

   mcFreeSteps.Clear
   Set mcFreeSteps = Nothing
   ReDim mbarrFree(0)

   mcInstances.Clear
   Set mcInstances = Nothing

   Set mcFailures = Nothing
   Set mcNavSteps = Nothing
   Set mcTermSteps = Nothing

   Exit Sub

Class_TerminateErr:
   Call LogErrors(Errors)
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
End Sub

Private Sub
mcTermSteps_TermStepExists(cStepDetails
As cTermStep)

   Call
RunNextStep(cStepDetails.TimeComplete,
cStepDetails.Index, _
         cStepDetails.InstanceId,
cStepDetails.ExecutionStatus)

End Sub

VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cRunItDetails"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:       cRunItDetails.cls
'             Microsoft TPC-H Kit Ver. 1.00
'             Copyright Microsoft, 1999
'             All Rights Reserved
'
'
'  PURPOSE:    This module encapsulates
the properties of iterator values
'             that are used by the step being
executed at runtime.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'

Option Explicit

' Used to indicate the source module name
when errors
' are raised by this class
Private Const mstrModuleName As String =
"cRunItDetails."
Private mstrSource As String

Private mstrIteratorName As String
Private mintType As ValueType
Private mlngSequence As Long
Private mlngFrom As Long
Private mlngTo As Long
Private mlngStep As Long
Private mstrValue As String

Public Property Get RangeTo() As Long

   RangeTo = mlngTo

End Property
Public Property Let RangeTo(ByVal vdata
As Long)

   mlngTo = vdata

End Property

Public Property Get RangeFrom() As Long

   RangeFrom = mlngFrom

End Property
Public Property Get Sequence() As Long

   Sequence = mlngSequence

End Property

Public Property Get RangeStep() As Long

   RangeStep = mlngStep

End Property
Public Property Let RangeStep(vdata As Long)

   mlngStep = vdata

End Property

Public Property Let RangeFrom(ByVal vdata
As Long)

   mlngFrom = vdata

End Property
Public Property Let Sequence(ByVal vdata As
Long)

   mlngSequence = vdata

End Property

Public Property Get IteratorType() As
ValueType

   IteratorType = mintType

End Property
Public Property Let IteratorType(ByVal vdata
As ValueType)

   On Error GoTo TypeErr
   mstrSource = mstrModuleName & "Type"

   ' These constants have been defined in the
enumeration,
   ' Type, which is exposed
   Select Case vdata
      Case gintFrom, gintTo, gintStep,
gintValue
         mintType = vdata

      Case Else
         On Error GoTo 0
         Err.Raise vbObjectError +
errTypeInvalid, _
            mstrSource,
LoadResString(errTypeInvalid)
   End Select

   Exit Property

TypeErr:
   LogErrors Errors
   mstrSource = mstrModuleName & "Type"
   On Error GoTo 0
   Err.Raise vbObjectError + errTypeInvalid, _
      mstrSource,
LoadResString(errTypeInvalid)

End Property
Private Sub IsList()

   If mintType <> gintValue Then
      On Error GoTo 0
      Err.Raise vbObjectError +
errInvalidProperty, mstrSource, _
            LoadResString(errInvalidProperty)
   End If

End Sub

Private Sub IsRange()

   If mintType = gintValue Then
      On Error GoTo 0
      Err.Raise vbObjectError + errInvalidProperty,
mstrSource, _
            LoadResString(errInvalidProperty)
   End If

End Sub

Public Property Get Value() As String

   Value = mstrValue

End Property
Public Property Let Value(vdata As String)

   mstrValue = vdata

End Property


Public Property Get IteratorName() As String

   IteratorName = mstrIteratorName

End Property
Public Property Let IteratorName(ByVal vdata As
String)

   mstrIteratorName = vdata

End Property

VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cRunItNode"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
' An iterator class containing the properties that are
used
' by the stpe being executed.
' These iterators might actually come from steps that
are at
' a higher level than the step actually being executed
(viz.
' direct ascendants of the step at any level).

Option Explicit

Public IteratorName As String
Public Value As String
Public StepId As Long

VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cRunOnly"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
Option Explicit

Public Event Done()
Private WithEvents mcRunWsp As
cRunWorkspace
Attribute mcRunWsp.VB_VarHelpID = -1
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
Public WspName As String
Public WorkspaceId As Long
Public WspLog As String

Public Sub RunWsp()

   On Error GoTo RunWspErr

   Set mcRunWsp = New cRunWorkspace
   Set mcRunWsp.LoadDb = dbsAttTool
   mcRunWsp.WorkspaceId = WorkspaceId
   mcRunWsp.CreateInputFiles = True
   mcRunWsp.RunWorkspace

   Exit Sub

RunWspErr:
   ' Log the VB error code
   LogErrors Errors

End Sub

Private Sub
mcRunWsp_RunComplete(dtmEndTime As
Currency)

   MsgBox "Completed executing
workspace: " & gstrSQ & WspName &
gstrSQ & " at " & _
         JulianDateToString(dtmEndTime) &
"." & vbCrLf & vbCrLf & _
         "The log file for the run is: " &
gstrSQ & WspLog & gstrSQ & "."
   RaiseEvent Done

End Sub

Private Sub
mcRunWsp_RunStart(dtmStartTime As
Currency, strWspLog As String, lRunId As
Long)
   WspLog = strWspLog
End Sub


VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cRunStep"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:       cRunStep.cls
'             Microsoft TPC-H Kit Ver. 1.00
'             Copyright Microsoft, 1999
'             All Rights Reserved
'
'
'  PURPOSE:    This class executes the step
that is assigned to the
'             ExecuteStep property. It executes
the pre-execution constraints
'             in sequence and then the step
itself. At the end it executes
'             the post-execution constraints.
Since these steps should always
'             be executed in sequence, each step
is only fired on the
'             completion of the previous step.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

' Used to indicate the source module name
when errors
' are raised by this class
Private Const mstrModuleName As String =
"cRunStep."
Private mstrSource As String

' Local variable(s) to hold property value(s)
Private mcStep As cStep
Private mcGlobals As cArrSteps
Private mcvntWspPreCons As Variant
Private mcvntWspPostCons As Variant
Private mcvntPreCons As Variant
Private mcvntPostCons As Variant
Private mcIterators As cRunColIt
Private mlngInstanceId As Long ' Identifier for
the current instance
Private mlngIndex As Long ' Index value for
the current instance
Private mstrCommand As String ' The
command string
Private msRunStepDtl As String ' Step text/file
name that will go into the run_step_details
table
Private mblnAbort As Boolean   ' Set to True
when the user aborts the run
Private msOutputFile As String
Private msErrorFile As String
Private miStatus As InstanceStatus
Private mcVBErr As cVBErrorsSM
Public WspParameters As cArrParameters
Public WspConnections As cConnections
Public WspConnDtls As cConnDtls

Private WithEvents mcTermProcess As
cTermProcess
Attribute mcTermProcess.VB_VarHelpID = -1
Public RunId As Long
Public CreateInputFiles As Boolean
Private msOutputDir As String

' Object that will execute the step
Private WithEvents mcExecObj As
EXECUTEDLLLib.Execute
Attribute mcExecObj.VB_VarHelpID = -1

' Holds the step that is currently being executed
(constraint or
' worker step)
Private mcExecStep As cStep

Private Const msCompareExe As String =
"\diff.exe"

Private Enum NextNodeType
   mintWspPreConstraint = 1
   mintPreConstraint
   mintStep
   mintWspPostConstraint
   mintPostConstraint
End Enum

' Public events to notify the calling function of
the
' start and end time for each step
Public Event StepStart(cStepRecord As cStep,
_
      dtmStartTime As Currency, InstanceId As
Long)
Public Event StepComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)
Public Event ProcessStart(cStepRecord As cStep, _
      strCommand As String, dtmStartTime As
Currency, _
      InstanceId As Long)
Public Event ProcessComplete(cStepRecord As
cStep, _
      dtmStartTime As Currency, InstanceId As
Long, lElapsed As Long)

Private Function AppendDiffErrors(sDiffFile As
String)
   ' The file containing the errors generated by the
diff utility is passed in
   ' These errors are appended to the error file for
the step

   Dim sTemp As String
   Dim InputFile As Integer

   If Not StringEmpty(sDiffFile) Then

      InputFile = FreeFile
      Open sDiffFile For Input Access Read As
InputFile

      Do While Not EOF(InputFile)        ' Loop
until end of file.
         Line Input #InputFile, sTemp    ' Read line
into variable.
         mcVBErr.LogMessage sTemp
      Loop

      Close InputFile
   End If

End Function

Private Sub CreateStepTextFile()
   ' Creates a file containing the step text being
executed
   On Error GoTo CreateStepTextFileErr

   Dim sInputFile As String

   If mcExecStep.ExecutionMechanism =
gintExecuteShell Then
      sInputFile = GetOutputFile(gsCmdFileSuffix)
   Else
      sInputFile = GetOutputFile(gsSqlFileSuffix)
   End If

   ' Generate a file containing the step text being
executed
   If Not StringEmpty(mcExecStep.StepTextFile)
Or mcExecStep.ExecutionMechanism =
gintExecuteShell Then
      FileCopy mstrCommand, sInputFile
   Else
      Call WriteCommandToFile(mstrCommand,
sInputFile)
   End If

   Exit Sub

CreateStepTextFileErr:

   mcVBErr.LogVBErrors

End Sub
Private Function GetOutputFile(strFileExt As
String) As String
   ' This function generates the output file name for
the step currently being executed
   ' The value of the built-in parameter 'DefaultDir'
is appended with the run identifier
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
    ' for the file location
    ' The step label is used for the file name
and a combination of all iterator values
    ' for the step is used to make the output
files unique for each instance
    Dim sFile As String
    Dim sIt As String
    Dim lIt As Long

    On Error GoTo GetOutputFileErr

    sFile =
SubstituteParametersIfPossible(mcExecStep
.StepLabel)

    sFile = TranslateStepLabel(sFile)

    If mcExecStep Is mcStep Then
        ' Use iterators that have been defined
for the worker or any of it's managers
        ' to make the error/log file unique for
this instance
        For lIt = mcIterators.Count - 1 To 0
Step -1
            sIt = sIt & gsExtSeparator &
mcIterators(lIt).Value
        Next lIt
    End If
    sIt = sIt & strFileExt

    ' Ensure that the length of the complete
path does not exceed 255 characters
    If Len(msOutputDir) + Len(sFile) +
Len(sIt) > MAX_PATH Then
        sFile = Mid(sFile, 1, MAX_PATH -
Len(sIt) - Len(msOutputDir))
    End If
    GetOutputFile = msOutputDir & sFile &
sIt
    Exit Function

GetOutputFileErr:

    ' Does not make sense to log error to the
error file yet. Write to the project
    ' log and return the step label as default
    GetOutputFile = mcExecStep.StepLabel
& gsExtSeparator & strFileExt

End Function


Private Sub HandleExecutionError()

    On Error GoTo HandleExecutionError

    ' Log the error code raised by Visual
Basic
    miStatus = gintFailed
    mcVBErr.LogVBErrors
    Call
mcVBErr.WriteError(errExecuteStepFailed,
_
            OptArgs:="Continuation criteria for
the step is: " &
gsContCriteria(mcStep.ContinuationCriteria
))

HandleExecutionError:

    ' Logging failed - return

End Sub

Public Property Get Index() As Long

    Index = mlngIndex

End Property
Public Property Let Index(ByVal vdata As
Long)

    mlngIndex = vdata

End Property
Private Function InitializeExecStatus() As
InstanceStatus
    Dim sCompareFile As String

    On Error GoTo InitializeExecStatusErr

    InitializeExecStatus =
mcExecObj.StepStatus

    If InitializeExecStatus = gintComplete Then
        If Not
StringEmpty(mcExecStep.FailureDetails) Then
            ' Compare output to determine whether
the step failed
            sCompareFile =
GetShortName(SubstituteParameters( _
                mcExecStep.FailureDetails,
mcExecStep.WorkspaceId, mcIterators, _
                WspParameters))
            InitializeExecStatus =
IIf(CompareOutput(sCompareFile,
msOutputFile), gintComplete, gintFailed)
        End If
    End If

    Exit Function

InitializeExecStatusErr:
    mcVBErr.LogVBErrors
'   Call LogErrors(Errors)
    InitializeExecStatus =
mcExecObj.StepStatus

End Function
Private Function
CompareOutput(sCompareFile As String,
sOutputFile As String) As Boolean

    Dim sCmpOutput As String
    Dim sDiffOutput As String

    On Error GoTo CompareOutputErr

    ' Create temporary files to store the file
compare output and
    ' the errors generated by the compare
function
    sCmpOutput = CreateTempFile()
    sDiffOutput = CreateTempFile()

    ' Run the compare utility and redirect it's
output and errors
    SyncShell ("cmd /c " & _
            GetShortName(App.Path &
msCompareExe) & gstrBlank & _
            sCompareFile & gstrBlank &
sOutputFile & _
            " > " & sCmpOutput & " 2> " &
sDiffOutput)

    If FileLen(sDiffOutput) > 0 Then
        ' The compare generated errors - append
error msgs to the error file
        Call AppendDiffErrors(sDiffOutput)
        CompareOutput = False
    Else
        CompareOutput = (FileLen(sCmpOutput) = 0)
    End If

    If Not CompareOutput Then
        mcVBErr.WriteError errDiffFailed
    End If

    ' Delete the temporary files used to store the
output of the compare and
    ' the errors generated by the compare
    Kill sDiffOutput
    Kill sCmpOutput

    Exit Function

CompareOutputErr:
    mcVBErr.LogVBErrors
    CompareOutput = False

End Function
Public Property Get InstanceId() As Long

    InstanceId = mlngInstanceId

End Property
Public Property Let InstanceId(ByVal vdata As
Long)

    mlngInstanceId = vdata

End Property

Private Function ExecuteConstraint(vntConstraints
As Variant, _
        ByRef intLoopIndex As Integer) As Boolean

    ' Returns True if there is a constraint in the
passed in
    ' array that remains to be executed

    If IsArray(vntConstraints) And Not
IsEmpty(vntConstraints) Then
        ExecuteConstraint = (LBound(vntConstraints)
<= intLoopIndex) And (intLoopIndex <=
UBound(vntConstraints))
    Else
        ExecuteConstraint = False
    End If

End Function
Private Function NextStep() As cStep

    ' Determines which is the next step to be executed
- it could
    ' be either a pre-execution step, the worker step
itself
    ' or a post-execution step

    Dim cConsRec As cConstraint
    Dim cNextStepRec As cStep
    Dim vntStepConstraints As Variant

    ' Static variable to remember exactly where we
are in the
    ' processing
    Static intIndex As Integer
    Static intNextStepType As NextNodeType

    On Error GoTo NextStepErr

    If mblnAbort = True Then
        ' The user has aborted the run - do not run any
more
        ' processes for the step
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
      Set NextStep = Nothing
      Exit Function
   End If

   If intNextStepType = 0 Then
      ' First time through this function - set the Index and
      ' node type to initial values
      intNextStepType = mintWspPreConstraint
      intIndex = 0
      RaiseEvent StepStart(mcStep, Determine64BitTime(), mlngInstanceId)
   End If

   Do
      Select Case intNextStepType
         Case mintWspPreConstraint
            vntStepConstraints = mcvntWspPreCons

         Case mintPreConstraint
            vntStepConstraints = mcvntPreCons

         Case mintStep
            ' CONS:
            If mcStep.StepType = gintWorkerStep Then
               Set cNextStepRec = mcStep
            End If

         Case mintWspPostConstraint
            vntStepConstraints = mcvntWspPostCons

         Case mintPostConstraint
            vntStepConstraints = mcvntPostCons

      End Select

      If intNextStepType <> mintStep Then
         ' Check if there is a constraint to be executed
         If ExecuteConstraint(vntStepConstraints, intIndex) Then
            ' Get the corresponding step record to be executed
            ' Query the global step record for the current
            ' constraint
            Set cConsRec = vntStepConstraints(intIndex)

            Set cNextStepRec = mcGlobals.QueryStep(cConsRec.GlobalStepId)
            intIndex = intIndex + 1
         Else
            If intNextStepType = mintPostConstraint Then
               ' No more stuff to be executed for the step
               ' Raise a Done event
               Set cNextStepRec = Nothing

               ' Set the next step type to an invalid value
               intNextStepType = -1
            Else
               Call NextType(intNextStepType, intIndex)
            End If
         End If
      Else
         ' Increment the step type so we look at the post-
         ' execution steps the next time through
         Call NextType(intNextStepType, intIndex)
      End If

   Loop Until (Not cNextStepRec Is Nothing) Or _
         intNextStepType = -1

   Set NextStep = cNextStepRec

   Exit Function

NextStepErr:
   ' Log the error code raised by Visual Basic
   Call LogErrors(Errors)
   On Error GoTo 0
   mstrSource = mstrModuleName & "NextStep"
   Err.Raise vbObjectError + errNextStepFailed, mstrSource, _
      LoadResString(errNextStepFailed)

End Function
Public Sub Execute()
   ' This procedure is the method that executes the step that
   ' is assigned to the ExecuteStep property. It call a procedure
   ' to determine the next step to be executed.
   ' Then it initializes all the properties of the cExecuteSM object
   ' and calls it's run method to execute it.
   Dim cConn As cConnection
   Dim cRunConnDtl As cConnDtl

   On Error GoTo ExecuteErr

   ' If this procedure is called after a step has completed,
   ' we would have to check if we created any temporary files
   ' while executing that step
   If Not mcExecStep Is Nothing Then
      If Not StringEmpty(mcExecStep.StepTextFile) Or mcExecStep.ExecutionMechanism = gintExecuteShell Then
         ' Remove the temporary file that we created while
         ' running this command
         Kill mstrCommand
      End If

      Call StepCompleted

      ' The VB errors class stores a reference to the Execute class since it uses
      ' a method of the class to write errors to the error log. Hence,
      ' release all references to the Execute object before destroying it.
      Set mcVBErr.ErrorFile = Nothing
      Set mcExecObj = Nothing

      ' Delete empty output and error files (generated by shell commands)
      ' (Can be done only after cleaning up cExecObj)
      Call DeleteEmptyOutputFiles
   Else
      ' First time through - initialize the location of output files
      msOutputDir = GetDefaultDir(mcStep.WorkspaceId, WspParameters)
      msOutputDir = msOutputDir & gstrFileSeparator & Trim(Str(RunId)) & gstrFileSeparator
      ' Dummy file since the function expects a file name
      MakePathValid (msOutputDir & "a.txt")
   End If

   ' Call a procedure to determine the next step to be executed
   ' - could be a constraint or the step itself
   ' Initialize a module-level variable to the step being
   ' executed
   Set mcExecStep = NextStep
   If mcExecStep Is Nothing Then
      RaiseEvent StepComplete(mcStep, Determine64BitTime(), mlngInstanceId, miStatus)
      ' No more stuff to execute
      Exit Sub
   End If

   Dim sStartDir As String

   Set mcExecObj = New EXECUTEDLLLib.Execute

   ' The VB errors class uses the WriteError method of the Execute class to write
   ' all VB errors to the error file for the step (this prevents a clash when the
   ' VB errors and Execution errors have to be written to the same log). Hence, store
   ' a reference to the Execute object in mcVBErr
   msErrorFile = GetOutputFile(gsErrorFileSuffix)
   mcExecObj.ErrorFile = msErrorFile
   Call DeleteFile(msErrorFile, bCheckIfEmpty:=False)
   Set mcVBErr.ErrorFile = mcExecObj

   If mcExecStep.ExecutionMechanism = gintExecuteShell Then
      sStartDir = Trim$(GetShortName(SubstituteParameters( _
            mcExecStep.StartDir, mcExecStep.WorkspaceId, mcIterators, WspParameters:=WspParameters)))
      ' Dummy connection object
      Set cConn = New cConnection
      Set cRunConnDtl = New cConnDtl
   Else
      ' Find the connection string value and substitute parameter values in it
      Set cRunConnDtl = WspConnDtls.GetConnectionDtl(mcExecStep.WorkspaceId, mcExecStep.StartDir)
      Set cConn = WspConnections.GetConnection(mcExecStep.WorkspaceId, cRunConnDtl.ConnectionString)
      sStartDir = Trim$(SubstituteParameters(cConn.ConnectionValue, _
            mcExecStep.WorkspaceId, mcIterators, WspParameters:=WspParameters))
   End If

   msOutputFile = GetOutputFile(gsOutputFileSuffix)
   Call DeleteFile(msOutputFile, bCheckIfEmpty:=False)
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   mcExecObj.OutputFile = msOutputFile
'   mcExecObj.LogFile =
GetShortName(SubstituteParameters( _
'          mcExecStep.LogFile,
mcExecStep.WorkspaceId, mcIterators,
WspParameters:=WspParameters))
   If mcExecStep.ExecutionMechanism =
gintExecuteODBC And _
         cRunConnDtl.ConnType =
ConnTypeDynamic Then
      Call
mcExecObj.DoExecute(BuildCommandStri
ng(), sStartDir,
mcExecStep.ExecutionMechanism, _
            cConn.NoCountDisplay,
cConn.NoExecute, cConn.ParseQueryOnly,
cConn.QuotedIdentifiers, _
            cConn.AnsiNulls,
cConn.ShowQueryPlan,
cConn.ShowStatsTime,
cConn.ShowStatsIO, _
            cConn.RowCount,
cConn.QueryTimeOut, gstrEmptyString)
   Else
      Call
mcExecObj.DoExecute(BuildCommandStri
ng(), sStartDir,
mcExecStep.ExecutionMechanism, _
            cConn.NoCountDisplay,
cConn.NoExecute, cConn.ParseQueryOnly,
cConn.QuotedIdentifiers, _
            cConn.AnsiNulls,
cConn.ShowQueryPlan,
cConn.ShowStatsTime,
cConn.ShowStatsIO, _
            cConn.RowCount,
cConn.QueryTimeOut,
mcExecStep.StartDir)
   End If

   Exit Sub

ExecuteErr:
   Call HandleExecutionError

   ' We can assume that if we are in this
function, a StepStart event has been
triggered already.
   RaiseEvent StepComplete(mcStep,
Determine64BitTime(), mlngInstanceId,
miStatus)

End Sub
Private Function BuildCommandString() As
String
   ' Process text to be executed - either from
the text
   ' field or read it from a file.
   ' This function will always return the
command text for ODBC commands
   ' and a file name for Shell commands
   Dim sFile As String
   Dim sCommand As String
   Dim sTemp As String

   On Error GoTo BuildCommandStringErr

   If Not
StringEmpty(mcExecStep.StepTextFile)
Then
      ' Substitute parameter values and
environment variables
      ' in the filename
      msRunStepDtl =
SubstituteParameters(mcExecStep.StepTextFil
e, _
            mcExecStep.WorkspaceId,
mcIterators, WspParameters:=WspParameters)

      sFile = GetShortName(msRunStepDtl)

      mstrCommand =
SubstituteParametersInText(sFile,
mcExecStep.WorkspaceId)

      If mcExecStep.ExecutionMechanism =
gintExecuteODBC Then
         ' Read the contents of the file and pass
it to ODBC
         BuildCommandString =
ReadCommandFromFile(mstrCommand)
      Else
         BuildCommandString = mstrCommand
      End If
   Else
      ' Substitute parameter values and
environment variables
      ' in the step text
      msRunStepDtl =
SubstituteParameters(mcExecStep.StepText, _
            mcExecStep.WorkspaceId,
mcIterators, WspParameters:=WspParameters)
      mstrCommand = msRunStepDtl

      If mcExecStep.ExecutionMechanism =
gintExecuteShell Then
         ' Write the command to a temp file
(enables us to execute multiple
         ' commands via the command
interpreter)
         mstrCommand =
WriteCommandToFile(msRunStepDtl)
         BuildCommandString = mstrCommand
      Else
         BuildCommandString =
SQLFixup(msRunStepDtl)
      End If
   End If

   If CreateInputFiles Then
      Call CreateStepTextFile
   End If

   Exit Function

BuildCommandStringErr:
   ' Log the error code raised by the Execute
procedure
'   Call LogErrors(Errors)
   mcVBErr.LogVBErrors

   On Error GoTo 0
   mstrSource = mstrModuleName &
"Execute"
   Err.Raise vbObjectError +
errExecuteStepFailed, mstrSource, _
      LoadResString(errExecuteStepFailed) &
mstrCommand

End Function
Public Sub Abort()

   On Error GoTo AbortErr

   ' Setting the Abort flag to True will ensure
that we
   ' don't execute any more processes for this
step
   mblnAbort = True

   If Not mcExecObj Is Nothing Then
      mcExecObj.Abort
   Else
      ' We are not in the middle of execution yet
   End If

   Exit Sub

AbortErr:
   Call LogErrors(Errors)
   On Error GoTo 0
   Err.Raise vbObjectError + errProgramError, _
      mstrModuleName & "Abort", _
      LoadResString(errProgramError)

End Sub
Private Sub NextType(ByRef StepType As
NextNodeType, _
      ByRef Position As Integer)

   StepType = StepType + 1
   Position = 0

End Sub
Private Sub StepCompleted()

   On Error GoTo StepCompletedErr

   If Not mcExecStep Is Nothing Then
      If mcExecStep Is mcStep Then
         miStatus = InitializeExecStatus
         If miStatus = gintFailed Then
            ' Create input files if the step failed
execution and one hasn't been created already
            If Not CreateInputFiles Then
CreateStepTextFile
            Call
mcVBErr.WriteError(errExecuteStepFailed, _
                  OptArgs:="Continuation criteria for
the step is: " &
gsContCriteria(mcStep.ContinuationCriteria))
         End If
      End If
   End If

   Exit Sub

StepCompletedErr:
   ' Log the error code raised by Visual Basic
   miStatus = gintFailed
   mcVBErr.LogVBErrors
   Call mcVBErr.WriteError(errExecuteStepFailed,
_
         OptArgs:="Continuation criteria for the step
is: " &
gsContCriteria(mcStep.ContinuationCriteria))

End Sub
Private Sub DeleteEmptyOutputFiles()

   On Error GoTo DeleteEmptyOutputFilesErr

   ' Delete empty output and error files
   If Not mcExecStep Is Nothing Then
      Call DeleteFile(msErrorFile,
bCheckIfEmpty:=True)
      Call DeleteFile(msOutputFile,
bCheckIfEmpty:=True)
   End If

   Exit Sub

DeleteEmptyOutputFilesErr:
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
    ' Not a critical error - continue

End Sub
Private Function
ReadCommandFromFile(strFileName As
String) As String

    ' Returns the contents of the passed in file

    Dim sCommand As String
    Dim sTemp As String
    Dim InputFile As Integer

    On Error GoTo
ReadCommandFromFileErr

    If Not StringEmpty(strFileName) Then

        InputFile = FreeFile
        Open strFileName For Input Access
Read As InputFile

        Line Input #InputFile, sCommand '
Read line into variable.

        Do While Not EOF(InputFile) ' Loop
until end of file.
            Line Input #InputFile, sTemp ' Read
line into variable.
            sCommand = sCommand & vbCrLf
& sTemp
        Loop

        Close InputFile
    End If

    ReadCommandFromFile = sCommand

    Exit Function

ReadCommandFromFileErr:

    ' Log the error code raised by Visual
Basic
    Call LogErrors(Errors)
    mstrSource = mstrModuleName &
"ReadCommandFromFile"
    On Error GoTo 0
    Err.Raise vbObjectError +
errSubValuesFailed, _
            gstrSource, _
            LoadResString(errSubValuesFailed)

End Function
Private Function
SubstituteParametersIfPossible(strLabel As
String)

    On Error GoTo
SubstituteParametersIfPossibleErr

    SubstituteParametersIfPossible =
SubstituteParameters(strLabel, _
            mcExecStep.WorkspaceId,
mcIterators,
WspParameters:=WspParameters)
    Exit Function

SubstituteParametersIfPossibleErr:
    SubstituteParametersIfPossible = strLabel

End Function
Private Function
SubstituteParametersInText(strFileName As
String, _
        lngWorkspace As Long) As String

    ' Reads each line in the passed in file,
substitutes parameter
    ' values in the line and writes out the
modified line to a
    ' temporary file that we create. The
temporary file will be
    ' removed once the step completes
execution.
    ' Returns the name of the newly created
temporary file.

    Dim strTempFile As String
    Dim strTemp As String
    Dim strOutput As String
    Dim InputFile As Integer
    Dim OutputFile As Integer

    On Error GoTo
SubstituteParametersInTextErr

    strTempFile = CreateTempFile()

    If Not StringEmpty(strFileName) Then

        InputFile = FreeFile
        Open strFileName For Input Access Read
As InputFile

        OutputFile = FreeFile
        Open strTempFile For Output Access
Write As OutputFile

        Do While Not EOF(InputFile) ' Loop until
end of file.
            Line Input #InputFile, strTemp ' Read
line into variable.
            strOutput =
SubstituteParameters(strTemp, lngWorkspace,
mcIterators, WspParameters:=WspParameters)

            If mcExecStep.ExecutionMechanism =
gintExecuteODBC Then strOutput =
SQLFixup(strOutput)

            Print #OutputFile, strOutput
            BugMessage strOutput
        Loop

    End If

    Close InputFile
    Close OutputFile

    SubstituteParametersInText = strTempFile

    Exit Function

SubstituteParametersInTextErr:

    ' Log the error code raised by Visual Basic
'   Call LogErrors(Errors)
    mcVBErr.LogVBErrors
    mstrSource = mstrModuleName &
"SubstituteParametersInText"
    On Error GoTo 0
    Err.Raise vbObjectError +
errSubValuesFailed, _
            gstrSource, _
            LoadResString(errSubValuesFailed)

End Function
Private Function WriteCommandToFile(sCommand
As String, Optional sFile As String =
gstrEmptyString) As String

    ' Writes the command text to a temporary file
    ' Returns the name of the temporary file

    Dim OutputFile As Integer

    On Error GoTo WriteCommandToFileErr

    If StringEmpty(sFile) Then
        sFile = CreateTempFile()
    End If

    OutputFile = FreeFile
    Open sFile For Output Access Write As
OutputFile

    Print #OutputFile, sCommand

    Close OutputFile

    WriteCommandToFile = sFile

    Exit Function

WriteCommandToFileErr:

    ' Log the error code raised by Visual Basic
    Call LogErrors(Errors)
    mstrSource = mstrModuleName &
"WriteCommandToFile"
    On Error GoTo 0
    Err.Raise vbObjectError + errSubValuesFailed, _
            gstrSource, _
            LoadResString(errSubValuesFailed)

End Function


Public Property Get WspPreCons() As Variant
    WspPreCons = mcvntWspPreCons
End Property
Public Property Let WspPreCons(ByVal vdata As
Variant)
    mcvntWspPreCons = vdata
End Property

Public Property Get WspPostCons() As Variant
    WspPostCons = mcvntWspPostCons
End Property
Public Property Let WspPostCons(ByVal vdata As
Variant)
    mcvntWspPostCons = vdata
End Property

Public Property Get PreCons() As Variant
    PreCons = mcvntPreCons
End Property
Public Property Let PreCons(ByVal vdata As
Variant)
    mcvntPreCons = vdata
End Property

Public Property Get PostCons() As Variant
    PostCons = mcvntPostCons
End Property
Public Property Let PostCons(ByVal vdata As
Variant)
    mcvntPostCons = vdata
End Property

Public Property Set Globals(cRunSteps As
cArrSteps)
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   Set mcGlobals = cRunSteps

End Property
Public Property Set ExecuteStep(cRunStep
As cStep)

   Set mcStep = cRunStep

End Property
Public Property Get Globals() As cArrSteps

   Set Globals = mcGlobals

End Property
Public Property Get ExecuteStep() As cStep

   Set ExecuteStep = mcStep

End Property
Public Property Set Iterators(vdata As
cRunColIt)

   Set mcIterators = vdata

End Property
Private Sub Class_Initialize()

   ' Initialize the Abort flag to False
   mblnAbort = False
   Set mcVBErr = New cVBErrorsSM
   Set mcTermProcess = New cTermProcess

End Sub

Private Sub Class_Terminate()

   On Error GoTo Class_TerminateErr

   Set mcExecObj = Nothing
   Set mcVBErr = Nothing
   Set mcTermProcess = Nothing

   Exit Sub

Class_TerminateErr:
   Call LogErrors(Errors)

End Sub

Private Sub mcExecObj_Start(ByVal
StartTime As Currency)
   ' Raise an event indicating that the step
has begun execution
   RaiseEvent ProcessStart(mcExecStep,
msRunStepDtl, StartTime, mlngInstanceId)
End Sub

Private Sub mcExecObj_Complete(ByVal
EndTime As Currency, ByVal Elapsed As
Long)

   On Error GoTo mcExecObj_CompleteErr

   Debug.Print Elapsed
   RaiseEvent
ProcessComplete(mcExecStep, EndTime,
mlngInstanceId, Elapsed)
   mcTermProcess.ProcessTerminated

   Exit Sub

mcExecObj_CompleteErr:
   Call LogErrors(Errors)
End Sub

Private Sub
mcTermProcess_TermProcessExists()

   On Error GoTo TermProcessExistsErr

   ' Call a procedure to execute the next step, if
any
   Call Execute

   Exit Sub

TermProcessExistsErr:
   ' Log the error code raised by the Execute
procedure
   Call LogErrors(Errors)

End Sub
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cRunWorkspace"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:       cRunWorkspace.cls
'              Microsoft TPC-H Kit Ver. 1.00
'              Copyright Microsoft, 1999
'              All Rights Reserved
'
'
'  PURPOSE:    This class loads all the
information necessary to
'              execute a workspace and calls
cRunInst to execute the workspace.
'              It also propagates Step start and
complete and
'              Run start and complete events.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

' Used to indicate the source module name
when errors
' are raised by this module
Private Const mstrModuleName As String =
"cRunWorkspace."
Private mstrSource As String

Private mcRunSteps As cArrSteps
Private mcRunParams As cArrParameters
Private mcRunConstraints As cArrConstraints
Private mcRunConnections As cConnections
Private mcRunConnDtls As cConnDtls
Private mcvntWspPreCons As Variant
Private mcvntWspPostCons As Variant
Private mdbsLoadDb As Database
Private mlngRunId As Long
Private mlngWorkspaceId As Long
Private mField As cStringSM
Public CreateInputFiles As Boolean

Private WithEvents mcRun As cRunInst
Attribute mcRun.VB_VarHelpID = -1

Public Event RunStart(dtmStartTime As
Currency, strWspLog As String, lRunId As
Long)
Public Event RunComplete(dtmEndTime As
Currency)
Public Event StepStart(cStepRecord As cStep,
dtmStartTime As Currency, lngInstanceId As Long,
_
      sPath As String, sIts As String)
Public Event StepComplete(cStepRecord As cStep,
dtmEndTime As Currency, lngInstanceId As Long)
Public Event ProcessStart(cStepRecord As cStep,
strCommand As String, _
      dtmStartTime As Currency, lngInstanceId As
Long)
Public Event ProcessComplete(cStepRecord As
cStep, dtmEndTime As Currency, lngInstanceId As
Long)
Public Function InstancesForStep(lngStepId As
Long, iStatus As InstanceStatus) As cInstances
   ' Returns an array of all the instances for a step

   If mcRun Is Nothing Then
      Set InstancesForStep = Nothing
   Else
      Set InstancesForStep =
mcRun.InstancesForStep(lngStepId, iStatus)
   End If

End Function
Private Sub InsertRunDetail(cStepRecord As cStep,
_
      strCommand As String, dtmStartTime As
Currency, _
      lngInstanceId As Long, lParentInstanceId As
Long, sItValue As String)
   ' Inserts a new run detail record into the database

   Dim strInsert As String
   Dim qy As QueryDef

   On Error GoTo InsertRunDetailErr
   mstrSource = mstrModuleName &
"InsertRunDetail"

   strInsert = "insert into run_step_details " & _
         "( run_id, step_id, version_no, instance_id,
parent_instance_id, " & _
         " command, start_time, iterator_value ) " &
_
         " values ( "

#If USE_JET Then

   strInsert = strInsert & " [r_id], [s_id], [ver_no],
[i_id], [p_i_id], " & _
         " [com], [s_date], [it_val] )"

   Set qy = mdbsLoadDb.CreateQueryDef( _
         gstrEmptyString, strInsert)

   ' Call a procedure to assign the Querydef
parameters
   Call AssignParameters(qy,
StartTime:=dtmStartTime, _
         StepId:=cStepRecord.StepId, _
         Version:=cStepRecord.VersionNo, _
         InstanceId:=lngInstanceId, _
         Command:=strCommand)

   qy.Execute dbFailOnError
   qy.Close

#Else

   strInsert = strInsert & Str(mlngRunId) _
      & ", " & Str(cStepRecord.StepId) _
      & ", " &
mField.MakeStringFieldValid(cStepRecord.Version
No) _
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
      & ", " & Str(lngInstanceId) _
      & ", " & Str(lParentInstanceId) _
      & ", " &
mField.MakeStringFieldValid(strCommand)
_
      & ", " & Str(dtmStartTime) _
      & ", " &
mField.MakeStringFieldValid(sItValue)

   strInsert = strInsert & " ) "

   mdbsLoadDb.Execute strInsert,
dbFailOnError

#End If

   Exit Sub

InsertRunDetailErr:
   LogErrors Errors
   mstrSource = mstrModuleName &
"InsertRunDetail"
   On Error GoTo 0
   Err.Raise vbObjectError +
errUpdateRunDataFailed, _
         mstrSource, _

LoadResString(errUpdateRunDataFailed)

End Sub
Private Sub UpdateRunDetail(cStepRecord
As cStep, _
      dtmEndTime As Currency,
lngInstanceId As Long, lElapsed As Long)
   ' Updates the run detail record in the
database

   Dim strUpdate As String
   Dim qy As QueryDef

   On Error GoTo UpdateRunDetailErr

   strUpdate = "update run_step_details " &
_
         " set end_time = [e_date],
elapsed_time = [elapsed] " & _
         " where run_id = [r_id] " & _
         " and step_id = [s_id] " & _
         " and version_no = [ver_no] " & _
         " and instance_id = [i_id] "

   Set qy = mdbsLoadDb.CreateQueryDef( _
         gstrEmptyString, strUpdate)

   ' Call a procedure to assign the Querydef
parameters
   Call AssignParameters(qy,
EndTime:=dtmEndTime, _
         StepId:=cStepRecord.StepId, _
         Version:=cStepRecord.VersionNo, _
         InstanceId:=lngInstanceId,
Elapsed:=lElapsed)

   qy.Execute dbFailOnError
   qy.Close

   Exit Sub

UpdateRunDetailErr:
   LogErrors Errors
   mstrSource = mstrModuleName &
"UpdateRunDetail"
   On Error GoTo 0
   Err.Raise vbObjectError +
errUpdateRunDataFailed, _
         mstrSource, _

LoadResString(errUpdateRunDataFailed)

End Sub
Private Function
InsertRunHeader(dtmStartTime As Currency)
As Long
   ' Inserts a new run header record into the
database
   ' and returns the id for the run

   Dim strInsert As String
   Dim qy As QueryDef

   On Error GoTo InsertRunHeaderErr

   strInsert = "insert into run_header " & _
      "( run_id, workspace_id, start_time ) " &
_
      " values ( " & _
      " [r_id], [w_id], [s_date] )"

   Set qy = mdbsLoadDb.CreateQueryDef( _
         gstrEmptyString, strInsert)

   ' Call a procedure to execute the Querydef
object
   Call AssignParameters(qy,
StartTime:=dtmStartTime)

   qy.Execute dbFailOnError
   qy.Close

   InsertRunHeader = mlngRunId
   Exit Function

InsertRunHeaderErr:
   LogErrors Errors
   mstrSource = mstrModuleName &
"InsertRunHeader"
   On Error GoTo 0
   Err.Raise vbObjectError +
errUpdateRunDataFailed, _
         mstrSource, _

LoadResString(errUpdateRunDataFailed)

End Function
Private Sub
InsertRunParameters(dtmStartTime As
Currency)
   ' Inserts a new run header record into the
database
   ' and returns the id for the run

   Dim strInsert As String
   Dim qy As QueryDef
   Dim cParamRec As cParameter
   Dim lngIndex As Long

   On Error GoTo InsertRunParametersErr

   strInsert = "insert into run_parameters " & _
      "( run_id, parameter_name,
parameter_value ) " & _
      " values ( " & _
      " [r_id], [p_name], [p_value] )"

   Set qy = mdbsLoadDb.CreateQueryDef( _
         gstrEmptyString, strInsert)
   qy.Parameters("r_id").Value = mlngRunId

   For lngIndex = 0 To
mcRunParams.ParameterCount - 1
      Set cParamRec = mcRunParams(lngIndex)

      qy.Parameters("p_name").Value =
cParamRec.ParameterName
      qy.Parameters("p_value").Value =
cParamRec.ParameterValue
      qy.Execute dbFailOnError

   Next lngIndex

   qy.Close

   Exit Sub

InsertRunParametersErr:
   LogErrors Errors
   mstrSource = mstrModuleName &
"InsertRunParameters"
   On Error GoTo 0
   Err.Raise vbObjectError +
errUpdateRunDataFailed, _
         mstrSource, _
         LoadResString(errUpdateRunDataFailed)

End Sub
Private Sub AssignParameters(qyExec As
DAO.QueryDef, _
      Optional StartTime As Currency = 0, _
      Optional EndTime As Currency = 0, _
      Optional StepId As Long = 0, _
      Optional Version As String = gstrEmptyString,
_
      Optional InstanceId As Long = 0, _
      Optional ParentInstanceId As Long = 0, _
      Optional Command As String =
gstrEmptyString, _
      Optional Elapsed As Long = 0, _
      Optional ItValue As String = gstrEmptyString)
   ' Assigns values to the parameters in the querydef
object

   Dim prmParam As DAO.Parameter

   On Error GoTo AssignParametersErr
   mstrSource = mstrModuleName &
"AssignParameters"

   For Each prmParam In qyExec.Parameters
      Select Case prmParam.Name
         Case "[w_id]"
            prmParam.Value = mlngWorkspaceId

         Case "[r_id]"
            prmParam.Value = mlngRunId

         Case "[s_id]"
            BugAssert StepId <> 0
            prmParam.Value = StepId

         Case "[ver_no]"
            BugAssert Not StringEmpty(Version)
            prmParam.Value = Version

         Case "[i_id]"
            BugAssert InstanceId <> 0
            prmParam.Value = InstanceId

         Case "[p_i_id]"
            prmParam.Value = ParentInstanceId

         Case "[com]"
            BugAssert Not StringEmpty(Command)
            prmParam.Value = Command

         Case "[s_date]"
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
            BugAssert StartTime <> 0
            prmParam.Value = StartTime

        Case "[e_date]"
            BugAssert EndTime <> 0
            prmParam.Value = EndTime

        Case "[elapsed]"
            prmParam.Value = Elapsed

        Case "[it_val]"
            prmParam.Value = ItValue

        Case Else
            ' Write the parameter name that is
faulty
            WriteError errInvalidParameter,
mstrSource, _
                prmParam.Name
            On Error GoTo 0
            Err.Raise errInvalidParameter,
mstrSource, _

LoadResString(errInvalidParameter)
        End Select
    Next prmParam

    Exit Sub

AssignParametersErr:

    mstrSource = mstrModuleName &
"AssignParameters"
    Call LogErrors(Errors)
    On Error GoTo 0
    Err.Raise vbObjectError +
errAssignParametersFailed, _
        mstrSource,
LoadResString(errAssignParametersFailed)

End Sub
Private Sub
RunStartProcessing(dtmStartTime As
Currency)

    On Error GoTo RunStartProcessingErr

    ' Insert the run header into the database
    Call InsertRunHeader(dtmStartTime)

    ' Insert the run parameters into the
database
    Call InsertRunParameters(dtmStartTime)

    Exit Sub

RunStartProcessingErr:
    ' Log the error code raised by Visual
Basic
    Call LogErrors(Errors)
    mstrSource = mstrModuleName &
"RunStartProcessing"
    ShowError errUpdateRunDataFailed
    WriteError errUpdateRunDataFailed,
mstrSource

End Sub
Private Sub
ProcessStartProcessing(cStepRecord As
cStep, _
        strCommand As String, dtmStartTime
As Currency, lngInstanceId As Long, _
        lParentInstanceId As Long, sItValue As
String)

    On Error GoTo ProcessStartProcessingErr

    ' Insert the run detail into the database
    Call InsertRunDetail(cStepRecord,
strCommand, dtmStartTime, lngInstanceId, _
            lParentInstanceId, sItValue)

    Exit Sub

ProcessStartProcessingErr:
    ' Log the error code raised by Visual Basic
    Call LogErrors(Errors)
    mstrSource = mstrModuleName &
"ProcessStartProcessing"
    ShowError errUpdateRunDataFailed
    WriteError errUpdateRunDataFailed,
mstrSource

End Sub
Private Sub StepStartProcessing(cStepRecord
As cStep, dtmStartTime As Currency, _
        lngInstanceId As Long, lParentInstanceId
As Long, sItValue As String)

    On Error GoTo StepStartProcessingErr

    ' Since ProcessStart events won't be
triggered for manager steps
    If cStepRecord.StepType = gintManagerStep
Then
        ' Insert the run detail into the database
        Call InsertRunDetail(cStepRecord,
cStepRecord.StepLabel, _
                dtmStartTime, lngInstanceId,
lParentInstanceId, sItValue)
    End If

    Exit Sub

StepStartProcessingErr:
    ' Log the error code raised by Visual Basic
    Call LogErrors(Errors)
    mstrSource = mstrModuleName &
"StepStartProcessing"
    ShowError errUpdateRunDataFailed

End Sub
Private Sub
ProcessCompleteProcessing(cStepRecord As
cStep, _
        dtmStartTime As Currency, lngInstanceId
As Long, lElapsed As Long)

    On Error GoTo
ProcessCompleteProcessingErr

    ' Insert the run detail into the database
    Call UpdateRunDetail(cStepRecord,
dtmStartTime, lngInstanceId, lElapsed)

    Exit Sub

ProcessCompleteProcessingErr:
    ' Log the error code raised by Visual Basic
    Call LogErrors(Errors)
    mstrSource = mstrModuleName &
"ProcessCompleteProcessing"
    ShowError errUpdateRunDataFailed

End Sub
Private Sub
StepCompleteProcessing(cStepRecord As
cStep, _
        dtmEndTime As Currency, lngInstanceId
As Long, lElapsed As Long)

    On Error GoTo StepCompleteProcessingErr

    ' Since ProcessComplete events won't be
triggered for manager steps
    If cStepRecord.StepType = gintManagerStep
Then
        ' Update the run detail in the database
        Call UpdateRunDetail(cStepRecord,
dtmEndTime, lngInstanceId, lElapsed)
    End If

    Exit Sub

StepCompleteProcessingErr:
    ' Log the error code raised by Visual Basic
    Call LogErrors(Errors)
    ShowError errUpdateRunDataFailed

End Sub
Private Sub RunCompleteProcessing(dtmEndTime
As Currency)

    On Error GoTo RunCompleteProcessingErr

    ' Update the header record with the end time for
the run
    Call UpdateRunHeader(dtmEndTime)

    Exit Sub

RunCompleteProcessingErr:
    ' Log the error code raised by Visual Basic
    Call LogErrors(Errors)
    ShowError errUpdateRunDataFailed

End Sub

Private Sub UpdateRunHeader(ByVal dtmEndTime
As Currency)
    ' Updates the run header record with the end date

    Dim strUpdate As String
    Dim qy As QueryDef

    On Error GoTo UpdateRunHeaderErr

    strUpdate = "update run_header " & _
            " set end_time = [e_date] " & _
            " where run_id = [r_id] "

    Set qy = mdbsLoadDb.CreateQueryDef( _
            gstrEmptyString, strUpdate)

    ' Call a procedure to execute the Querydef object
    Call AssignParameters(qy,
EndTime:=dtmEndTime)

    qy.Execute dbFailOnError
    qy.Close

    Exit Sub

UpdateRunHeaderErr:
    LogErrors Errors
    mstrSource = mstrModuleName &
"UpdateRunHeader"
    On Error GoTo 0
    Err.Raise vbObjectError +
errUpdateRunDataFailed, _
            mstrSource, _
            LoadResString(errUpdateRunDataFailed)

End Sub
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
Public Property Let WorkspaceId(ByVal
vdata As Long)
   mlngWorkspaceId = vdata
End Property
Public Property Get WorkspaceId() As Long
   WorkspaceId = mlngWorkspaceId
End Property
Public Sub RunWorkspace()

   Dim cRunSeq As cSequence

   On Error GoTo RunWorkspaceErr

   ' Call a procedure to load the module-
level structures
   ' with all the step and parameter data for
the run
   If LoadRunData = False Then
      ' Error handled by the function already
      Exit Sub
   End If

   ' Retrieve the next run identifier using the
sequence class
   Set cRunSeq = New cSequence
   Set cRunSeq.IdDatabase = dbsAttTool
   cRunSeq.IdentifierColumn = "run_id"
   mlngRunId = cRunSeq.Identifier
   Set cRunSeq = Nothing

   Call
mcRunParams.InitBuiltInsForRun(mlngWor
kspaceId, mlngRunId)

   Set mcRun.Constraints =
mcRunConstraints
   mcRun.WspPreExecution =
mcvntWspPreCons
   mcRun.WspPostExecution =
mcvntWspPostCons

   Set mcRun.Steps = mcRunSteps
   Set mcRun.Parameters = mcRunParams
   Set mcRun.RunConnections =
mcRunConnections
   Set mcRun.RunConnDtls =
mcRunConnDtls

   mcRun.WspId = mlngWorkspaceId
   mcRun.RootKey =
LabelStep(mlngWorkspaceId)
   mcRun.RunId = mlngRunId
   mcRun.CreateInputFiles =
CreateInputFiles

   mcRun.Run

   Exit Sub

RunWorkspaceErr:
   ' Log the error code raised by Visual
Basic
   Call LogErrors(Errors)

End Sub
Public Property Get LoadDb() As Database

   Set LoadDb = mdbsLoadDb

End Property
Public Property Set LoadDb(vdata As
Database)

   Set mdbsLoadDb = vdata
End Property
Private Function LoadRunData() As Boolean

   ' Loads the step, parameter and constraint
arrays
   ' with all the data for the workspace. Returns
False
   ' if a failure occurs

   Dim strWorkspaceName As String
   Dim recWspSteps As Recordset
   Dim qySteps As DAO.QueryDef
   Dim recWspParams As Recordset
   Dim qyParams As DAO.QueryDef
   Dim recWspConns As Recordset
   Dim qyConns As DAO.QueryDef
   Dim recWspConnDtls As Recordset
   Dim qyConnDtls As DAO.QueryDef

   On Error GoTo LoadRunDataErr

   Set mcRunSteps.StepDB = mdbsLoadDb
   Set mcRunParams.ParamDatabase =
mdbsLoadDb
   Set mcRunConstraints.ConstraintDB =
mdbsLoadDb
   Set mcRunConnections.ConnDb =
mdbsLoadDb
   Set mcRunConnDtls.ConnDb =
mdbsLoadDb

   ' Read all the step and parameter data for the
workspace
   Call
ReadWorkspaceData(mlngWorkspaceId,
mcRunSteps, _
         mcRunParams, mcRunConstraints,
mcRunConnections, mcRunConnDtls, _
         recWspSteps, qySteps, recWspParams,
qyParams, recWspConns, qyConns, _
         recWspConnDtls, qyConnDtls)

   ' Load all the pre- and post-execution
constraints that
   ' have been defined for the workspace
   mcvntWspPreCons =
mcRunConstraints.ConstraintsForWsp( _
      mlngWorkspaceId, _
      gintPreStep, _
      blnSort:=True, _
      blnGlobalConstraintsOnly:=True)
   mcvntWspPostCons =
mcRunConstraints.ConstraintsForWsp( _
      mlngWorkspaceId, _
      gintPostStep, _
      blnSort:=True, _
      blnGlobalConstraintsOnly:=True)

   On Error Resume Next
   recWspSteps.Close
   qySteps.Close
   recWspParams.Close
   qyParams.Close
   recWspConns.Close
   qyConns.Close

   LoadRunData = True

   Exit Function

LoadRunDataErr:
   ' Log the error code raised by Visual Basic
   Call LogErrors(Errors)
   ShowError errLoadRunDataFailed
   LoadRunData = False
End Function
Public Sub StopRun()

   On Error GoTo StopRunErr

   If mcRun Is Nothing Then
      ' We haven't been the run yet, so do nothing
   Else
      mcRun.StopRun
   End If

   Exit Sub

StopRunErr:
   ' Log the error code raised by Visual Basic
   Call LogErrors(Errors)
   ' Errors would have been displayed by the called
process

End Sub
Public Sub AbortRun()

   On Error GoTo AbortRunErr

   If mcRun Is Nothing Then
      ' We haven't been the run yet, so do nothing
   Else
      mcRun.Abort
   End If

   Exit Sub

AbortRunErr:
   ' Log the error code raised by Visual Basic
   Call LogErrors(Errors)
   ' Errors would have been displayed by the called
process

End Sub

Private Sub Class_Initialize()

   ' Create instances of the step, parameter and
constraint arrays
   Set mcRunSteps = New cArrSteps
   Set mcRunParams = New cArrParameters
   Set mcRunConstraints = New cArrConstraints
   Set mcRunConnections = New cConnections
   Set mcRunConnDtls = New cConnDtls
   Set mcRun = New cRunInst
   Set mField = New cStringSM

End Sub
Private Sub Class_Terminate()

   On Error GoTo UnLoadRunDataErr

   ' Clears the step, parameter and constraint arrays
   Set mcRunSteps = Nothing
   Set mcRunParams = Nothing
   Set mcRunConstraints = Nothing
   Set mcRunConnections = Nothing
   Set mcRunConnDtls = Nothing

   Set mcRun = Nothing
   Set mdbsLoadDb = Nothing
   Set mField = Nothing

   Exit Sub

UnLoadRunDataErr:
   ' Log the error code raised by Visual Basic
   Call LogErrors(Errors)
   ' Not a critical error - continue
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
    Resume Next

End Sub

Private Sub
mcRun_ProcessComplete(cStepRecord As
cStep, _
        dtmEndTime As Currency,
lngInstanceId As Long, lElapsed As Long)

    RaiseEvent
ProcessComplete(cStepRecord,
dtmEndTime, lngInstanceId)
    Call
ProcessCompleteProcessing(cStepRecord,
dtmEndTime, lngInstanceId, lElapsed)

End Sub

Private Sub
mcRun_ProcessStart(cStepRecord As cStep,
_
        strCommand As String, dtmStartTime
As Currency, lngInstanceId As Long, _
        lParentInstanceId As Long, sItValue As
String)

    RaiseEvent ProcessStart(cStepRecord,
strCommand, dtmStartTime, lngInstanceId)
    Call ProcessStartProcessing(cStepRecord,
strCommand, dtmStartTime, lngInstanceId,
_
            lParentInstanceId, sItValue)

End Sub

Private Sub
mcRun_RunComplete(dtmEndTime As
Currency)

    Debug.Print "Run ended at: " &
CStr(dtmEndTime)
    Call
RunCompleteProcessing(dtmEndTime)

    RaiseEvent RunComplete(dtmEndTime)

End Sub
Private Sub mcRun_RunStart(dtmStartTime
As Currency, strWspLog As String)

    RaiseEvent RunStart(dtmStartTime,
strWspLog, mlngRunId)
    Debug.Print "Run started at: " &
CStr(dtmStartTime)

    Call RunStartProcessing(dtmStartTime)

End Sub
Private Sub
mcRun_StepComplete(cStepRecord As
cStep, _
        dtmEndTime As Currency,
lngInstanceId As Long, lElapsed As Long)

    RaiseEvent StepComplete(cStepRecord,
dtmEndTime, lngInstanceId)
'    BugMessage "Step: " &
cStepRecord.StepLabel & " has completed!"

    Call
StepCompleteProcessing(cStepRecord,
dtmEndTime, lngInstanceId, lElapsed)

End Sub
Private Sub mcRun_StepStart(cStepRecord As
cStep, dtmStartTime As Currency, _
        lngInstanceId As Long, lParentInstanceId
As Long, sPath As String, sIts As String,
sItValue As String)

    RaiseEvent StepStart(cStepRecord,
dtmStartTime, lngInstanceId, sPath, sIts)
    'bugmessage "Step: " &
cStepRecord.StepLabel & " has started."

    Call StepStartProcessing(cStepRecord,
dtmStartTime, lngInstanceId,
lParentInstanceId, sItValue)

End Sub

VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cSequence"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:       cSequence.cls
'              Microsoft TPC-H Kit Ver. 1.00
'              Copyright Microsoft, 1999
'              All Rights Reserved
'
'
'  PURPOSE:     This class uses the
att_identifiers table to generate unique
'              identifiers.
'  Contact:     Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

Private mlngIdentifier As Long
Private mstrIdentifierColumn As String
Private mrecIdentifiers As Recordset
Private mdbsDatabase As Database

Private Const mstrEmptyString = ""

' Used to indicate the source module name
when errors
' are raised by this class
Private mstrSource As String
Private Const mstrModuleName As String =
"cSequence."

Private Sub CreateIdRecord()
    ' Creates a record with all identifiers having
an initial value of 1

    Dim sSql As String
    Dim pId As DAO.Parameter
    Dim qyId As DAO.QueryDef

    sSql = "insert into att_identifiers (" & _
            " workspace_id, parameter_id, step_id,
" & _
            " constraint_id, run_id, connection_id "
& _
            ", " & FLD_ID_CONN_NAME & _
            " ) values ( " & _
            "[w_id], [p_id], [s_id], [c_id], [r_id],
[conn_id], [conn_dtl_id] )"
    Set qyId =
mdbsDatabase.CreateQueryDef(gstrEmptyString, sSql)
    For Each pId In qyId.Parameters
        pId.Value = glMinId
    Next pId
    qyId.Execute dbFailOnError
    qyId.Close

End Sub
Private Sub CreateIdRecordset()

    Dim strSql As String

    ' Initialize the recordset with all identifiers
    strSql = "select * from att_identifiers"
    Set mrecIdentifiers =
mdbsDatabase.OpenRecordset(strSql,
dbOpenForwardOnly)

    If mrecIdentifiers.RecordCount = 0 Then
        CreateIdRecord
        Set mrecIdentifiers =
mdbsDatabase.OpenRecordset(strSql,
dbOpenForwardOnly)
    End If

    BugAssert mrecIdentifiers.RecordCount <> 0

End Sub

Public Property Set IdDatabase(vdata As Database)

    Set mdbsDatabase = vdata

End Property

Public Property Let IdentifierColumn(vdata As
String)

    Dim intIndex As Integer

    On Error GoTo IdentifierColumnErr

    ' Initialize the return value to an empty string
    mstrIdentifierColumn = mstrEmptyString
    Call CreateIdRecordset

    For intIndex = 0 To mrecIdentifiers.Fields.Count
- 1

        If
LCase(Trim(mrecIdentifiers.Fields(intIndex).Name
)) = _
            LCase(Trim(vdata)) Then

            ' Valid column name
            mstrIdentifierColumn = vdata
            Exit Property
        End If

    Next intIndex

    BugAssert True, "Invalid column name!"

    Exit Property

IdentifierColumnErr:
    LogErrors Errors
    mstrSource = mstrModuleName &
"IdentifierColumn"
    On Error GoTo 0
    Err.Raise vbObjectError +
errIdentifierColumnFailed, _
            mstrSource, _
            LoadResString(errIdentifierColumnFailed)

End Property
Public Property Get Identifier() As Long
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
    Dim strSql As String

    On Error GoTo GetIdentifierErr

    BugAssert mstrIdentifierColumn <>
mstrEmptyString

    ' Increment the identifier column by 1
    strSql = "update att_identifiers " & _
        " set " & mstrIdentifierColumn & _
        " = " & mstrIdentifierColumn & " + 1"
    mdbsDatabase.Execute strSql,
dbFailOnError

    ' Refresh the recordset with identifier
values
    Call CreateIdRecordset

    mlngIdentifier =
mrecIdentifiers.Fields(mstrIdentifierColumn
).Value

    Identifier = mlngIdentifier

    Exit Property

GetIdentifierErr:
    LogErrors Errors
    mstrSource = mstrModuleName &
"Identifier"
    On Error GoTo 0
    Err.Raise vbObjectError +
errGetIdentifierFailed, _
            mstrSource, _

LoadResString(errGetIdentifierFailed)

End Property
Private Sub Class_Terminate()

    mrecIdentifiers.Close

End Sub

VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cStack"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:        cStack.cls
'               Microsoft TPC-H Kit Ver. 1.00
'               Copyright Microsoft, 1999
'               All Rights Reserved
'
'
'  PURPOSE:     This class implements a
stack of objects.
'  Contact:     Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

' Used to indicate the source module name
when errors
' are raised by this class
Private Const mstrModuleName As String =
"cStack."
Private mstrSource As String

Private mcVector As cVector
Private mlngCount As Long
Public Property Get Item(ByVal Position As
Long) As Object
Attribute Item.VB_UserMemId = 0

    Set Item = mcVector(Position)

End Property

Public Sub Push(objToPush As Object)

    mcVector.Add objToPush

End Sub
Public Sub Clear()

    mcVector.Clear

End Sub

Public Function Pop() As Object

    If mcVector.Count > 0 Then
        Set Pop =
mcVector.Delete(mcVector.Count - 1)
    Else
        Set Pop = Nothing
    End If

End Function
Public Function Count() As Long

    Count = mcVector.Count

End Function


Private Sub Class_Initialize()

    Set mcVector = New cVector

End Sub


Private Sub Class_Terminate()

    Set mcVector = Nothing

End Sub


VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cStep"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
Attribute VB_Ext_KEY =
"SavedWithClassBuilder" ,"Yes"
Attribute VB_Ext_KEY = "Top_Level" ,"Yes"
'  FILE:        cStep.cls
'               Microsoft TPC-H Kit Ver. 1.00
'               Copyright Microsoft, 1999
'               All Rights Reserved
'
'
'  PURPOSE:     Encapsulates the properties
and methods of a step.
'               Contains functions to insert, update
and delete
'               att_steps records from the database.
'  Contact:     Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

' Local variable(s) to hold property value(s)
Private mlngStepId As Long
Private mstrVersionNo As String
Private mstrStepLabel As String
Private mstrStepTextFile As String
Private mstrStepText As String
Private mstrStartDir As String
Private mlngWorkspaceId As Integer
Private mlngParentStepId As Integer
Private mstrParentVersionNo As String
Private mintSequenceNo As Integer
Private mintStepLevel As Integer
Private mblnEnabledFlag As Boolean
Private mstrDegreeParallelism As String
Private mintExecutionMechanism As Integer
Private mstrFailureDetails As String
Private mintContinuationCriteria As Integer
Private mblnGlobalFlag As Boolean
Private mblnArchivedFlag As Boolean
Private mstrOutputFile As String
'Private mstrLogFile As String
Private mstrErrorFile As String
Private mdbsDatabase As Database
Private mintStepType As Integer
Private mintOperation As Operation
Private mlngPosition As Long
Private mstrIteratorName As String
Private mcIterators As cNodeCollections
Private mbIsNewVersion As Boolean
Private msOldVersion As String

' The following constants are used throughout the
project to
' indicate the different options selected by the user
' The options are presented to the user as control
arrays of
' option buttons. These constants have to be in sync
with the
' indexes of the option buttons.
' All the control arrays have an lbound of 1. The
value 0 is
' used to indicate that the property being represented
by the
' control array is not valid for the step
' Public enums are used since we cannot expose
public constants
' in class modules. gintNoOption is applicable to all
enums,
' but declared in the Execution method enum, since
we cannot
' declare it more than once.

' Is here as a comment
' Has been defined in public.bas with the other
object types
'Public Enum gintStepType
'    gintGlobalStep = 3
'    gintManagerStep
'    gintWorkerStep
'End Enum

' Execution Method options
Public Enum ExecutionMethod
    gintNoOption = 0
    gintExecuteODBC
    gintExecuteShell
End Enum

' Failure criteria options
Public Enum FailureCriteria
    gintFailureODBC = 1
    gintFailureTextCompare
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
End Enum

' Continuation criteria options
' Note: Update the initialization of
gsContCriteria in Initialize() if the
' continuation criteria are modified
Public Enum ContinuationCriteria
   gintOnFailureAbort = 1
   gintOnFailureContinue
   gintOnFailureCompleteSiblings
   gintOnFailureAbortSiblings
   gintOnFailureSkipSiblings
   gintOnFailureAsk
End Enum

' The initial version #
Private Const mstrMinVersion As String =
"0.0"

' End of constants for option button control
arrays
' Used to indicate the source module name
when errors
' are raised by this class
Private mstrSource As String
Private Const mstrModuleName As String =
"cStep."

' The cSequence class is used to generate
unique step identifiers
Private mStepSeq As cSequence

' The StringSM class is used to carry out
string operations
Private mFieldValue As cStringSM
Private Sub NewVersion()

mbIsNewVersion = True
msOldVersion = mstrVersionNo

End Sub
Public Function IsNewVersion() As Boolean
   IsNewVersion = mbIsNewVersion
End Function

Public Function OldVersionNo() As String
   OldVersionNo = msOldVersion
End Function

Public Sub SaveIterators()
   ' This procedure checks if any changes
have been made
   ' to the iterators for the step. If so, it calls
the
   ' methods of the iterator class to commit
the changes
   Dim cItRec As cIterator
   Dim lngIndex As Long

   On Error GoTo SaveIteratorsErr

   For lngIndex = 0 To mcIterators.Count - 1
      Set cItRec = mcIterators(lngIndex)

      Select Case cItRec.IndOperation
         Case QueryOp
            ' No changes were made to the
queried Step.
            ' Do nothing

         Case InsertOp
            cItRec.Add mlngStepId,
mstrVersionNo
            cItRec.IndOperation = QueryOp

         Case UpdateOp
            cItRec.Update mlngStepId,
mstrVersionNo
            cItRec.IndOperation = QueryOp

         Case DeleteOp
            cItRec.Delete mlngStepId,
mstrVersionNo
            ' Remove the record from the
collection
            mcIterators.Delete lngIndex

      End Select
   Next lngIndex

   Exit Sub

SaveIteratorsErr:
   LogErrors Errors
   mstrSource = mstrModuleName &
"SaveIterators"
   On Error GoTo 0
   Err.Raise vbObjectError + errSaveFailed, _
         mstrSource, _
         LoadResString(errSaveFailed)

End Sub
Public Property Get IndOperation() As
Operation

   IndOperation = mintOperation

End Property
Public Property Let IndOperation(ByVal vdata
As Operation)

   BugAssert vdata = QueryOp Or vdata =
InsertOp Or vdata = UpdateOp Or vdata =
DeleteOp, "Invalid operation"
   mintOperation = vdata

End Property

Public Function Iterators() As Variant
   ' Returns a variant containing all the iterators
that
   ' have been defined for the step

   Dim cStepIterators() As cIterator
   Dim cTempIt As cIterator
   Dim lngIndex As Long
   Dim lngItCount As Long

   On Error GoTo IteratorsErr

   lngItCount = 0
   For lngIndex = 0 To mcIterators.Count - 1
      ' Increase the array dimension and add the
constraint
      ' to it
      Set cTempIt = mcIterators(lngIndex)

      If cTempIt.IndOperation <> DeleteOp
Then
         ReDim Preserve
cStepIterators(lngItCount)
         Set cStepIterators(lngItCount) =
cTempIt
         lngItCount = lngItCount + 1
      End If

   Next lngIndex

   If lngItCount = 0 Then
      Iterators = Empty
   Else
      Iterators = cStepIterators()
   End If

   Call QuickSort(Iterators)

   Exit Function

IteratorsErr:
   LogErrors Errors
   On Error GoTo 0
   Err.Raise vbObjectError + errIteratorsFailed, _
         mstrModuleName & "Iterators", _
         LoadResString(errIteratorsFailed)

End Function
Public Function IteratorCount() As Long
   ' Returns a count of all the iterators for the step

   Dim lngItCount As Long
   Dim lngIndex As Long
   Dim cTempIt As cIterator

   On Error GoTo IteratorsErr

   lngItCount = 0
   For lngIndex = 0 To mcIterators.Count - 1

      If mcIterators(lngIndex).IndOperation <>
DeleteOp Then
         lngItCount = lngItCount + 1
      End If

   Next lngIndex

   IteratorCount = lngItCount

   Exit Function

IteratorsErr:
   LogErrors Errors
   On Error GoTo 0
   Err.Raise vbObjectError + errIteratorsFailed, _
         mstrSource, _
         LoadResString(errIteratorsFailed)

End Function
Public Sub Validate()
   ' Each distinct object will have a Validate method
which
   ' will check if the class properties are valid. This
method
   ' will be used to check interdependant properties
that
   ' cannot be validated by the let procedures.
   ' It should be called by the add and modify
methods of the class

   ' Check if the step label has been specified
   If StringEmpty(mstrStepLabel) Then
      ShowError errStepLabelMandatory
      On Error GoTo 0
      Err.Raise vbObjectError + errValidateFailed, _
            "Validate",
LoadResString(errValidateFailed)
   End If

   If Not IsStringEmpty(mstrStepText) And Not
IsStringEmpty(mstrStepTextFile) Then
      ShowError errStepTextOrFile
      On Error GoTo 0
      Err.Raise vbObjectError + errStepTextOrFile,
_
            "Validate",
LoadResString(errStepTextOrFile)
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
    End If

End Sub

Public Function IncVersionY() As String
    ' The version number for a step is stored
in the x.y
    ' format where x is the parent component
and y is the
    ' child component of the step. This
function will increment
    ' the y component of the step by 1

    On Error GoTo IncVersionYErr

    ' Store the old version number for the step
    Call NewVersion

    mstrVersionNo =
Trim$(Str$(GetX(mstrVersionNo))) &
gstrVerSeparator & _
            Trim$(Str(GetY(mstrVersionNo) +
1))
    IncVersionY = mstrVersionNo

    Exit Function

IncVersionYErr:
    ' Log the error code raised by Visual
Basic
    Call LogErrors(Errors)
    gstrSource = mstrModuleName &
"IncVersionY"
    On Error GoTo 0
    Err.Raise vbObjectError +
errIncVersionYFailed, _
            gstrSource, _

LoadResString(errIncVersionYFailed)

End Function
Public Function IncVersionX() As String
    ' The version number for a step is stored
in the x.y
    ' format where x is the parent component
and y is the
    ' child component of the step. This
function will increment
    ' the y component of the step by 1 and
reset the x component
    ' to 0

    On Error GoTo IncVersionXErr

    ' Store the old version number for the step
    Call NewVersion

    mstrVersionNo =
Trim$(Str$(GetX(mstrVersionNo) + 1)) &
gstrVerSeparator & "0"
    IncVersionX = mstrVersionNo

    Exit Function

IncVersionXErr:
    ' Log the error code raised by Visual
Basic
    Call LogErrors(Errors)
    gstrSource = mstrModuleName &
"IncVersionX"
    On Error GoTo 0
    Err.Raise vbObjectError +
errIncVersionXFailed, _
            gstrSource, _
            LoadResString(errIncVersionXFailed)

End Function


Private Function GetY(strVersion As String)
As Long
    ' The version number for a step is stored in
the x.y
    ' format where x is the parent component and
y is the
    ' child component of the step. Given an
argument of type
    ' x.y, it returns y

    ' Truncate the fractional part to get the parent
component
    ' of the version number (x.y)
    GetY = Val(Mid(strVersion,
InStr(strVersion, gstrVerSeparator) + 1))

End Function
Private Function GetX(strVersion As String)
As Long
    ' The version number for a step is stored in
the x.y
    ' format where x is the parent component and
y is the
    ' child component of the step. Given an
argument of type
    ' x.y, it returns x

    ' Truncate the fractional part to get the parent
component
    ' of the version number (x.y)
    GetX = Val(Left(strVersion,
InStr(strVersion, gstrVerSeparator) - 1))

End Function

Public Function Clone(Optional cCloneStep As
cStep) As cStep

    ' Creates a copy of a given step

    Dim lngIndex As Long
    Dim cItRec As cIterator
    Dim cItClone As cIterator

    On Error GoTo CloneErr

    If cCloneStep Is Nothing Then
        Set cCloneStep = New cStep
    End If

    ' Copy all the step properties to the newly
created step
    ' Initialize the global flag first since
subsequent
    ' validations might depend on it
    cCloneStep.GlobalFlag = mblnGlobalFlag
'   cCloneStep.GlobalRunMethod =
mintGlobalRunMethod

    cCloneStep.StepType = mintStepType
    cCloneStep.StepId = mlngStepId
    cCloneStep.VersionNo = mstrVersionNo
    cCloneStep.StepLabel = mstrStepLabel
    cCloneStep.StepTextFile = mstrStepTextFile
    cCloneStep.StepText = mstrStepText
    cCloneStep.StartDir = mstrStartDir
    cCloneStep.WorkspaceId =
mlngWorkspaceId
    cCloneStep.ParentStepId =
mlngParentStepId
    cCloneStep.ParentVersionNo =
mstrParentVersionNo
    cCloneStep.StepLevel = mintStepLevel
    cCloneStep.SequenceNo = mintSequenceNo
    cCloneStep.EnabledFlag = mblnEnabledFlag
    cCloneStep.DegreeParallelism =
mstrDegreeParallelism
    cCloneStep.ExecutionMechanism =
mintExecutionMechanism
    cCloneStep.FailureDetails = mstrFailureDetails
    cCloneStep.ContinuationCriteria =
mintContinuationCriteria
    cCloneStep.ArchivedFlag = mblnArchivedFlag
    cCloneStep.OutputFile = mstrOutputFile
'   cCloneStep.LogFile = mstrLogFile
    cCloneStep.ErrorFile = mstrErrorFile
    cCloneStep.IteratorName = mstrIteratorName

    cCloneStep.IndOperation = mintOperation
    cCloneStep.Position = mlngPosition

    Set cCloneStep.NodeDB = mdbsDatabase

    ' Clone all the iterators for the step
    For lngIndex = 0 To mcIterators.Count - 1
        Set cItRec = mcIterators(lngIndex)
        Set cItClone = cItRec.Clone
        cCloneStep.LoadIterator cItClone
    Next lngIndex

    ' And set the return value to the newly created
step
    Set Clone = cCloneStep

    Exit Function

CloneErr:
    LogErrors Errors
    mstrSource = mstrModuleName & "Clone"
    On Error GoTo 0
    Err.Raise vbObjectError + errCloneFailed, _
            mstrSource, LoadResString(errCloneFailed)

End Function
'End Sub
'
Public Property Let OutputFile(ByVal vdata As
String)

    mstrOutputFile = vdata

End Property

Public Property Get OutputFile() As String

    OutputFile = mstrOutputFile

End Property

'Public Property Let LogFile(ByVal vdata As
String)
'
'   mstrLogFile = vdata
'
'End Property
'
'Public Property Get LogFile() As String
'
'   LogFile = mstrLogFile
'
'End Property

Public Property Let ErrorFile(ByVal vdata As
String)
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
    mstrErrorFile = vdata

End Property
Public Property Let IteratorName(ByVal
vdata As String)

    mstrIteratorName = vdata

End Property

Public Property Get ErrorFile() As String

    ErrorFile = mstrErrorFile

End Property
Public Property Get IteratorName() As
String

    IteratorName = mstrIteratorName

End Property

Public Property Set NodeDB(vdata As
Database)

    Set mdbsDatabase = vdata
    Set mcIterators.NodeDB = vdata

End Property
Public Property Get NodeDB() As Database

    Set NodeDB = mdbsDatabase

End Property

Private Function IsStringEmpty(strToCheck
As String) As Boolean

    IsStringEmpty = (strToCheck =
gstrEmptyString)

End Function

Public Property Let EnabledFlag(ByVal
vdata As Boolean)

    ' The enabled flag must be False for all
global steps.
    ' This check must be made by the global
step class. Only
    ' generic step validations will be carried
out by this
    ' class
    mblnEnabledFlag = vdata

End Property

Public Property Let GlobalFlag(ByVal vdata
As Boolean)

    mblnGlobalFlag = vdata

End Property
Public Property Get EnabledFlag() As
Boolean

    EnabledFlag = mblnEnabledFlag

End Property

Public Property Let ArchivedFlag(ByVal
vdata As Boolean)

    mblnArchivedFlag = vdata

End Property

Public Property Get ArchivedFlag() As
Boolean

    ArchivedFlag = mblnArchivedFlag

End Property

Public Property Get GlobalFlag() As Boolean

    GlobalFlag = mblnGlobalFlag

End Property

Public Sub Add()
    ' Inserts a step record into the database - it
initializes
    ' the necessary properties for the step and
calls InsertStepRec
    ' to do the database work

    On Error GoTo AddErr

    ' A new record would have the deleted_flag
turned off!
    mblnArchivedFlag = False

    Call InsertStepRec

    ' If a new version of a step has been created,
reset the old version info, since
    ' it's already been saved to the db
    If IsNewVersion() Then
        mbIsNewVersion = False
        msOldVersion = gstrEmptyString
    End If

    Exit Sub

AddErr:
    LogErrors Errors
    On Error GoTo 0
    Err.Raise vbObjectError +
errAddStepFailed, _
            mstrModuleName & "Add",
LoadResString(errAddStepFailed)
End Sub
Private Sub InsertStepRec()
    ' Inserts a step record into the database
    ' It first generates the insert statement using
the different
    ' step properties and then executes it

    Dim strInsert As String
    Dim qy As DAO.QueryDef

    On Error GoTo InsertStepRecErr

    ' First check if the database object is valid
    Call CheckDB

    ' Check if the step record is valid
    Call Validate

    If IsNewVersion() Then
        Call UpdOldVersionsArchFlg
    End If

    ' Create a temporary querydef object
    strInsert = "insert into att_steps " & _
        "( workspace_id, step_id, version_no, " &
_
        " step_label, step_file_name, step_text,
start_directory, " & _
        " parent_step_id, parent_version_no,
sequence_no, " & _
        " enabled_flag, step_level, " & _
        " degree_parallelism, execution_mechanism, "
& _
        " failure_details, " & _
        " continuation_criteria, global_flag, " & _
        " archived_flag, " & _
        " output_file_name, error_file_name, " & _
        " iterator_name ) values ( "

    ' log_file_name,

#If USE_JET Then

    strInsert = strInsert & " [w_id], [s_id], [ver_no], "
& _
        " [s_label], [s_file_name], [s_text],
[s_start_dir], " & _
        " [p_step_id], [p_version_no], [seq_no], " & _
        " [enabled], [s_level], [deg_parallelism], " & _
        " [exec_mechanism], [fail_dtls], " & _
        " [cont_criteria], [global], [archived], " & _
        " [output_file], [error_file], " & _
        " [it_name] ) "

    ' [log_file],

    Set qy =
mdbsDatabase.CreateQueryDef(gstrEmptyString,
strInsert)

    ' Call a procedure to execute the Querydef object
    Call AssignParameters(qy)

    qy.Execute dbFailOnError
    qy.Close
#Else

        strInsert = strInsert & Str(mlngWorkspaceId)
& ", " & Str(mlngStepId) & _
        ", " &
mFieldValue.MakeStringFieldValid(mstrVersionNo
)

    ' For fields that may be null, call a function to
determine
    ' the string to be appended to the insert statement
    strInsert = strInsert & ", " &
mFieldValue.MakeStringFieldValid(mstrStepLabel)
    strInsert = strInsert & ", " &
mFieldValue.MakeStringFieldValid(mstrStepTextF
ile)
    strInsert = strInsert & ", " &
mFieldValue.MakeStringFieldValid(mstrStepText)
    strInsert = strInsert & ", " &
mFieldValue.MakeStringFieldValid(mstrStartDir)

    strInsert = strInsert & ", " &
Str(mlngParentStepId) & _
        ", " &
mFieldValue.MakeStringFieldValid(mstrParentVer
sionNo) & _
        ", " & Str(mintSequenceNo) & _
        ", " & Str(mblnEnabledFlag) & ", " &
Str(mintStepLevel)

    strInsert = strInsert & ", " &
mFieldValue.MakeStringFieldValid(mstrDegreePar
allelism)
    strInsert = strInsert & ", " &
Str(mintExecutionMechanism)
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   strInsert = strInsert & ", " &
mFieldValue.MakeStringFieldValid(mstrFai
lureDetails) & _
       ", " & Str(mintContinuationCriteria) &
_
       ", " & Str(mblnGlobalFlag) & _
       ", " & Str(mblnArchivedFlag)

   strInsert = strInsert & ", " &
mFieldValue.MakeStringFieldValid(mstrOu
tputFile)
'   strInsert = strInsert & ", " &
mFieldValue.MakeStringFieldValid(mstrLo
gFile)
   strInsert = strInsert & ", " &
mFieldValue.MakeStringFieldValid(mstrErr
orFile)
   strInsert = strInsert & ", " &
mFieldValue.MakeStringFieldValid(mstrIter
atorName)

   strInsert = strInsert & " ) "

   BugMessage strInsert
   mdbsDatabase.Execute strInsert,
dbFailOnError

#End If

   Exit Sub

InsertStepRecErr:
   mstrSource = mstrModuleName &
"InsertStepRec"
   LogErrors Errors
   On Error GoTo 0
   Err.Raise vbObjectError +
errInsertStepFailed, _
         mstrSource,
LoadResString(errInsertStepFailed)
End Sub
Private Sub UpdOldVersionsArchFlg()
   ' Updates the archived flag on all old
version for the step to True

   Dim sUpdate As String
   Dim qy As DAO.QueryDef

   On Error GoTo
UpdOldVersionsArchFlgErr
   mstrSource = mstrModuleName &
"UpdOldVersionsArchFlg"

#If USE_JET Then

   sUpdate = "update att_steps " & _
      " set archived_flag = True "

   ' Append the Where clause
   sUpdate = sUpdate & " where step_id =
[s_id] " & _
      " and version_no <> [ver_no]"

   Set qy =
mdbsDatabase.CreateQueryDef(gstrEmptyS
tring, sUpdate)

   ' Call a procedure to execute the Querydef
object
   Call AssignParameters(qy)
   qy.Execute dbFailOnError

   If qy.RecordsAffected = 0 Then
      On Error GoTo 0
      Err.Raise vbObjectError +
errModifyStepFailed, _
            mstrSource,
LoadResString(errModifyStepFailed)
   End If

   qy.Close

#Else

   sUpdate = "update att_steps " & _
      " set archived_flag = True "

   sUpdate = sUpdate & " where step_id = " &
Str(mlngStepId) & _
      " and version_no <> " &
mFieldValue.MakeStringFieldValid(mstrVersi
onNo)

   BugMessage sUpdate
   mdbsDatabase.Execute sUpdate,
dbFailOnError
#End If

   Exit Sub

UpdOldVersionsArchFlgErr:
   mstrSource = mstrModuleName &
"UpdOldVersionsArchFlg"
   LogErrors Errors
   On Error GoTo 0
   Err.Raise vbObjectError +
errModifyStepFailed, _
         mstrSource,
LoadResString(errModifyStepFailed)
End Sub
Public Sub InsertIterator(cItRecord As
cIterator)
   ' Inserts the iterator record into the database

   Call cItRecord.Add(mlngStepId,
mstrVersionNo)

End Sub
Public Sub UpdateIterator(cItRecord As
cIterator)
   ' Updates the iterator record in the database

   Call cItRecord.Update(mlngStepId,
mstrVersionNo)

End Sub
Public Sub UpdateIteratorVersion()
   ' Updates the iterator record in the database

   Dim lngIndex As Long
   Dim cTempIt As cIterator

   On Error GoTo UpdateIteratorVersionErr

   For lngIndex = 0 To mcIterators.Count - 1
      ' Increase the array dimension and add the
constraint
      ' to it
      Set cTempIt = mcIterators(lngIndex)

      If cTempIt.IndOperation <> DeleteOp
Then
         ' Set the operation to indicate an insert
         cTempIt.IndOperation = InsertOp
      End If

   Next lngIndex

   Exit Sub

UpdateIteratorVersionErr:
   mstrSource = mstrModuleName &
"UpdateIteratorVersion"
   LogErrors Errors
   On Error GoTo 0
   Err.Raise vbObjectError + errUpdateFailed, _
         mstrSource,
LoadResString(errUpdateFailed)

End Sub
Public Sub AddIterator(cItRecord As cIterator)
   ' Adds the iterator record to the collection of
iterators
   ' for the step

   Call mcIterators.Add(cItRecord)

End Sub
Public Sub AddAllIterators()
   ' Sets the indicator variable for all iterators to
insert

   Dim lngIndex As Long

   For lngIndex = 0 To mcIterators.Count - 1
      mcIterators(lngIndex).Validate
      mcIterators(lngIndex).IndOperation = InsertOp
   Next lngIndex

End Sub

Public Sub LoadIterator(cItRecord As cIterator)
   ' Adds the iterator record to the collection of
iterators
   ' for the step

   Call mcIterators.Load(cItRecord)

End Sub
Public Sub UnloadIterators()
   ' Unloads all iterator records for the step

   Dim lngIndex As Long

   For lngIndex = mcIterators.Count - 1 To 0 Step -
1
      ' Calls the collection method to unload the
node
      ' from the array
      mcIterators.Unload lngIndex
   Next lngIndex

End Sub
Public Sub ModifyIterator(cItRecord As cIterator)
   ' Modifies the iterator record in the collection

   Call mcIterators.Modify(cItRecord)

End Sub
Public Sub DeleteIterator(cItRecord As cIterator)
   ' Deletes the iterator record from the database

   Call cItRecord.Delete(mlngStepId,
mstrVersionNo)

End Sub
Public Sub RemoveIterator(cItRecord As cIterator)
   ' Marks the iterator record in the collection to
   ' indicate a delete

   Call mcIterators.Delete(cItRecord.Position)

End Sub
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
Private Sub AssignParameters(qyExec As
DAO.QueryDef)
   ' Assigns values to the parameters in the
querydef object
   ' The parameter names are cryptic to make
them different
   ' from the actual field names. When the
parameter names
   ' are the same as the field names,
parameters in the
   ' where clause do not get created.

   Dim prmParam As DAO.Parameter

   On Error GoTo AssignParametersErr
   mstrSource = mstrModuleName &
"AssignParameters"

   For Each prmParam In
qyExec.Parameters
      Select Case prmParam.Name
         Case "[w_id]"
            prmParam.Value =
mlngWorkspaceId
         Case "[s_id]"
            prmParam.Value = mlngStepId
         Case "[ver_no]"
            prmParam.Value =
mstrVersionNo
         Case "[s_label]"
            prmParam.Value = mstrStepLabel
         Case "[s_file_name]"
            prmParam.Value =
mstrStepTextFile
         Case "[s_text]"
            prmParam.Value = mstrStepText
         Case "[s_start_dir]"
            prmParam.Value = mstrStartDir
         Case "[p_step_id]"
            prmParam.Value =
mlngParentStepId
         Case "[p_version_no]"
            prmParam.Value =
mstrParentVersionNo
         Case "[seq_no]"
            prmParam.Value =
mintSequenceNo
         Case "[enabled]"
            prmParam.Value =
mblnEnabledFlag
         Case "[s_level]"
            prmParam.Value = mintStepLevel
         Case "[deg_parallelism]"
            prmParam.Value =
mstrDegreeParallelism
         Case "[exec_mechanism]"
            prmParam.Value =
mintExecutionMechanism
         Case "[fail_dtls]"
            prmParam.Value =
mstrFailureDetails
         Case "[cont_criteria]"
            prmParam.Value =
mintContinuationCriteria
         Case "[global]"
            prmParam.Value =
mblnGlobalFlag
         Case "[archived]"
            prmParam.Value =
mblnArchivedFlag
         Case "[output_file]"
            prmParam.Value = mstrOutputFile
'        Case "[log_file]"
'           prmParam.Value = mstrLogFile
         Case "[error_file]"
            prmParam.Value = mstrErrorFile
         Case "[it_name]"
            prmParam.Value = mstrIteratorName
         Case Else
            ' Write the parameter name that is
faulty
            WriteError errInvalidParameter,
mstrSource, _
                prmParam.Name
            On Error GoTo 0
            Err.Raise errInvalidParameter,
mstrSource, _

LoadResString(errInvalidParameter)
      End Select
   Next prmParam

   If qyExec.Parameters("s_id") = 0 Or
StringEmpty(qyExec.Parameters("ver_no"))
Then
      WriteError errInvalidParameter,
mstrSource
      On Error GoTo 0
      Err.Raise errInvalidParameter,
mstrSource,
LoadResString(errInvalidParameter)
   End If

   Exit Sub

AssignParametersErr:

   mstrSource = mstrModuleName &
"AssignParameters"
   Call LogErrors(Errors)
   On Error GoTo 0
   Err.Raise vbObjectError +
errAssignParametersFailed, _
      mstrSource,
LoadResString(errAssignParametersFailed)

End Sub

Public Sub Modify()

   Dim strUpdate As String
   Dim qy As QueryDef

   On Error GoTo ModifyErr
   mstrSource = mstrModuleName & "Modify"

   ' Check if the database object is valid
   Call CheckDB

   ' Check if the step record is valid
   Call Validate

   ' The step_id and version_no will never be
updated -
   ' whenever a step is modified a copy of the
old step will
   ' be created with an incremented version_no

#If USE_JET Then

   strUpdate = "update att_steps " & _
      " set step_label = [s_label] " & _
      " , step_file_name = [s_file_name] " & _
      " , step_text = [s_text] " & _
      " , start_directory = [s_start_dir] " & _
      " , workspace_id = [w_id] " & _
      " , parent_step_id = [p_step_id] " & _
      " , parent_version_no = [p_version_no] "
& _
      " , sequence_no = [seq_no] " & _
      " , step_level = [s_level] " & _
      " , enabled_flag = [enabled] " & _
      " , degree_parallelism = [deg_parallelism] " &
_
      " , execution_mechanism = [exec_mechanism]
" & _
      " , failure_details = [fail_dtls] " & _
      " , continuation_criteria = [cont_criteria] " & _
      " , global_flag = [global] " & _
      " , archived_flag = [archived] " & _
      " , output_file_name = [output_file] " & _
      " , error_file_name = [error_file] " & _
      " , iterator_name = [it_name] "

      ' " , log_file_name = [log_file] " & _

   ' Append the Where clause
   strUpdate = strUpdate & " where step_id = [s_id]
" & _
      " and version_no = [ver_no]"

   Set qy =
mdbsDatabase.CreateQueryDef(gstrEmptyString,
strUpdate)

   ' Call a procedure to execute the Querydef object
   Call AssignParameters(qy)
   qy.Execute dbFailOnError

   If qy.RecordsAffected = 0 Then
      On Error GoTo 0
      Err.Raise vbObjectError +
errModifyStepFailed, _
            mstrSource,
LoadResString(errModifyStepFailed)
   End If

   qy.Close

#Else

   strUpdate = "update att_steps " & _
      " set step_label = "

   ' For fields that may be null, call a function to
determine
   ' the string to be appended to the update statement
   strUpdate = strUpdate &
mFieldValue.MakeStringFieldValid(mstrStepLabel)

   strUpdate = strUpdate & " , step_file_name = " &
mFieldValue.MakeStringFieldValid(mstrStepTextF
ile)
   strUpdate = strUpdate & " , step_text = " &
mFieldValue.MakeStringFieldValid(mstrStepText)
   strUpdate = strUpdate & " , start_directory = " &
mFieldValue.MakeStringFieldValid(mstrStartDir)

   strUpdate = strUpdate & " , workspace_id = " &
Str(mlngWorkspaceId) & _
      " , parent_step_id = " & Str(mlngParentStepId)
& _
      " , parent_version_no = " &
mFieldValue.MakeStringFieldValid(mstrParentVer
sionNo) & _
      " , sequence_no = " & Str(mintSequenceNo) &
_
      " , step_level = " & Str(mintStepLevel) & _
      " , enabled_flag = " & Str(mblnEnabledFlag)
& _
      " , degree_parallelism = " &
mFieldValue.MakeStringFieldValid(mstrDegreePar
allelism) & _
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
        " , execution_mechanism = " &
Str(mintExecutionMechanism) & _
        " , failure_details = " &
mFieldValue.MakeStringFieldValid(mstrFai
lureDetails) & _
        " , continuation_criteria = " &
Str(mintContinuationCriteria) & _
        " , global_flag = " &
Str(mblnGlobalFlag) & _
        " , archived_flag = " &
Str(mblnArchivedFlag) & _
        " , output_file_name = " &
mFieldValue.MakeStringFieldValid(mstrOu
tputFile) & _
        " , error_file_name = " &
mFieldValue.MakeStringFieldValid(mstrErr
orFile) & _
        " , iterator_name = " &
mFieldValue.MakeStringFieldValid(mstrIter
atorName)

'       " , log_file_name = " &
mFieldValue.MakeStringFieldValid(mstrLo
gFile) & _

    strUpdate = strUpdate & " where step_id
= " & Str(mlngStepId) & _
        " and version_no = " &
mFieldValue.MakeStringFieldValid(mstrVe
rsionNo)

    BugMessage strUpdate
    mdbsDatabase.Execute strUpdate,
dbFailOnError
#End If

    Exit Sub

ModifyErr:
    LogErrors Errors
    mstrSource = mstrModuleName &
"Modify"
    On Error GoTo 0
    Err.Raise vbObjectError +
errModifyStepFailed, _
            mstrSource,
LoadResString(errModifyStepFailed)
End Sub
Private Sub CheckDB()
    ' Check if the database object has been
initialized

    If mdbsDatabase Is Nothing Then
        ShowError errInvalidDB
        On Error GoTo 0
        Err.Raise vbObjectError +
errInvalidDB, _
            mstrModuleName,
LoadResString(errInvalidDB)
    End If

End Sub

Public Sub Delete()

    Dim strDelete As String
    Dim qy As DAO.QueryDef

    On Error GoTo DeleteErr

    Call CheckDB

    strDelete = "delete from att_steps " & _
            " where step_id = [s_id] " & _
            " and version_no = [ver_no] "
'    mdbsDatabase.Execute strDelete,
dbFailOnError
    Set qy =
mdbsDatabase.CreateQueryDef(gstrEmptyStri
ng, strDelete)

    Call AssignParameters(qy)
    qy.Execute dbFailOnError

    qy.Close

    Exit Sub

DeleteErr:
    LogErrors Errors
    On Error GoTo 0
    Err.Raise vbObjectError +
errDeleteStepFailed, _
            mstrModuleName & "Delete",
LoadResString(errDeleteStepFailed)
End Sub
Public Property Get DegreeParallelism() As
String

    DegreeParallelism = mstrDegreeParallelism

End Property
Public Property Get Position() As Long

    Position = mlngPosition

End Property

Public Property Let DegreeParallelism(ByVal
vdata As String)

    ' The degree of parallelism must be zero for
all global steps
    ' This check must be made by the global step
class. Only
    ' generic step validations will be carried out
by this
    ' class
    mstrDegreeParallelism = vdata

End Property

Public Property Let
ExecutionMechanism(ByVal vdata As
ExecutionMethod)

    BugAssert vdata = gintExecuteODBC Or
vdata = gintExecuteShell Or vdata =
gintNoOption, _
            "Execution mechanism invalid"
    mintExecutionMechanism = vdata

End Property

Public Property Let FailureDetails(ByVal
vdata As String)

    mstrFailureDetails = vdata

End Property
Public Property Let SequenceNo(ByVal vdata
As Integer)
    mintSequenceNo = vdata
End Property

Public Property Let Position(ByVal vdata As
Long)
    mlngPosition = vdata
End Property
Public Property Let ParentStepId(ByVal vdata As
Long)
    mlngParentStepId = vdata
End Property

Public Property Get SequenceNo() As Integer

    SequenceNo = mintSequenceNo

End Property

Public Property Get StepLevel() As Integer
    StepLevel = mintStepLevel
End Property

Public Property Get ParentVersionNo() As String
    ParentVersionNo = mstrParentVersionNo
End Property

Public Property Let ParentVersionNo(ByVal vdata
As String)
    mstrParentVersionNo = vdata
End Property

Public Property Get ParentStepId() As Long
    ParentStepId = mlngParentStepId
End Property

Public Property Let WorkspaceId(ByVal vdata As
Long)
    mlngWorkspaceId = vdata
End Property

Public Property Let VersionNo(ByVal vdata As
String)
    ' The version number of a step is stored in the x.y
format where
    ' x represents a change to the step as a result of
modifications
    ' to any of the step properties
    ' y represents a change to the step as a result of
modifications
    ' to the sub-steps associated with it. Hence the y-
component
    ' of the version will be incremented when a sub-
step is added,
    ' modified or deleted
    ' x will be referred to throughout this code as the
parent
    ' component of the version and y will be referred
to as the
    ' child component of the version
    ' The version information for a step is maintained
by the
    ' calling function

    mstrVersionNo = vdata

End Property

Public Property Get StepType() As gintStepType

    On Error GoTo StepTypeErr

    If mintStepType = 0 Then
        ' The step type variable has not been initialized
-
        If mblnGlobalFlag Then
            mintStepType = gintGlobalStep
        ElseIf IsStringEmpty(mstrStepText) And _
                IsStringEmpty(mstrStepTextFile) Then
            mintStepType = gintManagerStep
        Else
            mintStepType = gintWorkerStep
        End If
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
    End If

    StepType = mintStepType

    Exit Property

StepTypeErr:
    LogErrors Errors
    mstrSource = mstrModuleName &
"StepType"
    On Error GoTo 0
    Err.Raise vbObjectError +
errGetStepTypeFailed, _
            mstrSource, _

LoadResString(errGetStepTypeFailed)

End Property

Public Property Let StepType(vdata As
gintStepType)

    On Error GoTo StepTypeErr

    Select Case vdata
        Case gintGlobalStep, gintManagerStep,
gintWorkerStep
            mintStepType = vdata

        Case Else
            On Error GoTo 0
            Err.Raise vbObjectError +
errStepTypeInvalid, _
                mstrModuleName &
"StepType",
LoadResString(errStepTypeInvalid)
    End Select
    Exit Property

StepTypeErr:
    LogErrors Errors
    mstrSource = mstrModuleName &
"StepType"
    On Error GoTo 0
    Err.Raise vbObjectError +
errLetStepTypeFailed, _
            mstrSource, _

LoadResString(errLetStepTypeFailed)

End Property

Public Property Get WorkspaceId() As Long
    WorkspaceId = mlngWorkspaceId
End Property

Public Property Get ContinuationCriteria()
As ContinuationCriteria

    ContinuationCriteria =
mintContinuationCriteria

End Property

Public Property Let
ContinuationCriteria(ByVal vdata As
ContinuationCriteria)

    ' The Continuation criteria must be null
for all global steps
    ' and non-null for all manager and worker
steps
    ' These checks will have to be made by
the corresponding
    ' classes - only generic step validations will
be made
    ' by this class
    BugAssert vdata =
gintOnFailureAbortSiblings Or vdata =
gintOnFailureCompleteSiblings _
                Or vdata =
gintOnFailureSkipSiblings Or vdata =
gintOnFailureAbort _
                Or vdata = gintOnFailureContinue
Or vdata = gintOnFailureAsk _
                Or vdata = gintNoOption, _
                "Invalid continuation criteria"
    mintContinuationCriteria = vdata

End Property
Public Property Get ExecutionMechanism() As
ExecutionMethod

    ExecutionMechanism =
mintExecutionMechanism

End Property


Public Property Get FailureDetails() As String

    FailureDetails = mstrFailureDetails

End Property

Public Property Let StepText(ByVal vdata As
String)
    ' Has to be null for manager steps
    ' The check will have to be made by the user
interface or
    ' by the manager step class
    mstrStepText = vdata
End Property
Public Property Let StepLevel(ByVal vdata As
Integer)

    ' The step level must be zero for all global
steps
    ' This check must be made in the global step
class
    mintStepLevel = vdata

End Property
Public Property Get StepText() As String
    StepText = mstrStepText
End Property

Public Property Let StepTextFile(ByVal vdata
As String)
    ' Has to be null for manager steps
    ' The check will have to be made by the user
interface and
    ' by the manager step class
    mstrStepTextFile = vdata
End Property

Public Property Get StepTextFile() As String
    StepTextFile = mstrStepTextFile
End Property

Public Property Let StepLabel(ByVal vdata As
String)
    ' Cannot be null for manager steps
    ' But this check cannot be made here since
we do not know
    ' at this point if the step being created is a
manager
    ' or a worker step
    ' The check will have to be made by the user
interface and
    ' by the manager step class
    mstrStepLabel = vdata
End Property

Public Property Get StepLabel() As String
    StepLabel = mstrStepLabel
End Property

Public Property Let StartDir(ByVal vdata As
String)
    mstrStartDir = vdata
End Property

Public Property Get StartDir() As String
    StartDir = mstrStartDir
End Property

Public Property Get VersionNo() As String
    ' The version number of a step is stored in the x.y
format where
    ' x represents a change to the step as a result of
modifications
    ' to any of the step properties
    ' y represents a change to the step as a result of
modifications
    ' to the sub-steps associated with it. Hence the y-
component
    ' of the version will be incremented when a sub-
step is added,
    ' modified or deleted
    ' x will be referred to throughout this code as the
parent
    ' component of the version and y will be referred
to as the
    ' child component of the version
    ' The version information for a step is maintained
by the
    ' calling function

    VersionNo = mstrVersionNo

End Property

Public Property Get StepId() As Long

    StepId = mlngStepId

End Property
Public Property Get NextStepId() As Long

    Dim lngNextId As Long

    On Error GoTo NextStepIdErr

    ' First check if the database object is valid
    Call CheckDB

    ' Retrieve the next identifier using the sequence
class
    Set mStepSeq = New cSequence
    Set mStepSeq.IdDatabase = mdbsDatabase
    mStepSeq.IdentifierColumn = "step_id"
    lngNextId = mStepSeq.Identifier
    Set mStepSeq = Nothing

    NextStepId = lngNextId
    Exit Property

NextStepIdErr:
    LogErrors Errors
    mstrSource = mstrModuleName & "NextStepId"
    On Error GoTo 0
    Err.Raise vbObjectError + errStepIdGetFailed, _
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
      mstrSource,
LoadResString(errStepIdGetFailed)

End Property
Public Property Let StepId(ByVal vdata As
Long)

   mlngStepId = vdata

End Property

Private Sub Class_Initialize()

   ' Initialize the operation indicator variable
to Query
   ' It will be modified later by the collection
class when
   ' inserts, updates or deletes are performed
   mintOperation = QueryOp
   mbIsNewVersion = False
   msOldVersion = gstrEmptyString

   Set mFieldValue = New cStringSM
   Set mcIterators = New cNodeCollections

End Sub

Private Sub Class_Terminate()

   Set mFieldValue = Nothing
   Set mcIterators = Nothing

End Sub


VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cStepTree"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:       cStepTree.cls
'             Microsoft TPC-H Kit Ver. 1.00
'             Copyright Microsoft, 1999
'             All Rights Reserved
'
'
'  PURPOSE:    Implements step navigation
functions such as determining
'             the child of a step and so on.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

' Used to indicate the source module name
when errors
' are raised by this class
Private Const mstrModuleName As String =
"cStepTree."
Private mstrSource As String

Public StepRecords As cArrSteps
Public Property Get HasChild(Optional
ByVal StepKey As String, _
      Optional ByVal StepId As Long = 0)
As Boolean

   Dim lTemp As Long

   HasChild = False
   StepId = GetStepId(StepKey, StepId)

   For lTemp = 0 To StepRecords.StepCount -
1
      If StepRecords(lTemp).StepType <>
gintGlobalStep And
StepRecords(lTemp).ParentStepId = StepId
Then
         HasChild = True
         Exit For
      End If
   Next lTemp

End Property
Public Property Get ChildStep(Optional ByVal
StepKey As String, _
      Optional ByVal StepId As Long = 0) As
cStep

   Dim lTemp As Long

   Set ChildStep = Nothing
   StepId = GetStepId(StepKey, StepId)

   For lTemp = 0 To StepRecords.StepCount -
1
      If StepRecords(lTemp).StepType <>
gintGlobalStep And
StepRecords(lTemp).ParentStepId = StepId
And _
            StepRecords(lTemp).SequenceNo =
gintMinSequenceNo Then
         Set ChildStep = StepRecords(lTemp)
         Exit For
      End If
   Next lTemp

End Property
Public Property Get NextStep(Optional ByVal
StepKey As String, _
      Optional ByVal StepId As Long = 0) As
cStep

   Dim lTemp As Long
   Dim cChildStep As cStep

   Set NextStep = Nothing
   StepId = GetStepId(StepKey, StepId)
   Set cChildStep =
StepRecords.QueryStep(StepId)

   For lTemp = 0 To StepRecords.StepCount -
1
      If StepRecords(lTemp).StepType <>
gintGlobalStep And _
            StepRecords(lTemp).ParentStepId =
cChildStep.ParentStepId And _
            StepRecords(lTemp).SequenceNo =
cChildStep.SequenceNo + 1 Then
         Set NextStep = StepRecords(lTemp)
         Exit For
      End If
   Next lTemp

End Property
Private Function GetStepId(Optional ByVal
StepKey As String, _
      Optional ByVal StepId As Long = 0) As
Long
   If StepId = 0 Then
      If StringEmpty(StepKey) Then
         Err.Raise vbObjectError +
errMandatoryParameterMissing, _
               mstrModuleName & "GetStepId",
LoadResString(errMandatoryParameterMissin
g)
      Else
         GetStepId = IIf(IsLabel(StepKey), 0,
MakeIdentifierValid(StepKey))
      End If
   Else
      GetStepId = StepId
   End If
End Function
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cStringSM"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:       cStringSM.cls
'             Microsoft TPC-H Kit Ver. 1.00
'             Copyright Microsoft, 1999
'             All Rights Reserved
'
'
'  PURPOSE:    This module contains common
procedures that can be used
'             to manipulate strings
'             It is called StringSM, since String is a
Visual Basic keyword
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

' Used to indicate the source module name when
errors
' are raised by this class
Private mstrSource As String
Private Const mstrModuleName As String =
"cStringSM."

Private mstrText As String

Private Const mstrNullValue = "null"
Private Const mstrSQ = """"
Private Const mstrEnvVarSeparator = "%"
Public Function InsertEnvVariables( _
      Optional ByVal strComString As String) As
String
   ' This function replaces all environment variables
in
   ' the passed in string with their values - they are
   ' enclosed by "%"

   Dim intPos As Integer
   Dim intEndPos As Integer
   Dim strEnvVariable As String
   Dim strValue As String
   Dim strCommand As String

   On Error GoTo InsertEnvVariablesErr
   mstrSource = mstrModuleName &
"InsertEnvVariables"

   ' Initialize the return value of the function to the
   ' passed in command
   If IsStringEmpty(strComString) Then
      strCommand = mstrText
   Else
      strCommand = strComString
   End If

   intPos = InStr(strCommand,
mstrEnvVarSeparator)
   Do While intPos <> 0
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
        ' Extract the environment variable from
the passed
        ' in string
        intEndPos = InStr(intPos + 1,
strCommand, mstrEnvVarSeparator)
        strEnvVariable = Mid(strCommand,
intPos + 1, intEndPos - intPos - 1)

        ' Get the value of the variable and call a
function
        ' to replace the variable with it's value
        strValue = Environ$(strEnvVariable)
        strCommand =
ReplaceSubString(strCommand, _
                mstrEnvVarSeparator &
strEnvVariable & mstrEnvVarSeparator, _
                strValue)

        intPos = InStr(strCommand,
mstrEnvVarSeparator)
    Loop

    InsertEnvVariables = strCommand
    Exit Function

InsertEnvVariablesErr:
    ' Log the error code raised by Visual
Basic
    Call LogErrors(Errors)
    ' Return an empty string
    InsertEnvVariables = gstrEmptyString

End Function
Public Function MakeStringFieldValid( _
        Optional strField As String =
gstrEmptyString) As String
    ' Returns a string that can be appended to
any insert
    ' or modify (sql) statement
    ' If an argument is not passed to this
function, the
    ' default text property is used

    Dim strTemp As String

    On Error GoTo MakeStringFieldValidErr

    If IsStringEmpty(strField) Then
        strTemp = mstrText
    Else
        strTemp = strField
    End If

    ' It checks whether the text is empty
    ' If so, it returns the string, "null"
    If IsStringEmpty(strTemp) Then
        MakeStringFieldValid =
mstrNullValue
    Else
        ' Single-quotes have to be replaced by
two single-quotes,
        ' since a single-quote is the identifier
delimiter
        ' character - call a procedure to do the
replace
        strTemp = ReplaceSubString(strTemp,
mstrSQ, mstrSQ & mstrSQ)

        ' Replace pipe characters with the
corresponding chr function
        strTemp = ReplaceSubString(strTemp,
"|", "' & Chr(124) & "')

        ' Enclose the string in single quotes
        MakeStringFieldValid = mstrSQ &
strTemp & mstrSQ

    End If

    Exit Function

MakeStringFieldValidErr:
    mstrSource = mstrModuleName &
"MakeStringFieldValid"
    LogErrors Errors
    On Error GoTo 0
    Err.Raise vbObjectError +
errMakeFieldValidFailed, _
            mstrSource,
LoadResString(errMakeFieldValidFailed)

End Function
Public Function MakeDateFieldValid( _
        Optional dtmField As Date = gdtmEmpty)
As String
    ' Returns a string that can be appended to
any insert
    ' or modify (sql) statement

    ' Enclose the date in single quotes
    MakeDateFieldValid = mstrSQ & dtmField
& mstrSQ

End Function

Private Function IsStringEmpty(strToCheck
As String) As Boolean

    If strToCheck = gstrEmptyString Then
        IsStringEmpty = True
    Else
        IsStringEmpty = False
    End If

End Function
Public Function ReplaceSubString(ByVal
MainString As String, _
        ByVal ReplaceString As String, _
        ByVal ReplaceWith As String) As String

    ' Replaces all occurrences of ReplaceString
in MainString with ReplaceWith

    Dim intPos As Integer
    Dim strTemp As String

    On Error GoTo ReplaceSubStringErr

    strTemp = MainString

    intPos = InStr(strTemp, ReplaceString)
    Do While intPos <> 0
        strTemp = Left(strTemp, intPos - 1) &
ReplaceWith & _
                Mid(strTemp, intPos +
Len(ReplaceString))
        intPos = InStr(intPos +
Len(ReplaceString) + 1, strTemp,
ReplaceString)
    Loop
    ReplaceSubString = strTemp

    Exit Function

ReplaceSubStringErr:
    Call LogErrors(Errors)
    mstrSource = mstrModuleName &
"ReplaceSubString"
    On Error GoTo 0
    Err.Raise vbObjectError + errParseStringFailed,
_
        mstrSource, _
        LoadResString(errParseStringFailed)

End Function

Public Property Get Text() As String
Attribute Text.VB_UserMemId = 0
    Text = mstrText
End Property

Public Property Let Text(ByVal vdata As String)
    mstrText = vdata
End Property

VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cSubStep"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:        cSubStep.cls
'               Microsoft TPC-H Kit Ver. 1.00
'               Copyright Microsoft, 1999
'               All Rights Reserved
'
'
'  PURPOSE:     This module encapsulates the
properties of sub-steps
'               that are used during the execution of a
workspace.
'  Contact:     Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

' Used to indicate the source module name when
errors
' are raised by this class
Private Const mstrModuleName As String =
"cSubStep"

Private mlngStepId As Long
Private mintRunning As Integer ' Number of
running tasks
Private mintComplete As Integer ' Number of
completed tasks
' The last iterator for this sub-step
Private mcLastIterator As cRunItDetails

Public Function NewIteration(cStepRec As cStep)
As cIterator
    ' Calls a procedure to determine the next iterator
value
    ' for the passed in step - returns the value to be
used
    ' in the iteration.
    ' It updates the instance node with the new
iteration
    ' for the step.

    Dim cItRec As cIterator

    On Error GoTo NewIterationErr

    ' Call a function that will populate an iterator
record
    ' with the iterator values
    Set cItRec = NextIteration(cStepRec)

    ' Initialize the run node with the new iterator
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
    ' values
    If Not mcLastIterator Is Nothing Then
        If cItRec Is Nothing Then
            mcLastIterator.Value =
gstrEmptyString
        Else
            mcLastIterator.Value = cItRec.Value

            ' And if the iterator is a list of values,
then update
            ' the sequence number as well
            If mcLastIterator.IteratorType =
gintValue Then
                mcLastIterator.Sequence =
cItRec.SequenceNo
            End If
        End If
    End If

    Set NewIteration = cItRec
    Set cItRec = Nothing

    Exit Function

NewIterationErr:
    ' Log the error code raised by Visual
Basic
    Call LogErrors(Errors)
    On Error GoTo 0
    Err.Raise vbObjectError +
errIterateFailed, mstrModuleName, _
        LoadResString(errIterateFailed)

End Function
Public Function NextIteration(cStepRec As
cStep) As cIterator

    ' Retrieves the next iterator value for the
passed in step -
    ' returns an iterator record with the new
iterator values

    Dim cItRec As cIterator
    Dim vntIterators As Variant
    Dim lngValue As String

    On Error GoTo NextIterationErr

    vntIterators = cStepRec.Iterators

    If Not mcLastIterator Is Nothing Then
        ' The run node contains the iterator
details
        ' Get the next value for the iterator
        If mcLastIterator.IteratorType =
gintValue Then
            ' Find the next iterator that appears in
the list of
            ' iterator values
            Set cItRec =
NextInSequence(vntIterators,
mcLastIterator.Sequence)
        Else
            lngValue =
CLng(Trim$(mcLastIterator.Value))
            ' Determine whether the new iterator
value falls in the
            ' range between From and To
            If (mcLastIterator.RangeStep > 0
And _
                (mcLastIterator.RangeFrom <=
mcLastIterator.RangeTo) And _
                (mcLastIterator.RangeStep +
lngValue) <= mcLastIterator.RangeTo) Or _
                (mcLastIterator.RangeStep < 0
And _
                (mcLastIterator.RangeFrom >=
mcLastIterator.RangeTo) And _
                (mcLastIterator.RangeStep +
lngValue) >= mcLastIterator.RangeTo) Then
                Set cItRec = New cIterator
                cItRec.Value =
Trim$(CStr(mcLastIterator.RangeStep +
lngValue))
            Else
                Set cItRec = Nothing
            End If
        End If
    Else
        Set cItRec = Nothing
    End If

    Set NextIteration = cItRec
    Exit Function

NextIterationErr:
    ' Log the error code raised by Visual Basic
    Call LogErrors(Errors)
    On Error GoTo 0
    Err.Raise vbObjectError + errIterateFailed,
mstrModuleName, _
        LoadResString(errIterateFailed)

End Function
Public Sub InitializeIt(cPendingStep As cStep,
_
        ColParameters As cArrParameters, _
        Optional vntIterators As Variant)

    ' Initializes the LastIteration structure with
the iterator details for the
    ' passed in step

    On Error GoTo InitializeItErr

    If IsMissing(vntIterators) Then
        vntIterators = cPendingStep.Iterators
    End If

    If IsArray(vntIterators) And Not
IsEmpty(vntIterators) Then
        mcLastIterator.IteratorName =
cPendingStep.IteratorName
        If
vntIterators(LBound(vntIterators)).IteratorTyp
e = _
                gintValue Then
            mcLastIterator.IteratorType =
gintValue
            ' Since the sequence numbers begin at 0
            mcLastIterator.Sequence =
gintMinIteratorSequence - 1
        Else
            mcLastIterator.IteratorType = gintFrom
            Call InitializeItRange(vntIterators,
cPendingStep.WorkspaceId, _
                ColParameters)
        End If
    Else
        Set mcLastIterator = Nothing
    End If

    Exit Sub

InitializeItErr:
    ' Log the error code raised by Visual Basic
    Call LogErrors(Errors)
    On Error GoTo 0
    Err.Raise vbObjectError + errIterateFailed,
mstrModuleName, _
        LoadResString(errIterateFailed)

End Sub

Private Sub InitializeItRange(vntIterators As
Variant, ByVal lWorkspace As Long, _
        ColParameters As cArrParameters)

    ' Initializes the LastIteration structure for range
iterators from the
    ' passed in variant containing the iterator records

    Dim lngIndex As Long
    Dim cItRec As cIterator

    On Error GoTo InitializeItRangeErr

    If IsArray(vntIterators) And Not
IsEmpty(vntIterators) Then

        ' Check if the iterator range has been
completely initialized
        RangeComplete (vntIterators)

        ' Initialize the Run node with the values for the
From,
        ' To and Step boundaries
        For lngIndex = LBound(vntIterators) To
UBound(vntIterators)
            Set cItRec = vntIterators(lngIndex)
            Select Case cItRec.IteratorType
                Case gintFrom
                    mcLastIterator.RangeFrom =
SubstituteParameters(cItRec.Value, lWorkspace,
WspParameters:=ColParameters)
                Case gintTo
                    mcLastIterator.RangeTo =
SubstituteParameters(cItRec.Value, lWorkspace,
WspParameters:=ColParameters)
                Case gintStep
                    mcLastIterator.RangeStep =
SubstituteParameters(cItRec.Value, lWorkspace,
WspParameters:=ColParameters)
                Case Else
                    On Error GoTo 0
                    Err.Raise vbObjectError +
errTypeInvalid, mstrModuleName, _
                        LoadResString(errTypeInvalid)
            End Select
        Next lngIndex

        mcLastIterator.Value =
Trim$(CStr(mcLastIterator.RangeFrom -
mcLastIterator.RangeStep))
    End If

    Exit Sub

InitializeItRangeErr:
    ' Log the error code raised by Visual Basic
    Call LogErrors(Errors)
    On Error GoTo 0
    Err.Raise vbObjectError + errIterateFailed,
mstrModuleName, _
        LoadResString(errIterateFailed)

End Sub
Private Function NextInSequence(vntIterators As
Variant, _
        lngOldSequence As Long) As cIterator

    Dim lngIndex As Long
    Dim cItRec As cIterator
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   On Error GoTo NextInSequenceErr

   If IsArray(vntIterators) And Not
IsEmpty(vntIterators) Then
      For lngIndex = LBound(vntIterators)
To UBound(vntIterators)
         Set cItRec = vntIterators(lngIndex)
         If cItRec.IteratorType <> gintValue
Then
            On Error GoTo 0
            Err.Raise vbObjectError +
errTypeInvalid, mstrModuleName, _

LoadResString(errTypeInvalid)
         End If
         If cItRec.SequenceNo =
lngOldSequence + 1 Then
            Exit For
         End If

      Next lngIndex

      If cItRec.SequenceNo <>
lngOldSequence + 1 Then
         Set cItRec = Nothing
      End If
   Else
      Set cItRec = Nothing
   End If

   Set NextInSequence = cItRec

   Exit Function

NextInSequenceErr:
   ' Log the error code raised by Visual
Basic
   Call LogErrors(Errors)
   On Error GoTo 0
   Err.Raise vbObjectError +
errIterateFailed, mstrModuleName, _
      LoadResString(errIterateFailed)

End Function

Public Property Get LastIterator() As
cRunItDetails

   Set LastIterator = mcLastIterator

End Property
Public Property Set LastIterator(vdata As
cRunItDetails)

   Set mcLastIterator = vdata

End Property

Public Property Get TasksRunning() As
Integer

   TasksRunning = mintRunning

End Property

Public Property Let TasksRunning(ByVal
vdata As Integer)

   mintRunning = vdata

End Property

Public Property Get TasksComplete() As
Integer

   TasksComplete = mintComplete

End Property
Public Property Let TasksComplete(ByVal
vdata As Integer)

   mintComplete = vdata

End Property
Public Property Get StepId() As Long

   StepId = mlngStepId

End Property
Public Property Let StepId(ByVal vdata As
Long)

   mlngStepId = vdata

End Property
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cSubSteps"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:       cSubSteps.cls
'             Microsoft TPC-H Kit Ver. 1.00
'             Copyright Microsoft, 1999
'             All Rights Reserved
'
'
'  PURPOSE:    This module provides a type-
safe wrapper around cVector to
'             implement a collection of cSubStep
objects.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

Private mcSubSteps As cVector

Public Sub Add(ByVal objItem As cSubStep)

   mcSubSteps.Add objItem

End Sub


Public Sub Clear()

   mcSubSteps.Clear

End Sub


Public Function Count() As Long

   Count = mcSubSteps.Count

End Function


Public Function Delete(ByVal lngDelete As
Long) As cSubStep

   Set Delete = mcSubSteps.Delete(lngDelete)

End Function

Public Property Get Item(ByVal Position As Long)
As cSubStep
Attribute Item.VB_UserMemId = 0

   Set Item = mcSubSteps.Item(Position)

End Property

Private Sub Class_Initialize()

   Set mcSubSteps = New cVector

End Sub

Private Sub Class_Terminate()

   Set mcSubSteps = Nothing

End Sub


VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cTermProcess"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:       cTermProcess.cls
'             Microsoft TPC-H Kit Ver. 1.00
'             Copyright Microsoft, 1999
'             All Rights Reserved
'
'
'  PURPOSE:    This module raises an event if a
completed step exists.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

Private WithEvents moTimer As cTimerSM
Attribute moTimer.VB_VarHelpID = -1
Private bTermProcessExists As Boolean
Public Event TermProcessExists()

Public Sub ProcessTerminated()

   bTermProcessExists = True
   moTimer.Enabled = True

End Sub

Private Sub Class_Initialize()

   bTermProcessExists = False

   Set moTimer = New cTimerSM
   moTimer.Enabled = False

End Sub

Private Sub Class_Terminate()

   Set moTimer = Nothing

End Sub

Private Sub moTimer_Timer()

   On Error GoTo moTimer_TimerErr
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   If bTermProcessExists Then
      RaiseEvent TermProcessExists
   End If

   moTimer.Enabled = False
   bTermProcessExists = False

   Exit Sub

moTimer_TimerErr:
   LogErrors Errors

End Sub

VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cTermStep"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:       cTermStep.cls
'             Microsoft TPC-H Kit Ver. 1.00
'             Copyright Microsoft, 1999
'             All Rights Reserved
'
'
'  PURPOSE:    This module encapsulates
the properties of steps that
'             have completed execution such as
status and time of completion.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

Public TimeComplete As Currency
Public Index As Long
Public InstanceId As Long
Public ExecutionStatus As InstanceStatus

VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cTermSteps"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:       cTermSteps.cls
'             Microsoft TPC-H Kit Ver. 1.00
'             Copyright Microsoft, 1999
'             All Rights Reserved
'
'
'  PURPOSE:    This module provides a
type-safe wrapper around cVector to
'             implement a collection of
cTermStep objects. Raises an
'             event if a step that has completed
execution exists.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

Private mcTermSteps As cVector
Private WithEvents moTimer As cTimerSM
Attribute moTimer.VB_VarHelpID = -1
Public Event TermStepExists(cStepDetails
As cTermStep)
Public Sub Add(ByVal citem As cTermStep)

   Call mcTermSteps.Add(citem)
   moTimer.Enabled = True

End Sub

Public Sub Clear()

   mcTermSteps.Clear

End Sub

Public Function Delete()

   Call mcTermSteps.Delete(0)
   ' Disable the timer if there are no more
pending events
   If mcTermSteps.Count = 0 Then
moTimer.Enabled = False

End Function

Public Property Get Item(ByVal Position As
Long) As cTermStep

   Set Item = mcTermSteps(Position)

End Property

Public Function Count() As Long

   Count = mcTermSteps.Count

End Function

Private Sub Class_Initialize()

   Set mcTermSteps = New cVector

   Set moTimer = New cTimerSM
   moTimer.Enabled = False

End Sub

Private Sub Class_Terminate()

   Set mcTermSteps = Nothing
   Set moTimer = Nothing

End Sub

Private Sub moTimer_Timer()

   On Error GoTo moTimer_TimerErr

   If mcTermSteps.Count > 0 Then
      ' Since items are appended to the end of
the array
      RaiseEvent
TermStepExists(mcTermSteps(0))
   Else
      moTimer.Enabled = False
   End If
   Exit Sub

moTimer_TimerErr:
   LogErrors Errors

End Sub
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cTimerSM"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:       cTimer.cls
'             Microsoft TPC-H Kit Ver. 1.00
'             Copyright Microsoft, 1999
'             All Rights Reserved
'
'
'  PURPOSE:    This module implements a timer.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'

Option Explicit

Public Event Timer()

Private Const mnDefaultInterval As Long = 1

Private mnTimerID As Long
Private mnInterval As Long
Private mfEnabled As Boolean

Public Property Get Interval() As Long
   Interval = mnInterval
End Property
Public Property Let Interval(Value As Long)
   If mnInterval <> Value Then
      mnInterval = Value
      If mfEnabled Then
         SetInterval mnInterval, mnTimerID
      End If
   End If
End Property

Public Property Get Enabled() As Boolean
   Enabled = mfEnabled
End Property
Public Property Let Enabled(Value As Boolean)
   If mfEnabled <> Value Then
      If Value Then
         mnTimerID = StartTimer(mnInterval)
         If mnTimerID <> 0 Then
            mfEnabled = True
            'Storing Me in the global would add a
reference to Me, which
            '  would prevent Me from being released,
which in turn would
            '  prevent my Class_Terminate code from
running. To prevent
            '  this, I store a "soft reference" - the
collection holds a
            '  pointer to me without incrementing my
reference count.
            gcTimerObjects.Add ObjPtr(Me),
Str$(mnTimerID)
         End If
      Else
         StopTimer mnTimerID
         mfEnabled = False
         gcTimerObjects.Remove Str$(mnTimerID)
      End If
   End If
End Property

Private Sub Class_Initialize()
   If gcTimerObjects Is Nothing Then Set
gcTimerObjects = New Collection
   mnInterval = mnDefaultInterval
End Sub

Private Sub Class_Terminate()
   Enabled = False
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
End Sub

Friend Sub Tick()
   RaiseEvent Timer
End Sub
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cVBErrorsSM"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
Attribute VB_Ext_KEY =
"SavedWithClassBuilder" ,"Yes"
Attribute VB_Ext_KEY = "Top_Level"
,"Yes"
'  FILE:       cVBErrors.cls
'             Microsoft TPC-H Kit Ver. 1.00
'             Copyright Microsoft, 1999
'             All Rights Reserved
'
'
'  PURPOSE:     This module encapsulates
the handling of Visual Basic errors.
'             This module does not do any error
handling - any error handler
'             will erase the errors object!
'  Contact:     Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

' The Execute class exposes a method,
WriteError through which we can write to
the
' error log that is currently being used by the
Execute object. Store a reference to
' Execute object locally.
Private mcExecObjRef As
EXECUTEDLLLib.Execute
Public Sub WriteError(ByVal ErrorCode As
errErrorConstants, _
       Optional ByVal ErrorSource As String
= gstrEmptyString, _
       Optional ByVal OptArgs As String =
gstrEmptyString)

   Dim sError As String

   sError = "StepMaster Error:" &
ErrorCode & vbCrLf &
LoadResString(ErrorCode) & vbCrLf

   If Not StringEmpty(ErrorSource) Then
      sError = sError & "(Source: " &
ErrorSource & ")" & vbCrLf
   End If
   sError = sError & OptArgs

   Call LogMessage(sError)

End Sub
Private Function InitErrorString() As String
   ' Initializes a string with all the properties
of the
   ' Err object

   Dim strError As String
   Dim errCode As Long

   If Err.Number = 0 Then
      InitErrorString = gstrEmptyString
   Else
   With Err
      If Err.Number > vbObjectError And
Err.Number < (vbObjectError + 65536) Then
         errCode = .Number - vbObjectError
      Else
         errCode = .Number
      End If
      strError = "Error #: " & errCode &
vbCrLf
      strError = strError & "Description: " &
.Description & vbCrLf
      strError = strError & "Source: " &
Err.Source & vbCrLf
   End With

   Debug.Print strError
   InitErrorString = strError
   End If

End Function
Public Sub LogVBErrors()

   Dim strErr As String

   strErr = InitErrorString

   On Error GoTo LogVBErrorsErr

   If Not StringEmpty(strErr) Then
      ' Write an error using the WriteError
method of the Execute object.
      If Not mcExecObjRef Is Nothing Then
         mcExecObjRef.WriteError strErr
      Else
         WriteMessage strErr
      End If
   End If

   Err.Clear

   Exit Sub

LogVBErrorsErr:
   Call LogErrors(Errors)
   ' Since write to the error file for the step has
failed, write to the project log
   Call WriteMessage(strErr)

End Sub
Public Sub DisplayErrors()

   Dim strErr As String

   strErr = InitErrorString

   If Not StringEmpty(strErr) Then
      ' Display the error message
      MsgBox strErr
   End If

   Err.Clear

End Sub
Public Sub LogMessage(strMsg As String)

   On Error GoTo LogMessageErr

   ' Write an error using the WriteError method
of the Execute object.
   If Not mcExecObjRef Is Nothing Then
      mcExecObjRef.WriteError strMsg
   Else
      WriteMessage strMsg
   End If
   Exit Sub

LogMessageErr:
   Call LogErrors(Errors)
   ' Since write to the error file for the step has
failed, write to the project log
   Call WriteMessage(strMsg)

End Sub
Public Property Set ErrorFile(vdata As
EXECUTEDLLLib.Execute)

   Set mcExecObjRef = vdata

End Property
Private Sub Class_Terminate()

   Set mcExecObjRef = Nothing

End Sub
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cVector"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:       cVector.cls
'             Microsoft TPC-H Kit Ver. 1.00
'             Copyright Microsoft, 1999
'             All Rights Reserved
'
'
'  PURPOSE:     This class implements an array of
objects.
'  Contact:     Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

' Used to indicate the source module name when
errors
' are raised by this class
Private mstrSource As String
Private Const mstrModuleName As String =
"cVector."

' Array counter
Private mlngCount As Long
Private mcarrItems() As Object

Public Sub Add(ByVal objItem As Object)
   ' Adds the passed in Object variable to the array

   On Error GoTo AddErr

   ReDim Preserve mcarrItems(mlngCount)

   ' Set the newly added element in the array to the
   ' passed in variable
   Set mcarrItems(mlngCount) = objItem
   mlngCount = mlngCount + 1

   Exit Sub

AddErr:
   LogErrors Errors
   gstrSource = mstrModuleName & "Add"
   On Error GoTo 0
   Err.Raise vbObjectError + errLoadInArrayFailed,
_
         mstrSource, _
         LoadResString(errLoadInArrayFailed)
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
End Sub
Public Sub Clear()

   ' Clear the array
   ReDim mcarrItems(0)
   mlngCount = 0

End Sub

Public Function Delete(ByVal lngDelete As
Long) As Object

   Dim lngIndex As Long

   On Error GoTo DeleteErr

   If lngDelete < (mlngCount - 1) Then

      ' We want to maintain the order of all
items in the
      ' array - so move all remaining
elements in the array
      ' up by 1
      For lngIndex = lngDelete To
mlngCount - 2
         MoveDown lngIndex
      Next lngIndex

   End If

   ' Return the deleted node
   Set Delete = mcarrItems(mlngCount - 1)

   ' Delete the last Node from the array
   mlngCount = mlngCount - 1
   If mlngCount > 0 Then
      ReDim Preserve mcarrItems(0 To
mlngCount - 1)
   Else
      ReDim mcarrItems(0)
   End If

   Exit Function

DeleteErr:
   LogErrors Errors
   mstrSource = mstrModuleName &
"Delete"
   On Error GoTo 0
   Err.Raise vbObjectError +
errDeleteArrayElementFailed, _
         mstrSource, _

LoadResString(errDeleteArrayElementFaile
d)

End Function
Public Property Get Item(ByVal Position As
Long) As Object
Attribute Item.VB_UserMemId = 0

   ' Returns the element at the passed in
position in the array
   If Position >= 0 And Position <
mlngCount Then
      Set Item = mcarrItems(Position)
   Else
      On Error GoTo 0
      Err.Raise vbObjectError +
errItemDoesNotExist, mstrSource, _

LoadResString(errItemDoesNotExist)
   End If
End Property
Public Property Set Item(ByVal Position As
Long, _
      ByVal Value As Object)

   ' Returns the element at the passed in
position in the array
   If Position >= 0 Then
      ' If the passed in position is outside the
array
      ' bounds, then resize the array
      If Position >= mlngCount Then
         ReDim Preserve mcarrItems(Position)
         mlngCount = Position + 1
      End If

      ' Set the newly added element in the array
to the
      ' passed in variable
      Set mcarrItems(Position) = Value
   Else
      On Error GoTo 0
      Err.Raise vbObjectError +
errItemDoesNotExist, mstrSource, _
         LoadResString(errItemDoesNotExist)
   End If

End Property
Public Sub MoveUp(ByVal Position As Long)
   ' Moves the element at the passed in position
up by 1

   Dim cTemp As Object

   If Position > 0 And Position < mlngCount
Then
      Set cTemp = mcarrItems(Position)

      Set mcarrItems(Position) =
mcarrItems(Position - 1)
      Set mcarrItems(Position - 1) = cTemp
   End If

End Sub
Public Sub MoveDown(ByVal Position As
Long)
   ' Moves the element at the passed in position
down by 1

   Dim cTemp As Object

   If Position >= 0 And Position < mlngCount -
1 Then
      Set cTemp = mcarrItems(Position)

      Set mcarrItems(Position) =
mcarrItems(Position + 1)
      Set mcarrItems(Position + 1) = cTemp
   End If

End Sub

Public Function Count() As Long

   Count = mlngCount

End Function


Private Sub Class_Initialize()

   mlngCount = 0

End Sub

Private Sub Class_Terminate()

   Call Clear

End Sub

VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cVectorLng"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:       cVectorLng.cls
'             Microsoft TPC-H Kit Ver. 1.00
'             Copyright Microsoft, 1999
'             All Rights Reserved
'
'
'  PURPOSE:    This class implements an array of
longs.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

' Used to indicate the source module name when
errors
' are raised by this class
Private mstrSource As String
Private Const mstrModuleName As String =
"cVectorLng."

' Array counter
Private mlngCount As Long
Private mcarrItems() As Long

Public Sub Add(ByVal lngItem As Long)
   ' Adds the passed in long variable to the array

   On Error GoTo AddErr

   ReDim Preserve mcarrItems(mlngCount)

   ' Set the newly added element in the array to the
   ' passed in variable
   mcarrItems(mlngCount) = lngItem
   mlngCount = mlngCount + 1

   Exit Sub

AddErr:
   LogErrors Errors
   gstrSource = mstrModuleName & "Add"
   On Error GoTo 0
   Err.Raise vbObjectError + errLoadInArrayFailed,
_
         mstrSource, _
         LoadResString(errLoadInArrayFailed)

End Sub
Public Sub Clear()

   ' Clear the array
   ReDim mcarrItems(0)

End Sub

Public Sub Delete(Optional ByVal Position As
Long = -1, _
      Optional ByVal Item As Long = -1)
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   ' The user can opt to delete either a
specific item in
   ' the list or the item at a specified position.
If no
   ' parameters are passed in, we delete the
element at
   ' position 0!

   Dim lngDelete As Long
   Dim lngIndex As Long

   On Error GoTo DeleteErr

   If Position = -1 Then
      ' Since we can never store an element at
position -1,
      ' we can be sure that the user is trying
to delete
      ' a given item
      lngDelete = Find(Item)
   Else
      lngDelete = Position
   End If

   If lngDelete < (mlngCount - 1) Then

      ' We want to maintain the order of all
items in the
      ' array - so move all remaining
elements in the array
      ' up by 1
      For lngIndex = lngDelete To
mlngCount - 2
         MoveDown lngIndex
      Next lngIndex

   End If

   ' Delete the last Node from the array
   mlngCount = mlngCount - 1
   If mlngCount > 0 Then
      ReDim Preserve mcarrItems(0 To
mlngCount - 1)
   Else
      ReDim mcarrItems(0)
   End If

   Exit Sub

DeleteErr:
   LogErrors Errors
   mstrSource = mstrModuleName &
"Delete"
   On Error GoTo 0
   Err.Raise vbObjectError +
errDeleteArrayElementFailed, _
         mstrSource, _

LoadResString(errDeleteArrayElementFaile
d)

End Sub
Public Function Find(ByVal Item As Long)
As Long

   ' Returns the position at which the passed
in value occurs
   ' in the array

   Dim lngIndex As Long

   On Error GoTo FindErr

   ' Find the element in the array to be
deleted
   For lngIndex = 0 To mlngCount - 1

      If mcarrItems(lngIndex) = Item Then
         Find = lngIndex
         Exit Function
      End If

   Next lngIndex

   Find = -1

   Exit Function

FindErr:
   LogErrors Errors
   mstrSource = mstrModuleName & "Find"
   On Error GoTo 0
   Err.Raise vbObjectError +
errItemNotFound, mstrSource, _
      LoadResString(errItemNotFound)

End Function
Public Property Get Item(ByVal Position As
Long) As Long
Attribute Item.VB_UserMemId = 0

   ' Returns the element at the passed in
position in the array
   If Position >= 0 And Position < mlngCount
Then
      Item = mcarrItems(Position)
   Else
      On Error GoTo 0
      Err.Raise vbObjectError +
errItemDoesNotExist, mstrSource, _
         LoadResString(errItemDoesNotExist)
   End If

End Property
Public Property Let Item(ByVal Position As
Long, _
      ByVal Value As Long)

   ' Returns the element at the passed in
position in the array
   If Position >= 0 Then
      ' If the passed in position is outside the
array
      ' bounds, then resize the array
      If Position >= mlngCount Then
         ReDim Preserve mcarrItems(Position)
         mlngCount = Position + 1
      End If

      ' Set the newly added element in the array
to the
      ' passed in variable
      mcarrItems(Position) = Value
   Else
      On Error GoTo 0
      Err.Raise vbObjectError +
errItemDoesNotExist, mstrSource, _
         LoadResString(errItemDoesNotExist)
   End If

End Property
Public Sub MoveUp(ByVal Position As Long)
   ' Moves the element at the passed in position
up by 1

   Dim lngTemp As Long

   If Position > 0 And Position < mlngCount
Then
      lngTemp = mcarrItems(Position)

      mcarrItems(Position) = mcarrItems(Position -
1)
      mcarrItems(Position - 1) = lngTemp
   End If

End Sub
Public Sub MoveDown(ByVal Position As Long)
   ' Moves the element at the passed in position
down by 1

   Dim lngTemp As Long

   If Position >= 0 And Position < mlngCount - 1
Then
      lngTemp = mcarrItems(Position)

      mcarrItems(Position) = mcarrItems(Position +
1)
      mcarrItems(Position + 1) = lngTemp
   End If

End Sub

Public Function Count() As Long

   Count = mlngCount

End Function


Private Sub Class_Initialize()

   mlngCount = 0

End Sub


Private Sub Class_Terminate()

   Call Clear

End Sub

VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cVectorStr"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:       cVectorStr.cls
'             Microsoft TPC-H Kit Ver. 1.00
'             Copyright Microsoft, 1999
'             All Rights Reserved
'
'
'  PURPOSE:    This class implements an array of
strings.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

' Used to indicate the source module name when
errors
' are raised by this class
Private mstrSource As String
Private Const mstrModuleName As String =
"cVectorStr."

' Array counter
Private mlngCount As Long
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
Private mcarrItems() As String

Public Sub Add(ByVal strItem As String)
   ' Adds the passed in string variable to the
array

   On Error GoTo AddErr

   ReDim Preserve mcarrItems(mlngCount)

   ' Set the newly added element in the array
to the
   ' passed in variable
   mcarrItems(mlngCount) = strItem
   mlngCount = mlngCount + 1

   Exit Sub

AddErr:
   Call LogErrors(Errors)
   gstrSource = mstrModuleName & "Add"
   On Error GoTo 0
   Err.Raise vbObjectError +
errLoadInArrayFailed, _
         mstrSource, _

LoadResString(errLoadInArrayFailed)

End Sub
Public Sub Clear()

   ' Clear the array
   ReDim mcarrItems(0)

End Sub

Public Sub Delete(Optional ByVal Position
As Long = -1, _
      Optional ByVal Item As String = -1)
   ' The user can opt to delete either a
specific item in
   ' the list or the item at a specified position.
If no
   ' parameters are passed in, we delete the
element at
   ' position 0!

   Dim lngDelete As Long
   Dim lngIndex As Long

   On Error GoTo DeleteErr
   mstrSource = mstrModuleName &
"Delete"

   If Position = -1 Then
      ' Since we can never store an element at
position -1,
      ' we can be sure that the user is trying
to delete
      ' a given item
      lngDelete = Find(Item)
   Else
      lngDelete = Position
   End If

   If lngDelete < (mlngCount - 1) Then

      ' We want to maintain the order of all
items in the
      ' array - so move all remaining
elements in the array
      ' up by 1
      For lngIndex = lngDelete To
mlngCount - 2
         MoveDown lngIndex
      Next lngIndex

   End If

   ' Delete the last Node from the array
   mlngCount = mlngCount - 1
   If mlngCount > 0 Then
      ReDim Preserve mcarrItems(0 To
mlngCount - 1)
   Else
      ReDim mcarrItems(0)
   End If

   Exit Sub

DeleteErr:
   Call LogErrors(Errors)
   mstrSource = mstrModuleName & "Delete"
   On Error GoTo 0
   Err.Raise vbObjectError +
errDeleteArrayElementFailed, _
         mstrSource, _

LoadResString(errDeleteArrayElementFailed)

End Sub
Public Function Find(ByVal Item As String)
As Long

   ' Returns the position at which the passed in
value occurs
   ' in the array

   Dim lngIndex As Long

   On Error GoTo FindErr
   mstrSource = mstrModuleName & "Find"

   ' Find the element in the array to be deleted
   For lngIndex = 0 To mlngCount - 1

      If mcarrItems(lngIndex) = Item Then
         Find = lngIndex
         Exit Function
      End If

   Next lngIndex

   Find = -1

   Exit Function

FindErr:
   Call LogErrors(Errors)
   mstrSource = mstrModuleName & "Find"
   On Error GoTo 0
   Err.Raise vbObjectError +
errItemNotFound, mstrSource, _
      LoadResString(errItemNotFound)

End Function
Public Property Get Item(ByVal Position As
Long) As String
Attribute Item.VB_UserMemId = 0

   ' Returns the element at the passed in
position in the array
   If Position >= 0 And Position < mlngCount
Then
      Item = mcarrItems(Position)
   Else
      On Error GoTo 0
      Err.Raise vbObjectError +
errItemDoesNotExist, mstrSource, _
         LoadResString(errItemDoesNotExist)
   End If

End Property
Public Property Let Item(ByVal Position As Long,
_
      ByVal Value As String)

   ' Returns the element at the passed in position in
the array
   If Position >= 0 Then
      ' If the passed in position is outside the array
      ' bounds, then resize the array
      If Position >= mlngCount Then
         ReDim Preserve mcarrItems(Position)
         mlngCount = Position + 1
      End If

      ' Set the newly added element in the array to
the
      ' passed in variable
      mcarrItems(Position) = Value
   Else
      On Error GoTo 0
      Err.Raise vbObjectError +
errItemDoesNotExist, mstrSource, _
         LoadResString(errItemDoesNotExist)
   End If

End Property
Public Sub MoveUp(ByVal Position As Long)
   ' Moves the element at the passed in position up
by 1

   Dim strTemp As String

   If Position > 0 And Position < mlngCount Then
      strTemp = mcarrItems(Position)

      mcarrItems(Position) = mcarrItems(Position -
1)
      mcarrItems(Position - 1) = strTemp
   End If

End Sub
Public Sub MoveDown(ByVal Position As Long)
   ' Moves the element at the passed in position
down by 1

   Dim strTemp As String

   If Position >= 0 And Position < mlngCount - 1
Then
      strTemp = mcarrItems(Position)

      mcarrItems(Position) = mcarrItems(Position +
1)
      mcarrItems(Position + 1) = strTemp
   End If

End Sub

Public Function Count() As Long

   Count = mlngCount

End Function


Private Sub Class_Initialize()

   mlngCount = 0

End Sub
Private Sub Class_Terminate()
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   Call Clear

End Sub


VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cWorker"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:       cWorker.cls
'              Microsoft TPC-H Kit Ver. 1.00
'              Copyright Microsoft, 1999
'              All Rights Reserved
'
'
'  PURPOSE:    Encapsulates the properties
and methods of a worker step.
'              Implements the cStep class -
carries out initializations
'              and validations that are specific to
worker steps.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

Implements cStep

' Object variable to keep the step reference
in
Private mcStep As cStep

' Used to indicate the source module name
when errors
' are raised by this class
Private mstrSource As String
Private Const mstrModuleName As String =
"cWorker."
Private Sub cStep_AddAllIterators()

   Call mcStep.AddAllIterators

End Sub

Private Property Let cStep_StartDir(ByVal
RHS As String)

   mcStep.StartDir = RHS

End Property

Private Property Get cStep_StartDir() As
String

   cStep_StartDir = mcStep.StartDir

End Property

Private Property Set cStep_NodeDB(RHS
As DAO.Database)

   Set mcStep.NodeDB = RHS

End Property

Private Property Get cStep_NodeDB() As
DAO.Database

   Set cStep_NodeDB = mcStep.NodeDB

End Property

Private Function cStep_IncVersionY() As
String

   cStep_IncVersionY = mcStep.IncVersionY

End Function
Private Function cStep_IsNewVersion() As
Boolean
   cStep_IsNewVersion =
mcStep.IsNewVersion
End Function
Private Function cStep_OldVersionNo() As
String
   cStep_OldVersionNo =
mcStep.OldVersionNo
End Function

Private Function cStep_IncVersionX() As
String

   cStep_IncVersionX = mcStep.IncVersionX

End Function
Private Sub cStep_UpdateIteratorVersion()

   Call mcStep.UpdateIteratorVersion

End Sub

Private Function cStep_IteratorCount() As
Long

   cStep_IteratorCount = mcStep.IteratorCount

End Function

Private Sub cStep_UnloadIterators()

   Call mcStep.UnloadIterators

End Sub

Private Sub cStep_SaveIterators()

   Call mcStep.SaveIterators

End Sub
Private Property Get cStep_IteratorName() As
String

   cStep_IteratorName = mcStep.IteratorName

End Property
Private Property Let
cStep_IteratorName(ByVal RHS As String)

   mcStep.IteratorName = RHS

End Property

Private Sub cStep_LoadIterator(cItRecord As
cIterator)

   Call mcStep.LoadIterator(cItRecord)

End Sub
Private Sub cStep_DeleteIterator(cItRecord As
cIterator)

   Call mcStep.DeleteIterator(cItRecord)

End Sub

Private Sub cStep_InsertIterator(cItRecord As
cIterator)

   Call mcStep.InsertIterator(cItRecord)

End Sub
Private Function cStep_Iterators() As Variant

   cStep_Iterators = mcStep.Iterators

End Function
Private Sub cStep_ModifyIterator(cItRecord As
cIterator)

   Call mcStep.ModifyIterator(cItRecord)

End Sub
Private Sub cStep_RemoveIterator(cItRecord As
cIterator)

   Call mcStep.RemoveIterator(cItRecord)

End Sub
Private Sub cStep_UpdateIterator(cItRecord As
cIterator)

   Call mcStep.UpdateIterator(cItRecord)

End Sub
Private Sub cStep_AddIterator(cItRecord As
cIterator)

   Call mcStep.AddIterator(cItRecord)

End Sub

Private Property Let cStep_Position(ByVal RHS As
Long)

   mcStep.Position = RHS

End Property

Private Property Get cStep_Position() As Long

   cStep_Position = mcStep.Position

End Property

Private Function cStep_Clone(Optional cCloneStep
As cStep) As cStep

   Dim cNewWorker As cWorker

   Set cNewWorker = New cWorker
   Set cStep_Clone = mcStep.Clone(cNewWorker)

End Function

Private Sub StepTextOrFileEntered()
   ' Checks if either the step text or the name of the
file containing
   ' the text has been entered
   ' If both of them are null or both of them are not
null,
   ' the worker step is invalid and an error is raised
   If StringEmpty(mcStep.StepText) And
StringEmpty(mcStep.StepTextFile) Then
      ShowError errStepTextAndFileNull
      On Error GoTo 0
      Err.Raise vbObjectError +
errStepTextAndFileNull, _
         mstrSource,
LoadResString(errStepTextAndFileNull)
   End If
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
End Sub

Private Property Get cStep_IndOperation() As Operation

   cStep_IndOperation = mcStep.IndOperation

End Property

Private Property Let cStep_IndOperation(ByVal RHS As Operation)

   mcStep.IndOperation = RHS

End Property

Private Property Get cStep_NextStepId() As Long

   cStep_NextStepId = mcStep.NextStepId

End Property

Private Property Let cStep_OutputFile(ByVal RHS As String)

   mcStep.OutputFile = RHS

End Property

Private Property Get cStep_OutputFile() As String

   cStep_OutputFile = mcStep.OutputFile

End Property

Private Property Let cStep_ErrorFile(ByVal RHS As String)

   mcStep.ErrorFile = RHS

End Property

Private Property Get cStep_ErrorFile() As String

   cStep_ErrorFile = mcStep.ErrorFile

End Property
'Private Property Let cStep_LogFile(ByVal RHS As String)
'
'   mcStep.LogFile = RHS
'
'End Property
'
'Private Property Get cStep_LogFile() As String
'
'   cStep_LogFile = mcStep.LogFile
'
'End Property

Private Property Let cStep_ArchivedFlag(ByVal RHS As Boolean)

   mcStep.ArchivedFlag = RHS

End Property

Private Property Get cStep_ArchivedFlag() As Boolean

   cStep_ArchivedFlag = mcStep.ArchivedFlag

End Property

Private Sub Class_Initialize()

   ' Create the object
   Set mcStep = New cStep

   ' Initialize the object with valid values for a Worker step
   ' The global flag should be the first field to be initialized
   ' since subsequent validations might try to check if the
   ' step being created is global
   mcStep.GlobalFlag = False
'   mcStep.GlobalRunMethod = gintNoOption
   mcStep.StepType = gintWorkerStep

End Sub
Private Sub Class_Terminate()

   ' Remove the step object
   Set mcStep = Nothing

End Sub
Private Sub cStep_Add()

   ' Call a private procedure to see if the step text has been
   ' entered - since a worker step actually executes a step, entry
   ' of the text is mandatory
   Call StepTextOrFileEntered

   ' Call the Add method of the step class to carry out the insert
   mcStep.Add

End Sub

Private Property Get cStep_ContinuationCriteria() As ContinuationCriteria

   cStep_ContinuationCriteria = mcStep.ContinuationCriteria

End Property

Private Property Let cStep_ContinuationCriteria(ByVal RHS As ContinuationCriteria)

   ' The Continuation criteria must be non-null for all worker steps.
   ' Check if the Continuation Criteria is valid
   Select Case RHS
      Case gintOnFailureAbortSiblings, gintOnFailureCompleteSiblings, _
            gintOnFailureSkipSiblings, gintOnFailureAbort, _
            gintOnFailureContinue, gintOnFailureAsk
         mcStep.ContinuationCriteria = RHS

      Case Else
         On Error GoTo 0
         Err.Raise vbObjectError + errContCriteriaInvalid, _
            mstrModuleName, LoadResString(errContCriteriaInvalid)
   End Select

End Property

Private Property Let cStep_DegreeParallelism(ByVal RHS As String)

   mcStep.DegreeParallelism = RHS

End Property

Private Property Get cStep_DegreeParallelism() As String

   cStep_DegreeParallelism = mcStep.DegreeParallelism

End Property

Private Sub cStep_Delete()

   mcStep.Delete

End Sub

Private Property Get cStep_EnabledFlag() As Boolean

   cStep_EnabledFlag = mcStep.EnabledFlag

End Property

Private Property Let cStep_EnabledFlag(ByVal RHS As Boolean)

   mcStep.EnabledFlag = RHS

End Property

Private Property Let cStep_ExecutionMechanism(ByVal RHS As ExecutionMethod)

   On Error GoTo ExecutionMechanismErr
   mstrSource = mstrModuleName & "cStep_ExecutionMechanism"

   Select Case RHS
      Case gintExecuteShell, gintExecuteODBC
         mcStep.ExecutionMechanism = RHS

      Case Else
         On Error GoTo 0
         Err.Raise vbObjectError + errExecutionMechanismInvalid, _
            mstrSource, LoadResString(errExecutionMechanismInvalid)
   End Select

   Exit Property

ExecutionMechanismErr:
   LogErrors Errors
   mstrSource = mstrModuleName & "cStep_ExecutionMechanism"
   On Error GoTo 0
   Err.Raise vbObjectError + errExecutionMechanismLetFailed, _
      mstrSource, LoadResString(errExecutionMechanismLetFailed)

End Property
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
Private Property Get
cStep_ExecutionMechanism() As
ExecutionMethod

   cStep_ExecutionMechanism =
mcStep.ExecutionMechanism

End Property

Private Property Let
cStep_FailureDetails(ByVal RHS As String)

   mcStep.FailureDetails = RHS

End Property

Private Property Get cStep_FailureDetails()
As String

   cStep_FailureDetails =
mcStep.FailureDetails

End Property

Private Property Get cStep_GlobalFlag() As
Boolean

   cStep_GlobalFlag = mcStep.GlobalFlag

End Property

Private Property Let
cStep_GlobalFlag(ByVal RHS As Boolean)

   ' Set the global flag to false - this flag is
initialized when
   ' an instance of the class is created. Just
making sure that
   ' nobody changes the value inadvertently
   mcStep.GlobalFlag = False

End Property
Private Sub cStep_Modify()

   ' Call a private procedure to see if the step
text has been
   ' entered - since a worker step actually
executes a step, entry
   ' of the text is mandatory
   Call StepTextOrFileEntered

   ' Call the Modify method of the step class
to carry out the update
   mcStep.Modify

End Sub

Private Property Let
cStep_ParentStepId(ByVal RHS As Long)

   mcStep.ParentStepId = RHS

End Property

Private Property Get cStep_ParentStepId()
As Long

   cStep_ParentStepId =
mcStep.ParentStepId

End Property

Private Property Let
cStep_ParentVersionNo(ByVal RHS As
String)

   mcStep.ParentVersionNo = RHS

End Property

Private Property Get cStep_ParentVersionNo()
As String

   cStep_ParentVersionNo =
mcStep.ParentVersionNo

End Property

Private Property Let
cStep_SequenceNo(ByVal RHS As Integer)

   mcStep.SequenceNo = RHS

End Property

Private Property Get cStep_SequenceNo() As
Integer

   cStep_SequenceNo = mcStep.SequenceNo

End Property

Private Property Let cStep_StepId(ByVal RHS
As Long)

   mcStep.StepId = RHS

End Property

Private Property Get cStep_StepId() As Long

   cStep_StepId = mcStep.StepId

End Property


Private Property Let cStep_StepLabel(ByVal
RHS As String)

   mcStep.StepLabel = RHS

End Property

Private Property Get cStep_StepLabel() As
String

   cStep_StepLabel = mcStep.StepLabel

End Property


Private Property Let cStep_StepLevel(ByVal
RHS As Integer)

   mcStep.StepLevel = RHS

End Property

Private Property Get cStep_StepLevel() As
Integer

   cStep_StepLevel = mcStep.StepLevel

End Property

Private Property Let cStep_StepText(ByVal
RHS As String)

   mcStep.StepText = RHS

End Property

Private Property Get cStep_StepText() As String

   cStep_StepText = mcStep.StepText

End Property

Private Property Let cStep_StepTextFile(ByVal
RHS As String)

   mcStep.StepTextFile = RHS

End Property

Private Property Get cStep_StepTextFile() As
String

   cStep_StepTextFile = mcStep.StepTextFile

End Property

Private Property Let cStep_StepType(RHS As
gintStepType)

   mcStep.StepType = gintWorkerStep

End Property

Private Property Get cStep_StepType() As
gintStepType

   cStep_StepType = mcStep.StepType

End Property

Private Sub cStep_Validate()
   ' The validate routines for each of the steps will
   ' carry out the specific validations for the type
and
   ' call the generic validation routine

   On Error GoTo cStep_ValidateErr

   ' Validations specific to worker steps

   ' Check if the step text or a file name has been
   ' specified
   Call StepTextOrFileEntered

   mcStep.Validate

   Exit Sub

cStep_ValidateErr:
   LogErrors Errors
   mstrSource = mstrModuleName &
"cStep_Validate"
   On Error GoTo 0
   Err.Raise vbObjectError + errValidateFailed, _
         mstrSource, _
         LoadResString(errValidateFailed)
End Sub

Private Property Let cStep_VersionNo(ByVal RHS
As String)

   mcStep.VersionNo = RHS

End Property

Private Property Get cStep_VersionNo() As String

   cStep_VersionNo = mcStep.VersionNo
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
End Property

Private Property Let
cStep_WorkspaceId(ByVal RHS As Long)

   mcStep.WorkspaceId = RHS

End Property

Private Property Get cStep_WorkspaceId()
As Long

   cStep_WorkspaceId =
mcStep.WorkspaceId

End Property
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cWorkspace"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:       cWorkspace.cls
'              Microsoft TPC-H Kit Ver. 1.00
'              Copyright Microsoft, 1999
'              All Rights Reserved
'
'
'  PURPOSE:    Encapsulates the properties
and methods of a workspace.
'              Contains functions to insert,
update and delete
'              att_workspaces records from the
database.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

' Local variable(s) to hold property value(s)
Private mlngWorkspaceId As Long
Private mstrWorkspaceName As String
Private mblnArchivedFlag As Boolean
Private mdbsStepMaster As Database

' Used to indicate the source module name
when errors
' are raised by this class
Private mstrSource As String
Private Const mstrModuleName As String =
"cWorkspace."

' The cSequence class is used to generate
unique workspace identifiers
Private mWorkspaceSeq As cSequence

' The StringSM class is used to carry out
string operations
Private mFieldValue As cStringSM

Public Function Clone() As cWorkspace

   ' Creates a copy of a given workspace

   Dim cCloneWsp As cWorkspace

   On Error GoTo CloneErr

   Set cCloneWsp = New cWorkspace

   ' Copy all the workspace properties to the
newly
   ' created workspace
   cCloneWsp.WorkspaceId =
mlngWorkspaceId
   cCloneWsp.WorkspaceName =
mstrWorkspaceName
   cCloneWsp.ArchivedFlag =
mblnArchivedFlag

   ' And set the return value to the newly
created workspace
   Set Clone = cCloneWsp

   Exit Function

CloneErr:
   LogErrors Errors
   mstrSource = mstrModuleName & "Clone"
   On Error GoTo 0
   Err.Raise vbObjectError + errCloneFailed, _
         mstrSource,
LoadResString(errCloneFailed)

End Function

Public Property Let ArchivedFlag(ByVal vdata
As Boolean)

   mblnArchivedFlag = vdata

End Property

Public Property Get ArchivedFlag() As
Boolean

   ArchivedFlag = mblnArchivedFlag

End Property

Public Property Set WorkDatabase(vdata As
Database)

   Set mdbsStepMaster = vdata

End Property

Private Sub WorkspaceNameDuplicate()
   ' Check if the workspace name already exists
in the workspace

   Dim rstWorkspace As Recordset
   Dim strSql As String
   Dim qy As DAO.QueryDef

   On Error GoTo
WorkspaceNameDuplicateErr
   mstrSource = mstrModuleName &
"WorkspaceNameDuplicate"

   ' Create a recordset to retrieve the count of
records
   ' having the same workspace name
   strSql = " Select count(*) as
workspace_count " & _
       " from att_workspaces " & _
       " where workspace_name = [w_name] "
& _
       " and workspace_id <> [w_id] "
   Set qy =
mdbsStepMaster.CreateQueryDef(gstrEmptySt
ring, strSql)

   ' Call a procedure to assign the parameter
values
   Call AssignParameters(qy)
   Set rstWorkspace =
qy.OpenRecordset(dbOpenForwardOnly)

'       mFieldValue.MakeStringFieldValid
(mstrWorkspaceName) & _
'       " and workspace_id <> " & _
'       Str(mlngWorkspaceId)
'
'   Set rstWorkspace =
mdbsStepMaster.OpenRecordset( _
'       strSQL, dbOpenForwardOnly)

   If rstWorkspace![workspace_count] > 0 Then
      rstWorkspace.Close
      qy.Close
      ShowError errDuplicateWorkspaceName
      On Error GoTo 0
      Err.Raise vbObjectError +
errDuplicateWorkspaceName, _
         mstrSource,
LoadResString(errDuplicateWorkspaceName)
   End If
   rstWorkspace.Close
   qy.Close

   Exit Sub

WorkspaceNameDuplicateErr:
   Call LogErrors(Errors)
   mstrSource = mstrModuleName &
"WorkspaceNameDuplicate"
   On Error GoTo 0
   Err.Raise vbObjectError +
errWorkspaceNameDuplicateFailed, _
      mstrSource,
LoadResString(errWorkspaceNameDuplicateFailed
)

End Sub
Public Property Let WorkspaceName(vdata As
String)

   On Error GoTo WorkspaceNameErr
   mstrSource = mstrModuleName &
"WorkspaceName"

   If vdata = gstrEmptyString Then

      On Error GoTo 0
      ' Propogate this error back to the caller
      Err.Raise vbObjectError +
errWorkspaceNameMandatory, _
         mstrSource,
LoadResString(errWorkspaceNameMandatory)
   Else
      mstrWorkspaceName = vdata
   End If
   Exit Property

WorkspaceNameErr:
   LogErrors Errors
   mstrSource = mstrModuleName &
"WorkspaceName"
   On Error GoTo 0
   Err.Raise vbObjectError +
errWorkspaceNameSetFailed, _
      mstrSource,
LoadResString(errWorkspaceNameSetFailed)

End Property

Public Property Let WorkspaceId(vdata As Long)

   On Error GoTo WorkspaceIdErr
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   mstrSource = mstrModuleName &
"WorkspaceId"

   If (vdata > 0) Then
      mlngWorkspaceId = vdata
   Else
      ' Propogate this error back to the caller
      On Error GoTo 0
      Err.Raise vbObjectError +
errWorkspaceIdInvalid, _
         mstrSource,
LoadResString(errWorkspaceIdInvalid)
   End If

   Exit Property

WorkspaceIdErr:
   LogErrors Errors
   mstrSource = mstrModuleName &
"WorkspaceId"
   On Error GoTo 0
   Err.Raise vbObjectError +
errWorkspaceIdSetFailed, _
      mstrSource,
LoadResString(errWorkspaceIdSetFailed)

End Property

Public Sub AddWorkspace()

   Dim strInsert As String
   Dim qy As DAO.QueryDef

   On Error GoTo AddWorkspaceErr

   ' Retrieve the next identifier using the
sequence class
   Set mWorkspaceSeq = New cSequence
   Set mWorkspaceSeq.IdDatabase =
mdbsStepMaster
   mWorkspaceSeq.IdentifierColumn =
FLD_ID_WORKSPACE
   mlngWorkspaceId =
mWorkspaceSeq.Identifier
   Set mWorkspaceSeq = Nothing

   ' Call procedure to raise an error if the
Workspace name
   ' already exists in the db
   Call WorkspaceNameDuplicate

   ' A new record will have the
archived_flag turned off
   mblnArchivedFlag = False

   ' Create a temporary querydef object
   strInsert = "insert into att_workspaces " &
_
         "( workspace_id, workspace_name, "
& _
         " archived_flag ) " & _
         " values ( [w_id], [w_name],
[archived] ) "
   Set qy =
mdbsStepMaster.CreateQueryDef(gstrEmpt
yString, strInsert)

   ' Call a procedure to assign the parameter
values
   Call AssignParameters(qy)

   qy.Execute dbFailOnError
   qy.Close
'   strInsert = "insert into att_workspaces " & _
'         "( workspace_id, workspace_name, "
& _
'         " archived_flag ) " & _
'         " values ( " & _
'         Str(mlngWorkspaceId) & _
'         ", " &
mFieldValue.MakeStringFieldValid(mstrWork
spaceName) & _
'         ", " & Str(mblnArchivedFlag) & _
'         " ) "
'   mdbsStepMaster.Execute strInsert,
dbFailOnError
'
   Exit Sub

AddWorkspaceErr:

   Call LogErrors(Errors)
   mstrSource = mstrModuleName &
"AddWorkspace"
   On Error GoTo 0
   Err.Raise vbObjectError +
errWorkspaceInsertFailed, _
      mstrSource,
LoadResString(errWorkspaceInsertFailed)

End Sub
Private Sub AssignParameters(qyExec As
DAO.QueryDef)
   ' Assigns values to the parameters in the
querydef object
   ' The parameter names are cryptic to make
them different
   ' from the field names. When the parameter
names are
   ' the same as the field names, parameters in
the where
   ' clause do not get created.

   Dim prmParam As DAO.Parameter

   On Error GoTo AssignParametersErr
   mstrSource = mstrModuleName &
"AssignParameters"

   For Each prmParam In qyExec.Parameters
      Select Case prmParam.Name
         Case "[w_id]"
            prmParam.Value =
mlngWorkspaceId

         Case "[w_name]"
            prmParam.Value =
mstrWorkspaceName

         Case "[archived]"
            prmParam.Value =
mblnArchivedFlag

         Case Else
            ' Write the parameter name that is
faulty
            WriteError errInvalidParameter,
mstrSource, _
               prmParam.Name
            On Error GoTo 0
            Err.Raise errInvalidParameter,
mstrSource, _

LoadResString(errInvalidParameter)
      End Select
   Next prmParam

   Exit Sub

AssignParametersErr:

   mstrSource = mstrModuleName &
"AssignParameters"
   Call LogErrors(Errors)
   On Error GoTo 0
   Err.Raise vbObjectError +
errAssignParametersFailed, _
      mstrSource,
LoadResString(errAssignParametersFailed)

End Sub
Public Sub DeleteWorkspace()

   Dim strDelete As String
   Dim qy As DAO.QueryDef

   On Error GoTo DeleteWorkspaceErr

   strDelete = "delete from att_workspaces " & _
      " where workspace_id = [w_id]"
   Set qy =
mdbsStepMaster.CreateQueryDef(gstrEmptyString,
strDelete)

   ' Call a procedure to assign the parameter values
   Call AssignParameters(qy)

   qy.Execute dbFailOnError
   qy.Close

'   mdbsStepMaster.Execute strDelete,
dbFailOnError
'      " where workspace_id = " & _
'      Str(mlngWorkspaceId)

   Exit Sub

DeleteWorkspaceErr:
   Call LogErrors(Errors)
   mstrSource = mstrModuleName &
"DeleteWorkspace"
   On Error GoTo 0
   Err.Raise vbObjectError +
errWorkspaceDeleteFailed, _
      mstrSource,
LoadResString(errWorkspaceDeleteFailed)
End Sub

Public Sub ModifyWorkspace()

   Dim strUpdate As String
   Dim qy As DAO.QueryDef

   On Error GoTo ModifyWorkspaceErr

   ' Call procedure to raise an error if the Workspace
name
   ' already exists in the db
   Call WorkspaceNameDuplicate

   strUpdate = "update att_workspaces " & _
      " set workspace_name = [w_name] " & _
      ", archived_flag = [archived] " & _
      " where workspace_id = [w_id] "
   Set qy =
mdbsStepMaster.CreateQueryDef(gstrEmptyString,
strUpdate)

   ' Call a procedure to assign the parameter values
   Call AssignParameters(qy)

   qy.Execute dbFailOnError
   qy.Close
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
'    strUpdate = "update att_workspaces " &
_
'        " set workspace_name = " & _
'
mFieldValue.MakeStringFieldValid(mstrW
orkspaceName) & _
'        ", archived_flag = " & _
'        Str(mblnArchivedFlag) & _
'        " where workspace_id = " & _
'        Str(mlngWorkspaceId)
'
'    mdbsStepMaster.Execute strUpdate,
dbFailOnError
'
    Exit Sub

ModifyWorkspaceErr:

    Call LogErrors(Errors)
    mstrSource = mstrModuleName &
"ModifyWorkspace"
    On Error GoTo 0
    Err.Raise vbObjectError +
errWorkspaceUpdateFailed, _
        mstrSource,
LoadResString(errWorkspaceUpdateFailed)

End Sub
Public Property Get WorkspaceName() As
String

    WorkspaceName = mstrWorkspaceName

End Property

Public Property Get WorkspaceId() As Long

    WorkspaceId = mlngWorkspaceId

End Property

Private Sub Class_Initialize()

    ' Each function will append it's own name
to this
    ' variable
    mstrSource = "cWorkspace."

    Set mFieldValue = New cStringSM

End Sub

Private Sub Class_Terminate()

    Set mdbsStepMaster = Nothing
    Set mFieldValue = Nothing

End Sub
Attribute VB_Name = "DatabaseSM"
'  FILE:      DatabaseSM.bas
'             Microsoft TPC-H Kit Ver. 1.00
'             Copyright Microsoft, 1999
'             All Rights Reserved
'
'
'  PURPOSE:    Contains all the database
initialization/cleanup
'             procedures for the project. Also
contains upgrade
'             database upgrade functions.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
' This module is called DatabaseSM, since
Database is a standard
' Visual Basic object and we want to avoid any
confusion with it.

Option Explicit

Public wrkJet As Workspace
Public dbsAttTool As Database
Public gblnDbOpen As Boolean
Public gRunEngine As rdoEngine

' Used to indicate the source module name
when errors
' are raised by this module
Private Const mstrModuleName As String =
"DatabaseSM."
Public Const gsDefDBFileExt As String =
".stp"
Private Const msDefDBFile As String =
"\SMData" & gsDefDBFileExt

Private Const merrFileNotFound As Integer =
3024
Private Const merrDaoTableMissing As
Integer = 3078

Private Const
STEPMASTER_SETTINGS_VAL_NAME_D
BFILE As String = "WorkspaceFile"

Public Const DEF_NO_COUNT_DISPLAY
As Boolean = False
Public Const DEF_NO_EXECUTE
As Boolean = False
Public Const DEF_PARSE_QUERY_ONLY
As Boolean = False
Public Const
DEF_ANSI_QUOTED_IDENTIFIERS    As
Boolean = False
Public Const DEF_ANSI_NULLS              As
Boolean = True
Public Const DEF_SHOW_QUERY_PLAN
As Boolean = False
Public Const DEF_SHOW_STATS_TIME
As Boolean = False
Public Const DEF_SHOW_STATS_IO
As Boolean = False
Public Const
DEF_PARSE_ODBC_MSG_PREFIXES    As
Boolean = True
Public Const DEF_ROW_COUNT
As Long = 0
Public Const
DEF_TSQL_BATCH_SEPARATOR       As
String = "GO"
Public Const DEF_QUERY_TIME_OUT
As Long = 0
Public Const DEF_SERVER_LANGUAGE
As String = "(Default)"
Public Const
DEF_CHARACTER_TRANSLATION      As
Boolean = True
Public Const DEF_REGIONAL_SETTINGS
As Boolean = False

Public Const PARAM_DEFAULT_DIR
As String = "DEFAULT_DIR"
Public Const
PARAM_DEFAULT_DIR_DESC         As
String = "Default destination directory " & _
            "for all output and error files. If it is
blank, the StepMaster installation directory
will be used."
Public Const PARAM_RUN_ID                As
String = "RUN_ID"
Public Const PARAM_RUN_ID_DESC           As
String = "The run identifier for a run. " & _
            "Any modifications will be overwritten
before each run."

Public Const PARAM_OUTPUT_DIR            As
String = "OUTPUT_DIR"
Public Const PARAM_OUTPUT_DIR_DESC
As String = "The output directory for a run. " & _
            "Any modifications will be overwritten
before each run."

Public Const
CONNECTION_STRINGS_TO_NAME_SUFFIX
As String = "_NAME"

Private Const TBL_RUN_STEP_HDR As String =
"run_header"
Private Const TBL_RUN_STEP_DTLS As String =
"run_step_details"
Public Const TBL_CONNECTION_DTLS As
String = "connection_dtls"
Public Const TBL_CONNECTION_STRINGS As
String = "workspace_connections"
Public Const TBL_STEPS As String = "att_steps"

Public Const FLD_ID_CONN_NAME As String =
"connection_name_id"
Public Const FLD_ID_WORKSPACE As String =
"workspace_id"
Public Const FLD_ID_STEP As String = "step_id"
Public Const FLD_ID_PARAMETER As String =
"parameter_id"

Public Const
FLD_CONN_DTL_CONNECTION_NAME As
String = "connection_name"
Public Const
FLD_CONN_DTL_CONNECTION_STRING As
String = "connection_string_name"
Public Const
FLD_CONN_DTL_CONNECTION_TYPE As
String = "connection_type"

Public Const
FLD_CONN_STR_CONNECTION_NAME As
String = "connection_name"

Public Const FLD_STEPS_EXEC_MECHANISM
As String = "execution_mechanism"
Public Const FLD_STEPS_EXEC_DTL As String
= "start_directory"
Public Const FLD_STEPS_VERSION_NO As
String = "version_no"

Public Const DATA_TYPE_CURRENCY As
String = "CURRENCY"
Public Const DATA_TYPE_LONG As String =
"Long"
Public Const DATA_TYPE_INTEGER As String =
"INTEGER"
Public Const DATA_TYPE_TEXT255 As String =
"Text(255)"

Private Sub InsertBuiltInParameter(dbFile As
Database, sParamName As String, _
        sParamValue As String, sParamDesc As
String)

    Dim sBuf As String
    Dim cTempStr As New cStringSM
    Dim lId As Long
    Dim rTemp As DAO.Recordset
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   Dim rParam As DAO.Recordset
   Dim cTempSeq As cSequence

   ' Create the passed in built-in parameter,
for each workspace in the db
   Set cTempSeq = New cSequence
   Set cTempSeq.IdDatabase = dbFile
   cTempSeq.IdentifierColumn =
FLD_ID_PARAMETER

   sBuf = "select * from att_workspaces "
   Set rTemp = dbFile.OpenRecordset(sBuf,
dbOpenSnapshot)
   If rTemp.RecordCount <> 0 Then
      rTemp.MoveFirst

      While Not rTemp.EOF
         sBuf = "select * from
workspace_parameters " & _
                 " where workspace_id = " &
Str(rTemp!workspace_id) & _
                 " and parameter_name = " &
cTempStr.MakeStringFieldValid(sParamNa
me)
         Set rParam =
dbFile.OpenRecordset(sBuf,
dbOpenSnapshot)
         If rParam.RecordCount <> 0 Then
            rParam.MoveFirst
            ' Since the parameter already
exists, change it to a built-in type
            sBuf = "update
workspace_parameters " & _
                   " set parameter_type = " &
CStr(gintParameterBuiltIn) & _
                   ", description = " &
cTempStr.MakeStringFieldValid(sParamDe
sc) & _
                   " where workspace_id = " &
Str(rTemp!workspace_id) & _
                   " and parameter_id = " &
Str(rParam!parameter_id)
         Else
            ' Else, insert a parameter record
            lId = cTempSeq.Identifier
            sBuf = "insert into
workspace_parameters " & _
                   "( workspace_id,
parameter_id, " & _
                   " parameter_name,
parameter_value, " & _
                   " description, parameter_type
) " & _
                   " values ( " & _
                   Str(rTemp!workspace_id) &
", " & Str(lId) & ", " & _

cTempStr.MakeStringFieldValid(sParamNa
me) & ", " & _

cTempStr.MakeStringFieldValid(sParamVal
ue) & ", " & _

cTempStr.MakeStringFieldValid(sParamDe
sc) & ", " & _
                   CStr(gintParameterBuiltIn) &
_
                   " ) "
         End If
         dbFile.Execute sBuf, dbFailOnError
         rParam.Close

         rTemp.MoveNext
      Wend
   End If
   rTemp.Close

End Sub
Public Sub InitRunEngine()

   Set gRunEngine = New rdoEngine
   gRunEngine.rdoDefaultCursorDriver =
rdUseServer

End Sub

Public Function DefaultDBFile() As String
   DefaultDBFile = GetSetting(App.Title,
"Settings",
STEPMASTER_SETTINGS_VAL_NAME_D
BFILE, App.Path & msDefDBFile)
End Function

Public Sub CloseDatabase()

   Dim dbsInstance As Database
   Dim recInstance As Recordset

   On Error GoTo CloseDatabaseErr

   ' Close all open recordsets and databases in
the workspace
   For Each dbsInstance In wrkJet.Databases

      For Each recInstance In
dbsAttTool.Recordsets
         recInstance.Close
      Next recInstance
      dbsInstance.Close

   Next dbsInstance

   Set dbsAttTool = Nothing

   gblnDbOpen = False
   wrkJet.Close

   Exit Sub

CloseDatabaseErr:

   Call LogErrors(Errors)
   Resume Next

End Sub

Private Function NoDbChanges(sVerTo As
String, sVerFrom As String) As Boolean

   If sVerTo = gsVersion242 And sVerFrom =
gsVersion241 Then
      NoDbChanges = True
   ElseIf sVerTo = gsVersion242 And
sVerFrom = gsVersion24 Then
      NoDbChanges = True
   ElseIf sVerTo = gsVersion253 And
sVerFrom = gsVersion251 Then
      NoDbChanges = True
   ElseIf sVerTo = gsVersion255 And
sVerFrom = gsVersion251 Then
      NoDbChanges = True
   Else
      NoDbChanges = False
   End If

End Function

Public Function SMOpenDatabase(Optional
strDbName As String = gstrEmptyString) As
Boolean
   Dim sVersion As String
   Dim bOpeningDb As Boolean ' This flag is used
to check if OpenDatabase failed

   On Error GoTo OpenDatabaseErr

   bOpeningDb = False
   SMOpenDatabase = False

   ' Create Microsoft Jet Workspace object.
   If Not gblnDbOpen Then
      Set wrkJet =
CreateWorkspace("att_tool_workspace_setup",
"admin", gstrEmptyString, dbUseJet)
   End If

   ' Prompt the user for the database file if it is not
passed in
   If StringEmpty(strDbName) Then
      strDbName = BrowseDBFile
      If StringEmpty(strDbName) Then
         Exit Function
      End If
   End If

   Do
      If gblnDbOpen Then
#If Not RUN_ONLY Then
         CloseOpenWorkspaces
#End If
         Set wrkJet =
CreateWorkspace("att_tool_workspace_setup",
"admin", gstrEmptyString, dbUseJet)
      End If

      ' Toggle the bOpeningDb flag around the
OpenDatabase method - the value
      ' of this flag will be checked by the error
handler to determine if it is
      ' the OpenDatabase that failed.
      BugMessage "DB File: " & strDbName

      bOpeningDb = True
      ' Open the database for exclusive use
      Set dbsAttTool =
wrkJet.OpenDatabase(strDbName, Options:=True)
      bOpeningDb = False

      If dbsAttTool Is Nothing Then
         ' If the file is not present in the directory,
display
         ' an error and ask the user to enter a new
path
         Call ShowError(errOpenDbFailed,
OptArgs:=strDbName)

         strDbName = BrowseDBFile
      Else
         sVersion = DBVersion(dbsAttTool)

         ' Make sure the application and db version
numbers match
         If sVersion = gsVersion Then
            Call InitializeData(strDbName)
            gblnDbOpen = True
            SMOpenDatabase = True
         Else
            If UpgradeDb(wrkJet, dbsAttTool,
gsVersion, sVersion) Then
               Call InitializeData(strDbName)
               gblnDbOpen = True
               SMOpenDatabase = True
            Else
               dbsAttTool.Close
               Set dbsAttTool = Nothing
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
                ShowError
errVersionMismatch, _
                    OptArgs:=" Please install
Version "' & gsVersion & "' of the
workspace definition file."
                strDbName = BrowseDBFile
            End If
        End If
    End If
  Loop While gblnDbOpen = False And
Not StringEmpty(strDbName)

  Exit Function

OpenDatabaseErr:
  Call DisplayErrors(Errors)

  ' If the OpenDatabase failed, continue
  If bOpeningDb Then
    Resume Next
  End If

  Call ShowError(errOpenDbFailed,
OptArgs:=strDbName)

End Function
Private Sub InitializeData(sDb As String)

  Set gcParameters = New cArrParameters
  Set gcParameters.ParamDatabase =
dbsAttTool

  Set gcSteps = New cArrSteps
  Set gcSteps.StepDB = dbsAttTool

  Set gcConstraints = New cArrConstraints
  Set gcConstraints.ConstraintDB =
dbsAttTool

  Set gcConnections = New cConnections
  Set gcConnections.ConnDb = dbsAttTool

  Set gcConnDtls = New cConnDtls
  Set gcConnDtls.ConnDb = dbsAttTool

  ' Disable the error handler since this is not
a critical step
  On Error GoTo 0
  SaveSetting App.Title, "Settings",
STEPMASTER_SETTINGS_VAL_NAME
_DBFILE, sDb
End Sub
Private Sub
UpdateContinuationCriteria(dbFile As
DAO.Database)

  Dim qyTemp As DAO.QueryDef
  Dim sBuf As String

  On Error GoTo
UpdateContinuationCriteriaErr

  sBuf = "Since this version of the
executable incorporates failure processing, "
& _
        "the upgrade will update the On
Failure field for each of the steps " & _
        "to 'Continue' to be compatible with
the existing behaviour. " & _
        "Proceed?"
  If Not Confirm(Buttons:=vbYesNo,
strMessage:=sBuf, strTitle:="Upgrade
database") Then
    Exit Sub
  End If

  ' Create a recordset object to retrieve all
steps for
  ' the given workspace
  sBuf = " update att_steps a " & _
    " set continuation_criteria = " &
CStr(gintOnFailureContinue) & _
    " where archived_flag = [archived] "

  ' Find the highest X-component of the
version number
  sBuf = sBuf & " AND cint( mid(
version_no, 1, instr( version_no, " & gstrDQ &
gstrVerSeparator & gstrDQ & " ) - 1 ) ) = " &
_
    " ( select max( cint( mid( version_no, 1,
instr( version_no, " & gstrDQ &
gstrVerSeparator & gstrDQ & " ) - 1 ) ) ) " & _
    " from att_steps AS d " & _
    " WHERE a.step_id = d.step_id ) "

  ' Find the highest Y-component of the
version number for the highest X-component
  sBuf = sBuf & " AND cint( mid(
version_no, instr( version_no, " & gstrDQ &
gstrVerSeparator & gstrDQ & " ) + 1 ) ) = " &
_
    " ( select max( cint( mid( version_no,
instr( version_no, " & gstrDQ &
gstrVerSeparator & gstrDQ & " ) + 1 ) ) ) " &
_
    " from att_steps AS b " & _
    " Where a.step_id = b.step_id " & _
    " AND cint( mid( version_no, 1, instr(
version_no, " & gstrDQ & gstrVerSeparator &
gstrDQ & " ) - 1 ) ) = " & _
    " ( select max( cint( mid( version_no, 1,
instr( version_no, " & gstrDQ &
gstrVerSeparator & gstrDQ & " ) - 1 ) ) ) " & _
    " from att_steps AS c " & _
    " WHERE a.step_id = c.step_id ) ) "

  ' Create a temporary Querydef object
  Set qyTemp =
dbFile.CreateQueryDef(gstrEmptyString,
sBuf)
  qyTemp.Parameters("archived").Value =
False

  qyTemp.Execute dbFailOnError
  qyTemp.Close

  Exit Sub

UpdateContinuationCriteriaErr:
  Call LogErrors(Errors)
  Err.Raise vbObjectError +
errModifyStepFailed, mstrModuleName, _
      LoadResString(errModifyStepFailed)

End Sub

Private Sub UpdateDbDtls(dbFile As
Database, sNewVersion As String)

  Dim sSql As String
  Dim cTemp As New cStringSM

  On Error GoTo UpdateDbDtlsErr

  sSql = "update db_details " & _
      " set db_version = " &
cTemp.MakeStringFieldValid(sNewVersion)
  dbFile.Execute sSql, dbFailOnError

  Exit Sub

UpdateDbDtlsErr:
  Call LogErrors(Errors)
  Err.Raise vbObjectError + errUpgradeFailed,
mstrModuleName, _
      LoadResString(errUpgradeFailed)

End Sub

Private Sub Upgrade10to21(UpgradeWsp As
DAO.Workspace, dbFile As Database, sVersion As
String)

  Dim sSql As String

  On Error GoTo Upgrade10to21Err

  Call UpdateDbDtls(dbFile, sVersion)

  Call UpdateContinuationCriteria(dbFile)

  Exit Sub

Upgrade10to21Err:
  UpgradeWsp.Rollback
  Call LogErrors(Errors)
  Err.Raise vbObjectError + errUpgradeFailed,
mstrModuleName, _
      LoadResString(errUpgradeFailed)

End Sub
Private Sub Upgrade21to23(UpgradeWsp As
DAO.Workspace, dbFile As Database, sVersion As
String)

  Dim sBuf As String
  Dim cTempStr As New cStringSM

  On Error GoTo Upgrade21to23Err

  ' Add a parameter type field and a description
field to the parameter table
  sBuf = "alter table workspace_parameters " & _
      " add column description TEXT(255) "
  dbFile.Execute sBuf, dbFailOnError

  sBuf = "alter table workspace_parameters " & _
      " add column parameter_type INTEGER "
  dbFile.Execute sBuf, dbFailOnError

  ' Initialize the parameter type on all parameters to
indicate generic parameters
  sBuf = "update workspace_parameters " & _
      " set parameter_type = " &
CStr(gintParameterGeneric)
  dbFile.Execute sBuf, dbFailOnError

  sBuf = "Release 2.3 onwards, connection string
parameters will be " & _
      "displayed in a separate node. After this
upgrade, all connection " & _
      "string parameters will appear under the
Globals/Connection Strings " & _
      "node in the workspace. "
  Call MsgBox(sBuf, vbOKOnly +
vbApplicationModal, "Upgrade database")

  ' Update the parameter type on all parameters that
look like db connection strings
  sBuf = "update workspace_parameters " & _
      " set parameter_type = " &
CStr(gintParameterConnect) & _
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
        " where UCase(parameter_value)
like '*DRIVER*' " & _
        " or UCase(parameter_value) like
'*DSN*'"
   dbFile.Execute sBuf, dbFailOnError

   ' Add an elapsed time field to the
run_step_details table - this field is
   ' needed to store the elapsed time in
milliseconds.
   sBuf = "alter table run_step_details " & _
        " add column elapsed_time LONG "
   dbFile.Execute sBuf, dbFailOnError

   ' The failure_details field has some data
for the case when an ODBC failure
   ' threshold was specified. Since that's no
longer relevant, update the failure_details
   ' field for records with failure_criteria =
gintFailureODBC to empty.
   ' failure_criteria = gintFailureODBC = 1
   sBuf = "update att_steps " & _
        " set failure_details = " &
cTempStr.MakeStringFieldValid(gstrEmpty
String) & _
        " where failure_criteria = '1'"
   dbFile.Execute sBuf, dbFailOnError

   Call UpdateDbDtls(dbFile, sVersion)

   UpgradeWsp.CommitTrans

   On Error GoTo DropColumnErr

   UpgradeWsp.BeginTrans

   ' This ddl cannot be in the same
transaction as the failure_details update
   ' But we can do this in a separate
transaction since we do not expect this
   ' statement to fail - AND, it doesn't matter
if this transaction fails
   ' Drop the failure_criteria column from
the att_steps table
   sBuf = "alter table att_steps " & _
        " drop column failure_criteria "
   dbFile.Execute sBuf, dbFailOnError

   Exit Sub

DropColumnErr:
   Call LogErrors(Errors)
   ShowError errDeleteColumnFailed
   Exit Sub

Upgrade21to23Err:
   UpgradeWsp.Rollback
   Call LogErrors(Errors)
   Err.Raise vbObjectError +
errUpgradeFailed, mstrModuleName, _
        LoadResString(errUpgradeFailed)

End Sub
Private Sub Upgrade23to24(UpgradeWsp
As DAO.Workspace, dbFile As Database,
sVersion As String)

   Dim sBuf As String
   Dim lId As Long
   Dim rTemp As DAO.Recordset
   Dim cTempStr As New cStringSM

   On Error GoTo Upgrade23to24Err

   ' Add a new table for connection properties
   sBuf = CreateConnectionsTableScript()
   ' TODO: Not sure of column sizes for row
count, tsql_batch_separator and
server_language
   dbFile.Execute sBuf, dbFailOnError

   ' Move all connection parameters from the
parameter table to the connections tables
   ' Insert default values for the newly added
connection properties
   sBuf = "select * from workspace_parameters
" & _
        "where parameter_type = " &
CStr(gintParameterConnect)
   Set rTemp = dbFile.OpenRecordset(sBuf,
dbOpenSnapshot)
   lId = 1
   If rTemp.RecordCount <> 0 Then
      rTemp.MoveFirst

      While Not rTemp.EOF
         sBuf = "insert into
workspace_connections " & _
         "( workspace_id, connection_id, " & _
         "connection_name, connection_value,
" & _
         "description, no_count_display, " & _
         "no_execute, parse_query_only, " & _
         "ANSI_quoted_identifiers,
ANSI_nulls, " & _
         "show_query_plan, show_stats_time, "
& _
         "show_stats_io,
parse_odbc_msg_prefixes, " & _
         "row_count, tsql_batch_separator, " &
_
         "query_time_out, server_language, " &
_
         "character_translation,
regional_settings ) " & _
         " values ( " & _
         Str(rTemp!workspace_id) & ", " &
Str(lId) & ", " & _
         cTempStr.MakeStringFieldValid("" &
rTemp!parameter_name) & ", " & _
         cTempStr.MakeStringFieldValid("" &
rTemp!parameter_value) & ", " & _
         cTempStr.MakeStringFieldValid("" &
rTemp!Description) & ", " & _
         Str(DEF_NO_COUNT_DISPLAY) &
", " & _
         Str(DEF_NO_EXECUTE) & ", " &
Str(DEF_PARSE_QUERY_ONLY) & ", " & _

Str(DEF_ANSI_QUOTED_IDENTIFIERS) &
", " & Str(DEF_ANSI_NULLS) & ", " & _
         Str(DEF_SHOW_QUERY_PLAN) &
", " & Str(DEF_SHOW_STATS_TIME) & ", "
& _
         Str(DEF_SHOW_STATS_IO) & ", " &
Str(DEF_PARSE_ODBC_MSG_PREFIXES)
& ", " & _
         Str(DEF_ROW_COUNT) & ", " &
cTempStr.MakeStringFieldValid(DEF_TSQL_
BATCH_SEPARATOR) & ", " & _
         Str(DEF_QUERY_TIME_OUT) & ", "
&
cTempStr.MakeStringFieldValid(DEF_SERV
ER_LANGUAGE) & ", " & _

Str(DEF_CHARACTER_TRANSLATION) &
", " & Str(DEF_REGIONAL_SETTINGS) & _
         " ) "
         dbFile.Execute sBuf, dbFailOnError

         lId = lId + 1
         rTemp.MoveNext
      Wend
   End If
   rTemp.Close

   ' Add an identifier column for the connection_id
field
   sBuf = "alter table att_identifiers " & _
        " add column connection_id long "
   dbFile.Execute sBuf, dbFailOnError

   ' Initialize the value of the connection identifier,
initialized above
   sBuf = "update att_identifiers " & _
        " set connection_id = " & Str(lId)
   dbFile.Execute sBuf, dbFailOnError

   ' Delete all connection strings from the parameter
table
   sBuf = "delete from workspace_parameters " & _
        "where parameter_type = " &
CStr(gintParameterConnect)
   dbFile.Execute sBuf, dbFailOnError

   ' Create the built-in parameter, default directory,
for each workspace in the db
   Call InsertBuiltInParameter(dbFile,
PARAM_DEFAULT_DIR, gstrEmptyString,
PARAM_DEFAULT_DIR_DESC)

   Call UpdateDbDtls(dbFile, sVersion)

   Exit Sub

Upgrade23to24Err:
   UpgradeWsp.Rollback
   Call LogErrors(Errors)
   Err.Raise vbObjectError + errUpgradeFailed,
mstrModuleName, _
        LoadResString(errUpgradeFailed)

End Sub
Private Sub Upgrade243to25(UpgradeWsp As
DAO.Workspace, dbFile As Database, sVersion As
String)

   Dim sBuf As String
   Dim qy As DAO.QueryDef
   Dim rTemp As DAO.Recordset
   Dim lId As Long
   Dim cTempStr As New cStringSM

   On Error GoTo Upgrade243to25Err

   sBuf = "Release " & gsVersion25 & " onwards,
new 'Connections' must be created for all " & _
        "connection strings. " & vbCrLf & vbCrLf
& _
        "Connections will appear under the
Globals/Connections " & _
        "node in the workspace. " & vbCrLf & _
        "A list of all 'Connections' (instead of
'Connection Strings') " & _
        "in the workspace will be displayed in the
'Connections' field for " & _
        "ODBC steps on the Step definition screen.
" & vbCrLf & vbCrLf & _
        "Each Connection can be marked as static or
dynamic. " & vbCrLf & _
        "Dynamic connections will be created when
a step starts execution and " & _
        "closed once the step completes. " & vbCrLf
& _
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
        "Static connections will be kept open
till the run completes." & vbCrLf & vbCrLf
& _
        "Currently dynamic 'Connections'
have been created for all existing
'Connection Strings' " & _
        "with the suffix " &
CONNECTION_STRINGS_TO_NAME_S
UFFIX
   Call MsgBox(sBuf, vbOKOnly +
vbApplicationModal, "Upgrade database")

   ' Add a new table for the connection name
entity
   ' This table has been added in order to
satisfy the TPC-H requirement that
   ' all the queries in a stream need to be
executed on a single connection.
   sBuf =
CreateConnectionDtlsTableScript()
   dbFile.Execute sBuf, dbFailOnError

   ' Add an identifier column for the
connection_name_id field
   sBuf = "alter table att_identifiers " & _
        " add column " &
FLD_ID_CONN_NAME & " long "
   dbFile.Execute sBuf, dbFailOnError

   Call UpdateDbDtls(dbFile, sVersion)

   ' insert connection_dtl records for each of
the connection strings
   sBuf = "select * from " &
TBL_CONNECTION_STRINGS
   Set rTemp = dbFile.OpenRecordset(sBuf,
dbOpenSnapshot)

   sBuf = "insert into " &
TBL_CONNECTION_DTLS & _
         "( " & FLD_ID_WORKSPACE & _
         ", " & FLD_ID_CONN_NAME & _
         ", " &
FLD_CONN_DTL_CONNECTION_NAM
E & _
         ", " &
FLD_CONN_DTL_CONNECTION_STRI
NG & _
         ", " &
FLD_CONN_DTL_CONNECTION_TYPE
& " ) " & _
        " values ( [w_id], [c_id], [c_name],
[c_str], [c_type] ) "
   Set qy = dbFile.CreateQueryDef("", sBuf)

   lId = glMinId
   If rTemp.RecordCount <> 0 Then
      rTemp.MoveFirst

      While Not rTemp.EOF
         qy.Parameters("w_id").Value =
rTemp.Fields(FLD_ID_WORKSPACE)
         qy.Parameters("c_id").Value = lId
         qy.Parameters("c_name").Value =
rTemp.Fields(FLD_CONN_STR_CONNEC
TION_NAME) &
CONNECTION_STRINGS_TO_NAME_S
UFFIX
         qy.Parameters("c_str").Value =
rTemp.Fields(FLD_CONN_STR_CONNEC
TION_NAME)
         qy.Parameters("c_type").Value =
ConnTypeDynamic

         qy.Execute dbFailOnError
         lId = lId + 1
         rTemp.MoveNext
      Wend
   End If
   qy.Close
   rTemp.Close

   ' Initialize the value of the
connection_name_id
   sBuf = "update att_identifiers " & _
         " set " & FLD_ID_CONN_NAME & "
= " & Str(lId)
   dbFile.Execute sBuf, dbFailOnError

   ' Update the start_directory field in att_steps
to point to the newly
   ' created connections
   Call ReadStepsInWorkspace(rTemp, qy,
glInvalidId, dbLoad:=dbFile, _
         bSelectArchivedRecords:=False)

   sBuf = "update " & TBL_STEPS & _
      " set " & FLD_STEPS_EXEC_DTL & " =
[c_name] " & _
      " where " & FLD_ID_STEP & " = [s_id] "
& _
      " and " & FLD_STEPS_VERSION_NO
& " = [ver_no] "
   Set qy = dbFile.CreateQueryDef("", sBuf)

   If rTemp.RecordCount <> 0 Then
      rTemp.MoveFirst

      While Not rTemp.EOF
         If
rTemp.Fields(FLD_STEPS_EXEC_MECHAN
ISM).Value = gintExecuteODBC Then
            If Not (StringEmpty("" &
rTemp.Fields(FLD_STEPS_EXEC_DTL)))
Then
               sBuf =
rTemp.Fields(FLD_STEPS_EXEC_DTL)
               ' Strip the enclosing "%"
characters
               sBuf = Mid(sBuf, 2, Len(sBuf) -
2) &
CONNECTION_STRINGS_TO_NAME_SUF
FIX

               qy.Parameters("c_name").Value =
sBuf
               qy.Parameters("s_id").Value =
rTemp.Fields(FLD_ID_STEP)
               qy.Parameters("ver_no").Value =
rTemp.Fields(FLD_STEPS_VERSION_NO)

               qy.Execute dbFailOnError
            End If
         End If
         rTemp.MoveNext
      Wend
   End If

   qy.Close
   rTemp.Close

   Exit Sub

Upgrade243to25Err:
   UpgradeWsp.Rollback
   Call LogErrors(Errors)
   Err.Raise vbObjectError +
errUpgradeFailed, mstrModuleName, _
         LoadResString(errUpgradeFailed)
End Sub
Private Sub Upgrade25to251(UpgradeWsp As
DAO.Workspace, dbFile As Database, sVersion As
String)

   On Error GoTo Upgrade25to251Err

   ' Create the built-in parameters, run_id and
output_dir, for each workspace in the db
   Call InsertBuiltInParameter(dbFile,
PARAM_RUN_ID, gstrEmptyString,
PARAM_RUN_ID_DESC)
   Call InsertBuiltInParameter(dbFile,
PARAM_OUTPUT_DIR, gstrEmptyString,
PARAM_OUTPUT_DIR_DESC)

   Call UpdateDbDtls(dbFile, sVersion)

   Exit Sub

Upgrade25to251Err:
   UpgradeWsp.Rollback
   Call LogErrors(Errors)
   Err.Raise vbObjectError + errUpgradeFailed,
mstrModuleName, _
         LoadResString(errUpgradeFailed)

End Sub

Private Sub Upgrade242to243(UpgradeWsp As
DAO.Workspace, dbFile As Database, sVersion As
String)

   Dim sBuf As String
   Dim cTempStr As New cStringSM
   Dim iResponse As Integer

   On Error GoTo DeleteHistoryErr

   Call DeleteRunHistory(dbFile)

   On Error GoTo Upgrade242to243Err

   UpgradeWsp.CommitTrans

   UpgradeWsp.BeginTrans

   ' Add a parameter type field and a description
field to the parameter table
   sBuf = "alter table run_step_details " & _
         " add column parent_instance_id LONG "

   dbFile.Execute sBuf, dbFailOnError

   sBuf = "alter table run_step_details " & _
         " add column iterator_value TEXT(255) "

   dbFile.Execute sBuf, dbFailOnError

   Call AlterFieldType(dbFile,
TBL_RUN_STEP_DTLS, "start_time",
DATA_TYPE_CURRENCY)
   Call AlterFieldType(dbFile,
TBL_RUN_STEP_DTLS, "end_time",
DATA_TYPE_CURRENCY)
   Call AlterFieldType(dbFile,
TBL_RUN_STEP_HDR, "start_time",
DATA_TYPE_CURRENCY)
   Call AlterFieldType(dbFile,
TBL_RUN_STEP_HDR, "end_time",
DATA_TYPE_CURRENCY)

   Call UpdateDbDtls(dbFile, sVersion)
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
    Exit Sub

DeleteHistoryErr:
    ' This is not a critical error - continue with
upgrade
    Call LogErrors(Errors)
    Resume Next

Upgrade242to243Err:
    UpgradeWsp.Rollback
    Call LogErrors(Errors)
    Err.Raise vbObjectError +
errUpgradeFailed, mstrModuleName, _
            LoadResString(errUpgradeFailed)

End Sub
'*************************************
******************************
' The AlterFieldType Sub procedure requires
three string
' parameters. The first string specifies the
name of the table
' containing the field to be changed. The
second string specifies
' the name of the field to be changed. The
third string specifies
' the new data type for the field.
'*************************************
******************************

Private Sub AlterFieldType(dbFile As
Database, TblName As String, FieldName
As String, _
                NewDataType As String)
    Dim qdf As DAO.QueryDef
    Dim sSql As String

    ' Add a temporary field to the table.
    sSql = "ALTER TABLE [" & TblName &
_
            "] ADD COLUMN AlterTempField
" & NewDataType
    Set qdf = dbFile.CreateQueryDef("",
sSql)
    qdf.Execute

    ' Copy the data from old field into the new
field.
    qdf.SQL = "UPDATE DISTINCTROW
[" & TblName & "] SET AlterTempField =
[" & FieldName & "]"
    qdf.Execute

    ' Delete the old field.
    qdf.SQL = "ALTER TABLE [" &
TblName & "] DROP COLUMN [" &
FieldName & "]"
    qdf.Execute

    ' Rename the temporary field to the old
field's name.
    dbFile.TableDefs("[" & TblName &
"]").Fields("AlterTempField").Name =
FieldName
    dbFile.TableDefs.Refresh

    ' Clean up.
End Sub
Private Sub Upgrade01to21(UpgradeWsp
As DAO.Workspace, dbFile As
DAO.Database, sVersion As String)
    Dim sSql As String

    On Error GoTo Upgrade01to21Err

    sSql = "Create table db_details (" & _
            "db_version              Text(50) " & _
    ");"

    dbFile.Execute sSql, dbFailOnError

    sSql = "insert into db_details " & _
            "( db_version ) values ( '" & sVersion
& "' ) "

    dbFile.Execute sSql, dbFailOnError

    Call UpdateContinuationCriteria(dbFile)

    Exit Sub

Upgrade01to21Err:
    Call LogErrors(Errors)
    UpgradeWsp.Rollback
    Err.Raise vbObjectError +
errUpgradeFailed, mstrModuleName, _
            LoadResString(errUpgradeFailed)

End Sub
Private Function UpgradeDb(UpgradeWsp As
DAO.Workspace, dbFile As Database, _
        sVerTo As String, sVerFrom As String)
As Boolean

    Dim sMsg As String

    On Error GoTo UpgradeDbErr

    UpgradeDb = False
    If Not ValidUpgrade(sVerTo, sVerFrom)
Then Exit Function

    If NoDbChanges(sVerTo, sVerFrom) Then
        UpgradeDb = True
        Exit Function
    End If

    sMsg = "The database needs to be upgraded
from Version " & sVerFrom & _
            " to Version " & sVerTo & "." &
vbCrLf & _
            "Proceed?"
    If Not Confirm(Buttons:=vbYesNo,
strMessage:=sMsg, strTitle:="Upgrade
database") Then
        Exit Function
    End If

    UpgradeWsp.BeginTrans

    Select Case sVerFrom
        Case gsVersion25
            Call Upgrade25to251(UpgradeWsp,
dbFile, gsVersion251)

        Case gsVersion243
            Call Upgrade243to25(UpgradeWsp,
dbFile, gsVersion25)
            Call Upgrade25to251(UpgradeWsp,
dbFile, gsVersion251)

        Case gsVersion24, gsVersion241,
gsVersion242
            sMsg = "After this upgrade, the run
history for previous runs will no longer be
available. " & _
                "Continue?"
            If Not Confirm(Buttons:=vbYesNo,
strMessage:=sMsg, strTitle:="Upgrade
database") Then
                UpgradeWsp.CommitTrans
                Exit Function
            End If
            Call Upgrade242to243(UpgradeWsp,
dbFile, gsVersion243)
            Call Upgrade243to25(UpgradeWsp, dbFile,
gsVersion25)
            Call Upgrade25to251(UpgradeWsp, dbFile,
gsVersion251)

        Case gsVersion23
            Call Upgrade23to24(UpgradeWsp, dbFile,
gsVersion24)
            Call Upgrade242to243(UpgradeWsp,
dbFile, gsVersion242)
            Call Upgrade243to25(UpgradeWsp, dbFile,
gsVersion25)
            Call Upgrade25to251(UpgradeWsp, dbFile,
gsVersion251)

        Case gsVersion21
            Call Upgrade21to23(UpgradeWsp, dbFile,
gsVersion23)
            Call Upgrade23to24(UpgradeWsp, dbFile,
gsVersion24)
            Call Upgrade242to243(UpgradeWsp,
dbFile, gsVersion242)
            Call Upgrade243to25(UpgradeWsp, dbFile,
gsVersion25)
            Call Upgrade25to251(UpgradeWsp, dbFile,
gsVersion251)

        Case gsVersion10
            Call Upgrade10to21(UpgradeWsp, dbFile,
gsVersion21)
            Call Upgrade21to23(UpgradeWsp, dbFile,
gsVersion23)
            Call Upgrade23to24(UpgradeWsp, dbFile,
gsVersion24)
            Call Upgrade242to243(UpgradeWsp,
dbFile, gsVersion242)
            Call Upgrade243to25(UpgradeWsp, dbFile,
gsVersion25)
            Call Upgrade25to251(UpgradeWsp, dbFile,
gsVersion251)

        Case gsVersion01
            Call Upgrade01to21(UpgradeWsp, dbFile,
gsVersion21)
            Call Upgrade21to23(UpgradeWsp, dbFile,
gsVersion23)
            Call Upgrade23to24(UpgradeWsp, dbFile,
gsVersion24)
            Call Upgrade242to243(UpgradeWsp,
dbFile, gsVersion242)
            Call Upgrade243to25(UpgradeWsp, dbFile,
gsVersion25)
            Call Upgrade25to251(UpgradeWsp, dbFile,
gsVersion251)

    End Select

    UpgradeWsp.CommitTrans

    UpgradeDb = True
    Exit Function

UpgradeDbErr:
    Call LogErrors(Errors)
    ShowError errUpgradeFailed

End Function
Private Function DBVersion(TestDb As Database)
As String
    ' Retrieves the database version
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   Dim rVersion As Recordset

   On Error GoTo DBVersionErr

   Set rVersion =
TestDb.OpenRecordset("Select db_version
from db_details ", _
         dbOpenForwardOnly)

   BugAssert rVersion.RecordCount <> 0
   DBVersion = rVersion!db_version

   rVersion.Close
   Exit Function

DBVersionErr:
   If Err.Number = merrDaoTableMissing
Then
      DBVersion = gsVersion01
   Else
      LogErrors Errors
      Err.Raise vbObjectError +
errUpgradeFailed, mstrModuleName, _
            LoadResString(errUpgradeFailed)
   End If

End Function

Private Function ValidUpgrade(sVerTo As
String, sVerFrom As String) As Boolean

   If sVerTo = gsVersion And sVerFrom =
gsVersion251 Then
      ValidUpgrade = True
   ElseIf sVerTo = gsVersion And
sVerFrom = gsVersion25 Then
      ValidUpgrade = True
   ElseIf sVerTo = gsVersion And
sVerFrom = gsVersion243 Then
      ValidUpgrade = True
   ElseIf sVerTo = gsVersion And
sVerFrom = gsVersion242 Then
      ValidUpgrade = True
   ElseIf sVerTo = gsVersion And
sVerFrom = gsVersion241 Then
      ValidUpgrade = True
   ElseIf sVerTo = gsVersion And
sVerFrom = gsVersion24 Then
      ValidUpgrade = True
   ElseIf sVerTo = gsVersion And
sVerFrom = gsVersion23 Then
      ValidUpgrade = True
   ElseIf sVerTo = gsVersion And
sVerFrom = gsVersion21 Then
      ValidUpgrade = True
   ElseIf sVerTo = gsVersion And
sVerFrom = gsVersion10 Then
      ValidUpgrade = True
   ElseIf sVerTo = gsVersion And
sVerFrom = gsVersion01 Then
      ValidUpgrade = True
   Else
      ValidUpgrade = False
   End If

End Function

Attribute VB_Name = "DebugSM"
'  FILE:      DebugSM.bas
'            Microsoft TPC-H Kit Ver. 1.00
'            Copyright Microsoft, 1999
'            All Rights Reserved
'
'  PURPOSE:    Contains all the functions that
carry out error/debug
'              processing for the project.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
' Most of the functions in this module that
manipulate the
' error object do not have an On Error GoTo
statement - this
' is because it will clear the passed in error
object - let
' the calling functions handle the errors raised
by this
' module, if any
Option Explicit

' Used to indicate the source module name
when errors
' are raised by this module
Private Const mstrModuleName As String =
"DebugSM."

Private mcLogFile As cFileSM
Private mcErrorFile As cFileSM

Private Const
FORMAT_MESSAGE_FROM_SYSTEM =
&H1000
Private Const
FORMAT_MESSAGE_IGNORE_INSERTS =
&H200
Private Const pNull = 0

Declare Function FormatMessage Lib
"kernel32" Alias "FormatMessageA" (ByVal
dwFlags As Long, lpSource As Any, ByVal
dwMessageId As Long, ByVal dwLanguageId
As Long, ByVal lpbuffer As String, ByVal
nSize As Long, Arguments As Long) As Long
Public Function Confirm(Optional
lngMessageCode As conConfirmMsgCodes, _
      Optional lngTitleCode As
conConfirmMsgTitleCodes, _
      Optional TitleParameter As String, _
      Optional ByVal Buttons As Integer = -1, _
      Optional strMessage As String =
gstrEmptyString, _
      Optional strTitle As String =
gstrEmptyString) _
       As Boolean
   ' Displays a confirmation message
corresponding to the
   ' passed in message code. Returns True if the
user says
   ' Ok and False otherwise

   Dim intResponse As Integer
   Dim intButtonStyle As Integer

   On Error GoTo ConfirmErr

   Confirm = False

   ' If the buttons style hasn't been specified, set
the
   ' default style to display OK and Cancel
buttons
   If Buttons = -1 Then
      intButtonStyle = vbOKCancel
   Else
      intButtonStyle = Buttons
   End If

   ' Find the message string for the passed in code
   If StringEmpty(strMessage) Then
      strMessage =
Trim$(LoadResString(lngMessageCode))
   End If

   If StringEmpty(strTitle) Then
      strTitle =
Trim$(LoadResString(lngTitleCode))
   End If

   If Not StringEmpty(TitleParameter) Then
      strTitle = strTitle & Chr$(vbKeySpace) & _
            gstrSQ & TitleParameter & gstrSQ
   End If

   ' Display the confirmation message with the
Cancel button
   ' set to the default - assume that we are
confirming
   ' potentially dangerous operations!
   intResponse = MsgBox(strMessage, _
         intButtonStyle + vbQuestion +
vbApplicationModal, _
         strTitle)

   ' Translate the user response into a True/False
return code
   If intButtonStyle = vbOKCancel Then
      If intResponse = vbOK Then
         Confirm = True
      Else
         Confirm = False
      End If
   Else
      If intResponse = vbYes Then
         Confirm = True
      Else
         Confirm = False
      End If
   End If

   Exit Function

ConfirmErr:
   ' Log the error code raised by Visual Basic
   Call LogErrors(Errors)
   On Error GoTo 0
   gstrSource = mstrModuleName & "Confirm"
   Err.Raise vbObjectError + errConfirmFailed, _
         gstrSource, _
         LoadResString(errConfirmFailed)

End Function
Public Sub LogSystemError()
   Dim eErrCode As Long

   eErrCode = GetLastError()
   If eErrCode <> 0 Then
      WriteToFile "System Error: " & eErrCode &
vbCrLf & ApiError(eErrCode), _
            blnError:=True
   End If

End Sub
Public Function ApiError(ByVal e As Long) As
String

   Dim s As String
   Dim c As Long

   s = String(256, 0)
   c =
FormatMessage(FORMAT_MESSAGE_FROM_S
YSTEM Or _
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
FORMAT_MESSAGE_IGNORE_INSERT
S, _
          pNull, e, 0&, s, Len(s), ByVal pNull)
   If c Then ApiError = e & ": " & Left$(s,
c)

End Function

' Output flags determine output destination
of BugAsserts and messages
#Const afLogfile = 1
#Const afMsgBox = 2
#Const afDebugWin = 4
#Const afAppLog = 8

' Display appropriate error message, and
then stop
' program.  These errors should NOT be
possible in
' shipping product.
Sub BugAssert(ByVal fExpression As
Boolean, _
           Optional sExpression As String)
#If afDebug Then
   If fExpression Then Exit Sub
   BugMessage "BugAssert failed: " &
sExpression
   Stop
#End If
End Sub

Sub BugMessage(sMsg As String)

#If afDebug And afLogfile Then
   ' Since we are writing log messages, the
error flag is turned off
   Call WriteToFile(sMsg, False)
#End If
#If afDebug And afMsgBox Then
   MsgBox sMsg
#End If
#If afDebug And afDebugWin Then
   Debug.Print sMsg
#End If
#If afDebug And afAppLog Then
   App.LogEvent sMsg
#End If

End Sub
Public Function ProjectLogFile() As String

   ProjectLogFile = mcLogFile.FileName

End Function
Public Function ProjectErrorFile() As String

   ProjectErrorFile = mcErrorFile.FileName

End Function

Private Sub WriteToFile(sMsg As String,
Optional ByVal blnError As Boolean)

   ' Calls procedures to write the passed in
message to the log -
   ' The blnError flag is used to indicate that
the message
   ' should be logged to the error file - by
default the log
   ' file is used

   Dim mcFileObj As cFileSM
   Dim strFileName As String
   Dim strFileHdr As String

   On Error GoTo WriteToFileErr

   If blnError Then
      If mcErrorFile Is Nothing Then
         Set mcErrorFile = New cFileSM
      End If
      Set mcFileObj = mcErrorFile
   Else
      If mcLogFile Is Nothing Then
         Set mcLogFile = New cFileSM
      End If
      Set mcFileObj = mcLogFile
   End If

   If StringEmpty(mcFileObj.FileName) Then
      If blnError Then
         strFileName = gstrProjectPath & "\" &
App.EXEName & ".ERR"
         strFileHdr = "Stepmaster Errors"
      Else
         strFileName = gstrProjectPath & "\" &
App.EXEName & ".DBG"
         strFileHdr = "Stepmaster Log"
      End If

      mcFileObj.FileName = strFileName
      mcFileObj.WriteLine strFileHdr
      mcFileObj.WriteLine "Log start time : "
& Now
   End If

   mcFileObj.WriteLine sMsg

   Exit Sub

WriteToFileErr:
   ' Display the error code raised by Visual
Basic
   Call DisplayErrors(Errors)
   ' An error message would've been displayed
by the called
   ' procedures

End Sub
Public Sub WriteMessage(sMsg As String)

   Call WriteToFile(sMsg, True)

End Sub

Sub BugTerm()
#If afDebug And afLogfile Then
   ' Close log file
   mcLogFile.CloseFile
#End If
End Sub

Public Sub ShowError(ByVal ErrorCode As
errErrorConstants, _
      Optional ByVal ErrorSource As String =
gstrEmptyString, _
      Optional ByVal OptArgs As String =
gstrEmptyString, _
      Optional ByVal DoWriteError As
Boolean = True)

   If DoWriteError Then
      ' Call a procedure to write the error to a
log file
      Call WriteError(ErrorCode, ErrorSource,
OptArgs)
   End If

   ' Re-initialize the values of the Error object
before
   ' displaying the error to the user
   Call InitErrObject(ErrorCode, ErrorSource,
OptArgs)

   Call DisplayErrors(Errors)

   Err.Clear

End Sub
Public Sub WriteError(ByVal ErrorCode As
errErrorConstants, _
      Optional ByVal ErrorSource As String =
gstrEmptyString, _
      Optional ByVal OptArgs As String =
gstrEmptyString)

   ' Initialize the values of the Error object before
   ' calling the log function
   Call InitErrObject(ErrorCode, ErrorSource,
OptArgs)

   Call LogErrors(Errors)

   Err.Clear

End Sub
Private Sub InitErrObject(ByVal ErrorCode As
errErrorConstants, _
      Optional ByVal ErrorSource As String =
gstrEmptyString, _
      Optional ByVal OptArgs As String =
gstrEmptyString)

   Dim lngError As Long

   lngError = IIf(ErrorCode > vbObjectError And
ErrorCode < vbObjectError + 65535, _
         ErrorCode - vbObjectError, ErrorCode)
   Err.Number = lngError + vbObjectError
   Err.Description = LoadResString(lngError) &
OptArgs
   Err.Source = App.EXEName & ErrorSource

End Sub
Public Sub ShowMessage(ByVal MessageCode As
errErrorConstants, _
      Optional ByVal OptArgs As String)

   Dim strMessage As String

   On Error GoTo ShowMessageErr

   strMessage = LoadResString(MessageCode) &
OptArgs

   ' Write the error to a log file
   BugMessage strMessage

   MsgBox strMessage, vbOKOnly

   Exit Sub

ShowMessageErr:
   ' Log the error and exit
   Call DisplayErrors(Errors)

End Sub
Public Sub ShowMessageStr(sMessage As String)

   ' Write the error to a log file
   BugMessage sMessage

   MsgBox sMessage, vbOKOnly
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
End Sub

Public Sub DisplayErrors(myErrCollection
As Errors)
   Dim strError As String
   Dim errLoop As Error
   Dim errCode As Long

   ' Enumerate Errors collection and display
properties of
   ' each Error object.
   If Err.Number <> 0 Then
      If Err.Number > vbObjectError And
Err.Number < (vbObjectError + 65536)
Then
         errCode = Err.Number -
vbObjectError
      Else
         errCode = Err.Number
      End If
      strError = "Error # " & Str(errCode) &
" was generated by " _
         & Err.Source & Chr(13) &
Err.Description
      MsgBox strError, , "Error",
Err.HelpFile, Err.HelpContext
   Else
      For Each errLoop In myErrCollection
         With errLoop
            If Err.Number > vbObjectError
And Err.Number < (vbObjectError + 65536)
Then
               errCode = .Number -
vbObjectError
            Else
               errCode = .Number
            End If
            strError = "Error #" & errCode &
vbCrLf
            strError = strError & "    " &
.Description & vbCrLf
            strError = strError & _
               "    (Source: " & .Source & ")"
& vbCrLf
'           strError = strError & _
'              "Press F1 to see topic " &
.HelpContext & vbCrLf
'           strError = strError & _
'              "    in the file " & .HelpFile &
"."
         End With

         MsgBox strError
      Next
   End If

End Sub
Public Sub LogErrors(myErrCollection As
Errors)
   Dim cColErrors As cVectorStr
   Dim strError As String
   Dim errLoop As Error
   Dim errCode As Long
   Dim lngIndex As Long

   Set cColErrors = New cVectorStr

   ' Enumerate Errors collection and display
properties of
   ' each Error object.
   If Err.Number <> 0 Then
      If Err.Number > vbObjectError And
Err.Number < (vbObjectError + 65536)
Then
         errCode = Err.Number - vbObjectError
      Else
         errCode = Err.Number
      End If
      strError = "Error # " & Str(errCode) & "
was generated by " _
         & Err.Source & vbCrLf &
Err.Description

      cColErrors.Add strError
   End If

   ' Log all database errors, if any
   For Each errLoop In myErrCollection
      With errLoop
         If Err.Number > vbObjectError And
Err.Number < (vbObjectError + 65536) Then
            errCode = .Number - vbObjectError
         Else
            errCode = .Number
         End If
         strError = "Error #" & errCode &
vbCrLf
         strError = strError & "    " &
.Description & vbCrLf
         strError = strError & _
            "    (Source: " & .Source & ")" &
vbCrLf
      End With

      cColErrors.Add strError
   Next

   ' We can have a error handler now that we
have stored all
   ' errors away safely! - having an error
handler before
   ' enumerating all the errors would have
cleared the error
   ' collection
   On Error GoTo LogErrorsErr
   gstrSource = mstrModuleName &
"LogErrors"

   For lngIndex = 0 To cColErrors.Count - 1
      strError = cColErrors(lngIndex)
      Debug.Print strError
      Call WriteToFile(strError, True)
   Next lngIndex

   Set cColErrors = Nothing

   Exit Sub

LogErrorsErr:
   ' Display the error code raised by Visual
Basic
   DisplayErrors Errors
   On Error GoTo 0
   ShowError errUnableToWriteError,
DoWriteError:=False

End Sub
Attribute VB_Name = "FileCommon"
'  FILE:      FileCommon.bas
'             Microsoft TPC-H Kit Ver. 1.00
'             Copyright Microsoft, 1999
'             All Rights Reserved
'
'
'  PURPOSE:    This module contains common
functionality to display
'             the File Open dialog.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

' Used to indicate the source module name when
errors
' are raised by this module
Private Const mstrModuleName As String =
"FileCommon."

Private Enum EOpenFile
   OFN_OVERWRITEPROMPT = &H2
   OFN_HIDEREADONLY = &H4
   OFN_FILEMUSTEXIST = &H1000
   OFN_EXPLORER = &H80000
End Enum

' The locations for the different output files are
presented to
' the user in a list box. These constants are used
while loading the
' data and while reading the data from the list box.
' These constants also represent the different file
types that are
' displayed to the user in File Open dialogs
Public Enum gFileTypes
   gintOutputFile = 0
'   gintLogFile = 1
   gintErrorFile
   gintStepTextFile
   gintOutputCompareFile
   gintDBFile
   gintDBFileNew
   gintImportFile
   gintExportFile
End Enum

Public Const gsSqlFileSuffix = ".sql"
Public Const gsCmdFileSuffix = ".cmd"

Public Const gsOutputFileSuffix = ".out"
Public Const gstrLogFileSuffix = ".log"
Public Const gsErrorFileSuffix = ".err"
Public Function BrowseDBFile() As String
   ' Prompts the user for a database file with the
workspace information
   ' Call CallFileDialog to display the open file
dialog
   BrowseDBFile = CallFileDialog(gintDBFile)

End Function
Public Function CallFileDialog(intFileType As
Integer, _
      Optional ByVal strDefaultFile As String =
gstrEmptyString) As String
   ' This function initializes the values of the filter
property,
   ' the dialog title and flags for the File Open dialog
depending
   ' on the FileType passed in
   ' It then calls ShowFileOpenDialog to set these
properties and
   ' display the File Open dialog to the user

   ' All the properties used by the File Open dialog
are defined
   ' as constants in this function and passed to
ShowFileOpenDialog
   ' as parameters. So if any of the dialog properties
need to be
   ' modified, these constants are what need to be
changed
   Const s_DLG_TITLE_OPEN = "Open"
   Const s_DLG_TITLE_NEW = "New"
   Const s_DLG_TITLE_IMPORT = "Import
From"
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   Const s_DLG_TITLE_EXPORT =
"Export To"

   Const mlng_FILE_STEP_TEXT_FLAGS
= OFN_EXPLORER Or
OFN_FILEMUSTEXIST Or
OFN_HIDEREADONLY
   Const
mlng_FILE_OUTPUT_COMPARE_FLAG
S = mlng_FILE_STEP_TEXT_FLAGS
   Const mlng_FILE_DB_FLAGS =
mlng_FILE_STEP_TEXT_FLAGS
   Const mlng_FILE_OUTPUT_FLAGS =
OFN_EXPLORER Or
OFN_HIDEREADONLY Or
OFN_OVERWRITEPROMPT
   Const mlng_FILE_LOG_FLAGS =
mlng_FILE_OUTPUT_FLAGS
   Const mlng_FILE_ERROR_FLAGS =
mlng_FILE_OUTPUT_FLAGS
   Const mlng_FILE_DB_NEW_FLAGS =
mlng_FILE_OUTPUT_FLAGS

   Const mstr_FILE_ALL_FILTER = "|All
Files (*.*)|*.*"
   Const mstr_FILE_STEP_TEXT_FILTER
= "Query Files (*" & gsSqlFileSuffix & _
         ")|*" & gsSqlFileSuffix &
"|Command Script Files (*" &
gsCmdFileSuffix & _
         ")|*" & gsCmdFileSuffix
   Const
mstr_FILE_OUTPUT_COMPARE_FILTE
R = "Text Files (*.txt)|*.txt"
   Const mstr_FILE_OUTPUT_FILTER =
"Output Files (*)|*.out"
   Const mstr_FILE_LOG_FILTER = "Log
Files (*.log)|*.log"
   Const mstr_FILE_ERROR_FILTER =
"Error Files (*.err)|*.err"
   Const mstr_FILE_DB_FILTER =
"Stepmaster Workspace Files (*" &
gsDefDBFileExt & ")|*" & gsDefDBFileExt

   Dim strFileName As String

   On Error GoTo CallFileDialogErr

   Select Case intFileType
      Case gintStepTextFile
         strFileName =
ShowFileOpenDialog( _

mstr_FILE_STEP_TEXT_FILTER &
mstr_FILE_ALL_FILTER, _
            s_DLG_TITLE_OPEN, _

mlng_FILE_STEP_TEXT_FLAGS, _
            strDefaultFile)

      Case gintOutputCompareFile
         strFileName =
ShowFileOpenDialog( _

mstr_FILE_OUTPUT_COMPARE_FILTE
R & mstr_FILE_ALL_FILTER, _
            s_DLG_TITLE_OPEN, _

mlng_FILE_OUTPUT_COMPARE_FLAG
S, _
            strDefaultFile)

      Case gintOutputFile
         strFileName =
ShowFileOpenDialog( _
            mstr_FILE_OUTPUT_FILTER &
mstr_FILE_ALL_FILTER, _
            s_DLG_TITLE_OPEN, _
            mlng_FILE_OUTPUT_FLAGS, _
            strDefaultFile)

'      Case gintLogFile
'         strFileName = ShowFileOpenDialog( _
'            mstr_FILE_LOG_FILTER &
mstr_FILE_ALL_FILTER, _
'            s_DLG_TITLE_OPEN, _
'            mlng_FILE_LOG_FLAGS, _
'            strDefaultFile)
'
      Case gintErrorFile
         strFileName = ShowFileOpenDialog( _
            mstr_FILE_ERROR_FILTER &
mstr_FILE_ALL_FILTER, _
            s_DLG_TITLE_OPEN, _
            mlng_FILE_ERROR_FLAGS, _
            strDefaultFile)

      Case gintDBFile
         strFileName = ShowFileOpenDialog( _
            mstr_FILE_DB_FILTER &
mstr_FILE_ALL_FILTER, _
            s_DLG_TITLE_OPEN, _
            mlng_FILE_DB_FLAGS, _
            strDefaultFile)

      Case gintDBFileNew
         strFileName = ShowFileOpenDialog( _
            mstr_FILE_DB_FILTER &
mstr_FILE_ALL_FILTER, _
            s_DLG_TITLE_NEW, _
            mlng_FILE_DB_NEW_FLAGS, _
            strDefaultFile)

      Case gintImportFile
         strFileName = ShowFileOpenDialog( _
            mstr_FILE_DB_FILTER &
mstr_FILE_ALL_FILTER, _
            s_DLG_TITLE_IMPORT, _
            mlng_FILE_DB_FLAGS, _
            strDefaultFile)

      Case gintExportFile
         strFileName = ShowFileOpenDialog( _
            mstr_FILE_DB_FILTER &
mstr_FILE_ALL_FILTER, _
            s_DLG_TITLE_EXPORT, _
            mlng_FILE_DB_FLAGS, _
            strDefaultFile)

      Case Else
         BugAssert True, "Incorrect file type
passed in."
         ' Default processing will be for the
output file
         strFileName = ShowFileOpenDialog( _
            mstr_FILE_OUTPUT_FILTER &
mstr_FILE_ALL_FILTER, _
            s_DLG_TITLE_OPEN, _
            mlng_FILE_OUTPUT_FLAGS, _
            strDefaultFile)

   End Select
   CallFileDialog = strFileName

   Exit Function

CallFileDialogErr:
   CallFileDialog = gstrEmptyString
   ' Log the error code raised by Visual Basic
   Call LogErrors(Errors)
   gstrSource = mstrModuleName &
"CallFileDialog"
   Call ShowError(errBrowseFailed)

End Function

VERSION 5.00
Begin VB.Form frmSplash
   BorderStyle    =   3  'Fixed Dialog
   ClientHeight   =   4710
   ClientLeft     =   45
   ClientTop      =   45
   ClientWidth    =   7455
   ControlBox     =   0  'False
   Icon           =   "frmSplash.frx":0000
   LinkTopic      =   "Form1"
   LockControls   =   -1  'True
   MaxButton      =   0  'False
   MinButton      =   0  'False
   ScaleHeight    =   4710
   ScaleWidth     =   7455
   ShowInTaskbar  =   0  'False
   StartUpPosition =  2  'CenterScreen
   Visible        =   0  'False
   Begin VB.Frame fraMainFrame
      Height         =   4590
      Left           =   45
      TabIndex       =   0
      Top            =   -15
      Width          =   7380
      Begin VB.PictureBox picLogo
         Height         =   2385
         Left           =   510
         Picture        =   "frmSplash.frx":0442
         ScaleHeight    =   2325
         ScaleWidth     =   1755
         TabIndex       =   1
         Top            =   855
         Width          =   1815
      End
      Begin VB.Label lblReserved
         AutoSize       =   -1  'True
         Caption        =   "All Rights Reserved."
         Height         =   195
         Left           =   5520
         TabIndex       =   8
         Top            =   4080
         Width          =   1440
      End
      Begin VB.Label lblCopyright
         AutoSize       =   -1  'True
         Caption        =   "Copyright © 1998"
         BeginProperty Font
            Name           =   "Arial"
            Size           =   8.25
            Charset        =   0
            Weight         =   400
            Underline      =   0  'False
            Italic         =   0  'False
            Strikethrough  =   0  'False
         EndProperty
         Height         =   210
         Left           =   5550
         TabIndex       =   7
         Tag            =   "Copyright"
         Top            =   3850
         Width          =   1380
         WordWrap       =   -1  'True
      End
      Begin VB.Label lblCompany
         AutoSize       =   -1  'True
         Caption        =   "Microsoft Corporation"
         BeginProperty Font
            Name           =   "Arial"
            Size           =   8.25
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
        Charset       =  0
        Weight        =  400
        Underline     =  0  'False
        Italic        =  0  'False
        Strikethrough =  0  'False
      EndProperty
      Height        =  210
      Left          =  5460
      TabIndex      =  6
      Tag           =  "Company"
      Top           =  3640
      Width         =  1560
    End
    Begin VB.Label lblRestrictions
      AutoSize      =  -1  'True
      Caption       =  "See License
Agreement for Restrictions"
      Height        =  195
      Left          =  460
      TabIndex      =  5
      Top           =  4080
      Width         =  2790
    End
    Begin VB.Label lblProductName
      AutoSize      =  -1  'True
      Caption       =  "StepMaster"
      BeginProperty Font
        Name          =  "Arial"
        Size          =  32.25
        Charset       =  0
        Weight        =  700
        Underline     =  0  'False
        Italic        =  0  'False
        Strikethrough =  0  'False
      EndProperty
      Height        =  765
      Left          =  2670
      TabIndex      =  4
      Tag           =  "Product"
      Top           =  1200
      Width         =  3495
    End
    Begin VB.Label lblVersion
      Alignment     =  1  'Right Justify
      AutoSize      =  -1  'True
      Caption       =  "Version 2.4"
      BeginProperty Font
        Name          =  "Arial"
        Size          =  12
        Charset       =  0
        Weight        =  700
        Underline     =  0  'False
        Italic        =  0  'False
        Strikethrough =  0  'False
      EndProperty
      Height        =  285
      Left          =  4800
      TabIndex      =  3
      Tag           =  "Version"
      Top           =  2040
      Width         =  1275
    End
    Begin VB.Label lblWarning
      AutoSize      =  -1  'True
      Caption       =  "Do Not Redistribute"
      BeginProperty Font
        Name          =  "Arial"
        Size          =  8.25
        Charset       =  0
        Weight        =  400
        Underline     =  0  'False
        Italic        =  0  'False
        Strikethrough =  0  'False
      EndProperty
      Height        =  210
      Left          =  480
      TabIndex      =  2
      Tag           =  "Warning"
      Top           =  3850
      Width         =  1380
    End
  End
End
Attribute VB_Name = "frmSplash"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
'  FILE:      frmSplash.frm
'             Microsoft TPC-H Kit Ver. 1.00
'             Copyright Microsoft, 1999
'             All Rights Reserved
'
'
'  PURPOSE:   Splash screen for StepMaster
'  Contact:   Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit


Private Sub Form_Load()
'   lblVersion.Caption = "Version " &
App.Major & "." & App.Minor & "." &
App.Revision
   lblProductName.Caption = App.Title
   lblVersion.Caption = GetVersionString

End Sub



VERSION 5.00
Begin VB.Form frmWorkspaceOpen
  BorderStyle     =  3  'Fixed Dialog
  Caption         =  "Open Workspace"
  ClientHeight    =  2550
  ClientLeft      =  45
  ClientTop       =  330
  ClientWidth     =  4695
  LinkTopic       =  "Form1"
  LockControls    =  -1  'True
  MaxButton       =  0  'False
  MinButton       =  0  'False
  ScaleHeight     =  2550
  ScaleWidth      =  4695
  ShowInTaskbar   =  0  'False
  StartUpPosition =  1  'CenterOwner
  Begin VB.CommandButton cmdOK
    Caption         =  "OK"
    Default         =  -1  'True
    Height          =  375
    Left            =  2160
    TabIndex        =  2
    Top             =  2040
    Width           =  1095
  End
  Begin VB.CommandButton cmdCancel
    Cancel          =  -1  'True
    Caption         =  "Cancel"
    Height          =  375
    Left            =  3360
    TabIndex        =  3
    Top             =  2040
    Width           =  1095
  End
  Begin VB.ListBox lstWorkspaces
    Height          =  1425
    ItemData        =
"frmWorkspaceOpen.frx":0000
    Left            =  240
    List            =  "frmWorkspaceOpen.frx":0002
    MultiSelect     =  2  'Extended
    TabIndex        =  1
    Top             =  480
    Width           =  4215
  End
  Begin VB.Label lblWorkspaces
    AutoSize        =  -1  'True
    Caption         =  "Workspace"
    Height          =  195
    Left            =  240
    TabIndex        =  0
    Top             =  120
    Width           =  825
  End
End
Attribute VB_Name = "frmWorkspaceOpen"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
'  FILE:      frmWorkspaceOpen.frm
'             Microsoft TPC-H Kit Ver. 1.00
'             Copyright Microsoft, 1999
'             All Rights Reserved
'
'
'  PURPOSE:   Used to display a list of all
workspaces in a
'             workspace definition file for Open, Import,
Export
'             and Run (in SMRunOnly) workspace
'  Contact:   Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit
#If RUN_ONLY Then
   Private WithEvents cRunOnlyInst As cRunOnly
Attribute cRunOnlyInst.VB_VarHelpID = -1

Private Sub cRunOnlyInst_Done()
   ShowFree
End Sub

Private Sub Run()
   Dim iIndex As Integer

   Set cRunOnlyInst = New cRunOnly

   For iIndex = 0 To Me.lstWorkspaces.ListCount -
1
      If Me.lstWorkspaces.Selected(iIndex) Then
         ShowBusy
         cRunOnlyInst.WorkspaceId =
Me.lstWorkspaces.ItemData(Me.lstWorkspaces.List
Index)
         cRunOnlyInst.WspName =
Me.lstWorkspaces.List(Me.lstWorkspaces.ListInde
x)
         cRunOnlyInst.RunWsp
      End If
   Next iIndex

End Sub

#End If

Private Sub cmdCancel_Click()

   Unload Me

End Sub

Private Sub cmdOK_Click()
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
#If RUN_ONLY Then
   Call Run
#Else
   Call WorkspaceOpenOk
#End If

End Sub


Private Sub lstWorkspaces_DblClick()

   ' A double click on the workspaces list
box is considered
   ' equivalent to selecting an item in the list
and then selecting
   ' the OK command
   Call cmdOK_Click

End Sub

Attribute VB_Name = "IteratorCommon"
'  FILE:      IteratorCommon.bas
'             Microsoft TPC-H Kit Ver. 1.00
'             Copyright Microsoft, 1999
'             All Rights Reserved
'
'
'  PURPOSE:   Contains functionality
common across StepMaster and
'             SMRunOnly, pertaining to iterators
'             Specifically, functions to read
iterators records
'             in the workspace, load them in an
array and so on.
'  Contact:   Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

' Used to indicate the source module name
when errors
' are raised by this module
Private Const mstrModuleName As String =
"IteratorCommon."

Public Const gintMinIteratorSequence As
Integer = 0

Public Sub RangeComplete(vntIterators As
Variant)
   ' This is a debug procedure
   ' Checks if the from, to and step values
are present in
   ' the array

   Dim bReset As Byte
   Dim bShift As Byte
   Dim lngIndex As Long

   ' Set the three lowest order bits to 1
   bReset = 7

   BugAssert IsArray(vntIterators) And Not
IsEmpty(vntIterators), _
         "Iterators not specified!"

   For lngIndex = LBound(vntIterators) To _
         UBound(vntIterators)
      bShift = 1
      bShift = bShift * (2 ^
(vntIterators(lngIndex).IteratorType - 1))

      bReset = bReset Xor bShift
   Next lngIndex

   ' Assert that all the elements are present
   BugAssert bReset = 0, "Range not
completely specified!"

End Sub
Public Sub LoadIteratorsForWsp(cStepsCol As
cArrSteps, _
      ByVal lngWorkspaceId As Long,
rstStepsInWsp As Recordset)
   ' Initializes the step records in with all the
iterator
   ' values for each step

   Dim recIterators As Recordset

   On Error GoTo LoadIteratorsForWspErr

#If QUERY_ALL Then
   Dim dtStart As Date

   dtStart = Now
   Set recIterators =
ReadWspIterators(lngWorkspaceId)

   Call LoadIteratorsArray(cStepsCol,
recIterators)

   recIterators.Close

   BugMessage "QueryAll Read + load took: "
& CStr(DateDiff("s", dtStart, Now))

#Else
   Dim dtStart As Date
   Dim qyIt As DAO.QueryDef
   Dim sSql As String

   dtStart = Now
   If rstStepsInWsp.RecordCount = 0 Then
      Exit Sub
   End If

   ' This method has the advantage that if the
steps are queried right, everything else follows
   sSql = "Select step_id, version_no, type,
iterator_value, " & _
      " sequence_no " & _
      " from iterator_values " & _
      " where step_id = [s_id] " & _
      " and version_no = [ver_no] "

   ' Order the iterators by sequence within a
step
   sSql = sSql & " order by sequence_no "

   Set qyIt =
dbsAttTool.CreateQueryDef(gstrEmptyString,
sSql)
   rstStepsInWsp.MoveFirst

   While Not rstStepsInWsp.EOF

      qyIt.Parameters("s_id").Value =
rstStepsInWsp!step_id
      qyIt.Parameters("ver_no").Value =
rstStepsInWsp!version_no

      Set recIterators =
qyIt.OpenRecordset(dbOpenSnapshot)

      Call LoadIteratorsArray(cStepsCol,
recIterators)
      recIterators.Close
      rstStepsInWsp.MoveNext
   Wend

   qyIt.Close

   BugMessage "Query step at a time Read + load
took: " & CStr(DateDiff("s", dtStart, Now))

#End If

   Exit Sub

LoadIteratorsForWspErr:
   LogErrors Errors
   gstrSource = mstrModuleName &
"LoadIteratorsForWsp"
   On Error GoTo 0
   Err.Raise vbObjectError +
errLoadRsInArrayFailed, _
         gstrSource, _
         LoadResString(errLoadRsInArrayFailed)

End Sub
Private Function ReadWspIterators(ByVal
lngWorkspaceId As Long) As Recordset

   ' This function will return a recordset that is
populated
   ' with the iterators for all the steps in a given
workspace

   Dim recIterators As Recordset
   Dim qyIt As DAO.QueryDef
   Dim strSql As String

   On Error GoTo ReadWspIteratorsErr
   gstrSource = mstrModuleName &
"ReadWspIterators"

   strSql = "Select i.step_id, i.version_no, " & _
      " i.type, i.iterator_value, " & _
      " i.sequence_no " & _
      " from iterator_values i, att_steps a " & _
      " where i.step_id = a.step_id " & _
      " and i.version_no = a.version_no " & _
      " and a.workspace_id = [w_id] " & _
      " and a.archived_flag = [archived] "

   ' Find the highest X-component of the version
number
   strSql = strSql & " AND cint( mid( a.version_no,
1, instr( a.version_no, " & gstrDQ &
gstrVerSeparator & gstrDQ & " ) - 1 ) ) = " & _
      " ( select max( cint( mid( version_no, 1, instr(
version_no, " & gstrDQ & gstrVerSeparator &
gstrDQ & " ) - 1 ) ) ) " & _
      " from att_steps AS d " & _
      " WHERE a.step_id = d.step_id ) "

   ' Find the highest Y-component of the version
number for the highest X-component
   strSql = strSql & " AND cint( mid( a.version_no,
instr( a.version_no, " & gstrDQ & gstrVerSeparator
& gstrDQ & " ) + 1 ) ) = " & _
      " ( select max( cint( mid( version_no, instr(
version_no, " & gstrDQ & gstrVerSeparator &
gstrDQ & " ) + 1 ) ) ) " & _
      " from att_steps AS b " & _
      " Where a.step_id = b.step_id " & _
      " AND cint( mid( version_no, 1, instr(
version_no, " & gstrDQ & gstrVerSeparator &
gstrDQ & " ) - 1 ) ) = " & _
      " ( select max( cint( mid( version_no, 1, instr(
version_no, " & gstrDQ & gstrVerSeparator &
gstrDQ & " ) - 1 ) ) ) " & _
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
      " from att_steps AS c " & _
      " WHERE a.step_id = c.step_id ) ) "


   ' Order the iterators by sequence within a
step
   strSql = strSql & " order by i.step_id,
i.sequence_no "

   Set qyIt =
dbsAttTool.CreateQueryDef(gstrEmptyStrin
g, strSql)
   qyIt.Parameters("w_id").Value =
lngWorkspaceId
   qyIt.Parameters("archived").Value =
False

   Set recIterators =
qyIt.OpenRecordset(dbOpenSnapshot)

   qyIt.Close
   Set ReadWspIterators = recIterators

   Exit Function

ReadWspIteratorsErr:
   ' Log the error code raised by Visual
Basic
   Call LogErrors(Errors)
   On Error GoTo 0
   Err.Raise vbObjectError +
errReadDataFailed, _
         gstrSource,
LoadResString(errReadDataFailed)

End Function
Private Sub LoadIteratorsArray(cStepsCol
As cArrSteps, _
      recIterators As Recordset)
   ' Initializes the step records with the
iterators for
   ' the step

   Dim cNewIt As cIterator
   Dim cStepRec As cStep
   Dim lngStepId As Long

   On Error GoTo LoadIteratorsArrayErr
   gstrSource = mstrModuleName &
"LoadIteratorsArray"

   If recIterators.RecordCount = 0 Then
      Exit Sub
   End If

   recIterators.MoveFirst
   While Not recIterators.EOF
      Set cNewIt = New cIterator

      lngStepId =
CLng(ErrorOnNullField(recIterators,
"step_id"))
      If Not cStepRec Is Nothing Then
         If cStepRec.StepId <> lngStepId
Then
            Set cStepRec =
cStepsCol.QueryStep(lngStepId)
         End If
      Else
         Set cStepRec =
cStepsCol.QueryStep(lngStepId)
      End If

      ' Initialize iterator values
      cNewIt.IteratorType =
CInt(ErrorOnNullField(recIterators, "type"))
      cNewIt.Value =
CStr(ErrorOnNullField(recIterators,
"iterator_value"))
      cNewIt.SequenceNo =
CInt(ErrorOnNullField(recIterators,
"sequence_no"))

      ' Add this record to the array of iterators
      cStepRec.LoadIterator cNewIt

      Set cNewIt = Nothing
      recIterators.MoveNext
   Wend

   Exit Sub

LoadIteratorsArrayErr:
   LogErrors Errors
   gstrSource = mstrModuleName &
"LoadIteratorsArray"
   On Error GoTo 0
   Err.Raise vbObjectError +
errLoadRsInArrayFailed, _
         gstrSource, _

LoadResString(errLoadRsInArrayFailed)

End Sub


Attribute VB_Name = "MsgConfirm"
'  FILE:      MsgConfirm.bas
'            Microsoft TPC-H Kit Ver. 1.00
'            Copyright Microsoft, 1999
'            All Rights Reserved
'
'
'  PURPOSE:    Contains constants for
confirmation messages that
'            will be displayed by StepMaster
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

' A public enum containing the codes for all the
confirmation
' messages that will be used by the project -
each of the codes
' has the prefix, con
Public Enum conConfirmMsgCodes
   conWspDelete = 2000
   conSave
   conStopRun
   conSaveConnect
   conSaveDB
End Enum

' A public enum containing the titles for all the
confirmation
' messages that will be used by the project -
each of the codes
' has the prefix, cont - most confirmation
message codes will
' have a corresponding title code in here
Public Enum conConfirmMsgTitleCodes
   contWspDelete = 3000
   contSave
   contStopRun
   contSaveConnect
   contSaveDB
End Enum
Attribute VB_Name = "OpenFiles"
'  FILE:      OpenFiles.bas
'            Microsoft TPC-H Kit Ver. 1.00
'            Copyright Microsoft, 1999
'            All Rights Reserved
'
'
'  PURPOSE:    This module holds a list of all files
that have been
'            opened by the project. This module is
needed since there
'            is no way to share static data between
different instances
'            of a class.
'            Many procedure in this module do not do
any error handling -
'            this is 'coz it is also used by procedures
that log error
'            messages and any error handler will erase
the collection
'            of errors!

'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit


' Used to indicate the source module name when
errors
' are raised by this class
Private Const mstrModuleName As String =
".OpenFiles."

Private mOpenFiles As cNodeCollections

Private Const mstrTempDir As String = "\Temp\"

' The maximum number of temporary files that we
can create in a
' session
Private Const mlngMaxFileIndex As Long =
999999
Private Const mstrFileIndexFormat As String =
"000000"
Private Const mstrTempFilePrefix As String =
"SM"
Private Const mstrTempFileSuffix As String =
".cmd"

Private Const merrFileNotFound As Long = 76
Private Function GetFileHandle(strFileName) As
cFileInfo

   Dim lngIndex As Long
   Dim blnFileOpen As Boolean

   If Not mOpenFiles Is Nothing Then

      blnFileOpen = False
      For lngIndex = 0 To mOpenFiles.Count - 1
         If mOpenFiles(lngIndex).FileName =
strFileName Then
            blnFileOpen = True
            Exit For
         End If
      Next lngIndex

      If blnFileOpen Then
         Set GetFileHandle = mOpenFiles(lngIndex)
      Else
         Set GetFileHandle = Nothing
      End If
   Else
      Set GetFileHandle = Nothing
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   End If

End Function

Private Function GetTempFileDir() As
String

   Dim strTempFileDir As String

   On Error GoTo GetTempFileDirErr

   strTempFileDir = gstrProjectPath &
mstrTempDir

   If StringEmpty(Dir$(strTempFileDir,
vbDirectory)) Then
      MkDir strTempFileDir
   End If

   GetTempFileDir = strTempFileDir

   Exit Function

GetTempFileDirErr:
   ' Log the error code raised by Visual
Basic
   Call LogErrors(Errors)
   gstrSource = mstrModuleName &
"GetTempFileDir"
   On Error GoTo 0
   Err.Raise vbObjectError +
errProgramError, gstrSource, _
         LoadResString(errProgramError)

End Function
Public Function
MakePathValid(strFileName As String) As
String
   ' Checks if the passed in file path is valid

   Dim strFileDir As String
   Dim strTempDir As String
   Dim strTempFile As String
   Dim intPos As Integer
   Dim intStart As Integer

   On Error GoTo MakePathValidErr
   gstrSource = mstrModuleName &
"MakePathValid"

   strTempFile = strFileName
   intPos = InstrR(strFileName,
gstrFileSeparator)

   If intPos > 0 Then
      strFileDir = Left$(strTempFile, intPos -
1)
      If StringEmpty(Dir$(strFileDir,
vbDirectory)) Then
         ' Loop through the entire path
starting at the root
         ' since Mkdir can create only one
level of sub-directory
         ' at a time
         intStart = InStr(strFileDir,
gstrFileSeparator)

         Do While strTempDir <> strFileDir

            If intStart > 0 Then
               strTempDir = Left$(strFileDir,
intStart - 1)
            Else
               strTempDir = strFileDir
            End If
            If StringEmpty(Dir$(strTempDir,
vbDirectory)) Then
               ' If the specified directory doesn't
exist, try to
               ' create it.
               MkDir strTempDir
            Else
               ' The directory exists - go to it's
sub-directory
            End If
            intStart = InStr(intStart + 1,
strFileDir, gstrFileSeparator)
         Loop

         ' Sanity check
         If StringEmpty(Dir$(strFileDir,
vbDirectory)) Then
            ' We were unable to create the file
directory
            ShowError errCreateDirectoryFailed,
gstrSource, _
                  strFileDir, DoWriteError:=False
            MakePathValid = gstrEmptyString
         Else
            MakePathValid = strTempFile
         End If
      Else
         ' The specified directory exists - we
should be able
         ' to create the output file in it
         MakePathValid = strTempFile
      End If
   Else
      ' The user has only specified a filename -
VB will try
      ' to create it in the current directory
      MakePathValid = strTempFile
   End If

   Exit Function

MakePathValidErr:
   ' Log the error code raised by Visual Basic
   Call LogErrors(Errors)
   gstrSource = mstrModuleName &
"MakePathValid"
   ' Log the filename for debug
   Call WriteError(errInvalidFile, gstrSource,
strTempFile)
   On Error GoTo 0
   Err.Raise vbObjectError + errProgramError,
gstrSource, _
         LoadResString(errProgramError)

End Function
Public Function OpenFileSM(strFileName As
String) As Integer
   Dim intHFile As Integer
   Dim NewFileInfo As cFileInfo

   On Error GoTo OpenFileSMErr
   gstrSource = mstrModuleName &
"OpenFileSM"

   If StringEmpty(strFileName) Then
      On Error GoTo 0
      Err.Raise vbObjectError + errInvalidFile,
gstrSource, _
            LoadResString(errInvalidFile)
   End If

   If mOpenFiles Is Nothing Then
      Set mOpenFiles = New cNodeCollections
   End If

   Set NewFileInfo = GetFileHandle(strFileName)

   If NewFileInfo Is Nothing Then
      ' The file has not been opened yet

      ' If the filename has not been initialized, do not
      ' attempt to open it
      strFileName = MakePathValid(strFileName)

      If strFileName <> gstrEmptyString Then
         intHFile = FreeFile
         Open strFileName For Output Shared As
intHFile

         Set NewFileInfo = New cFileInfo
         NewFileInfo.FileHandle = intHFile
         NewFileInfo.FileName = strFileName
         mOpenFiles.Load NewFileInfo
      Else
         ' Either the directory was invalid or s'thing
failed
         ' Display the error to the user instead of
trying
         ' to log to the file
         ShowError errInvalidFile, gstrSource,
strFileName, _
               DoWriteError:=False
         intHFile = 0
      End If
   Else
      intHFile = NewFileInfo.FileHandle
   End If

   OpenFileSM = intHFile

   Exit Function

OpenFileSMErr:
   ' Log the error code raised by Visual Basic
   Call LogErrors(Errors)
   ' The Open command failed for some reason -
write an error
   ' and let the calling function handle the error
   ShowError errInvalidFile, gstrSource,
strFileName, _
         DoWriteError:=False
   OpenFileSM = 0

End Function
Public Function CreateTempFile() As String

   Dim strTempFileDir As String
   Dim strTempFileName As String

   Static lngLastFileIndex As Long

   On Error GoTo CreateTempFileErr

   strTempFileDir = GetTempFileDir()

   Do
      If lngLastFileIndex = mlngMaxFileIndex Then
         On Error GoTo 0
         Err.Raise vbObjectError +
errMaxTempFiles, gstrSource, _
               LoadResString(errMaxTempFiles)
      End If

      lngLastFileIndex = lngLastFileIndex + 1
      strTempFileName = mstrTempFilePrefix & _
            Format$(lngLastFileIndex,
mstrFileIndexFormat) & _
            mstrTempFileSuffix
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
      If Not
StringEmpty(Dir$(strTempFileDir &
strTempFileName)) Then
          ' Remove any files left over from a
previous run,
          ' if they still exist
          Kill strTempFileDir &
strTempFileName
      End If

   ' Looping in case the file delete doesn't go
through for
   ' some reason
   Loop While Not
StringEmpty(Dir$(strTempFileDir &
strTempFileName))

   CreateTempFile =
GetShortName(strTempFileDir)
   CreateTempFile = CreateTempFile &
strTempFileName

   Exit Function

CreateTempFileErr:
   ' Log the error code raised by Visual
Basic
   Call LogErrors(Errors)
   gstrSource = gstrSource &
"CreateTempFile"
   On Error GoTo 0
   Err.Raise vbObjectError +
errProgramError, gstrSource, _
         LoadResString(errProgramError)

End Function
Public Sub CloseFileSM(strFileName As
String)
   Dim FileToClose As cFileInfo

   If Not mOpenFiles Is Nothing Then

      ' Get the handle to the open file, if it
exists
      Set FileToClose =
GetFileHandle(strFileName)

      If Not FileToClose Is Nothing Then
         Close FileToClose.FileHandle

         ' Remove the file info from the
collection of open files
         mOpenFiles.Unload
FileToClose.Position
      End If
   End If

End Sub
Public Sub CloseOpenFiles()
   Dim lIndex As Long

   If Not mOpenFiles Is Nothing Then
      For lIndex = mOpenFiles.Count - 1 To
0
         CloseFileSM
(mOpenFiles(lIndex).FileName)
      Next lIndex
   End If

End Sub

Attribute VB_Name = "ParameterCommon"
'  FILE:      ParameterCommon.bas
'            Microsoft TPC-H Kit Ver. 1.00
'            Copyright Microsoft, 1999
'            All Rights Reserved
'
'
'  PURPOSE:    Contains functionality common
across StepMaster and
'            SMRunOnly, pertaining to parameters
'            Specifically, functions to load
parameter records
'            in an array.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

' Used to indicate the source module name
when errors
' are raised by this module
Private Const mstrModuleName As String =
"ParameterCommon."

Public Sub
LoadRecordsetInParameterArray(rstWorkSpac
eParameters As Recordset, _
   cParamCol As cArrParameters)

   Dim cNewParameter As cParameter

   On Error GoTo
LoadRecordsetInParameterArrayErr

   If rstWorkSpaceParameters.RecordCount =
0 Then
      Exit Sub
   End If

   rstWorkSpaceParameters.MoveFirst
   While Not rstWorkSpaceParameters.EOF

      Set cNewParameter = New cParameter

      ' Initialize parameter values
      cNewParameter.ParameterId =
rstWorkSpaceParameters.Fields(0)

      ' Call a procedure to raise an error if
mandatory fields are
      ' null.
      cNewParameter.ParameterName = CStr( _

ErrorOnNullField(rstWorkSpaceParameters,
"parameter_name"))
      cNewParameter.ParameterValue =
CheckForNullField( _
            rstWorkSpaceParameters,
"parameter_value")
      cNewParameter.WorkspaceId = CStr( _

ErrorOnNullField(rstWorkSpaceParameters,
FLD_ID_WORKSPACE))
      cNewParameter.ParameterType = CStr( _

ErrorOnNullField(rstWorkSpaceParameters,
"parameter_type"))
      cNewParameter.Description =
CheckForNullField( _
            rstWorkSpaceParameters,
"description")

      cParamCol.Load cNewParameter

      Set cNewParameter = Nothing
      rstWorkSpaceParameters.MoveNext
   Wend

   Exit Sub

LoadRecordsetInParameterArrayErr:
   LogErrors Errors
   gstrSource = mstrModuleName &
"LoadRecordsetInParameterArray"
   On Error GoTo 0
   Err.Raise vbObjectError +
errLoadRsInArrayFailed, gstrSource, _
         LoadResString(errLoadRsInArrayFailed)
End Sub
Attribute VB_Name = "Public"
'  FILE:      Public.bas
'            Microsoft TPC-H Kit Ver. 1.00
'            Copyright Microsoft, 1999
'            All Rights Reserved
'
'
'  PURPOSE:    This module contains all the public
constants for this project
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

Public Const gsVersion01 As String = "0.1"
Public Const gsVersion10 As String = "1.0"
Public Const gsVersion21 As String = "2.1"
Public Const gsVersion23 As String = "2.3"
Public Const gsVersion24 As String = "2.4"
Public Const gsVersion241 As String = "2.4.1"
Public Const gsVersion242 As String = "2.4.2"
Public Const gsVersion243 As String = "2.4.3"
Public Const gsVersion25 As String = "2.5"
Public Const gsVersion251 As String = "2.5.1"
Public Const gsVersion253 As String = "2.5.3"
Public Const gsVersion254 As String = "2.5.4"
Public Const gsVersion255 As String = "2.5.5"
Public Const gsVersion As String = gsVersion255

' The same form is used for the creation of new
nodes and
' updates to existing nodes (where each node can be
a parameter,
' global step, etc.) A tag is set on each flag is used to
indicate
' whether it is being called in the insert or update
mode. The
' constants for these modes are defined below
Public Const gstrInsertMode = "Insert"
Public Const gstrUpdateMode = "Update"
Public Const gstrPropertiesMode = "View"

Public Const gstrEmptyString = ""
Public Const gstrSQ = "'"
Public Const gstrDQ = """"
Public Const gstrVerSeparator = "."
Public Const gstrBlank = " "

' Constants used to indicate type of node being
processed
' The constants for the different objects correspond
to the
' indexes in the menu control arrays (for both the
main and popup
' menus) that are used to create new objects. That
way we can
' use the index passed in by the click event to
determine the
' type of node being processed
Public Const gintWorkspace = 1

' Decided to leave it here after some debate over
whether it
' actually belongs in the cStep class definition
Public Enum gintStepType
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   gintGlobalStep = 3
   gintManagerStep
   gintWorkerStep
End Enum

Public Const gintRunManager = 6
Public Const gintRunWorker = 7

Public Enum gintParameterNodeType
   gintParameter = 8
   gintNodeParamConnection
   gintNodeParamExtension
   gintNodeParamBuiltIn
End Enum

' Leave some constants free for newer types
of parameters (?)
Public Const gintConnectionDtl = 15

Public Enum gintLabelNodeType
   gintGlobalsLabel = 21
   gintParameterLabel
   gintParamConnectionLabel
   gintParamExtensionLabel
   gintParamBuiltInLabel
   gintConnDtlLabel
   gintGlobalStepLabel
   gintStepLabel
End Enum

Public Enum ConnectionType
   ConnTypeStatic = 1
   ConnTypeDynamic
End Enum

Public Const giDefaultConnType As Integer
= ConnTypeStatic

' The constants defined below are used to
identify the different
' tabs. If any more step properties and
thereby tabs are added
' to the tabbed dialog on the Step Properties
form, they should
' be defined here and accessed in the code
only using these
' pre-defined constants
' Note: These constants will mainly be used
by the functions that
' initialize, customize and display the Step
Properties form
Public Const gintDefinition = 0
Public Const gintExecution = 1
Public Const gintMgrDefinition = 2
Public Const gintPreExecutionSteps = 3
Public Const gintPostExecutionSteps = 4
Public Const gintFileLocations = 5

' These constants correspond to the index
values in the imagelist
' associated with the tree view control. The
imagelist contains
' the icons that will be displayed for each
node.
Public Enum TreeImages
   gintImageWorkspaceClosed = 1
   gintImageWorkspaceOpen
   gintImageLabelClosed
   gintImageLabelOpen
   gintImageManagerClosedDis
   gintImageManagerClosedEn
   gintImageManagerOpenDis
   gintImageManagerOpenEn
   gintImageWorkerDis
   gintImageWorkerEn
   gintImageGlobalClosed
   gintImageGlobalOpen
   gintImageParameter
   gintImageRun
   gintImagePending
   gintImageStop
   gintImageDisabled
   gintImageAborted
   gintImageFailed
End Enum

' Public variable used to indicate the name of
the function
' that raises an error
Public gstrSource As String

' Public instances of the different collections
Public gcParameters As cArrParameters
Public gcSteps As cArrSteps
Public gcConstraints As cArrConstraints
Public gcConnections As cConnections
Public gcConnDtls As cConnDtls

' Public constants for the index values of the
different toolbar
' options. Will be used while dynamically
enabling/disabling
' these options.
Public Const tbNew = 1
Public Const tbOpen = 2
Public Const tbSave = 3

Public Const tbCut = 5
Public Const tbCopy = 6
Public Const tbPaste = 7
Public Const tbDelete = 8

Public Const tbProperties = 10
Public Const tbRun = 11
Public Const tbStop = 12

' The initial version #
Public Const gstrMinVersion As String = "0.0"
Public Const gstrGlobalParallelism As String =
"0"
Public Const gintMinParallelism As Integer =
1
Public Const gintMaxParallelism As Integer =
100

' Constant for the minimum identifier, used for
all identifier, viz.
' step, workspace, etc.
Public Const glMinId As Long = 1
Public Const glInvalidId As Long = -1

' A parameter that has a special meaning to
Stepmaster
' The system time will be substituted wherever
it occurs
' (typically as a part of the error, log ... file
names
Public Const gstrTimeStamp As String =
"TIMESTAMP"
Public Const gstrEnvVarSeparator = "%"
Public Const gstrFileSeparator = "\"
Public Const gstrUnderscore = "_"

' Constants used by date and time formatting
functions
Public Const gsTimeSeparator = ":"
Public Const gsDateSeparator = "-"
Public Const gsMsSeparator = "."
Public Const gsDtFormat = "00"
Public Const gsYearFormat = "0000"
Public Const gsTmFormat = "00"
Public Const gsMSecondFormat = "000"

' Default nothing value for a date variable
Public Const gdtmEmpty As Currency = 0

Public Const FMT_WSP_LOG_FILE As String =
"yyyymmdd-hhnnss"

Public gsContCriteria() As String
' Note: Update the initialization of
gsExecutionStatus in Initialize() if the
' InstanceStatus values are modified - also the
boundary checks
Public gsExecutionStatus() As String

Public Const gsConnTypeStatic As String =
"Static"
Public Const gsConnTypeDynamic As String =
"Dynamic"

#If RUN_ONLY Then
Public Const gsCaptionRunWsp As String = "Run
Workspace"
#End If

' Valid operations on a cNode object
Public Enum Operation
   QueryOp = 1
   InsertOp = 2
   UpdateOp = 3
   DeleteOp = 4
End Enum

Attribute VB_Name = "RunCommon"
'  FILE:      RunCommon.bas
'             Microsoft TPC-H Kit Ver. 1.00
'             Copyright Microsoft, 1999
'             All Rights Reserved
'
'
'  PURPOSE:   Contains common functions that are
used during the execution
'             of a workspace.
'  Contact:   Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

' Used to indicate the source module name when
errors
' are raised by this class
Private Const mstrModuleName As String =
".RunCommon."


Public Function GetInstanceItValue(cInstanceRec
As cInstance) As String

   ' Returns the iterator value for the instance, if an
   ' iterator has been defined for it
   Dim cStepIt As cRunColIt
   Dim cRunIterator As cRunItNode

   On Error GoTo GetInstanceItValueErr

   ' Since we create a dummy instance for Disabled
and Pending steps,
   ' doesn't make sense to look at their iterators
   If cInstanceRec.Status <> gintDisabled And
cInstanceRec.Status <> gintPending Then
      Set cStepIt = cInstanceRec.Iterators
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
      If Not
StringEmpty(cInstanceRec.Step.IteratorNam
e) Then
         If cStepIt.Count > 0 Then
            Set cRunIterator = cStepIt(0)
            BugAssert
cRunIterator.IteratorName =
cInstanceRec.Step.IteratorName, _
                  "The first iterator in the
collection is the " & _
                  "one that has been defined for
the step."
            If cRunIterator.IteratorName =
cInstanceRec.Step.IteratorName Then
               GetInstanceItValue =
cRunIterator.Value
            Else
               GetInstanceItValue =
gstrEmptyString
            End If
         Else
            GetInstanceItValue =
gstrEmptyString
         End If
      Else
         GetInstanceItValue =
gstrEmptyString
      End If
   Else
      GetInstanceItValue = gstrEmptyString
   End If

   Exit Function

GetInstanceItValueErr:
   ' Log the error code raised by Visual
Basic
   Call LogErrors(Errors)
   On Error GoTo 0
   gstrSource = mstrModuleName &
"GetInstanceItValue"
   Err.Raise vbObjectError +
errProgramError, gstrSource, _
         LoadResString(errProgramError)

End Function


Attribute VB_Name = "RunInstHelper"
'  FILE:      RunInstHelper.bas
'            Microsoft TPC-H Kit Ver. 1.00
'            Copyright Microsoft, 1999
'            All Rights Reserved
'
'
'  PURPOSE:    This module contains helper
procedures that are called by
'            cRunInst.cls
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

' Used to indicate the source module name
when errors
' are raised by this class
Private Const mstrModuleName As String =
"RunInstHelper."

' Should be equal to the number of steps
defined in cRunInst.cls
Public Const glngNumConcurrentProcesses
As Long = 99
Public Const gintBitsPerByte = 8
Public Function AnyStepRunning(cFreeSteps
As cVectorLng, arrFree() As Byte) As Boolean

   Dim lngIndex As Long
   Dim intPosInByte As Integer
   Dim lngTemp As Long

   ' Check if there are any running instances to
wait for
   If cFreeSteps.Count <>
glngNumConcurrentProcesses Then

      ' For every free step, reset the
corresponding element
      ' in the byte array to 0
      For lngIndex = 0 To cFreeSteps.Count - 1

         lngTemp = cFreeSteps(lngIndex) \
gintBitsPerByte
         intPosInByte = cFreeSteps(lngIndex)
Mod gintBitsPerByte

         arrFree(lngTemp) = arrFree(lngTemp)
Xor 2 ^ intPosInByte
      Next lngIndex

      AnyStepRunning = False

      ' Check if we have a non-zero bit in the
byte array
      For lngIndex = LBound(arrFree) To
UBound(arrFree) Step 1
         If arrFree(lngIndex) <> 0 Then
            ' We are waiting for a step to
complete
            AnyStepRunning = True
            Exit For
         End If
      Next lngIndex

   Else
      AnyStepRunning = False
   End If

End Function


Public Function
OrderConstraints(vntTempCons() As Variant,
_
      intConsType As ConstraintType) As
Variant
   ' Returns a variant containing all the
constraint records in the order
   ' in which they should be executed

   Dim vntTemp As Variant
   Dim lngOuter As Long
   Dim lngInner As Long
   Dim cTempConstraint As cConstraint
   Dim cConstraints() As cConstraint
   Dim lngConsCount As Long
   Dim lngLbound As Long
   Dim lngUbound As Long
   Dim lngStep As Long

   On Error GoTo OrderConstraintsErr

   If intConsType = gintPreStep Then
      ' Since we are travelling up and we need
to execute the constraints
      ' for the top-level steps first, reverse the
order that they
      ' have been stored in the array
      lngLbound = UBound(vntTempCons)
      lngUbound = LBound(vntTempCons)
      lngStep = -1
   Else
      lngLbound = LBound(vntTempCons)
      lngUbound = UBound(vntTempCons)
      lngStep = 1
   End If

   lngConsCount = 0

   For lngOuter = lngLbound To lngUbound Step
lngStep
      vntTemp = vntTempCons(lngOuter)

      If Not IsEmpty(vntTemp) Then
         ' Each of the elements is an array
         For lngInner = LBound(vntTemp) To
UBound(vntTemp) Step 1
            If Not IsEmpty(vntTemp(lngInner)) Then
               Set cTempConstraint =
vntTemp(lngInner)

               If Not cTempConstraint Is Nothing
Then
                  ReDim Preserve
cConstraints(lngConsCount)
                  Set cConstraints(lngConsCount) =
cTempConstraint
                  lngConsCount = lngConsCount + 1
               End If
            End If
         Next lngInner
      End If
   Next lngOuter

   ' Set the return value of the function to the array
of
   ' constraints that has been built above
   If lngConsCount = 0 Then
      OrderConstraints = Empty
   Else
      OrderConstraints = cConstraints()
   End If

   Exit Function

OrderConstraintsErr:
   ' Log the error code raised by Visual Basic
   Call LogErrors(Errors)
   On Error GoTo 0
   Err.Raise vbObjectError +
errExecInstanceFailed, _
         mstrModuleName,
LoadResString(errExecInstanceFailed)

End Function
Attribute VB_Name = "ShellSM"
'  FILE:      ShellSM.bas
'            Microsoft TPC-H Kit Ver. 1.00
'            Copyright Microsoft, 1999
'            All Rights Reserved
'
'
'  PURPOSE:    This module contains a function
that creates a process and
'            waits for it to complete.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

Public Function SyncShell(CommandLine As
String, Optional Timeout As Long, _
   Optional WaitForInputIdle As Boolean) As
Boolean
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   Dim proc As
PROCESS_INFORMATION
   Dim Start As STARTUPINFO
   Dim ret As Long
   Dim nMilliseconds As Long

   BugMessage "Executing: " &
CommandLine
   If Timeout > 0 Then
      nMilliseconds = Timeout
   Else
      nMilliseconds = INFINITE
   End If

   'Initialize the STARTUPINFO structure:
   Start.cb = Len(Start)
   Start.dwFlags =
STARTF_USESHOWWINDOW
   Start.wShowWindow =
SW_SHOWMINNOACTIVE

   'Start the shelled application:
   CreateProcessA 0&, CommandLine, 0&,
0&, 1&, _
      NORMAL_PRIORITY_CLASS, 0&,
0&, Start, proc

   If WaitForInputIdle Then
      'Wait for the shelled application to
finish setting up its UI:
      ret = InputIdle(proc.hProcess,
nMilliseconds)
   Else
      'Wait for the shelled application to
terminate:
      ret =
WaitForSingleObject(proc.hProcess,
nMilliseconds)
   End If

   CloseHandle proc.hProcess

   'Return True if the application finished.
Otherwise it timed out or erred.
   SyncShell = (ret = WAIT_OBJECT_0)
End Function

VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cArrConstraints"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:      cArrConstraints.cls
'            Microsoft TPC-H Kit Ver. 1.00
'            Copyright Microsoft, 1999
'            All Rights Reserved
'
'
'  PURPOSE:    Implements an array of
cConstraint objects.
'            Type-safe wrapper around
cNodeCollections.
'            Also contains additional functions
that determine all the
'            constraints for a step, all
constraints in a workspace,
'            validation functions, etc.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

Private mcarrConstraints As cNodeCollections

' Used to indicate the source module name
when errors
' are raised by this class
Private mstrSource As String
Private Const mstrModuleName As String =
"cArrConstraints."
Public Sub SaveWspConstraints(ByVal
lngWorkspace As Long)
   ' Calls a procedure to commit all changes to
the constraints
   ' in the passed in workspace.

   Call mcarrConstraints.Save(lngWorkspace)

End Sub
Public Property Set ConstraintDB(vdata As
Database)

   Set mcarrConstraints.NodeDB = vdata

End Property
Public Property Get ConstraintDB() As
Database

   Set ConstraintDB =
mcarrConstraints.NodeDB

End Property

Public Sub Modify(cConsToUpdate As
cConstraint)

   ' Modify the constraint record
   Call
mcarrConstraints.Modify(cConsToUpdate)

End Sub
Public Sub
CreateNewConstraintVersion(ByVal lngStepId
As Long, _
      ByVal strNewVersion As String, _
      ByVal strOldVersion As String, _
      ByVal intStepType As Integer)

   ' Does all the processing needed to create
new versions of
   ' all the constraints for a given step
   ' It inserts new constraint records in the
database with
   ' the new version numbers on them
   ' It also updates the version number on all
constraints
   ' for the step in the array to the new version
passed in
   ' Since it handles both global and
manager/worker steps,
   ' it checks for the step_id or global_step_id
fields,
   ' depending on the type of step

   Dim lngIndex As Long
   Dim cUpdateConstraint As cConstraint

   On Error GoTo
CreateNewConstraintVersionErr
   mstrSource = mstrModuleName &
"CreateNewConstraintVersion"

   ' Update the version/global version on
Constraint with the
   ' passed in step/global step id
   For lngIndex = 0 To mcarrConstraints.Count - 1
      Set cUpdateConstraint =
mcarrConstraints(lngIndex)
      If intStepType = gintGlobalStep Then
         If cUpdateConstraint.GlobalStepId =
lngStepId And _
            cUpdateConstraint.IndOperation <>
DeleteOp Then
            cUpdateConstraint.GlobalVersionNo =
strNewVersion

            ' Set the operation to indicate an insert
            cUpdateConstraint.IndOperation =
InsertOp
         End If
      Else
         If cUpdateConstraint.StepId = lngStepId
And _
            cUpdateConstraint.IndOperation <>
DeleteOp Then
            cUpdateConstraint.VersionNo =
strNewVersion

            ' Set the operation to indicate an insert
            cUpdateConstraint.IndOperation =
InsertOp
         End If
      End If
   Next lngIndex

   Exit Sub

CreateNewConstraintVersionErr:
   LogErrors Errors
   gstrSource = mstrModuleName &
"CreateNewConstraintVersion"
   On Error GoTo 0
   Err.Raise vbObjectError +
errCreateNewConstraintVersionFailed, _
         mstrSource, _

LoadResString(errCreateNewConstraintVersionFail
ed)

End Sub
Private Sub Class_Initialize()

   Set mcarrConstraints = New cNodeCollections
   BugMessage "cArrConstraints: Initialize event -
setting Constraint count to 0"

End Sub

Private Sub Class_Terminate()

   Set mcarrConstraints = Nothing
   BugMessage "cArrConstraints: Terminate event
triggered"

End Sub

Public Sub Add(ByVal cConstraintToAdd As
cConstraint)

   Set cConstraintToAdd.NodeDB =
mcarrConstraints.NodeDB

   ' Retrieve a unique constraint identifier
   cConstraintToAdd.ConstraintId =
cConstraintToAdd.NextIdentifier

   ' Call a procedure to load the constraint record in
the array
   Call mcarrConstraints.Add(cConstraintToAdd)
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
End Sub
Public Sub Delete(ByVal cOldConstraint As
cConstraint)

   Dim lngDeleteElement As Long
   Dim cConsToDelete As cConstraint

   lngDeleteElement =
QueryConstraintIndex(cOldConstraint.Const
raintId)
   Set cConsToDelete =
mcarrConstraints(lngDeleteElement)

   Call
mcarrConstraints.Delete(cConsToDelete.Pos
ition)

   Set cConsToDelete = Nothing

End Sub
Private Function
QueryConstraintIndex(lngConstraintId As
Long) _
      As Long

   Dim lngIndex As Integer

   ' Find the element in the array to be
deleted
   For lngIndex = 0 To
mcarrConstraints.Count - 1

      ' Note: The constraint id is not a
primary key field in
      ' the database - there can be multiple
records with the
      ' same constraint_id but for different
versions of a step
      ' However, since we'll always load the
constraint information
      ' for the latest version of a step, we'll
have just one
      ' constraint record with a given
constraint_id
      If
mcarrConstraints(lngIndex).ConstraintId =
lngConstraintId Then
         QueryConstraintIndex = lngIndex
         Exit Function
      End If

   Next lngIndex

   ' Raise error that Constraint has not been
found
   ShowError errConstraintNotFound
   On Error GoTo 0
   Err.Raise vbObjectError +
errConstraintNotFound, mstrSource, _
      LoadResString(errConstraintNotFound)

End Function

Public Function QueryConstraint(ByVal
lngConstraintId As Long) _
   As cConstraint

   ' Returns a cConstraint object with the
property values
   ' corresponding to the Constraint
Identifier, lngConstraintId

   Dim lngQueryElement As Long
   lngQueryElement =
QueryConstraintIndex(lngConstraintId)

   ' Set the return value to the queried
Constraint
   Set QueryConstraint =
mcarrConstraints(lngQueryElement)

End Function

Public Sub LoadConstraints(ByVal
lngWorkspaceId As Long, rstStepsInWsp As
Recordset)

   ' Loads the constraints array with all the
constraints
   ' for the workspace
   Dim recConstraints As Recordset
   Dim qyCons As DAO.QueryDef
   Dim strSql As String
   Dim dtStart As Date

   On Error GoTo LoadConstraintsErr
   mstrSource = mstrModuleName &
"LoadConstraints"

   If rstStepsInWsp.RecordCount = 0 Then
      Exit Sub
   End If

   ' First check if the database object has been
set
   If mcarrConstraints.NodeDB Is Nothing
Then
      On Error GoTo 0
      Err.Raise vbObjectError +
errSetDBBeforeLoad, _
            mstrSource, _

LoadResString(errSetDBBeforeLoad)
   End If

   dtStart = Now

   ' Select based on the global step id since
there might
   ' be constraints for a global step that run are
executed
   ' for the workspace
   ' This method has the advantage that if the
steps are queried right, everything else follows
   strSql = "Select a.constraint_id, a.step_id,
a.version_no, " & _
      " a.constraint_type, a.global_step_id,
a.global_version_no, " & _
      " a.sequence_no, b.workspace_id " & _
      " from step_constraints a, att_steps b " &
_
      " where a.global_step_id = b.step_id " &
_
      " and a.global_version_no = b.version_no
" & _
      " and a.global_step_id = [g_s_id] " & _
      " and a.global_version_no = [g_ver_no] "
& _
      " and b.archived_flag = [archived] "

   ' Find the highest X-component of the
version number
   strSql = strSql & " AND ( a.step_id = 0 or (
cint( mid( a.version_no, 1, instr( a.version_no,
" & gstrDQ & gstrVerSeparator & gstrDQ & "
) - 1 ) ) = " & _
      " ( select max( cint( mid( version_no, 1, instr(
version_no, " & gstrDQ & gstrVerSeparator &
gstrDQ & " ) - 1 ) ) ) " & _
      " from att_steps AS d " & _
      " WHERE a.step_id = d.step_id " & _
      " and d.archived_flag = [archived] ) "

   ' Find the highest Y-component of the version
number for the highest X-component
   strSql = strSql & " AND cint( mid( a.version_no,
instr( a.version_no, " & gstrDQ & gstrVerSeparator
& gstrDQ & " ) + 1 ) ) = " & _
      " ( select max( cint( mid( version_no, instr(
version_no, " & gstrDQ & gstrVerSeparator &
gstrDQ & " ) + 1 ) ) ) " & _
      " from att_steps AS y " & _
      " Where a.step_id = y.step_id " & _
      " AND cint( mid( version_no, 1, instr(
version_no, " & gstrDQ & gstrVerSeparator &
gstrDQ & " ) - 1 ) ) = " & _
      " ( select max( cint( mid( version_no, 1, instr(
version_no, " & gstrDQ & gstrVerSeparator &
gstrDQ & " ) - 1 ) ) ) " & _
      " from att_steps AS c " & _
      " WHERE y.step_id = c.step_id " & _
      " and c.archived_flag = [archived] ) ) ) ) "

   ' Order the constraints by sequence within a given
step
   strSql = strSql & " order by a.sequence_no "

   Set qyCons =
mcarrConstraints.NodeDB.CreateQueryDef(gstrEm
ptyString, strSql)
   qyCons.Parameters("archived").Value = False

   rstStepsInWsp.MoveFirst

   While Not rstStepsInWsp.EOF

      If Not (rstStepsInWsp!global_flag) Then
         qyCons.Close
         BugMessage "Query constraints Read +
load took: " & CStr(DateDiff("s", dtStart, Now))
         Exit Sub
      End If

      qyCons.Parameters("g_s_id").Value =
rstStepsInWsp!step_id
      qyCons.Parameters("g_ver_no").Value =
rstStepsInWsp!version_no

      Set recConstraints =
qyCons.OpenRecordset(dbOpenSnapshot)

      Call
LoadRecordsetInConstraintArray(recConstraints)
      recConstraints.Close

      rstStepsInWsp.MoveNext
   Wend

   qyCons.Close
   BugMessage "Query constraints Read + load
took: " & CStr(DateDiff("s", dtStart, Now))

   Exit Sub

LoadConstraintsErr:
   LogErrors Errors
   gstrSource = mstrModuleName &
"LoadConstraints"
   On Error GoTo 0
   Err.Raise vbObjectError + errLoadDataFailed, _
         mstrSource, _
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
        LoadResString(errLoadDataFailed)

End Sub
Public Sub UnloadStepConstraints(ByVal
lngStepId As Long)

   ' Unloads all the constraints for the
workspace from
   ' the constraints array

   Dim lngIndex As Long

   ' Find all constraints in the array with a
matching step id
   ' It is important to step in reverse order
through the array,
   ' since we delete constraint records!
   For lngIndex = mcarrConstraints.Count -
1 To 0 Step -1
      If
mcarrConstraints(lngIndex).GlobalStepId =
lngStepId Then

         ' Unload the constraint from the
array
         Call
mcarrConstraints.Unload(lngIndex)

      End If
   Next lngIndex

End Sub
Public Sub
UnloadConstraint(cOldConstraint As
cConstraint)
   ' Unloads the constraint from the
constraints array

   Dim lngDeleteElement As Long

   lngDeleteElement =
QueryConstraintIndex(cOldConstraint.Const
raintId)

   Call
mcarrConstraints.Unload(lngDeleteElement)

End Sub
Private Sub
LoadRecordsetInConstraintArray(ByVal
recConstraints As Recordset)
   ' Loads all the constraint records in the
passed in
   ' recordset into the array

   Dim cNewConstraint As cConstraint

   On Error GoTo
LoadRecordsetInConsArrayErr
   mstrSource = mstrModuleName &
"LoadRecordsetInConstraintArray"

   If recConstraints.RecordCount = 0 Then
      Exit Sub
   End If

   recConstraints.MoveFirst
   While Not recConstraints.EOF
      Set cNewConstraint = New cConstraint

      ' Initialize Constraint values
      cNewConstraint.ConstraintId =
CLng(ErrorOnNullField(recConstraints,
"Constraint_id"))
      cNewConstraint.StepId =
CLng(ErrorOnNullField(recConstraints,
"step_id"))
      cNewConstraint.VersionNo =
CStr(ErrorOnNullField(recConstraints,
"version_no"))

      cNewConstraint.GlobalStepId =
CLng(ErrorOnNullField(recConstraints,
"global_step_id"))
      cNewConstraint.GlobalVersionNo =
CStr(ErrorOnNullField(recConstraints,
"global_version_no"))
      cNewConstraint.SequenceNo =
CInt(ErrorOnNullField(recConstraints,
"sequence_no"))

      cNewConstraint.WorkspaceId =
CLng(ErrorOnNullField(recConstraints,
FLD_ID_WORKSPACE))
      cNewConstraint.ConstraintType =
CInt(ErrorOnNullField(recConstraints,
"constraint_type"))

      ' Add this record to the array of
Constraints
      mcarrConstraints.Load cNewConstraint

      Set cNewConstraint = Nothing
      recConstraints.MoveNext
   Wend

   Exit Sub

LoadRecordsetInConsArrayErr:
   LogErrors Errors
   gstrSource = mstrModuleName &
"LoadRecordsetInConstraintArray"
   On Error GoTo 0
   Err.Raise vbObjectError +
errLoadRsInArrayFailed, _
         mstrSource, _

LoadResString(errLoadRsInArrayFailed)

End Sub

Public Function ConstraintsForStep( _
      ByVal lngStepId As Long, _
      ByVal strVersionNo As String, _
      Optional ByVal intConstraintType As
ConstraintType = 0, _
      Optional ByVal blnSort As Boolean =
True, _
      Optional ByVal blnGlobal As Boolean =
False, _
      Optional ByVal
blnGlobalConstraintsOnly As Boolean = False)
_
      As Variant

   ' Returns a variant containing an array of
cConstraint objects,
   ' containing all the constraints that have been
defined for the
   ' given step. If the Global flag is set to true,
the
   ' search will be made for all the constraints
that have
   ' a matching global_step_id

   Dim lngIndex As Long
   Dim cStepConstraint() As cConstraint
   Dim lngConstraintCount As Long
   Dim cTempConstraint As cConstraint

   On Error GoTo ConstraintsForStepErr
   mstrSource = mstrModuleName &
"ConstraintsForStep"

   lngConstraintCount = 0

   ' Find each element in the constraints array
   For lngIndex = 0 To mcarrConstraints.Count - 1
      ' If a constraint type has been specified then
check
      ' if the constraint type for the record matches
the
      ' passed in type
      Set cTempConstraint =
mcarrConstraints(lngIndex)
      If Not blnGlobal Then
         If cTempConstraint.StepId = lngStepId And
_
            cTempConstraint.VersionNo =
strVersionNo And _
            cTempConstraint.IndOperation <>
DeleteOp And _
            (intConstraintType = 0 Or _
            cTempConstraint.ConstraintType =
intConstraintType) Then
            ' We have a matching constraint for the
given step
            AddArrayElement cStepConstraint, _
               cTempConstraint,
lngConstraintCount
         End If
      Else
         If cTempConstraint.GlobalStepId =
lngStepId And _
            cTempConstraint.GlobalVersionNo =
strVersionNo And _
            cTempConstraint.IndOperation <>
DeleteOp Then
            If blnGlobalConstraintsOnly = False Or _
               (blnGlobalConstraintsOnly And _
               cTempConstraint.StepId = 0 And _
               cTempConstraint.VersionNo =
gstrMinVersion) Then

            ' We have a matching constraint for the
global step
            AddArrayElement cStepConstraint, _
               cTempConstraint,
lngConstraintCount
            End If
         End If
      End If

   Next lngIndex

   ' Set the return value of the function to the array
of
   ' constraints that has been built above
   If lngConstraintCount = 0 Then
      ConstraintsForStep = Empty
   Else
      ConstraintsForStep = cStepConstraint()
   End If

   ' Sort the constraints
   If blnSort Then
      Call QuickSort(ConstraintsForStep)
   End If

   Exit Function

ConstraintsForStepErr:
   LogErrors Errors
   On Error GoTo 0
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   Err.Raise vbObjectError +
errConstraintsForStepFailed, _
         mstrSource, _

LoadResString(errConstraintsForStepFailed)

End Function
Private Sub AddArrayElement(ByRef
arrNodes() As cConstraint, _
      ByVal objToAdd As cConstraint, _
      ByRef lngCount As Long)
   ' Adds the passed in object to the array

   ' Increase the array dimension and add the
object to it
   ReDim Preserve arrNodes(lngCount)
   Set arrNodes(lngCount) = objToAdd
   lngCount = lngCount + 1

End Sub

Public Function ConstraintsForWsp( _
      ByVal lngWorkspaceId As Long, _
      Optional ByVal intConstraintType As
Integer = 0, _
      Optional ByVal blnSort As Boolean =
True, _
      Optional ByVal
blnGlobalConstraintsOnly As Boolean =
False) _
       As Variant

   ' Returns a variant containing an array of
cConstraint objects,
   ' containing all the constraints that have
been defined for the
   ' given workspace.

   Dim lngIndex As Long
   Dim cWspConstraint() As cConstraint
   Dim lngConstraintCount As Long
   Dim cTempConstraint As cConstraint

   On Error GoTo ConstraintsForWspErr
   mstrSource = mstrModuleName &
"ConstraintsForWsp"

   lngConstraintCount = 0

   ' Find each element in the constraints
array
   For lngIndex = 0 To
mcarrConstraints.Count - 1
      ' If a constraint type has been specified
then check
      ' if the constraint type for the record
matches the
      ' passed in type
      Set cTempConstraint =
mcarrConstraints(lngIndex)
      If cTempConstraint.WorkspaceId =
lngWorkspaceId And _
            cTempConstraint.IndOperation <>
DeleteOp And _
            (intConstraintType = 0 Or _
            cTempConstraint.ConstraintType
= intConstraintType) Then

         If blnGlobalConstraintsOnly = False
Or _
               (blnGlobalConstraintsOnly And
_
               cTempConstraint.StepId = 0
And _
               cTempConstraint.VersionNo =
gstrMinVersion) Then

            ' We have a matching constraint for
the workspace
            AddArrayElement cWspConstraint,
_
                  cTempConstraint,
lngConstraintCount
         End If
      End If
   Next lngIndex

   ' Set the return value of the function to the
array of
   ' constraints that has been built above
   If lngConstraintCount = 0 Then
      ConstraintsForWsp = Empty
   Else
      ConstraintsForWsp = cWspConstraint()
   End If

   ' Sort the constraints
   If blnSort Then
      Call QuickSort(ConstraintsForWsp)
   End If

   Exit Function

ConstraintsForWspErr:
   LogErrors Errors
   On Error GoTo 0
   Err.Raise vbObjectError +
errConstraintsForWspFailed, _
         mstrSource, _

LoadResString(errConstraintsForWspFailed)

End Function
Public Function PreConstraintsForStep( _
      ByVal lngStepId As Long, _
      ByVal strVersionNo As String, _
      Optional ByVal blnSort As Boolean) As
Variant

   ' Returns a variant containing an array of
cConstraint objects,
   ' containing all the pre-execution constraints
that have
   ' been defined for the given step_id and
version

   ' Call a function that will return a variant
containing
   ' all the constraints of the passed in type
   PreConstraintsForStep =
ConstraintsForStep(lngStepId, _
         strVersionNo, gintPreStep, blnSort)

End Function
Public Function PostConstraintsForStep( _
      ByVal lngStepId As Long, _
      ByVal strVersionNo As String, _
      Optional ByVal blnSort As Boolean) As
Variant

   ' Returns a variant containing an array of
cConstraint objects,
   ' containing all the Post-execution
constraints that have
   ' been defined for the given step_id and
version

   ' Call a function that will return a variant
containing
   ' all the constraints of the passed in type
   PostConstraintsForStep =
ConstraintsForStep(lngStepId, _
         strVersionNo, gintPostStep, blnSort)

End Function
Public Function PostConstraintsForWsp( _
      ByVal lngWorkspaceId As Long, _
      Optional ByVal blnSort As Boolean) As
Variant

   ' Returns a variant containing an array of
cConstraint objects,
   ' containing all the Post-execution globals that
have
   ' been defined for the workspace

   ' Call a function that will return a variant
containing
   ' all the constraints of the passed in type
   PostConstraintsForWsp =
ConstraintsForWsp(lngWorkspaceId, _
         gintPostStep, blnSort, True)

End Function
Public Function PreConstraintsForWsp( _
      ByVal lngWorkspaceId As Long, _
      Optional ByVal blnSort As Boolean) As
Variant

   ' Returns a variant containing an array of
cConstraint objects,
   ' containing all the Pre-execution globals that
have
   ' been defined for the workspace

   ' Call a function that will return a variant
containing
   ' all the constraints of the passed in type
   PreConstraintsForWsp =
ConstraintsForWsp(lngWorkspaceId, _
         gintPreStep, blnSort, True)

End Function
Public Property Get ConstraintCount() As Long

   ConstraintCount = mcarrConstraints.Count

End Property

VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cArrParameters"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:      cArrParameters.cls
'            Microsoft TPC-H Kit Ver. 1.00
'            Copyright Microsoft, 1999
'            All Rights Reserved
'
'
'  PURPOSE:    Implements an array of cParameter
objects.
'            Type-safe wrapper around
cNodeCollections.
'            Also contains additional functions to
determine parameter
'            values, validation functions, etc.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
Option Explicit

Private mcarrParameters As
cNodeCollections

' Used to indicate the source module name
when errors
' are raised by this class
Private mstrSource As String
Private Const mstrModuleName As String =
"cArrParameters."

Public Sub InitBuiltInsForRun(lWspId As
Long, lRunId As Long)

   Dim cParamRec As cParameter

   ' Initialize the values of the run_id and
output_dir built-in parameters and save them
   ' to the database
   Set cParamRec =
GetParameterValue(lWspId,
PARAM_RUN_ID)
   cParamRec.ParameterValue =
CStr(lRunId)
   Call Modify(cParamRec)

   Set cParamRec =
GetParameterValue(lWspId,
PARAM_OUTPUT_DIR)
   cParamRec.ParameterValue =
GetDefaultDir(lWspId, Me)
   cParamRec.ParameterValue =
cParamRec.ParameterValue &
gstrFileSeparator & CStr(lRunId)
   Call Modify(cParamRec)

   Call SaveParametersInWsp(lWspId)

End Sub


Public Property Set ParamDatabase(vdata
As Database)

   Set mcarrParameters.NodeDB = vdata

End Property
Public Sub Modify(cModifiedParam As
cParameter)

   ' First check if the parameter record is
valid
   Call
CheckDupParamName(cModifiedParam)

   Call
mcarrParameters.Modify(cModifiedParam)

End Sub
Public Sub Load(ByRef cParamToAdd As
cParameter)

   Call
mcarrParameters.Load(cParamToAdd)

End Sub
Public Sub Add(ByRef cParamToAdd As
cParameter)

   Set cParamToAdd.NodeDB =
mcarrParameters.NodeDB

   ' First check if the parameter record is
valid
   Call Validate(cParamToAdd)

   ' Retrieve a unique parameter identifier
   cParamToAdd.ParameterId =
cParamToAdd.NextIdentifier

   Call mcarrParameters.Add(cParamToAdd)

End Sub

Public Sub Unload(lngParamToDelete As
Long)

   Dim lngDeleteElement As Long

   lngDeleteElement =
QueryIndex(lngParamToDelete)

   Call
mcarrParameters.Unload(lngDeleteElement)

End Sub

Public Sub SaveParametersInWsp(ByVal
lngWorkspace As Long)
   ' Calls a procedure to commit all changes to
the parameters
   ' for the passed in workspace.

   ' Call a procedure to save all parameter
records for the
   ' workspace
   Call mcarrParameters.Save(lngWorkspace)

End Sub
Public Function GetParameterValue(ByVal
lngWorkspace As Long, _
      ByVal strParamName As String) As
cParameter
   ' Returns the value for the passed in
workspace parameter

   Dim cParamRec As cParameter
   Dim lngIndex As Long

   On Error GoTo GetParameterValueErr

   ' Find all parameters in the array with a
matching workspace id
   For lngIndex = 0 To mcarrParameters.Count
- 1
      Set cParamRec =
mcarrParameters(lngIndex)
      If cParamRec.WorkspaceId =
lngWorkspace And _
            cParamRec.ParameterName =
strParamName Then

         Set GetParameterValue = cParamRec
         Exit For
      End If
   Next lngIndex

   If lngIndex > mcarrParameters.Count - 1
Then
      ' The parameter has not been defined for
the workspace
      ' Raise an error
      On Error GoTo 0
      Err.Raise vbObjectError +
errParamNameInvalid, _
            mstrModuleName &
"GetParameterValue", _

LoadResString(errParamNameInvalid)
   End If

   Exit Function

GetParameterValueErr:
   ' Log the error code raised by Visual Basic
   Call LogErrors(Errors)
   gstrSource = mstrModuleName &
"GetParameterValue"
   On Error GoTo 0
   Err.Raise vbObjectError +
errGetParamValueFailed, _
         gstrSource, _
         LoadResString(errGetParamValueFailed)

End Function
Public Sub Delete(lngParamToDelete As Long)
   ' Delete the passed in parameter

   Dim lngDeleteElement As Long

   lngDeleteElement =
QueryIndex(lngParamToDelete)
   Call mcarrParameters.Delete(lngDeleteElement)

End Sub
Private Function QueryIndex(lngParameterId As
Long) As Long

   Dim lngIndex As Long

   ' Find the matching parameter record in the array
   For lngIndex = 0 To mcarrParameters.Count - 1
      If mcarrParameters(lngIndex).ParameterId =
lngParameterId And _
            mcarrParameters(lngIndex).IndOperation
<> DeleteOp Then
         QueryIndex = lngIndex
         Exit Function
      End If
   Next lngIndex

   ' Raise error that parameter has not been found
   On Error GoTo 0
   Err.Raise vbObjectError + errParamNotFound,
"cArrParameters.QueryIndex", _
      LoadResString(errParamNotFound)

End Function

Public Function QueryParameter(lngParameterId
As Long) _
      As cParameter

   Dim lngQueryElement As Long

   lngQueryElement = QueryIndex(lngParameterId)

   ' Return the queried parameter object
   Set QueryParameter =
mcarrParameters(lngQueryElement)

End Function
Public Property Get ParameterCount() As Long

   ParameterCount = mcarrParameters.Count

End Property
Public Property Get Item(lngIndex As Long) As
cParameter
Attribute Item.VB_UserMemId = 0

   Set Item = mcarrParameters(lngIndex)

End Property
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
Public Sub Validate(ByVal
cParamToValidate As cParameter)
   ' This procedure is necessary since the
class cannot validate
   ' all the parameter properties on it's own.
This is 'coz we
   ' might have created new parameters in
the workspace, but not
   ' saved them to the database yet - hence
the duplicate check
   ' has to be repeated in the array

   Dim lngIndex As Long
   Dim cTempParam As cParameter

   On Error GoTo ValidateErr

   ' Check if the parameter name already
exists in the workspace
   For lngIndex = 0 To
mcarrParameters.Count - 1
      Set cTempParam =
mcarrParameters(lngIndex)
      If cTempParam.WorkspaceId =
cParamToValidate.WorkspaceId And _
            cTempParam.ParameterName =
cParamToValidate.ParameterName And _
            cTempParam.IndOperation <>
DeleteOp Then
         On Error GoTo 0
         Err.Raise vbObjectError +
errDuplicateParameterName, _
            mstrSource,
LoadResString(errDuplicateParameterName
)
      End If
   Next lngIndex

   Exit Sub

ValidateErr:
   LogErrors Errors
   mstrSource = mstrModuleName &
"Validate"
   On Error GoTo 0
   Err.Raise vbObjectError +
errValidateFailed, _
      mstrSource,
LoadResString(errValidateFailed)

End Sub
Public Sub CheckDupParamName(ByVal
cParamToValidate As cParameter)

   Dim lngIndex As Long
   Dim cTempParam As cParameter

   ' Check if the parameter name already
exists in the workspace
   For lngIndex = 0 To
mcarrParameters.Count - 1
      Set cTempParam =
mcarrParameters(lngIndex)
      If cTempParam.WorkspaceId =
cParamToValidate.WorkspaceId And _
            cTempParam.ParameterName =
cParamToValidate.ParameterName And _
            cTempParam.ParameterId <>
cParamToValidate.ParameterId And _
            cTempParam.IndOperation <>
DeleteOp Then
         ShowError
errDuplicateParameterName
         On Error GoTo 0
         Err.Raise vbObjectError +
errDuplicateParameterName, _
            mstrSource,
LoadResString(errDuplicateParameterName)
      End If
   Next lngIndex

End Sub

Private Sub Class_Initialize()

   'bugmessage "cArrParameters: Initialize
event - setting parameter count to 0"
   Set mcarrParameters = New
cNodeCollections

End Sub


Private Sub Class_Terminate()

   Set mcarrParameters = Nothing

End Sub
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cArrSteps"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:      cArrSteps.cls
'            Microsoft TPC-H Kit Ver. 1.00
'            Copyright Microsoft, 1999
'            All Rights Reserved
'
'
'  PURPOSE:    Implements an array of cStep
objects.
'            Type-safe wrapper around
cNodeCollections.
'            Also contains additional functions to
update parent version
'            on substeps, validation functions, etc.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

Private mcarrSteps As cNodeCollections

' Used to indicate the source module name
when errors
' are raised by this class
Private mstrSource As String
Private Const mstrModuleName As String =
"cArrSteps."

Public Sub Unload(lngStepToDelete As Long)

   Dim lngDeleteElement As Long
   Dim cUnloadStep As cStep

   lngDeleteElement =
QueryStepIndex(lngStepToDelete)
   Set cUnloadStep =
QueryStep(lngStepToDelete)

   ' First unload all iterators for the step
   Call cUnloadStep.UnloadIterators

   ' Unload the step from the collection
   Call mcarrSteps.Unload(lngDeleteElement)

End Sub
Public Sub Modify(cModifiedStep As cStep)

   Dim iAppend As Integer
   Dim sLabel As String

   On Error GoTo ModifyErr

   iAppend = 0
   sLabel = cModifiedStep.StepLabel

   Validate cModifiedStep

   Call mcarrSteps.Modify(cModifiedStep)

   Exit Sub

ModifyErr:
   ' If the error raised by the add function is due to a
duplication
   ' of the step label, then try to generate a unique
label
   If Err.Number - vbObjectError =
errStepLabelUnique Then
      iAppend = iAppend + 1
      cModifiedStep.StepLabel = sLabel &
CStr(iAppend)
      '  Try to insert the step record again
      Resume
   End If

   ' Log the error code raised by Visual Basic
   Call LogErrors(Errors)
   On Error GoTo 0
   Err.Raise vbObjectError + errModifyStepFailed,
_
         gstrSource, _
         LoadResString(errModifyStepFailed)

End Sub
Public Sub UpdateParentVersion(ByVal lngStepId
As Long, _
      ByVal strNewVersion As String, _
      ByVal strOldVersion As String, _
      ByVal intStepType As Integer)

   ' Does all the processing needed to update the
parent version
   ' number on all the sub-steps for a given step
   ' It updates the parent version no in the database
for all
   ' sub-steps of the passed in step id
   ' It also updates the parent version number on all
sub-steps
   ' in the array to the new version passed in

   Dim lngIndex As Long
   Dim cUpdateStep As cStep

   On Error GoTo UpdateParentVersionErr

   If intStepType <> gintManagerStep Then
      ' Only a manager can have sub-steps - if the
passed
      ' in step is not a manager, exit
      Exit Sub
   End If

   ' For all steps in the array
   For lngIndex = 0 To mcarrSteps.Count - 1

      Set cUpdateStep = mcarrSteps(lngIndex)
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
      ' If the current step is a sub-step of the
passed in step
      If cUpdateStep.ParentStepId =
lngStepId And _
            cUpdateStep.ParentVersionNo =
strOldVersion And _
            Not cUpdateStep.ArchivedFlag
Then

         ' Update the parent version number
for the sub-step
         ' in the array
         cUpdateStep.ParentVersionNo =
strNewVersion

         ' Update the parent version number
for the sub-step
         ' in the array
         Call Modify(cUpdateStep)

      End If
   Next lngIndex

   Exit Sub

UpdateParentVersionErr:
   LogErrors Errors
   mstrSource = mstrModuleName &
"UpdateParentVersion"
   On Error GoTo 0
   Err.Raise vbObjectError +
errUpdateParentVersionFailed, _
         mstrSource, _

LoadResString(errUpdateParentVersionFail
ed)

End Sub
Private Sub Validate(cCheckStep As cStep)

   ' Step validations that depend on other
steps in the collection

   Dim lngIndex As Long

   ' Ensure that the step label is unique in the
workspace
   For lngIndex = 0 To mcarrSteps.Count - 1

      ' If the current step is a sub-step of the
passed in step
      If mcarrSteps(lngIndex).WorkspaceId
= cCheckStep.WorkspaceId And _
            mcarrSteps(lngIndex).StepLabel =
cCheckStep.StepLabel And _
            mcarrSteps(lngIndex).StepId <>
cCheckStep.StepId And _

mcarrSteps(lngIndex).IndOperation <>
DeleteOp Then
         On Error GoTo 0
         Err.Raise vbObjectError +
errStepLabelUnique, _
               mstrModuleName &
"Validate", _

LoadResString(errStepLabelUnique)
      End If
   Next lngIndex

End Sub

Public Sub ValidateStep(cCheckStep As
cStep)

   On Error GoTo ValidateStepErr

   'Public wrapper for Validate function (2
many Validates)
   Call Validate(cCheckStep)

   Exit Sub

ValidateStepErr:
   ShowError errStepLabelUnique
   On Error GoTo 0
   Err.Raise vbObjectError +
errValidateFailed, _
         gstrSource, _
         LoadResString(errValidateFailed)
End Sub

Private Sub Class_Initialize()

   BugMessage "cArrSteps: Initialize event -
setting step count to 0"
   Set mcarrSteps = New cNodeCollections

End Sub

Private Sub Class_Terminate()

   BugMessage "cArrSteps: Terminate event
triggered"
   Set mcarrSteps = Nothing

End Sub

Public Sub Add(ByVal cStepToAdd As cStep)

   Dim iAppend As Integer
   Dim sLabel As String

   On Error GoTo AddErr

   iAppend = 0
   sLabel = cStepToAdd.StepLabel

   Set cStepToAdd.NodeDB =
mcarrSteps.NodeDB

   ' Retrieve a unique step identifier
   cStepToAdd.StepId =
cStepToAdd.NextStepId

   Validate cStepToAdd

   ' Call a procedure to add the step record
   Call mcarrSteps.Add(cStepToAdd)

   ' Call a procedure to add all iterators for the
step
   cStepToAdd.AddAllIterators

   Exit Sub

AddErr:
   ' If the error raised by the add function is due
to a duplication
   ' of the step label, then try to generate a
unique label
   If Err.Number - vbObjectError =
errStepLabelUnique Then
      iAppend = iAppend + 1
      cStepToAdd.StepLabel = sLabel &
CStr(iAppend)
      ' Try to insert the step record again
      Resume
   End If
   ' Log the error code raised by Visual Basic
   Call LogErrors(Errors)
   On Error GoTo 0
   Err.Raise vbObjectError + errInsertStepFailed, _
         gstrSource, _
         LoadResString(errInsertStepFailed)

End Sub
Public Sub Load(cStepToLoad As cStep)

   Call mcarrSteps.Load(cStepToLoad)

End Sub
Public Sub SaveStepsInWsp(ByVal lngWorkspace
As Long)
   ' Calls a procedure to commit all changes to the
steps
   ' in the passed in workspace.

   Dim lngIndex As Integer

   ' Find all steps in the array with a matching
workspace id
   ' It is important to step in reverse order through
the array,
   ' since we delete step records sometimes!
   For lngIndex = mcarrSteps.Count - 1 To 0 Step -
1
      If mcarrSteps(lngIndex).WorkspaceId =
lngWorkspace Then

         ' Call a procedure to commit all changes to
the
         ' Step record, if any
         Call CommitStep(mcarrSteps(lngIndex),
lngIndex)

      End If
   Next lngIndex

End Sub
Private Sub CommitStep(ByVal cCommitStep As
cStep, _
      ByVal intIndex As Integer)
   ' This procedure checks if any changes have been
made to the
   ' passed in Step. If so, it calls the step methods to
commit
   ' the changes.

   ' First commit all changes to the iterator records
for
   ' the step
   cCommitStep.SaveIterators

   Call mcarrSteps.Commit(cCommitStep, intIndex)

End Sub
Public Sub Delete(lngStepToDelete As Long)

   Dim lngDeleteElement As Long

   lngDeleteElement =
QueryStepIndex(lngStepToDelete)
   Call mcarrSteps.Delete(lngDeleteElement)

End Sub
Public Function QueryStepIndex(lngStepId As
Long) As Long

   Dim lngIndex As Long

   ' Find the element in the array that corresponds to
the
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
    ' passed in step id - note that while there
will be multiple
    ' versions of a step in the database, only
one version will
    ' be currently loaded in the array -
meaning that the stepid
    ' is enough to uniquely identify a step
    For lngIndex = 0 To mcarrSteps.Count - 1
        If mcarrSteps(lngIndex).StepId =
lngStepId Then
            QueryStepIndex = lngIndex
            Exit Function
        End If
    Next lngIndex

    ' Raise error that step has not been found
    On Error GoTo 0
    Err.Raise vbObjectError +
errStepNotFound, mstrSource, _
        LoadResString(errStepNotFound)

End Function

Public Function QueryStep(ByVal lngStepId
As Long) As cStep

    ' Populates the passed in cStep object with
the property
    ' values corresponding to the Step
Identifier, lngStepId

    Dim lngQueryElement As Integer

    lngQueryElement =
QueryStepIndex(lngStepId)

    ' Initialize the passed in step object to the
queried step
    Set QueryStep =
mcarrSteps(lngQueryElement)

End Function
Public Property Get Item(ByVal Position As
Long) As cStep
Attribute Item.VB_UserMemId = 0

    ' Returns the element at the passed in
position in the array
    If Position >= 0 And Position <
mcarrSteps.Count Then
        Set Item = mcarrSteps(Position)
    Else
        On Error GoTo 0
        Err.Raise vbObjectError +
errItemDoesNotExist, mstrSource, _

LoadResString(errItemDoesNotExist)
    End If

End Property
Public Property Set Item(ByVal Position As
Long, _
        ByVal cStepRec As cStep)

    ' Returns the element at the passed in
position in the array
    If Position >= 0 And Position <
mcarrSteps.Count Then
        Set mcarrSteps(Position) = cStepRec
    Else
        On Error GoTo 0
        Err.Raise vbObjectError +
errItemDoesNotExist, mstrSource, _

LoadResString(errItemDoesNotExist)
    End If

End Property
Public Property Set StepDB(vdata As
Database)

    Set mcarrSteps.NodeDB = vdata

End Property
Public Function SubSteps(ByVal lngStepId As
Long, _
        ByVal strVersionNo As String) As
Variant

    ' Returns a variant containing an array of all
the substeps
    ' for the passed in step

    Dim intIndex As Integer
    Dim cSubSteps() As cStep
    Dim lngStepCount As Long
    Dim cQueryStep As cStep

    On Error GoTo SubStepsErr

    lngStepCount = 0

    Set cQueryStep = QueryStep(lngStepId)

    ' Only a manager can have sub-steps
    If cQueryStep.StepType = gintManagerStep
Then

        ' For each element in the Steps array
        For intIndex = 0 To mcarrSteps.Count - 1
            ' Check if the parent step id and parent
version number
            ' match the passed in step
            If mcarrSteps(intIndex).ParentStepId =
lngStepId And _

mcarrSteps(intIndex).ParentVersionNo =
strVersionNo And _

mcarrSteps(intIndex).IndOperation <>
DeleteOp Then

                ' Increase the array dimension and
add the step
                ' to it
                ReDim Preserve
cSubSteps(lngStepCount)
                Set cSubSteps(lngStepCount) =
mcarrSteps(intIndex)
                lngStepCount = lngStepCount + 1

            End If
        Next intIndex

    End If

    ' Set the return value of the function to the
array of
    ' Steps that has been built above
    If lngStepCount = 0 Then
        SubSteps = Empty
    Else
        SubSteps = cSubSteps()
    End If

    Exit Function

SubStepsErr:
    LogErrors Errors
    mstrSource = mstrModuleName & "SubSteps"
    On Error GoTo 0
    Err.Raise vbObjectError + errSubStepsFailed, _
            mstrSource, _
            LoadResString(errSubStepsFailed)

End Function

Public Property Get StepCount() As Integer

    StepCount = mcarrSteps.Count

End Property

VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cAsyncShell"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'-----------------------------------------------------------------
' Copyright © 1997 Microsoft Corporation. All
rights reserved.
'
' You have a royalty-free right to use, modify,
reproduce and distribute the
' Sample Application Files (and/or any modified
version) in any way you find
' useful, provided that you agree that Microsoft has
no warranty, obligations or
' liability for any Sample Application Files.
'-----------------------------------------------------------------

Option Explicit

' Used to indicate the source module name when
errors
' are raised by this class
Private mstrSource As String
Private Const mstrModuleName As String =
"cAsyncShell."

Public Event Terminated()

Private WithEvents moTimer As cTimerSM
Attribute moTimer.VB_VarHelpID = -1
Private proc As PROCESS_INFORMATION
Private mfShelling As Boolean


'-----------------------------------------------------------------
'Initialization and cleanup:

Private Sub Class_Initialize()
    Set moTimer = New cTimerSM
End Sub

Private Sub Class_Terminate()
    If mfShelling Then CloseHandle proc.hProcess
End Sub

'-----------------------------------------------------------------
'Shelling:

Public Sub Shell(CommandLine As String,
Optional PollingInterval As Long = 1000)
    Dim Start As STARTUPINFO
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   If mfShelling Then
      On Error GoTo 0
      Err.Raise vbObjectError +
errInstanceInUse, _
            mstrSource, _
            LoadResString(errInstanceInUse)
   End If
   mfShelling = True

   ' Initialize the STARTUPINFO structure:
   Start.cb = Len(Start)
   Start.dwFlags =
STARTF_USESHOWWINDOW
   Start.wShowWindow =
SW_SHOWMINNOACTIVE

   ' Start the shelled application:
   CreateProcessA 0&, CommandLine, 0&,
0&, 1&, _
      NORMAL_PRIORITY_CLASS, 0&,
0&, Start, proc

   With moTimer
      If PollingInterval > 0 Then
         .Interval = PollingInterval
      Else
         .Interval = 1000
      End If
      .Enabled = True
   End With
End Sub
'-----------------------------------------------------
--------------------------
'Aborting:
Public Sub Abort()
   Dim nCode As Long
'    Dim X As Integer
'    Dim ReturnVal As Integer

   On Error GoTo AbortErr

   If Not mfShelling Then
      Call WriteError(errProgramError,
mstrSource)
   Else
'       If IsWindow(proc.hProcess) = False
Then Exit Sub
'
'       If (GetWindowLong(proc.hProcess,
GWL_STYLE) And WS_DISABLED)
Then Exit Sub
'
'       If IsWindow(proc.hProcess) Then
'           If Not
(GetWindowLong(proc.hProcess,
GWL_STYLE) And WS_DISABLED)
Then
'               X = PostMessage(proc.hProcess,
WM_CANCELMODE, 0, 0&)
'               X = PostMessage(proc.hProcess,
WM_CLOSE, 0, 0&)
'           End If
'       End If

       If TerminateProcess(proc.hProcess,
0&) = 0 Then
          Debug.Print "Unable to terminate
process: " & proc.hProcess
          Call
WriteError(errTerminateProcessFailed,
mstrSource, _
                ApiError(GetLastError()))
       Else
          ' Should always come here!
          GetExitCodeProcess proc.hProcess,
nCode
          If nCode = STILL_ACTIVE Then
             ' Write an error and close the handles
to the
             ' process anyway
             Call
WriteError(errTerminateProcessFailed,
mstrSource)
          End If
       End If

       ' Close all open handles to the shelled
process, even
       ' if any of the above calls error out
       CloseHandle proc.hProcess
       moTimer.Enabled = False
       mfShelling = False
       RaiseEvent Terminated

   End If

   Exit Sub

AbortErr:
   Call LogErrors(Errors)
   mstrSource = mstrModuleName & "Abort"
   On Error GoTo 0
   Err.Raise vbObjectError + errProgramError,
_
      mstrSource, _
      LoadResString(errProgramError)

End Sub
Private Sub moTimer_Timer()
   Dim nCode As Long

   GetExitCodeProcess proc.hProcess, nCode
   If nCode <> STILL_ACTIVE Then
      CloseHandle proc.hProcess
      moTimer.Enabled = False
      mfShelling = False
      RaiseEvent Terminated
   End If
End Sub
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cConnDtl"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:       cConnDtl.cls
'              Microsoft TPC-H Kit Ver. 1.00
'              Copyright Microsoft, 1999
'              All Rights Reserved
'
'
'  PURPOSE:    Encapsulates the properties
and methods of a connection.
'              Contains functions to insert, update
and delete
'              connection_dtls records from the
database.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit
Option Base 0

' Local variable(s) to hold property value(s)
Public WorkspaceId As Long
Public ConnNameId As Long
Public ConnName As String
Public ConnectionString As String
Public ConnType As ConnectionType
Public Position As Long
Public NodeDB As Database

Private mintOperation As Operation

' Used to indicate the source module name when
errors
' are raised by this class
Private mstrSource As String
Private Const mstrModuleName As String =
"cConnDtl."

' The cSequence class is used to generate unique
Connection identifiers
Private mConnectionSeq As cSequence

' The StringSM class is used to carry out string
operations
Private mFieldValue As cStringSM

Private Sub AssignParameters(qyExec As
DAO.QueryDef)
   ' Assigns values to the parameters in the querydef
object
   ' The parameter names are cryptic to differentiate
them from the field names.
   ' When the parameter names are the same as the
field names, parameters in the where
   ' clause do not get created.

   Dim prmParam As DAO.Parameter

   On Error GoTo AssignParametersErr

   For Each prmParam In qyExec.Parameters
      Select Case prmParam.Name
         Case "[w_id]"
            prmParam.Value = WorkspaceId

         Case "[c_id]"
            prmParam.Value = ConnNameId

         Case "[c_name]"
            prmParam.Value = ConnName

         Case "[c_str]"
            prmParam.Value = ConnectionString

         Case "[c_type]"
            prmParam.Value = ConnType

         Case Else
            ' Write the parameter name that is faulty
            WriteError errInvalidParameter,
mstrSource, prmParam.Name
            On Error GoTo 0
            Err.Raise errInvalidParameter,
mstrModuleName & "AssignParameters", _
                LoadResString(errInvalidParameter)
      End Select
   Next prmParam

   Exit Sub

AssignParametersErr:

   Call LogErrors(Errors)
   On Error GoTo 0
   Err.Raise vbObjectError +
errAssignParametersFailed, _
      mstrModuleName & "AssignParameters",
LoadResString(errAssignParametersFailed)
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
End Sub

Public Function Clone() As cConnDtl

   ' Creates a copy of a given Connection

   Dim cCloneConn As cConnDtl

   On Error GoTo CloneErr

   Set cCloneConn = New cConnDtl

   ' Copy all the Connection properties to the
newly created Connection
   cCloneConn.WorkspaceId = WorkspaceId
   cCloneConn.ConnNameId =
ConnNameId
   cCloneConn.ConnName = ConnName
   cCloneConn.ConnectionString =
ConnectionString
   cCloneConn.ConnType = ConnType
   cCloneConn.IndOperation =
mintOperation
   cCloneConn.Position = Position

   ' And set the return value to the newly
created Connection
   Set Clone = cCloneConn
   Set cCloneConn = Nothing

   Exit Function

CloneErr:
   LogErrors Errors
   mstrSource = mstrModuleName &
"Clone"
   On Error GoTo 0
   Err.Raise vbObjectError +
errCloneFailed, mstrSource,
LoadResString(errCloneFailed)

End Function
Private Sub CheckDupConnectionName()
   ' Check if the Connection name already
exists in the workspace

   Dim rstConnection As Recordset
   Dim strSql As String
   Dim qy As DAO.QueryDef

   On Error GoTo
CheckDupConnectionNameErr
   mstrSource = mstrModuleName &
"CheckDupConnectionName"

   ' Create a recordset object to retrieve the
count of all Connections
   ' for the workspace with the same name
   strSql = "Select count(*) as
Connection_count " & _
       " from " &
TBL_CONNECTION_DTLS & _
       " where " & FLD_ID_WORKSPACE
& " = [w_id]" & _
       " and " &
FLD_CONN_DTL_CONNECTION_NAM
E & " = [c_name]" & _
       " and " & FLD_ID_CONN_NAME & "
<> [c_id]"

   Set qy =
dbsAttTool.CreateQueryDef(gstrEmptyStrin
g, strSql)
   Call AssignParameters(qy)

   Set rstConnection =
qy.OpenRecordset(dbOpenForwardOnly)

   If rstConnection![Connection_count] > 0
Then
      rstConnection.Close
      qy.Close
      ShowError errDupConnDtlName
      On Error GoTo 0
      Err.Raise vbObjectError +
errDupConnDtlName, _
         mstrSource,
LoadResString(errDupConnDtlName)
   End If

   rstConnection.Close
   qy.Close

   Exit Sub

CheckDupConnectionNameErr:
   LogErrors Errors
   mstrSource = mstrModuleName &
"CheckDupConnectionName"
   On Error GoTo 0
   Err.Raise vbObjectError + errProgramError,
_
      mstrSource,
LoadResString(errProgramError)

End Sub
Public Property Let IndOperation(ByVal vdata
As Operation)

   ' The valid operations are define in the
cOperations
   ' class. Check if the operation is valid
   Select Case vdata
      Case QueryOp, InsertOp, UpdateOp,
DeleteOp
         mintOperation = vdata

      Case Else
         BugAssert True
   End Select

End Property
Public Sub Validate()
   ' Each distinct object will have a Validate
method which
   ' will check if the class properties are valid.
This method
   ' will be used to check interdependant
properties that
   ' cannot be validated by the let procedures.
   ' It should be called by the add and modify
methods of the class

   If ConnName = gstrEmptyString Then

      ShowError
errConnectionNameMandatory
      On Error GoTo 0
      ' Propogate this error back to the caller
      Err.Raise vbObjectError +
errConnectionNameMandatory, _
         mstrSource,
LoadResString(errConnectionNameMandatory
)
   End If

   ' Raise an error if the Connection name
already exists in the workspace
   Call CheckDupConnectionName
End Sub
Public Sub Add()

   Dim strInsert As String
   Dim qy As DAO.QueryDef

   On Error GoTo AddErr

   ' Validate the record before trying to insert the
record
   Call Validate

   ' Create a temporary querydef object
   strInsert = "insert into " &
TBL_CONNECTION_DTLS & _
         "( " & FLD_ID_WORKSPACE & _
         ", " & FLD_ID_CONN_NAME & _
         ", " &
FLD_CONN_DTL_CONNECTION_NAME & _
         ", " &
FLD_CONN_DTL_CONNECTION_STRING & _
         ", " &
FLD_CONN_DTL_CONNECTION_TYPE & " ) "
& _
         " values ( [w_id], [c_id], " & _
         " [c_name], [c_str], [c_type] ) "

   Set qy =
dbsAttTool.CreateQueryDef(gstrEmptyString,
strInsert)

   ' Call a procedure to assign the Connection values
   Call AssignParameters(qy)

   qy.Execute dbFailOnError
   qy.Close

   Exit Sub

AddErr:

   Call LogErrors(Errors)
   On Error GoTo 0
   Err.Raise vbObjectError + errInsertFailed, _
      mstrModuleName & "Add",
LoadResString(errInsertFailed)

End Sub
Public Sub Delete()

   Dim strDelete As String
   Dim qy As DAO.QueryDef

   On Error GoTo DeleteErr

   strDelete = "delete from " &
TBL_CONNECTION_DTLS & _
         " where " & FLD_ID_CONN_NAME & " =
[c_id]"
   Set qy =
dbsAttTool.CreateQueryDef(gstrEmptyString,
strDelete)

   Call AssignParameters(qy)
   qy.Execute dbFailOnError

   qy.Close

   Exit Sub

DeleteErr:
   LogErrors Errors
   On Error GoTo 0
   Err.Raise vbObjectError + errDeleteFailed, _
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
        mstrModuleName & "Delete",
LoadResString(errDeleteFailed)

End Sub

Public Sub Modify()

   Dim strUpdate As String
   Dim qy As QueryDef

   On Error GoTo ModifyErr

   ' Validate the updated values before trying
to modify the db
   Call Validate

   ' Create a temporary querydef object with
the modify string
   strUpdate = "update " &
TBL_CONNECTION_DTLS & _
        " set " & FLD_ID_WORKSPACE &
" = [w_id], " & _

FLD_CONN_DTL_CONNECTION_NAM
E & " = [c_name], " & _

FLD_CONN_DTL_CONNECTION_STRI
NG & " = [c_str], " & _

FLD_CONN_DTL_CONNECTION_TYPE
& " = [c_type] " & _
        " where " &
FLD_ID_CONN_NAME & " = [c_id]"
   Set qy =
dbsAttTool.CreateQueryDef(gstrEmptyStrin
g, strUpdate)

   ' Call a procedure to assign the
Connection values to the
   ' querydef object
   Call AssignParameters(qy)
   qy.Execute dbFailOnError

   qy.Close

   Exit Sub

ModifyErr:

   Call LogErrors(Errors)
   On Error GoTo 0
   Err.Raise vbObjectError +
errModifyFailed, _
      mstrModuleName & "Modify",
LoadResString(errModifyFailed)

End Sub
Public Property Get NextIdentifier() As
Long

   Dim lngNextId As Long

   On Error GoTo NextIdentifierErr

   ' Retrieve the next identifier using the
sequence class
   Set mConnectionSeq = New cSequence
   Set mConnectionSeq.IdDatabase =
dbsAttTool
   mConnectionSeq.IdentifierColumn =
FLD_ID_CONN_NAME
   lngNextId = mConnectionSeq.Identifier
   Set mConnectionSeq = Nothing

   NextIdentifier = lngNextId
   Exit Property

NextIdentifierErr:
   LogErrors Errors
   On Error GoTo 0
   Err.Raise vbObjectError + errIdGetFailed, _
      mstrModuleName & "NextIdentifier",
LoadResString(errIdGetFailed)

End Property
Public Property Get IndOperation() As
Operation

   IndOperation = mintOperation

End Property

Private Sub Class_Initialize()

   Set mFieldValue = New cStringSM

   ' Initialize the operation indicator variable to
Query
   ' It will be modified later by the collection
class when
   ' inserts, updates or deletes are performed
   mintOperation = QueryOp

   ConnType = giDefaultConnType

End Sub

Private Sub Class_Terminate()

   Set mFieldValue = Nothing

End Sub
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cConnDtls"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:      cConnDtls.cls
'            Microsoft TPC-H Kit Ver. 1.00
'            Copyright Microsoft, 1999
'            All Rights Reserved
'
'
'  PURPOSE:    Implements an array of
cConnDtl objects.
'            Type-safe wrapper around
cNodeCollections.
'            Also contains additional functions to
determine the connection
'            string value, validation functions, etc.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

Private mcarrConnDtls As cNodeCollections

' Used to indicate the source module name
when errors
' are raised by this class
Private mstrSource As String
Private Const mstrModuleName As String =
"cConnDtls."

Public Property Set ConnDb(vdata As
Database)

   Set mcarrConnDtls.NodeDB = vdata

End Property
Public Sub Modify(cModifiedConn As cConnDtl)

   ' First check if the parameter record is valid
   Call CheckDupConnName(cModifiedConn)

   Call mcarrConnDtls.Modify(cModifiedConn)

End Sub
Public Sub Load(ByRef cConnToAdd As
cConnDtl)

   Call mcarrConnDtls.Load(cConnToAdd)

End Sub
Public Sub Add(ByRef cConnToAdd As cConnDtl)

   ' First check if the record is valid
   Call Validate(cConnToAdd)

   ' Retrieve a unique identifier
   cConnToAdd.ConnNameId =
cConnToAdd.NextIdentifier

   Call mcarrConnDtls.Add(cConnToAdd)

End Sub

Public Sub Unload(lConnNameId As Long)

   Dim lngDeleteElement As Long

   lngDeleteElement = QueryIndex(lConnNameId)

   Call mcarrConnDtls.Unload(lngDeleteElement)

End Sub

Public Sub SaveConnDtlsInWsp(ByVal
lngWorkspace As Long)
   ' Call a procedure to save all connection details
records for the workspace
   Call mcarrConnDtls.Save(lngWorkspace)

End Sub
Public Function GetConnectionDtl(ByVal
lngWorkspace As Long, _
      ByVal strConnectionName As String) As
cConnDtl
   ' Returns the connection dtl for the passed in
connection name

   Dim lngIndex As Long

   ' Find all parameters in the array with a matching
workspace id
   For lngIndex = 0 To mcarrConnDtls.Count - 1
      If mcarrConnDtls(lngIndex).WorkspaceId =
lngWorkspace And _
            mcarrConnDtls(lngIndex).ConnName =
strConnectionName Then

         Set GetConnectionDtl =
mcarrConnDtls(lngIndex)
         Exit For
      End If
   Next lngIndex

   If lngIndex > mcarrConnDtls.Count - 1 Then
      ' The parameter has not been defined for the
workspace
      ' Raise an error
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
      On Error GoTo 0
      Err.Raise vbObjectError +
errConnNameInvalid, mstrModuleName &
"GetConnection", _

LoadResString(errConnNameInvalid)
   End If

End Function
Public Sub Delete(lConnNameId As Long)
   ' Delete the passed in parameter

   Dim lngDeleteElement As Long

   lngDeleteElement =
QueryIndex(lConnNameId)
   Call
mcarrConnDtls.Delete(lngDeleteElement)

End Sub
Private Function QueryIndex(lConnNameId
As Long) As Long

   Dim lngIndex As Long

   ' Find the matching parameter record in
the array
   For lngIndex = 0 To
mcarrConnDtls.Count - 1
      If
mcarrConnDtls(lngIndex).ConnNameId =
lConnNameId And _

mcarrConnDtls(lngIndex).IndOperation <>
DeleteOp Then
         QueryIndex = lngIndex
         Exit Function
      End If
   Next lngIndex

   ' Raise error that parameter has not been
found
   On Error GoTo 0
   Err.Raise vbObjectError +
errQueryIndexFailed,
"cArrParameters.QueryIndex", _
      LoadResString(errQueryIndexFailed)

End Function

Public Function
QueryConnDtl(lConnNameId As Long) As
cConnDtl

   Dim lngQueryElement As Long

   lngQueryElement =
QueryIndex(lConnNameId)

   ' Return the queried connection object
   Set QueryConnDtl =
mcarrConnDtls(lngQueryElement)

End Function
Public Property Get Count() As Long

   Count = mcarrConnDtls.Count

End Property
Public Property Get Item(lngIndex As Long)
As cConnDtl
Attribute Item.VB_UserMemId = 0

   Set Item = mcarrConnDtls(lngIndex)

End Property

Private Sub Validate(ByVal cConnToValidate
As cConnDtl)
   ' This procedure is necessary since the class
cannot validate
   ' all the connection_dtl properties on it's
own. This is 'coz we
   ' might have created new connections in the
workspace, but not
   ' saved them to the database yet - hence the
duplicate check
   ' has to be repeated in the array

   Dim lngIndex As Long
   Dim cTempParam As cConnDtl

   ' Check if the parameter name already exists
in the workspace
   For lngIndex = 0 To mcarrConnDtls.Count -
1
      Set cTempParam =
mcarrConnDtls(lngIndex)
      If cTempParam.WorkspaceId =
cConnToValidate.WorkspaceId And _
            cTempParam.ConnName =
cConnToValidate.ConnName And _
            cTempParam.IndOperation <>
DeleteOp Then
         On Error GoTo 0
         Err.Raise vbObjectError +
errDupConnDtlName, _
            mstrSource,
LoadResString(errDupConnDtlName)
      End If
   Next lngIndex

End Sub
Private Sub CheckDupConnName(ByVal
cConnToValidate As cConnDtl)

   Dim lngIndex As Long
   Dim cTempParam As cConnDtl

   ' Check if the parameter name already exists
in the workspace
   For lngIndex = 0 To mcarrConnDtls.Count -
1
      Set cTempParam =
mcarrConnDtls(lngIndex)
      If cTempParam.WorkspaceId =
cConnToValidate.WorkspaceId And _
            cTempParam.ConnName =
cConnToValidate.ConnName And _
            cTempParam.ConnNameId <>
cConnToValidate.ConnNameId And _
            cTempParam.IndOperation <>
DeleteOp Then
         ShowError errDupConnDtlName
         On Error GoTo 0
         Err.Raise vbObjectError +
errDupConnDtlName, _
            mstrSource,
LoadResString(errDupConnDtlName)
      End If
   Next lngIndex

End Sub

Private Sub Class_Initialize()

   Set mcarrConnDtls = New cNodeCollections

End Sub

Private Sub Class_Terminate()

   Set mcarrConnDtls = Nothing

End Sub
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cConnection"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:       cConnection.cls
'            Microsoft TPC-H Kit Ver. 1.00
'            Copyright Microsoft, 1999
'            All Rights Reserved
'
'
'  PURPOSE:    Encapsulates the properties and
methods of a connection string.
'            Contains functions to insert, update and
delete
'            workspace_connections records from the
database.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit
Option Base 0

' Local variable(s) to hold property value(s)
Public WorkspaceId As Long
Public ConnectionId As Long
Public ConnectionValue As String
Public Description As String
Public NodeDB As Database
Public Position As Long
Public NoCountDisplay As Boolean
Public NoExecute As Boolean
Public ParseQueryOnly As Boolean
Public QuotedIdentifiers As Boolean
Public AnsiNulls As Boolean
Public ShowQueryPlan As Boolean
Public ShowStatsTime As Boolean
Public ShowStatsIO As Boolean
Public ParseOdbcMsg As Boolean
Public RowCount As Long
Public TsqlBatchSeparator As String
Public QueryTimeOut As Long
Public ServerLanguage As String
Public CharacterTranslation As Boolean
Public RegionalSettings As Boolean

Private mstrConnectionName As String
Private mintOperation As Operation

' Used to indicate the source module name when
errors
' are raised by this class
Private mstrSource As String
Private Const mstrModuleName As String =
"cConnection."

' The cSequence class is used to generate unique
Connection identifiers
Private mConnectionSeq As cSequence

' The StringSM class is used to carry out string
operations
Private mFieldValue As cStringSM

Private Sub AssignParameters(qyExec As
DAO.QueryDef)
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
    ' Assigns values to the parameters in the
querydef object
    ' The parameter names are cryptic to
differentiate them from the field names.
    ' When the parameter names are the same
as the field names, parameters in the where
    ' clause do not get created.

    Dim prmParam As DAO.Parameter

    On Error GoTo AssignParametersErr

    For Each prmParam In
qyExec.Parameters
        Select Case prmParam.Name
            Case "[w_id]"
                prmParam.Value = WorkspaceId

            Case "[c_id]"
                prmParam.Value = ConnectionId

            Case "[c_name]"
                prmParam.Value =
mstrConnectionName

            Case "[c_value]"
                prmParam.Value =
ConnectionValue

            Case "[desc]"
                prmParam.Value = Description

            Case "[no_count]"
                prmParam.Value =
NoCountDisplay

            Case "[no_exec]"
                prmParam.Value = NoExecute

            Case "[parse_only]"
                prmParam.Value =
ParseQueryOnly

            Case "[quoted_id]"
                prmParam.Value =
QuotedIdentifiers

            Case "[a_nulls]"
                prmParam.Value = AnsiNulls

            Case "[show_qp]"
                prmParam.Value =
ShowQueryPlan

            Case "[stats_tm]"
                prmParam.Value =
ShowStatsTime

            Case "[stats_io]"
                prmParam.Value = ShowStatsIO

            Case "[parse_odbc]"
                prmParam.Value = ParseOdbcMsg

            Case "[row_cnt]"
                prmParam.Value = RowCount

            Case "[batch_sep]"
                prmParam.Value =
TsqlBatchSeparator

            Case "[qry_tmout]"
                prmParam.Value = QueryTimeOut

            Case "[lang]"
                prmParam.Value = ServerLanguage

            Case "[char_trans]"
                prmParam.Value =
CharacterTranslation

            Case "[reg_settings]"
                prmParam.Value = RegionalSettings

            Case Else
                ' Write the parameter name that is
faulty
                WriteError errInvalidParameter,
mstrSource, prmParam.Name
                On Error GoTo 0
                Err.Raise errInvalidParameter,
mstrModuleName & "AssignParameters", _

LoadResString(errInvalidParameter)
        End Select
    Next prmParam

    Exit Sub

AssignParametersErr:

    Call LogErrors(Errors)
    On Error GoTo 0
    Err.Raise vbObjectError +
errAssignParametersFailed, _
        mstrModuleName & "AssignParameters",
LoadResString(errAssignParametersFailed)

End Sub

Public Function Clone() As cConnection

    ' Creates a copy of a given Connection

    Dim cCloneConn As cConnection

    On Error GoTo CloneErr

    Set cCloneConn = New cConnection

    ' Copy all the Connection properties to the
newly
    ' created Connection
    Set cCloneConn.NodeDB = NodeDB
    cCloneConn.WorkspaceId = WorkspaceId
    cCloneConn.ConnectionId = ConnectionId
    cCloneConn.ConnectionName =
mstrConnectionName
    cCloneConn.ConnectionValue =
ConnectionValue
    cCloneConn.Description = Description
    cCloneConn.IndOperation = mintOperation
    cCloneConn.Position = Position
    cCloneConn.NoCountDisplay =
NoCountDisplay
    cCloneConn.NoExecute = NoExecute
    cCloneConn.ParseQueryOnly =
ParseQueryOnly
    cCloneConn.QuotedIdentifiers =
QuotedIdentifiers
    cCloneConn.AnsiNulls = AnsiNulls
    cCloneConn.ShowQueryPlan =
ShowQueryPlan
    cCloneConn.ShowStatsTime =
ShowStatsTime
    cCloneConn.ShowStatsIO = ShowStatsIO
    cCloneConn.ParseOdbcMsg =
ParseOdbcMsg
    cCloneConn.RowCount = RowCount
    cCloneConn.TsqlBatchSeparator =
TsqlBatchSeparator
    cCloneConn.QueryTimeOut = QueryTimeOut
    cCloneConn.ServerLanguage = ServerLanguage
    cCloneConn.CharacterTranslation =
CharacterTranslation
    cCloneConn.RegionalSettings = RegionalSettings

    ' And set the return value to the newly created
Connection
    Set Clone = cCloneConn
    Set cCloneConn = Nothing

    Exit Function

CloneErr:
    LogErrors Errors
    mstrSource = mstrModuleName & "Clone"
    On Error GoTo 0
    Err.Raise vbObjectError + errCloneFailed,
mstrSource, LoadResString(errCloneFailed)

End Function
Private Sub CheckDupConnectionName()
    ' Check if the Connection name already exists in
the workspace

    Dim rstConnection As Recordset
    Dim strSql As String
    Dim qy As DAO.QueryDef

    On Error GoTo CheckDupConnectionNameErr
    mstrSource = mstrModuleName &
"CheckDupConnectionName"

    ' Create a recordset object to retrieve the count of
all Connections
    ' for the workspace with the same name
    strSql = "Select count(*) as Connection_count "
& _
        " from workspace_connections " & _
        " where workspace_id = [w_id]" & _
        " and connection_name = [c_name]" & _
        " and connection_id <> [c_id]"

    Set qy =
NodeDB.CreateQueryDef(gstrEmptyString, strSql)
    Call AssignParameters(qy)

    Set rstConnection =
qy.OpenRecordset(dbOpenForwardOnly)

    If rstConnection![Connection_count] > 0 Then
        rstConnection.Close
        qy.Close
        ShowError errDuplicateConnectionName
        On Error GoTo 0
        Err.Raise vbObjectError +
errDuplicateConnectionName, _
            mstrSource,
LoadResString(errDuplicateConnectionName)
    End If

    rstConnection.Close
    qy.Close

    Exit Sub

CheckDupConnectionNameErr:
    LogErrors Errors
    mstrSource = mstrModuleName &
"CheckDupConnectionName"
    On Error GoTo 0
    Err.Raise vbObjectError + errProgramError, _
        mstrSource, LoadResString(errProgramError)
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
End Sub
Private Sub CheckDB()
   ' Check if the database object has been
initialized

   If NodeDB Is Nothing Then
      On Error GoTo 0
      Err.Raise vbObjectError +
errInvalidDB, _
         mstrModuleName & "CheckDB",
LoadResString(errInvalidDB)
   End If

End Sub
Public Property Let ConnectionName(vdata
As String)

   If vdata = gstrEmptyString Then

      ShowError
errConnectionNameMandatory
      On Error GoTo 0
      ' Propogate this error back to the caller
      Err.Raise vbObjectError +
errConnectionNameMandatory, _
         mstrSource,
LoadResString(errConnectionNameMandato
ry)
   Else
      mstrConnectionName = vdata
   End If

End Property

Public Property Let IndOperation(ByVal
vdata As Operation)

   ' The valid operations are define in the
cOperations
   ' class. Check if the operation is valid
   Select Case vdata
      Case QueryOp, InsertOp, UpdateOp,
DeleteOp
         mintOperation = vdata

      Case Else
         BugAssert True
   End Select

End Property
Public Sub Validate()
   ' Each distinct object will have a Validate
method which
   ' will check if the class properties are
valid. This method
   ' will be used to check interdependant
properties that
   ' cannot be validated by the let
procedures.
   ' It should be called by the add and
modify methods of the class

   ' Check if the db object is valid
   Call CheckDB

   ' Raise an error if the Connection name
already exists in the workspace
   Call CheckDupConnectionName

End Sub
Public Sub Add()

   Dim strInsert As String
   Dim qy As DAO.QueryDef

   On Error GoTo AddErr

   ' Validate the record before trying to insert
the record
   Call Validate

   ' Create a temporary querydef object
   strInsert = "insert into
workspace_connections " & _
         "( workspace_id, connection_id, " & _
         "connection_name, connection_value,
" & _
         "description, no_count_display, " & _
         "no_execute, parse_query_only, " & _
         "ANSI_quoted_identifiers,
ANSI_nulls, " & _
         "show_query_plan, show_stats_time, "
& _
         "show_stats_io,
parse_odbc_msg_prefixes, " & _
         "row_count, tsql_batch_separator, " &
_
         "query_time_out, server_language, " &
_
         "character_translation,
regional_settings ) " & _
         " values ( [w_id], [c_id], [c_name],
[c_value], " & _
         " [desc], [no_count], [no_exec],
[parse_only], " & _
         " [quoted_id], [a_nulls], [show_qp],
[stats_tm], " & _
         " [stats_io], [parse_odbc], [row_cnt],
[batch_sep], " & _
         " [qry_tmout], [lang], [char_trans],
[reg_settings] ) "

   Set qy =
NodeDB.CreateQueryDef(gstrEmptyString,
strInsert)

   ' Call a procedure to assign the Connection
values
   Call AssignParameters(qy)

   qy.Execute dbFailOnError
   qy.Close

   Exit Sub

AddErr:

   Call LogErrors(Errors)
   On Error GoTo 0
   Err.Raise vbObjectError + errInsertFailed, _
      mstrModuleName & "Add",
LoadResString(errInsertFailed)

End Sub
Public Sub Delete()

   Dim strDelete As String
   Dim qy As DAO.QueryDef

   On Error GoTo DeleteErr

   ' Check if the db object is valid
   Call CheckDB

   strDelete = "delete from
workspace_connections " & _
         " where connection_id = [c_id]"
   Set qy =
NodeDB.CreateQueryDef(gstrEmptyString,
strDelete)

   Call AssignParameters(qy)
   qy.Execute dbFailOnError

   qy.Close

   Exit Sub

DeleteErr:
   LogErrors Errors
   On Error GoTo 0
   Err.Raise vbObjectError + errDeleteFailed, _
         mstrModuleName & "Delete",
LoadResString(errDeleteFailed)

End Sub

Public Sub Modify()

   Dim strUpdate As String
   Dim qy As QueryDef

   On Error GoTo ModifyErr

   ' Validate the updated values before trying to
modify the db
   Call Validate

   ' Create a temporary querydef object with the
modify string
   strUpdate = "update workspace_connections " &
_
         " set workspace_id = [w_id], " & _
         "connection_name = [c_name], " & _
         "connection_value = [c_value], " & _
         "description = [desc], " & _
         "no_count_display = [no_count], " & _
         "no_execute = [no_exec], " & _
         "parse_query_only = [parse_only], " & _
         "ANSI_quoted_identifiers = [quoted_id], "
& _
         "ANSI_nulls = [a_nulls], " & _
         "show_query_plan = [show_qp], " & _
         "show_stats_time = [stats_tm], " & _
         "show_stats_io = [stats_io], " & _
         "parse_odbc_msg_prefixes = [parse_odbc],
" & _
         "row_count = [row_cnt], " & _
         "tsql_batch_separator = [batch_sep], " & _
         "query_time_out = [qry_tmout], " & _
         "server_language = [lang], " & _
         "character_translation = [char_trans], " & _
         "regional_settings = [reg_settings] " & _
         " where connection_id = [c_id]"
   Set qy =
NodeDB.CreateQueryDef(gstrEmptyString,
strUpdate)

   ' Call a procedure to assign the Connection values
to the
   ' querydef object
   Call AssignParameters(qy)
   qy.Execute dbFailOnError

   qy.Close

   Exit Sub

ModifyErr:

   Call LogErrors(Errors)
   On Error GoTo 0
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   Err.Raise vbObjectError +
errModifyFailed, _
       mstrModuleName & "Modify",
LoadResString(errModifyFailed)

End Sub
Public Property Get ConnectionName() As
String

   ConnectionName = mstrConnectionName

End Property

Public Property Get NextIdentifier() As
Long

   Dim lngNextId As Long

   On Error GoTo NextIdentifierErr

   ' First check if the database object is valid
   Call CheckDB

   ' Retrieve the next identifier using the
sequence class
   Set mConnectionSeq = New cSequence
   Set mConnectionSeq.IdDatabase =
NodeDB
   mConnectionSeq.IdentifierColumn =
"connection_id"
   lngNextId = mConnectionSeq.Identifier
   Set mConnectionSeq = Nothing

   NextIdentifier = lngNextId
   Exit Property

NextIdentifierErr:
   LogErrors Errors
   On Error GoTo 0
   Err.Raise vbObjectError + errIdGetFailed,
_
       mstrModuleName & "NextIdentifier",
LoadResString(errIdGetFailed)

End Property
Public Property Get IndOperation() As
Operation

   IndOperation = mintOperation

End Property

Private Sub Class_Initialize()

   Set mFieldValue = New cStringSM

   ' Initialize the operation indicator variable
to Query
   ' It will be modified later by the collection
class when
   ' inserts, updates or deletes are performed
   mintOperation = QueryOp

   ' Initialize connection properties to their
default values
   NoCountDisplay =
DEF_NO_COUNT_DISPLAY
   NoExecute = DEF_NO_EXECUTE
   ParseQueryOnly =
DEF_PARSE_QUERY_ONLY
   QuotedIdentifiers =
DEF_ANSI_QUOTED_IDENTIFIERS
    AnsiNulls = DEF_ANSI_NULLS
   ShowQueryPlan =
DEF_SHOW_QUERY_PLAN
   ShowStatsTime =
DEF_SHOW_STATS_TIME
   ShowStatsIO = DEF_SHOW_STATS_IO
   ParseOdbcMsg =
DEF_PARSE_ODBC_MSG_PREFIXES
   RowCount = DEF_ROW_COUNT
   TsqlBatchSeparator =
DEF_TSQL_BATCH_SEPARATOR
   QueryTimeOut =
DEF_QUERY_TIME_OUT
   ServerLanguage =
DEF_SERVER_LANGUAGE
   CharacterTranslation =
DEF_CHARACTER_TRANSLATION
   RegionalSettings =
DEF_REGIONAL_SETTINGS

End Sub

Private Sub Class_Terminate()

   Set NodeDB = Nothing
   Set mFieldValue = Nothing

End Sub
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cConnections"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:      cConnections.cls
'             Microsoft TPC-H Kit Ver. 1.00
'             Copyright Microsoft, 1999
'             All Rights Reserved
'
'
'  PURPOSE:    Implements an array of
cConnection objects.
'             Type-safe wrapper around
cNodeCollections.
'             Also contains validation functions,
etc.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

Private mcarrConnections As
cNodeCollections

' Used to indicate the source module name
when errors
' are raised by this class
Private mstrSource As String
Private Const mstrModuleName As String =
"cConnections."

Public Property Set ConnDb(vdata As
Database)

   Set mcarrConnections.NodeDB = vdata

End Property
Public Sub Modify(cModifiedConn As
cConnection)

   ' First check if the parameter record is valid
   Call CheckDupConnName(cModifiedConn)

   Call
mcarrConnections.Modify(cModifiedConn)

End Sub
Public Sub Load(ByRef cConnToAdd As
cConnection)

   Call mcarrConnections.Load(cConnToAdd)

End Sub
Public Sub Add(ByRef cConnToAdd As
cConnection)

   Set cConnToAdd.NodeDB =
mcarrConnections.NodeDB

   ' First check if the record is valid
   Call Validate(cConnToAdd)

   ' Retrieve a unique identifier
   cConnToAdd.ConnectionId =
cConnToAdd.NextIdentifier

   Call mcarrConnections.Add(cConnToAdd)

End Sub

Public Sub Unload(lngConnId As Long)

   Dim lngDeleteElement As Long

   lngDeleteElement = QueryIndex(lngConnId)

   Call
mcarrConnections.Unload(lngDeleteElement)

End Sub

Public Sub SaveConnectionsInWsp(ByVal
lngWorkspace As Long)
   ' Call a procedure to save all connection records
for the workspace
   Call mcarrConnections.Save(lngWorkspace)

End Sub
Public Function GetConnection(ByVal
lngWorkspace As Long, _
       ByVal strConnectionName As String) As
cConnection
   ' Returns the connection string for the passed in
connection name

   Dim lngIndex As Long

   ' Find all parameters in the array with a matching
workspace id
   For lngIndex = 0 To mcarrConnections.Count - 1
       If mcarrConnections(lngIndex).WorkspaceId =
lngWorkspace And _

mcarrConnections(lngIndex).ConnectionName =
strConnectionName Then

           Set GetConnection =
mcarrConnections(lngIndex)
           Exit For
       End If
   Next lngIndex

   If lngIndex > mcarrConnections.Count - 1 Then
       ' The parameter has not been defined for the
workspace
       ' Raise an error
       On Error GoTo 0
       Err.Raise vbObjectError +
errConnNameInvalid, mstrModuleName &
"GetConnection", _
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
LoadResString(errConnNameInvalid)
   End If

End Function
Public Sub Delete(lngConnId As Long)
   ' Delete the passed in parameter

   Dim lngDeleteElement As Long

   lngDeleteElement =
QueryIndex(lngConnId)
   Call
mcarrConnections.Delete(lngDeleteElement
)

End Sub
Private Function QueryIndex(lngConnId As
Long) As Long

   Dim lngIndex As Long

   ' Find the matching parameter record in
the array
   For lngIndex = 0 To
mcarrConnections.Count - 1
      If
mcarrConnections(lngIndex).ConnectionId
= lngConnId And _

mcarrConnections(lngIndex).IndOperation
<> DeleteOp Then
         QueryIndex = lngIndex
         Exit Function
      End If
   Next lngIndex

   ' Raise error that parameter has not been
found
   On Error GoTo 0
   Err.Raise vbObjectError +
errQueryIndexFailed,
"cArrParameters.QueryIndex", _
      LoadResString(errQueryIndexFailed)

End Function

Public Function
QueryConnection(lngConnId As Long) As
cConnection

   Dim lngQueryElement As Long

   lngQueryElement =
QueryIndex(lngConnId)

   ' Return the queried connection object
   Set QueryConnection =
mcarrConnections(lngQueryElement)

End Function
Public Property Get Count() As Long

   Count = mcarrConnections.Count

End Property
Public Property Get Item(lngIndex As Long)
As cConnection
Attribute Item.VB_UserMemId = 0

   Set Item = mcarrConnections(lngIndex)

End Property
Public Sub Validate(ByVal cConnToValidate
As cConnection)
   ' This procedure is necessary since the class
cannot validate
   ' all the parameter properties on it's own.
This is 'coz we
   ' might have created new parameters in the
workspace, but not
   ' saved them to the database yet - hence the
duplicate check
   ' has to be repeated in the array

   Dim lngIndex As Long
   Dim cTempParam As cConnection

   ' Check if the parameter name already exists
in the workspace
   For lngIndex = 0 To
mcarrConnections.Count - 1
      Set cTempParam =
mcarrConnections(lngIndex)
      If cTempParam.WorkspaceId =
cConnToValidate.WorkspaceId And _
         cTempParam.ConnectionName =
cConnToValidate.ConnectionName And _
         cTempParam.IndOperation <>
DeleteOp Then
         On Error GoTo 0
         Err.Raise vbObjectError +
errDuplicateConnectionName, _
         mstrSource,
LoadResString(errDuplicateConnectionName)
      End If
   Next lngIndex

End Sub
Public Sub CheckDupConnName(ByVal
cConnToValidate As cConnection)

   Dim lngIndex As Long
   Dim cTempParam As cConnection

   ' Check if the parameter name already exists
in the workspace
   For lngIndex = 0 To
mcarrConnections.Count - 1
      Set cTempParam =
mcarrConnections(lngIndex)
      If cTempParam.WorkspaceId =
cConnToValidate.WorkspaceId And _
         cTempParam.ConnectionName =
cConnToValidate.ConnectionName And _
         cTempParam.ConnectionId <>
cConnToValidate.ConnectionId And _
         cTempParam.IndOperation <>
DeleteOp Then
         ShowError
errDuplicateConnectionName
         On Error GoTo 0
         Err.Raise vbObjectError +
errDuplicateConnectionName, _
         mstrSource,
LoadResString(errDuplicateConnectionName)
      End If
   Next lngIndex

End Sub

Private Sub Class_Initialize()

   Set mcarrConnections = New
cNodeCollections

End Sub
Private Sub Class_Terminate()

   Set mcarrConnections = Nothing

End Sub
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cConstraint"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:       cConstraint.cls
'            Microsoft TPC-H Kit Ver. 1.00
'            Copyright Microsoft, 1999
'            All Rights Reserved
'
'
'  PURPOSE:    Encapsulates the properties and
methods of a constraint.
'            Contains functions to insert, update and
delete
'            step_constraints records from the
database.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

' Module level variables to store the property values
Private mlngConstraintId As Long
Private mlngStepId As Long
Private mstrVersionNo As String
Private mintConstraintType As Integer
Private mlngGlobalStepId As Long
Private mstrGlobalVersionNo As String
Private mintSequenceNo As Integer
Private mdbsConstraintDB As Database
Private mlngWorkspaceId As Integer
Private mintOperation As Operation
Private mlngPosition As Long

' The cSequence class is used to generate unique
step identifiers
Private mConstraintSeq As cSequence

Private Const mstrModuleName As String =
".cConstraint."
Private mstrSource As String

Public Enum ConstraintType
   gintPreStep = 1
   gintPostStep = 2
End Enum

Private Const mstrSQ As String = """
Public Property Get WorkspaceId() As Long
   WorkspaceId = mlngWorkspaceId
End Property
Public Property Let WorkspaceId(ByVal vdata As
Long)
   mlngWorkspaceId = vdata
End Property

Public Property Get IndOperation() As Operation

   IndOperation = mintOperation

End Property
Public Property Let IndOperation(ByVal vdata As
Operation)

   On Error GoTo IndOperationErr
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   mstrSource = mstrModuleName &
"IndOperation"

   ' The valid operations are define in the
cOperations
   ' class. Check if the operation is valid
   Select Case vdata
      Case QueryOp, InsertOp, UpdateOp,
DeleteOp
         mintOperation = vdata

      Case Else
         On Error GoTo 0
         Err.Raise vbObjectError +
errInvalidOperation, _
               mstrSource,
LoadResString(errInvalidOperation)
   End Select

   Exit Property

IndOperationErr:
   LogErrors Errors
   mstrSource = mstrModuleName &
"IndOperation"
   On Error GoTo 0
   Err.Raise vbObjectError +
errLetOperationFailed, _
      mstrSource,
LoadResString(errLetOperationFailed)

End Property

Public Function Clone() As cConstraint

   ' Creates a copy of a given constraint

   Dim cConsClone As cConstraint

   On Error GoTo CloneErr
   mstrSource = mstrModuleName &
"Clone"

   Set cConsClone = New cConstraint

   ' Copy all the workspace properties to the
newly
   ' created workspace
   cConsClone.ConstraintId =
mlngConstraintId
   cConsClone.StepId = mlngStepId
   cConsClone.VersionNo = mstrVersionNo
   cConsClone.ConstraintType =
mintConstraintType
   cConsClone.GlobalStepId =
mlngGlobalStepId
   cConsClone.GlobalVersionNo =
mstrGlobalVersionNo
   cConsClone.SequenceNo =
mintSequenceNo
   cConsClone.WorkspaceId =
mlngWorkspaceId
   cConsClone.IndOperation =
mintOperation

   ' And set the return value to the newly
created constraint
   Set Clone = cConsClone

   Exit Function

CloneErr:
   LogErrors Errors
   mstrSource = mstrModuleName &
"Clone"
   On Error GoTo 0
   Err.Raise vbObjectError + errCloneFailed, _
         mstrSource,
LoadResString(errCloneFailed)

End Function

Public Property Get SequenceNo() As Integer

   SequenceNo = mintSequenceNo

End Property

Public Property Let SequenceNo(ByVal vdata
As Integer)
   mintSequenceNo = vdata
End Property

Public Sub Add()
   ' Inserts a new step constraint into the
database

   Dim strInsert As String
   Dim qy As DAO.QueryDef

   On Error GoTo AddErr

   ' First check if the database object is valid
   Call CheckDB

   ' Any record validations
   Call Validate

   ' Create a temporary querydef object
   strInsert = "insert into step_constraints " & _
         "( constraint_id, step_id, version_no, "
& _
         " constraint_type, global_step_id,
global_version_no, sequence_no ) " & _
         " values ( [cons_id], [s_id], [ver_no], "
& _
         " [cons_type], [g_step_id], [g_ver_no],
" & _
         " [seq_no] )"
   Set qy =
mdbsConstraintDB.CreateQueryDef(gstrEmpt
yString, strInsert)

   ' Call a procedure to execute the Querydef
object
   Call AssignParameters(qy)

   qy.Execute dbFailOnError
   qy.Close

'   strInsert = "insert into step_constraints " &
_
'      "( constraint_id, step_id, version_no, " &
_
'      " constraint_type, global_step_id,
global_version_no, sequence_no ) " & _
'      " values ( " & _
'      Str(mlngConstraintId) & ", " &
Str(mlngStepId) & ", " & _
'      mstrSQ & mstrVersionNo & mstrSQ & ",
" & Str(mintConstraintType) & ", " & _
'      Str(mlngGlobalStepId) & ", " & mstrSQ
& mstrGlobalVersionNo & mstrSQ & ", " & _
'      Str(mintSequenceNo) & " ) "
'
'   BugMessage strInsert
'   mdbsConstraintDB.Execute strInsert,
dbFailOnError
   Exit Sub

AddErr:
   LogErrors Errors
   mstrSource = mstrModuleName & "Add"
   On Error GoTo 0
   Err.Raise vbObjectError +
errAddConstraintFailed, _
         mstrSource, _
         LoadResString(errAddConstraintFailed)
End Sub
Private Sub AssignParameters(qyExec As
DAO.QueryDef)
   ' Assigns values to the parameters in the querydef
object
   ' The parameter names are cryptic to make them
different
   ' from the field names. When the parameter
names are
   ' the same as the field names, parameters in the
where
   ' clause do not get created.

   Dim prmParam As DAO.Parameter

   On Error GoTo AssignParametersErr
   mstrSource = mstrModuleName &
"AssignParameters"

   For Each prmParam In qyExec.Parameters
      Select Case prmParam.Name
         Case "[cons_id]"
            prmParam.Value = mlngConstraintId

         Case "[s_id]"
            prmParam.Value = mlngStepId

         Case "[ver_no]"
            prmParam.Value = mstrVersionNo

         Case "[cons_type]"
            prmParam.Value = mintConstraintType

         Case "[g_step_id]"
            prmParam.Value = mlngGlobalStepId

         Case "[g_ver_no]"
            prmParam.Value = mstrGlobalVersionNo

         Case "[seq_no]"
            prmParam.Value = mintSequenceNo

         Case Else
            ' Write the parameter name that is faulty
            WriteError errInvalidParameter,
mstrSource, _
                  prmParam.Name
            On Error GoTo 0
            Err.Raise errInvalidParameter,
mstrSource, _
                  LoadResString(errInvalidParameter)
      End Select
   Next prmParam

   Exit Sub

AssignParametersErr:

   mstrSource = mstrModuleName &
"AssignParameters"
   Call LogErrors(Errors)
   On Error GoTo 0
   Err.Raise vbObjectError +
errAssignParametersFailed, _
      mstrSource,
LoadResString(errAssignParametersFailed)
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
End Sub
Public Property Get NextIdentifier() As
Long

   Dim lngNextId As Long

   On Error GoTo NextIdentifierErr

   ' First check if the database object is valid
   Call CheckDB

   ' Retrieve the next constraint identifier
using the
   ' sequence class
   Set mConstraintSeq = New cSequence
   Set mConstraintSeq.IdDatabase =
mdbsConstraintDB
   mConstraintSeq.IdentifierColumn =
"constraint_id"
   lngNextId = mConstraintSeq.Identifier
   Set mConstraintSeq = Nothing

   NextIdentifier = lngNextId
   Exit Property

NextIdentifierErr:
   LogErrors Errors
   mstrSource = mstrModuleName &
"NextIdentifier"
   On Error GoTo 0
   Err.Raise vbObjectError +
errStepIdGetFailed, _
       mstrSource,
LoadResString(errStepIdGetFailed)

End Property

Private Sub CheckDB()
   ' Check if the database object has been
initialized

   If mdbsConstraintDB Is Nothing Then
       ShowError errInvalidDB
       On Error GoTo 0
       Err.Raise vbObjectError +
errInvalidDB, _
           mstrModuleName,
LoadResString(errInvalidDB)
   End If

End Sub

Public Sub Delete()
   ' Deletes the step constraint record from
the database

   Dim strDelete As String
   Dim qy As DAO.QueryDef

   On Error GoTo DeleteErr
   mstrSource = mstrModuleName &
"Delete"

   ' There can be multiple constraints for a
step,
   ' meaning that there can be multiple
constraint records
   ' with the same constraint_id. Only a
combination
   ' of the step_id, version and constraint_id
will be
   ' unique
   strDelete = "delete from step_constraints
" & _
           " where constraint_id = [cons_id]" & _
           " and step_id = [s_id] " & _
           " and version_no = [ver_no] "
   Set qy =
mdbsConstraintDB.CreateQueryDef(gstrEmpt
yString, strDelete)

   Call AssignParameters(qy)
   qy.Execute dbFailOnError

   qy.Close

'   strDelete = "Delete from step_constraints "
& _
'           " where constraint_id = " &
Str(mlngConstraintId) & _
'           " and step_id = " & Str(mlngStepId) &
_
'           " and version_no = " & mstrSQ &
mstrVersionNo & mstrSQ
'
'   'BugMessage strDelete
'   mdbsConstraintDB.Execute strDelete,
dbFailOnError

   Exit Sub

DeleteErr:
   LogErrors Errors
   mstrSource = mstrModuleName & "Delete"
   On Error GoTo 0
   Err.Raise vbObjectError +
errDeleteConstraintFailed, _
           mstrSource, _

LoadResString(errDeleteConstraintFailed)
End Sub
Public Sub Modify()
   ' Updates the sequence no of the step
constraint record
   ' in the database

   Dim strUpdate As String
   Dim qy As QueryDef

   On Error GoTo Modify

   ' First check if the database object is valid
   Call CheckDB

   ' Any record validations
   Call Validate

   ' There can be multiple constraints for a step,
   ' meaning that there can be multiple
constraint records
   ' with the same constraint_id. Only a
combination
   ' of the step_id, version and constraint_id
will be
   ' unique
   ' Create a temporary querydef object with
the modify string
   strUpdate = "Update step_constraints " & _
           " set sequence_no = [seq_no] " & _
           " where constraint_id = [cons_id] " & _
           " and step_id = [s_id] " & _
           " and version_no = [ver_no] "
   Set qy =
mdbsConstraintDB.CreateQueryDef(gstrEmpt
yString, strUpdate)

   ' Call a procedure to assign the parameter
values to the
   ' querydef object
   Call AssignParameters(qy)
   qy.Execute dbFailOnError

   qy.Close

'   strUpdate = "Update step_constraints " & _
'           " set sequence_no = " &
Str(mintSequenceNo) & _
'           " where constraint_id = " &
Str(mlngConstraintId) & _
'           " and step_id = " & Str(mlngStepId) & _
'           " and version_no = " & mstrSQ &
mstrVersionNo & mstrSQ
'
'   'BugMessage strUpdate
'   mdbsConstraintDB.Execute strUpdate,
dbFailOnError
   Exit Sub

Modify:
   LogErrors Errors
   mstrSource = mstrModuleName & "Modify"
   On Error GoTo 0
   Err.Raise vbObjectError +
errUpdateConstraintFailed, _
           mstrSource, _
           LoadResString(errUpdateConstraintFailed)
End Sub
Public Property Get Position() As Long

   Position = mlngPosition

End Property
Public Property Let Position(ByVal RHS As Long)

   mlngPosition = RHS

End Property

Public Sub Validate()
   ' Each distinct object will have a Validate method
which
   ' will check if the class properties are valid. This
method
   ' will be used to check interdependant properties
that
   ' cannot be validated by the let procedures.
   ' It should be called by the add and modify
methods of the class

   ' No validations are necessary for the constraint
object

End Sub

Public Property Set NodeDB(vdata As Database)

   Set mdbsConstraintDB = vdata

End Property

Public Property Get NodeDB() As Database

   Set NodeDB = mdbsConstraintDB

End Property

Public Property Get GlobalVersionNo() As String

   GlobalVersionNo = mstrGlobalVersionNo

End Property

Public Property Let GlobalVersionNo(ByVal vdata
As String)
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   mstrGlobalVersionNo = vdata

End Property

Public Property Get GlobalStepId() As Long

   GlobalStepId = mlngGlobalStepId

End Property

Public Property Get ConstraintId() As Long

   ConstraintId = mlngConstraintId

End Property

Public Property Get VersionNo() As String

    VersionNo = mstrVersionNo

End Property

Public Property Get StepId() As Long

   StepId = mlngStepId

End Property

Public Property Let VersionNo(ByVal vdata
As String)

   mstrVersionNo = vdata

End Property

Public Property Let StepId(ByVal vdata As
Long)

   mlngStepId = vdata

End Property

Public Property Let ConstraintId(ByVal
vdata As Long)
   On Error GoTo ConstraintIdErr
   mstrSource = mstrModuleName &
"ConstraintId"

   If (vdata > 0) Then
      mlngConstraintId = vdata
   Else
      ' Propogate this error back to the caller
      On Error GoTo 0
      Err.Raise vbObjectError +
errConstraintIdInvalid, _
         mstrSource,
LoadResString(errConstraintIdInvalid)
   End If

   Exit Property

ConstraintIdErr:
   LogErrors Errors
   mstrSource = mstrModuleName &
"ConstraintId"
   On Error GoTo 0
   Err.Raise vbObjectError +
errConstraintIdSetFailed, _
      mstrSource,
LoadResString(errConstraintIdSetFailed)

End Property

Public Property Let GlobalStepId(ByVal vdata
As Long)

   On Error GoTo GlobalStepIdErr
   mstrSource = mstrModuleName &
"GlobalStepId"

   If (vdata > 0) Then
      mlngGlobalStepId = vdata
   Else
      ' Propogate this error back to the caller
      On Error GoTo 0
      Err.Raise vbObjectError +
errGlobalStepIdInvalid, _
         mstrSource,
LoadResString(errGlobalStepIdInvalid)
   End If

   Exit Property

GlobalStepIdErr:
   LogErrors Errors
   mstrSource = mstrModuleName &
"GlobalStepId"
   On Error GoTo 0
   Err.Raise vbObjectError +
errGlobalStepIdSetFailed, _
      mstrSource,
LoadResString(errGlobalStepIdSetFailed)

End Property

Public Property Let ConstraintType(ByVal
vdata As ConstraintType)

   On Error GoTo ConstraintTypeErr

   ' A global step can be either a pre- or a post-
execution step.
   ' These constants have been defined in the
enumeration,
   ' ConstraintType, which is exposed
   Select Case vdata
      Case gintPreStep, gintPostStep
         mintConstraintType = vdata

      Case Else
         On Error GoTo 0
         Err.Raise vbObjectError +
errConstraintTypeInvalid, _
            mstrSource,
LoadResString(errConstraintTypeInvalid)
   End Select

   Exit Property

ConstraintTypeErr:
   LogErrors Errors
   mstrSource = mstrModuleName &
"ConstraintType"
   On Error GoTo 0
   Err.Raise vbObjectError +
errConstraintTypeLetFailed, _
      mstrSource,
LoadResString(errConstraintTypeLetFailed)

End Property

Public Property Get ConstraintType() As
ConstraintType

   ConstraintType = mintConstraintType

End Property

Private Sub Class_Initialize()

   ' Initialize the operation indicator variable to
Query
   ' It will be modified later by the collection class
when
   ' inserts, updates or deletes are performed
   mintOperation = QueryOp

End Sub


VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cFailedStep"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:       cFailedStep.cls
'             Microsoft TPC-H Kit Ver. 1.00
'             Copyright Microsoft, 1999
'             All Rights Reserved
'
'
'  PURPOSE:    Properties of a step execution
failure.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

Public InstanceId As Long
Public StepId As Long
Public ParentStepId As Long
Public ContCriteria As ContinuationCriteria
Public EndTime As Currency
Public AskResponse As Long
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cFailedSteps"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:       cFailedSteps.cls
'             Microsoft TPC-H Kit Ver. 1.00
'             Copyright Microsoft, 1999
'             All Rights Reserved
'
'
'  PURPOSE:    This module encapsulates a
collection of failed steps. It
'             also determines whether sub-steps of a
passed in step need
'             to be skipped due to a failure.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

Private mcFailedSteps As cVector
Public Function ExecuteSubStep(lParentStepId As
Long) As Boolean
   ' Returns False if there is any condition that
prevents sub-steps of the passed
   ' in instance from being executed
   Dim lIndex As Long

   ExecuteSubStep = True
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
    For lIndex = 0 To Count() - 1
        If mcFailedSteps(lIndex).ContCriteria
= gintOnFailureCompleteSiblings And _
            lParentStepId <>
mcFailedSteps(lIndex).ParentStepId Then
            ExecuteSubStep = False
            Exit For
        End If

        If mcFailedSteps(lIndex).ContCriteria
= gintOnFailureAbortSiblings And _
            lParentStepId =
mcFailedSteps(lIndex).ParentStepId Then
            ExecuteSubStep = False
            Exit For
        End If

        If mcFailedSteps(lIndex).ContCriteria
= gintOnFailureSkipSiblings And _
            lParentStepId =
mcFailedSteps(lIndex).ParentStepId Then
            ExecuteSubStep = False
            Exit For
        End If

        If mcFailedSteps(lIndex).ContCriteria
= gintOnFailureAbort Then
            ExecuteSubStep = False
            Exit For
        End If

    Next lIndex

End Function
Public Sub Add(ByVal objItem As
cFailedStep)

    mcFailedSteps.Add objItem

End Sub
Public Function Delete(ByVal lPosition As
Long) As cFailedStep

    Set Delete =
mcFailedSteps.Delete(lPosition)

End Function

Public Sub Clear()

    mcFailedSteps.Clear

End Sub
Public Function Count() As Long

    Count = mcFailedSteps.Count

End Function
Public Property Get Item(ByVal Position As
Long) As cFailedStep
Attribute Item.VB_UserMemId = 0

    Set Item = mcFailedSteps.Item(Position)

End Property

Public Function StepFailed(lStepId As
Long) As Boolean

    ' Returns True if a failure record already
exists for the passed in step
    Dim lIndex As Long

    StepFailed = False

    For lIndex = 0 To Count() - 1
        If mcFailedSteps(lIndex).StepId = lStepId
Then
            StepFailed = True
            Exit For
        End If
    Next lIndex

End Function

Private Sub Class_Initialize()

    Set mcFailedSteps = New cVector

End Sub

Private Sub Class_Terminate()

    Set mcFailedSteps = Nothing

End Sub
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cFileInfo"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:        cFileInfo.cls
'              Microsoft TPC-H Kit Ver. 1.00
'              Copyright Microsoft, 1999
'              All Rights Reserved
'
'
'  PURPOSE:    File Properties viz. name,
handle, etc.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

Private mstrFileName As String
Private mintFileHandle As Integer
Private mdbsNodeDb As Database ' Since it is
used to form a cNodeCollection
Private mlngPosition As Long ' Since it is used
to form a cNodeCollection
Public Property Get FileName() As String

    FileName = mstrFileName

End Property
Public Property Let FileName(ByVal vdata As
String)

    mstrFileName = vdata

End Property
Public Property Let FileHandle(ByVal vdata
As Integer)

    mintFileHandle = vdata

End Property
Public Property Set NodeDB(vdata As
Database)

    Set mdbsNodeDb = vdata

End Property

Public Property Get NodeDB() As Database

    Set NodeDB = mdbsNodeDb

End Property
Public Property Get Position() As Long

    Position = mlngPosition

End Property
Public Property Let Position(ByVal vdata As Long)

    mlngPosition = vdata

End Property

Public Property Get FileHandle() As Integer

    FileHandle = mintFileHandle

End Property

VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cFileSM"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
Attribute VB_Ext_KEY =
"SavedWithClassBuilder" ,"Yes"
Attribute VB_Ext_KEY = "Top_Level" ,"Yes"
'  FILE:        cFileSM.cls
'              Microsoft TPC-H Kit Ver. 1.00
'              Copyright Microsoft, 1999
'              All Rights Reserved
'
'
'  PURPOSE:    Encapsulates functions to open a
file and write to it.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

' Used to indicate the source module name when
errors
' are raised by this class
Private Const mstrModuleName As String =
"cFileSM."
Private mstrSource As String

Private mstrFileName As String
Private mintHFile As Integer
Private mstrFileHeader As String
Private mstrProjectName As String

Public Sub CloseFile()

    ' Close the file
    If mintHFile > 0 Then
        Call CloseFileSM(mstrFileName)
        mintHFile = 0
    End If

End Sub

Public Property Let ProjectName(ByVal vdata As
String)
    ' An optional field - will be appended to the file
    ' header string if specified

    Const strProjectHdr As String = "Project Name:"

    mstrProjectName = vdata
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   mstrFileHeader = mstrFileHeader & _
         Space$(1) & strProjectHdr &
Space$(1) & _
         gstrSQ & vdata & gstrSQ

End Property
Public Property Get ProjectName() As
String

   ProjectName = mstrProjectName

End Property
Public Property Get FileName() As String

   FileName = mstrFileName

End Property
Public Property Let FileName(ByVal vdata
As String)

   mstrFileName = vdata

End Property
Public Sub WriteLine(strMsg As String)

   ' Writes the passed in string to the file
   Call WriteToFile(strMsg, False)

End Sub

Public Sub WriteField(strMsg As String)

   ' Writes the passed in string to the file
   Call WriteToFile(strMsg, True)

End Sub

Private Sub WriteToFile(strMsg As String, _
      blnContinue As Boolean)
   ' Writes the passed in string to the file -
the
   ' Continue flag indicates whether the next
line will
   ' be continued on the same line or printed
on a new one

   On Error GoTo WriteToFileErr

   ' Open the file if it hasn't been already
   If mintHFile = 0 Then

      ' If the filename has not been
initialized, do not
      ' attempt to open it
      If mstrFileName <> gstrEmptyString
Then

         mintHFile =
OpenFileSM(mstrFileName)

         If mintHFile = 0 Then
            ' The Open File command failed
for some reason
            ' No point in trying to write the
file header
         Else
            ' Print a file header, if a header
string has been
            ' initialized
            If mstrFileHeader <>
gstrEmptyString Then
               Print #mintHFile,
               Print #mintHFile,
mstrFileHeader
               Print #mintHFile,
            End If
         End If
      End If
   End If

   If mintHFile <> 0 Then
      If strMsg = gstrEmptyString Then
         Print #mintHFile,
      Else
         If blnContinue Then
            ' Write the message to the file -
continue
            ' all subsequent characters on the
same line
            Print #mintHFile, strMsg;
         Else
            ' Write the message to the file
            Print #mintHFile, strMsg
         End If
      End If
   Else
      ' Display the string to the user instead of
      ' trying to write it to the file
      ' This could be the project error log that
we were
      ' trying to open! Play it safe and display
errors - do
      ' not try to log them.
      MsgBox strMsg, vbOKOnly
   End If

   Exit Sub

WriteToFileErr:
   ' Log the error code raised by Visual Basic
   Call DisplayErrors(Errors)

   ' Display the string to the user instead of
   ' trying to write it to the file
   MsgBox strMsg, vbOKOnly

End Sub
Public Property Let FileHeader(ByVal vdata
As String)

   mstrFileHeader = vdata

End Property
Public Property Get FileHeader() As String

   FileHeader = mstrFileHeader

End Property

Private Sub Class_Terminate()

   ' Close the file opened by this instance
   Call CloseFile

End Sub


VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cGlobalStep"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:       cGlobalStep.cls
'             Microsoft TPC-H Kit Ver. 1.00
'             Copyright Microsoft, 1999
'             All Rights Reserved
'
'
'  PURPOSE:     Encapsulates the properties and
methods of a global step.
'             Implements the cStep class - carries out
initializations
'             and validations that are specific to global
steps.
'  Contact:     Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

Implements cStep

' Object variable to keep the reference in
Private mcStep As cStep

' Used to indicate the source module name when
errors
' are raised by this class
Private mstrSource As String
Private Const mstrModuleName As String =
"cGlobalStep."

Private Sub cStep_AddAllIterators()

   Call mcStep.AddAllIterators

End Sub

Private Sub cStep_AddIterator(cItRecord As
cIterator)

   Call mcStep.AddIterator(cItRecord)

End Sub

Private Property Let cStep_ArchivedFlag(ByVal
RHS As Boolean)

   mcStep.ArchivedFlag = RHS

End Property

Private Property Get cStep_ArchivedFlag() As
Boolean

   cStep_ArchivedFlag = mcStep.ArchivedFlag

End Property

Private Sub Class_Initialize()

   ' Create the object
   Set mcStep = New cStep

   ' Initialize the object with valid values for a
global step
   ' The global flag should be the first field to be
initialized
   ' since subsequent validations might try to check
if the
   ' step being created is global
   mcStep.GlobalFlag = True
   mcStep.StepType = gintGlobalStep

   ' A global step cannot have any sub-steps
associated with it
   ' Hence, it will always be at Step Level 0
   mcStep.ParentStepId = 0
   mcStep.ParentVersionNo = gstrMinVersion
   mcStep.StepLevel = 0
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
    ' The enabled flag must be False for all
global steps
    ' Global steps can be of two types
    ' a. Those that are run globally within a
workspace either
    '    before every step, after every step or
during the entire
    '    run, depending on the global run
method
    ' b. Those that are not run globally, but
qualify to be either
    '    pre or post-execution steps for other
steps in the workspace.
    '    Whether or not such a step will be
executed depends on
    '    whether the step for which it is defined
as a pre/post
    '    step will be executed
    mcStep.EnabledFlag = False

    mcStep.ContinuationCriteria =
gintNoOption
    mcStep.DegreeParallelism =
gstrGlobalParallelism

End Sub
Private Sub Class_Terminate()

    ' Remove the step object
    Set mcStep = Nothing

End Sub

Private Sub cStep_Add()

    ' Call a private procedure to see if the step
text has been
    ' entered - since a global step actually
executes a step, entry
    ' of the text is mandatory
    Call StepTextOrFileEntered

    ' Call the Add method of the step class to
carry out the insert
    mcStep.Add

End Sub

Private Function cStep_Clone(Optional
cCloneStep As cStep) As cStep

    Dim cNewGlobal As cGlobalStep

    Set cNewGlobal = New cGlobalStep
    Set cStep_Clone =
mcStep.Clone(cNewGlobal)

End Function


Private Property Get
cStep_ContinuationCriteria() As
ContinuationCriteria

    cStep_ContinuationCriteria =
mcStep.ContinuationCriteria

End Property

Private Property Let
cStep_ContinuationCriteria(ByVal RHS As
ContinuationCriteria)

    ' The continuation criteria field will
always be empty for a
    ' global step
    mcStep.ContinuationCriteria = 0

End Property

Private Property Let
cStep_DegreeParallelism(ByVal RHS As
String)

    ' Will always be zero for a global step
    mcStep.DegreeParallelism =
gstrGlobalParallelism

End Property

Private Property Get
cStep_DegreeParallelism() As String

    cStep_DegreeParallelism =
mcStep.DegreeParallelism

End Property

Private Sub cStep_DeleteIterator(cItRecord As
cIterator)

    Call mcStep.DeleteIterator(cItRecord)

End Sub

Private Sub cStep_Delete()

    mcStep.Delete

End Sub

Private Property Get cStep_EnabledFlag() As
Boolean

    cStep_EnabledFlag = mcStep.EnabledFlag

End Property

Private Property Let
cStep_EnabledFlag(ByVal RHS As Boolean)

    ' The enabled flag must be False for all
global steps
    ' Global steps can be of two types
    ' a. Those that are run globally within a
workspace either
    '    before every step, after every step or
during the entire
    '    run, depending on the global run method
    ' b. Those that are not run globally, but
qualify to be either
    '    pre or post-execution steps for other steps
in the workspace.
    '    Whether or not such a step will be
executed depends on
    '    whether the step for which it is defined as
a pre/post
    '    step will be executed
    mcStep.EnabledFlag = False

End Property

Private Property Let cStep_ErrorFile(ByVal
RHS As String)

    mcStep.ErrorFile = RHS

End Property

Private Property Get cStep_ErrorFile() As String

    cStep_ErrorFile = mcStep.ErrorFile

End Property

Private Property Let
cStep_ExecutionMechanism(ByVal RHS As
ExecutionMethod)

    ' Whether or not the Execution Mechanism is
valid will be
    ' checked by the Step class
    mcStep.ExecutionMechanism = RHS

End Property

Private Property Get cStep_ExecutionMechanism()
As ExecutionMethod

    cStep_ExecutionMechanism =
mcStep.ExecutionMechanism

End Property

Private Property Let cStep_FailureDetails(ByVal
RHS As String)

    ' Whether or not the Failure Details are valid for
the
    ' selected failure criteria will be checked by the
Step class
    mcStep.FailureDetails = RHS

End Property

Private Property Get cStep_FailureDetails() As
String

    cStep_FailureDetails = mcStep.FailureDetails

End Property

Private Property Get cStep_GlobalFlag() As
Boolean

    cStep_GlobalFlag = mcStep.GlobalFlag

End Property

Private Property Let cStep_GlobalFlag(ByVal RHS
As Boolean)

    ' Set the global flag to true
    mcStep.GlobalFlag = True

End Property

Private Function cStep_IncVersionX() As String

    cStep_IncVersionX = mcStep.IncVersionX

End Function

Private Function cStep_IncVersionY() As String

    cStep_IncVersionY = mcStep.IncVersionY

End Function


'Private Property Let
cStep_GlobalRunMethod(ByVal RHS As Integer)
'
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
'    ' Whether or not the Global Run Method
is valid for the step
'    ' will be checked by the Step class
'    mcStep.GlobalRunMethod = RHS
'
'End Property
'
'Private Property Get
cStep_GlobalRunMethod() As Integer
'
'    cStep_GlobalRunMethod =
mcStep.GlobalRunMethod
'
'End Property
'
Private Property Get cStep_IndOperation()
As Operation

    cStep_IndOperation =
mcStep.IndOperation

End Property

Private Property Let
cStep_IndOperation(ByVal RHS As
Operation)

    mcStep.IndOperation = RHS

End Property

Private Sub cStep_InsertIterator(cItRecord
As cIterator)

    Call mcStep.InsertIterator(cItRecord)

End Sub

Private Function cStep_IsNewVersion() As
Boolean
    cStep_IsNewVersion =
mcStep.IsNewVersion
End Function

Private Function cStep_IteratorCount() As
Long

    cStep_IteratorCount =
mcStep.IteratorCount

End Function

Private Property Let
cStep_IteratorName(ByVal RHS As String)

    mcStep.IteratorName = RHS

End Property

Private Property Get cStep_IteratorName()
As String

    cStep_IteratorName =
mcStep.IteratorName

End Property

Private Function cStep_Iterators() As
Variant

    cStep_Iterators = mcStep.Iterators

End Function

Private Sub cStep_LoadIterator(cItRecord As
cIterator)

    Call mcStep.LoadIterator(cItRecord)

End Sub

'Private Property Let cStep_LogFile(ByVal
RHS As String)
'
'    mcStep.LogFile = RHS
'
'End Property
'
'Private Property Get cStep_LogFile() As
String
'
'    cStep_LogFile = mcStep.LogFile
'
'End Property

Private Sub cStep_ModifyIterator(cItRecord
As cIterator)

    Call mcStep.ModifyIterator(cItRecord)

End Sub

Private Sub cStep_Modify()

    ' Call a private procedure to see if the step
text has been
    ' entered - since a global step actually
executes a step,
    ' entry of the text is mandatory
    Call StepTextOrFileEntered

    ' Call the Modify method of the step class to
carry out the update
    mcStep.Modify

End Sub

Private Property Get cStep_NextStepId() As
Long

    cStep_NextStepId = mcStep.NextStepId

End Property

Private Property Set cStep_NodeDB(RHS As
DAO.Database)

    Set mcStep.NodeDB = RHS

End Property

Private Property Get cStep_NodeDB() As
DAO.Database

    Set cStep_NodeDB = mcStep.NodeDB

End Property

Private Function cStep_OldVersionNo() As
String
    cStep_OldVersionNo =
mcStep.OldVersionNo
End Function

Private Property Let cStep_OutputFile(ByVal
RHS As String)

    mcStep.OutputFile = RHS

End Property

Private Property Get cStep_OutputFile() As String

    cStep_OutputFile = mcStep.OutputFile

End Property

Private Property Let cStep_ParentStepId(ByVal
RHS As Long)

    ' A global step cannot have any sub-steps
associated with it
    ' Hence, the parent step id and parent version
number will be zero
    mcStep.ParentStepId = 0

End Property

Private Property Get cStep_ParentStepId() As Long

    cStep_ParentStepId = mcStep.ParentStepId

End Property

Private Property Let cStep_ParentVersionNo(ByVal
RHS As String)

    ' A global step cannot have any sub-steps
associated with it
    ' Hence, the parent step id and parent version
number will be zero
    mcStep.ParentVersionNo = gstrMinVersion

End Property

Private Property Get cStep_ParentVersionNo() As
String

    cStep_ParentVersionNo =
mcStep.ParentVersionNo

End Property

Private Property Let cStep_Position(ByVal RHS As
Long)

    mcStep.Position = RHS

End Property

Private Property Get cStep_Position() As Long

    cStep_Position = mcStep.Position

End Property

Private Sub cStep_RemoveIterator(cItRecord As
cIterator)

    Call mcStep.RemoveIterator(cItRecord)

End Sub

Private Sub cStep_SaveIterators()

    Call mcStep.SaveIterators

End Sub

Private Property Let cStep_SequenceNo(ByVal
RHS As Integer)

    mcStep.SequenceNo = RHS
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
End Property

Private Property Get cStep_SequenceNo() As Integer

   cStep_SequenceNo = mcStep.SequenceNo

End Property

Private Property Let cStep_StepId(ByVal RHS As Long)

   mcStep.StepId = RHS

End Property
Private Property Get cStep_StepId() As Long

   cStep_StepId = mcStep.StepId

End Property


Private Property Let cStep_StepLabel(ByVal RHS As String)

   mcStep.StepLabel = RHS

End Property

Private Property Get cStep_StepLabel() As String

   cStep_StepLabel = mcStep.StepLabel

End Property

Private Property Let cStep_StartDir(ByVal RHS As String)

   mcStep.StartDir = RHS

End Property

Private Property Get cStep_StartDir() As String

   cStep_StartDir = mcStep.StartDir

End Property

Private Property Let cStep_StepLevel(ByVal RHS As Integer)

   ' A global step cannot have any sub-steps associated with it
   ' Hence, it will always be at step level 0
   mcStep.StepLevel = 0

End Property

Private Property Get cStep_StepLevel() As Integer

   cStep_StepLevel = mcStep.StepLevel

End Property

Private Property Let cStep_StepText(ByVal RHS As String)

   mcStep.StepText = RHS

End Property

Private Property Get cStep_StepText() As String

   cStep_StepText = mcStep.StepText

End Property

Private Property Let cStep_StepTextFile(ByVal RHS As String)

   mcStep.StepTextFile = RHS

End Property

Private Property Get cStep_StepTextFile() As String

   cStep_StepTextFile = mcStep.StepTextFile

End Property

Private Property Let cStep_StepType(RHS As gintStepType)

   mcStep.StepType = gintGlobalStep

End Property

Private Property Get cStep_StepType() As gintStepType

   cStep_StepType = mcStep.StepType

End Property

Private Sub cStep_UnloadIterators()

   Call mcStep.UnloadIterators

End Sub

Private Sub cStep_UpdateIterator(cItRecord As cIterator)

   Call mcStep.UpdateIterator(cItRecord)

End Sub

Private Sub cStep_UpdateIteratorVersion()

   Call mcStep.UpdateIteratorVersion

End Sub

Private Sub cStep_Validate()
   ' The validate routines for each of the steps will
   ' carry out the specific validations for the type and
   ' call the generic validation routine

   On Error GoTo cStep_ValidateErr
   mstrSource = mstrModuleName & "cStep_Validate"

   ' Validations specific to global steps

   ' Check if the step text or a file name has been
   ' specified
   Call StepTextOrFileEntered

   ' The step level must be zero for all globals
   If mcStep.StepLevel <> 0 Then
      ShowError errStepLevelZeroForGlobal
      On Error GoTo 0
      Err.Raise vbObjectError + errValidateFailed, _
            gstrSource, _
            LoadResString(errValidateFailed)
   End If

   If mcStep.EnabledFlag Then
      ShowError errEnabledFlagFalseForGlobal
      On Error GoTo 0
      Err.Raise vbObjectError + errValidateFailed, _
            gstrSource, _
            LoadResString(errValidateFailed)
   End If

   If mcStep.DegreeParallelism > 0 Then
      ShowError errDegParallelismNullForGlobal
      On Error GoTo 0
      Err.Raise vbObjectError + errValidateFailed, _
            gstrSource, _
            LoadResString(errValidateFailed)
   End If

   If mcStep.ContinuationCriteria > 0 Then
      ShowError errContCriteriaNullForGlobal
      On Error GoTo 0
      Err.Raise vbObjectError + errValidateFailed, _
            gstrSource, _
            LoadResString(errValidateFailed)
   End If

   mcStep.Validate

   Exit Sub

cStep_ValidateErr:
   LogErrors Errors
   mstrSource = mstrModuleName & "cStep_Validate"
   On Error GoTo 0
   Err.Raise vbObjectError + errValidateFailed, _
         mstrSource, _
         LoadResString(errValidateFailed)
End Sub
Private Sub StepTextOrFileEntered()
   ' Checks if either the step text or the name of the file containing
   ' the text has been entered
   ' If both of them are null or both of them are not null,
   ' the global step is invalid and an error is raised

   If StringEmpty(mcStep.StepText) And StringEmpty(mcStep.StepTextFile) Then
      ShowError errStepTextAndFileNull
      On Error GoTo 0
      Err.Raise vbObjectError + errStepTextAndFileNull, _
         mstrSource, LoadResString(errStepTextAndFileNull)
   ElseIf Not StringEmpty(mcStep.StepText) And Not StringEmpty(mcStep.StepTextFile) Then
      ShowError errStepTextOrFile
      On Error GoTo 0
      Err.Raise vbObjectError + errStepTextOrFile, _
         mstrSource, LoadResString(errStepTextOrFile)
   End If

End Sub

Private Property Let cStep_VersionNo(ByVal RHS As String)
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   mcStep.VersionNo = RHS

End Property

Private Property Get cStep_VersionNo() As
String

   cStep_VersionNo = mcStep.VersionNo

End Property

Private Property Let
cStep_WorkspaceId(ByVal RHS As Long)

   mcStep.WorkspaceId = RHS

End Property

Private Property Get cStep_WorkspaceId()
As Long

   cStep_WorkspaceId =
mcStep.WorkspaceId

End Property
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cInstance"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:       cInstance.cls
'             Microsoft TPC-H Kit Ver. 1.00
'             Copyright Microsoft, 1999
'             All Rights Reserved
'
'
'  PURPOSE:    Encapsulates the properties
and methods of an instance.
'             An instance is created when a step
is executed for a
'             particular iterator value (if
applicable) at 'run' time.
'             Contains functions to determine if
an instance is running,
'             complete, and so on.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

' Used to indicate the source module name
when errors
' are raised by this class
Private Const mstrModuleName As String =
"cInstance."
Private mstrSource As String

Private mcStep As cStep
Public Key As String ' Node key for the step
being executed
Public InstanceId As Long
Public ParentInstanceId As Long ' The
parent instance
Private mblnNoMoreToStart As Boolean
Private mblnComplete As Boolean
Public StartTime As Currency
Public EndTime As Currency
Public ElapsedTime As Currency
Private mintStatus As InstanceStatus
Public DegreeParallelism As Integer
Private mcIterators As cRunColIt

' A collection of all the sub-steps for this step
Private mcSubSteps As cSubSteps
Public Sub UpdateStartTime(lStepId As Long,
Optional ByVal StartTm As Currency =
gdtmEmpty, _
      Optional ByVal EndTm As Currency =
gdtmEmpty, _
      Optional ByVal Elapsed As Currency = 0)
   ' We do not maintain start and end
timestamps for the constraint
   ' of a step. Hence we check if the process
that just started/
   ' terminated is the worker step that is being
executed. If so,
   ' we update the start/end time and status on
the instance record.

   BugAssert (StartTm <> gdtmEmpty) Or
(EndTm <> gdtmEmpty), "Mandatory
parameter missing."

   ' Make sure that we are executing the actual
step and not
   ' a pre or post-execution constraint
   If mcStep.StepId = lStepId Then
      If StartTm <> 0 Then
         StartTime = StartTm
         mintStatus = gintRunning
      Else
         EndTime = EndTm
         ElapsedTime = Elapsed
         mintStatus = gintComplete
      End If
   End If

End Sub
Public Function
ValidForIteration(cParentInstance As
cInstance, _
      ByVal intConsType As ConstraintType)
As Boolean
   ' Returns true if the instance passed in is the
first or
   ' last iteration for the step, depending on the
constraint type

   Dim cSubStepRec As cSubStep
   Dim vntIterators As Variant

   On Error GoTo ValidForIterationErr

   If cParentInstance Is Nothing Then
      ' This will only be true for the dummy
instance, which
      ' cannot have any iterators defined for it
      ValidForIteration = True
      Exit Function
   End If

   vntIterators = mcStep.Iterators

   If Not StringEmpty(mcStep.IteratorName)
And Not IsEmpty(vntIterators) Then

      Set cSubStepRec =
cParentInstance.QuerySubStep(mcStep.StepId)

      If intConsType = gintPreStep Then
         ' Pre-execution constraints will only be
executed
         ' before the first iteration
         If
cSubStepRec.LastIterator.IteratorType =
gintValue Then
            ValidForIteration =
(cSubStepRec.LastIterator.Sequence = _
                gintMinIteratorSequence)
         Else
            ValidForIteration =
(cSubStepRec.LastIterator.Value = _

cSubStepRec.LastIterator.RangeFrom)
         End If
      Else
         ' Post-execution constraints will only be
executed
         ' after the last iteration - check if there are
any
         ' pending iterations
         ValidForIteration =
cSubStepRec.NextIteration(mcStep) Is Nothing
      End If
   Else
      ValidForIteration = True
   End If

   Exit Function

ValidForIterationErr:
   ' Log the error code raised by Visual Basic
   Call LogErrors(Errors)
   On Error GoTo 0
   mstrSource = mstrModuleName &
"ValidForIteration"
   Err.Raise vbObjectError +
errExecInstanceFailed, _
         mstrSource,
LoadResString(errExecInstanceFailed)

End Function

Public Sub CreateSubStep(cSubStepDtls As cStep,
RunParams As cArrParameters)

   Dim cNewSubStep As cSubStep

   On Error GoTo CreateSubStepErr

   Set cNewSubStep = New cSubStep

   cNewSubStep.StepId = cSubStepDtls.StepId
   cNewSubStep.TasksComplete = 0
   cNewSubStep.TasksRunning = 0

   ' Initialize the iterator for the instance
   Set cNewSubStep.LastIterator = New
cRunItDetails
   Call cNewSubStep.InitializeIt(cSubStepDtls,
RunParams)

   ' Add add the substep to the collection
   mcSubSteps.Add cNewSubStep

   Set cNewSubStep = Nothing

   Exit Sub

CreateSubStepErr:
   ' Log the error code raised by Visual Basic
   Call LogErrors(Errors)
   On Error GoTo 0
   mstrSource = mstrModuleName &
"CreateSubStep"
   Err.Raise vbObjectError + errProgramError,
mstrSource, _
      LoadResString(errProgramError)

End Sub
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
Public Function QuerySubStep(ByVal
SubStepId As Long) As cSubStep
   ' Retrieves the sub-step record for the
passed in sub-step id

   Dim lngIndex As Long

   On Error GoTo QuerySubStepErr

   ' Find the sub-step node with the matching
step id
   For lngIndex = 0 To mcSubSteps.Count -
1
      If mcSubSteps(lngIndex).StepId =
SubStepId Then
         Set QuerySubStep =
mcSubSteps(lngIndex)
         Exit For
      End If
   Next lngIndex

   Exit Function

QuerySubStepErr:
   ' Log the error code raised by Visual
Basic
   Call LogErrors(Errors)
   On Error GoTo 0
   mstrSource = mstrModuleName &
"QuerySubStep"
   Err.Raise vbObjectError +
errNavInstancesFailed, _
         mstrSource,
LoadResString(errNavInstancesFailed)

End Function
Public Property Let AllStarted(ByVal vdata
As Boolean)

   'bugmessage "Set All Started to " & vData
& " for : " & _
      mstrKey

   mblnNoMoreToStart = vdata

End Property
Public Property Get AllStarted() As Boolean

   AllStarted = mblnNoMoreToStart

End Property
Public Property Let AllComplete(ByVal
vdata As Boolean)

   'bugmessage "Set All Complete to " &
vData & " for : " & _
      mstrKey

   mblnComplete = vdata

End Property

Public Property Get AllComplete() As
Boolean

   AllComplete = mblnComplete

End Property

Public Sub ChildExecuted(mlngStepId As
Long)
   ' This procedure is called when a sub-step
executes.

   Dim lngIndex As Long

   On Error GoTo ChildExecutedErr

   BugAssert mcStep.StepType =
gintManagerStep

   For lngIndex = 0 To mcSubSteps.Count - 1
      If mcSubSteps(lngIndex).StepId =
mlngStepId Then
         mcSubSteps(lngIndex).TasksRunning =
_

mcSubSteps(lngIndex).TasksRunning + 1
'         BugMessage "Tasks Running for Step
Id : " & _
'
CStr(mcSubSteps(lngIndex).StepId) & _
'                  " Instance Id: " & InstanceId & _
'                  " = " &
mcSubSteps(lngIndex).TasksRunning
         Exit For
      End If
   Next lngIndex

   If lngIndex > mcSubSteps.Count - 1 Then
      ' The child step wasn't found - raise an
error
      On Error GoTo 0
      Err.Raise vbObjectError +
errInvalidChild, mstrModuleName, _
            LoadResString(errInvalidChild)
   End If

   Exit Sub

ChildExecutedErr:
   ' Log the error code raised by Visual Basic
   Call LogErrors(Errors)
   On Error GoTo 0
   Err.Raise vbObjectError +
errInstanceOpFailed, mstrModuleName &
"ChildExecuted", _
         LoadResString(errInstanceOpFailed)

End Sub
Public Sub ChildTerminated(mlngStepId As
Long)
   ' This procedure is called when any sub-step
process
   ' terminates. Note: The TasksComplete field
will be
   ' updated only when all the instances for a
sub-step
   ' complete execution.
   Dim lngIndex As Long

   On Error GoTo ChildTerminatedErr

   BugAssert mcStep.StepType =
gintManagerStep

   For lngIndex = 0 To mcSubSteps.Count - 1

      If mcSubSteps(lngIndex).StepId =
mlngStepId Then
         mcSubSteps(lngIndex).TasksRunning =
_

mcSubSteps(lngIndex).TasksRunning - 1
'         BugMessage "Tasks Running for Step
Id : " & _
'
CStr(mcSubSteps(lngIndex).StepId) & _
'                  " Instance Id: " & InstanceId & _
'                  " = " &
mcSubSteps(lngIndex).TasksRunning

         BugAssert
mcSubSteps(lngIndex).TasksRunning >= 0, _
            "Tasks running for " &
CStr(mlngStepId) & _
            " Instance Id " & InstanceId & " is less
than 0."
         Exit For
      End If
   Next lngIndex

   If lngIndex > mcSubSteps.Count - 1 Then
      ' The child step wasn't found - raise an error
      On Error GoTo 0
      Err.Raise errInvalidChild, mstrModuleName &
"ChildTerminated", _
            LoadResString(errInvalidChild)
   End If

   Exit Sub

ChildTerminatedErr:
   ' Log the error code raised by Visual Basic
   Call LogErrors(Errors)
   On Error GoTo 0
   mstrSource = mstrModuleName &
"ChildTerminated"
   Err.Raise vbObjectError + errInstanceOpFailed,
mstrSource, _
         LoadResString(errInstanceOpFailed)

End Sub
Public Sub ChildCompleted(mlngStepId As Long)
   ' This procedure is called when any a sub-step
completes
   ' execution. Note: The TasksComplete field will
be
   ' incremented.
   Dim lngIndex As Long

   On Error GoTo ChildCompletedErr

   BugAssert mcStep.StepType = gintManagerStep

   For lngIndex = 0 To mcSubSteps.Count - 1
      BugAssert
mcSubSteps(lngIndex).TasksComplete >= 0, _
            "Tasks complete for " &
CStr(mcSubSteps(lngIndex).StepId) & _
            " Instance Id " & InstanceId & " is less
than 0."

      If mcSubSteps(lngIndex).StepId = mlngStepId
Then
         mcSubSteps(lngIndex).TasksComplete = _
               mcSubSteps(lngIndex).TasksComplete
+ 1
'         BugMessage "Tasks Complete for Step Id :
" & _
'                  CStr(mcSubSteps(lngIndex).StepId) &
_
'                  " Instance Id: " & InstanceId & _
'                  " = " &
mcSubSteps(lngIndex).TasksComplete
         Exit For
      End If
   Next lngIndex

   If lngIndex > mcSubSteps.Count - 1 Then
      ' The child step wasn't found - raise an error
      On Error GoTo 0
      Err.Raise errInvalidChild, mstrModuleName, _
            LoadResString(errInvalidChild)
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   End If

   Exit Sub

ChildCompletedErr:
   ' Log the error code raised by Visual
Basic
   Call LogErrors(Errors)
   On Error GoTo 0
   Err.Raise vbObjectError +
errInstanceOpFailed, mstrModuleName &
"ChildCompleted", _
         LoadResString(errInstanceOpFailed)

End Sub
Public Sub ChildDeleted(mlngStepId As
Long)
   ' This procedure is called when a sub-step
needs to be re-executed
   ' Note: The TasksComplete field is
decremented. We needn't worry about
   ' the TasksRunning field since no steps
are currently running.
   Dim lngIndex As Long

   On Error GoTo ChildDeletedErr

   BugAssert mcStep.StepType =
gintManagerStep

   For lngIndex = 0 To mcSubSteps.Count -
1

      If mcSubSteps(lngIndex).StepId =
mlngStepId Then

mcSubSteps(lngIndex).TasksRunning = _

mcSubSteps(lngIndex).TasksRunning - 1

         BugAssert
mcSubSteps(lngIndex).TasksRunning >= 0,
_
               "Tasks running for " &
CStr(mcSubSteps(lngIndex).StepId) & _
               " Instance Id " & InstanceId & "
is less than 0."
         Exit For
      End If
   Next lngIndex

   If lngIndex > mcSubSteps.Count - 1 Then
      ' The child step wasn't found - raise an
error
      On Error GoTo 0
      Err.Raise errInvalidChild,
mstrModuleName, _
            LoadResString(errInvalidChild)
   End If

   Exit Sub

ChildDeletedErr:
   ' Log the error code raised by Visual
Basic
   Call LogErrors(Errors)
   On Error GoTo 0
   Err.Raise vbObjectError +
errInstanceOpFailed, mstrModuleName &
"ChildDeleted", _
         LoadResString(errInstanceOpFailed)

End Sub
Private Sub RaiseErrForWorker()

   If mcStep.StepType <> gintManagerStep
Then
      On Error GoTo 0
      mstrSource = mstrModuleName &
"RaiseErrForWorker"
      Err.Raise vbObjectError +
errInvalidForWorker, _
            mstrSource, _
            LoadResString(errInvalidForWorker)
   End If

End Sub

Public Property Get Step() As cStep

   Set Step = mcStep

End Property
Public Property Get Iterators() As cRunColIt

   Set Iterators = mcIterators

End Property
Public Property Get SubSteps() As cSubSteps

   Call RaiseErrForWorker

   Set SubSteps = mcSubSteps

End Property
Public Property Set Step(cRunStep As cStep)

   Set mcStep = cRunStep

End Property

Public Property Set Iterators(cIts As
cRunColIt)

   Set mcIterators = cIts

End Property
Public Property Get IsPending() As Boolean
   ' Returns true if the step has any substeps
that need
   ' execution
   Dim lngIndex As Long
   Dim lngRunning As Long

   Call RaiseErrForWorker

   If Not mblnComplete And Not
mblnNoMoreToStart Then
      ' Get a count of all the substeps that are
already being
      ' executed
      lngRunning = 0
      For lngIndex = 0 To mcSubSteps.Count -
1
         lngRunning = lngRunning +
mcSubSteps(lngIndex).TasksRunning
      Next lngIndex

      IsPending = (lngRunning <
DegreeParallelism)
   Else
      ' This should be sufficient to prove that
there r no
      ' more sub-steps to be executed.
      ' mblnComplete: Handles the case where
all steps have
      ' been executed
      ' mblnNoMoreToStart: Handles the case
where the step
      ' has a degree of parallelism greater than the
total
      ' number of sub-steps available to execute
      IsPending = False
   End If

End Property
Public Property Get IsRunning() As Boolean
   ' Returns true if the any one of the substeps is still
   ' executing
   Dim lngIndex As Long

   Call RaiseErrForWorker

   IsRunning = False

   ' If a substep has no currently executing tasks and
   ' the tasks completed is greater than zero, then we
can
   ' assume that it has completed execution
(otherwise we
   ' would've run a new task the moment one
completed!)
   For lngIndex = 0 To mcSubSteps.Count - 1
      If mcSubSteps(lngIndex).TasksRunning > 0
Then
         IsRunning = True
         Exit For
      End If
   Next lngIndex

End Property
Public Property Get TotalRunning() As Long
   ' Returns the total number of substeps that are
executing
   Dim lngTotalProcesses As Long
   Dim lngIndex As Long

   Call RaiseErrForWorker

   lngTotalProcesses = 0
   For lngIndex = 0 To mcSubSteps.Count - 1
      BugAssert
mcSubSteps(lngIndex).TasksRunning >= 0, _
            "Tasks running for " &
CStr(mcSubSteps(lngIndex).StepId) & _
            " is less than 0."
      lngTotalProcesses = lngTotalProcesses +
mcSubSteps(lngIndex).TasksRunning
   Next lngIndex

   TotalRunning = lngTotalProcesses
End Property
Public Property Get RunningForStep(lngSubStepId
As Long) As Long
   ' Returns the total number of instances of the
substep
   ' that are executing
   Dim lngIndex As Long

   Call RaiseErrForWorker

   For lngIndex = 0 To mcSubSteps.Count - 1
      BugAssert
mcSubSteps(lngIndex).TasksRunning >= 0, _
            "Tasks running for " &
CStr(mcSubSteps(lngIndex).StepId) & _
            " is less than 0."

      If mcSubSteps(lngIndex).StepId =
lngSubStepId Then
         RunningForStep =
mcSubSteps(lngIndex).TasksRunning
         Exit For
      End If
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
    Next lngIndex

    If lngIndex > mcSubSteps.Count - 1 Then
        ' The child step wasn't found - raise an
error
        On Error GoTo 0
        Err.Raise errInvalidChild, mstrSource,
_
                LoadResString(errInvalidChild)
    End If

End Property

Public Property Let Status(ByVal vdata As
InstanceStatus)

    mintStatus = vdata

End Property

Public Property Get Status() As
InstanceStatus

    Status = mintStatus

End Property
Private Sub Class_Initialize()

    Set mcSubSteps = New cSubSteps

    mblnNoMoreToStart = False
    mblnComplete = False
    StartTime = gdtmEmpty
    EndTime = gdtmEmpty

End Sub

Private Sub Class_Terminate()

    mcSubSteps.Clear
    Set mcSubSteps = Nothing

End Sub
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cInstances"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:      cInstances.cls
'            Microsoft TPC-H Kit Ver. 1.00
'            Copyright Microsoft, 1999
'            All Rights Reserved
'
'
'  PURPOSE:    Implements a collection of
cInstance objects.
'            Type-safe wrapper around cVector.
'            Also contains additional functions
to query an instance, etc.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

' Used to indicate the source module name
when errors
' are raised by this class
Private Const mstrModuleName As String =
"cInstance."
Private mstrSource As String
Private mcInstances As cVector

Public Function QueryInstance(ByVal
InstanceId As Long) As cInstance
    ' Retrieves the record for the passed in
instance from
    ' the collection

    Dim lngIndex As Long

    On Error GoTo QueryInstanceErr

    ' Check for valid values of the instance id
    If InstanceId > 0 Then
        ' Find the run node with the matching step
id
        For lngIndex = 0 To Count() - 1
            If mcInstances(lngIndex).InstanceId =
InstanceId Then
                Set QueryInstance =
mcInstances(lngIndex)
                Exit For
            End If
        Next lngIndex

        If lngIndex > mcInstances.Count - 1 Then
            On Error GoTo 0
            Err.Raise vbObjectError +
errQueryFailed, mstrSource, _
                    LoadResString(errQueryFailed)
        End If
    Else
        On Error GoTo 0
        Err.Raise vbObjectError +
errQueryFailed, mstrSource, _
                LoadResString(errQueryFailed)
    End If

    Exit Function

QueryInstanceErr:
    ' Log the error code raised by Visual Basic
    Call LogErrors(Errors)
    On Error GoTo 0
    mstrSource = mstrModuleName &
"QueryInstance"
    Err.Raise vbObjectError + errQueryFailed, _
            mstrSource,
LoadResString(errQueryFailed)

End Function

Public Function QueryPendingInstance(ByVal
ParentInstanceId As Long, _
        ByVal lngSubStepId As Long) As
cInstance
    ' Retrieves a pending instance for the passed
in substep
    ' and the given parent instance id.

    Dim lngIndex As Long

    On Error GoTo QueryPendingInstanceErr

    ' Find the run node with the matching step id
    For lngIndex = 0 To Count() - 1
        If mcInstances(lngIndex).ParentInstanceId
= ParentInstanceId And _
                mcInstances(lngIndex).Step.StepId =
lngSubStepId Then
            ' Put in a separate if condition since the
IsPending
            ' property is valid only for manager
steps. If the
            ' calling procedure does not pass a manager
step
            ' identifier, the procedure will error out.
            If mcInstances(lngIndex).IsPending Then
                Set QueryPendingInstance =
mcInstances(lngIndex)
                Exit For
            End If
        End If
    Next lngIndex

    Exit Function

QueryPendingInstanceErr:
    ' Log the error code raised by Visual Basic
    Call LogErrors(Errors)
    On Error GoTo 0
    mstrSource = mstrModuleName &
"QueryPendingInstance"
    Err.Raise vbObjectError + errQueryFailed, _
            mstrSource, LoadResString(errQueryFailed)

End Function
Public Function InstanceAborted(cSubStepRec As
cSubStep) As Boolean

    Dim lIndex As Long

    InstanceAborted = False

    For lIndex = 0 To Count() - 1
        If mcInstances(lIndex).Step.StepId =
cSubStepRec.StepId And _
                mcInstances(lIndex).Status = gintAborted
Then
            InstanceAborted = True
            Exit For
        End If
    Next lIndex

End Function
Public Function
CompletedInstanceExists(lParentInstance As Long,
_
        cSubStepDtls As cStep) As Boolean
    ' Checks if there is a completed instance of the
passed in step

    Dim lngIndex As Long

    CompletedInstanceExists = False

    If cSubStepDtls.StepType = gintManagerStep
Then
        ' Find the run node with the matching step id
        For lngIndex = 0 To Count() - 1
            If mcInstances(lngIndex).ParentInstanceId =
lParentInstance And _
                    mcInstances(lngIndex).Step.StepId =
cSubStepDtls.StepId Then
                ' Put in a separate if condition since the
IsPending
                ' property is valid only for manager steps.
                BugAssert (Not
mcInstances(lngIndex).IsPending), "Pending
instance exists!"

                CompletedInstanceExists = True
                Exit Function
            End If
        Next lngIndex
    End If

End Function
Public Sub Add(ByVal objItem As cInstance)
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   mcInstances.Add objItem

End Sub


Public Sub Clear()

   mcInstances.Clear

End Sub


Public Function Count() As Long

   Count = mcInstances.Count

End Function


Public Function Delete(ByVal lngDelete As
Long) As cInstance

   Set Delete =
mcInstances.Delete(lngDelete)

End Function


Public Property Set Item(Optional ByVal
Position As Long, _
      RHS As cInstance)

   If Position = -1 Then
      Position = 0
   End If
   Set mcInstances(Position) = RHS

End Property

Public Property Get Item(Optional ByVal
Position As Long = -1) _
      As cInstance
Attribute Item.VB_UserMemId = 0

   If Position = -1 Then
      Position = 0
   End If
   Set Item = mcInstances.Item(Position)

End Property

Private Sub Class_Initialize()

   Set mcInstances = New cVector

End Sub


Private Sub Class_Terminate()

   Set mcInstances = Nothing

End Sub


VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cIterator"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:       cIterator.cls
'              Microsoft TPC-H Kit Ver. 1.00
'              Copyright Microsoft, 1999
'              All Rights Reserved
'
'
'  PURPOSE:    Encapsulates the properties
and methods of an iterator.
'              Contains functions to insert, update
and delete
'              iterator_values records from the
database.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

Implements cNode

' Module level variables to store the property
values
Private mintType As Integer
Private mintSequenceNo As Integer
Private mstrValue As String
Private mdbsIteratorDB As Database
Private mintOperation As Integer
Private mlngPosition As Long

Private Const mstrModuleName As String =
"cIterator."
Private mstrSource As String

Public Enum ValueType
   gintFrom = 1
   gintTo
   gintStep
   gintValue
End Enum
Public Property Get Value() As String

   Value = mstrValue

End Property
Public Property Let Value(ByVal vdata As
String)

   mstrValue = vdata

End Property

Public Property Get IndOperation() As
Operation

   IndOperation = mintOperation

End Property
Public Property Let IndOperation(ByVal vdata
As Operation)

   On Error GoTo IndOperationErr
   mstrSource = mstrModuleName &
"IndOperation"

   ' The valid operations are define in the
cOperations
   ' class. Check if the operation is valid
   Select Case vdata
      Case QueryOp, InsertOp, UpdateOp,
DeleteOp
         mintOperation = vdata

      Case Else
         On Error GoTo 0
         Err.Raise vbObjectError +
errInvalidOperation, _
            mstrSource,
LoadResString(errInvalidOperation)
   End Select

   Exit Property

IndOperationErr:
   LogErrors Errors
   mstrSource = mstrModuleName &
"IndOperation"
   On Error GoTo 0
   Err.Raise vbObjectError +
errLetOperationFailed, _
      mstrSource,
LoadResString(errLetOperationFailed)

End Property

Public Function Clone() As cIterator

   ' Creates a copy of a given Iterator

   Dim cItClone As cIterator

   On Error GoTo CloneErr

   Set cItClone = New cIterator

   ' Copy all the iterator properties to the newly
   ' created object
   cItClone.IteratorType = mintType
   cItClone.SequenceNo = mintSequenceNo
   cItClone.IndOperation = mintOperation
   cItClone.Value = mstrValue

   ' And set the return value to the newly created
Iterator
   Set Clone = cItClone

   Exit Function

CloneErr:
   LogErrors Errors
   mstrSource = mstrModuleName & "Clone"
   On Error GoTo 0
   Err.Raise vbObjectError + errCloneFailed, _
         mstrSource, LoadResString(errCloneFailed)

End Function
Public Property Get SequenceNo() As Integer

   SequenceNo = mintSequenceNo

End Property

Public Property Let SequenceNo(ByVal vdata As
Integer)
   mintSequenceNo = vdata
End Property

Public Sub Add(ByVal lngStepId As Long, _
      strVersion As String)
   ' Inserts a new iterator values record into the
database

   Dim strInsert As String
   Dim qy As DAO.QueryDef

   On Error GoTo AddIteratorErr

   ' First check if the database object is valid
   Call CheckDB

   ' Create a temporary querydef object
   strInsert = "insert into iterator_values " & _
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
        "( step_id, version_no, type, " & _
        " iterator_value, sequence_no ) " & _
        " values ( [st_id], [ver_no], [it_typ], " & _
        " [it_val], [seq_no] )"
   Set qy = mdbsIteratorDB.CreateQueryDef(gstrEmptyString, strInsert)

   ' Call a procedure to execute the Querydef object
   Call AssignParameters(qy, lngStepId, strVersion)

   qy.Execute dbFailOnError
   qy.Close

   Exit Sub

AddIteratorErr:
   LogErrors Errors
   mstrSource = mstrModuleName & "AddIterator"
   On Error GoTo 0
   Err.Raise vbObjectError + errInsertIteratorFailed, _
        mstrSource, _
LoadResString(errInsertIteratorFailed)
End Sub

Private Sub AssignParameters(qyExec As DAO.QueryDef, _
      ByVal lngStepId As Long, _
      strVersion As String)
   ' Assigns values to the parameters in the querydef object
   ' The parameter names are cryptic to make them different
   ' from the field names. When the parameter names are
   ' the same as the field names, parameters in the where
   ' clause do not get created.

   Dim prmParam As DAO.Parameter

   On Error GoTo AssignParametersErr
   mstrSource = mstrModuleName & "AssignParameters"

   For Each prmParam In qyExec.Parameters
      Select Case prmParam.Name
         Case "[st_id]"
            prmParam.Value = lngStepId

         Case "[ver_no]"
            prmParam.Value = strVersion

         Case "[it_typ]"
            prmParam.Value = mintType

         Case "[it_val]"
            prmParam.Value = mstrValue

         Case "[seq_no]"
            prmParam.Value = mintSequenceNo

         Case Else
            ' Write the parameter name that is faulty
            WriteError errInvalidParameter, mstrSource, _
                  prmParam.Name
            On Error GoTo 0
            Err.Raise errInvalidParameter, mstrSource, _
LoadResString(errInvalidParameter)
      End Select
   Next prmParam

   Exit Sub

AssignParametersErr:

   mstrSource = mstrModuleName & "AssignParameters"
   Call LogErrors(Errors)
   On Error GoTo 0
   Err.Raise vbObjectError + errAssignParametersFailed, _
      mstrSource, LoadResString(errAssignParametersFailed)

End Sub
Private Sub CheckDB()
   ' Check if the database object has been initialized

   If mdbsIteratorDB Is Nothing Then
      ShowError errInvalidDB
      On Error GoTo 0
      Err.Raise vbObjectError + errInvalidDB, _
         mstrModuleName, LoadResString(errInvalidDB)
   End If

End Sub

Public Sub Delete(ByVal lngStepId As Long, _
      strVersion As String)
   ' Deletes the step iterator record from the database

   Dim strDelete As String
   Dim qy As DAO.QueryDef

   On Error GoTo DeleteIteratorErr
   mstrSource = mstrModuleName & "DeleteIterator"

   ' There can be multiple iterators for a step.
   ' However the values that an iterator for a step can
   ' assume will be unique, meaning that a combination of
   ' the iterator_id and value will be unique.
   strDelete = "delete from iterator_values " & _
         " where step_id = [st_id]" & _
         " and version_no = [ver_no]" & _
         " and iterator_value = [it_val] "
   Set qy = mdbsIteratorDB.CreateQueryDef(gstrEmptyString, strDelete)

   Call AssignParameters(qy, lngStepId, strVersion)
   qy.Execute dbFailOnError

   qy.Close

   Exit Sub

DeleteIteratorErr:
   LogErrors Errors
   mstrSource = mstrModuleName & "DeleteIterator"
   On Error GoTo 0
   Err.Raise vbObjectError + errDeleteIteratorFailed, _
         mstrSource, _
         LoadResString(errDeleteIteratorFailed)
End Sub
Public Sub Update(ByVal lngStepId As Long, strVersion As String)
   ' Updates the sequence no of the step iterator record
   ' in the database

   Dim strUpdate As String
   Dim qy As QueryDef

   On Error GoTo UpdateErr

   ' First check if the database object is valid
   Call CheckDB

   If mintType = gintValue Then
      ' If the iterator is of type value, only the sequence of the values can get updated
      strUpdate = "Update iterator_values " & _
            " set sequence_no = [seq_no] " & _
            " where step_id = [st_id]" & _
            " and version_no = [ver_no]" & _
            " and iterator_value = [it_val] "
   Else
      ' If the iterator is of type range, only the values can get updated
      strUpdate = "Update iterator_values " & _
            " set iterator_value = [it_val] " & _
            " where step_id = [st_id]" & _
            " and version_no = [ver_no]" & _
            " and type = [it_typ] "
   End If

   Set qy = mdbsIteratorDB.CreateQueryDef(gstrEmptyString, strUpdate)

   ' Call a procedure to assign the parameter values to the
   ' querydef object
   Call AssignParameters(qy, lngStepId, strVersion)
   qy.Execute dbFailOnError

   qy.Close

   Exit Sub

UpdateErr:
   LogErrors Errors
   mstrSource = mstrModuleName & "Update"
   On Error GoTo 0
   Err.Raise vbObjectError + errUpdateConstraintFailed, _
         mstrSource, _
         LoadResString(errUpdateConstraintFailed)

End Sub
Public Property Set NodeDB(vdata As Database)

   Set mdbsIteratorDB = vdata

End Property

Public Property Get NodeDB() As Database

   Set NodeDB = mdbsIteratorDB

End Property
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
Public Property Get Position() As Long

   Position = mlngPosition

End Property
Public Property Let Position(ByVal vdata
As Long)

   mlngPosition = vdata

End Property

Public Property Let IteratorType(ByVal
vdata As ValueType)

   On Error GoTo TypeErr
   mstrSource = mstrModuleName &
"Type"

   ' These constants have been defined in the
enumeration,
   ' Type, which is exposed
   Select Case vdata
      Case gintFrom, gintTo, gintStep,
gintValue
         mintType = vdata

      Case Else
         On Error GoTo 0
         Err.Raise vbObjectError +
errTypeInvalid, _
            mstrSource,
LoadResString(errTypeInvalid)
   End Select

   Exit Property

TypeErr:
   LogErrors Errors
   mstrSource = mstrModuleName &
"Type"
   On Error GoTo 0
   Err.Raise vbObjectError +
errTypeInvalid, _
      mstrSource,
LoadResString(errTypeInvalid)

End Property

Public Property Get IteratorType() As
ValueType

   IteratorType = mintType

End Property
Public Sub Validate()

   ' No validations necessary for the iterator
class

End Sub

Private Sub Class_Initialize()

   ' Initialize the operation indicator variable
to Query
   ' It will be modified later by the collection
class when
   ' inserts, updates or deletes are performed
   mintOperation = QueryOp

End Sub

Private Property Let
cNode_IndOperation(ByVal vdata As
Operation)

   On Error GoTo IndOperationErr
   mstrSource = mstrModuleName &
"IndOperation"

   ' The valid operations are define in the
cOperations
   ' class. Check if the operation is valid
   Select Case vdata
      Case QueryOp, InsertOp, UpdateOp,
DeleteOp
         mintOperation = vdata

      Case Else
         On Error GoTo 0
         Err.Raise vbObjectError +
errInvalidOperation, _
            mstrSource,
LoadResString(errInvalidOperation)
   End Select

   Exit Property

IndOperationErr:
   LogErrors Errors
   mstrSource = mstrModuleName &
"IndOperation"
   On Error GoTo 0
   Err.Raise vbObjectError +
errLetOperationFailed, _
      mstrSource,
LoadResString(errLetOperationFailed)

End Property

Private Property Get cNode_IndOperation() As
Operation

   IndOperation = mintOperation

End Property

Private Property Set cNode_NodeDB(RHS As
DAO.Database)

   Set mdbsIteratorDB = RHS

End Property

Private Property Get cNode_NodeDB() As
DAO.Database

   Set cNode_NodeDB = mdbsIteratorDB

End Property


Private Property Let cNode_Position(ByVal
vdata As Long)

   mlngPosition = vdata

End Property


Private Property Get cNode_Position() As
Long

   cNode_Position = mlngPosition

End Property

Private Sub cNode_Validate()

   ' No validations necessary for the iterator class

End Sub


Private Property Let cNode_Value(ByVal vdata As
String)

   mstrValue = vdata

End Property

Private Property Get cNode_Value() As String

   Value = mstrValue

End Property
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cManager"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:       cManager.cls
'             Microsoft TPC-H Kit Ver. 1.00
'             Copyright Microsoft, 1999
'             All Rights Reserved
'
'
'  PURPOSE:    Encapsulates the properties and
methods of a manager step.
'             Implements the cStep class - carries out
initializations
'             and validations that are specific to
manager steps.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

Implements cStep

' Object variable to keep the step reference in
Private mcStep As cStep

' Used to indicate the source module name when
errors
' are raised by this class
Private mstrSource As String
Private Const mstrModuleName As String =
"cManager."
Private Sub cStep_AddAllIterators()

   Call mcStep.AddAllIterators

End Sub

Private Property Let cStep_StartDir(ByVal RHS As
String)

   mcStep.StartDir = RHS

End Property

Private Property Get cStep_StartDir() As String

   cStep_StartDir = mcStep.StartDir

End Property
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
Private Sub cStep_Delete()

   Call mcStep.Delete

End Sub

Private Property Set cStep_NodeDB(RHS
As DAO.Database)

   Set mcStep.NodeDB = RHS

End Property

Private Function cStep_IncVersionY() As
String

   cStep_IncVersionY =
mcStep.IncVersionY

End Function
Private Function cStep_IsNewVersion() As
Boolean
   cStep_IsNewVersion =
mcStep.IsNewVersion
End Function
Private Function cStep_OldVersionNo() As
String
   cStep_OldVersionNo =
mcStep.OldVersionNo
End Function

Private Function cStep_IncVersionX() As
String

   cStep_IncVersionX =
mcStep.IncVersionX

End Function
Private Sub cStep_UpdateIteratorVersion()

   Call mcStep.UpdateIteratorVersion

End Sub

Private Function cStep_IteratorCount() As
Long

   cStep_IteratorCount =
mcStep.IteratorCount

End Function

Private Sub cStep_UnloadIterators()

   Call mcStep.UnloadIterators

End Sub

Private Sub cStep_DeleteIterator(cItRecord
As cIterator)

   Call mcStep.DeleteIterator(cItRecord)

End Sub
Private Property Get cStep_IteratorName()
As String

   cStep_IteratorName =
mcStep.IteratorName

End Property
Private Property Let
cStep_IteratorName(ByVal RHS As String)

   mcStep.IteratorName = RHS

End Property

Private Sub cStep_SaveIterators()

   Call mcStep.SaveIterators

End Sub
Private Sub cStep_LoadIterator(cItRecord As
cIterator)

   Call mcStep.LoadIterator(cItRecord)

End Sub

Private Property Let cStep_Position(ByVal
RHS As Long)

   mcStep.Position = RHS

End Property
Private Sub cStep_InsertIterator(cItRecord As
cIterator)

   Call mcStep.InsertIterator(cItRecord)

End Sub
Private Function cStep_Iterators() As Variant

   cStep_Iterators = mcStep.Iterators

End Function
Private Sub cStep_ModifyIterator(cItRecord
As cIterator)

   Call mcStep.ModifyIterator(cItRecord)

End Sub
Private Sub cStep_RemoveIterator(cItRecord
As cIterator)

   Call mcStep.RemoveIterator(cItRecord)

End Sub
Private Sub cStep_UpdateIterator(cItRecord
As cIterator)

   Call mcStep.UpdateIterator(cItRecord)

End Sub
Private Sub cStep_AddIterator(cItRecord As
cIterator)

   Call mcStep.AddIterator(cItRecord)

End Sub

Private Property Get cStep_Position() As Long

   cStep_Position = mcStep.Position

End Property

Private Function cStep_Clone(Optional
cCloneStep As cStep) As cStep

   Dim cNewManager As cManager

   Set cNewManager = New cManager
   Set cStep_Clone =
mcStep.Clone(cNewManager)

End Function

Private Property Get cStep_IndOperation() As
Operation

   cStep_IndOperation = mcStep.IndOperation

End Property

Private Property Let cStep_IndOperation(ByVal
RHS As Operation)

   mcStep.IndOperation = RHS

End Property

Private Property Get cStep_NextStepId() As Long

   cStep_NextStepId = mcStep.NextStepId

End Property

Private Property Let cStep_OutputFile(ByVal RHS
As String)

   mcStep.OutputFile = RHS

End Property

Private Property Get cStep_OutputFile() As String

   cStep_OutputFile = mcStep.OutputFile

End Property

Private Property Let cStep_ErrorFile(ByVal RHS
As String)

   mcStep.ErrorFile = RHS

End Property

Private Property Get cStep_ErrorFile() As String

   cStep_ErrorFile = mcStep.ErrorFile

End Property
'Private Property Let cStep_LogFile(ByVal RHS As
String)
'
'   mcStep.LogFile = RHS
'
'End Property
'
'Private Property Get cStep_LogFile() As String
'
'   cStep_LogFile = mcStep.LogFile
'
'End Property

Private Property Let cStep_ArchivedFlag(ByVal
RHS As Boolean)

   mcStep.ArchivedFlag = RHS

End Property

Private Property Get cStep_ArchivedFlag() As
Boolean

   cStep_ArchivedFlag = mcStep.ArchivedFlag

End Property

Private Property Get cStep_NodeDB() As
DAO.Database
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   Set cStep_NodeDB = mcStep.NodeDB

End Property

Private Sub Class_Initialize()

   ' Create the object
   Set mcStep = New cStep

   ' Initialize the object with valid values for
a manager step
   ' The global flag should be the first field
to be initialized
   ' since subsequent validations might try to
check if the
   ' step being created is global
   mcStep.GlobalFlag = False
'   mcStep.GlobalRunMethod =
gintNoOption
   mcStep.StepType = gintManagerStep

   ' Since the manager step does not take any
action, the step
   ' text and file name will always be empty
   mcStep.StepText = gstrEmptyString
   mcStep.StepTextFile = gstrEmptyString

   ' Since the manager step does not take any
action, execution
   ' properties for the step will be empty
   mcStep.ExecutionMechanism =
gintNoOption
   mcStep.FailureDetails = gstrEmptyString
   mcStep.ContinuationCriteria =
gintNoOption

End Sub
Private Sub Class_Terminate()

   ' Remove the step object
   Set mcStep = Nothing

End Sub
Private Sub cStep_Add()

   ' Call the Add method of the step class to
carry out the insert
   mcStep.Add

End Sub
Private Property Get
cStep_ContinuationCriteria() As
ContinuationCriteria

   cStep_ContinuationCriteria =
mcStep.ContinuationCriteria

End Property

Private Property Let
cStep_ContinuationCriteria(ByVal RHS As
ContinuationCriteria)

   ' Since a manager step cannot take any
action, the continuation
   ' criteria property does not apply to it
   mcStep.ContinuationCriteria =
gintNoOption

End Property

Private Property Let
cStep_DegreeParallelism(ByVal RHS As
String)

   mcStep.DegreeParallelism = RHS

End Property

Private Property Get
cStep_DegreeParallelism() As String

   cStep_DegreeParallelism =
mcStep.DegreeParallelism

End Property

Private Sub cStep_DeleteStep()

   On Error GoTo cStep_DeleteStepErr
   mstrSource = mstrModuleName &
"cStep_DeleteStep"

   mcStep.Delete
   Exit Sub

cStep_DeleteStepErr:
   LogErrors Errors
   mstrSource = mstrModuleName &
"cStep_DeleteStep"
   On Error GoTo 0
   Err.Raise vbObjectError +
errDeleteStepFailed, _
         mstrSource, _
         LoadResString(errDeleteStepFailed)

End Sub

Private Property Get cStep_EnabledFlag() As
Boolean

   cStep_EnabledFlag = mcStep.EnabledFlag

End Property

Private Property Let
cStep_EnabledFlag(ByVal RHS As Boolean)

   mcStep.EnabledFlag = RHS

End Property

Private Property Let
cStep_ExecutionMechanism(ByVal RHS As
ExecutionMethod)

   ' Since a manager step cannot take any
action, the Execution
   ' Mechanism property does not apply to it
   mcStep.ExecutionMechanism =
gintNoOption

End Property

Private Property Get
cStep_ExecutionMechanism() As
ExecutionMethod

   cStep_ExecutionMechanism =
mcStep.ExecutionMechanism

End Property

Private Property Let
cStep_FailureDetails(ByVal RHS As String)

   ' Since a manager step cannot take any
action, the Failure
   ' Details property does not apply to it
   mcStep.FailureDetails = gstrEmptyString

End Property

Private Property Get cStep_FailureDetails() As
String

   cStep_FailureDetails = mcStep.FailureDetails

End Property

Private Property Get cStep_GlobalFlag() As
Boolean

   cStep_GlobalFlag = mcStep.GlobalFlag

End Property

Private Property Let cStep_GlobalFlag(ByVal RHS
As Boolean)

   ' Set the global flag to false - this flag is
initialized when
   ' an instance of the class is created. Just making
sure that
   ' nobody changes the value inadvertently
   mcStep.GlobalFlag = False

End Property
Private Sub cStep_Modify()

   ' Call the Modify method of the step class to
carry out the update
   mcStep.Modify

End Sub

Private Property Let cStep_ParentStepId(ByVal
RHS As Long)

   mcStep.ParentStepId = RHS

End Property

Private Property Get cStep_ParentStepId() As Long

   cStep_ParentStepId = mcStep.ParentStepId

End Property

Private Property Let cStep_ParentVersionNo(ByVal
RHS As String)

   mcStep.ParentVersionNo = RHS

End Property

Private Property Get cStep_ParentVersionNo() As
String

   cStep_ParentVersionNo =
mcStep.ParentVersionNo

End Property

Private Property Let cStep_SequenceNo(ByVal
RHS As Integer)

   mcStep.SequenceNo = RHS

End Property

Private Property Get cStep_SequenceNo() As
Integer

   cStep_SequenceNo = mcStep.SequenceNo
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
End Property

Private Property Let cStep_StepId(ByVal
RHS As Long)

   mcStep.StepId = RHS

End Property

Private Property Get cStep_StepId() As
Long

   cStep_StepId = mcStep.StepId

End Property


Private Property Let
cStep_StepLabel(ByVal RHS As String)

   mcStep.StepLabel = RHS

End Property

Private Property Get cStep_StepLabel() As
String

   cStep_StepLabel = mcStep.StepLabel

End Property


Private Property Let
cStep_StepLevel(ByVal RHS As Integer)

   mcStep.StepLevel = RHS

End Property

Private Property Get cStep_StepLevel() As
Integer

   cStep_StepLevel = mcStep.StepLevel

End Property

Private Property Let cStep_StepText(ByVal
RHS As String)

   ' Since the manager step does not take any
action, the step
   ' text and file name will always be empty
   mcStep.StepText = gstrEmptyString

End Property

Private Property Get cStep_StepText() As
String

   cStep_StepText = mcStep.StepText

End Property

Private Property Let
cStep_StepTextFile(ByVal RHS As String)

   ' Since the manager step does not take any
action, the step
   ' text and file name will always be empty
   mcStep.StepTextFile = gstrEmptyString

End Property

Private Property Get cStep_StepTextFile() As
String

   cStep_StepTextFile = mcStep.StepTextFile

End Property

Private Property Let cStep_StepType(RHS As
gintStepType)

   mcStep.StepType = gintManagerStep

End Property

Private Property Get cStep_StepType() As
gintStepType

   cStep_StepType = mcStep.StepType

End Property

Private Sub cStep_Validate()
   ' The validate routines for each of the steps
will
   ' carry out the specific validations for the
type and
   ' call the generic validation routine

   On Error GoTo cStep_ValidateErr
   mstrSource = mstrModuleName &
"cStep_Validate"

   ' Validations specific to manager steps

   ' Check if the step text or a file name has
been
   ' specified
   If Not StringEmpty(mcStep.StepText) Or
Not StringEmpty(mcStep.StepTextFile) Then
      ShowError
errTextAndFileNullForManager
      On Error GoTo 0
      Err.Raise vbObjectError +
errValidateFailed, _
            gstrSource, _
            LoadResString(errValidateFailed)
   End If

   If mcStep.ExecutionMechanism <>
gintNoOption Then
      ShowError
errExecutionMechanismInvalid
      On Error GoTo 0
      Err.Raise vbObjectError +
errValidateFailed, _
            gstrSource, _
            LoadResString(errValidateFailed)
   End If

   If mcStep.FailureDetails <>
gstrEmptyString Then
      ShowError errFailureDetailsNullForMgr
      On Error GoTo 0
      Err.Raise vbObjectError +
errValidateFailed, _
            gstrSource, _
            LoadResString(errValidateFailed)
   End If

   If mcStep.ContinuationCriteria <>
gintNoOption Then
      ShowError errContCriteriaInvalid
      On Error GoTo 0
      Err.Raise vbObjectError +
errValidateFailed, _
            gstrSource, _
            LoadResString(errValidateFailed)
   End If

   mcStep.Validate

   Exit Sub

cStep_ValidateErr:
   LogErrors Errors
   mstrSource = mstrModuleName &
"cStep_Validate"
   On Error GoTo 0
   Err.Raise vbObjectError + errValidateFailed, _
         mstrSource, _
         LoadResString(errValidateFailed)
End Sub

Private Property Let cStep_VersionNo(ByVal RHS
As String)

   mcStep.VersionNo = RHS

End Property

Private Property Get cStep_VersionNo() As String

   cStep_VersionNo = mcStep.VersionNo

End Property

Private Property Let cStep_WorkspaceId(ByVal
RHS As Long)

   mcStep.WorkspaceId = RHS

End Property

Private Property Get cStep_WorkspaceId() As Long

   cStep_WorkspaceId = mcStep.WorkspaceId

End Property
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cNode"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:       cNode.cls
'             Microsoft TPC-H Kit Ver. 1.00
'             Copyright Microsoft, 1999
'             All Rights Reserved
'
'
'  PURPOSE:    Defines the properties that an
object has to implement.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

Public Property Get IndOperation() As Operation
End Property
Public Property Let IndOperation(ByVal vdata As
Operation)
End Property
Public Sub Validate()
End Sub
Public Property Get Value() As String
End Property
Public Property Let Value(ByVal vdata As String)
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
End Property

Public Property Get NodeDB() As Database
End Property
Public Property Set NodeDB(vdata As
Database)
End Property

Public Property Get Position() As Long
End Property
Public Property Let Position(ByVal vdata
As Long)
End Property

VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cNodeCollections"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:      cNodeCollections.cls
'             Microsoft TPC-H Kit Ver. 1.00
'             Copyright Microsoft, 1999
'             All Rights Reserved
'
'
'  PURPOSE:    Implements an array of
objects.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

' Node counter
Private mlngNodeCount As Long
Private mdbsNodeDb As Database
Private mcarrNodes() As Object

' Used to indicate the source module name
when errors
' are raised by this class
Private mstrSource As String
Private Const mstrModuleName As String =
"cNodeCollections."

Public Property Set Item(ByVal Position As
Long, _
      ByVal objNode As Object)

   ' Returns the element at the passed in
position in the array
   If Position >= 0 And Position <
mlngNodeCount Then
      Set mcarrNodes(Position) = objNode
   Else
      On Error GoTo 0
      Err.Raise vbObjectError +
errItemDoesNotExist, mstrSource, _

LoadResString(errItemDoesNotExist)
   End If

End Property
Public Property Get Item(ByVal Position As
Long) As Object
Attribute Item.VB_UserMemId = 0

   ' Returns the element at the passed in
position in the array
   If Position >= 0 And Position <
mlngNodeCount Then
      Set Item = mcarrNodes(Position)
   Else
      On Error GoTo 0
      Err.Raise vbObjectError +
errItemDoesNotExist, mstrSource, _
         LoadResString(errItemDoesNotExist)
   End If

End Property

Public Sub Commit(ByVal cSaveObj As
Object, _
      ByVal lngIndex As Long)
   ' This procedure checks if any changes have
been made to the
   ' passed in object. If so, it calls the
corresponding method
   ' to commit the changes.

   On Error GoTo CommitErr
   mstrSource = mstrModuleName &
"Commit"

   Select Case cSaveObj.IndOperation
      Case QueryOp
         ' No changes were made to the queried
parameter.
         ' Do nothing

      Case InsertOp
         cSaveObj.Add
         cSaveObj.IndOperation = QueryOp

      Case UpdateOp
         cSaveObj.Modify
         cSaveObj.IndOperation = QueryOp

      Case DeleteOp
         cSaveObj.Delete
         ' Now we can remove the record from
the array
         Call Unload(lngIndex)

   End Select

   Exit Sub

CommitErr:
   LogErrors Errors
   mstrSource = mstrModuleName &
"Commit"
   On Error GoTo 0
   Err.Raise vbObjectError + errCommitFailed,
_
         mstrSource, _
         LoadResString(errCommitFailed)

End Sub
Public Sub Save(ByVal lngWorkspace As
Long)
   ' Calls a procedure to commit all changes for
the passed
   ' in workspace.

   Dim lngIndex As Long

   On Error GoTo SaveErr

   ' Find all parameters in the array with a
matching workspace id
   ' It is important to step backwards through
the array, since
   ' we delete parameter records as we go
along!
   For lngIndex = mlngNodeCount - 1 To 0
Step -1
      If mcarrNodes(lngIndex).WorkspaceId =
lngWorkspace Then

         ' Call a procedure to commit all changes to
the
         ' parameter record, if any
         Call Commit(mcarrNodes(lngIndex),
lngIndex)

      End If
   Next lngIndex

   Exit Sub

SaveErr:
   LogErrors Errors
   mstrSource = mstrModuleName & "Save"
   On Error GoTo 0
   Err.Raise vbObjectError + errSaveFailed, _
         mstrSource, _
         LoadResString(errSaveFailed)

End Sub
Public Property Get Count() As Long

   Count = mlngNodeCount

End Property

Public Property Get NodeDB() As Database

   Set NodeDB = mdbsNodeDb

End Property
Public Property Set NodeDB(vdata As Database)

   Set mdbsNodeDb = vdata

End Property

Public Sub Load(cNodeToLoad As Object)
   ' Adds the passed in object to the array

   On Error GoTo LoadErr

   ' If this procedure is called by the add to array
procedure,
   ' the database object has already been initialized
   If cNodeToLoad.NodeDB Is Nothing Then

      ' All the Nodes will be initialized with the
database
      ' objects before being added to the array
      Set cNodeToLoad.NodeDB = mdbsNodeDb

   End If

   ReDim Preserve mcarrNodes(mlngNodeCount)

   ' Set the newly added element in the array to the
passed in Node
   cNodeToLoad.Position = mlngNodeCount
   Set mcarrNodes(mlngNodeCount) =
cNodeToLoad

   mlngNodeCount = mlngNodeCount + 1

   Exit Sub

LoadErr:
   LogErrors Errors
   On Error GoTo 0
   Err.Raise vbObjectError + errLoadFailed,
mstrModuleName & "Load", _
         LoadResString(errLoadFailed)
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
End Sub
Public Sub Unload(lngDeletePosition As
Long)
   ' Unloads the passed in object from the
array

   On Error GoTo UnloadErr

   If lngDeletePosition < (mlngNodeCount -
1) Then

      ' Set the Node at the position being
deleted to
      ' the last Node in the Node array
      Set mcarrNodes(lngDeletePosition) =
mcarrNodes(mlngNodeCount - 1)

mcarrNodes(lngDeletePosition).Position =
lngDeletePosition
   End If

   ' Delete the last Node from the array
   mlngNodeCount = mlngNodeCount - 1
   If mlngNodeCount > 0 Then
      ReDim Preserve mcarrNodes(0 To
mlngNodeCount - 1)
   Else
      ReDim mcarrNodes(0)
   End If

   Exit Sub

UnloadErr:
   LogErrors Errors
   mstrSource = mstrModuleName &
"Unload"
   On Error GoTo 0
   Err.Raise vbObjectError +
errUnloadFailed, _
         mstrSource, _
         LoadResString(errUnloadFailed)

End Sub
Public Sub Delete(lngDeletePosition As
Long)
   ' Deletes the object at the specified
position in the
   ' array

   Dim cDeleteObj As Object

   On Error GoTo DeleteErr
   mstrSource = mstrModuleName &
"Delete"

   Set cDeleteObj =
mcarrNodes(lngDeletePosition)

   If cDeleteObj.IndOperation = InsertOp
Then
      ' If we are deleting a record that has just
been inserted,
      ' blow it away
      Call Unload(lngDeletePosition)
   Else
      ' Set the operation for the deleted object
to indicate a
      ' delete - we actually delete the element
only at the time
      ' of a save operation
      cDeleteObj.IndOperation = DeleteOp
   End If

   Exit Sub

DeleteErr:
   LogErrors Errors
   mstrSource = mstrModuleName & "Delete"
   On Error GoTo 0
   Err.Raise vbObjectError + errDeleteFailed, _
         mstrSource, _
         LoadResString(errDeleteFailed)

End Sub
Public Sub Modify(cModifiedNode As Object)
   ' Sets the object at the passed in position to
the
   ' modified object passed in

   On Error GoTo ModifyErr

   ' First check if the record is valid - all objects
that
   ' use this collection class must have a
Validate routine
   cModifiedNode.Validate

   ' If we are updating a record that hasn't yet
been inserted,
   ' do not change the operation indicator - or
we try to update
   ' a non-existant record
   If cModifiedNode.IndOperation <> InsertOp
Then
      ' Set the operations to indicate an update
      cModifiedNode.IndOperation =
UpdateOp
   End If

   ' Modify the object at the queried position -
the Position
   ' will be maintained by this class
   Set mcarrNodes(cModifiedNode.Position) =
cModifiedNode

   Exit Sub

ModifyErr:
   LogErrors Errors
   mstrSource = mstrModuleName & "Modify"
   On Error GoTo 0
   Err.Raise vbObjectError + errModifyFailed,
_
         mstrSource, _
         LoadResString(errModifyFailed)

End Sub
Public Sub Add(cNodeToAdd As Object)

   On Error GoTo AddErr

   Set cNodeToAdd.NodeDB = mdbsNodeDb

   ' First check if the record is valid
   cNodeToAdd.Validate

   ' Set the operation to indicate an insert
   cNodeToAdd.IndOperation = InsertOp

   ' Call a procedure to load the record in the
array
   Call Load(cNodeToAdd)

   Exit Sub

AddErr:
   LogErrors Errors
   mstrSource = mstrModuleName & "Add"
   On Error GoTo 0
   Err.Raise vbObjectError + errAddFailed, _
         mstrSource, _
         LoadResString(errAddFailed)

End Sub

Private Sub Class_Terminate()

   ReDim mcarrNodes(0)
   mlngNodeCount = 0

End Sub


Attribute VB_Name = "Common"
'  FILE:      Common.bas
'             Microsoft TPC-H Kit Ver. 1.00
'             Copyright Microsoft, 1999
'             All Rights Reserved
'
'
'  PURPOSE:   Module containing common
functionality throughout
'             StepMaster
'  Contact:   Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

Private Const mstrModuleName As String =
"Common."

' Used to separate the variable data from the
constant error
' message being raised when a context-sensitive
error is displayed
Private Const mintDelimiter As String = " : "
Private Const mstrFormatString = "mmddyy"

' Identifiers for the different labels that need to be
loaded
' into the tree view for each workspace
Public Const mstrWorkspacePrefix = "W"
Public Const mstrParameterPrefix = "P"
Public Const mstrParamConnectionPrefix = "C"
Public Const mstrConnectionDtlPrefix = "N"
Public Const mstrParamExtensionPrefix = "E"
Public Const mstrParamBuiltInPrefix = "B"
Public Const gstrGlobalStepPrefix = "G"
Public Const gstrManagerStepPrefix = "M"
Public Const gstrWorkerStepPrefix = "S"
Public Const gstrDummyPrefix = "D"
Public Const mstrLabelPrefix = "L"
Public Const mstrInstancePrefix = "I"
Public Function LabelStep(lngWorkspaceIdentifier
As Long) As String
   ' Returns the step label for the workspace
identifier passed in
   ' Basically this is a wrapper around the
MakeKeyValid function

   LabelStep = MakeKeyValid(gintStepLabel,
gintStepLabel, lngWorkspaceIdentifier)

End Function


Public Function JulianDateToString(dt64Bit As
Currency) As String

   Dim lYear As Long
   Dim lMonth As Long
   Dim lDay As Long
   Dim lHour As Long
   Dim lMin As Long
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   Dim lSec As Long
   Dim lMs As Long

   Call JulianToTime(dt64Bit, lYear,
lMonth, lDay, lHour, lMin, lSec, lMs)
   JulianDateToString = Format$(lYear,
gsYearFormat) & gsDateSeparator & _
         Format$(lMonth, gsDtFormat) &
gsDateSeparator & _
         Format$(lDay, gsDtFormat) &
gstrBlank & _
         Format$(lHour, gsTmFormat) &
gsTimeSeparator & _
         Format$(lMin, gsTmFormat) &
gsTimeSeparator & _
         Format$(lSec, gsTmFormat) &
gsMsSeparator & _
         Format$(lMs, gsMSecondFormat)

End Function
Public Sub DeleteFile(strFile As String,
Optional ByVal bCheckIfEmpty As Boolean
= False)

   ' Ensure that there is only a single file of
the name before delete, since
   ' Kill supports wildcards and can
potentially delete a number of files
   Dim strTemp As String

   If CheckFileExists(strFile) Then
      If bCheckIfEmpty Then
         If FileLen(strFile) = 0 Then
            Kill strFile
         End If
      Else
         Kill strFile
      End If
   End If

End Sub
Public Function CheckFileExists(strFile As
String) As Boolean

   ' Returns true if the passed in file exists
   ' Raises an error if multiple files are found
(filename contains a wildcard)
   CheckFileExists = False

   If Not StringEmpty(Dir(strFile)) Then
      If Not StringEmpty(Dir()) Then
         On Error GoTo 0
         Err.Raise vbObjectError +
errDeleteSingleFile, _
            mstrModuleName & "DeleteFile",
LoadResString(errDeleteSingleFile)
      End If

      CheckFileExists = True
   End If

End Function

Public Function GetVersionString() As
String
GetVersionString = "Version " & gsVersion
End Function

Function IsLabel(strKey As String) As
Boolean

   ' The tree view control on frmMain can
contain two types of
   ' nodes -
   ' 1. Nodes that contain data for the
workspace - this could
   ' be data for the different types of steps or
parameters
   ' 2. Nodes that display static data - these kind
of nodes
   ' are referred to as label nodes e.g. "Global
Steps" is a
   ' label node
   ' This function returns True if the passed in
key corresponds
   ' to a label node

   IsLabel = InStr(strKey, mstrLabelPrefix) > 0

End Function

Function MakeKeyValid(lngIdentifier As
Long, _
   intTypeOfNode As Integer, _
   Optional ByVal WorkspaceId As Long = 0,
_
   Optional ByVal InstanceId As Long = 0) As
String

   ' We use a numbering scheme while loading
the tree view with
   ' all node data, since it needs a unique key
and we want to
   ' use the key to identify the data it contains.
   ' Moreover, add a character to the beginning
of the identifier
   ' so that the tree view control accepts it as a
valid string,
   ' viz. "456" doesn't work, so change it to
"W456"
   ' The general scheme is to concatenate a
Label with the Identifier
   ' e.g A Global Step Node will have the
Label, G and the Step Id
   ' concatenated to form the unique key
   ' The list of all such node types is given
below
   ' 1. "W" + Workspace_Id for Workspace
nodes
   ' 2. "P" + Parameter_Id for Parameter nodes
   ' 3. "M" + Step_Id for Manager Step nodes
   ' 4. "S" + Step_Id for Worker Step nodes
   ' 5. "G" + Step_Id for Global Step nodes
   ' 6. Instance_id + "I" + Step_Id for Instance
nodes
   ' 7. Workspace_id + "L" + the label
identifier = node type for all Label nodes
   ' Since the manager, worker and global steps
are stored in the
   ' same table and the step identifiers will
always be unique, we
   ' can use the same character as the prefix, but
this is a
   ' convenient way to know the type of step
being processed.
   ' The workspace id is appended to the label
identifier to make
   ' it unique, since multiple workspaces may
be open during a session
   ' Strip the prefix characters off while saving
the Ids to the db

   Dim strPrefixChar As String

   On Error GoTo MakeKeyValidErr
   gstrSource = mstrModuleName &
"MakeKeyValid"

   Select Case intTypeOfNode
      Case gintWorkspace
         strPrefixChar = mstrWorkspacePrefix
      Case gintGlobalStep
         strPrefixChar = gstrGlobalStepPrefix
      Case gintManagerStep
         strPrefixChar = gstrManagerStepPrefix
      Case gintWorkerStep
         strPrefixChar = gstrWorkerStepPrefix
      Case gintRunManager, gintRunWorker
         If InstanceId = 0 Then
            On Error GoTo 0
            Err.Raise vbObjectError +
errMandatoryParameterMissing, _
               gstrSource, _

LoadResString(errMandatoryParameterMissing)
         End If
         ' Concatenate the instance identifier and the
step
         ' identifier to form a unique key
         strPrefixChar = Trim$(Str$(InstanceId)) &
mstrInstancePrefix
      Case gintParameter
         strPrefixChar = mstrParameterPrefix
      Case gintNodeParamConnection
         strPrefixChar = mstrParamConnectionPrefix
      Case gintConnectionDtl
         strPrefixChar = mstrConnectionDtlPrefix
      Case gintNodeParamExtension
         strPrefixChar = mstrParamExtensionPrefix
      Case gintNodeParamBuiltIn
         strPrefixChar = mstrParamBuiltInPrefix
      Case gintGlobalsLabel, gintParameterLabel,
gintParamConnectionLabel, _
            gintConnDtlLabel, _
            gintParamExtensionLabel,
gintParamBuiltInLabel, gintGlobalStepLabel, _
            gintStepLabel
         If WorkspaceId = 0 Then
            ' The Workspace Id has to be specified for
a label node
            ' Otherwise it will not be possible to
generate unique label
            ' identifiers if multiple workspaces are
open
            On Error GoTo 0
            Err.Raise vbObjectError +
errWorkspaceIdMandatory, _
               gstrSource, _

LoadResString(errWorkspaceIdMandatory)
         End If
         ' For all labels, the workspace identifier and
the
         ' label prefix are concatenated to form the
key
         strPrefixChar = Trim$(Str$(WorkspaceId))
& mstrLabelPrefix
      Case Else
            On Error GoTo 0
            Err.Raise vbObjectError +
errInvalidNodeType, _
               gstrSource, _
               LoadResString(errInvalidNodeType)
   End Select

   MakeKeyValid = strPrefixChar &
Trim$(Str$(lngIdentifier))

   Exit Function

MakeKeyValidErr:
   Call LogErrors(Errors)
   On Error GoTo 0
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   Err.Raise vbObjectError +
errMakeKeyValidFailed, _
         gstrSource, _

LoadResString(errMakeKeyValidFailed)

End Function

Function MakeIdentifierValid(strKey As
String) As Long

   ' Returns the Identifier corresponding to
the passed in key
   ' (Reverse of what was done in
MakeKeyValid)

   On Error GoTo MakeIdentifierValidErr

   If IsLabel(strKey) Then
      ' If the key corresponds to a label node,
the identifier
      ' appears to the right of the label prefix
      MakeIdentifierValid = Val(Mid(strKey,
InStr(strKey, mstrLabelPrefix) + 1))
   ElseIf InStr(strKey, mstrInstancePrefix) =
0 Then
      ' For all other nodes, stripping the first
character off
      ' returns a valid Id
      MakeIdentifierValid = Val(Mid(strKey,
2))
   Else
      ' Instance node - strip of all characters
till the
      ' instance prefix
      MakeIdentifierValid = Val(Mid(strKey,
InStr(strKey, mstrInstancePrefix) + 1))
   End If

   Exit Function

MakeIdentifierValidErr:
   Call LogErrors(Errors)
   On Error GoTo 0
   Err.Raise vbObjectError +
errMakeIdentifierValidFailed, _
         mstrModuleName &
"MakeIdentifierValid", _

LoadResString(errMakeIdentifierValidFaile
d)

End Function
Public Function IsInstanceNode(strNodeKey
As String) As Boolean

   ' Returns true if the passed in node key
corresponds to a step instance
   IsInstanceNode = InStr(strNodeKey,
mstrInstancePrefix) > 0

End Function
Public Function IsBuiltInLabel(strNodeKey
As String) As Boolean

   ' Returns true if the passed in node key
corresponds to a step instance
   IsBuiltInLabel = (IsLabel(strNodeKey)
And _
      (MakeIdentifierValid(strNodeKey) =
gintParamBuiltInLabel))

End Function

Public Sub ShowBusy()
   ' Modifies the mousepointer to indicate that
the
   ' application is busy

   On Error Resume Next

   Screen.MousePointer = vbHourglass

End Sub
Public Sub ShowFree()
   ' Modifies the mousepointer to indicate that
the
   ' application has finished processing and is
ready
   ' to accept user input

   On Error Resume Next

   Screen.MousePointer = vbDefault

End Sub

Public Function InstrR(strMain As String, _
      strSearch As String) As Integer
   ' Finds the last occurrence of the passed in
string

   Dim intPos As Integer
   Dim intPrev As Integer

   On Error GoTo InstrRErr

   intPrev = intPos
   intPos = InStr(1, strMain, strSearch)

   Do While intPos > 0
      intPrev = intPos
      intPos = InStr(intPos + 1, strMain,
strSearch)
   Loop
   InstrR = intPrev

   Exit Function

InstrRErr:
   Call LogErrors(Errors)
   gstrSource = mstrModuleName & "InstrR"
   On Error GoTo 0
   Err.Raise vbObjectError + errInstrRFailed, _
      gstrSource, _
      LoadResString(errInstrRFailed)

End Function

Public Function GetDefaultDir(lWspId As
Long, WspParameters As cArrParameters) As
String

   Dim sDir As String
   sDir = SubstituteParameters( _
         gstrEnvVarSeparator &
PARAM_DEFAULT_DIR &
gstrEnvVarSeparator, _
         lWspId,
WspParameters:=WspParameters)
   MakePathValid (sDir & gstrFileSeparator &
"a.txt")
   GetDefaultDir = GetShortName(sDir)
   If StringEmpty(GetDefaultDir) Then
      GetDefaultDir = App.Path
   End If

End Function

Public Sub AddArrayElement(ByRef arrNodes() As
Object, _
      ByVal objToAdd As Object, _
      ByRef lngCount As Long)
   ' Adds the passed in object to the array

   On Error GoTo AddArrayElementErr

   ' Increase the array dimension and add the object
to it
   ReDim Preserve arrNodes(lngCount)
   Set arrNodes(lngCount) = objToAdd
   lngCount = lngCount + 1

   Exit Sub

AddArrayElementErr:
   LogErrors Errors
   gstrSource = mstrModuleName &
"AddArrayElement"
   On Error GoTo 0
   Err.Raise vbObjectError +
errAddArrayElementFailed, _
         gstrSource, _
         LoadResString(errAddArrayElementFailed)

End Sub


Public Function CheckForNullField(rstRecords As
Recordset, strFieldName As String) As String

   ' Returns an empty string if a given field is null
   On Error GoTo CheckForNullFieldErr

   If IsNull(rstRecords.Fields(strFieldName)) Then
      CheckForNullField = gstrEmptyString
   Else
      CheckForNullField =
rstRecords.Fields(strFieldName)
   End If
   Exit Function

CheckForNullFieldErr:
   Call LogErrors(Errors)
   On Error GoTo 0
   Err.Raise vbObjectError +
errCheckForNullFieldFailed, _
         mstrModuleName & "CheckForNullField",
_

LoadResString(errCheckForNullFieldFailed)

End Function

Public Function ErrorOnNullField(rstRecords As
Recordset, strFieldName As String) As Variant

   ' If a given field is null, raises an error
   ' Else, returns the field value in a variant
   ' The calling function must convert the return
value to the
   ' appropriate type
   On Error GoTo ErrorOnNullFieldErr
   gstrSource = mstrModuleName &
"ErrorOnNullField"

   If IsNull(rstRecords.Fields(strFieldName)) Then
      On Error GoTo 0
      Err.Raise vbObjectError +
errMandatoryFieldNull, _
         gstrSource, _
         strFieldName & mintDelimiter &
LoadResString(errMandatoryFieldNull)
   Else
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
      ErrorOnNullField =
rstRecords.Fields(strFieldName)
   End If
   Exit Function

ErrorOnNullFieldErr:
   Call LogErrors(Errors)
   On Error GoTo 0
   Err.Raise vbObjectError +
errUnableToCheckNull, _
      gstrSource, _
      strFieldName & mintDelimiter &
LoadResString(errUnableToCheckNull)

End Function
Public Function
StringEmpty(strCheckString As String) As
Boolean

   StringEmpty = (strCheckString =
gstrEmptyString)

End Function
Public Function
GetIteratorValue(cStepIterators As
cRunColIt, _
      ByVal strItName As String)

   Dim lngIndex As Long
   Dim strValue As String

   On Error GoTo GetIteratorValueErr
   gstrSource = mstrModuleName &
"GetIteratorValue"

   ' Find the iterator in the Iterators
collection
   For lngIndex = 0 To cStepIterators.Count
- 1
      If
cStepIterators(lngIndex).IteratorName =
strItName Then
         strValue =
cStepIterators(lngIndex).Value
         Exit For
      End If
   Next lngIndex

   If lngIndex > cStepIterators.Count - 1
Then
      ' The iterator has not been defined for
the branch
      ' Raise an error
      On Error GoTo 0
      Err.Raise vbObjectError +
errParamNameInvalid, _
            gstrSource, _

LoadResString(errParamNameInvalid)
   End If

   GetIteratorValue = strValue
   Exit Function

GetIteratorValueErr:
   ' Log the error code raised by Visual
Basic
   Call LogErrors(Errors)
   gstrSource = mstrModuleName &
"GetIteratorValue"
   On Error GoTo 0
   Err.Raise vbObjectError +
errGetParamValueFailed, _
         gstrSource, _

LoadResString(errGetParamValueFailed)

End Function
Public Function SubstituteParameters(ByVal
strComString As String, _
      ByVal lngWorkspaceId As Long, _
      Optional StepIterators As cRunColIt =
Nothing, _
      Optional WspParameters As
cArrParameters = Nothing) As String
   ' This function substitutes all parameter
names and
   ' environment variables in the passed in
string with
   ' their values. It also substitutes the value for
the
   ' iterators, if any.
   ' Since the syntax is to enclose parameter
names and
   ' environment variables in "%", we check if
a given
   ' variable is a parameter - if so, we substitute
the
   ' parameter value - else we try to get the
value from
   ' the environment

   Dim intPos As Integer
   Dim intEndPos As Integer
   Dim strEnvVariable As String
   Dim strValue As String
   Dim strCommand As String
   Dim cTempStr As cStringSM

   ' Initialize the return value of the function to
the
   ' passed in command
   strCommand = strComString

   If WspParameters Is Nothing Then Set
WspParameters = gcParameters

   Set cTempStr = New cStringSM

   intPos = InStr(strCommand,
gstrEnvVarSeparator)
   Do While intPos <> 0
      If Mid(strCommand, intPos + 1, 1) =
gstrEnvVarSeparator Then
         ' Wildcard character - to be substituted
by a single % - later!
         intPos = intPos + 2
         If intPos > Len(strCommand) Then
Exit Do
      Else
         ' Extract the environment variable from
the passed
         ' in string
         intEndPos = InStr(intPos + 1,
strCommand, gstrEnvVarSeparator)

         If intEndPos > 0 Then
            strEnvVariable = Mid(strCommand,
intPos + 1, intEndPos - intPos - 1)
         Else
            On Error GoTo 0
            Err.Raise vbObjectError +
errParamSeparatorMissing, _
                  gstrSource, _

LoadResString(errParamSeparatorMissing)
         End If
         strValue = gstrEmptyString
         ' Get the value of the variable and call a
function
         ' to replace the variable with it's value
         strValue = GetValue(strEnvVariable,
lngWorkspaceId, StepIterators, WspParameters)
         ' The function raises an error if the variable
is
         ' not found
         strCommand =
cTempStr.ReplaceSubString(strCommand, _
               gstrEnvVarSeparator & strEnvVariable
& gstrEnvVarSeparator, _
               strValue)
      End If

      intPos = InStr(intPos, strCommand,
gstrEnvVarSeparator)
   Loop

   strCommand =
cTempStr.ReplaceSubString(strCommand, _
         gstrEnvVarSeparator &
gstrEnvVarSeparator, gstrEnvVarSeparator)

   Set cTempStr = Nothing
   SubstituteParameters = strCommand

End Function
Private Function GetValue(ByVal strParameter As
String, _
      ByVal lngWorkspaceId As Long, _
      cStepIterators As cRunColIt, _
      WspParameters As cArrParameters) As String
   ' This function returns the value for the passed in
   ' parameter - it may be a workspace parameter, an
   ' environment variable or an iterator

   Dim intPos As Integer
   Dim intEndPos As Integer
   Dim strVariable As String
   Dim strValue As String
   Dim cParamRec As cParameter

   On Error GoTo GetValueErr

   ' Initialize the return value of the function to the
   ' empty
   strValue = gstrEmptyString

   intPos = InStr(strParameter,
gstrEnvVarSeparator)
   If intPos > 0 Then
      ' Extract the variable from the passed in string
      intEndPos = InStr(intPos + 1, strParameter,
gstrEnvVarSeparator)
      If intEndPos = 0 Then
         intEndPos = Len(strParameter)
      End If

      strVariable = Mid(strParameter, intPos + 1,
intEndPos - intPos - 1)
   Else
      ' The separator charactor has not been passed
in -
      ' try to find the value of the passed in
parameter
      strVariable = strParameter
   End If

   If Not StringEmpty(strVariable) Then
      ' Check if this is the timestamp parameter first
      If strVariable = gstrTimeStamp Then
         strValue = Format$(Now, mstrFormatString,
_
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
                vbUseSystemDayOfWeek,
vbUseSystem)
        Else
          ' Try to find a parameter for the
workspace with
          ' the same name
          Set cParamRec =
WspParameters.GetParameterValue(lngWor
kspaceId, _
                strVariable)
          If cParamRec Is Nothing Then
            If Not cStepIterators Is Nothing
Then
                ' If the string is not a parameter,
then check
                ' if it is an iterator
                strValue =
GetIteratorValue(cStepIterators, strVariable)
            End If

            If StringEmpty(strValue) Then
                ' Neither - Check if it is an
environment variable
                strValue =
Environ$(strVariable)
                If StringEmpty(strValue) Then
                    On Error GoTo 0
                    WriteError
errSubValuesFailed, _
                        OptArgs:="Invalid
parameter: " & gstrSQ & strVariable &
gstrSQ
                    Err.Raise vbObjectError +
errSubValuesFailed, _
                        mstrModuleName &
"GetValue", _

LoadResString(errSubValuesFailed) &
"Invalid parameter: " & gstrSQ &
strVariable & gstrSQ
                End If
            End If
          Else
            strValue =
cParamRec.ParameterValue
          End If
        End If
    End If

    GetValue = strValue

    Exit Function

GetValueErr:
    If Err.Number = vbObjectError +
errParamNameInvalid Then
        ' If the parameter has not been defined
for the
        ' workspace then check if it is an
environment
        ' variable
        Resume Next
    End If

    ' Log the error code raised by Visual
Basic
    Call LogErrors(Errors)
    gstrSource = mstrModuleName &
"GetValue"
    WriteError errSubValuesFailed,
gstrSource, "Parameter: " & gstrSQ &
strVariable & gstrSQ
    On Error GoTo 0
    Err.Raise vbObjectError +
errSubValuesFailed, _
        gstrSource, _
        LoadResString(errSubValuesFailed) &
"Parameter: " & gstrSQ & strVariable &
gstrSQ

End Function
Public Function SQLFixup(strField As String)
As String
    ' Returns a string that can be executed by
SQL Server

    Dim cMyStr As New cStringSM
    Dim strTemp As String

    On Error GoTo SQLFixupErr

    strTemp = strField
    SQLFixup = strTemp

    ' Single-quotes have to be replaced by two
single-quotes,
    ' since a single-quote is the identifier
delimiter
    ' character - call a procedure to do the
replace
'    SQLFixup =
cMyStr.ReplaceSubString(strTemp, gstrDQ,
"\" & gstrDQ)

    ' Replace pipe characters with the
corresponding chr function
    ' SQLFixup =
cMyStr.ReplaceSubString(strTemp, gstrDQ,
gstrDQ & gstrDQ)

    Exit Function

SQLFixupErr:
    gstrSource = mstrModuleName &
"SQLFixup"
    LogErrors Errors
    On Error GoTo 0
    Err.Raise vbObjectError +
errMakeFieldValidFailed, _
        gstrSource,
LoadResString(errMakeFieldValidFailed)

End Function
Public Function TranslateStepLabel(sLabel As
String) As String
    ' Translates the passed in step label to a valid
file name
    ' All characters in the label that are invalid
for filenames (viz. \ / : * ? " < > |)
    ' and spaces are substituted with underscores
- also ensure that the resulting filename
    ' is not greater than 255 characters
    Dim cTempStr As New cStringSM
    TranslateStepLabel =
cTempStr.ReplaceSubString(sLabel,
gstrFileSeparator, "_")
    TranslateStepLabel =
cTempStr.ReplaceSubString(TranslateStepLab
el, "/", gstrUnderscore)
    TranslateStepLabel =
cTempStr.ReplaceSubString(TranslateStepLab
el, ":", gstrUnderscore)
    TranslateStepLabel =
cTempStr.ReplaceSubString(TranslateStepLab
el, "*", gstrUnderscore)
    TranslateStepLabel =
cTempStr.ReplaceSubString(TranslateStepLab
el, "?", gstrUnderscore)
    TranslateStepLabel =
cTempStr.ReplaceSubString(TranslateStepLabel,
gstrDQ, gstrUnderscore)
    TranslateStepLabel =
cTempStr.ReplaceSubString(TranslateStepLabel,
"<", gstrUnderscore)
    TranslateStepLabel =
cTempStr.ReplaceSubString(TranslateStepLabel,
">", gstrUnderscore)
    TranslateStepLabel =
cTempStr.ReplaceSubString(TranslateStepLabel,
"|", gstrUnderscore)
    TranslateStepLabel =
cTempStr.ReplaceSubString(TranslateStepLabel,
gstrBlank, gstrUnderscore)

    ' Commas are substituted with underscores since
the command shell uses a comma to
    ' delimit commands
    TranslateStepLabel =
cTempStr.ReplaceSubString(TranslateStepLabel,
",", gstrUnderscore)

    If Len(TranslateStepLabel) > MAX_PATH Then
        TranslateStepLabel = Mid(TranslateStepLabel,
1, MAX_PATH)
    End If

End Function

Public Function TypeOfObject(ByVal objNode As
Object) As Integer
    ' Determines the type of object that is passed in

    On Error GoTo TypeOfObjectErr
    gstrSource = mstrModuleName &
"TypeOfObject"

    Select Case TypeName(objNode)
        Case "cWorkspace"
            TypeOfObject = gintWorkspace

        Case "cParameter"
            TypeOfObject = gintParameter

        Case "cConnection"
            TypeOfObject = gintParameterConnect

        Case "cConnDtl"
            TypeOfObject = gintConnectionDtl

        Case "cGlobalStep"
            TypeOfObject = gintGlobalStep

        Case "cManager"
            TypeOfObject = gintManagerStep

        Case "cWorker"
            TypeOfObject = gintWorkerStep

        Case "cStep"
            ' If a step record is passed in, call a function
            ' to determine the type of step
            TypeOfObject =
TypeOfStep(StepClass:=objNode)

        Case Else
            WriteError errTypeOfObjectFailed,
gstrSource, _
                TypeName(objNode)
            On Error GoTo 0
            Err.Raise vbObjectError +
errTypeOfObjectFailed, _
                gstrSource, _
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
LoadResString(errTypeOfObjectFailed)
   End Select

   Exit Function

TypeOfObjectErr:
   ' Log the error code raised by Visual
Basic
   Call LogErrors(Errors)
   On Error GoTo 0
   Err.Raise vbObjectError +
errTypeOfObjectFailed, _
          gstrSource, _

LoadResString(errTypeOfObjectFailed)

End Function
Attribute VB_Name = "ConnDtlCommon"
'  FILE:      ConnDtlCommon.bas
'            Microsoft TPC-H Kit Ver. 1.00
'            Copyright Microsoft, 1999
'            All Rights Reserved
'
'
'  PURPOSE:    Contains functionality
common across StepMaster and
'            SMRunOnly, pertaining to
connections
'            Specifically, functions to load
connections in an array
'            and so on.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

' Used to indicate the source module name
when errors
' are raised by this module
Private Const mstrModuleName As String =
"ConnDtlCommon."

Public Sub
LoadRSInConnDtlArray(rstConns As
Recordset, cConns As cConnDtls)

   Dim cNewConnDtl As cConnDtl

   On Error GoTo
LoadRSInConnDtlArrayErr

   If rstConns.RecordCount = 0 Then
      Exit Sub
   End If

   rstConns.MoveFirst
   While Not rstConns.EOF

      Set cNewConnDtl = New cConnDtl

      ' Initialize ConnDtl values
      ' Call a procedure to raise an error if
mandatory fields are null.
      cNewConnDtl.ConnNameId =
ErrorOnNullField(rstConns,
FLD_ID_CONN_NAME)
      cNewConnDtl.WorkspaceId =
ErrorOnNullField(rstConns,
FLD_ID_WORKSPACE)
      cNewConnDtl.ConnName =
CStr(ErrorOnNullField(rstConns,
FLD_CONN_DTL_CONNECTION_NAM
E))
      cNewConnDtl.ConnectionString =
CheckForNullField(rstConns,
FLD_CONN_DTL_CONNECTION_STRING
)
      cNewConnDtl.ConnType =
CheckForNullField(rstConns,
FLD_CONN_DTL_CONNECTION_TYPE)

      cConns.Load cNewConnDtl

      Set cNewConnDtl = Nothing
      rstConns.MoveNext
   Wend

   Exit Sub

LoadRSInConnDtlArrayErr:
   LogErrors Errors
   gstrSource = mstrModuleName &
"LoadRSInConnDtlArray"
   On Error GoTo 0
   Err.Raise vbObjectError +
errLoadRsInArrayFailed, gstrSource, _

LoadResString(errLoadRsInArrayFailed)
End Sub
Attribute VB_Name = "ConnectionCommon"
'  FILE:      ConnnectionCommon.bas
'            Microsoft TPC-H Kit Ver. 1.00
'            Copyright Microsoft, 1999
'            All Rights Reserved
'
'
'  PURPOSE:    Contains functionality common
across StepMaster and
'            SMRunOnly, pertaining to connection
strings
'            Specifically, functions to load
connections strings
'            in an array and so on.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

' Used to indicate the source module name
when errors
' are raised by this module
Private Const mstrModuleName As String =
"ConnectionCommon."

Public Sub
LoadRecordsetInConnectionArray(rstConns
As Recordset, cConns As cConnections)

   Dim cNewConnection As cConnection

   On Error GoTo
LoadRecordsetInConnectionArrayErr

   If rstConns.RecordCount = 0 Then
      Exit Sub
   End If

   rstConns.MoveFirst
   While Not rstConns.EOF

      Set cNewConnection = New cConnection

      ' Initialize Connection values
      ' Call a procedure to raise an error if
mandatory fields are null.
      cNewConnection.ConnectionId =
ErrorOnNullField(rstConns, "connection_id")
      cNewConnection.WorkspaceId =
CStr(ErrorOnNullField(rstConns,
FLD_ID_WORKSPACE))
      cNewConnection.ConnectionName =
CStr(ErrorOnNullField(rstConns,
"connection_name"))
      cNewConnection.ConnectionValue =
CheckForNullField(rstConns, "connection_value")
      cNewConnection.Description =
CheckForNullField(rstConns, "description")

      cNewConnection.NoCountDisplay =
CheckForNullField(rstConns, "no_count_display")
      cNewConnection.NoExecute =
CheckForNullField(rstConns, "no_execute")
      cNewConnection.ParseQueryOnly =
CheckForNullField(rstConns, "parse_query_only")
      cNewConnection.QuotedIdentifiers =
CheckForNullField(rstConns,
"ANSI_quoted_identifiers")
      cNewConnection.AnsiNulls =
CheckForNullField(rstConns, "ANSI_nulls")
      cNewConnection.ShowQueryPlan =
CheckForNullField(rstConns, "show_query_plan")
      cNewConnection.ShowStatsTime =
CheckForNullField(rstConns, "show_stats_time")
      cNewConnection.ShowStatsIO =
CheckForNullField(rstConns, "show_stats_io")
      cNewConnection.ParseOdbcMsg =
CheckForNullField(rstConns,
"parse_odbc_msg_prefixes")
      cNewConnection.RowCount =
CheckForNullField(rstConns, "row_count")
      cNewConnection.TsqlBatchSeparator =
CheckForNullField(rstConns,
"tsql_batch_separator")
      cNewConnection.QueryTimeOut =
CheckForNullField(rstConns, "query_time_out")
      cNewConnection.ServerLanguage =
CheckForNullField(rstConns, "server_language")
      cNewConnection.CharacterTranslation =
CheckForNullField(rstConns,
"character_translation")
      cNewConnection.RegionalSettings =
CheckForNullField(rstConns, "regional_settings")

      cConns.Load cNewConnection

      Set cNewConnection = Nothing
      rstConns.MoveNext
   Wend

   Exit Sub

LoadRecordsetInConnectionArrayErr:
   LogErrors Errors
   gstrSource = mstrModuleName &
"LoadRecordsetInConnectionArray"
   On Error GoTo 0
   Err.Raise vbObjectError +
errLoadRsInArrayFailed, gstrSource, _
          LoadResString(errLoadRsInArrayFailed)
End Sub
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cParameter"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:      cParameter.cls
'            Microsoft TPC-H Kit Ver. 1.00
'            Copyright Microsoft, 1999
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
'           All Rights Reserved
'
'
'  PURPOSE:    Encapsulates the properties
and methods of a parameter.
'           Contains functions to insert,
update and delete
'           workspace_parameters records
from the database.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit
Option Base 0

' Local variable(s) to hold property value(s)
Private mlngWorkspaceId As Long
Private mlngParameterId As Long
Private mstrParameterName As String
Private mstrParameterValue As String
Private mstrDescription As String
Private mintParameterType As Integer
Private mdbsStepMaster As Database
Private mintOperation As Operation
Private mlngPosition As Long

' Used to indicate the source module name
when errors
' are raised by this class
Private mstrSource As String
Private Const mstrModuleName As String =
"cParameter."

' The cSequence class is used to generate
unique parameter identifiers
Private mParameterSeq As cSequence

' The StringSM class is used to carry out
string operations
Private mFieldValue As cStringSM

' Parameter types
Public Enum ParameterType
   gintParameterGeneric = 0
   gintParameterConnect
   gintParameterApplication
   gintParameterBuiltIn
End Enum


Private Sub AssignParameters(qyExec As
DAO.QueryDef)
   ' Assigns values to the parameters in the
querydef object
   ' The parameter names are cryptic to make
them different
   ' from the field names. When the
parameter names are
   ' the same as the field names, parameters
in the where
   ' clause do not get created.

   Dim prmParam As DAO.Parameter

   On Error GoTo AssignParametersErr

   For Each prmParam In
qyExec.Parameters
      Select Case prmParam.Name
         Case "[w_id]"
            prmParam.Value =
mlngWorkspaceId

         Case "[p_id]"
            prmParam.Value = mlngParameterId

         Case "[p_name]"
            prmParam.Value =
mstrParameterName

         Case "[p_value]"
            prmParam.Value =
mstrParameterValue

         Case "[desc]"
            prmParam.Value = mstrDescription

         Case "[p_type]"
            prmParam.Value =
mintParameterType

         Case Else
            ' Write the parameter name that is
faulty
            WriteError errInvalidParameter,
mstrSource, _
                  prmParam.Name
            On Error GoTo 0
            Err.Raise errInvalidParameter,
mstrModuleName & "AssignParameters", _

LoadResString(errInvalidParameter)
      End Select
   Next prmParam

'   qyExec.Parameters("w_id").Value =
mlngWorkspaceId
'   qyExec.Parameters("p_id").Value =
mlngParameterId
'   qyExec.Parameters("p_name").Value =
mstrParameterName
'   qyExec.Parameters("p_value").Value =
mstrParameterValue
'

   Exit Sub

AssignParametersErr:

   mstrSource = mstrModuleName &
"AssignParameters"
   Call LogErrors(Errors)
   On Error GoTo 0
   Err.Raise vbObjectError +
errAssignParametersFailed, _
      mstrSource,
LoadResString(errAssignParametersFailed)

End Sub

Public Property Let Position(ByVal RHS As
Long)

   mlngPosition = RHS

End Property

Public Property Get Position() As Long

   Position = mlngPosition

End Property

Public Function Clone() As cParameter

   ' Creates a copy of a given parameter

   Dim cCloneParam As cParameter

   On Error GoTo CloneErr
   mstrSource = mstrModuleName & "Clone"

   Set cCloneParam = New cParameter

   ' Copy all the parameter properties to the newly
   ' created parameter
   Set cCloneParam.NodeDB = mdbsStepMaster
   cCloneParam.WorkspaceId = mlngWorkspaceId
   cCloneParam.ParameterId = mlngParameterId
   cCloneParam.ParameterName =
mstrParameterName
   cCloneParam.ParameterValue =
mstrParameterValue
   cCloneParam.Description = mstrDescription
   cCloneParam.ParameterType =
mintParameterType
   cCloneParam.IndOperation = mintOperation
   cCloneParam.Position = mlngPosition

   ' And set the return value to the newly created
parameter
   Set Clone = cCloneParam
   Set cCloneParam = Nothing

   Exit Function

CloneErr:
   LogErrors Errors
   mstrSource = mstrModuleName & "Clone"
   On Error GoTo 0
   Err.Raise vbObjectError + errCloneFailed, _
         mstrSource, LoadResString(errCloneFailed)

End Function
Public Property Set NodeDB(vdata As Database)

   Set mdbsStepMaster = vdata

End Property
Public Property Get NodeDB() As Database

   Set NodeDB = mdbsStepMaster

End Property

Private Sub CheckDupParameterName()
   ' Check if the parameter name already exists in
the workspace

   Dim rstParameter As Recordset
   Dim strSql As String
   Dim qy As DAO.QueryDef

   On Error GoTo CheckDupParameterNameErr
   mstrSource = mstrModuleName &
"CheckDupParameterName"

   ' Create a recordset object to retrieve the count of
all parameters
   ' for the workspace with the same name
   strSql = "Select count(*) as parameter_count " &
_
      " from workspace_parameters " & _
      " where workspace_id = [w_id]" & _
      " and parameter_name = [p_name]" & _
      " and parameter_id <> [p_id]"

   Set qy =
mdbsStepMaster.CreateQueryDef(gstrEmptyString,
strSql)
   Call AssignParameters(qy)

   Set rstParameter =
qy.OpenRecordset(dbOpenForwardOnly)
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   If rstParameter![parameter_count] > 0
Then
      rstParameter.Close
      qy.Close
      ShowError
errDuplicateParameterName
      On Error GoTo 0
      Err.Raise vbObjectError +
errDuplicateParameterName, _
         mstrSource,
LoadResString(errDuplicateParameterName
)
   End If

   rstParameter.Close
   qy.Close

   Exit Sub

CheckDupParameterNameErr:
   LogErrors Errors
   mstrSource = mstrModuleName &
"CheckDupParameterName"
   On Error GoTo 0
   Err.Raise vbObjectError +
errCheckDupParameterNameFailed, _
      mstrSource,
LoadResString(errCheckDupParameterNam
eFailed)

End Sub
Private Sub CheckDB()
   ' Check if the database object has been
initialized

   If mdbsStepMaster Is Nothing Then
      On Error GoTo 0
      Err.Raise vbObjectError +
errInvalidDB, _
         mstrModuleName & "CheckDB",
LoadResString(errInvalidDB)
   End If

End Sub
Public Property Let ParameterValue(vdata
As String)

   mstrParameterValue = vdata

End Property
Public Property Let Description(vdata As
String)

   mstrDescription = vdata

End Property
Public Property Let ParameterType(vdata
As ParameterType)

   mintParameterType = vdata

End Property

Public Property Let ParameterName(vdata
As String)

   If vdata = gstrEmptyString Then

      ShowError
errParameterNameMandatory
      On Error GoTo 0
      ' Propogate this error back to the caller
      Err.Raise vbObjectError +
errParameterNameMandatory, _
         mstrSource,
LoadResString(errParameterNameMandatory)
   Else
      mstrParameterName = vdata
   End If

End Property

Public Property Let ParameterId(vdata As
Long)
   mlngParameterId = vdata
End Property

Public Property Let IndOperation(ByVal vdata
As Operation)

   ' The valid operations are define in the
cOperations
   ' class. Check if the operation is valid
   Select Case vdata
      Case QueryOp, InsertOp, UpdateOp,
DeleteOp
         mintOperation = vdata

      Case Else
         On Error GoTo 0
         Err.Raise vbObjectError +
errInvalidOperation, _
            mstrSource,
LoadResString(errInvalidOperation)
   End Select

End Property
Public Sub Validate()
   ' Each distinct object will have a Validate
method which
   ' will check if the class properties are valid.
This method
   ' will be used to check interdependant
properties that
   ' cannot be validated by the let procedures.
   ' It should be called by the add and modify
methods of the class

   On Error GoTo ValidateErr

   ' Check if the db object is valid
   Call CheckDB

   ' Call procedure to raise an error if the
parameter name
   ' already exists in the workspace -
   ' if there are duplicates, we don't know what
value for the
   ' parameter to use at runtime
   Call CheckDupParameterName

   Exit Sub

ValidateErr:

   mstrSource = mstrModuleName &
"Validate"
   Call LogErrors(Errors)
   On Error GoTo 0
   Err.Raise vbObjectError +
errValidateFailed, _
      mstrSource,
LoadResString(errValidateFailed)

End Sub
Public Property Let WorkspaceId(vdata As
Long)

   mlngWorkspaceId = vdata

End Property

Public Sub Add()

   Dim strInsert As String
   Dim qy As DAO.QueryDef

   On Error GoTo AddErr

   ' Validate the record before trying to insert the
record
   Call Validate

   ' Create a temporary querydef object
   strInsert = "insert into workspace_parameters " &
_
         "( workspace_id, parameter_id, " & _
         " parameter_name, parameter_value, " & _
         " description, parameter_type ) " & _
         " values ( [w_id], [p_id], [p_name],
[p_value], [desc], [p_type] ) "
   Set qy =
mdbsStepMaster.CreateQueryDef(gstrEmptyString,
strInsert)

   ' Call a procedure to assign the parameter values
   Call AssignParameters(qy)

   qy.Execute dbFailOnError
   qy.Close

'   strInsert = "insert into workspace_parameters "
& _
'         "( workspace_id, parameter_id, " & _
'         " parameter_name, parameter_value ) " & _
'         " values ( " & _
'         Str(mlngWorkspaceId) & ", " &
Str(mlngParameterId) & _
'         ", " &
mFieldValue.MakeStringFieldValid(mstrParameter
Name) & _
'         ", " &
mFieldValue.MakeStringFieldValid(mstrParameter
Value) & " ) "
'   mdbsStepMaster.Execute strInsert,
dbFailOnError + dbSQLPassThrough

   Exit Sub

AddErr:

   mstrSource = mstrModuleName & "Add"
   Call LogErrors(Errors)
   On Error GoTo 0
   Err.Raise vbObjectError +
errParameterInsertFailed, _
      mstrSource,
LoadResString(errParameterInsertFailed)

End Sub
Public Sub Delete()

   Dim strDelete As String
   Dim qy As DAO.QueryDef

   On Error GoTo DeleteErr

   ' Check if the db object is valid
   Call CheckDB

   strDelete = "delete from workspace_parameters "
& _
         " where parameter_id = [p_id]"
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   Set qy =
mdbsStepMaster.CreateQueryDef(gstrEmpt
yString, strDelete)

   Call AssignParameters(qy)
   qy.Execute dbFailOnError

   qy.Close

   Exit Sub

DeleteErr:
   LogErrors Errors
   mstrSource = mstrModuleName &
"Delete"
   On Error GoTo 0
   Err.Raise vbObjectError +
errDeleteParameterFailed, _
         mstrSource, _

LoadResString(errDeleteParameterFailed)

End Sub

Public Sub Modify()

   Dim strUpdate As String
   Dim qy As QueryDef

   On Error GoTo ModifyErr

   ' Validate the updated values before trying
to modify the db
   Call Validate

   ' Create a temporary querydef object with
the modify string
   strUpdate = "update
workspace_parameters " & _
         " set workspace_id = [w_id], " & _
         "parameter_name = [p_name], " & _
         "parameter_value = [p_value], " & _
         "description = [desc], " & _
         "parameter_type = [p_type] " & _
         " where parameter_id = [p_id]"
   Set qy =
mdbsStepMaster.CreateQueryDef(gstrEmpt
yString, strUpdate)

   ' Call a procedure to assign the parameter
values to the
   ' querydef object
   Call AssignParameters(qy)
   qy.Execute dbFailOnError

   qy.Close

'   mdbsStepMaster.Execute strUpdate,
dbFailOnError
'
   Exit Sub

ModifyErr:

   mstrSource = mstrModuleName &
"Modify"
   Call LogErrors(Errors)
   On Error GoTo 0
   Err.Raise vbObjectError +
errParameterUpdateFailed, _
      mstrSource,
LoadResString(errParameterUpdateFailed)

End Sub
Public Property Get ParameterName() As
String

   ParameterName = mstrParameterName

End Property

Public Property Get ParameterId() As Long

   ParameterId = mlngParameterId

End Property
Public Property Get NextIdentifier() As Long

   Dim lngNextId As Long

   On Error GoTo NextIdentifierErr

   ' First check if the database object is valid
   Call CheckDB

   ' Retrieve the next identifier using the
sequence class
   Set mParameterSeq = New cSequence
   Set mParameterSeq.IdDatabase =
mdbsStepMaster
   mParameterSeq.IdentifierColumn =
FLD_ID_PARAMETER
   lngNextId = mParameterSeq.Identifier
   Set mParameterSeq = Nothing

   NextIdentifier = lngNextId
   Exit Property

NextIdentifierErr:
   LogErrors Errors
   mstrSource = mstrModuleName &
"NextIdentifier"
   On Error GoTo 0
   Err.Raise vbObjectError + errIdGetFailed, _
      mstrSource,
LoadResString(errIdGetFailed)

End Property
Public Property Get IndOperation() As
Operation

   IndOperation = mintOperation

End Property

Public Property Get WorkspaceId() As Long

   WorkspaceId = mlngWorkspaceId

End Property

Public Property Get ParameterValue() As
String

   ParameterValue = mstrParameterValue

End Property
Public Property Get Description() As String

   Description = mstrDescription

End Property
Public Property Get ParameterType() As
ParameterType

   ParameterType = mintParameterType

End Property

Private Sub Class_Initialize()

   Set mFieldValue = New cStringSM

   ' Initialize the operation indicator variable to
Query
   ' It will be modified later by the collection class
when
   ' inserts, updates or deletes are performed
   mintOperation = QueryOp

End Sub

Private Sub Class_Terminate()

   Set mdbsStepMaster = Nothing
   Set mFieldValue = Nothing

End Sub
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cRunColIt"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:       cRunColIt.cls
'             Microsoft TPC-H Kit Ver. 1.00
'             Copyright Microsoft, 1999
'             All Rights Reserved
'
'
'  PURPOSE:    This module implements a stack of
Iterator nodes.
'             Ensures that only cRunItNode objects are
stored in the stack.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

' Used to indicate the source module name when
errors
' are raised by this class
Private Const mstrModuleName As String =
"cRunColIt."
Private mstrSource As String

Private mcIterators As cStack
Public Sub Clear()

   mcIterators.Clear

End Sub

Private Sub Class_Initialize()

   Set mcIterators = New cStack

End Sub

Private Sub Class_Terminate()

   Set mcIterators = Nothing

End Sub


Public Function Value(strItName As String) As
String

   Dim lngIndex As Long
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   For lngIndex = 0 To mcIterators.Count - 1
      If mcIterators(lngIndex).IteratorName
= strItName Then
         Value = mcIterators(lngIndex).Value
         Exit For
      End If
   Next lngIndex

End Function

Public Property Get Item(ByVal Position As
Long) As cRunItNode
Attribute Item.VB_UserMemId = 0

   Set Item = mcIterators(Position)

End Property


Public Function Count() As Long

   Count = mcIterators.Count

End Function

Public Function Pop() As cRunItNode

   Set Pop = mcIterators.Pop

End Function

Public Sub Push(objToPush As
cRunItNode)

   Call mcIterators.Push(objToPush)

End Sub


VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
  Persistable = 0  'NotPersistable
  DataBindingBehavior = 0  'vbNone
  DataSourceBehavior  = 0  'vbNone
  MTSTransactionMode  = 0
'NotAnMTSObject
END
Attribute VB_Name = "cRunInst"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:       cRunColIt.cls
'            Microsoft TPC-H Kit Ver. 1.00
'            Copyright Microsoft, 1999
'            All Rights Reserved
'
'
'  PURPOSE:    This module controls the
run processing. It runs a branch
'            at a time and raises events when
each step completes execution.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

' Used to indicate the source module name
when errors
' are raised by this class
Private Const mstrModuleName As String =
"cRunInst."
Private mstrSource As String

' Local variable(s) to hold property value(s)
Private mstrRootKey As String
Public WspId As Long
Private mcParameters As cArrParameters
Private mcRunSteps As cArrSteps
Private mcRunConstraints As cArrConstraints
Public RunConnections As cConnections
Public RunConnDtls As cConnDtls
Private mcvntWspPreCons As Variant
Private mcvntWspPostCons As Variant
Private mcNavSteps As cStepTree

Private mcInstances As cInstances
Private mcFreeSteps As cVectorLng
Private mcFailures As cFailedSteps
Private mblnAsk As Boolean   ' Set to True
when the a step with continuation criteria=Ask
fails
Private mblnAbort As Boolean ' Set to True
when the run is aborted
Private msAbortDtls As String
Private mbarrFree() As Byte
Private WithEvents mcTermSteps As
cTermSteps
Attribute mcTermSteps.VB_VarHelpID = -1
Public RunId As Long
Public CreateInputFiles As Boolean
Private Enum WspLogEvents
   mintRunStart
   mintRunComplete
   mintStepStart
   mintStepComplete
End Enum

Private mcWspLog As cFileSM
Private mstrCurBranchRoot As String
Private mcDummyRootInstance As cInstance
' Key for the dummy root instance - Should be
a key that is invalid for an actual step record
Private Const mstrDummyRootKey As String
= "D"
' Public events to notify the calling function of
the
' start and end time for each step
Public Event RunStart(dtmStartTime As
Currency, strWspLog As String)
Public Event RunComplete(dtmEndTime As
Currency)
Public Event StepStart(cStepRecord As cStep,
dtmStartTime As Currency, _
      lngInstanceId As Long, lParentInstanceId
As Long, sPath As String, _
      sIts As String, sItValue As String)
Public Event StepComplete(cStepRecord As
cStep, dtmEndTime As Currency,
lngInstanceId As Long, lElapsed As Long)
Public Event ProcessStart(cStepRecord As
cStep, strCommand As String, _
      dtmStartTime As Currency, lngInstanceId
As Long, lParentInstanceId As Long, _
      sItValue As String)
Public Event ProcessComplete(cStepRecord
As cStep, dtmEndTime As Currency,
lngInstanceId As Long, lElapsed As Long)
' The class that will execute each step - we trap
the events
' that are raised by it when a step
starts/completes
' execution
Private WithEvents cExecStep1 As cRunStep
Attribute cExecStep1.VB_VarHelpID = -1
Private WithEvents cExecStep2 As cRunStep
Attribute cExecStep2.VB_VarHelpID = -1
Private WithEvents cExecStep3 As cRunStep
Attribute cExecStep3.VB_VarHelpID = -1
Private WithEvents cExecStep4 As cRunStep
Attribute cExecStep4.VB_VarHelpID = -1
Private WithEvents cExecStep5 As cRunStep
Attribute cExecStep5.VB_VarHelpID = -1
Private WithEvents cExecStep6 As cRunStep
Attribute cExecStep6.VB_VarHelpID = -1
Private WithEvents cExecStep7 As cRunStep
Attribute cExecStep7.VB_VarHelpID = -1
Private WithEvents cExecStep8 As cRunStep
Attribute cExecStep8.VB_VarHelpID = -1
Private WithEvents cExecStep9 As cRunStep
Attribute cExecStep9.VB_VarHelpID = -1

Private WithEvents cExecStep10 As cRunStep
Attribute cExecStep10.VB_VarHelpID = -1
Private WithEvents cExecStep11 As cRunStep
Attribute cExecStep11.VB_VarHelpID = -1
Private WithEvents cExecStep12 As cRunStep
Attribute cExecStep12.VB_VarHelpID = -1
Private WithEvents cExecStep13 As cRunStep
Attribute cExecStep13.VB_VarHelpID = -1
Private WithEvents cExecStep14 As cRunStep
Attribute cExecStep14.VB_VarHelpID = -1
Private WithEvents cExecStep15 As cRunStep
Attribute cExecStep15.VB_VarHelpID = -1
Private WithEvents cExecStep16 As cRunStep
Attribute cExecStep16.VB_VarHelpID = -1
Private WithEvents cExecStep17 As cRunStep
Attribute cExecStep17.VB_VarHelpID = -1
Private WithEvents cExecStep18 As cRunStep
Attribute cExecStep18.VB_VarHelpID = -1
Private WithEvents cExecStep19 As cRunStep
Attribute cExecStep19.VB_VarHelpID = -1

Private WithEvents cExecStep20 As cRunStep
Attribute cExecStep20.VB_VarHelpID = -1
Private WithEvents cExecStep21 As cRunStep
Attribute cExecStep21.VB_VarHelpID = -1
Private WithEvents cExecStep22 As cRunStep
Attribute cExecStep22.VB_VarHelpID = -1
Private WithEvents cExecStep23 As cRunStep
Attribute cExecStep23.VB_VarHelpID = -1
Private WithEvents cExecStep24 As cRunStep
Attribute cExecStep24.VB_VarHelpID = -1
Private WithEvents cExecStep25 As cRunStep
Attribute cExecStep25.VB_VarHelpID = -1
Private WithEvents cExecStep26 As cRunStep
Attribute cExecStep26.VB_VarHelpID = -1
Private WithEvents cExecStep27 As cRunStep
Attribute cExecStep27.VB_VarHelpID = -1
Private WithEvents cExecStep28 As cRunStep
Attribute cExecStep28.VB_VarHelpID = -1
Private WithEvents cExecStep29 As cRunStep
Attribute cExecStep29.VB_VarHelpID = -1

Private WithEvents cExecStep30 As cRunStep
Attribute cExecStep30.VB_VarHelpID = -1
Private WithEvents cExecStep31 As cRunStep
Attribute cExecStep31.VB_VarHelpID = -1
Private WithEvents cExecStep32 As cRunStep
Attribute cExecStep32.VB_VarHelpID = -1
Private WithEvents cExecStep33 As cRunStep
Attribute cExecStep33.VB_VarHelpID = -1
Private WithEvents cExecStep34 As cRunStep
Attribute cExecStep34.VB_VarHelpID = -1
Private WithEvents cExecStep35 As cRunStep
Attribute cExecStep35.VB_VarHelpID = -1
Private WithEvents cExecStep36 As cRunStep
Attribute cExecStep36.VB_VarHelpID = -1
Private WithEvents cExecStep37 As cRunStep
Attribute cExecStep37.VB_VarHelpID = -1
Private WithEvents cExecStep38 As cRunStep
Attribute cExecStep38.VB_VarHelpID = -1
Private WithEvents cExecStep39 As cRunStep
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
Attribute cExecStep39.VB_VarHelpID = -1

Private WithEvents cExecStep40 As
cRunStep
Attribute cExecStep40.VB_VarHelpID = -1
Private WithEvents cExecStep41 As
cRunStep
Attribute cExecStep41.VB_VarHelpID = -1
Private WithEvents cExecStep42 As
cRunStep
Attribute cExecStep42.VB_VarHelpID = -1
Private WithEvents cExecStep43 As
cRunStep
Attribute cExecStep43.VB_VarHelpID = -1
Private WithEvents cExecStep44 As
cRunStep
Attribute cExecStep44.VB_VarHelpID = -1
Private WithEvents cExecStep45 As
cRunStep
Attribute cExecStep45.VB_VarHelpID = -1
Private WithEvents cExecStep46 As
cRunStep
Attribute cExecStep46.VB_VarHelpID = -1
Private WithEvents cExecStep47 As
cRunStep
Attribute cExecStep47.VB_VarHelpID = -1
Private WithEvents cExecStep48 As
cRunStep
Attribute cExecStep48.VB_VarHelpID = -1
Private WithEvents cExecStep49 As
cRunStep
Attribute cExecStep49.VB_VarHelpID = -1

Private WithEvents cExecStep50 As
cRunStep
Attribute cExecStep50.VB_VarHelpID = -1
Private WithEvents cExecStep51 As
cRunStep
Attribute cExecStep51.VB_VarHelpID = -1
Private WithEvents cExecStep52 As
cRunStep
Attribute cExecStep52.VB_VarHelpID = -1
Private WithEvents cExecStep53 As
cRunStep
Attribute cExecStep53.VB_VarHelpID = -1
Private WithEvents cExecStep54 As
cRunStep
Attribute cExecStep54.VB_VarHelpID = -1
Private WithEvents cExecStep55 As
cRunStep
Attribute cExecStep55.VB_VarHelpID = -1
Private WithEvents cExecStep56 As
cRunStep
Attribute cExecStep56.VB_VarHelpID = -1
Private WithEvents cExecStep57 As
cRunStep
Attribute cExecStep57.VB_VarHelpID = -1
Private WithEvents cExecStep58 As
cRunStep
Attribute cExecStep58.VB_VarHelpID = -1
Private WithEvents cExecStep59 As
cRunStep
Attribute cExecStep59.VB_VarHelpID = -1

Private WithEvents cExecStep60 As
cRunStep
Attribute cExecStep60.VB_VarHelpID = -1
Private WithEvents cExecStep61 As
cRunStep
Attribute cExecStep61.VB_VarHelpID = -1
Private WithEvents cExecStep62 As
cRunStep
Attribute cExecStep62.VB_VarHelpID = -1
Private WithEvents cExecStep63 As
cRunStep
Attribute cExecStep63.VB_VarHelpID = -1
Private WithEvents cExecStep64 As cRunStep
Attribute cExecStep64.VB_VarHelpID = -1
Private WithEvents cExecStep65 As cRunStep
Attribute cExecStep65.VB_VarHelpID = -1
Private WithEvents cExecStep66 As cRunStep
Attribute cExecStep66.VB_VarHelpID = -1
Private WithEvents cExecStep67 As cRunStep
Attribute cExecStep67.VB_VarHelpID = -1
Private WithEvents cExecStep68 As cRunStep
Attribute cExecStep68.VB_VarHelpID = -1
Private WithEvents cExecStep69 As cRunStep
Attribute cExecStep69.VB_VarHelpID = -1

Private WithEvents cExecStep70 As cRunStep
Attribute cExecStep70.VB_VarHelpID = -1
Private WithEvents cExecStep71 As cRunStep
Attribute cExecStep71.VB_VarHelpID = -1
Private WithEvents cExecStep72 As cRunStep
Attribute cExecStep72.VB_VarHelpID = -1
Private WithEvents cExecStep73 As cRunStep
Attribute cExecStep73.VB_VarHelpID = -1
Private WithEvents cExecStep74 As cRunStep
Attribute cExecStep74.VB_VarHelpID = -1
Private WithEvents cExecStep75 As cRunStep
Attribute cExecStep75.VB_VarHelpID = -1
Private WithEvents cExecStep76 As cRunStep
Attribute cExecStep76.VB_VarHelpID = -1
Private WithEvents cExecStep77 As cRunStep
Attribute cExecStep77.VB_VarHelpID = -1
Private WithEvents cExecStep78 As cRunStep
Attribute cExecStep78.VB_VarHelpID = -1
Private WithEvents cExecStep79 As cRunStep
Attribute cExecStep79.VB_VarHelpID = -1

Private WithEvents cExecStep80 As cRunStep
Attribute cExecStep80.VB_VarHelpID = -1
Private WithEvents cExecStep81 As cRunStep
Attribute cExecStep81.VB_VarHelpID = -1
Private WithEvents cExecStep82 As cRunStep
Attribute cExecStep82.VB_VarHelpID = -1
Private WithEvents cExecStep83 As cRunStep
Attribute cExecStep83.VB_VarHelpID = -1
Private WithEvents cExecStep84 As cRunStep
Attribute cExecStep84.VB_VarHelpID = -1
Private WithEvents cExecStep85 As cRunStep
Attribute cExecStep85.VB_VarHelpID = -1
Private WithEvents cExecStep86 As cRunStep
Attribute cExecStep86.VB_VarHelpID = -1
Private WithEvents cExecStep87 As cRunStep
Attribute cExecStep87.VB_VarHelpID = -1
Private WithEvents cExecStep88 As cRunStep
Attribute cExecStep88.VB_VarHelpID = -1
Private WithEvents cExecStep89 As cRunStep
Attribute cExecStep89.VB_VarHelpID = -1

Private WithEvents cExecStep90 As cRunStep
Attribute cExecStep90.VB_VarHelpID = -1
Private WithEvents cExecStep91 As cRunStep
Attribute cExecStep91.VB_VarHelpID = -1
Private WithEvents cExecStep92 As cRunStep
Attribute cExecStep92.VB_VarHelpID = -1
Private WithEvents cExecStep93 As cRunStep
Attribute cExecStep93.VB_VarHelpID = -1
Private WithEvents cExecStep94 As cRunStep
Attribute cExecStep94.VB_VarHelpID = -1
Private WithEvents cExecStep95 As cRunStep
Attribute cExecStep95.VB_VarHelpID = -1
Private WithEvents cExecStep96 As cRunStep
Attribute cExecStep96.VB_VarHelpID = -1
Private WithEvents cExecStep97 As cRunStep
Attribute cExecStep97.VB_VarHelpID = -1
Private WithEvents cExecStep98 As cRunStep
Attribute cExecStep98.VB_VarHelpID = -1
Private WithEvents cExecStep99 As cRunStep
Attribute cExecStep99.VB_VarHelpID = -1

Private Const msIt As String = " Iterator: "
Private Const msItValue As String = " Value: "
Public Sub Abort()

   On Error GoTo AbortErr

   ' Make sure that we don't execute any more steps
   Call StopRun

   If cExecStep1 Is Nothing And cExecStep2 Is
Nothing And cExecStep3 Is Nothing And
cExecStep4 Is Nothing And cExecStep5 Is Nothing
And cExecStep6 Is Nothing And cExecStep7 Is
Nothing And cExecStep8 Is Nothing And
cExecStep9 Is Nothing And _
      cExecStep10 Is Nothing And cExecStep11 Is
Nothing And cExecStep12 Is Nothing And
cExecStep13 Is Nothing And cExecStep14 Is
Nothing And cExecStep15 Is Nothing And
cExecStep16 Is Nothing And cExecStep17 Is
Nothing And cExecStep18 Is Nothing And
cExecStep19 Is Nothing And _
      cExecStep20 Is Nothing And cExecStep21 Is
Nothing And cExecStep22 Is Nothing And
cExecStep23 Is Nothing And cExecStep24 Is
Nothing And cExecStep25 Is Nothing And
cExecStep26 Is Nothing And cExecStep27 Is
Nothing And cExecStep28 Is Nothing And
cExecStep29 Is Nothing And _
      cExecStep30 Is Nothing And cExecStep31 Is
Nothing And cExecStep32 Is Nothing And
cExecStep33 Is Nothing And cExecStep34 Is
Nothing And cExecStep35 Is Nothing And
cExecStep36 Is Nothing And cExecStep37 Is
Nothing And cExecStep38 Is Nothing And
cExecStep39 Is Nothing And _
      cExecStep40 Is Nothing And cExecStep41 Is
Nothing And cExecStep42 Is Nothing And
cExecStep43 Is Nothing And cExecStep44 Is
Nothing And cExecStep45 Is Nothing And
cExecStep46 Is Nothing And cExecStep47 Is
Nothing And cExecStep48 Is Nothing And
cExecStep49 Is Nothing And _
      cExecStep50 Is Nothing And cExecStep51 Is
Nothing And cExecStep52 Is Nothing And
cExecStep53 Is Nothing And cExecStep54 Is
Nothing And cExecStep55 Is Nothing And
cExecStep56 Is Nothing And cExecStep57 Is
Nothing And cExecStep58 Is Nothing And
cExecStep59 Is Nothing And _
      cExecStep60 Is Nothing And cExecStep61 Is
Nothing And cExecStep62 Is Nothing And
cExecStep63 Is Nothing And cExecStep64 Is
Nothing And cExecStep65 Is Nothing And
cExecStep66 Is Nothing And cExecStep67 Is
Nothing And cExecStep68 Is Nothing And
cExecStep69 Is Nothing And _
      cExecStep70 Is Nothing And cExecStep71 Is
Nothing And cExecStep72 Is Nothing And
cExecStep73 Is Nothing And cExecStep74 Is
Nothing And cExecStep75 Is Nothing And
cExecStep76 Is Nothing And cExecStep77 Is
Nothing And cExecStep78 Is Nothing And
cExecStep79 Is Nothing And _
      cExecStep80 Is Nothing And cExecStep81 Is
Nothing And cExecStep82 Is Nothing And
cExecStep83 Is Nothing And cExecStep84 Is
Nothing And cExecStep85 Is Nothing And
cExecStep86 Is Nothing And cExecStep87 Is
Nothing And cExecStep88 Is Nothing And
cExecStep89 Is Nothing And _
      cExecStep90 Is Nothing And cExecStep91 Is
Nothing And cExecStep92 Is Nothing And
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
cExecStep93 Is Nothing And cExecStep94
Is Nothing And cExecStep95 Is Nothing
And cExecStep96 Is Nothing And
cExecStep97 Is Nothing And cExecStep98
Is Nothing And cExecStep99 Is Nothing
Then
      ' Then...
      WriteToWspLog (mintRunComplete)
      RaiseEvent
RunComplete(Determine64BitTime())
   Else
      ' Abort each of the steps that is
currently executing.
      If Not cExecStep1 Is Nothing Then
         cExecStep1.Abort
      End If
      If Not cExecStep2 Is Nothing Then
         cExecStep2.Abort
      End If
      If Not cExecStep3 Is Nothing Then
         cExecStep3.Abort
      End If
      If Not cExecStep4 Is Nothing Then
         cExecStep4.Abort
      End If
      If Not cExecStep5 Is Nothing Then
         cExecStep5.Abort
      End If
      If Not cExecStep6 Is Nothing Then
         cExecStep6.Abort
      End If
      If Not cExecStep7 Is Nothing Then
         cExecStep7.Abort
      End If
      If Not cExecStep8 Is Nothing Then
         cExecStep8.Abort
      End If
      If Not cExecStep9 Is Nothing Then
         cExecStep9.Abort
      End If
      If Not cExecStep10 Is Nothing Then
         cExecStep10.Abort
      End If
      If Not cExecStep11 Is Nothing Then
         cExecStep11.Abort
      End If
      If Not cExecStep12 Is Nothing Then
         cExecStep12.Abort
      End If
      If Not cExecStep13 Is Nothing Then
         cExecStep13.Abort
      End If

      If Not cExecStep14 Is Nothing Then
         cExecStep14.Abort
      End If

      If Not cExecStep15 Is Nothing Then
         cExecStep15.Abort
      End If

      If Not cExecStep16 Is Nothing Then
         cExecStep16.Abort
      End If

      If Not cExecStep17 Is Nothing Then
         cExecStep17.Abort
      End If

      If Not cExecStep18 Is Nothing Then
         cExecStep18.Abort
      End If

      If Not cExecStep19 Is Nothing Then
         cExecStep19.Abort
      End If

      If Not cExecStep20 Is Nothing Then
         cExecStep20.Abort
      End If

      If Not cExecStep21 Is Nothing Then
         cExecStep21.Abort
      End If

      If Not cExecStep22 Is Nothing Then
         cExecStep22.Abort
      End If

      If Not cExecStep23 Is Nothing Then
         cExecStep23.Abort
      End If

      If Not cExecStep24 Is Nothing Then
         cExecStep24.Abort
      End If

      If Not cExecStep25 Is Nothing Then
         cExecStep25.Abort
      End If

      If Not cExecStep26 Is Nothing Then
         cExecStep26.Abort
      End If

      If Not cExecStep27 Is Nothing Then
         cExecStep27.Abort
      End If

      If Not cExecStep28 Is Nothing Then
         cExecStep28.Abort
      End If

      If Not cExecStep29 Is Nothing Then
         cExecStep29.Abort
      End If

      ' ============== 30 - 39
==============
      If Not cExecStep30 Is Nothing Then
         cExecStep30.Abort
      End If

      If Not cExecStep31 Is Nothing Then
         cExecStep31.Abort
      End If

      If Not cExecStep32 Is Nothing Then
         cExecStep32.Abort
      End If

      If Not cExecStep33 Is Nothing Then
         cExecStep33.Abort
      End If

      If Not cExecStep34 Is Nothing Then
         cExecStep34.Abort
      End If

      If Not cExecStep35 Is Nothing Then
         cExecStep35.Abort
      End If

      If Not cExecStep36 Is Nothing Then
         cExecStep36.Abort
      End If

      If Not cExecStep37 Is Nothing Then
         cExecStep37.Abort
      End If

      If Not cExecStep38 Is Nothing Then
         cExecStep38.Abort
      End If

      If Not cExecStep39 Is Nothing Then
         cExecStep39.Abort
      End If

      ' ============== 40 - 49
==============
      If Not cExecStep40 Is Nothing Then
         cExecStep40.Abort
      End If

      If Not cExecStep41 Is Nothing Then
         cExecStep41.Abort
      End If

      If Not cExecStep42 Is Nothing Then
         cExecStep42.Abort
      End If

      If Not cExecStep43 Is Nothing Then
         cExecStep43.Abort
      End If

      If Not cExecStep44 Is Nothing Then
         cExecStep44.Abort
      End If

      If Not cExecStep45 Is Nothing Then
         cExecStep45.Abort
      End If

      If Not cExecStep46 Is Nothing Then
         cExecStep46.Abort
      End If

      If Not cExecStep47 Is Nothing Then
         cExecStep47.Abort
      End If

      If Not cExecStep48 Is Nothing Then
         cExecStep48.Abort
      End If

      If Not cExecStep49 Is Nothing Then
         cExecStep49.Abort
      End If

      ' ============== 50 - 59
==============
      If Not cExecStep50 Is Nothing Then
         cExecStep50.Abort
      End If

      If Not cExecStep51 Is Nothing Then
         cExecStep51.Abort
      End If

      If Not cExecStep52 Is Nothing Then
         cExecStep52.Abort
      End If

      If Not cExecStep53 Is Nothing Then
         cExecStep53.Abort
      End If

      If Not cExecStep54 Is Nothing Then
         cExecStep54.Abort
      End If

      If Not cExecStep55 Is Nothing Then
         cExecStep55.Abort
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
    End If

    If Not cExecStep56 Is Nothing Then
        cExecStep56.Abort
    End If

    If Not cExecStep57 Is Nothing Then
        cExecStep57.Abort
    End If

    If Not cExecStep58 Is Nothing Then
        cExecStep58.Abort
    End If

    If Not cExecStep59 Is Nothing Then
        cExecStep59.Abort
    End If

    ' ============== 60 - 69
==============
    If Not cExecStep60 Is Nothing Then
        cExecStep60.Abort
    End If

    If Not cExecStep61 Is Nothing Then
        cExecStep61.Abort
    End If

    If Not cExecStep62 Is Nothing Then
        cExecStep62.Abort
    End If

    If Not cExecStep63 Is Nothing Then
        cExecStep63.Abort
    End If

    If Not cExecStep64 Is Nothing Then
        cExecStep64.Abort
    End If

    If Not cExecStep65 Is Nothing Then
        cExecStep65.Abort
    End If

    If Not cExecStep66 Is Nothing Then
        cExecStep66.Abort
    End If

    If Not cExecStep67 Is Nothing Then
        cExecStep67.Abort
    End If

    If Not cExecStep68 Is Nothing Then
        cExecStep68.Abort
    End If

    If Not cExecStep69 Is Nothing Then
        cExecStep69.Abort
    End If

    ' ============== 70 - 79
==============
    If Not cExecStep70 Is Nothing Then
        cExecStep70.Abort
    End If

    If Not cExecStep71 Is Nothing Then
        cExecStep71.Abort
    End If

    If Not cExecStep72 Is Nothing Then
        cExecStep72.Abort
    End If

    If Not cExecStep73 Is Nothing Then
        cExecStep73.Abort
    End If

    If Not cExecStep74 Is Nothing Then
        cExecStep74.Abort
    End If

    If Not cExecStep75 Is Nothing Then
        cExecStep75.Abort
    End If

    If Not cExecStep76 Is Nothing Then
        cExecStep76.Abort
    End If

    If Not cExecStep77 Is Nothing Then
        cExecStep77.Abort
    End If

    If Not cExecStep78 Is Nothing Then
        cExecStep78.Abort
    End If

    If Not cExecStep79 Is Nothing Then
        cExecStep79.Abort
    End If

    ' ============== 80 - 89
==============
    If Not cExecStep80 Is Nothing Then
        cExecStep80.Abort
    End If

    If Not cExecStep81 Is Nothing Then
        cExecStep81.Abort
    End If

    If Not cExecStep82 Is Nothing Then
        cExecStep82.Abort
    End If

    If Not cExecStep83 Is Nothing Then
        cExecStep83.Abort
    End If

    If Not cExecStep84 Is Nothing Then
        cExecStep84.Abort
    End If

    If Not cExecStep85 Is Nothing Then
        cExecStep85.Abort
    End If

    If Not cExecStep86 Is Nothing Then
        cExecStep86.Abort
    End If

    If Not cExecStep87 Is Nothing Then
        cExecStep87.Abort
    End If

    If Not cExecStep88 Is Nothing Then
        cExecStep88.Abort
    End If

    If Not cExecStep89 Is Nothing Then
        cExecStep89.Abort
    End If

    ' ============== 90 - 99
==============
    If Not cExecStep90 Is Nothing Then
        cExecStep90.Abort
    End If

    If Not cExecStep91 Is Nothing Then
        cExecStep91.Abort
    End If

    If Not cExecStep92 Is Nothing Then
        cExecStep92.Abort
    End If

    If Not cExecStep93 Is Nothing Then
        cExecStep93.Abort
    End If

    If Not cExecStep94 Is Nothing Then
        cExecStep94.Abort
    End If

    If Not cExecStep95 Is Nothing Then
        cExecStep95.Abort
    End If

    If Not cExecStep96 Is Nothing Then
        cExecStep96.Abort
    End If

    If Not cExecStep97 Is Nothing Then
        cExecStep97.Abort
    End If

    If Not cExecStep98 Is Nothing Then
        cExecStep98.Abort
    End If

    If Not cExecStep99 Is Nothing Then
        cExecStep99.Abort
    End If

  End If

  Exit Sub

AbortErr:
  Call LogErrors(Errors)
  On Error GoTo 0
  ShowError errAbortFailed
  ' Try to abort the remaining steps, if any
  Resume Next

End Sub
Public Sub AbortSiblings(cTermInstance As
cInstance)

  On Error GoTo AbortSiblingsErr

  ' Abort each of the steps that is currently
executing.
  If Not cExecStep1 Is Nothing Then
    If cExecStep1.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
      cExecStep1.Abort
    End If
  End If

  If Not cExecStep2 Is Nothing Then
    If cExecStep2.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
      cExecStep2.Abort
    End If
  End If

  If Not cExecStep3 Is Nothing Then
    If cExecStep3.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
      cExecStep3.Abort
    End If
  End If
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
    If Not cExecStep4 Is Nothing Then
        If
cExecStep4.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
            cExecStep4.Abort
        End If
    End If

    If Not cExecStep5 Is Nothing Then
        If
cExecStep5.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
            cExecStep5.Abort
        End If
    End If

    If Not cExecStep6 Is Nothing Then
        If
cExecStep6.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
            cExecStep6.Abort
        End If
    End If

    If Not cExecStep7 Is Nothing Then
        If
cExecStep7.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
            cExecStep7.Abort
        End If
    End If

    If Not cExecStep8 Is Nothing Then
        If
cExecStep8.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
            cExecStep8.Abort
        End If
    End If

    If Not cExecStep9 Is Nothing Then
        If
cExecStep9.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
            cExecStep9.Abort
        End If
    End If

    If Not cExecStep10 Is Nothing Then
        If
cExecStep10.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
            cExecStep10.Abort
        End If
    End If

    If Not cExecStep11 Is Nothing Then
        If
cExecStep11.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
            cExecStep11.Abort
        End If
    End If

    If Not cExecStep12 Is Nothing Then
        If
cExecStep12.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
            cExecStep12.Abort
        End If
    End If

    If Not cExecStep13 Is Nothing Then
        If cExecStep13.ExecuteStep.ParentStepId
= cTermInstance.Step.ParentStepId Then
            cExecStep13.Abort
        End If
    End If

    If Not cExecStep14 Is Nothing Then
        If cExecStep14.ExecuteStep.ParentStepId
= cTermInstance.Step.ParentStepId Then
            cExecStep14.Abort
        End If
    End If

    If Not cExecStep15 Is Nothing Then
        If cExecStep15.ExecuteStep.ParentStepId
= cTermInstance.Step.ParentStepId Then
            cExecStep15.Abort
        End If
    End If

    If Not cExecStep16 Is Nothing Then
        If cExecStep16.ExecuteStep.ParentStepId
= cTermInstance.Step.ParentStepId Then
            cExecStep16.Abort
        End If
    End If

    If Not cExecStep17 Is Nothing Then
        If cExecStep17.ExecuteStep.ParentStepId
= cTermInstance.Step.ParentStepId Then
            cExecStep17.Abort
        End If
    End If

    If Not cExecStep18 Is Nothing Then
        If cExecStep18.ExecuteStep.ParentStepId
= cTermInstance.Step.ParentStepId Then
            cExecStep18.Abort
        End If
    End If

    If Not cExecStep19 Is Nothing Then
        If cExecStep19.ExecuteStep.ParentStepId
= cTermInstance.Step.ParentStepId Then
            cExecStep19.Abort
        End If
    End If

    If Not cExecStep20 Is Nothing Then
        If cExecStep20.ExecuteStep.ParentStepId
= cTermInstance.Step.ParentStepId Then
            cExecStep20.Abort
        End If
    End If

    If Not cExecStep21 Is Nothing Then
        If cExecStep21.ExecuteStep.ParentStepId
= cTermInstance.Step.ParentStepId Then
            cExecStep21.Abort
        End If
    End If

    If Not cExecStep22 Is Nothing Then
        If cExecStep22.ExecuteStep.ParentStepId
= cTermInstance.Step.ParentStepId Then
            cExecStep22.Abort
        End If
    End If

    If Not cExecStep23 Is Nothing Then
        If cExecStep23.ExecuteStep.ParentStepId
= cTermInstance.Step.ParentStepId Then
            cExecStep23.Abort
        End If
    End If

    If Not cExecStep24 Is Nothing Then
        If cExecStep24.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
            cExecStep24.Abort
        End If
    End If

    If Not cExecStep25 Is Nothing Then
        If cExecStep25.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
            cExecStep25.Abort
        End If
    End If

    If Not cExecStep26 Is Nothing Then
        If cExecStep26.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
            cExecStep26.Abort
        End If
    End If

    If Not cExecStep27 Is Nothing Then
        If cExecStep27.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
            cExecStep27.Abort
        End If
    End If

    If Not cExecStep28 Is Nothing Then
        If cExecStep28.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
            cExecStep28.Abort
        End If
    End If

    If Not cExecStep29 Is Nothing Then
        If cExecStep29.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
            cExecStep29.Abort
        End If
    End If

    ' ============ 30 =================
    If Not cExecStep30 Is Nothing Then
        If cExecStep30.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
            cExecStep30.Abort
        End If
    End If

    If Not cExecStep31 Is Nothing Then
        If cExecStep31.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
            cExecStep31.Abort
        End If
    End If

    If Not cExecStep32 Is Nothing Then
        If cExecStep32.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
            cExecStep32.Abort
        End If
    End If

    If Not cExecStep33 Is Nothing Then
        If cExecStep33.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
            cExecStep33.Abort
        End If
    End If

    If Not cExecStep34 Is Nothing Then
        If cExecStep34.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
        cExecStep34.Abort
      End If
    End If

    If Not cExecStep35 Is Nothing Then
      If
cExecStep35.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
        cExecStep35.Abort
      End If
    End If

    If Not cExecStep36 Is Nothing Then
      If
cExecStep36.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
        cExecStep36.Abort
      End If
    End If

    If Not cExecStep37 Is Nothing Then
      If
cExecStep37.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
        cExecStep37.Abort
      End If
    End If

    If Not cExecStep38 Is Nothing Then
      If
cExecStep38.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
        cExecStep38.Abort
      End If
    End If

    If Not cExecStep39 Is Nothing Then
      If
cExecStep39.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
        cExecStep39.Abort
      End If
    End If

    ' ============ 40
================
    If Not cExecStep40 Is Nothing Then
      If
cExecStep40.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
        cExecStep40.Abort
      End If
    End If

    If Not cExecStep41 Is Nothing Then
      If
cExecStep41.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
        cExecStep41.Abort
      End If
    End If

    If Not cExecStep42 Is Nothing Then
      If
cExecStep42.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
        cExecStep42.Abort
      End If
    End If

    If Not cExecStep43 Is Nothing Then
      If
cExecStep43.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
        cExecStep43.Abort
      End If
    End If

    If Not cExecStep44 Is Nothing Then
      If cExecStep44.ExecuteStep.ParentStepId
= cTermInstance.Step.ParentStepId Then
        cExecStep44.Abort
      End If
    End If

    If Not cExecStep45 Is Nothing Then
      If cExecStep45.ExecuteStep.ParentStepId
= cTermInstance.Step.ParentStepId Then
        cExecStep45.Abort
      End If
    End If

    If Not cExecStep46 Is Nothing Then
      If cExecStep46.ExecuteStep.ParentStepId
= cTermInstance.Step.ParentStepId Then
        cExecStep46.Abort
      End If
    End If

    If Not cExecStep47 Is Nothing Then
      If cExecStep47.ExecuteStep.ParentStepId
= cTermInstance.Step.ParentStepId Then
        cExecStep47.Abort
      End If
    End If

    If Not cExecStep48 Is Nothing Then
      If cExecStep48.ExecuteStep.ParentStepId
= cTermInstance.Step.ParentStepId Then
        cExecStep48.Abort
      End If
    End If

    If Not cExecStep49 Is Nothing Then
      If cExecStep49.ExecuteStep.ParentStepId
= cTermInstance.Step.ParentStepId Then
        cExecStep49.Abort
      End If
    End If

    ' ============ 50
================
    If Not cExecStep50 Is Nothing Then
      If cExecStep50.ExecuteStep.ParentStepId
= cTermInstance.Step.ParentStepId Then
        cExecStep50.Abort
      End If
    End If

    If Not cExecStep51 Is Nothing Then
      If cExecStep51.ExecuteStep.ParentStepId
= cTermInstance.Step.ParentStepId Then
        cExecStep51.Abort
      End If
    End If

    If Not cExecStep52 Is Nothing Then
      If cExecStep52.ExecuteStep.ParentStepId
= cTermInstance.Step.ParentStepId Then
        cExecStep52.Abort
      End If
    End If

    If Not cExecStep53 Is Nothing Then
      If cExecStep53.ExecuteStep.ParentStepId
= cTermInstance.Step.ParentStepId Then
        cExecStep53.Abort
      End If
    End If

    If Not cExecStep54 Is Nothing Then
      If cExecStep54.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
        cExecStep54.Abort
      End If
    End If

    If Not cExecStep55 Is Nothing Then
      If cExecStep55.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
        cExecStep55.Abort
      End If
    End If

    If Not cExecStep56 Is Nothing Then
      If cExecStep56.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
        cExecStep56.Abort
      End If
    End If

    If Not cExecStep57 Is Nothing Then
      If cExecStep57.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
        cExecStep57.Abort
      End If
    End If

    If Not cExecStep58 Is Nothing Then
      If cExecStep58.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
        cExecStep58.Abort
      End If
    End If

    If Not cExecStep59 Is Nothing Then
      If cExecStep59.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
        cExecStep59.Abort
      End If
    End If

    ' ============ 60 ================
    If Not cExecStep60 Is Nothing Then
      If cExecStep60.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
        cExecStep60.Abort
      End If
    End If

    If Not cExecStep61 Is Nothing Then
      If cExecStep61.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
        cExecStep61.Abort
      End If
    End If

    If Not cExecStep62 Is Nothing Then
      If cExecStep62.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
        cExecStep62.Abort
      End If
    End If

    If Not cExecStep63 Is Nothing Then
      If cExecStep63.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
        cExecStep63.Abort
      End If
    End If

    If Not cExecStep64 Is Nothing Then
      If cExecStep64.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
        cExecStep64.Abort
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
        End If
    End If

    If Not cExecStep65 Is Nothing Then
        If
cExecStep65.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
            cExecStep65.Abort
        End If
    End If

    If Not cExecStep66 Is Nothing Then
        If
cExecStep66.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
            cExecStep66.Abort
        End If
    End If

    If Not cExecStep67 Is Nothing Then
        If
cExecStep67.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
            cExecStep67.Abort
        End If
    End If

    If Not cExecStep68 Is Nothing Then
        If
cExecStep68.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
            cExecStep68.Abort
        End If
    End If

    If Not cExecStep69 Is Nothing Then
        If
cExecStep69.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
            cExecStep69.Abort
        End If
    End If

    ' ============ 70
=================
    If Not cExecStep70 Is Nothing Then
        If
cExecStep70.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
            cExecStep70.Abort
        End If
    End If

    If Not cExecStep71 Is Nothing Then
        If
cExecStep71.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
            cExecStep71.Abort
        End If
    End If

    If Not cExecStep72 Is Nothing Then
        If
cExecStep72.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
            cExecStep72.Abort
        End If
    End If

    If Not cExecStep73 Is Nothing Then
        If
cExecStep73.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
            cExecStep73.Abort
        End If
    End If

    If Not cExecStep74 Is Nothing Then
        If cExecStep74.ExecuteStep.ParentStepId
= cTermInstance.Step.ParentStepId Then
            cExecStep74.Abort
        End If
    End If

    If Not cExecStep75 Is Nothing Then
        If cExecStep75.ExecuteStep.ParentStepId
= cTermInstance.Step.ParentStepId Then
            cExecStep75.Abort
        End If
    End If

    If Not cExecStep76 Is Nothing Then
        If cExecStep76.ExecuteStep.ParentStepId
= cTermInstance.Step.ParentStepId Then
            cExecStep76.Abort
        End If
    End If

    If Not cExecStep77 Is Nothing Then
        If cExecStep77.ExecuteStep.ParentStepId
= cTermInstance.Step.ParentStepId Then
            cExecStep77.Abort
        End If
    End If

    If Not cExecStep78 Is Nothing Then
        If cExecStep78.ExecuteStep.ParentStepId
= cTermInstance.Step.ParentStepId Then
            cExecStep78.Abort
        End If
    End If

    If Not cExecStep79 Is Nothing Then
        If cExecStep79.ExecuteStep.ParentStepId
= cTermInstance.Step.ParentStepId Then
            cExecStep79.Abort
        End If
    End If

    ' ============ 80
=================
    If Not cExecStep80 Is Nothing Then
        If cExecStep80.ExecuteStep.ParentStepId
= cTermInstance.Step.ParentStepId Then
            cExecStep80.Abort
        End If
    End If

    If Not cExecStep81 Is Nothing Then
        If cExecStep81.ExecuteStep.ParentStepId
= cTermInstance.Step.ParentStepId Then
            cExecStep81.Abort
        End If
    End If

    If Not cExecStep82 Is Nothing Then
        If cExecStep82.ExecuteStep.ParentStepId
= cTermInstance.Step.ParentStepId Then
            cExecStep82.Abort
        End If
    End If

    If Not cExecStep83 Is Nothing Then
        If cExecStep83.ExecuteStep.ParentStepId
= cTermInstance.Step.ParentStepId Then
            cExecStep83.Abort
        End If
    End If

    If Not cExecStep84 Is Nothing Then
        If cExecStep84.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
            cExecStep84.Abort
        End If
    End If

    If Not cExecStep85 Is Nothing Then
        If cExecStep85.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
            cExecStep85.Abort
        End If
    End If

    If Not cExecStep86 Is Nothing Then
        If cExecStep86.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
            cExecStep86.Abort
        End If
    End If

    If Not cExecStep87 Is Nothing Then
        If cExecStep87.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
            cExecStep87.Abort
        End If
    End If

    If Not cExecStep88 Is Nothing Then
        If cExecStep88.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
            cExecStep88.Abort
        End If
    End If

    If Not cExecStep89 Is Nothing Then
        If cExecStep89.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
            cExecStep89.Abort
        End If
    End If

    ' ============ 90 =================
    If Not cExecStep90 Is Nothing Then
        If cExecStep90.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
            cExecStep90.Abort
        End If
    End If

    If Not cExecStep91 Is Nothing Then
        If cExecStep91.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
            cExecStep91.Abort
        End If
    End If

    If Not cExecStep92 Is Nothing Then
        If cExecStep92.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
            cExecStep92.Abort
        End If
    End If

    If Not cExecStep93 Is Nothing Then
        If cExecStep93.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
            cExecStep93.Abort
        End If
    End If

    If Not cExecStep94 Is Nothing Then
        If cExecStep94.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
            cExecStep94.Abort
        End If
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
    End If

    If Not cExecStep95 Is Nothing Then
        If
cExecStep95.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
            cExecStep95.Abort
        End If
    End If

    If Not cExecStep96 Is Nothing Then
        If
cExecStep96.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
            cExecStep96.Abort
        End If
    End If

    If Not cExecStep97 Is Nothing Then
        If
cExecStep97.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
            cExecStep97.Abort
        End If
    End If

    If Not cExecStep98 Is Nothing Then
        If
cExecStep98.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
            cExecStep98.Abort
        End If
    End If

    If Not cExecStep99 Is Nothing Then
        If
cExecStep99.ExecuteStep.ParentStepId =
cTermInstance.Step.ParentStepId Then
            cExecStep99.Abort
        End If
    End If

    Exit Sub

AbortSiblingsErr:
    Call LogErrors(Errors)
    On Error GoTo 0
    ShowError errAbortFailed
    ' Try to abort the remaining steps, if any
    Resume Next

End Sub
Private Sub ExecutionFailed(cTermStep As
cRunStep)
    ' Called when execution of a step fails for
any reason - ensure that execution
    ' continues

    On Error GoTo ExecutionFailedErr

    Call AddFreeProcess(cTermStep.Index)

    Call RunBranch(mstrCurBranchRoot)

    Exit Sub

ExecutionFailedErr:
    ' Log the error code raised by Visual
Basic - do not raise an error here!
    Call LogErrors(Errors)

End Sub
Private Sub FreeExecStep(lngIndex As
Long)
    ' Frees an instance of a cExecuteSM object
depending on the index
    On Error GoTo FreeExecStepErr

    Select Case lngIndex + 1
        Case 1
            Set cExecStep1 = Nothing
        Case 2
            Set cExecStep2 = Nothing
        Case 3
            Set cExecStep3 = Nothing
        Case 4
            Set cExecStep4 = Nothing
        Case 5
            Set cExecStep5 = Nothing
        Case 6
            Set cExecStep6 = Nothing
        Case 7
            Set cExecStep7 = Nothing
        Case 8
            Set cExecStep8 = Nothing
        Case 9
            Set cExecStep9 = Nothing
        Case 10
            Set cExecStep10 = Nothing
        Case 11
            Set cExecStep11 = Nothing
        Case 12
            Set cExecStep12 = Nothing
        Case 13
            Set cExecStep13 = Nothing
        Case 14
            Set cExecStep14 = Nothing
        Case 15
            Set cExecStep15 = Nothing
        Case 16
            Set cExecStep16 = Nothing
        Case 17
            Set cExecStep17 = Nothing
        Case 18
            Set cExecStep18 = Nothing
        Case 19
            Set cExecStep19 = Nothing
        Case 20
            Set cExecStep20 = Nothing
        Case 21
            Set cExecStep21 = Nothing
        Case 22
            Set cExecStep22 = Nothing
        Case 23
            Set cExecStep23 = Nothing
        Case 24
            Set cExecStep24 = Nothing
        Case 25
            Set cExecStep25 = Nothing
        Case 26
            Set cExecStep26 = Nothing
        Case 27
            Set cExecStep27 = Nothing
        Case 28
            Set cExecStep28 = Nothing
        Case 29
            Set cExecStep29 = Nothing
        Case 30
            Set cExecStep30 = Nothing
        Case 31
            Set cExecStep31 = Nothing
        Case 32
            Set cExecStep32 = Nothing
        Case 33
            Set cExecStep33 = Nothing
        Case 34
            Set cExecStep34 = Nothing
        Case 35
            Set cExecStep35 = Nothing
        Case 36
            Set cExecStep36 = Nothing
        Case 37
            Set cExecStep37 = Nothing
        Case 38
            Set cExecStep38 = Nothing
        Case 39
            Set cExecStep39 = Nothing
        Case 40
            Set cExecStep40 = Nothing
        Case 41
            Set cExecStep41 = Nothing
        Case 42
            Set cExecStep42 = Nothing
        Case 43
            Set cExecStep43 = Nothing
        Case 44
            Set cExecStep44 = Nothing
        Case 45
            Set cExecStep45 = Nothing
        Case 46
            Set cExecStep46 = Nothing
        Case 47
            Set cExecStep47 = Nothing
        Case 48
            Set cExecStep48 = Nothing
        Case 49
            Set cExecStep49 = Nothing
        Case 50
            Set cExecStep50 = Nothing
        Case 51
            Set cExecStep51 = Nothing
        Case 52
            Set cExecStep52 = Nothing
        Case 53
            Set cExecStep53 = Nothing
        Case 54
            Set cExecStep54 = Nothing
        Case 55
            Set cExecStep55 = Nothing
        Case 56
            Set cExecStep56 = Nothing
        Case 57
            Set cExecStep57 = Nothing
        Case 58
            Set cExecStep58 = Nothing
        Case 59
            Set cExecStep59 = Nothing
        Case 60
            Set cExecStep60 = Nothing
        Case 61
            Set cExecStep61 = Nothing
        Case 62
            Set cExecStep62 = Nothing
        Case 63
            Set cExecStep63 = Nothing
        Case 64
            Set cExecStep64 = Nothing
        Case 65
            Set cExecStep65 = Nothing
        Case 66
            Set cExecStep66 = Nothing
        Case 67
            Set cExecStep67 = Nothing
        Case 68
            Set cExecStep68 = Nothing
        Case 69
            Set cExecStep69 = Nothing
        Case 70
            Set cExecStep70 = Nothing
        Case 71
            Set cExecStep71 = Nothing
        Case 72
            Set cExecStep72 = Nothing
        Case 73
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
        Set cExecStep73 = Nothing
      Case 74
        Set cExecStep74 = Nothing
      Case 75
        Set cExecStep75 = Nothing
      Case 76
        Set cExecStep76 = Nothing
      Case 77
        Set cExecStep77 = Nothing
      Case 78
        Set cExecStep78 = Nothing
      Case 79
        Set cExecStep79 = Nothing
      Case 80
        Set cExecStep80 = Nothing
      Case 81
        Set cExecStep81 = Nothing
      Case 82
        Set cExecStep82 = Nothing
      Case 83
        Set cExecStep83 = Nothing
      Case 84
        Set cExecStep84 = Nothing
      Case 85
        Set cExecStep85 = Nothing
      Case 86
        Set cExecStep86 = Nothing
      Case 87
        Set cExecStep87 = Nothing
      Case 88
        Set cExecStep88 = Nothing
      Case 89
        Set cExecStep89 = Nothing
      Case 90
        Set cExecStep90 = Nothing
      Case 91
        Set cExecStep91 = Nothing
      Case 92
        Set cExecStep92 = Nothing
      Case 93
        Set cExecStep93 = Nothing
      Case 94
        Set cExecStep94 = Nothing
      Case 95
        Set cExecStep95 = Nothing
      Case 96
        Set cExecStep96 = Nothing
      Case 97
        Set cExecStep97 = Nothing
      Case 98
        Set cExecStep98 = Nothing
      Case 99
        Set cExecStep99 = Nothing
      Case Else
        BugAssert False, "FreeExecStep:
Invalid index value!"
    End Select

    Exit Sub

FreeExecStepErr:
    ' Log the error code raised by Visual
Basic
    Call LogErrors(Errors)

End Sub
Private Sub ProcessAskFailures()
    ' This procedure is called when a step
with a continuation criteria = Ask has failed.
    ' Wait for all running processes to
complete before displaying an
Abort/Retry/Fail
    ' message to the user. We process every
Ask step that has failed and use a simple
    ' algorithm to determine what to do next.
    ' 1. An abort response to any failure results
in an immediate abort of the run
    ' 2. A continue means the run continues - this
failure is popped off the failure list.
    ' 3. A retry means that the execution details
for the instance are cleared and the
    '    step is re-executed.
    Dim lIndex As Long
    Dim cStepRec As cStep
    Dim cNextInst As cInstance
    Dim cFailureRec As cFailedStep

    On Error GoTo ProcessAskFailuresErr

    ' Display a popup message for all steps that
have failed with a continuation
    ' criteria of Ask
    For lIndex = mcFailures.Count - 1 To 0 Step
-1

        Set cFailureRec = mcFailures(lIndex)

        If cFailureRec.ContCriteria =
gintOnFailureAsk Then
            Set cStepRec =
mcRunSteps.QueryStep(cFailureRec.StepId)
            ' Ask the user whether to
abort/retry/continue
            #If RUN_ONLY Then
                cFailureRec.AskResponse =
ShowMessageBox(0, _
                    "Step '" &
GetStepNodeText(cStepRec) & "' failed. " & _
                    "Select Abort to abort run and
Ignore to continue. " & _
                    "Select Retry to re-execute the
failed step.", _
                    "Step Failure", _
                    MB_ABORTRETRYIGNORE
+ MB_APPLMODAL +
MB_ICONEXCLAMATION)
            #Else
                cFailureRec.AskResponse =
ShowMessageBox(frmRunning.hWnd, _
                    "Step '" &
GetStepNodeText(cStepRec) & "' failed. " & _
                    "Select Abort to abort run and
Ignore to continue. " & _
                    "Select Retry to re-execute the
failed step.", _
                    "Step Failure", _
                    MB_ABORTRETRYIGNORE
+ MB_APPLMODAL +
MB_ICONEXCLAMATION)
            #End If

            ' Process an abort response immediately
            If cFailureRec.AskResponse =
IDABORT Then
                mblnAbort = True
                Set cNextInst =
mcInstances.QueryInstance(cFailureRec.Instan
ceId)
                Call RunPendingSiblings(cNextInst,
cFailureRec.EndTime)
                Exit For
            End If
        End If

    Next lIndex

    ' Process all failed steps for which we have
Ignore and Retry responses.
    If Not mblnAbort Then
        ' Navigate in reverse order since we'll be
deleting items from the collection
        For lIndex = mcFailures.Count - 1 To 0 Step -1
            If mcFailures(lIndex).ContCriteria =
gintOnFailureAsk Then
                mblnAsk = False
                Set cFailureRec =
mcFailures.Delete(lIndex)

                Select Case cFailureRec.AskResponse
                    Case IDABORT
                        BugAssert True

                    Case IDRETRY
                        ' Delete all instances for the failed
step and re-try
                        ' Returns a parent instance reference
                        Set cNextInst =
ProcessRetryStep(cFailureRec)
                        Call
RunPendingStepInBranch(mstrCurBranchRoot,
cNextInst)

                    Case IDIGNORE
                        Set cNextInst =
mcInstances.QueryInstance(cFailureRec.InstanceId)
                        Call RunPendingSiblings(cNextInst,
cFailureRec.EndTime)

                End Select
            End If
        Next lIndex
    End If

    Exit Sub

ProcessAskFailuresErr:
    ' Log the error code raised by Visual Basic
    Call LogErrors(Errors)
    Err.Raise vbObjectError +
errExecuteBranchFailed, mstrModuleName, _
            LoadResString(errExecuteBranchFailed)

End Sub
Private Function ProcessRetryStep(cFailureRec As
cFailedStep) As cInstance
    ' This procedure is called when a step with a
continuation criteria = Ask has failed
    ' and the user wants to re-execute the step.
    ' We delete all existing instances for the step and
reset the iterator, if
    ' any on the parent instance - this way we ensure
that the step will be executed
    ' in the next pass.
    Dim lIndex As Long
    Dim cParentInstance As cInstance
    Dim cSubStepRec As cSubStep
    Dim cStepRec As cStep

    On Error GoTo ProcessRetryStepErr

    ' Navigate in reverse order since we'll be deleting
items from the collection
    For lIndex = mcInstances.Count - 1 To 0 Step -1

        If mcInstances(lIndex).Step.StepId =
cFailureRec.StepId Then
            Set cParentInstance =
mcInstances.QueryInstance(mcInstances(lIndex).Pa
rentInstanceId)
            Set cSubStepRec =
cParentInstance.QuerySubStep(cFailureRec.StepId)
            Set cStepRec =
mcRunSteps.QueryStep(cFailureRec.StepId)
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
        ' Decrement the child count on the
parent instance and reset the
        ' step iterators on the sub-step record,
if any -
        ' all the iterations of the step will be
re-executed.
        cParentInstance.ChildDeleted
cFailureRec.StepId
        cParentInstance.AllComplete = False
        cParentInstance.AllStarted = False

        cSubStepRec.InitializeIt cStepRec,
mcParameters

        ' Now delete the current instance
        Set ProcessRetryStep =
mcInstances.Delete(lIndex)
      End If
   Next lIndex

   Exit Function

ProcessRetryStepErr:
   ' Log the error code raised by Visual
Basic
   Call LogErrors(Errors)
   Err.Raise vbObjectError +
errExecuteBranchFailed, mstrModuleName,
_

LoadResString(errExecuteBranchFailed)

End Function


Private Sub RunNextStep(ByVal
dtmCompleteTime As Currency, ByVal
lngIndex As Long, _
      ByVal InstanceId As Long, ByVal
ExecutionStatus As InstanceStatus)
   ' Checks if there are any steps remaining
to be
   ' executed in the current branch. If so, it
executes
   ' the step.
   Dim cTermInstance As cInstance
   Dim cFailure As cFailedStep

   On Error GoTo RunNextStepErr

   BugMessage "RunNextStep: cExecStep"
& CStr(lngIndex + 1) & " has completed."

   Call mcTermSteps.Delete
   Call FreeExecStep(lngIndex)

   ' Call a procedure to add the freed up
object to the list
   Call AddFreeProcess(lngIndex)

   Set cTermInstance =
mcInstances.QueryInstance(InstanceId)
   cTermInstance.Status = ExecutionStatus

   If ExecutionStatus = gintFailed Then
      If
cTermInstance.Step.ContinuationCriteria =
gintOnFailureAbortSiblings Then
         Call AbortSiblings(cTermInstance)
      End If

      If Not
mcFailures.StepFailed(cTermInstance.Step.
StepId) Then
         Set cFailure = New cFailedStep
         cFailure.InstanceId =
cTermInstance.InstanceId
         cFailure.StepId =
cTermInstance.Step.StepId
         cFailure.ParentStepId =
cTermInstance.Step.ParentStepId
         cFailure.ContCriteria =
cTermInstance.Step.ContinuationCriteria
         cFailure.EndTime = dtmCompleteTime
         mcFailures.Add cFailure
         Set cFailure = Nothing
      End If
   End If

   If ExecutionStatus = gintFailed And
cTermInstance.Step.ContinuationCriteria =
gintOnFailureAbort Then
      If StringEmpty(msAbortDtls) Then
         ' Initialize the abort message
         msAbortDtls = "Step '" &
GetStepNodeText(cTermInstance.Step) & "'
failed. " & _
               "Aborting execution. Please check
the error file for details."
      End If
      Call Abort
   ElseIf ExecutionStatus = gintFailed And
cTermInstance.Step.ContinuationCriteria =
gintOnFailureAsk Then
      mblnAsk = True

      ' If the step failed due to a Cancel
operation (Abort), abort the run
      If mblnAbort Then
         Call
RunPendingSiblings(cTermInstance,
dtmCompleteTime)
      End If
   Else
      Call RunPendingSiblings(cTermInstance,
dtmCompleteTime)
   End If

   If mblnAbort Then
      If Not AnyStepRunning(mcFreeSteps,
mbarrFree) And Not
StringEmpty(msAbortDtls) Then
         ' Display an error only if the abort is
due to a failure
         ' We had to abort since a step failed -
since no other steps are currently
         ' running, we can display a message to
the user saying that we had to abort
         #If RUN_ONLY Then
            Call ShowMessageBox(0,
msAbortDtls, "Run Aborted", _
                  MB_APPLMODAL + MB_OK
+ MB_ICONEXCLAMATION)
         #Else
            Call
ShowMessageBox(frmRunning.hWnd,
msAbortDtls, "Run Aborted", _
                  MB_APPLMODAL + MB_OK
+ MB_ICONEXCLAMATION)
         #End If
'         MsgBox msAbortDtls, vbOKOnly,
"Run Aborted"
      End If
   ElseIf mblnAsk Then
      If Not AnyStepRunning(mcFreeSteps,
mbarrFree) Then
         ' Ask the user whether to
abort/retry/ignore failed steps
         Call ProcessAskFailures
      End If
   End If

   Exit Sub

RunNextStepErr:
   ' Log the error code raised by Visual Basic
   Call LogErrors(Errors)
   WriteError errExecuteBranchFailed, mstrSource
   Call ResetForm(lngIndex)

End Sub
Public Sub StopRun()

   ' Setting the Abort flag to True will ensure that
we
   ' don't execute any more steps
   mblnAbort = True

End Sub

Private Sub CreateDummyInstance(strRootKey As
String)

   Dim cNewInstance As cInstance
   Dim cSubStepDtls  As cStep
   Dim lngSubStepId As Long

   On Error GoTo CreateDummyInstanceErr

   ' Create a new instance of the step
   ' initialize substeps for the step
   Set cNewInstance = New cInstance

   ' There can be multiple iterations of the top level
nodes
   ' running at the same time, but only one branch at
any
   ' time - so enforce a degree of parallelism of 1 on
this
   ' node!
   Set cNewInstance.Step = New cStep
   cNewInstance.DegreeParallelism = 1
   cNewInstance.Key = mstrDummyRootKey

   cNewInstance.InstanceId = NewInstanceId
   cNewInstance.ParentInstanceId = 0

   lngSubStepId =
MakeIdentifierValid(strRootKey)

   Set cSubStepDtls =
mcRunSteps.QueryStep(lngSubStepId)
   If cSubStepDtls.EnabledFlag Then
      ' Create a child node for the step corresponding
to
      ' the root node of the branch being currently
executed,
      ' only if it has been enabled
      Call
cNewInstance.CreateSubStep(cSubStepDtls,
mcParameters)
   End If

   mcInstances.Add cNewInstance
   Set cNewInstance.Iterators =
DetermineIterators(cNewInstance)

   ' Set a reference to the newly created dummy
instance
   Set mcDummyRootInstance = cNewInstance

   Set cNewInstance = Nothing

   Exit Sub
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
CreateDummyInstanceErr:
   ' Log the error code raised by Visual
Basic
   Call LogErrors(Errors)
   On Error GoTo 0
   mstrSource = mstrModuleName &
"CreateDummyInstance"
   Err.Raise vbObjectError +
errCreateInstanceFailed, _
         mstrSource,
LoadResString(errCreateInstanceFailed)

End Sub
Private Function CreateInstance(cExecStep
As cStep, _
      cParentInstance As cInstance) As
cInstance
   ' Creates a new instance of the passed in
step. Returns
   ' a reference to the newly created instance
object.

   Dim cNewInstance As cInstance
   Dim nodChild As cStep
   Dim lngSubStepId As Long

   On Error GoTo CreateInstanceErr

   ' Create a new instance of the step
   ' initialize substeps for the step
   Set cNewInstance = New cInstance
   Set cNewInstance.Step = cExecStep
   cNewInstance.Key =
MakeKeyValid(cExecStep.StepId,
cExecStep.StepType)
   cNewInstance.ParentInstanceId =
cParentInstance.InstanceId
   cNewInstance.InstanceId =
NewInstanceId
   ' Validate the degree of parallelism field
before assigning it to the instance -
   ' (the parameter value might have been set
to an invalid value at runtime)
   Call
ValidateParallelism(cExecStep.DegreeParall
elism, _
         cExecStep.WorkspaceId,
ParamsInWsp:=mcParameters)
   cNewInstance.DegreeParallelism =
SubstituteParameters(cExecStep.DegreePara
llelism, _
         cExecStep.WorkspaceId,
WspParameters:=mcParameters)

   If
mcNavSteps.HasChild(StepKey:=cNewInsta
nce.Key) Then
      Set nodChild =
mcNavSteps.ChildStep(StepKey:=cNewInst
ance.Key)
      Do
         If nodChild.EnabledFlag Then
            ' Create nodes for all it's substeps
only
            ' if the substeps have been enabled
            Call
cNewInstance.CreateSubStep(nodChild,
mcParameters)
         End If

         Set nodChild =
mcNavSteps.NextStep(StepId:=nodChild.St
epId)
      Loop While (Not nodChild Is Nothing)
   End If
   mcInstances.Add cNewInstance
   Set cNewInstance.Iterators =
DetermineIterators(cNewInstance)

   ' Increment the number of executing steps on
the parent
   cParentInstance.ChildExecuted
(cExecStep.StepId)

   Set CreateInstance = cNewInstance

   Exit Function

CreateInstanceErr:
   ' Log the error code raised by Visual Basic
   Call LogErrors(Errors)
   On Error GoTo 0
   mstrSource = mstrModuleName &
"CreateInstance"
   Err.Raise vbObjectError +
errCreateInstanceFailed, _
         mstrSource,
LoadResString(errCreateInstanceFailed)

End Function
Private Function
DetermineIterators(cInstanceRec As cInstance)
As cRunColIt
   ' Returns a collection of all the iterator
values for this
   ' instance - since an iterator that is defined at
a
   ' particular level can be used in all it's
substeps, we
   ' need to navigate the step tree all the way to
the root

   Dim cRunIts As cRunColIt
   Dim cRunIt As cRunItNode
   Dim cStepIt As cIterator
   Dim cParentInst As cInstance
   Dim cSubStepRec As cSubStep
   Dim cSubStepDtls As cStep
   Dim lngSubStepId As Long
   Dim lngIndex As Long

   On Error GoTo DetermineIteratorsErr

   Set cRunIts = New cRunColIt

   If cInstanceRec.ParentInstanceId > 0 Then
      ' The last iterator for an instance of a step
is stored
      ' on it's parent! So navigate up before
beginning the
      ' search for iterator values.
      Set cParentInst =
mcInstances.QueryInstance(cInstanceRec.Pare
ntInstanceId)

      ' Get the sub-step record for the current
step
      ' on it's parent's instance!
      lngSubStepId = cInstanceRec.Step.StepId
      Set cSubStepRec =
cParentInst.QuerySubStep(lngSubStepId)
      Set cSubStepDtls =
mcRunSteps.QueryStep(lngSubStepId)

      ' And determine the next iteration value
for the
      ' substep in this instance
      Set cStepIt =
cSubStepRec.NewIteration(cSubStepDtls)
      If Not cStepIt Is Nothing Then
         ' Add the iterator details to the collection
since
         ' an iterator has been defined for the step
         Set cRunIt = New cRunItNode
         cRunIt.IteratorName =
cSubStepDtls.IteratorName
         cRunIt.Value =
SubstituteParameters(cStepIt.Value,
cSubStepDtls.WorkspaceId,
WspParameters:=mcParameters)
         cRunIt.StepId = cSubStepRec.StepId
         cRunIts.Push cRunIt
      End If

      ' Since the parent instance has all the iterators
upto
      ' that level, read them and push them on to the
stack for
      ' this instance
      For lngIndex = 0 To
cParentInst.Iterators.Count - 1
         Set cRunIt = cParentInst.Iterators(lngIndex)
         cRunIts.Push cRunIt
      Next lngIndex
   End If

   Set DetermineIterators = cRunIts

   Exit Function

DetermineIteratorsErr:
   ' Log the error code raised by Visual Basic
   Call LogErrors(Errors)
   On Error GoTo 0
   mstrSource = mstrModuleName &
"DetermineIterators"
   Err.Raise vbObjectError +
errExecInstanceFailed, _
         mstrSource,
LoadResString(errExecInstanceFailed)

End Function
Private Function
DetermineConstraints(cInstanceRec As cInstance, _
      intConsType As ConstraintType) As Variant
   ' Returns a collection of all the constraints for this
   ' instance of the passed in type - all the
constraints defined
   ' for the manager are executed first, followed by
those defined
   ' for the step. If a step has an iterator defined for
it, each
   ' constraint is executed only once.

   Dim cParentInst As cInstance
   Dim cTempInst As cInstance
   Dim vntConstraints As Variant
   Dim vntTempCons As Variant
   Dim cColConstraints() As Variant
   Dim lngConsCount As Long

   On Error GoTo DetermineConstraintsErr

   Set cTempInst = cInstanceRec
   lngConsCount = 0

   ' Go all the way to the root
   Do
      If cTempInst.ParentInstanceId > 0 Then
         Set cParentInst =
mcInstances.QueryInstance(cTempInst.ParentInstan
ceId)
      Else
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
        Set cParentInst = Nothing
      End If

      ' Check if the step has an iterator
defined for it
      If
cTempInst.ValidForIteration(cParentInst,
intConsType) Then
          vntTempCons =
mcRunConstraints.ConstraintsForStep( _
                cTempInst.Step.StepId,
cTempInst.Step.VersionNo, _
                intConsType, blnSort:=True, _
                blnGlobal:=False,
blnGlobalConstraintsOnly:=False)

          If Not IsEmpty(vntTempCons) Then
            ReDim Preserve
cColConstraints(lngConsCount)
            cColConstraints(lngConsCount) =
vntTempCons
            lngConsCount = lngConsCount +
1
          End If
      End If

      Set cTempInst = cParentInst

   Loop While Not cTempInst Is Nothing

   If lngConsCount > 0 Then
      vntTempCons =
OrderConstraints(cColConstraints,
intConsType)
   End If

   DetermineConstraints = vntTempCons

   Exit Function

DetermineConstraintsErr:
   ' Log the error code raised by Visual
Basic
   Call LogErrors(Errors)
   On Error GoTo 0
   mstrSource = mstrModuleName &
"DetermineConstraints"
   Err.Raise vbObjectError +
errExecInstanceFailed, _
          mstrSource,
LoadResString(errExecInstanceFailed)

End Function
Private Function
GetInstanceToExecute(cParentNode As
cInstance, _
      cSubStepRec As cSubStep, _
      cSubStepDtls As cStep) As cInstance

   Dim cSubStepInst As cInstance

   On Error GoTo GetInstanceToExecuteErr

   BugAssert Not (cParentNode Is Nothing
Or _
          cSubStepRec Is Nothing Or _
          cSubStepDtls Is Nothing), _
          "GetInstanceToExecute: Input
invalid"

   ' Check if it has iterators
   If cSubStepDtls.IteratorCount = 0 Then
      ' Check if the step has been executed
      If cSubStepRec.TasksRunning = 0 And
cSubStepRec.TasksComplete = 0 And _
            Not
mcInstances.CompletedInstanceExists(cParent
Node.InstanceId, cSubStepDtls) Then
          ' The sub-step hasn't been executed yet.
          ' Create an instance for it and exit
          Set cSubStepInst =
CreateInstance(cSubStepDtls, cParentNode)
      Else
          Set cSubStepInst = Nothing
      End If
   Else
      ' Check if there are pending iterations for
the sub-step
      If Not
cSubStepRec.NextIteration(cSubStepDtls) Is
Nothing Then
          ' Pending iterations exist - create an
instance for the sub-step and exit
          Set cSubStepInst =
CreateInstance(cSubStepDtls, cParentNode)
      Else
          ' No more iterations - continue with the
next substep
          Set cSubStepInst = Nothing
      End If
   End If

   Set GetInstanceToExecute = cSubStepInst
   Exit Function

GetInstanceToExecuteErr:
   ' Log the error code raised by Visual Basic
   Call LogErrors(Errors)
   On Error GoTo 0
   mstrSource = mstrModuleName &
"GetInstanceToExecute"
   Err.Raise vbObjectError +
errNavInstancesFailed, _
          mstrSource,
LoadResString(errNavInstancesFailed)

End Function

Public Function InstancesForStep(lngStepId
As Long, ByRef StepStatus As InstanceStatus)
As cInstances
   ' Returns an array of all the instances for a
step
   Dim lngIndex As Long
   Dim cTempInst As cInstance
   Dim cStepInstances As cInstances
   Dim cStepRec As cStep

   On Error GoTo InstancesForStepErr

   Set cStepInstances = New cInstances

   For lngIndex = 0 To mcInstances.Count - 1
      Set cTempInst = mcInstances(lngIndex)

      If cTempInst.Step.StepId = lngStepId
Then
          cStepInstances.Add cTempInst
      End If
   Next lngIndex

   If cStepInstances.Count = 0 Then
      Set cStepRec =
mcRunSteps.QueryStep(lngStepId)
      If Not
mcFailures.ExecuteSubStep(cStepRec.ParentSt
epId) Then
          StepStatus = gintAborted
      End If
      Set cStepRec = Nothing
   End If

   ' Set the return value of the function to the array
of
   ' constraints that has been built above
   Set InstancesForStep = cStepInstances

   Set cStepInstances = Nothing
   Exit Function

InstancesForStepErr:
   ' Log the error code raised by Visual Basic
   Call LogErrors(Errors)
   On Error GoTo 0
   mstrSource = mstrModuleName &
"InstancesForStep"
   Err.Raise vbObjectError +
errNavInstancesFailed, mstrSource, _
          LoadResString(errNavInstancesFailed)

End Function
Private Sub
RemoveFreeProcess(lngRunningProcess As Long)
   ' Removes the passed in element from the
collection of
   ' free objects

   ' Confirm that the last element in the array is the
one
   ' we need to delete
   If mcFreeSteps(mcFreeSteps.Count - 1) =
lngRunningProcess Then
      mcFreeSteps.Delete
Position:=mcFreeSteps.Count - 1
   Else
      ' Ask the class to find the element and delete it
      mcFreeSteps.Delete Item:=lngRunningProcess
   End If

End Sub
Private Sub AddFreeProcess(lngTerminatedProcess
As Long)
   ' Adds the passed in element to the collection of
   ' free objects

   mcFreeSteps.Add lngTerminatedProcess

End Sub

Private Sub ResetForm(Optional ByVal lngIndex
As Long)

   Dim lngTemp As Long

   On Error GoTo ResetFormErr

   ' Check if there are any running instances to wait
for
   If mcFreeSteps.Count <>
glngNumConcurrentProcesses Then

      For lngTemp = 0 To mcFreeSteps.Count - 1
         If mcFreeSteps(lngTemp) = lngIndex Then
            Exit For
         End If
      Next lngTemp

      If lngTemp <= mcFreeSteps.Count - 1 Then
         ' This process that just completed did not
exist in the list of
         ' free processes
         Call AddFreeProcess(lngIndex)
      End If
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
      If Not AnyStepRunning(mcFreeSteps,
mbarrFree) Then
         WriteToWspLog
(mintRunComplete)
         ' All steps are complete
         RaiseEvent
RunComplete(Determine64BitTime())
      End If
   Else
      WriteToWspLog (mintRunComplete)
      RaiseEvent
RunComplete(Determine64BitTime())
   End If

   Exit Sub

ResetFormErr:

End Sub
Private Function NewInstanceId() As Long
   ' Will return new instance id's - uses a
static counter
   ' that it increments each time
   Static lngInstance As Long

   lngInstance = lngInstance + 1
   NewInstanceId = lngInstance

End Function

Private Function
RunPendingStepInBranch(strCurBranchRoo
t As String, _
      Optional cExecInstance As cInstance =
Nothing) As cInstance
   ' Runs a worker step in the branch being
executed, if
   ' there are any pending execution
   ' This function is also called when a step
has just completed
   ' execution - in which case the terminated
instance is
   ' passed in as the optional parameter.
When that happens,
   ' we first try to execute the siblings of the
terminated
   ' step if any are pending execution.
   ' If the terminated instance has not been
passed in, we
   ' start with the dummy root instance and
navigate down,
   ' trying to find a pending worker step.

   Dim cExecSubStep As cStep
   Dim cParentInstance As cInstance
   Dim cNextInst As cInstance

   On Error GoTo
RunPendingStepInBranchErr

   If Not cExecInstance Is Nothing Then
      ' Called when an instance has
terminated
      ' When a worker step terminates, then
we need to
      ' decremement the number of running
steps on it's
      ' manager
      Set cParentInstance = _

mcInstances.QueryInstance(cExecInstance.P
arentInstanceId)

   Else
      If StringEmpty(strCurBranchRoot) Or
mcDummyRootInstance Is Nothing Then
         ' Run complete - event raised by Run
method
         Set RunPendingStepInBranch =
Nothing
         Exit Function
      End If

      ' If there are no pending steps on the root
instance,
      ' then there are no steps within the branch
that need
      ' to be executed
      If mcDummyRootInstance.AllComplete
Or mcDummyRootInstance.AllStarted Then
         Set RunPendingStepInBranch =
Nothing
         Exit Function
      End If

      Set cParentInstance =
mcDummyRootInstance
   End If

   Do
      Set cNextInst =
GetSubStepToExecute(cParentInstance)
      If cNextInst Is Nothing Then
         ' There are no steps within the branch
that can
         ' be executed - If we are at the dummy
instance,
         ' this branch has completed executing
         If cParentInstance.Key =
mstrDummyRootKey Then
            Set cNextInst = Nothing
            Exit Do
         Else
            ' Go to the parent instance and try to
find
            ' some other sibling is pending
execution
            Set cNextInst =
mcInstances.QueryInstance(cParentInstance.Pa
rentInstanceId)

            If cParentInstance.SubSteps.Count =
0 Then
               cNextInst.ChildTerminated
cParentInstance.Step.StepId
            End If
         End If
      End If

      BugAssert Not cNextInst Is Nothing
      Set cParentInstance = cNextInst

   Loop While cNextInst.Step.StepType <>
gintWorkerStep

   If Not cNextInst Is Nothing Then
      Call ExecuteStep(cNextInst)
   End If

   Set RunPendingStepInBranch = cNextInst

   Exit Function

RunPendingStepInBranchErr:
   ' Log the error code raised by Visual Basic
   Call LogErrors(Errors)
   On Error GoTo 0
   Err.Raise vbObjectError +
errNavInstancesFailed, _
         mstrModuleName &
"RunPendingStepInBranch",
LoadResString(errNavInstancesFailed)

End Function
Private Function
RunPendingSibling(cTermInstance As cInstance, _
      dtmCompleteTime As Currency) As cInstance
   ' This process is called when a step terminates.
Tries to
   ' run a sibling of the terminated step, if one is
pending
   ' execution.

   Dim cParentInstance As cInstance
   Dim cNextInst As cInstance

   On Error GoTo RunPendingSiblingErr

   If StringEmpty(mstrCurBranchRoot) Or
mcDummyRootInstance Is Nothing Then
      ' Run complete - event raised by Run method
      Set RunPendingSibling = Nothing
      Exit Function
   End If

   BugAssert cTermInstance.ParentInstanceId > 0,
"Orphaned instance in array!"

   ' When a worker step terminates, then we need to
   ' decremement the number of running steps on it's
   ' manager
   Set cParentInstance =
mcInstances.QueryInstance(cTermInstance.ParentIn
stanceId)

   ' Decrement the number of running processes on
the
   ' parent by 1
   Call
cParentInstance.ChildTerminated(cTermInstance.St
ep.StepId)

   ' The first step that terminates has to be a worker
   ' If it is complete, update the completed steps on
the
   ' parent by 1.
   Call
cParentInstance.ChildCompleted(cTermInstance.St
ep.StepId)
   cParentInstance.AllStarted = False

   Do
      Set cNextInst =
GetSubStepToExecute(cParentInstance,
dtmCompleteTime)
      If cNextInst Is Nothing Then
         If cParentInstance.Key =
mstrDummyRootKey Then
            Set cNextInst = Nothing
            Exit Do
         Else
            ' Go to the parent instance and try to find
            ' some other sibling is pending execution
            Set cNextInst =
mcInstances.QueryInstance(cParentInstance.ParentI
nstanceId)
            If cParentInstance.IsRunning Then
               cNextInst.AllStarted = True
            Else
               ' No more sub-steps to execute
               Call
cNextInst.ChildCompleted(cParentInstance.Step.Ste
pId)
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
                Call
cNextInst.ChildTerminated(cParentInstance.
Step.StepId)
                cNextInst.AllStarted = False
            End If
        End If
    End If

    BugAssert Not cNextInst Is Nothing
    Set cParentInstance = cNextInst

  Loop While cNextInst.Step.StepType <>
gintWorkerStep

  If Not cNextInst Is Nothing Then
    Call ExecuteStep(cNextInst)
  End If

  Set RunPendingSibling = cNextInst

  Exit Function

RunPendingSiblingErr:
  ' Log the error code raised by Visual
Basic
  Call LogErrors(Errors)
  On Error GoTo 0
  mstrSource = mstrModuleName &
"RunPendingSibling"
  Err.Raise vbObjectError +
errNavInstancesFailed, mstrSource, _
    LoadResString(errNavInstancesFailed)

End Function
Private Sub
RunPendingSiblings(cTermInstance As
cInstance, _
    dtmCompleteTime As Currency)
  ' This process is called when a step
terminates. Tries to
  ' run siblings of the terminated step, if
they are pending
  ' execution.

  Dim cExecInst As cInstance

  On Error GoTo RunPendingSiblingsErr
  BugMessage "In RunPendingSiblings"

  ' Call a procedure to run the sibling of the
terminated
  ' step, if any. This procedure will also
update the
  ' number of complete/running tasks on the
manager steps.
  Set cExecInst =
RunPendingSibling(cTermInstance,
dtmCompleteTime)

  If Not cExecInst Is Nothing Then
    Do
      ' Execute any other pending steps in
the branch.
      ' The step that has just terminated
might be
      ' the last one that was executing in a
sub-branch.
      ' That would mean that we can
execute another
      ' sub-branch that might involve more
than 1 step.
      ' Pass the just executed step as a
parameter.
      Set cExecInst =
RunPendingStepInBranch(mstrCurBranchRoot
, cExecInst)
    Loop While Not cExecInst Is Nothing
  Else
    If Not mcDummyRootInstance.IsRunning
Then
      ' All steps have been executed in the
branch - run
      ' a new branch
      Call RunNewBranch
    Else
      ' There are no more steps to execute in
the current
      ' branch but we have running processes.
    End If
  End If

  Exit Sub

RunPendingSiblingsErr:
  ' Log the error code raised by Visual Basic
  Call LogErrors(Errors)
  On Error GoTo 0
  mstrSource = mstrModuleName &
"RunPendingSiblings"
  Err.Raise vbObjectError +
errNavInstancesFailed, _
      mstrSource,
LoadResString(errNavInstancesFailed)

End Sub

Private Sub
NoSubStepsToExecute(cMgrInstance As
cInstance, Optional dtmCompleteTime As
Currency = gdtmEmpty)
  ' Called when we cannot find any more
substeps to run for
  ' manager step - set the allcomplete or
allstarted
  ' properties to true

  If cMgrInstance.IsRunning() Then
    cMgrInstance.AllStarted = True
  Else
    cMgrInstance.AllComplete = True
    If dtmCompleteTime <> gdtmEmpty
Then
      ' Update the end time on the manager
step
      Call
TimeCompleteUpdateForStep(cMgrInstance,
dtmCompleteTime)
    End If
  End If

End Sub

Private Function
GetSubStepToExecute(cParentNode As
cInstance, _
    Optional dtmCompleteTime As Currency
= 0) As cInstance
  ' Returns the child of the passed in node that
is to be
  ' executed next. Checks if we are in the
middle of an instance
  ' being executed in which case it returns the
pending
  ' instance. Creates a new instance if there are
pending
  ' instances for a sub-step.

  Dim lngIndex As Long
  Dim cSubStepRec As cSubStep
  Dim cSubStepDtls As cStep
  Dim cSubStepInst As cInstance

  On Error GoTo GetSubStepToExecuteErr

  ' There are a number of cases that need to be
accounted
  ' for here.
  ' 1. While traversing through all enabled nodes
for the
  ' first time - instance records may not exist for the
  ' substeps.
  ' 2. Instance records exist, and there are processes
  ' that need to be executed for a sub-step
  ' 3. There are no more processes that need to be
currently
  ' executed (till a process completes)
  ' 4. There are no more processes that need to be
executed
  ' (All substeps have completed execution)

  ' This is the only point where we check the Abort
flag -
  ' since this is the heart of the navigation routine
that
  ' selects processes to execute. Also, when a step
terminates
  ' selection of the next process goes through here.
  If mblnAbort Then
    Set GetSubStepToExecute = Nothing
    cParentNode.Status = gintAborted
    Exit Function
  End If

  If mblnAsk Then
    Set GetSubStepToExecute = Nothing
    Exit Function
  End If

  If Not
mcFailures.ExecuteSubStep(cParentNode.Step.Step
Id) Then
    Set GetSubStepToExecute = Nothing
    cParentNode.Status = gintAborted
    Exit Function
  End If

  ' First check if there are pending steps for the
parent!
  If cParentNode.IsPending Then
    ' Loop through all the sub-steps for the parent
node
    For lngIndex = 0 To
cParentNode.SubSteps.Count - 1
      Set cSubStepRec =
cParentNode.SubSteps(lngIndex)
      Set cSubStepDtls =
mcRunSteps.QueryStep(cSubStepRec.StepId)
      If Not
mcInstances.InstanceAborted(cSubStepRec) Then
        ' Check if the sub-step is a worker
        If cSubStepDtls.StepType =
gintWorkerStep Then
          ' Find/create an instance to execute
          Set cSubStepInst =
GetInstanceToExecute( _
              cParentNode, cSubStepRec,
cSubStepDtls)
          If Not cSubStepInst Is Nothing Then
            Exit For
          Else
            ' Continue w/ the next sub-step
          End If
        Else
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
            ' The sub-step is a manager step
            ' Check if there are any pending
instances for
            ' the manager
            Set cSubStepInst =
mcInstances.QueryPendingInstance( _
                cParentNode.InstanceId,
cSubStepRec.StepId)
            If cSubStepInst Is Nothing Then
              ' Find/create an instance to
execute
              Set cSubStepInst =
GetInstanceToExecute( _
                  cParentNode,
cSubStepRec, cSubStepDtls)
              If Not cSubStepInst Is
Nothing Then
                Exit For
              Else
                ' Continue w/ the next sub-
step
              End If
            Else
              ' We have found a pending
instance for the
              ' sub-step (manager) - exit the
loop
              Exit For
            End If
          End If
        End If
      Next lngIndex

      If lngIndex >
cParentNode.SubSteps.Count - 1 Or
cParentNode.SubSteps.Count = 0 Then
        ' If we could not find any sub-steps
to execute,
        ' mark the parent node as
complete/all started
        Call
NoSubStepsToExecute(cParentNode,
dtmCompleteTime)
        Set cSubStepInst = Nothing
      End If
    End If

    Set GetSubStepToExecute = cSubStepInst
    Exit Function

GetSubStepToExecuteErr:
    ' Log the error code raised by Visual
Basic
    Call LogErrors(Errors)
    On Error GoTo 0
    mstrSource = mstrModuleName &
"GetSubStepToExecute"
    Err.Raise vbObjectError +
errNavInstancesFailed, mstrSource, _
      LoadResString(errNavInstancesFailed)

End Function

Private Sub
TimeCompleteUpdateForStep(cMgrInstance
As cInstance, ByVal EndTime As Currency)

    ' Called when there are no more sub-steps
to execute for
    ' the manager step. It updates the end time
and status on
    ' the manager.
    Dim lElapsed As Long
    On Error GoTo
TimeCompleteUpdateForStepErr

    If cMgrInstance.Key <>
mstrDummyRootKey Then
      cMgrInstance.EndTime = EndTime
      cMgrInstance.Status = gintComplete
      lElapsed = (EndTime -
cMgrInstance.StartTime) * 10000
      cMgrInstance.ElapsedTime = lElapsed
      RaiseEvent
StepComplete(cMgrInstance.Step, EndTime,
cMgrInstance.InstanceId, lElapsed)
    End If

    Exit Sub

TimeCompleteUpdateForStepErr:
    ' Log the error code raised by Visual Basic
    Call LogErrors(Errors)
    WriteError errUpdateDisplayFailed,
mstrModuleName &
"TimeCompleteUpdateForStep"

End Sub

Private Function GetFreeObject() As Long

    ' Check the array of free objects and retrieve
the first one
    If mcFreeSteps.Count > 0 Then
      GetFreeObject =
mcFreeSteps(mcFreeSteps.Count - 1)
    Else
      mstrSource = mstrModuleName &
"GetFreeObject"
      ShowError errMaxProcessesExceeded
      On Error GoTo 0
      Err.Raise vbObjectError +
errMaxProcessesExceeded, _
          mstrSource, _

LoadResString(errMaxProcessesExceeded)
    End If

End Function
Private Function
StepTerminated(cCompleteStep As cStep,
ByVal dtmCompleteTime As Currency, _
      ByVal lngIndex As Long, ByVal
InstanceId As Long, ByVal ExecutionStatus
As InstanceStatus) As cStep
    ' This procedure is called whenever a step
terminates.
    Dim cTermRec As cTermStep
    Dim cInstRec As cInstance
    Dim cStartInst As cInstance
    Dim lElapsed As Long
    Dim sLogLabel As String
    Dim LogLabels As New cVectorStr
    Dim iItIndex As Long

    On Error GoTo StepTerminatedErr

    Set cInstRec =
mcInstances.QueryInstance(InstanceId)
    If dtmCompleteTime <> 0 And
cInstRec.StartTime <> 0 Then
      ' Convert to milliseconds since that is the
default precision
      lElapsed = (dtmCompleteTime -
cInstRec.StartTime) * 10000
    Else
      lElapsed = 0
    End If
    Set cStartInst = cInstRec
    iItIndex = 0
    Do While cInstRec.Key <> mstrDummyRootKey
      sLogLabel = gstrSQ &
cInstRec.Step.StepLabel & gstrSQ

      If iItIndex < cInstRec.Iterators.Count Then
        If cStartInst.Iterators(iItIndex).StepId =
cInstRec.Step.StepId Then
          sLogLabel = sLogLabel & msIt & gstrSQ
& cStartInst.Iterators(iItIndex).IteratorName &
gstrSQ & _
              msItValue & gstrSQ &
cStartInst.Iterators(iItIndex).Value & gstrSQ
          iItIndex = iItIndex + 1
        End If
      End If

      If cInstRec.Key = cStartInst.Key Then
        ' Append the execution status
        sLogLabel = sLogLabel & " Status: " &
gstrSQ & gsExecutionStatus(ExecutionStatus) &
gstrSQ
        If ExecutionStatus = gintFailed Then
          ' Append the continuation criteria for the
step since it failed
          sLogLabel = sLogLabel & " Continuation
Criteria: " & gstrSQ &
gsContCriteria(cInstRec.Step.ContinuationCriteria)
& gstrSQ
        End If
      End If
      LogLabels.Add sLogLabel

      Set cInstRec =
mcInstances.QueryInstance(cInstRec.ParentInstanc
eId)
    Loop

    Call WriteToWspLog(mintStepComplete,
LogLabels, dtmCompleteTime)
    Set LogLabels = Nothing

    ' Adds the terminated step details to a queue.
    Set cTermRec = New cTermStep
    cTermRec.ExecutionStatus = ExecutionStatus
    cTermRec.Index = lngIndex
    cTermRec.InstanceId = InstanceId
    cTermRec.TimeComplete = dtmCompleteTime
    Call mcTermSteps.Add(cTermRec)
    Set cTermRec = Nothing

    RaiseEvent StepComplete(cCompleteStep,
dtmCompleteTime, InstanceId, lElapsed)

    Exit Function

StepTerminatedErr:
    ' Log the error code raised by Visual Basic
    Call LogErrors(Errors)
    WriteError errExecuteBranchFailed, mstrSource
    Call ResetForm(lngIndex)

End Function
Public Property Let RootKey(ByVal vdata As
String)

    mstrRootKey = vdata

End Property

Public Property Get RootKey() As String
    RootKey = mstrRootKey
End Property
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
Private Function InitExecStep() As
cRunStep
   ' Since arrays of objects cannot be
declared as WithEvents,
   ' we use a limited number of objects and
set a maximum
   ' on the number of steps that can run in
parallel
   ' This is a wrapper that will create an
instance of
   ' a cExecuteSM object depending on the
index
   Dim lngIndex As Long

   On Error GoTo InitExecStepErr

   lngIndex = GetFreeObject

   Select Case lngIndex + 1
      Case 1
         Set cExecStep1 = New cRunStep
         Set InitExecStep = cExecStep1
      Case 2
         Set cExecStep2 = New cRunStep
         Set InitExecStep = cExecStep2
      Case 3
         Set cExecStep3 = New cRunStep
         Set InitExecStep = cExecStep3
      Case 4
         Set cExecStep4 = New cRunStep
         Set InitExecStep = cExecStep4
      Case 5
         Set cExecStep5 = New cRunStep
         Set InitExecStep = cExecStep5
      Case 6
         Set cExecStep6 = New cRunStep
         Set InitExecStep = cExecStep6
      Case 7
         Set cExecStep7 = New cRunStep
         Set InitExecStep = cExecStep7
      Case 8
         Set cExecStep8 = New cRunStep
         Set InitExecStep = cExecStep8
      Case 9
         Set cExecStep9 = New cRunStep
         Set InitExecStep = cExecStep9
      Case 10
         Set cExecStep10 = New cRunStep
         Set InitExecStep = cExecStep10
      Case 11
         Set cExecStep11 = New cRunStep
         Set InitExecStep = cExecStep11
      Case 12
         Set cExecStep12 = New cRunStep
         Set InitExecStep = cExecStep12
      Case 13
         Set cExecStep13 = New cRunStep
         Set InitExecStep = cExecStep13
      Case 14
         Set cExecStep14 = New cRunStep
         Set InitExecStep = cExecStep14
      Case 15
         Set cExecStep15 = New cRunStep
         Set InitExecStep = cExecStep15
      Case 16
         Set cExecStep16 = New cRunStep
         Set InitExecStep = cExecStep16
      Case 17
         Set cExecStep17 = New cRunStep
         Set InitExecStep = cExecStep17
      Case 18
         Set cExecStep18 = New cRunStep
         Set InitExecStep = cExecStep18
      Case 19
         Set cExecStep19 = New cRunStep
         Set InitExecStep = cExecStep19
      Case 20
         Set cExecStep20 = New cRunStep
         Set InitExecStep = cExecStep20
      Case 21
         Set cExecStep21 = New cRunStep
         Set InitExecStep = cExecStep21
      Case 22
         Set cExecStep22 = New cRunStep
         Set InitExecStep = cExecStep22
      Case 23
         Set cExecStep23 = New cRunStep
         Set InitExecStep = cExecStep23
      Case 24
         Set cExecStep24 = New cRunStep
         Set InitExecStep = cExecStep24
      Case 25
         Set cExecStep25 = New cRunStep
         Set InitExecStep = cExecStep25
      Case 26
         Set cExecStep26 = New cRunStep
         Set InitExecStep = cExecStep26
      Case 27
         Set cExecStep27 = New cRunStep
         Set InitExecStep = cExecStep27
      Case 28
         Set cExecStep28 = New cRunStep
         Set InitExecStep = cExecStep28
      Case 29
         Set cExecStep29 = New cRunStep
         Set InitExecStep = cExecStep29
      Case 30
         Set cExecStep30 = New cRunStep
         Set InitExecStep = cExecStep30
      Case 31
         Set cExecStep31 = New cRunStep
         Set InitExecStep = cExecStep31
      Case 32
         Set cExecStep32 = New cRunStep
         Set InitExecStep = cExecStep32
      Case 33
         Set cExecStep33 = New cRunStep
         Set InitExecStep = cExecStep33
      Case 34
         Set cExecStep34 = New cRunStep
         Set InitExecStep = cExecStep34
      Case 35
         Set cExecStep35 = New cRunStep
         Set InitExecStep = cExecStep35
      Case 36
         Set cExecStep36 = New cRunStep
         Set InitExecStep = cExecStep36
      Case 37
         Set cExecStep37 = New cRunStep
         Set InitExecStep = cExecStep37
      Case 38
         Set cExecStep38 = New cRunStep
         Set InitExecStep = cExecStep38
      Case 39
         Set cExecStep39 = New cRunStep
         Set InitExecStep = cExecStep39
      Case 40
         Set cExecStep40 = New cRunStep
         Set InitExecStep = cExecStep40
      Case 41
         Set cExecStep41 = New cRunStep
         Set InitExecStep = cExecStep41
      Case 42
         Set cExecStep42 = New cRunStep
         Set InitExecStep = cExecStep42
      Case 43
         Set cExecStep43 = New cRunStep
         Set InitExecStep = cExecStep43
      Case 44
         Set cExecStep44 = New cRunStep
         Set InitExecStep = cExecStep44
      Case 45
         Set cExecStep45 = New cRunStep
         Set InitExecStep = cExecStep45
      Case 46
         Set cExecStep46 = New cRunStep
         Set InitExecStep = cExecStep46
      Case 47
         Set cExecStep47 = New cRunStep
         Set InitExecStep = cExecStep47
      Case 48
         Set cExecStep48 = New cRunStep
         Set InitExecStep = cExecStep48
      Case 49
         Set cExecStep49 = New cRunStep
         Set InitExecStep = cExecStep49
      Case 50
         Set cExecStep50 = New cRunStep
         Set InitExecStep = cExecStep50
      Case 51
         Set cExecStep51 = New cRunStep
         Set InitExecStep = cExecStep51
      Case 52
         Set cExecStep52 = New cRunStep
         Set InitExecStep = cExecStep52
      Case 53
         Set cExecStep53 = New cRunStep
         Set InitExecStep = cExecStep53
      Case 54
         Set cExecStep54 = New cRunStep
         Set InitExecStep = cExecStep54
      Case 55
         Set cExecStep55 = New cRunStep
         Set InitExecStep = cExecStep55
      Case 56
         Set cExecStep56 = New cRunStep
         Set InitExecStep = cExecStep56
      Case 57
         Set cExecStep57 = New cRunStep
         Set InitExecStep = cExecStep57
      Case 58
         Set cExecStep58 = New cRunStep
         Set InitExecStep = cExecStep58
      Case 59
         Set cExecStep59 = New cRunStep
         Set InitExecStep = cExecStep59
      Case 60
         Set cExecStep60 = New cRunStep
         Set InitExecStep = cExecStep60
      Case 61
         Set cExecStep61 = New cRunStep
         Set InitExecStep = cExecStep61
      Case 62
         Set cExecStep62 = New cRunStep
         Set InitExecStep = cExecStep62
      Case 63
         Set cExecStep63 = New cRunStep
         Set InitExecStep = cExecStep63
      Case 64
         Set cExecStep64 = New cRunStep
         Set InitExecStep = cExecStep64
      Case 65
         Set cExecStep65 = New cRunStep
         Set InitExecStep = cExecStep65
      Case 66
         Set cExecStep66 = New cRunStep
         Set InitExecStep = cExecStep66
      Case 67
         Set cExecStep67 = New cRunStep
         Set InitExecStep = cExecStep67
      Case 68
         Set cExecStep68 = New cRunStep
         Set InitExecStep = cExecStep68
      Case 69
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
        Set cExecStep69 = New cRunStep
        Set InitExecStep = cExecStep69
    Case 70
        Set cExecStep70 = New cRunStep
        Set InitExecStep = cExecStep70
    Case 71
        Set cExecStep71 = New cRunStep
        Set InitExecStep = cExecStep71
    Case 72
        Set cExecStep72 = New cRunStep
        Set InitExecStep = cExecStep72
    Case 73
        Set cExecStep73 = New cRunStep
        Set InitExecStep = cExecStep73
    Case 74
        Set cExecStep74 = New cRunStep
        Set InitExecStep = cExecStep74
    Case 75
        Set cExecStep75 = New cRunStep
        Set InitExecStep = cExecStep75
    Case 76
        Set cExecStep76 = New cRunStep
        Set InitExecStep = cExecStep76
    Case 77
        Set cExecStep77 = New cRunStep
        Set InitExecStep = cExecStep77
    Case 78
        Set cExecStep78 = New cRunStep
        Set InitExecStep = cExecStep78
    Case 79
        Set cExecStep79 = New cRunStep
        Set InitExecStep = cExecStep79
    Case 80
        Set cExecStep80 = New cRunStep
        Set InitExecStep = cExecStep80
    Case 81
        Set cExecStep81 = New cRunStep
        Set InitExecStep = cExecStep81
    Case 82
        Set cExecStep82 = New cRunStep
        Set InitExecStep = cExecStep82
    Case 83
        Set cExecStep83 = New cRunStep
        Set InitExecStep = cExecStep83
    Case 84
        Set cExecStep84 = New cRunStep
        Set InitExecStep = cExecStep84
    Case 85
        Set cExecStep85 = New cRunStep
        Set InitExecStep = cExecStep85
    Case 86
        Set cExecStep86 = New cRunStep
        Set InitExecStep = cExecStep86
    Case 87
        Set cExecStep87 = New cRunStep
        Set InitExecStep = cExecStep87
    Case 88
        Set cExecStep88 = New cRunStep
        Set InitExecStep = cExecStep88
    Case 89
        Set cExecStep89 = New cRunStep
        Set InitExecStep = cExecStep89
    Case 90
        Set cExecStep90 = New cRunStep
        Set InitExecStep = cExecStep90
    Case 91
        Set cExecStep91 = New cRunStep
        Set InitExecStep = cExecStep91
    Case 92
        Set cExecStep92 = New cRunStep
        Set InitExecStep = cExecStep92
    Case 93
        Set cExecStep93 = New cRunStep
        Set InitExecStep = cExecStep93
    Case 94
        Set cExecStep94 = New cRunStep
        Set InitExecStep = cExecStep94
    Case 95
        Set cExecStep95 = New cRunStep
        Set InitExecStep = cExecStep95
    Case 96
        Set cExecStep96 = New cRunStep
        Set InitExecStep = cExecStep96
    Case 97
        Set cExecStep97 = New cRunStep
        Set InitExecStep = cExecStep97
    Case 98
        Set cExecStep98 = New cRunStep
        Set InitExecStep = cExecStep98
    Case 99
        Set cExecStep99 = New cRunStep
        Set InitExecStep = cExecStep99
    Case Else
        Set InitExecStep = Nothing
  End Select

  BugMessage "Sending cExecStep" &
(lngIndex + 1) & "!"

  If Not InitExecStep Is Nothing Then
     InitExecStep.Index = lngIndex

     ' Remove this element from the collection
of free objects
     Call RemoveFreeProcess(lngIndex)
  End If

  Exit Function

InitExecStepErr:
  ' Log the error code raised by Visual Basic
  Call LogErrors(Errors)
  Set InitExecStep = Nothing

End Function
Public Sub Run()
  ' Calls procedures to build a list of all the
steps that
  ' need to be executed and to execute them
  ' Determines whether the run has
started/terminated and
  ' raises the Run Start and Complete events.
  Dim cTempStep As cStep

  On Error GoTo RunErr

  If StringEmpty(mstrRootKey) Then
     Call ShowError(errExecuteBranchFailed)
     On Error GoTo 0
     Err.Raise vbObjectError +
errExecuteBranchFailed, mstrModuleName &
"Run", _

LoadResString(errExecuteBranchFailed)
  Else
     ' Execute the first branch
     WriteToWspLog (mintRunStart)
     RaiseEvent
RunStart(Determine64BitTime(),
mcWspLog.FileName)

     If
mcNavSteps.HasChild(StepKey:=mstrRootKe
y) Then
        Set cTempStep =
mcNavSteps.ChildStep(StepKey:=mstrRootKe
y)
        mstrCurBranchRoot =
MakeKeyValid(cTempStep.StepId,
cTempStep.StepType)
        Call
CreateDummyInstance(mstrCurBranchRoot)

        ' Run all pending steps in the branch
        If Not RunBranch(mstrCurBranchRoot)
Then
           ' Execute a new branch if there aren't any
           ' steps to run
           Call RunNewBranch
        End If
     Else
        WriteToWspLog (mintRunComplete)
        ' No children to execute - the run is complete
        RaiseEvent
RunComplete(Determine64BitTime())
     End If
  End If

  Exit Sub

RunErr:
  ' Log the error code raised by Visual Basic
  Call LogErrors(Errors)
  Call ShowError(errExecuteBranchFailed,
OptArgs:=mstrCurBranchRoot)
  Call ResetForm

End Sub
Private Sub RunNewBranch()
  ' We will build a tree of all instances that occur
and
  ' the count of the sub-steps that are running will
be
  ' stored at each node in the tree (maintained
internally
  ' as an array). Since there can be multiple
iterations
  ' of the top level nodes running at the same time,
we
  ' create a dummy node at the root that keeps a
record of
  ' the instances of the top level node.

  ' Determines whether the run has
started/terminated and
  ' raises the Run Start and Complete events.
  Dim cNextStep As cStep
  Dim bRunComplete As Boolean

  On Error GoTo RunNewBranchErr

  bRunComplete = False

  Do
     If StringEmpty(mstrCurBranchRoot) Then
        Exit Do
'       On Error GoTo 0
'       Err.Raise vbObjectError +
errExecuteBranchFailed, mstrSource, _
'          LoadResString(errExecuteBranchFailed)
     Else
        Set cNextStep =
mcNavSteps.NextStep(StepKey:=mstrCurBranchRo
ot)
        If cNextStep Is Nothing Then
           mstrCurBranchRoot = gstrEmptyString
           bRunComplete = True
           Exit Do
        Else
           ' Starting execution of a new branch -
initialize the
           ' module-level variable
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
            mstrCurBranchRoot =
MakeKeyValid(cNextStep.StepId,
cNextStep.StepType)
            Call
CreateDummyInstance(mstrCurBranchRoot)
        End If
      End If
      Debug.Print "Running new branch: " &
mstrCurBranchRoot

   ' Loop until we find a branch that has
steps to execute
   Loop While Not
RunBranch(mstrCurBranchRoot)

   If bRunComplete Then
      WriteToWspLog (mintRunComplete)
      ' Run is complete
      RaiseEvent
RunComplete(Determine64BitTime())
   End If

   Exit Sub

RunNewBranchErr:
   ' Log the error code raised by Visual
Basic
   Call LogErrors(Errors)
   Call ShowError(errExecuteBranchFailed,
OptArgs:=mstrCurBranchRoot)
   On Error GoTo 0
   mstrSource = mstrModuleName &
"RunNewBranch"
   Err.Raise vbObjectError +
errExecuteBranchFailed, mstrSource, _

LoadResString(errExecuteBranchFailed)

End Sub
Private Function RunBranch(strRootNode
As String) As Boolean
   ' This procedure is called to run all the
necessary steps
   ' in a branch. It can also be called when a
step terminates,
   ' in which case the terminated step is
passed in as the
   ' optional parameter. When a step
terminates, we need to
   ' either wait for some other steps to
terminate before
   ' we execute more steps or run as many
steps as necessary
   ' Returns True if there are steps currently
executing
   ' in the branch, else returns False
   Dim cRunning As cInstance

   On Error GoTo RunBranchErr

   If Not StringEmpty(strRootNode) Then
      ' Call a procedure to execute all the
enabled steps
      ' in the branch - will return the step
node that is
      ' being executed - nothing means 'No
more steps to
      ' execute in the branch'.
      Do
         Set cRunning =
RunPendingStepInBranch(strRootNode,
cRunning)

      Loop While Not cRunning Is Nothing
      RunBranch =
mcDummyRootInstance.IsRunning
   End If

   Exit Function

RunBranchErr:
   ' Log the error code raised by Visual Basic
   Call LogErrors(Errors)
   On Error GoTo 0
   mstrSource = mstrModuleName &
"RunBranch"
   Err.Raise vbObjectError +
errExecuteBranchFailed, _
         mstrSource,
LoadResString(errExecuteBranchFailed)

End Function
Private Sub
TimeUpdateForProcess(StepRecord As cStep,
_
      ByVal InstanceId As Long, _
      Optional ByVal StartTime As Currency =
0, _
      Optional ByVal EndTime As Currency =
0, _
      Optional ByVal ElapsedTime As Long =
0, _
      Optional Command As String)
   ' We do not maintain start and end
timestamps for the constraint
   ' of a step. Hence we check if the process
that just started/
   ' terminated is the worker step that is being
executed. If so,
   ' we update the start/end time and status on
the instance record.

   Dim cInstanceRec As cInstance
   Dim sItVal As String

   On Error GoTo TimeUpdateForProcessErr

   Set cInstanceRec =
mcInstances.QueryInstance(InstanceId)

   If StartTime = 0 Then
      RaiseEvent ProcessComplete(StepRecord,
EndTime, InstanceId, ElapsedTime)
   Else
      sItVal =
GetInstanceItValue(cInstanceRec)
      RaiseEvent ProcessStart(StepRecord,
Command, StartTime, InstanceId, _
            cInstanceRec.ParentInstanceId,
sItVal)
   End If

   Call
cInstanceRec.UpdateStartTime(StepRecord.Ste
pId, StartTime, EndTime, ElapsedTime)

   Exit Sub

TimeUpdateForProcessErr:
   ' Log the error code raised by Visual Basic
   Call LogErrors(Errors)
   WriteError errUpdateDisplayFailed,
mstrModuleName & "TimeUpdateForProcess"

End Sub
Private Sub
TimeStartUpdateForStep(StepRecord As
cStep, _
      ByVal InstanceId As Long, _
      ByVal StartTime As Currency)

   ' Called when a step starts execution. Checks if
this is the
   ' first enabled child of the manager step. If so,
updates
   ' the start time and status on the manager.
   ' Also raises the Step Start event for the
completed step.

   Dim cStartInst As cInstance
   Dim cInstanceRec As cInstance
   Dim LogLabels As New cVectorStr
   Dim iItIndex As Long
   Dim sLogLabel As String
   Dim sPath As String
   Dim sIt As String
   Dim sItVal As String

   On Error GoTo TimeStartUpdateForStepErr

   Set cStartInst =
mcInstances.QueryInstance(InstanceId)

   ' Determine the step path and iterator values for
the step and raise a step start event
   Set cInstanceRec = cStartInst
   Do While cInstanceRec.Key <>
mstrDummyRootKey
      If Not StringEmpty(sPath) Then
         sPath = sPath & gstrFileSeparator
      End If
      sPath = sPath & gstrSQ &
cInstanceRec.Step.StepLabel & gstrSQ
      Set cInstanceRec =
mcInstances.QueryInstance(cInstanceRec.ParentIns
tanceId)
   Loop

   For iItIndex = cStartInst.Iterators.Count - 1 To 0
Step -1
      If Not StringEmpty(sIt) Then
         sIt = sIt & gstrFileSeparator
      End If
      sIt = sIt & gstrSQ &
cStartInst.Iterators(iItIndex).Value & gstrSQ
   Next iItIndex

   sItVal = GetInstanceItValue(cStartInst)
   RaiseEvent StepStart(StepRecord, StartTime,
InstanceId, cStartInst.ParentInstanceId, _
         sPath, sIt, sItVal)

   iItIndex = 0
   Set cInstanceRec = cStartInst
   ' Raise a StepStart event for the manager step, if
this is it's first sub-step being executed
   Do While cInstanceRec.Key <>
mstrDummyRootKey

      sLogLabel = gstrSQ &
cInstanceRec.Step.StepLabel & gstrSQ
      If iItIndex < cStartInst.Iterators.Count Then
         If cStartInst.Iterators(iItIndex).StepId =
cInstanceRec.Step.StepId Then
            sLogLabel = sLogLabel & msIt & gstrSQ
& cStartInst.Iterators(iItIndex).IteratorName &
gstrSQ & _
                msItValue & gstrSQ &
cStartInst.Iterators(iItIndex).Value & gstrSQ
            iItIndex = iItIndex + 1
         End If
      End If
      LogLabels.Add sLogLabel
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
      If cInstanceRec.Key <> cStartInst.Key
And cInstanceRec.StartTime = 0 Then
         cInstanceRec.StartTime = StartTime
         cInstanceRec.Status = gintRunning
         sItVal =
GetInstanceItValue(cInstanceRec)
         ' The step path and iterator values are
not needed for manager steps, since
         ' they are primarily used by the run
status form
         RaiseEvent
StepStart(cInstanceRec.Step, StartTime,
cInstanceRec.InstanceId, _
            cInstanceRec.ParentInstanceId,
gstrEmptyString, gstrEmptyString, _
            sItVal)
      End If

      Set cInstanceRec =
mcInstances.QueryInstance(cInstanceRec.Pa
rentInstanceId)
   Loop

   Call WriteToWspLog(mintStepStart,
LogLabels, StartTime)
   Set LogLabels = Nothing

   Exit Sub

TimeStartUpdateForStepErr:
   ' Log the error code raised by Visual
Basic
   Call LogErrors(Errors)
   WriteError errUpdateDisplayFailed,
mstrModuleName &
"TimeStartUpdateForStep"

End Sub
Private Sub WriteToWspLog(iLogEvent As
WspLogEvents, Optional StepDtls As
cVectorStr, _
      Optional dtStamp As Currency =
gdtmEmpty)

   ' Writes to the workspace log that is
generated for the run. The last three
   ' parameters are valid only for Step Start
and Step Complete events.
   Static bError As Boolean
   Dim sLabel As String
   Dim lIndex As Long
   Dim bHdr As Boolean
   Dim cTempConn As cConnection

   On Error GoTo WriteToWspLogErr

   Select Case iLogEvent
      Case mintRunStart
         Set mcWspLog = New cFileSM
         mcWspLog.FileName =
GetDefaultDir(WspId, mcParameters) &
gstrFileSeparator & _
            Trim(Str(RunId)) &
gstrFileSeparator & "SMLog-" &
Format(Now, FMT_WSP_LOG_FILE) &
gstrLogFileSuffix
         mcWspLog.WriteLine
(JulianDateToString(Determine64BitTime()
) & " Start Run: " & vbTab & gstrSQ &
GetWorkspaceDetails(WorkspaceId:=WspId
)) & gstrSQ

         ' Write all current parameter values
to the log
         bHdr = False
         For lIndex = 0 To
mcParameters.ParameterCount - 1
            If
mcParameters(lIndex).ParameterType <>
gintParameterApplication Then
               If Not bHdr Then
                  mcWspLog.WriteField
JulianDateToString(Determine64BitTime()) &
" Parameters: "
                  bHdr = True
               Else
                  mcWspLog.WriteField vbTab
& vbTab & vbTab
               End If
               mcWspLog.WriteLine vbTab &
gstrSQ &
mcParameters(lIndex).ParameterName &
gstrSQ & vbTab & vbTab & gstrSQ &
mcParameters(lIndex).ParameterValue &
gstrSQ
            End If
         Next lIndex

         ' Write all connection properties to the
log
         For lIndex = 0 To
RunConnections.Count - 1
            Set cTempConn =
RunConnections(lIndex)
            If lIndex = 0 Then
               mcWspLog.WriteField
JulianDateToString(Determine64BitTime()) &
" Connections: "
            Else
               mcWspLog.WriteField vbTab &
vbTab & vbTab
            End If
            mcWspLog.WriteLine vbTab &
gstrSQ & cTempConn.ConnectionName &
gstrSQ & _
               vbTab & vbTab & gstrSQ &
cTempConn.ConnectionValue & gstrSQ & _
               vbTab & "No Count: " &
gstrSQ & cTempConn.NoCountDisplay &
gstrSQ & gstrBlank & _
               "No Execute: " & gstrSQ &
cTempConn.NoExecute & gstrSQ & gstrBlank
& _
               "Parse Query Only: " & gstrSQ
& cTempConn.ParseQueryOnly & gstrSQ &
gstrBlank & _
               "Quoted Identifiers: " & gstrSQ
& cTempConn.QuotedIdentifiers & gstrSQ &
gstrBlank & _
               "ANSI Nulls: " & gstrSQ &
cTempConn.AnsiNulls & gstrSQ & gstrBlank
& _
               "Show Query Plan: " & gstrSQ
& cTempConn.ShowQueryPlan & gstrSQ &
gstrBlank & _
               "Show Stats Time: " & gstrSQ
& cTempConn.ShowStatsTime & gstrSQ &
gstrBlank & _
               "Show Stats IO: " & gstrSQ &
cTempConn.ShowStatsIO & gstrSQ &
gstrBlank & _
               "Row Count" & gstrSQ &
cTempConn.RowCount & gstrSQ & gstrBlank
& _
               "Query Timeout" & gstrSQ &
cTempConn.QueryTimeOut & gstrSQ
         Next lIndex

      Case mintRunComplete
         BugAssert Not mcWspLog Is Nothing
         mcWspLog.WriteLine
(JulianDateToString(Determine64BitTime()) &
" Comp. Run: " & vbTab & gstrSQ &
GetWorkspaceDetails(WorkspaceId:=WspId)) &
gstrSQ
         Set mcWspLog = Nothing

      Case mintStepStart
         For lIndex = StepDtls.Count - 1 To 0 Step -1
            sLabel = StepDtls(lIndex)
            If lIndex = StepDtls.Count - 1 Then
               mcWspLog.WriteLine
JulianDateToString(dtStamp) & " Start Step: " &
vbTab & sLabel
            Else
               mcWspLog.WriteLine vbTab & vbTab
& vbTab & vbTab & sLabel
            End If
         Next lIndex

      Case mintStepComplete
         For lIndex = StepDtls.Count - 1 To 0 Step -1
            sLabel = StepDtls(lIndex)
            If lIndex = StepDtls.Count - 1 Then
               mcWspLog.WriteLine
JulianDateToString(dtStamp) & " Comp. Step: " &
vbTab & sLabel
            Else
               mcWspLog.WriteLine vbTab & vbTab
& vbTab & vbTab & sLabel
            End If
         Next lIndex

   End Select

   Exit Sub

WriteToWspLogErr:
   If Not bError Then
      bError = True
   End If

End Sub
'Private Sub WriteToWspLog(iLogEvent As
WspLogEvents, Optional StepDtls As cVectorStr, _
'      Optional dtStamp As Date = gdtmEmpty)
'
'   This function uses the LogWriter dll - memory
corruption problems since the vb exe
'   and the vc Execute Dll both use the same dll to
write.
'   ' Writes to the workspace log that is generated
for the run. The last three
'   ' parameters are valid only for StepStart and
StepComplete events.
'   Static bError As Boolean
'   Static sFile As String
'   Dim sLabel As String
'   Dim lIndex As Long
'   Dim bHdr As Boolean
'
'   On Error GoTo WriteToWspLogErr
'
'   Select Case iLogEvent
'      Case mintRunStart
'         Set mcWspLog = New
LOGWRITERLib.SMLog
'         sFile = App.Path & "\" & "SMLog-" &
Format(Now, FMT_WSP_LOG_FILE) &
gstrLogFileSuffix
'         mcWspLog.FileName = sFile
'         mcWspLog.Init
'         mcWspLog.WriteLine (Format(Now,
FMT_WSP_LOG_DATE) & " Start Run: " &
vbTab & gstrSQ &
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
GetWorkspaceDetails(WorkspaceId:=WspId
)) & gstrSQ
'
'          ' Write all current parameter values
to the log
'          bHdr = False
'          For lIndex = 0 To
mcParameters.ParameterCount - 1
'              If
mcParameters(lIndex).ParameterType <>
gintParameterApplication Then
'                  If Not bHdr Then
'                      'mcWspLog.WriteLine
Format(Now, FMT_WSP_LOG_DATE) &
" Parameters: " & vbTab & gstrSQ &
mcParameters(lIndex).ParameterName &
gstrSQ & vbTab & vbTab & gstrSQ &
mcParameters(lIndex).ParameterValue &
gstrSQ
'                      bHdr = True
'                  Else
'                      'mcWspLog.WriteLine
vbTab & vbTab & vbTab & vbTab &
gstrSQ &
mcParameters(lIndex).ParameterName &
gstrSQ & vbTab & vbTab & gstrSQ &
mcParameters(lIndex).ParameterValue &
gstrSQ
'                  End If
'              End If
'          Next lIndex
'
'      Case mintRunComplete
'          BugAssert Not mcWspLog Is
Nothing
'          mcWspLog.WriteLine
(Format(Now, FMT_WSP_LOG_DATE) &
" Comp. Run: " & vbTab & gstrSQ &
GetWorkspaceDetails(WorkspaceId:=WspId
)) & gstrSQ
'          Set mcWspLog = Nothing
'
'      Case mintStepStart
'          For lIndex = StepDtls.Count - 1 To
0 Step -1
'              sLabel = StepDtls(lIndex)
'              If lIndex = StepDtls.Count - 1
Then
'                  mcWspLog.WriteLine
Format(dtStamp, FMT_WSP_LOG_DATE)
& " Start Step: " & vbTab & sLabel
'              Else
'                  mcWspLog.WriteLine vbTab
& vbTab & vbTab & vbTab & sLabel
'              End If
'          Next lIndex
'
'      Case mintStepComplete
'          For lIndex = StepDtls.Count - 1 To
0 Step -1
'              sLabel = StepDtls(lIndex)
'              If lIndex = StepDtls.Count - 1
Then
'                  mcWspLog.WriteLine
Format(dtStamp, FMT_WSP_LOG_DATE)
& " Comp. Step: " & vbTab & sLabel
'              Else
'                  mcWspLog.WriteLine vbTab
& vbTab & vbTab & vbTab & sLabel
'              End If
'          Next lIndex
'
'    End Select
'
'    Exit Sub
'
'WriteToWspLogErr:
'    If Not bError Then
'        bError = True
'    End If
'
'End Sub
'
Public Property Get WspPreExecution() As
Variant
    WspPreExecution = mcvntWspPreCons
End Property
Public Property Let WspPreExecution(ByVal
vdata As Variant)
    mcvntWspPreCons = vdata
End Property

Public Property Get WspPostExecution() As
Variant
    WspPostExecution = mcvntWspPostCons
End Property
Public Property Let WspPostExecution(ByVal
vdata As Variant)
    mcvntWspPostCons = vdata
End Property

Private Sub ExecuteStep(cCurStep As
cInstance)
    ' Initializes a cRunStep object with all the
properties
    ' corresponding to the step to be executed
and calls it's
    ' execute method to execute the step

    Dim cExecStep As cRunStep

    On Error GoTo ExecuteStepErr
    mstrSource = mstrModuleName &
"ExecuteStep"

    ' Confirm that the step is a worker
    If cCurStep.Step.StepType <>
gintWorkerStep Then
        On Error GoTo 0
        Err.Raise vbObjectError +
errExecInstanceFailed, mstrSource, _

LoadResString(errExecInstanceFailed)
    End If

    Set cExecStep = InitExecStep()
    ' Exceeded the number of processes that we
can run simultaneously
    If cExecStep Is Nothing Then
        ' Raise an error
        On Error GoTo 0
        Err.Raise vbObjectError +
errProgramError, mstrSource, _
            LoadResString(errProgramError)
    End If
    ' Initialize the instance id - not needed for
step execution
    ' but necessary to identify later which
instance completed
    cExecStep.InstanceId = cCurStep.InstanceId

    Set cExecStep.ExecuteStep = cCurStep.Step
    Set cExecStep.Iterators = cCurStep.Iterators
    Set cExecStep.Globals = mcRunSteps
    Set cExecStep.WspParameters =
mcParameters
    Set cExecStep.WspConnections =
RunConnections
    Set cExecStep.WspConnDtls =
RunConnDtls

    ' Initialize all the pre and post-execution
constraints that
    ' have been defined globally for the workspace
    cExecStep.WspPreCons = mcvntWspPreCons
    cExecStep.WspPostCons = mcvntWspPostCons

    ' Initialize all the pre and post-execution
constraints for
    ' the step being executed
    cExecStep.PreCons =
DetermineConstraints(cCurStep, gintPreStep)
    cExecStep.PostCons =
DetermineConstraints(cCurStep, gintPostStep)

    cExecStep.RunId = RunId
    cExecStep.CreateInputFiles = CreateInputFiles

    ' Call the execute method to execute the step
    cExecStep.Execute

    Set cExecStep = Nothing

    Exit Sub

ExecuteStepErr:
    ' Log the error code raised by Visual Basic
    Call LogErrors(Errors)
    On Error GoTo 0
    Call ExecutionFailed(cExecStep)

End Sub

Public Property Set Steps(cRunSteps As cArrSteps)

    Set mcRunSteps = cRunSteps
    Set mcNavSteps.StepRecords = cRunSteps

End Property
Public Property Set Parameters(cParameters As
cArrParameters)
    ' A reference to the parameter array - we use it to
    ' substitute parameter values in the step text

    Set mcParameters = cParameters

End Property
Public Property Get Steps() As cArrSteps

    Set Steps = mcRunSteps

End Property
Public Property Get Constraints() As
cArrConstraints

    Set Constraints = mcRunConstraints

End Property
Public Property Set Constraints(vdata As
cArrConstraints)

    Set mcRunConstraints = vdata

End Property


Private Sub
cExecStep1_ProcessComplete(cStepRecord As
cStep, _
        dtmEndTime As Currency, lngInstanceId As
Long, lElapsed As Long)

    Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
End Sub


Private Sub
cExecStep1_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime
As Currency, lngInstanceId As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub


Private Sub
cExecStep1_StepComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, InstanceId
As Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep1.Index, InstanceId,
Status)

End Sub

Private Sub
cExecStep1_StepStart(cStepRecord As
cStep, _
      dtmStartTime As Currency, InstanceId
As Long)

   Call
TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep9_ProcessComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency,
lngInstanceId As Long, lElapsed As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub


Private Sub
cExecStep9_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime
As Currency, lngInstanceId As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub


Private Sub
cExecStep9_StepComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, InstanceId
As Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep9.Index, InstanceId,
Status)

End Sub

Private Sub
cExecStep9_StepStart(cStepRecord As cStep,
_
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep10_ProcessComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, lngInstanceId
As Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub


Private Sub
cExecStep10_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub


Private Sub
cExecStep10_StepComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep10.Index, InstanceId,
Status)

End Sub

Private Sub
cExecStep10_StepStart(cStepRecord As cStep,
_
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep11_ProcessComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, lngInstanceId
As Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub


Private Sub
cExecStep11_ProcessStart(cStepRecord As cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub


Private Sub
cExecStep11_StepComplete(cStepRecord As cStep,
_
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord, dtmEndTime,
cExecStep11.Index, InstanceId, Status)

End Sub

Private Sub cExecStep11_StepStart(cStepRecord
As cStep, _
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub
Private Sub
cExecStep12_ProcessComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, lngInstanceId As
Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub


Private Sub
cExecStep12_ProcessStart(cStepRecord As cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub


Private Sub
cExecStep12_StepComplete(cStepRecord As cStep,
_
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord, dtmEndTime,
cExecStep12.Index, InstanceId, Status)

End Sub
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
Private Sub
cExecStep12_StepStart(cStepRecord As
cStep, _
      dtmStartTime As Currency, InstanceId
As Long)

   Call
TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub
Private Sub
cExecStep13_ProcessComplete(cStepRecor
d As cStep, _
      dtmEndTime As Currency,
lngInstanceId As Long, lElapsed As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub


Private Sub
cExecStep13_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime
As Currency, lngInstanceId As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub


Private Sub
cExecStep13_StepComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, InstanceId
As Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep13.Index,
InstanceId, Status)

End Sub

Private Sub
cExecStep13_StepStart(cStepRecord As
cStep, _
      dtmStartTime As Currency, InstanceId
As Long)

   Call
TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub
Private Sub
cExecStep14_ProcessComplete(cStepRecor
d As cStep, _
      dtmEndTime As Currency,
lngInstanceId As Long, lElapsed As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep14_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub


Private Sub
cExecStep14_StepComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep14.Index, InstanceId,
Status)

End Sub

Private Sub
cExecStep14_StepStart(cStepRecord As cStep,
_
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub
Private Sub
cExecStep15_ProcessComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, lngInstanceId
As Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub


Private Sub
cExecStep15_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub


Private Sub
cExecStep15_StepComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep15.Index, InstanceId,
Status)

End Sub

Private Sub cExecStep15_StepStart(cStepRecord
As cStep, _
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub
Private Sub
cExecStep16_ProcessComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, lngInstanceId As
Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub


Private Sub
cExecStep16_ProcessStart(cStepRecord As cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub


Private Sub
cExecStep16_StepComplete(cStepRecord As cStep,
_
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord, dtmEndTime,
cExecStep16.Index, InstanceId, Status)

End Sub

Private Sub cExecStep16_StepStart(cStepRecord
As cStep, _
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub
Private Sub
cExecStep17_ProcessComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, lngInstanceId As
Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub


Private Sub
cExecStep17_ProcessStart(cStepRecord As cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
End Sub


Private Sub
cExecStep17_StepComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, InstanceId
As Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep17.Index,
InstanceId, Status)

End Sub

Private Sub
cExecStep17_StepStart(cStepRecord As
cStep, _
      dtmStartTime As Currency, InstanceId
As Long)

   Call
TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep18_ProcessComplete(cStepRecor
d As cStep, _
      dtmEndTime As Currency,
lngInstanceId As Long, lElapsed As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub


Private Sub
cExecStep18_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime
As Currency, lngInstanceId As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub


Private Sub
cExecStep18_StepComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, InstanceId
As Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep18.Index,
InstanceId, Status)

End Sub

Private Sub
cExecStep18_StepStart(cStepRecord As
cStep, _
      dtmStartTime As Currency, InstanceId
As Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep19_ProcessComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, lngInstanceId
As Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub


Private Sub
cExecStep19_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub


Private Sub
cExecStep19_StepComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep19.Index, InstanceId,
Status)

End Sub

Private Sub
cExecStep19_StepStart(cStepRecord As cStep,
_
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep20_ProcessComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, lngInstanceId
As Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep20_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep20_StepComplete(cStepRecord As cStep,
_
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord, dtmEndTime,
cExecStep20.Index, InstanceId, Status)

End Sub

Private Sub cExecStep20_StepStart(cStepRecord
As cStep, _
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep21_ProcessComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, lngInstanceId As
Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep21_ProcessStart(cStepRecord As cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep21_StepComplete(cStepRecord As cStep,
_
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord, dtmEndTime,
cExecStep21.Index, InstanceId, Status)

End Sub

Private Sub cExecStep21_StepStart(cStepRecord
As cStep, _
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep22_ProcessComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, lngInstanceId As
Long, lElapsed As Long)
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep22_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime
As Currency, lngInstanceId As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep22_StepComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, InstanceId
As Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep22.Index,
InstanceId, Status)

End Sub

Private Sub
cExecStep22_StepStart(cStepRecord As
cStep, _
      dtmStartTime As Currency, InstanceId
As Long)

   Call
TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep23_ProcessComplete(cStepRecor
d As cStep, _
      dtmEndTime As Currency,
lngInstanceId As Long, lElapsed As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep23_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime
As Currency, lngInstanceId As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep23_StepComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, InstanceId
As Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep23.Index,
InstanceId, Status)

End Sub

Private Sub
cExecStep23_StepStart(cStepRecord As cStep,
_
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep24_ProcessComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, lngInstanceId
As Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep24_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep24_StepComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep24.Index, InstanceId,
Status)

End Sub

Private Sub
cExecStep24_StepStart(cStepRecord As cStep,
_
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep25_ProcessComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, lngInstanceId
As Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep25_ProcessStart(cStepRecord As cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep25_StepComplete(cStepRecord As cStep,
_
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord, dtmEndTime,
cExecStep25.Index, InstanceId, Status)

End Sub

Private Sub cExecStep25_StepStart(cStepRecord
As cStep, _
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep26_ProcessComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, lngInstanceId As
Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep26_ProcessStart(cStepRecord As cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep26_StepComplete(cStepRecord As cStep,
_
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord, dtmEndTime,
cExecStep26.Index, InstanceId, Status)

End Sub

Private Sub cExecStep26_StepStart(cStepRecord
As cStep, _
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
End Sub

Private Sub
cExecStep27_ProcessComplete(cStepRecor
d As cStep, _
      dtmEndTime As Currency,
lngInstanceId As Long, lElapsed As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep27_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime
As Currency, lngInstanceId As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep27_StepComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, InstanceId
As Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep27.Index,
InstanceId, Status)

End Sub

Private Sub
cExecStep27_StepStart(cStepRecord As
cStep, _
      dtmStartTime As Currency, InstanceId
As Long)

   Call
TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep28_ProcessComplete(cStepRecor
d As cStep, _
      dtmEndTime As Currency,
lngInstanceId As Long, lElapsed As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep28_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime
As Currency, lngInstanceId As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep28_StepComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep28.Index, InstanceId,
Status)

End Sub

Private Sub
cExecStep28_StepStart(cStepRecord As cStep,
_
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep29_ProcessComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, lngInstanceId
As Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep29_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep29_StepComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep29.Index, InstanceId,
Status)

End Sub

Private Sub
cExecStep29_StepStart(cStepRecord As cStep,
_
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep30_ProcessComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, lngInstanceId As
Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep30_ProcessStart(cStepRecord As cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep30_StepComplete(cStepRecord As cStep,
_
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord, dtmEndTime,
cExecStep30.Index, InstanceId, Status)

End Sub

Private Sub cExecStep30_StepStart(cStepRecord
As cStep, _
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep31_ProcessComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, lngInstanceId As
Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep31_ProcessStart(cStepRecord As cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep31_StepComplete(cStepRecord As cStep,
_
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord, dtmEndTime,
cExecStep31.Index, InstanceId, Status)
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
End Sub

Private Sub
cExecStep31_StepStart(cStepRecord As
cStep, _
      dtmStartTime As Currency, InstanceId
As Long)

   Call
TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep32_ProcessComplete(cStepRecor
d As cStep, _
      dtmEndTime As Currency,
lngInstanceId As Long, lElapsed As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep32_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime
As Currency, lngInstanceId As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep32_StepComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, InstanceId
As Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep32.Index,
InstanceId, Status)

End Sub

Private Sub
cExecStep32_StepStart(cStepRecord As
cStep, _
      dtmStartTime As Currency, InstanceId
As Long)

   Call
TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep33_ProcessComplete(cStepRecor
d As cStep, _
      dtmEndTime As Currency,
lngInstanceId As Long, lElapsed As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep33_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep33_StepComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep33.Index, InstanceId,
Status)

End Sub

Private Sub
cExecStep33_StepStart(cStepRecord As cStep,
_
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep34_ProcessComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, lngInstanceId
As Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep34_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep34_StepComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep34.Index, InstanceId,
Status)

End Sub

Private Sub cExecStep34_StepStart(cStepRecord
As cStep, _
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep35_ProcessComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, lngInstanceId As
Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep35_ProcessStart(cStepRecord As cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep35_StepComplete(cStepRecord As cStep,
_
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord, dtmEndTime,
cExecStep35.Index, InstanceId, Status)

End Sub

Private Sub cExecStep35_StepStart(cStepRecord
As cStep, _
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep36_ProcessComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, lngInstanceId As
Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep36_ProcessStart(cStepRecord As cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
End Sub

Private Sub
cExecStep36_StepComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, InstanceId
As Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep36.Index,
InstanceId, Status)

End Sub

Private Sub
cExecStep36_StepStart(cStepRecord As
cStep, _
      dtmStartTime As Currency, InstanceId
As Long)

   Call
TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep37_ProcessComplete(cStepRecor
d As cStep, _
      dtmEndTime As Currency,
lngInstanceId As Long, lElapsed As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep37_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime
As Currency, lngInstanceId As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep37_StepComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, InstanceId
As Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep37.Index,
InstanceId, Status)

End Sub

Private Sub
cExecStep37_StepStart(cStepRecord As
cStep, _
      dtmStartTime As Currency, InstanceId
As Long)

   Call
TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep38_ProcessComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, lngInstanceId
As Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep38_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep38_StepComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep38.Index, InstanceId,
Status)

End Sub

Private Sub
cExecStep38_StepStart(cStepRecord As cStep,
_
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep39_ProcessComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, lngInstanceId
As Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep39_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep39_StepComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord, dtmEndTime,
cExecStep39.Index, InstanceId, Status)

End Sub

Private Sub cExecStep39_StepStart(cStepRecord
As cStep, _
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep40_ProcessComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, lngInstanceId As
Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep40_ProcessStart(cStepRecord As cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep40_StepComplete(cStepRecord As cStep,
_
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord, dtmEndTime,
cExecStep40.Index, InstanceId, Status)

End Sub

Private Sub cExecStep40_StepStart(cStepRecord
As cStep, _
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep41_ProcessComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, lngInstanceId As
Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
Private Sub
cExecStep41_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime
As Currency, lngInstanceId As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep41_StepComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, InstanceId
As Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep41.Index,
InstanceId, Status)

End Sub

Private Sub
cExecStep41_StepStart(cStepRecord As
cStep, _
      dtmStartTime As Currency, InstanceId
As Long)

   Call
TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep42_ProcessComplete(cStepRecor
d As cStep, _
      dtmEndTime As Currency,
lngInstanceId As Long, lElapsed As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep42_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime
As Currency, lngInstanceId As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep42_StepComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, InstanceId
As Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep42.Index,
InstanceId, Status)

End Sub

Private Sub
cExecStep42_StepStart(cStepRecord As cStep,
_
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep43_ProcessComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, lngInstanceId
As Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep43_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep43_StepComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep43.Index, InstanceId,
Status)

End Sub

Private Sub
cExecStep43_StepStart(cStepRecord As cStep,
_
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep44_ProcessComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, lngInstanceId
As Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep44_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep44_StepComplete(cStepRecord As cStep,
_
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord, dtmEndTime,
cExecStep44.Index, InstanceId, Status)

End Sub

Private Sub cExecStep44_StepStart(cStepRecord
As cStep, _
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep45_ProcessComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, lngInstanceId As
Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep45_ProcessStart(cStepRecord As cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep45_StepComplete(cStepRecord As cStep,
_
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord, dtmEndTime,
cExecStep45.Index, InstanceId, Status)

End Sub

Private Sub cExecStep45_StepStart(cStepRecord
As cStep, _
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep46_ProcessComplete(cStepRecord As
cStep, _
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
      dtmEndTime As Currency,
lngInstanceId As Long, lElapsed As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep46_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime
As Currency, lngInstanceId As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep46_StepComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, InstanceId
As Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep46.Index,
InstanceId, Status)

End Sub

Private Sub
cExecStep46_StepStart(cStepRecord As
cStep, _
      dtmStartTime As Currency, InstanceId
As Long)

   Call
TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep47_ProcessComplete(cStepRecor
d As cStep, _
      dtmEndTime As Currency,
lngInstanceId As Long, lElapsed As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep47_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime
As Currency, lngInstanceId As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep47_StepComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep47.Index, InstanceId,
Status)

End Sub

Private Sub
cExecStep47_StepStart(cStepRecord As cStep,
_
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep48_ProcessComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, lngInstanceId
As Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep48_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep48_StepComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep48.Index, InstanceId,
Status)

End Sub

Private Sub
cExecStep48_StepStart(cStepRecord As cStep,
_
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep49_ProcessComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, lngInstanceId
As Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep49_ProcessStart(cStepRecord As cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep49_StepComplete(cStepRecord As cStep,
_
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord, dtmEndTime,
cExecStep49.Index, InstanceId, Status)

End Sub

Private Sub cExecStep49_StepStart(cStepRecord
As cStep, _
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep50_ProcessComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, lngInstanceId As
Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep50_ProcessStart(cStepRecord As cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep50_StepComplete(cStepRecord As cStep,
_
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord, dtmEndTime,
cExecStep50.Index, InstanceId, Status)

End Sub

Private Sub cExecStep50_StepStart(cStepRecord
As cStep, _
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
      dtmStartTime As Currency, InstanceId
As Long)

   Call
TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep51_ProcessComplete(cStepRecor
d As cStep, _
      dtmEndTime As Currency,
lngInstanceId As Long, lElapsed As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep51_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime
As Currency, lngInstanceId As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep51_StepComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, InstanceId
As Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep51.Index,
InstanceId, Status)

End Sub

Private Sub
cExecStep51_StepStart(cStepRecord As
cStep, _
      dtmStartTime As Currency, InstanceId
As Long)

   Call
TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep52_ProcessComplete(cStepRecor
d As cStep, _
      dtmEndTime As Currency,
lngInstanceId As Long, lElapsed As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep52_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep52_StepComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep52.Index, InstanceId,
Status)

End Sub

Private Sub
cExecStep52_StepStart(cStepRecord As cStep,
_
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep53_ProcessComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, lngInstanceId
As Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep53_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep53_StepComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep53.Index, InstanceId,
Status)

End Sub

Private Sub
cExecStep53_StepStart(cStepRecord As cStep,
_
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep54_ProcessComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, lngInstanceId As
Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep54_ProcessStart(cStepRecord As cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep54_StepComplete(cStepRecord As cStep,
_
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord, dtmEndTime,
cExecStep54.Index, InstanceId, Status)

End Sub

Private Sub cExecStep54_StepStart(cStepRecord
As cStep, _
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep55_ProcessComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, lngInstanceId As
Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep55_ProcessStart(cStepRecord As cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep55_StepComplete(cStepRecord As cStep,
_
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
    dtmEndTime As Currency, InstanceId
As Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep55.Index,
InstanceId, Status)

End Sub

Private Sub
cExecStep55_StepStart(cStepRecord As
cStep, _
    dtmStartTime As Currency, InstanceId
As Long)

   Call
TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep56_ProcessComplete(cStepRecor
d As cStep, _
    dtmEndTime As Currency,
lngInstanceId As Long, lElapsed As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep56_ProcessStart(cStepRecord As
cStep, _
    strCommand As String, dtmStartTime
As Currency, lngInstanceId As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep56_StepComplete(cStepRecord
As cStep, _
    dtmEndTime As Currency, InstanceId
As Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep56.Index,
InstanceId, Status)

End Sub

Private Sub
cExecStep56_StepStart(cStepRecord As
cStep, _
    dtmStartTime As Currency, InstanceId
As Long)

   Call
TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep57_ProcessComplete(cStepRecor
d As cStep, _
    dtmEndTime As Currency, lngInstanceId
As Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep57_ProcessStart(cStepRecord As
cStep, _
    strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep57_StepComplete(cStepRecord As
cStep, _
    dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep57.Index, InstanceId,
Status)

End Sub

Private Sub
cExecStep57_StepStart(cStepRecord As cStep,
_
    dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep58_ProcessComplete(cStepRecord
As cStep, _
    dtmEndTime As Currency, lngInstanceId
As Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep58_ProcessStart(cStepRecord As
cStep, _
    strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep58_StepComplete(cStepRecord As
cStep, _
    dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord, dtmEndTime,
cExecStep58.Index, InstanceId, Status)

End Sub

Private Sub cExecStep58_StepStart(cStepRecord
As cStep, _
    dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep59_ProcessComplete(cStepRecord As
cStep, _
    dtmEndTime As Currency, lngInstanceId As
Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep59_ProcessStart(cStepRecord As cStep, _
    strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep59_StepComplete(cStepRecord As cStep,
_
    dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord, dtmEndTime,
cExecStep59.Index, InstanceId, Status)

End Sub

Private Sub cExecStep59_StepStart(cStepRecord
As cStep, _
    dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep60_ProcessComplete(cStepRecord As
cStep, _
    dtmEndTime As Currency, lngInstanceId As
Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep60_ProcessStart(cStepRecord As cStep, _
    strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep60_StepComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, InstanceId
As Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep60.Index,
InstanceId, Status)

End Sub

Private Sub
cExecStep60_StepStart(cStepRecord As
cStep, _
      dtmStartTime As Currency, InstanceId
As Long)

   Call
TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep61_ProcessComplete(cStepRecor
d As cStep, _
      dtmEndTime As Currency,
lngInstanceId As Long, lElapsed As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep61_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime
As Currency, lngInstanceId As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep61_StepComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, InstanceId
As Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep61.Index,
InstanceId, Status)

End Sub

Private Sub
cExecStep61_StepStart(cStepRecord As
cStep, _
      dtmStartTime As Currency, InstanceId
As Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep62_ProcessComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, lngInstanceId
As Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep62_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep62_StepComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep62.Index, InstanceId,
Status)

End Sub

Private Sub
cExecStep62_StepStart(cStepRecord As cStep,
_
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep63_ProcessComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, lngInstanceId
As Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep63_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep63_StepComplete(cStepRecord As cStep,
_
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord, dtmEndTime,
cExecStep63.Index, InstanceId, Status)

End Sub

Private Sub cExecStep63_StepStart(cStepRecord
As cStep, _
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep64_ProcessComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, lngInstanceId As
Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep64_ProcessStart(cStepRecord As cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep64_StepComplete(cStepRecord As cStep,
_
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord, dtmEndTime,
cExecStep64.Index, InstanceId, Status)

End Sub

Private Sub cExecStep64_StepStart(cStepRecord
As cStep, _
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep65_ProcessComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, lngInstanceId As
Long, lElapsed As Long)
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep65_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime
As Currency, lngInstanceId As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep65_StepComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, InstanceId
As Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep65.Index,
InstanceId, Status)

End Sub

Private Sub
cExecStep65_StepStart(cStepRecord As
cStep, _
      dtmStartTime As Currency, InstanceId
As Long)

   Call
TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep66_ProcessComplete(cStepRecor
d As cStep, _
      dtmEndTime As Currency,
lngInstanceId As Long, lElapsed As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep66_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime
As Currency, lngInstanceId As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep66_StepComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, InstanceId
As Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep66.Index, InstanceId,
Status)

End Sub

Private Sub
cExecStep66_StepStart(cStepRecord As cStep,
_
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep67_ProcessComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, lngInstanceId
As Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep67_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep67_StepComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep67.Index, InstanceId,
Status)

End Sub

Private Sub
cExecStep67_StepStart(cStepRecord As cStep,
_
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep68_ProcessComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, lngInstanceId
As Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep68_ProcessStart(cStepRecord As cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep68_StepComplete(cStepRecord As cStep,
_
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord, dtmEndTime,
cExecStep68.Index, InstanceId, Status)

End Sub

Private Sub cExecStep68_StepStart(cStepRecord
As cStep, _
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep69_ProcessComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, lngInstanceId As
Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep69_ProcessStart(cStepRecord As cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep69_StepComplete(cStepRecord As cStep,
_
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord, dtmEndTime,
cExecStep69.Index, InstanceId, Status)

End Sub

Private Sub cExecStep69_StepStart(cStepRecord
As cStep, _
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
End Sub

Private Sub
cExecStep70_ProcessComplete(cStepRecor
d As cStep, _
      dtmEndTime As Currency,
lngInstanceId As Long, lElapsed As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep70_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime
As Currency, lngInstanceId As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep70_StepComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, InstanceId
As Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep70.Index,
InstanceId, Status)

End Sub

Private Sub
cExecStep70_StepStart(cStepRecord As
cStep, _
      dtmStartTime As Currency, InstanceId
As Long)

   Call
TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep71_ProcessComplete(cStepRecor
d As cStep, _
      dtmEndTime As Currency,
lngInstanceId As Long, lElapsed As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep71_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime
As Currency, lngInstanceId As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep71_StepComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep71.Index, InstanceId,
Status)

End Sub

Private Sub
cExecStep71_StepStart(cStepRecord As cStep,
_
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep72_ProcessComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, lngInstanceId
As Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep72_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep72_StepComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep72.Index, InstanceId,
Status)

End Sub

Private Sub
cExecStep72_StepStart(cStepRecord As cStep,
_
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep73_ProcessComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, lngInstanceId As
Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep73_ProcessStart(cStepRecord As cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep73_StepComplete(cStepRecord As cStep,
_
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord, dtmEndTime,
cExecStep73.Index, InstanceId, Status)

End Sub

Private Sub cExecStep73_StepStart(cStepRecord
As cStep, _
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep74_ProcessComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, lngInstanceId As
Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep74_ProcessStart(cStepRecord As cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep74_StepComplete(cStepRecord As cStep,
_
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord, dtmEndTime,
cExecStep74.Index, InstanceId, Status)
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
End Sub

Private Sub
cExecStep74_StepStart(cStepRecord As
cStep, _
      dtmStartTime As Currency, InstanceId
As Long)

   Call
TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep75_ProcessComplete(cStepRecor
d As cStep, _
      dtmEndTime As Currency,
lngInstanceId As Long, lElapsed As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep75_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime
As Currency, lngInstanceId As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep75_StepComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, InstanceId
As Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep75.Index,
InstanceId, Status)

End Sub

Private Sub
cExecStep75_StepStart(cStepRecord As
cStep, _
      dtmStartTime As Currency, InstanceId
As Long)

   Call
TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep76_ProcessComplete(cStepRecor
d As cStep, _
      dtmEndTime As Currency,
lngInstanceId As Long, lElapsed As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep76_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep76_StepComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep76.Index, InstanceId,
Status)

End Sub

Private Sub
cExecStep76_StepStart(cStepRecord As cStep,
_
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep77_ProcessComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, lngInstanceId
As Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep77_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep77_StepComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep77.Index, InstanceId,
Status)

End Sub

Private Sub cExecStep77_StepStart(cStepRecord
As cStep, _
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep78_ProcessComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, lngInstanceId As
Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep78_ProcessStart(cStepRecord As cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep78_StepComplete(cStepRecord As cStep,
_
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord, dtmEndTime,
cExecStep78.Index, InstanceId, Status)

End Sub

Private Sub cExecStep78_StepStart(cStepRecord
As cStep, _
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep79_ProcessComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, lngInstanceId As
Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep79_ProcessStart(cStepRecord As cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
End Sub

Private Sub
cExecStep79_StepComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, InstanceId
As Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep79.Index,
InstanceId, Status)

End Sub

Private Sub
cExecStep79_StepStart(cStepRecord As
cStep, _
      dtmStartTime As Currency, InstanceId
As Long)

   Call
TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep80_ProcessComplete(cStepRecor
d As cStep, _
      dtmEndTime As Currency,
lngInstanceId As Long, lElapsed As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep80_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime
As Currency, lngInstanceId As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep80_StepComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, InstanceId
As Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep80.Index,
InstanceId, Status)

End Sub

Private Sub
cExecStep80_StepStart(cStepRecord As
cStep, _
      dtmStartTime As Currency, InstanceId
As Long)

   Call
TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep81_ProcessComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, lngInstanceId
As Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep81_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep81_StepComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep81.Index, InstanceId,
Status)

End Sub

Private Sub
cExecStep81_StepStart(cStepRecord As cStep,
_
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep82_ProcessComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, lngInstanceId
As Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep82_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep82_StepComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord, dtmEndTime,
cExecStep82.Index, InstanceId, Status)

End Sub

Private Sub cExecStep82_StepStart(cStepRecord
As cStep, _
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep83_ProcessComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, lngInstanceId As
Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep83_ProcessStart(cStepRecord As cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep83_StepComplete(cStepRecord As cStep,
_
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord, dtmEndTime,
cExecStep83.Index, InstanceId, Status)

End Sub

Private Sub cExecStep83_StepStart(cStepRecord
As cStep, _
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep84_ProcessComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, lngInstanceId As
Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
Private Sub
cExecStep84_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime
As Currency, lngInstanceId As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep84_StepComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, InstanceId
As Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep84.Index,
InstanceId, Status)

End Sub

Private Sub
cExecStep84_StepStart(cStepRecord As
cStep, _
      dtmStartTime As Currency, InstanceId
As Long)

   Call
TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep85_ProcessComplete(cStepRecor
d As cStep, _
      dtmEndTime As Currency,
lngInstanceId As Long, lElapsed As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep85_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime
As Currency, lngInstanceId As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep85_StepComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, InstanceId
As Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep85.Index,
InstanceId, Status)

End Sub

Private Sub
cExecStep85_StepStart(cStepRecord As cStep,
_
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep86_ProcessComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, lngInstanceId
As Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep86_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep86_StepComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep86.Index, InstanceId,
Status)

End Sub

Private Sub
cExecStep86_StepStart(cStepRecord As cStep,
_
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep87_ProcessComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency, lngInstanceId
As Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep87_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep87_StepComplete(cStepRecord As cStep,
_
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord, dtmEndTime,
cExecStep87.Index, InstanceId, Status)

End Sub

Private Sub cExecStep87_StepStart(cStepRecord
As cStep, _
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep88_ProcessComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, lngInstanceId As
Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep88_ProcessStart(cStepRecord As cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep88_StepComplete(cStepRecord As cStep,
_
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord, dtmEndTime,
cExecStep88.Index, InstanceId, Status)

End Sub

Private Sub cExecStep88_StepStart(cStepRecord
As cStep, _
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep89_ProcessComplete(cStepRecord As
cStep, _
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
        dtmEndTime As Currency,
lngInstanceId As Long, lElapsed As Long)

    Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep89_ProcessStart(cStepRecord As
cStep, _
        strCommand As String, dtmStartTime
As Currency, lngInstanceId As Long)

    Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep89_StepComplete(cStepRecord
As cStep, _
        dtmEndTime As Currency, InstanceId
As Long, Status As InstanceStatus)

    Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep89.Index,
InstanceId, Status)

End Sub

Private Sub
cExecStep89_StepStart(cStepRecord As
cStep, _
        dtmStartTime As Currency, InstanceId
As Long)

    Call
TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep90_ProcessComplete(cStepRecor
d As cStep, _
        dtmEndTime As Currency,
lngInstanceId As Long, lElapsed As Long)

    Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep90_ProcessStart(cStepRecord As
cStep, _
        strCommand As String, dtmStartTime
As Currency, lngInstanceId As Long)

    Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep90_StepComplete(cStepRecord As
cStep, _
        dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

    Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep90.Index, InstanceId,
Status)

End Sub

Private Sub
cExecStep90_StepStart(cStepRecord As cStep,
_
        dtmStartTime As Currency, InstanceId As
Long)

    Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep91_ProcessComplete(cStepRecord
As cStep, _
        dtmEndTime As Currency, lngInstanceId
As Long, lElapsed As Long)

    Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep91_ProcessStart(cStepRecord As
cStep, _
        strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

    Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep91_StepComplete(cStepRecord As
cStep, _
        dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

    Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep91.Index, InstanceId,
Status)

End Sub

Private Sub
cExecStep91_StepStart(cStepRecord As cStep,
_
        dtmStartTime As Currency, InstanceId As
Long)

    Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep92_ProcessComplete(cStepRecord
As cStep, _
        dtmEndTime As Currency, lngInstanceId
As Long, lElapsed As Long)

    Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep92_ProcessStart(cStepRecord As cStep, _
        strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

    Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep92_StepComplete(cStepRecord As cStep,
_
        dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

    Call StepTerminated(cStepRecord, dtmEndTime,
cExecStep92.Index, InstanceId, Status)

End Sub

Private Sub cExecStep92_StepStart(cStepRecord
As cStep, _
        dtmStartTime As Currency, InstanceId As
Long)

    Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep93_ProcessComplete(cStepRecord As
cStep, _
        dtmEndTime As Currency, lngInstanceId As
Long, lElapsed As Long)

    Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep93_ProcessStart(cStepRecord As cStep, _
        strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

    Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep93_StepComplete(cStepRecord As cStep,
_
        dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

    Call StepTerminated(cStepRecord, dtmEndTime,
cExecStep93.Index, InstanceId, Status)

End Sub

Private Sub cExecStep93_StepStart(cStepRecord
As cStep, _
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
    dtmStartTime As Currency, InstanceId
As Long)

   Call
TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep94_ProcessComplete(cStepRecor
d As cStep, _
    dtmEndTime As Currency,
lngInstanceId As Long, lElapsed As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep94_ProcessStart(cStepRecord As
cStep, _
    strCommand As String, dtmStartTime
As Currency, lngInstanceId As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep94_StepComplete(cStepRecord
As cStep, _
    dtmEndTime As Currency, InstanceId
As Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep94.Index,
InstanceId, Status)

End Sub

Private Sub
cExecStep94_StepStart(cStepRecord As
cStep, _
    dtmStartTime As Currency, InstanceId
As Long)

   Call
TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep95_ProcessComplete(cStepRecor
d As cStep, _
    dtmEndTime As Currency,
lngInstanceId As Long, lElapsed As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep95_ProcessStart(cStepRecord As
cStep, _
    strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep95_StepComplete(cStepRecord As
cStep, _
    dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep95.Index, InstanceId,
Status)

End Sub

Private Sub
cExecStep95_StepStart(cStepRecord As cStep,
_
    dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep96_ProcessComplete(cStepRecord
As cStep, _
    dtmEndTime As Currency, lngInstanceId
As Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep96_ProcessStart(cStepRecord As
cStep, _
    strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep96_StepComplete(cStepRecord As
cStep, _
    dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep96.Index, InstanceId,
Status)

End Sub

Private Sub
cExecStep96_StepStart(cStepRecord As cStep,
_
    dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep97_ProcessComplete(cStepRecord As
cStep, _
    dtmEndTime As Currency, lngInstanceId As
Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep97_ProcessStart(cStepRecord As cStep, _
    strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep97_StepComplete(cStepRecord As cStep,
_
    dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord, dtmEndTime,
cExecStep97.Index, InstanceId, Status)

End Sub

Private Sub cExecStep97_StepStart(cStepRecord
As cStep, _
    dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep98_ProcessComplete(cStepRecord As
cStep, _
    dtmEndTime As Currency, lngInstanceId As
Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep98_ProcessStart(cStepRecord As cStep, _
    strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep98_StepComplete(cStepRecord As cStep,
_
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
        dtmEndTime As Currency, InstanceId
As Long, Status As InstanceStatus)

    Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep98.Index,
InstanceId, Status)

End Sub

Private Sub
cExecStep98_StepStart(cStepRecord As
cStep, _
        dtmStartTime As Currency, InstanceId
As Long)

    Call
TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep99_ProcessComplete(cStepRecor
d As cStep, _
        dtmEndTime As Currency,
lngInstanceId As Long, lElapsed As Long)

    Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep99_ProcessStart(cStepRecord As
cStep, _
        strCommand As String, dtmStartTime
As Currency, lngInstanceId As Long)

    Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep99_StepComplete(cStepRecord
As cStep, _
        dtmEndTime As Currency, InstanceId
As Long, Status As InstanceStatus)

    Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep99.Index,
InstanceId, Status)

End Sub

Private Sub
cExecStep99_StepStart(cStepRecord As
cStep, _
        dtmStartTime As Currency, InstanceId
As Long)

    Call
TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep2_ProcessComplete(cStepRecord
As cStep, _
        dtmEndTime As Currency, lngInstanceId
As Long, lElapsed As Long)

    Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep2_ProcessStart(cStepRecord As
cStep, _
        strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

    Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub


Private Sub
cExecStep2_StepComplete(cStepRecord As
cStep, _
        dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

    Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep2.Index, _
            InstanceId, Status)

End Sub

Private Sub
cExecStep2_StepStart(cStepRecord As cStep,
_
        dtmStartTime As Currency, InstanceId As
Long)

    Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub


Private Sub
cExecStep3_ProcessComplete(cStepRecord As
cStep, _
        dtmEndTime As Currency, lngInstanceId
As Long, lElapsed As Long)

    Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep3_ProcessStart(cStepRecord As
cStep, _
        strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

    Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub


Private Sub
cExecStep3_StepComplete(cStepRecord As
cStep, _
        dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

    Call StepTerminated(cStepRecord, dtmEndTime,
cExecStep3.Index, _
            InstanceId, Status)

End Sub

Private Sub cExecStep3_StepStart(cStepRecord As
cStep, _
        dtmStartTime As Currency, InstanceId As
Long)

    Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub


Private Sub
cExecStep4_ProcessComplete(cStepRecord As
cStep, _
        dtmEndTime As Currency, lngInstanceId As
Long, lElapsed As Long)

    Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub cExecStep4_ProcessStart(cStepRecord
As cStep, _
        strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

    Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep4_StepComplete(cStepRecord As cStep,
_
        dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

    Call StepTerminated(cStepRecord, dtmEndTime,
cExecStep4.Index, _
            InstanceId, Status)

End Sub


Private Sub cExecStep4_StepStart(cStepRecord As
cStep, _
        dtmStartTime As Currency, InstanceId As
Long)

    Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep5_ProcessComplete(cStepRecord As
cStep, _
        dtmEndTime As Currency, lngInstanceId As
Long, lElapsed As Long)

    Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
End Sub

Private Sub
cExecStep5_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime
As Currency, lngInstanceId As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep5_StepComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, InstanceId
As Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep5.Index, _
         InstanceId, Status)

End Sub

Private Sub
cExecStep5_StepStart(cStepRecord As
cStep, _
      dtmStartTime As Currency, InstanceId
As Long)

   Call
TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep6_ProcessComplete(cStepRecord
As cStep, _
      dtmEndTime As Currency,
lngInstanceId As Long, lElapsed As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep6_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime
As Currency, lngInstanceId As Long)

   Call
TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep6_StepComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, InstanceId
As Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep6.Index, _
         InstanceId, Status)

End Sub

Private Sub
cExecStep6_StepStart(cStepRecord As cStep,
_
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep7_ProcessComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, lngInstanceId
As Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub
cExecStep7_ProcessStart(cStepRecord As
cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep7_StepComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord,
dtmEndTime, cExecStep7.Index, _
         InstanceId, Status)

End Sub

Private Sub
cExecStep7_StepStart(cStepRecord As cStep,
_
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub
cExecStep8_ProcessComplete(cStepRecord As
cStep, _
      dtmEndTime As Currency, lngInstanceId
As Long, lElapsed As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, EndTime:=dtmEndTime,
ElapsedTime:=lElapsed)

End Sub

Private Sub cExecStep8_ProcessStart(cStepRecord
As cStep, _
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long)

   Call TimeUpdateForProcess(cStepRecord,
lngInstanceId, StartTime:=dtmStartTime,
Command:=strCommand)

End Sub

Private Sub
cExecStep8_StepComplete(cStepRecord As cStep,
_
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)

   Call StepTerminated(cStepRecord, dtmEndTime,
cExecStep8.Index, _
         InstanceId, Status)

End Sub

Private Sub cExecStep8_StepStart(cStepRecord As
cStep, _
      dtmStartTime As Currency, InstanceId As
Long)

   Call TimeStartUpdateForStep(cStepRecord,
InstanceId, dtmStartTime)

End Sub

Private Sub Class_Initialize()

   Dim lngCount As Long
   Dim lngTemp As Long

   On Error GoTo InitializeErr

   Set mcFreeSteps = New cVectorLng
   ' Initialize the array of free objects with all
elements
   ' for now
   For lngCount = 0 To
glngNumConcurrentProcesses - 1 Step 1
      mcFreeSteps.Add lngCount
   Next lngCount

   ' Initialize a byte array with the number of free
processes. It will
   ' be used later to determine if any step is running
   ' Each element in the array can represent 8 steps,
1 for each bit
   ReDim mbarrFree(glngNumConcurrentProcesses
\ gintBitsPerByte)

   ' Initialize each element in the byte array w/ all
1's
   ' (upto glngNumConcurrentProcesses)
   For lngCount = LBound(mbarrFree) To
UBound(mbarrFree) Step 1
      lngTemp = IIf( _
            glngNumConcurrentProcesses -
(gintBitsPerByte * lngCount) > gintBitsPerByte, _
            gintBitsPerByte, _
            glngNumConcurrentProcesses -
(gintBitsPerByte * lngCount))

      mbarrFree(lngCount) = (2 ^ lngTemp) - 1
   Next lngCount

   Set mcInstances = New cInstances
   Set mcFailures = New cFailedSteps
   Set mcNavSteps = New cStepTree
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   Set mcTermSteps = New cTermSteps

   ' Initialize the Abort flag to False
   mblnAbort = False
   mblnAsk = False

   Exit Sub

InitializeErr:
   ' Log the error code raised by Visual
Basic
   Call LogErrors(Errors)
   On Error GoTo 0
   Err.Raise vbObjectError +
errInitializeFailed, mstrModuleName &
"Initialize", _
          LoadResString(errInitializeFailed)

End Sub
Private Sub Class_Terminate()

   On Error GoTo Class_TerminateErr

   mcFreeSteps.Clear
   Set mcFreeSteps = Nothing
   ReDim mbarrFree(0)

   mcInstances.Clear
   Set mcInstances = Nothing

   Set mcFailures = Nothing
   Set mcNavSteps = Nothing
   Set mcTermSteps = Nothing

   Exit Sub

Class_TerminateErr:
   Call LogErrors(Errors)

End Sub

Private Sub
mcTermSteps_TermStepExists(cStepDetails
As cTermStep)

   Call
RunNextStep(cStepDetails.TimeComplete,
cStepDetails.Index, _
          cStepDetails.InstanceId,
cStepDetails.ExecutionStatus)

End Sub

VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cRunItDetails"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:       cRunItDetails.cls
'             Microsoft TPC-H Kit Ver. 1.00
'             Copyright Microsoft, 1999
'             All Rights Reserved
'
'
'  PURPOSE:    This module encapsulates
the properties of iterator values
'             that are used by the step being
executed at runtime.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'

Option Explicit

' Used to indicate the source module name
when errors
' are raised by this class
Private Const mstrModuleName As String =
"cRunItDetails."
Private mstrSource As String

Private mstrIteratorName As String
Private mintType As ValueType
Private mlngSequence As Long
Private mlngFrom As Long
Private mlngTo As Long
Private mlngStep As Long
Private mstrValue As String

Public Property Get RangeTo() As Long

   RangeTo = mlngTo

End Property
Public Property Let RangeTo(ByVal vdata As
Long)

   mlngTo = vdata

End Property

Public Property Get RangeFrom() As Long

   RangeFrom = mlngFrom

End Property
Public Property Get Sequence() As Long

   Sequence = mlngSequence

End Property

Public Property Get RangeStep() As Long

   RangeStep = mlngStep

End Property
Public Property Let RangeStep(vdata As Long)

   mlngStep = vdata

End Property

Public Property Let RangeFrom(ByVal vdata
As Long)

   mlngFrom = vdata

End Property
Public Property Let Sequence(ByVal vdata As
Long)

   mlngSequence = vdata

End Property

Public Property Get IteratorType() As
ValueType

   IteratorType = mintType

End Property
Public Property Let IteratorType(ByVal vdata
As ValueType)

   On Error GoTo TypeErr
   mstrSource = mstrModuleName & "Type"

   ' These constants have been defined in the
enumeration,
   ' Type, which is exposed
   Select Case vdata
      Case gintFrom, gintTo, gintStep, gintValue
         mintType = vdata

      Case Else
         On Error GoTo 0
         Err.Raise vbObjectError + errTypeInvalid, _
            mstrSource,
LoadResString(errTypeInvalid)
   End Select

   Exit Property

TypeErr:
   LogErrors Errors
   mstrSource = mstrModuleName & "Type"
   On Error GoTo 0
   Err.Raise vbObjectError + errTypeInvalid, _
      mstrSource, LoadResString(errTypeInvalid)

End Property
Private Sub IsList()

   If mintType <> gintValue Then
      On Error GoTo 0
      Err.Raise vbObjectError + errInvalidProperty,
mstrSource, _
            LoadResString(errInvalidProperty)
   End If

End Sub
Private Sub IsRange()

   If mintType = gintValue Then
      On Error GoTo 0
      Err.Raise vbObjectError + errInvalidProperty,
mstrSource, _
            LoadResString(errInvalidProperty)
   End If

End Sub

Public Property Get Value() As String

    Value = mstrValue

End Property
Public Property Let Value(vdata As String)

   mstrValue = vdata

End Property


Public Property Get IteratorName() As String

   IteratorName = mstrIteratorName

End Property
Public Property Let IteratorName(ByVal vdata As
String)

   mstrIteratorName = vdata

End Property

VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
Attribute VB_Name = "cRunItNode"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
' An iterator class containing the properties
that are used
' by the stpe being executed.
' These iterators might actually come from
steps that are at
' a higher level than the step actually being
executed (viz.
' direct ascendants of the step at any level).

Option Explicit

Public IteratorName As String
Public Value As String
Public StepId As Long

VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cRunOnly"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
Option Explicit

Public Event Done()
Private WithEvents mcRunWsp As
cRunWorkspace
Attribute mcRunWsp.VB_VarHelpID = -1

Public WspName As String
Public WorkspaceId As Long
Public WspLog As String

Public Sub RunWsp()

   On Error GoTo RunWspErr

   Set mcRunWsp = New cRunWorkspace
   Set mcRunWsp.LoadDb = dbsAttTool
   mcRunWsp.WorkspaceId = WorkspaceId
   mcRunWsp.CreateInputFiles = True
   mcRunWsp.RunWorkspace

   Exit Sub

RunWspErr:
   ' Log the VB error code
   LogErrors Errors

End Sub

Private Sub
mcRunWsp_RunComplete(dtmEndTime As
Currency)

   MsgBox "Completed executing
workspace: " & gstrSQ & WspName &
gstrSQ & " at " & _
         JulianDateToString(dtmEndTime) &
"." & vbCrLf & vbCrLf & _
         "The log file for the run is: " &
gstrSQ & WspLog & gstrSQ & "."
   RaiseEvent Done

End Sub

Private Sub
mcRunWsp_RunStart(dtmStartTime As
Currency, strWspLog As String, lRunId As
Long)
   WspLog = strWspLog
End Sub


VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cRunStep"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:       cRunStep.cls
'             Microsoft TPC-H Kit Ver. 1.00
'             Copyright Microsoft, 1999
'             All Rights Reserved
'
'
'  PURPOSE:     This class executes the step
that is assigned to the
'             ExecuteStep property. It executes the
pre-execution constraints
'             in sequence and then the step itself.
At the end it executes
'             the post-execution constraints. Since
these steps should always
'             be executed in sequence, each step is
only fired on the
'             completion of the previous step.
'  Contact:     Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

' Used to indicate the source module name
when errors
' are raised by this class
Private Const mstrModuleName As String =
"cRunStep."
Private mstrSource As String

' Local variable(s) to hold property value(s)
Private mcStep As cStep
Private mcGlobals As cArrSteps
Private mcvntWspPreCons As Variant
Private mcvntWspPostCons As Variant
Private mcvntPreCons As Variant
Private mcvntPostCons As Variant
Private mcIterators As cRunColIt
Private mlngInstanceId As Long ' Identifier for
the current instance
Private mlngIndex As Long ' Index value for
the current instance
Private mstrCommand As String ' The
command string
Private msRunStepDtl As String ' Step text/file
name that will go into the run_step_details
table
Private mblnAbort As Boolean   ' Set to True
when the user aborts the run
Private msOutputFile As String
Private msErrorFile As String
Private miStatus As InstanceStatus
Private mcVBErr As cVBErrorsSM
Public WspParameters As cArrParameters
Public WspConnections As cConnections
Public WspConnDtls As cConnDtls

Private WithEvents mcTermProcess As
cTermProcess
Attribute mcTermProcess.VB_VarHelpID = -1
Public RunId As Long
Public CreateInputFiles As Boolean
Private msOutputDir As String

' Object that will execute the step
Private WithEvents mcExecObj As
EXECUTEDLLLib.Execute
Attribute mcExecObj.VB_VarHelpID = -1

' Holds the step that is currently being executed
(constraint or
' worker step)
Private mcExecStep As cStep

Private Const msCompareExe As String =
"\diff.exe"

Private Enum NextNodeType
   mintWspPreConstraint = 1
   mintPreConstraint
   mintStep
   mintWspPostConstraint
   mintPostConstraint
End Enum

' Public events to notify the calling function of the
' start and end time for each step
Public Event StepStart(cStepRecord As cStep, _
      dtmStartTime As Currency, InstanceId As
Long)
Public Event StepComplete(cStepRecord As cStep,
_
      dtmEndTime As Currency, InstanceId As
Long, Status As InstanceStatus)
Public Event ProcessStart(cStepRecord As cStep, _
      strCommand As String, dtmStartTime As
Currency, _
      InstanceId As Long)
Public Event ProcessComplete(cStepRecord As
cStep, _
      dtmStartTime As Currency, InstanceId As
Long, lElapsed As Long)

Private Function AppendDiffErrors(sDiffFile As
String)
   ' The file containing the errors generated by the
diff utility is passed in
   ' These errors are appended to the error file for
the step

   Dim sTemp As String
   Dim InputFile As Integer

   If Not StringEmpty(sDiffFile) Then

      InputFile = FreeFile
      Open sDiffFile For Input Access Read As
InputFile

      Do While Not EOF(InputFile)        ' Loop
until end of file.
         Line Input #InputFile, sTemp    ' Read line
into variable.
         mcVBErr.LogMessage sTemp
      Loop

      Close InputFile
   End If

End Function

Private Sub CreateStepTextFile()
   ' Creates a file containing the step text being
executed
   On Error GoTo CreateStepTextFileErr
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   Dim sInputFile As String

   If mcExecStep.ExecutionMechanism =
gintExecuteShell Then
      sInputFile =
GetOutputFile(gsCmdFileSuffix)
   Else
      sInputFile =
GetOutputFile(gsSqlFileSuffix)
   End If

   ' Generate a file containing the step text
being executed
   If Not
StringEmpty(mcExecStep.StepTextFile) Or
mcExecStep.ExecutionMechanism =
gintExecuteShell Then
      FileCopy mstrCommand, sInputFile
   Else
      Call
WriteCommandToFile(mstrCommand,
sInputFile)
   End If

   Exit Sub

CreateStepTextFileErr:

   mcVBErr.LogVBErrors

End Sub
Private Function GetOutputFile(strFileExt
As String) As String
   ' This function generates the output file
name for the step currently being executed
   ' The value of the built-in parameter
'DefaultDir' is appended with the run
identifier
   ' for the file location
   ' The step label is used for the file name
and a combination of all iterator values
   ' for the step is used to make the output
files unique for each instance
   Dim sFile As String
   Dim sIt As String
   Dim lIt As Long

   On Error GoTo GetOutputFileErr

   sFile =
SubstituteParametersIfPossible(mcExecStep
.StepLabel)

   sFile = TranslateStepLabel(sFile)

   If mcExecStep Is mcStep Then
      ' Use iterators that have been defined
for the worker or any of it's managers
      ' to make the error/log file unique for
this instance
      For lIt = mcIterators.Count - 1 To 0
Step -1
         sIt = sIt & gsExtSeparator &
mcIterators(lIt).Value
      Next lIt
   End If
   sIt = sIt & strFileExt

   ' Ensure that the length of the complete
path does not exceed 255 characters
   If Len(msOutputDir) + Len(sFile) +
Len(sIt) > MAX_PATH Then
      sFile = Mid(sFile, 1, MAX_PATH -
Len(sIt) - Len(msOutputDir))
   End If
   GetOutputFile = msOutputDir & sFile & sIt
   Exit Function

GetOutputFileErr:

   ' Does not make sense to log error to the
error file yet. Write to the project
   ' log and return the step label as default
   GetOutputFile = mcExecStep.StepLabel &
gsExtSeparator & strFileExt

End Function


Private Sub HandleExecutionError()

   On Error GoTo HandleExecutionError

   ' Log the error code raised by Visual Basic
   miStatus = gintFailed
   mcVBErr.LogVBErrors
   Call
mcVBErr.WriteError(errExecuteStepFailed, _
         OptArgs:="Continuation criteria for the
step is: " &
gsContCriteria(mcStep.ContinuationCriteria))

HandleExecutionError:

   ' Logging failed - return

End Sub

Public Property Get Index() As Long

   Index = mlngIndex

End Property
Public Property Let Index(ByVal vdata As
Long)

   mlngIndex = vdata

End Property
Private Function InitializeExecStatus() As
InstanceStatus
   Dim sCompareFile As String

   On Error GoTo InitializeExecStatusErr

   InitializeExecStatus =
mcExecObj.StepStatus

   If InitializeExecStatus = gintComplete Then
      If Not
StringEmpty(mcExecStep.FailureDetails) Then
         ' Compare output to determine whether
the step failed
         sCompareFile =
GetShortName(SubstituteParameters( _
               mcExecStep.FailureDetails,
mcExecStep.WorkspaceId, mcIterators, _
               WspParameters))
         InitializeExecStatus =
IIf(CompareOutput(sCompareFile,
msOutputFile), gintComplete, gintFailed)
      End If
   End If

   Exit Function

InitializeExecStatusErr:
   mcVBErr.LogVBErrors
'   Call LogErrors(Errors)
   InitializeExecStatus = mcExecObj.StepStatus

End Function
Private Function CompareOutput(sCompareFile As
String, sOutputFile As String) As Boolean

   Dim sCmpOutput As String
   Dim sDiffOutput As String

   On Error GoTo CompareOutputErr

   ' Create temporary files to store the file compare
output and
   ' the errors generated by the compare function
   sCmpOutput = CreateTempFile()
   sDiffOutput = CreateTempFile()

   ' Run the compare utility and redirect it's output
and errors
   SyncShell ("cmd /c " & _
         GetShortName(App.Path &
msCompareExe) & gstrBlank & _
         sCompareFile & gstrBlank & sOutputFile &
_
         " > " & sCmpOutput & " 2> " &
sDiffOutput)

   If FileLen(sDiffOutput) > 0 Then
      ' The compare generated errors - append error
msgs to the error file
      Call AppendDiffErrors(sDiffOutput)
      CompareOutput = False
   Else
      CompareOutput = (FileLen(sCmpOutput) = 0)
   End If

   If Not CompareOutput Then
      mcVBErr.WriteError errDiffFailed
   End If

   ' Delete the temporary files used to store the
output of the compare and
   ' the errors generated by the compare
   Kill sDiffOutput
   Kill sCmpOutput

   Exit Function

CompareOutputErr:
   mcVBErr.LogVBErrors
   CompareOutput = False

End Function
Public Property Get InstanceId() As Long

   InstanceId = mlngInstanceId

End Property
Public Property Let InstanceId(ByVal vdata As
Long)

   mlngInstanceId = vdata

End Property

Private Function ExecuteConstraint(vntConstraints
As Variant, _
      ByRef intLoopIndex As Integer) As Boolean

   ' Returns True if there is a constraint in the
passed in
   ' array that remains to be executed

   If IsArray(vntConstraints) And Not
IsEmpty(vntConstraints) Then
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
      ExecuteConstraint =
(LBound(vntConstraints) <= intLoopIndex)
And (intLoopIndex <=
UBound(vntConstraints))
   Else
      ExecuteConstraint = False
   End If

End Function
Private Function NextStep() As cStep

   ' Determines which is the next step to be
executed - it could
   ' be either a pre-execution step, the
worker step itself
   ' or a post-execution step

   Dim cConsRec As cConstraint
   Dim cNextStepRec As cStep
   Dim vntStepConstraints As Variant

   ' Static variable to remember exactly
where we are in the
   ' processing
   Static intIndex As Integer
   Static intNextStepType As
NextNodeType

   On Error GoTo NextStepErr

   If mblnAbort = True Then
      ' The user has aborted the run - do not
run any more
      ' processes for the step
      Set NextStep = Nothing
      Exit Function
   End If

   If intNextStepType = 0 Then
      ' First time through this function - set
the Index and
      ' node type to initial values
      intNextStepType =
mintWspPreConstraint
      intIndex = 0
      RaiseEvent StepStart(mcStep,
Determine64BitTime(), mlngInstanceId)
   End If

   Do
      Select Case intNextStepType
         Case mintWspPreConstraint
            vntStepConstraints =
mcvntWspPreCons

         Case mintPreConstraint
            vntStepConstraints =
mcvntPreCons

         Case mintStep
            ' CONS:
            If mcStep.StepType =
gintWorkerStep Then
               Set cNextStepRec = mcStep
            End If

         Case mintWspPostConstraint
            vntStepConstraints =
mcvntWspPostCons

         Case mintPostConstraint
            vntStepConstraints =
mcvntPostCons

      End Select

      If intNextStepType <> mintStep Then
         ' Check if there is a constraint to be
executed
         If
ExecuteConstraint(vntStepConstraints,
intIndex) Then
            ' Get the corresponding step record to
be executed
            ' Query the global step record for the
current
            ' constraint
            Set cConsRec =
vntStepConstraints(intIndex)

            Set cNextStepRec =
mcGlobals.QueryStep(cConsRec.GlobalStepId
)
            intIndex = intIndex + 1
         Else
            If intNextStepType =
mintPostConstraint Then
               ' No more stuff to be executed for
the step
               ' Raise a Done event
               Set cNextStepRec = Nothing

               ' Set the next step type to an
invalid value
               intNextStepType = -1
            Else
               Call NextType(intNextStepType,
intIndex)
            End If
         End If
      Else
         ' Increment the step type so we look at
the post-
         ' execution steps the next time through
         Call NextType(intNextStepType,
intIndex)
      End If

   Loop Until (Not cNextStepRec Is Nothing)
Or _
         intNextStepType = -1

   Set NextStep = cNextStepRec

   Exit Function

NextStepErr:
   ' Log the error code raised by Visual Basic
   Call LogErrors(Errors)
   On Error GoTo 0
   mstrSource = mstrModuleName &
"NextStep"
   Err.Raise vbObjectError +
errNextStepFailed, mstrSource, _
      LoadResString(errNextStepFailed)

End Function
Public Sub Execute()
   ' This procedure is the method that executes
the step that
   ' is assigned to the ExecuteStep property. It
call a procedure
   ' to determine the next step to be executed.
   ' Then it initializes all the properties of the
cExecuteSM object
   ' and calls it's run method to execute it.
   Dim cConn As cConnection
   Dim cRunConnDtl As cConnDtl

   On Error GoTo ExecuteErr

   ' If this procedure is called after a step has
completed,
   ' we would have to check if we created any
temporary files
   ' while executing that step
   If Not mcExecStep Is Nothing Then
      If Not StringEmpty(mcExecStep.StepTextFile)
Or mcExecStep.ExecutionMechanism =
gintExecuteShell Then
         ' Remove the temporary file that we created
while
         ' running this command
         Kill mstrCommand
      End If

      Call StepCompleted

      ' The VB errors class stores a reference to the
Execute class since it uses
      ' a method of the class to write errors to the
error log. Hence,
      ' release all references to the Execute object
before destroying it.
      Set mcVBErr.ErrorFile = Nothing
      Set mcExecObj = Nothing

      ' Delete empty output and error files (generated
by shell commands)
      ' (Can be done only after cleaning up
cExecObj)
      Call DeleteEmptyOutputFiles
   Else
      ' First time through - initialize the location of
output files
      msOutputDir =
GetDefaultDir(mcStep.WorkspaceId,
WspParameters)
      msOutputDir = msOutputDir &
gstrFileSeparator & Trim(Str(RunId)) &
gstrFileSeparator
      ' Dummy file since the function expects a file
name
      MakePathValid (msOutputDir & "a.txt")
   End If

   ' Call a procedure to determine the next step to be
executed
   ' - could be a constraint or the step itself
   ' Initialize a module-level variable to the step
being
   ' executed
   Set mcExecStep = NextStep
   If mcExecStep Is Nothing Then
      RaiseEvent StepComplete(mcStep,
Determine64BitTime(), mlngInstanceId, miStatus)
      ' No more stuff to execute
      Exit Sub
   End If

   Dim sStartDir As String

   Set mcExecObj = New
EXECUTEDLLLib.Execute

   ' The VB errors class uses the WriteError method
of the Execute class to write
   ' all VB errors to the error file for the step (this
prevents a clash when the
   ' VB errors and Execution errors have to be
written to the same log). Hence, store
   ' a reference to the Execute object in mcVBErr
   msErrorFile = GetOutputFile(gsErrorFileSuffix)
   mcExecObj.ErrorFile = msErrorFile
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   Call DeleteFile(msErrorFile,
bCheckIfEmpty:=False)
   Set mcVBErr.ErrorFile = mcExecObj

   If mcExecStep.ExecutionMechanism =
gintExecuteShell Then
      sStartDir =
Trim$(GetShortName(SubstituteParameters(
_
                mcExecStep.StartDir,
mcExecStep.WorkspaceId, mcIterators,
WspParameters:=WspParameters)))
      ' Dummy connection object
      Set cConn = New cConnection
      Set cRunConnDtl = New cConnDtl
   Else
      ' Find the connection string value and
substitute parameter values in it
      Set cRunConnDtl =
WspConnDtls.GetConnectionDtl(mcExecSt
ep.WorkspaceId, mcExecStep.StartDir)
      Set cConn =
WspConnections.GetConnection(mcExecSte
p.WorkspaceId,
cRunConnDtl.ConnectionString)
      sStartDir =
Trim$(SubstituteParameters(cConn.Connect
ionValue, _
            mcExecStep.WorkspaceId,
mcIterators,
WspParameters:=WspParameters))
   End If

   msOutputFile =
GetOutputFile(gsOutputFileSuffix)
   Call DeleteFile(msOutputFile,
bCheckIfEmpty:=False)
   mcExecObj.OutputFile = msOutputFile
'   mcExecObj.LogFile =
GetShortName(SubstituteParameters( _
'          mcExecStep.LogFile,
mcExecStep.WorkspaceId, mcIterators,
WspParameters:=WspParameters))
   If mcExecStep.ExecutionMechanism =
gintExecuteODBC And _
         cRunConnDtl.ConnType =
ConnTypeDynamic Then
      Call
mcExecObj.DoExecute(BuildCommandStri
ng(), sStartDir,
mcExecStep.ExecutionMechanism, _
            cConn.NoCountDisplay,
cConn.NoExecute, cConn.ParseQueryOnly,
cConn.QuotedIdentifiers, _
            cConn.AnsiNulls,
cConn.ShowQueryPlan,
cConn.ShowStatsTime,
cConn.ShowStatsIO, _
            cConn.RowCount,
cConn.QueryTimeOut, gstrEmptyString)
   Else
      Call
mcExecObj.DoExecute(BuildCommandStri
ng(), sStartDir,
mcExecStep.ExecutionMechanism, _
            cConn.NoCountDisplay,
cConn.NoExecute, cConn.ParseQueryOnly,
cConn.QuotedIdentifiers, _
            cConn.AnsiNulls,
cConn.ShowQueryPlan,
cConn.ShowStatsTime,
cConn.ShowStatsIO, _
            cConn.RowCount,
cConn.QueryTimeOut,
mcExecStep.StartDir)
   End If

   Exit Sub

ExecuteErr:
   Call HandleExecutionError

   ' We can assume that if we are in this
function, a StepStart event has been triggered
already.
   RaiseEvent StepComplete(mcStep,
Determine64BitTime(), mlngInstanceId,
miStatus)

End Sub
Private Function BuildCommandString() As
String
   ' Process text to be executed - either from the
text
   ' field or read it from a file.
   ' This function will always return the
command text for ODBC commands
   ' and a file name for Shell commands
   Dim sFile As String
   Dim sCommand As String
   Dim sTemp As String

   On Error GoTo BuildCommandStringErr

   If Not
StringEmpty(mcExecStep.StepTextFile) Then
      ' Substitute parameter values and
environment variables
      ' in the filename
      msRunStepDtl =
SubstituteParameters(mcExecStep.StepTextFil
e, _
            mcExecStep.WorkspaceId,
mcIterators, WspParameters:=WspParameters)

      sFile = GetShortName(msRunStepDtl)

      mstrCommand =
SubstituteParametersInText(sFile,
mcExecStep.WorkspaceId)

      If mcExecStep.ExecutionMechanism =
gintExecuteODBC Then
         ' Read the contents of the file and pass
it to ODBC
         BuildCommandString =
ReadCommandFromFile(mstrCommand)
      Else
         BuildCommandString = mstrCommand
      End If
   Else
      ' Substitute parameter values and
environment variables
      ' in the step text
      msRunStepDtl =
SubstituteParameters(mcExecStep.StepText, _
            mcExecStep.WorkspaceId,
mcIterators, WspParameters:=WspParameters)
      mstrCommand = msRunStepDtl

      If mcExecStep.ExecutionMechanism =
gintExecuteShell Then
         ' Write the command to a temp file
(enables us to execute multiple
         ' commands via the command
interpreter)
         mstrCommand =
WriteCommandToFile(msRunStepDtl)
         BuildCommandString = mstrCommand
      Else
         BuildCommandString =
SQLFixup(msRunStepDtl)
      End If
   End If

   If CreateInputFiles Then
      Call CreateStepTextFile
   End If

   Exit Function

BuildCommandStringErr:
   ' Log the error code raised by the Execute
procedure
'   Call LogErrors(Errors)
   mcVBErr.LogVBErrors

   On Error GoTo 0
   mstrSource = mstrModuleName & "Execute"
   Err.Raise vbObjectError + errExecuteStepFailed,
mstrSource, _
      LoadResString(errExecuteStepFailed) &
mstrCommand

End Function
Public Sub Abort()

   On Error GoTo AbortErr

   ' Setting the Abort flag to True will ensure that
we
   ' don't execute any more processes for this step
   mblnAbort = True

   If Not mcExecObj Is Nothing Then
      mcExecObj.Abort
   Else
      ' We are not in the middle of execution yet
   End If

   Exit Sub

AbortErr:
   Call LogErrors(Errors)
   On Error GoTo 0
   Err.Raise vbObjectError + errProgramError, _
      mstrModuleName & "Abort", _
      LoadResString(errProgramError)

End Sub
Private Sub NextType(ByRef StepType As
NextNodeType, _
      ByRef Position As Integer)

   StepType = StepType + 1
   Position = 0

End Sub
Private Sub StepCompleted()

   On Error GoTo StepCompletedErr

   If Not mcExecStep Is Nothing Then
      If mcExecStep Is mcStep Then
         miStatus = InitializeExecStatus
         If miStatus = gintFailed Then
            ' Create input files if the step failed
execution and one hasn't been created already
            If Not CreateInputFiles Then
CreateStepTextFile
            Call
mcVBErr.WriteError(errExecuteStepFailed, _
                  OptArgs:="Continuation criteria for
the step is: " &
gsContCriteria(mcStep.ContinuationCriteria))
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
        End If
      End If
    End If

    Exit Sub

StepCompletedErr:
    ' Log the error code raised by Visual
Basic
    miStatus = gintFailed
    mcVBErr.LogVBErrors
    Call
mcVBErr.WriteError(errExecuteStepFailed,
_
          OptArgs:="Continuation criteria for
the step is: " &
gsContCriteria(mcStep.ContinuationCriteria
))

End Sub
Private Sub DeleteEmptyOutputFiles()

    On Error GoTo
DeleteEmptyOutputFilesErr

    ' Delete empty output and error files
    If Not mcExecStep Is Nothing Then
      Call DeleteFile(msErrorFile,
bCheckIfEmpty:=True)
      Call DeleteFile(msOutputFile,
bCheckIfEmpty:=True)
    End If

    Exit Sub

DeleteEmptyOutputFilesErr:
    ' Not a critical error - continue

End Sub
Private Function
ReadCommandFromFile(strFileName As
String) As String

    ' Returns the contents of the passed in file

    Dim sCommand As String
    Dim sTemp As String
    Dim InputFile As Integer

    On Error GoTo
ReadCommandFromFileErr

    If Not StringEmpty(strFileName) Then

      InputFile = FreeFile
      Open strFileName For Input Access
Read As InputFile

      Line Input #InputFile, sCommand '
Read line into variable.

      Do While Not EOF(InputFile) ' Loop
until end of file.
          Line Input #InputFile, sTemp ' Read
line into variable.
          sCommand = sCommand & vbCrLf
& sTemp
      Loop

      Close InputFile
    End If

    ReadCommandFromFile = sCommand

    Exit Function

ReadCommandFromFileErr:

    ' Log the error code raised by Visual Basic
    Call LogErrors(Errors)
    mstrSource = mstrModuleName &
"ReadCommandFromFile"
    On Error GoTo 0
    Err.Raise vbObjectError +
errSubValuesFailed, _
          gstrSource, _
          LoadResString(errSubValuesFailed)

End Function
Private Function
SubstituteParametersIfPossible(strLabel As
String)

    On Error GoTo
SubstituteParametersIfPossibleErr

    SubstituteParametersIfPossible =
SubstituteParameters(strLabel, _
          mcExecStep.WorkspaceId, mcIterators,
WspParameters:=WspParameters)
    Exit Function

SubstituteParametersIfPossibleErr:
    SubstituteParametersIfPossible = strLabel

End Function
Private Function
SubstituteParametersInText(strFileName As
String, _
      lngWorkspace As Long) As String

    ' Reads each line in the passed in file,
substitutes parameter
    ' values in the line and writes out the
modified line to a
    ' temporary file that we create. The
temporary file will be
    ' removed once the step completes
execution.
    ' Returns the name of the newly created
temporary file.

    Dim strTempFile As String
    Dim strTemp As String
    Dim strOutput As String
    Dim InputFile As Integer
    Dim OutputFile As Integer

    On Error GoTo
SubstituteParametersInTextErr

    strTempFile = CreateTempFile()

    If Not StringEmpty(strFileName) Then

      InputFile = FreeFile
      Open strFileName For Input Access Read
As InputFile

      OutputFile = FreeFile
      Open strTempFile For Output Access
Write As OutputFile

      Do While Not EOF(InputFile) ' Loop until
end of file.
          Line Input #InputFile, strTemp ' Read
line into variable.
          strOutput =
SubstituteParameters(strTemp, lngWorkspace,
mcIterators, WspParameters:=WspParameters)

          If mcExecStep.ExecutionMechanism =
gintExecuteODBC Then strOutput =
SQLFixup(strOutput)

          Print #OutputFile, strOutput
          BugMessage strOutput
      Loop

    End If

    Close InputFile
    Close OutputFile

    SubstituteParametersInText = strTempFile

    Exit Function

SubstituteParametersInTextErr:

    ' Log the error code raised by Visual Basic
'   Call LogErrors(Errors)
    mcVBErr.LogVBErrors
    mstrSource = mstrModuleName &
"SubstituteParametersInText"
    On Error GoTo 0
    Err.Raise vbObjectError + errSubValuesFailed, _
          gstrSource, _
          LoadResString(errSubValuesFailed)

End Function

Private Function WriteCommandToFile(sCommand
As String, Optional sFile As String =
gstrEmptyString) As String

    ' Writes the command text to a temporary file
    ' Returns the name of the temporary file

    Dim OutputFile As Integer

    On Error GoTo WriteCommandToFileErr

    If StringEmpty(sFile) Then
      sFile = CreateTempFile()
    End If

    OutputFile = FreeFile
    Open sFile For Output Access Write As
OutputFile

    Print #OutputFile, sCommand

    Close OutputFile

    WriteCommandToFile = sFile

    Exit Function

WriteCommandToFileErr:

    ' Log the error code raised by Visual Basic
    Call LogErrors(Errors)
    mstrSource = mstrModuleName &
"WriteCommandToFile"
    On Error GoTo 0
    Err.Raise vbObjectError + errSubValuesFailed, _
          gstrSource, _
          LoadResString(errSubValuesFailed)

End Function


Public Property Get WspPreCons() As Variant
    WspPreCons = mcvntWspPreCons
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
End Property
Public Property Let WspPreCons(ByVal
vdata As Variant)
   mcvntWspPreCons = vdata
End Property

Public Property Get WspPostCons() As
Variant
   WspPostCons = mcvntWspPostCons
End Property
Public Property Let WspPostCons(ByVal
vdata As Variant)
   mcvntWspPostCons = vdata
End Property

Public Property Get PreCons() As Variant
   PreCons = mcvntPreCons
End Property
Public Property Let PreCons(ByVal vdata
As Variant)
   mcvntPreCons = vdata
End Property

Public Property Get PostCons() As Variant
   PostCons = mcvntPostCons
End Property
Public Property Let PostCons(ByVal vdata
As Variant)
   mcvntPostCons = vdata
End Property

Public Property Set Globals(cRunSteps As
cArrSteps)

   Set mcGlobals = cRunSteps

End Property
Public Property Set ExecuteStep(cRunStep
As cStep)

   Set mcStep = cRunStep

End Property
Public Property Get Globals() As cArrSteps

   Set Globals = mcGlobals

End Property
Public Property Get ExecuteStep() As cStep

   Set ExecuteStep = mcStep

End Property
Public Property Set Iterators(vdata As
cRunColIt)

   Set mcIterators = vdata

End Property
Private Sub Class_Initialize()

   ' Initialize the Abort flag to False
   mblnAbort = False
   Set mcVBErr = New cVBErrorsSM
   Set mcTermProcess = New cTermProcess

End Sub

Private Sub Class_Terminate()

   On Error GoTo Class_TerminateErr

   Set mcExecObj = Nothing
   Set mcVBErr = Nothing
   Set mcTermProcess = Nothing

   Exit Sub

Class_TerminateErr:
   Call LogErrors(Errors)

End Sub

Private Sub mcExecObj_Start(ByVal
StartTime As Currency)
   ' Raise an event indicating that the step has
begun execution
   RaiseEvent ProcessStart(mcExecStep,
msRunStepDtl, StartTime, mlngInstanceId)
End Sub

Private Sub mcExecObj_Complete(ByVal
EndTime As Currency, ByVal Elapsed As
Long)

   On Error GoTo mcExecObj_CompleteErr

   Debug.Print Elapsed
   RaiseEvent ProcessComplete(mcExecStep,
EndTime, mlngInstanceId, Elapsed)
   mcTermProcess.ProcessTerminated

   Exit Sub

mcExecObj_CompleteErr:
   Call LogErrors(Errors)

End Sub

Private Sub
mcTermProcess_TermProcessExists()

   On Error GoTo TermProcessExistsErr

   ' Call a procedure to execute the next step, if
any
   Call Execute

   Exit Sub

TermProcessExistsErr:
   ' Log the error code raised by the Execute
procedure
   Call LogErrors(Errors)

End Sub
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cRunWorkspace"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:       cRunWorkspace.cls
'              Microsoft TPC-H Kit Ver. 1.00
'              Copyright Microsoft, 1999
'              All Rights Reserved
'
'
'  PURPOSE:    This class loads all the
information necessary to
'              execute a workspace and calls
cRunInst to execute the workspace.
'              It also propagates Step start and
complete and
'              Run start and complete events.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

' Used to indicate the source module name when
errors
' are raised by this module
Private Const mstrModuleName As String =
"cRunWorkspace."
Private mstrSource As String

Private mcRunSteps As cArrSteps
Private mcRunParams As cArrParameters
Private mcRunConstraints As cArrConstraints
Private mcRunConnections As cConnections
Private mcRunConnDtls As cConnDtls
Private mcvntWspPreCons As Variant
Private mcvntWspPostCons As Variant
Private mdbsLoadDb As Database
Private mlngRunId As Long
Private mlngWorkspaceId As Long
Private mField As cStringSM
Public CreateInputFiles As Boolean

Private WithEvents mcRun As cRunInst
Attribute mcRun.VB_VarHelpID = -1

Public Event RunStart(dtmStartTime As Currency,
strWspLog As String, lRunId As Long)
Public Event RunComplete(dtmEndTime As
Currency)
Public Event StepStart(cStepRecord As cStep,
dtmStartTime As Currency, lngInstanceId As Long,
_
      sPath As String, sIts As String)
Public Event StepComplete(cStepRecord As cStep,
dtmEndTime As Currency, lngInstanceId As Long)
Public Event ProcessStart(cStepRecord As cStep,
strCommand As String, _
      dtmStartTime As Currency, lngInstanceId As
Long)
Public Event ProcessComplete(cStepRecord As
cStep, dtmEndTime As Currency, lngInstanceId As
Long)
Public Function InstancesForStep(lngStepId As
Long, iStatus As InstanceStatus) As cInstances
   ' Returns an array of all the instances for a step

   If mcRun Is Nothing Then
      Set InstancesForStep = Nothing
   Else
      Set InstancesForStep =
mcRun.InstancesForStep(lngStepId, iStatus)
   End If

End Function
Private Sub InsertRunDetail(cStepRecord As cStep,
_
      strCommand As String, dtmStartTime As
Currency, _
      lngInstanceId As Long, lParentInstanceId As
Long, sItValue As String)
   ' Inserts a new run detail record into the database

   Dim strInsert As String
   Dim qy As QueryDef

   On Error GoTo InsertRunDetailErr
   mstrSource = mstrModuleName &
"InsertRunDetail"

   strInsert = "insert into run_step_details " & _
         "( run_id, step_id, version_no, instance_id,
parent_instance_id, " & _
         " command, start_time, iterator_value ) " &
_
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
        " values ( "

#If USE_JET Then

   strInsert = strInsert & " [r_id], [s_id],
[ver_no], [i_id], [p_i_id], " & _
        " [com], [s_date], [it_val] )"

   Set qy = mdbsLoadDb.CreateQueryDef( _
        gstrEmptyString, strInsert)

   ' Call a procedure to assign the Querydef
parameters
   Call AssignParameters(qy,
StartTime:=dtmStartTime, _
        StepId:=cStepRecord.StepId, _
        Version:=cStepRecord.VersionNo, _
        InstanceId:=lngInstanceId, _
        Command:=strCommand)

   qy.Execute dbFailOnError
   qy.Close

#Else

   strInsert = strInsert & Str(mlngRunId) _
      & ", " & Str(cStepRecord.StepId) _
      & ", " &
mField.MakeStringFieldValid(cStepRecord.
VersionNo) _
      & ", " & Str(lngInstanceId) _
      & ", " & Str(lParentInstanceId) _
      & ", " &
mField.MakeStringFieldValid(strCommand)
_
      & ", " & Str(dtmStartTime) _
      & ", " &
mField.MakeStringFieldValid(sItValue)

   strInsert = strInsert & " ) "

   mdbsLoadDb.Execute strInsert,
dbFailOnError

#End If

   Exit Sub

InsertRunDetailErr:
   LogErrors Errors
   mstrSource = mstrModuleName &
"InsertRunDetail"
   On Error GoTo 0
   Err.Raise vbObjectError +
errUpdateRunDataFailed, _
        mstrSource, _

LoadResString(errUpdateRunDataFailed)

End Sub
Private Sub UpdateRunDetail(cStepRecord
As cStep, _
      dtmEndTime As Currency,
lngInstanceId As Long, lElapsed As Long)
   ' Updates the run detail record in the
database

   Dim strUpdate As String
   Dim qy As QueryDef

   On Error GoTo UpdateRunDetailErr

   strUpdate = "update run_step_details " &
_
        " set end_time = [e_date], elapsed_time
= [elapsed] " & _
        " where run_id = [r_id] " & _
        " and step_id = [s_id] " & _
        " and version_no = [ver_no] " & _
        " and instance_id = [i_id] "

   Set qy = mdbsLoadDb.CreateQueryDef( _
        gstrEmptyString, strUpdate)

   ' Call a procedure to assign the Querydef
parameters
   Call AssignParameters(qy,
EndTime:=dtmEndTime, _
        StepId:=cStepRecord.StepId, _
        Version:=cStepRecord.VersionNo, _
        InstanceId:=lngInstanceId,
Elapsed:=lElapsed)

   qy.Execute dbFailOnError
   qy.Close

   Exit Sub

UpdateRunDetailErr:
   LogErrors Errors
   mstrSource = mstrModuleName &
"UpdateRunDetail"
   On Error GoTo 0
   Err.Raise vbObjectError +
errUpdateRunDataFailed, _
        mstrSource, _

LoadResString(errUpdateRunDataFailed)

End Sub
Private Function
InsertRunHeader(dtmStartTime As Currency)
As Long
   ' Inserts a new run header record into the
database
   ' and returns the id for the run

   Dim strInsert As String
   Dim qy As QueryDef

   On Error GoTo InsertRunHeaderErr

   strInsert = "insert into run_header " & _
      "( run_id, workspace_id, start_time ) " &
_
      " values ( " & _
      " [r_id], [w_id], [s_date] )"

   Set qy = mdbsLoadDb.CreateQueryDef( _
        gstrEmptyString, strInsert)

   ' Call a procedure to execute the Querydef
object
   Call AssignParameters(qy,
StartTime:=dtmStartTime)

   qy.Execute dbFailOnError
   qy.Close

   InsertRunHeader = mlngRunId
   Exit Function

InsertRunHeaderErr:
   LogErrors Errors
   mstrSource = mstrModuleName &
"InsertRunHeader"
   On Error GoTo 0
   Err.Raise vbObjectError +
errUpdateRunDataFailed, _
        mstrSource, _
        LoadResString(errUpdateRunDataFailed)

End Function
Private Sub InsertRunParameters(dtmStartTime As
Currency)
   ' Inserts a new run header record into the database
   ' and returns the id for the run

   Dim strInsert As String
   Dim qy As QueryDef
   Dim cParamRec As cParameter
   Dim lngIndex As Long

   On Error GoTo InsertRunParametersErr

   strInsert = "insert into run_parameters " & _
      "( run_id, parameter_name, parameter_value )
" & _
      " values ( " & _
      " [r_id], [p_name], [p_value] )"

   Set qy = mdbsLoadDb.CreateQueryDef( _
        gstrEmptyString, strInsert)
   qy.Parameters("r_id").Value = mlngRunId

   For lngIndex = 0 To
mcRunParams.ParameterCount - 1
      Set cParamRec = mcRunParams(lngIndex)

      qy.Parameters("p_name").Value =
cParamRec.ParameterName
      qy.Parameters("p_value").Value =
cParamRec.ParameterValue
      qy.Execute dbFailOnError

   Next lngIndex

   qy.Close

   Exit Sub

InsertRunParametersErr:
   LogErrors Errors
   mstrSource = mstrModuleName &
"InsertRunParameters"
   On Error GoTo 0
   Err.Raise vbObjectError +
errUpdateRunDataFailed, _
        mstrSource, _
        LoadResString(errUpdateRunDataFailed)

End Sub
Private Sub AssignParameters(qyExec As
DAO.QueryDef, _
      Optional StartTime As Currency = 0, _
      Optional EndTime As Currency = 0, _
      Optional StepId As Long = 0, _
      Optional Version As String = gstrEmptyString,
_
      Optional InstanceId As Long = 0, _
      Optional ParentInstanceId As Long = 0, _
      Optional Command As String =
gstrEmptyString, _
      Optional Elapsed As Long = 0, _
      Optional ItValue As String = gstrEmptyString)
   ' Assigns values to the parameters in the querydef
object

   Dim prmParam As DAO.Parameter

   On Error GoTo AssignParametersErr
   mstrSource = mstrModuleName &
"AssignParameters"
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   For Each prmParam In
qyExec.Parameters
      Select Case prmParam.Name
         Case "[w_id]"
            prmParam.Value =
mlngWorkspaceId

         Case "[r_id]"
            prmParam.Value = mlngRunId

         Case "[s_id]"
            BugAssert StepId <> 0
            prmParam.Value = StepId

         Case "[ver_no]"
            BugAssert Not
StringEmpty(Version)
            prmParam.Value = Version

         Case "[i_id]"
            BugAssert InstanceId <> 0
            prmParam.Value = InstanceId

         Case "[p_i_id]"
            prmParam.Value =
ParentInstanceId

         Case "[com]"
            BugAssert Not
StringEmpty(Command)
            prmParam.Value = Command

         Case "[s_date]"
            BugAssert StartTime <> 0
            prmParam.Value = StartTime

         Case "[e_date]"
            BugAssert EndTime <> 0
            prmParam.Value = EndTime

         Case "[elapsed]"
            prmParam.Value = Elapsed

         Case "[it_val]"
            prmParam.Value = ItValue

         Case Else
            ' Write the parameter name that is
faulty
            WriteError errInvalidParameter,
mstrSource, _
                  prmParam.Name
            On Error GoTo 0
            Err.Raise errInvalidParameter,
mstrSource, _

LoadResString(errInvalidParameter)
      End Select
   Next prmParam

   Exit Sub

AssignParametersErr:

   mstrSource = mstrModuleName &
"AssignParameters"
   Call LogErrors(Errors)
   On Error GoTo 0
   Err.Raise vbObjectError +
errAssignParametersFailed, _
         mstrSource,
LoadResString(errAssignParametersFailed)

End Sub
Private Sub RunStartProcessing(dtmStartTime
As Currency)

   On Error GoTo RunStartProcessingErr

   ' Insert the run header into the database
   Call InsertRunHeader(dtmStartTime)

   ' Insert the run parameters into the database
   Call InsertRunParameters(dtmStartTime)

   Exit Sub

RunStartProcessingErr:
   ' Log the error code raised by Visual Basic
   Call LogErrors(Errors)
   mstrSource = mstrModuleName &
"RunStartProcessing"
   ShowError errUpdateRunDataFailed
   WriteError errUpdateRunDataFailed,
mstrSource

End Sub
Private Sub
ProcessStartProcessing(cStepRecord As cStep,
_
      strCommand As String, dtmStartTime As
Currency, lngInstanceId As Long, _
      lParentInstanceId As Long, sItValue As
String)

   On Error GoTo ProcessStartProcessingErr

   ' Insert the run detail into the database
   Call InsertRunDetail(cStepRecord,
strCommand, dtmStartTime, lngInstanceId, _
         lParentInstanceId, sItValue)

   Exit Sub

ProcessStartProcessingErr:
   ' Log the error code raised by Visual Basic
   Call LogErrors(Errors)
   mstrSource = mstrModuleName &
"ProcessStartProcessing"
   ShowError errUpdateRunDataFailed
   WriteError errUpdateRunDataFailed,
mstrSource

End Sub
Private Sub StepStartProcessing(cStepRecord
As cStep, dtmStartTime As Currency, _
      lngInstanceId As Long, lParentInstanceId
As Long, sItValue As String)

   On Error GoTo StepStartProcessingErr

   ' Since ProcessStart events won't be
triggered for manager steps
   If cStepRecord.StepType = gintManagerStep
Then
      ' Insert the run detail into the database
      Call InsertRunDetail(cStepRecord,
cStepRecord.StepLabel, _
            dtmStartTime, lngInstanceId,
lParentInstanceId, sItValue)
   End If

   Exit Sub

StepStartProcessingErr:
   ' Log the error code raised by Visual Basic
   Call LogErrors(Errors)
   mstrSource = mstrModuleName &
"StepStartProcessing"
   ShowError errUpdateRunDataFailed

End Sub
Private Sub
ProcessCompleteProcessing(cStepRecord As cStep,
_
      dtmStartTime As Currency, lngInstanceId As
Long, lElapsed As Long)

   On Error GoTo ProcessCompleteProcessingErr

   ' Insert the run detail into the database
   Call UpdateRunDetail(cStepRecord,
dtmStartTime, lngInstanceId, lElapsed)

   Exit Sub

ProcessCompleteProcessingErr:
   ' Log the error code raised by Visual Basic
   Call LogErrors(Errors)
   mstrSource = mstrModuleName &
"ProcessCompleteProcessing"
   ShowError errUpdateRunDataFailed

End Sub
Private Sub StepCompleteProcessing(cStepRecord
As cStep, _
      dtmEndTime As Currency, lngInstanceId As
Long, lElapsed As Long)

   On Error GoTo StepCompleteProcessingErr

   ' Since ProcessComplete events won't be
triggered for manager steps
   If cStepRecord.StepType = gintManagerStep
Then
      ' Update the run detail in the database
      Call UpdateRunDetail(cStepRecord,
dtmEndTime, lngInstanceId, lElapsed)
   End If

   Exit Sub

StepCompleteProcessingErr:
   ' Log the error code raised by Visual Basic
   Call LogErrors(Errors)
   ShowError errUpdateRunDataFailed

End Sub
Private Sub RunCompleteProcessing(dtmEndTime
As Currency)

   On Error GoTo RunCompleteProcessingErr

   ' Update the header record with the end time for
the run
   Call UpdateRunHeader(dtmEndTime)

   Exit Sub

RunCompleteProcessingErr:
   ' Log the error code raised by Visual Basic
   Call LogErrors(Errors)
   ShowError errUpdateRunDataFailed

End Sub

Private Sub UpdateRunHeader(ByVal dtmEndTime
As Currency)
   ' Updates the run header record with the end date

   Dim strUpdate As String
   Dim qy As QueryDef

   On Error GoTo UpdateRunHeaderErr
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   strUpdate = "update run_header " & _
         " set end_time = [e_date] " & _
         " where run_id = [r_id] "

   Set qy = mdbsLoadDb.CreateQueryDef( _
         gstrEmptyString, strUpdate)

   ' Call a procedure to execute the Querydef
object
   Call AssignParameters(qy,
EndTime:=dtmEndTime)

   qy.Execute dbFailOnError
   qy.Close

   Exit Sub

UpdateRunHeaderErr:
   LogErrors Errors
   mstrSource = mstrModuleName &
"UpdateRunHeader"
   On Error GoTo 0
   Err.Raise vbObjectError +
errUpdateRunDataFailed, _
         mstrSource, _

LoadResString(errUpdateRunDataFailed)

End Sub

Public Property Let WorkspaceId(ByVal
vdata As Long)
   mlngWorkspaceId = vdata
End Property
Public Property Get WorkspaceId() As Long
   WorkspaceId = mlngWorkspaceId
End Property
Public Sub RunWorkspace()

   Dim cRunSeq As cSequence

   On Error GoTo RunWorkspaceErr

   ' Call a procedure to load the module-
level structures
   ' with all the step and parameter data for
the run
   If LoadRunData = False Then
      ' Error handled by the function already
      Exit Sub
   End If

   ' Retrieve the next run identifier using the
sequence class
   Set cRunSeq = New cSequence
   Set cRunSeq.IdDatabase = dbsAttTool
   cRunSeq.IdentifierColumn = "run_id"
   mlngRunId = cRunSeq.Identifier
   Set cRunSeq = Nothing

   Call
mcRunParams.InitBuiltInsForRun(mlngWor
kspaceId, mlngRunId)

   Set mcRun.Constraints =
mcRunConstraints
   mcRun.WspPreExecution =
mcvntWspPreCons
   mcRun.WspPostExecution =
mcvntWspPostCons

   Set mcRun.Steps = mcRunSteps
   Set mcRun.Parameters = mcRunParams
   Set mcRun.RunConnections =
mcRunConnections
   Set mcRun.RunConnDtls = mcRunConnDtls

   mcRun.WspId = mlngWorkspaceId
   mcRun.RootKey =
LabelStep(mlngWorkspaceId)
   mcRun.RunId = mlngRunId
   mcRun.CreateInputFiles = CreateInputFiles

   mcRun.Run

   Exit Sub

RunWorkspaceErr:
   ' Log the error code raised by Visual Basic
   Call LogErrors(Errors)

End Sub
Public Property Get LoadDb() As Database

   Set LoadDb = mdbsLoadDb

End Property
Public Property Set LoadDb(vdata As
Database)

   Set mdbsLoadDb = vdata

End Property
Private Function LoadRunData() As Boolean

   ' Loads the step, parameter and constraint
arrays
   ' with all the data for the workspace. Returns
False
   ' if a failure occurs

   Dim strWorkspaceName As String
   Dim recWspSteps As Recordset
   Dim qySteps As DAO.QueryDef
   Dim recWspParams As Recordset
   Dim qyParams As DAO.QueryDef
   Dim recWspConns As Recordset
   Dim qyConns As DAO.QueryDef
   Dim recWspConnDtls As Recordset
   Dim qyConnDtls As DAO.QueryDef

   On Error GoTo LoadRunDataErr

   Set mcRunSteps.StepDB = mdbsLoadDb
   Set mcRunParams.ParamDatabase =
mdbsLoadDb
   Set mcRunConstraints.ConstraintDB =
mdbsLoadDb
   Set mcRunConnections.ConnDb =
mdbsLoadDb
   Set mcRunConnDtls.ConnDb =
mdbsLoadDb

   ' Read all the step and parameter data for the
workspace
   Call
ReadWorkspaceData(mlngWorkspaceId,
mcRunSteps, _
         mcRunParams, mcRunConstraints,
mcRunConnections, mcRunConnDtls, _
         recWspSteps, qySteps, recWspParams,
qyParams, recWspConns, qyConns, _
         recWspConnDtls, qyConnDtls)

   ' Load all the pre- and post-execution
constraints that
   ' have been defined for the workspace
   mcvntWspPreCons =
mcRunConstraints.ConstraintsForWsp( _
      mlngWorkspaceId, _
      gintPreStep, _
      blnSort:=True, _
      blnGlobalConstraintsOnly:=True)
   mcvntWspPostCons =
mcRunConstraints.ConstraintsForWsp( _
      mlngWorkspaceId, _
      gintPostStep, _
      blnSort:=True, _
      blnGlobalConstraintsOnly:=True)

   On Error Resume Next
   recWspSteps.Close
   qySteps.Close
   recWspParams.Close
   qyParams.Close
   recWspConns.Close
   qyConns.Close

   LoadRunData = True

   Exit Function

LoadRunDataErr:
   ' Log the error code raised by Visual Basic
   Call LogErrors(Errors)
   ShowError errLoadRunDataFailed
   LoadRunData = False

End Function
Public Sub StopRun()

   On Error GoTo StopRunErr

   If mcRun Is Nothing Then
      ' We haven't been the run yet, so do nothing
   Else
      mcRun.StopRun
   End If

   Exit Sub

StopRunErr:
   ' Log the error code raised by Visual Basic
   Call LogErrors(Errors)
   ' Errors would have been displayed by the called
process

End Sub
Public Sub AbortRun()

   On Error GoTo AbortRunErr

   If mcRun Is Nothing Then
      ' We haven't been the run yet, so do nothing
   Else
      mcRun.Abort
   End If

   Exit Sub

AbortRunErr:
   ' Log the error code raised by Visual Basic
   Call LogErrors(Errors)
   ' Errors would have been displayed by the called
process

End Sub

Private Sub Class_Initialize()

   ' Create instances of the step, parameter and
constraint arrays
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   Set mcRunSteps = New cArrSteps
   Set mcRunParams = New cArrParameters
   Set mcRunConstraints = New
cArrConstraints
   Set mcRunConnections = New
cConnections
   Set mcRunConnDtls = New cConnDtls
   Set mcRun = New cRunInst
   Set mField = New cStringSM

End Sub
Private Sub Class_Terminate()

   On Error GoTo UnLoadRunDataErr

   ' Clears the step, parameter and constraint
arrays
   Set mcRunSteps = Nothing
   Set mcRunParams = Nothing
   Set mcRunConstraints = Nothing
   Set mcRunConnections = Nothing
   Set mcRunConnDtls = Nothing

   Set mcRun = Nothing
   Set mdbsLoadDb = Nothing
   Set mField = Nothing

   Exit Sub

UnLoadRunDataErr:
   ' Log the error code raised by Visual
Basic
   Call LogErrors(Errors)
   ' Not a critical error - continue
   Resume Next

End Sub

Private Sub
mcRun_ProcessComplete(cStepRecord As
cStep, _
       dtmEndTime As Currency,
lngInstanceId As Long, lElapsed As Long)

   RaiseEvent
ProcessComplete(cStepRecord,
dtmEndTime, lngInstanceId)
   Call
ProcessCompleteProcessing(cStepRecord,
dtmEndTime, lngInstanceId, lElapsed)

End Sub

Private Sub
mcRun_ProcessStart(cStepRecord As cStep,
_
       strCommand As String, dtmStartTime
As Currency, lngInstanceId As Long, _
       lParentInstanceId As Long, sItValue As
String)

   RaiseEvent ProcessStart(cStepRecord,
strCommand, dtmStartTime, lngInstanceId)
   Call ProcessStartProcessing(cStepRecord,
strCommand, dtmStartTime, lngInstanceId,
_
           lParentInstanceId, sItValue)

End Sub

Private Sub
mcRun_RunComplete(dtmEndTime As
Currency)

   Debug.Print "Run ended at: " &
CStr(dtmEndTime)
   Call RunCompleteProcessing(dtmEndTime)

   RaiseEvent RunComplete(dtmEndTime)

End Sub
Private Sub mcRun_RunStart(dtmStartTime
As Currency, strWspLog As String)

   RaiseEvent RunStart(dtmStartTime,
strWspLog, mlngRunId)
   Debug.Print "Run started at: " &
CStr(dtmStartTime)

   Call RunStartProcessing(dtmStartTime)

End Sub
Private Sub
mcRun_StepComplete(cStepRecord As cStep,
_
       dtmEndTime As Currency, lngInstanceId
As Long, lElapsed As Long)

   RaiseEvent StepComplete(cStepRecord,
dtmEndTime, lngInstanceId)
'    BugMessage "Step: " &
cStepRecord.StepLabel & " has completed!"

   Call StepCompleteProcessing(cStepRecord,
dtmEndTime, lngInstanceId, lElapsed)

End Sub
Private Sub mcRun_StepStart(cStepRecord As
cStep, dtmStartTime As Currency, _
       lngInstanceId As Long, lParentInstanceId
As Long, sPath As String, sIts As String,
sItValue As String)

   RaiseEvent StepStart(cStepRecord,
dtmStartTime, lngInstanceId, sPath, sIts)
   'bugmessage "Step: " &
cStepRecord.StepLabel & " has started."

   Call StepStartProcessing(cStepRecord,
dtmStartTime, lngInstanceId,
lParentInstanceId, sItValue)

End Sub

VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cSequence"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:       cSequence.cls
'              Microsoft TPC-H Kit Ver. 1.00
'              Copyright Microsoft, 1999
'              All Rights Reserved
'
'
'  PURPOSE:     This class uses the
att_identifiers table to generate unique
'              identifiers.
'  Contact:     Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

Private mlngIdentifier As Long
Private mstrIdentifierColumn As String
Private mrecIdentifiers As Recordset
Private mdbsDatabase As Database

Private Const mstrEmptyString = ""

' Used to indicate the source module name when
errors
' are raised by this class
Private mstrSource As String
Private Const mstrModuleName As String =
"cSequence."

Private Sub CreateIdRecord()
   ' Creates a record with all identifiers having an
initial value of 1

   Dim sSql As String
   Dim pId As DAO.Parameter
   Dim qyId As DAO.QueryDef

   sSql = "insert into att_identifiers (" & _
          " workspace_id, parameter_id, step_id, " &
_
          " constraint_id, run_id, connection_id " & _
          ", " & FLD_ID_CONN_NAME & _
          " ) values ( " & _
          "[w_id], [p_id], [s_id], [c_id], [r_id],
[conn_id], [conn_dtl_id] )"
   Set qyId =
mdbsDatabase.CreateQueryDef(gstrEmptyString,
sSql)
   For Each pId In qyId.Parameters
      pId.Value = glMinId
   Next pId
   qyId.Execute dbFailOnError
   qyId.Close

End Sub
Private Sub CreateIdRecordset()

   Dim strSql As String

   ' Initialize the recordset with all identifiers
   strSql = "select * from att_identifiers"
   Set mrecIdentifiers =
mdbsDatabase.OpenRecordset(strSql,
dbOpenForwardOnly)

   If mrecIdentifiers.RecordCount = 0 Then
      CreateIdRecord
      Set mrecIdentifiers =
mdbsDatabase.OpenRecordset(strSql,
dbOpenForwardOnly)
   End If

   BugAssert mrecIdentifiers.RecordCount <> 0

End Sub

Public Property Set IdDatabase(vdata As Database)

   Set mdbsDatabase = vdata

End Property

Public Property Let IdentifierColumn(vdata As
String)

   Dim intIndex As Integer

   On Error GoTo IdentifierColumnErr

   ' Initialize the return value to an empty string
   mstrIdentifierColumn = mstrEmptyString
   Call CreateIdRecordset
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   For intIndex = 0 To
mrecIdentifiers.Fields.Count - 1

      If
LCase(Trim(mrecIdentifiers.Fields(intIndex
).Name)) = _
            LCase(Trim(vdata)) Then

         ' Valid column name
         mstrIdentifierColumn = vdata
         Exit Property
      End If

   Next intIndex

   BugAssert True, "Invalid column name!"

   Exit Property

IdentifierColumnErr:
   LogErrors Errors
   mstrSource = mstrModuleName &
"IdentifierColumn"
   On Error GoTo 0
   Err.Raise vbObjectError +
errIdentifierColumnFailed, _
         mstrSource, _

LoadResString(errIdentifierColumnFailed)

End Property
Public Property Get Identifier() As Long
   Dim strSql As String

   On Error GoTo GetIdentifierErr

   BugAssert mstrIdentifierColumn <>
mstrEmptyString

   ' Increment the identifier column by 1
   strSql = "update att_identifiers " & _
      " set " & mstrIdentifierColumn & _
      " = " & mstrIdentifierColumn & " + 1"
   mdbsDatabase.Execute strSql,
dbFailOnError

   ' Refresh the recordset with identifier
values
   Call CreateIdRecordset

   mlngIdentifier =
mrecIdentifiers.Fields(mstrIdentifierColumn
).Value

   Identifier = mlngIdentifier

   Exit Property

GetIdentifierErr:
   LogErrors Errors
   mstrSource = mstrModuleName &
"Identifier"
   On Error GoTo 0
   Err.Raise vbObjectError +
errGetIdentifierFailed, _
         mstrSource, _

LoadResString(errGetIdentifierFailed)

End Property
Private Sub Class_Terminate()

   mrecIdentifiers.Close
End Sub

VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cStack"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:       cStack.cls
'             Microsoft TPC-H Kit Ver. 1.00
'             Copyright Microsoft, 1999
'             All Rights Reserved
'
'
'  PURPOSE:     This class implements a stack
of objects.
'  Contact:     Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

' Used to indicate the source module name
when errors
' are raised by this class
Private Const mstrModuleName As String =
"cStack."
Private mstrSource As String

Private mcVector As cVector
Private mlngCount As Long
Public Property Get Item(ByVal Position As
Long) As Object
Attribute Item.VB_UserMemId = 0

   Set Item = mcVector(Position)

End Property

Public Sub Push(objToPush As Object)

   mcVector.Add objToPush

End Sub
Public Sub Clear()

   mcVector.Clear

End Sub

Public Function Pop() As Object

   If mcVector.Count > 0 Then
      Set Pop =
mcVector.Delete(mcVector.Count - 1)
   Else
      Set Pop = Nothing
   End If

End Function
Public Function Count() As Long

   Count = mcVector.Count

End Function


Private Sub Class_Initialize()

   Set mcVector = New cVector

End Sub

Private Sub Class_Terminate()

   Set mcVector = Nothing

End Sub


VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cStep"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
Attribute VB_Ext_KEY =
"SavedWithClassBuilder" ,"Yes"
Attribute VB_Ext_KEY = "Top_Level" ,"Yes"
'  FILE:       cStep.cls
'             Microsoft TPC-H Kit Ver. 1.00
'             Copyright Microsoft, 1999
'             All Rights Reserved
'
'
'  PURPOSE:     Encapsulates the properties and
methods of a step.
'             Contains functions to insert, update and
delete
'             att_steps records from the database.
'  Contact:     Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

' Local variable(s) to hold property value(s)
Private mlngStepId As Long
Private mstrVersionNo As String
Private mstrStepLabel As String
Private mstrStepTextFile As String
Private mstrStepText As String
Private mstrStartDir As String
Private mlngWorkspaceId As Integer
Private mlngParentStepId As Integer
Private mstrParentVersionNo As String
Private mintSequenceNo As Integer
Private mintStepLevel As Integer
Private mblnEnabledFlag As Boolean
Private mstrDegreeParallelism As String
Private mintExecutionMechanism As Integer
Private mstrFailureDetails As String
Private mintContinuationCriteria As Integer
Private mblnGlobalFlag As Boolean
Private mblnArchivedFlag As Boolean
Private mstrOutputFile As String
'Private mstrLogFile As String
Private mstrErrorFile As String
Private mdbsDatabase As Database
Private mintStepType As Integer
Private mintOperation As Operation
Private mlngPosition As Long
Private mstrIteratorName As String
Private mcIterators As cNodeCollections
Private mbIsNewVersion As Boolean
Private msOldVersion As String

' The following constants are used throughout the
project to
' indicate the different options selected by the user
' The options are presented to the user as control
arrays of
' option buttons. These constants have to be in sync
with the
' indexes of the option buttons.
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
' All the control arrays have an lbound of 1.
The value 0 is
' used to indicate that the property being
represented by the
' control array is not valid for the step
' Public enums are used since we cannot
expose public constants
' in class modules. gintNoOption is
applicable to all enums,
' but declared in the Execution method
enum, since we cannot
' declare it more than once.

' Is here as a comment
' Has been defined in public.bas with the
other object types
'Public Enum gintStepType
'    gintGlobalStep = 3
'    gintManagerStep
'    gintWorkerStep
'End Enum

' Execution Method options
Public Enum ExecutionMethod
    gintNoOption = 0
    gintExecuteODBC
    gintExecuteShell
End Enum

' Failure criteria options
Public Enum FailureCriteria
    gintFailureODBC = 1
    gintFailureTextCompare
End Enum

' Continuation criteria options
' Note: Update the initialization of
gsContCriteria in Initialize() if the
' continuation criteria are modified
Public Enum ContinuationCriteria
    gintOnFailureAbort = 1
    gintOnFailureContinue
    gintOnFailureCompleteSiblings
    gintOnFailureAbortSiblings
    gintOnFailureSkipSiblings
    gintOnFailureAsk
End Enum

' The initial version #
Private Const mstrMinVersion As String =
"0.0"

' End of constants for option button control
arrays
' Used to indicate the source module name
when errors
' are raised by this class
Private mstrSource As String
Private Const mstrModuleName As String =
"cStep."

' The cSequence class is used to generate
unique step identifiers
Private mStepSeq As cSequence

' The StringSM class is used to carry out
string operations
Private mFieldValue As cStringSM
Private Sub NewVersion()

mbIsNewVersion = True
msOldVersion = mstrVersionNo

End Sub
Public Function IsNewVersion() As Boolean
    IsNewVersion = mbIsNewVersion
End Function

Public Function OldVersionNo() As String
    OldVersionNo = msOldVersion
End Function

Public Sub SaveIterators()
    ' This procedure checks if any changes have
been made
    ' to the iterators for the step. If so, it calls the
    ' methods of the iterator class to commit the
changes
    Dim cItRec As cIterator
    Dim lngIndex As Long

    On Error GoTo SaveIteratorsErr

    For lngIndex = 0 To mcIterators.Count - 1
        Set cItRec = mcIterators(lngIndex)

        Select Case cItRec.IndOperation
            Case QueryOp
                ' No changes were made to the
queried Step.
                ' Do nothing

            Case InsertOp
                cItRec.Add mlngStepId,
mstrVersionNo
                cItRec.IndOperation = QueryOp

            Case UpdateOp
                cItRec.Update mlngStepId,
mstrVersionNo
                cItRec.IndOperation = QueryOp

            Case DeleteOp
                cItRec.Delete mlngStepId,
mstrVersionNo
                ' Remove the record from the
collection
                mcIterators.Delete lngIndex

        End Select
    Next lngIndex

    Exit Sub

SaveIteratorsErr:
    LogErrors Errors
    mstrSource = mstrModuleName &
"SaveIterators"
    On Error GoTo 0
    Err.Raise vbObjectError + errSaveFailed, _
            mstrSource, _
            LoadResString(errSaveFailed)

End Sub
Public Property Get IndOperation() As
Operation

    IndOperation = mintOperation

End Property
Public Property Let IndOperation(ByVal vdata
As Operation)

    BugAssert vdata = QueryOp Or vdata =
InsertOp Or vdata = UpdateOp Or vdata =
DeleteOp, "Invalid operation"
    mintOperation = vdata

End Property
Public Function Iterators() As Variant
    ' Returns a variant containing all the iterators that
    ' have been defined for the step

    Dim cStepIterators() As cIterator
    Dim cTempIt As cIterator
    Dim lngIndex As Long
    Dim lngItCount As Long

    On Error GoTo IteratorsErr

    lngItCount = 0
    For lngIndex = 0 To mcIterators.Count - 1
        ' Increase the array dimension and add the
constraint
        ' to it
        Set cTempIt = mcIterators(lngIndex)

        If cTempIt.IndOperation <> DeleteOp Then
            ReDim Preserve cStepIterators(lngItCount)
            Set cStepIterators(lngItCount) = cTempIt
            lngItCount = lngItCount + 1
        End If

    Next lngIndex

    If lngItCount = 0 Then
        Iterators = Empty
    Else
        Iterators = cStepIterators()
    End If

    Call QuickSort(Iterators)

    Exit Function

IteratorsErr:
    LogErrors Errors
    On Error GoTo 0
    Err.Raise vbObjectError + errIteratorsFailed, _
            mstrModuleName & "Iterators", _
            LoadResString(errIteratorsFailed)

End Function
Public Function IteratorCount() As Long
    ' Returns a count of all the iterators for the step

    Dim lngItCount As Long
    Dim lngIndex As Long
    Dim cTempIt As cIterator

    On Error GoTo IteratorsErr

    lngItCount = 0
    For lngIndex = 0 To mcIterators.Count - 1

        If mcIterators(lngIndex).IndOperation <>
DeleteOp Then
            lngItCount = lngItCount + 1
        End If

    Next lngIndex

    IteratorCount = lngItCount

    Exit Function

IteratorsErr:
    LogErrors Errors
    On Error GoTo 0
    Err.Raise vbObjectError + errIteratorsFailed, _
            mstrSource, _
            LoadResString(errIteratorsFailed)

End Function
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
Public Sub Validate()
   ' Each distinct object will have a Validate
method which
   ' will check if the class properties are
valid. This method
   ' will be used to check interdependant
properties that
   ' cannot be validated by the let
procedures.
   ' It should be called by the add and
modify methods of the class

   ' Check if the step label has been specified
   If StringEmpty(mstrStepLabel) Then
      ShowError errStepLabelMandatory
      On Error GoTo 0
      Err.Raise vbObjectError +
errValidateFailed, _
            "Validate",
LoadResString(errValidateFailed)
   End If

   If Not IsStringEmpty(mstrStepText) And
Not IsStringEmpty(mstrStepTextFile) Then
      ShowError errStepTextOrFile
      On Error GoTo 0
      Err.Raise vbObjectError +
errStepTextOrFile, _
         "Validate",
LoadResString(errStepTextOrFile)
   End If

End Sub

Public Function IncVersionY() As String
   ' The version number for a step is stored
in the x.y
   ' format where x is the parent component
and y is the
   ' child component of the step. This
function will increment
   ' the y component of the step by 1

   On Error GoTo IncVersionYErr

   ' Store the old version number for the step
   Call NewVersion

   mstrVersionNo =
Trim$(Str$(GetX(mstrVersionNo))) &
gstrVerSeparator & _
         Trim$(Str(GetY(mstrVersionNo) +
1))
   IncVersionY = mstrVersionNo

   Exit Function

IncVersionYErr:
   ' Log the error code raised by Visual
Basic
   Call LogErrors(Errors)
   gstrSource = mstrModuleName &
"IncVersionY"
   On Error GoTo 0
   Err.Raise vbObjectError +
errIncVersionYFailed, _
         gstrSource, _

LoadResString(errIncVersionYFailed)

End Function
Public Function IncVersionX() As String
   ' The version number for a step is stored
in the x.y
   ' format where x is the parent component and
y is the
   ' child component of the step. This function
will increment
   ' the y component of the step by 1 and reset
the x component
   ' to 0

   On Error GoTo IncVersionXErr

   ' Store the old version number for the step
   Call NewVersion

   mstrVersionNo =
Trim$(Str$(GetX(mstrVersionNo) + 1)) &
gstrVerSeparator & "0"
   IncVersionX = mstrVersionNo

   Exit Function

IncVersionXErr:
   ' Log the error code raised by Visual Basic
   Call LogErrors(Errors)
   gstrSource = mstrModuleName &
"IncVersionX"
   On Error GoTo 0
   Err.Raise vbObjectError +
errIncVersionXFailed, _
         gstrSource, _
         LoadResString(errIncVersionXFailed)

End Function


Private Function GetY(strVersion As String)
As Long
   ' The version number for a step is stored in
the x.y
   ' format where x is the parent component and
y is the
   ' child component of the step. Given an
argument of type
   ' x.y, it returns y

   ' Truncate the fractional part to get the parent
component
   ' of the version number (x.y)
   GetY = Val(Mid(strVersion,
InStr(strVersion, gstrVerSeparator) + 1))

End Function
Private Function GetX(strVersion As String)
As Long
   ' The version number for a step is stored in
the x.y
   ' format where x is the parent component and
y is the
   ' child component of the step. Given an
argument of type
   ' x.y, it returns x

   ' Truncate the fractional part to get the parent
component
   ' of the version number (x.y)
   GetX = Val(Left(strVersion,
InStr(strVersion, gstrVerSeparator) - 1))

End Function

Public Function Clone(Optional cCloneStep As
cStep) As cStep

   ' Creates a copy of a given step

   Dim lngIndex As Long
   Dim cItRec As cIterator
   Dim cItClone As cIterator

   On Error GoTo CloneErr

   If cCloneStep Is Nothing Then
      Set cCloneStep = New cStep
   End If

   ' Copy all the step properties to the newly created
step
   ' Initialize the global flag first since subsequent
   ' validations might depend on it
   cCloneStep.GlobalFlag = mblnGlobalFlag
'   cCloneStep.GlobalRunMethod =
mintGlobalRunMethod

   cCloneStep.StepType = mintStepType
   cCloneStep.StepId = mlngStepId
   cCloneStep.VersionNo = mstrVersionNo
   cCloneStep.StepLabel = mstrStepLabel
   cCloneStep.StepTextFile = mstrStepTextFile
   cCloneStep.StepText = mstrStepText
   cCloneStep.StartDir = mstrStartDir
   cCloneStep.WorkspaceId = mlngWorkspaceId
   cCloneStep.ParentStepId = mlngParentStepId
   cCloneStep.ParentVersionNo =
mstrParentVersionNo
   cCloneStep.StepLevel = mintStepLevel
   cCloneStep.SequenceNo = mintSequenceNo
   cCloneStep.EnabledFlag = mblnEnabledFlag
   cCloneStep.DegreeParallelism =
mstrDegreeParallelism
   cCloneStep.ExecutionMechanism =
mintExecutionMechanism
   cCloneStep.FailureDetails = mstrFailureDetails
   cCloneStep.ContinuationCriteria =
mintContinuationCriteria
   cCloneStep.ArchivedFlag = mblnArchivedFlag
   cCloneStep.OutputFile = mstrOutputFile
'   cCloneStep.LogFile = mstrLogFile
   cCloneStep.ErrorFile = mstrErrorFile
   cCloneStep.IteratorName = mstrIteratorName

   cCloneStep.IndOperation = mintOperation
   cCloneStep.Position = mlngPosition

   Set cCloneStep.NodeDB = mdbsDatabase

   ' Clone all the iterators for the step
   For lngIndex = 0 To mcIterators.Count - 1
      Set cItRec = mcIterators(lngIndex)
      Set cItClone = cItRec.Clone
      cCloneStep.LoadIterator cItClone
   Next lngIndex

   ' And set the return value to the newly created
step
   Set Clone = cCloneStep

   Exit Function

CloneErr:
   LogErrors Errors
   mstrSource = mstrModuleName & "Clone"
   On Error GoTo 0
   Err.Raise vbObjectError + errCloneFailed, _
         mstrSource, LoadResString(errCloneFailed)

End Function
'End Sub
'
Public Property Let OutputFile(ByVal vdata As
String)
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
    mstrOutputFile = vdata

End Property

Public Property Get OutputFile() As String

    OutputFile = mstrOutputFile

End Property

'Public Property Let LogFile(ByVal vdata
As String)
'
'    mstrLogFile = vdata
'
'End Property
'
'Public Property Get LogFile() As String
'
'    LogFile = mstrLogFile
'
'End Property

Public Property Let ErrorFile(ByVal vdata
As String)

    mstrErrorFile = vdata

End Property
Public Property Let IteratorName(ByVal
vdata As String)

    mstrIteratorName = vdata

End Property

Public Property Get ErrorFile() As String

    ErrorFile = mstrErrorFile

End Property
Public Property Get IteratorName() As
String

    IteratorName = mstrIteratorName

End Property

Public Property Set NodeDB(vdata As
Database)

    Set mdbsDatabase = vdata
    Set mcIterators.NodeDB = vdata

End Property
Public Property Get NodeDB() As Database

    Set NodeDB = mdbsDatabase

End Property

Private Function IsStringEmpty(strToCheck
As String) As Boolean

    IsStringEmpty = (strToCheck =
gstrEmptyString)

End Function

Public Property Let EnabledFlag(ByVal
vdata As Boolean)

    ' The enabled flag must be False for all
global steps.
    ' This check must be made by the global step
class. Only
    ' generic step validations will be carried out
by this
    ' class
    mblnEnabledFlag = vdata

End Property

Public Property Let GlobalFlag(ByVal vdata
As Boolean)

    mblnGlobalFlag = vdata

End Property
Public Property Get EnabledFlag() As Boolean

    EnabledFlag = mblnEnabledFlag

End Property

Public Property Let ArchivedFlag(ByVal vdata
As Boolean)

    mblnArchivedFlag = vdata

End Property

Public Property Get ArchivedFlag() As
Boolean

    ArchivedFlag = mblnArchivedFlag

End Property

Public Property Get GlobalFlag() As Boolean

    GlobalFlag = mblnGlobalFlag

End Property

Public Sub Add()
    ' Inserts a step record into the database - it
initializes
    ' the necessary properties for the step and
calls InsertStepRec
    ' to do the database work

    On Error GoTo AddErr

    ' A new record would have the deleted_flag
turned off!
    mblnArchivedFlag = False

    Call InsertStepRec

    ' If a new version of a step has been created,
reset the old version info, since
    ' it's already been saved to the db
    If IsNewVersion() Then
        mbIsNewVersion = False
        msOldVersion = gstrEmptyString
    End If

    Exit Sub

AddErr:
    LogErrors Errors
    On Error GoTo 0
    Err.Raise vbObjectError +
errAddStepFailed, _
            mstrModuleName & "Add",
LoadResString(errAddStepFailed)
End Sub
Private Sub InsertStepRec()
    ' Inserts a step record into the database
    ' It first generates the insert statement using the
different
    ' step properties and then executes it

    Dim strInsert As String
    Dim qy As DAO.QueryDef

    On Error GoTo InsertStepRecErr

    ' First check if the database object is valid
    Call CheckDB

    ' Check if the step record is valid
    Call Validate

    If IsNewVersion() Then
        Call UpdOldVersionsArchFlg
    End If

    ' Create a temporary querydef object
    strInsert = "insert into att_steps " & _
        "( workspace_id, step_id, version_no, " & _
        " step_label, step_file_name, step_text,
start_directory, " & _
        " parent_step_id, parent_version_no,
sequence_no, " & _
        " enabled_flag, step_level, " & _
        " degree_parallelism, execution_mechanism, "
& _
        " failure_details, " & _
        " continuation_criteria, global_flag, " & _
        " archived_flag, " & _
        " output_file_name, error_file_name, " & _
        " iterator_name ) values ( "

    ' log_file_name,

#If USE_JET Then

    strInsert = strInsert & " [w_id], [s_id], [ver_no], "
& _
        " [s_label], [s_file_name], [s_text],
[s_start_dir], " & _
        " [p_step_id], [p_version_no], [seq_no], " & _
        " [enabled], [s_level], [deg_parallelism], " & _
        " [exec_mechanism], [fail_dtls], " & _
        " [cont_criteria], [global], [archived], " & _
        " [output_file], [error_file], " & _
        " [it_name] ) "

    ' [log_file],

    Set qy =
mdbsDatabase.CreateQueryDef(gstrEmptyString,
strInsert)

    ' Call a procedure to execute the Querydef object
    Call AssignParameters(qy)

    qy.Execute dbFailOnError
    qy.Close
#Else

        strInsert = strInsert & Str(mlngWorkspaceId)
& ", " & Str(mlngStepId) & _
        ", " &
mFieldValue.MakeStringFieldValid(mstrVersionNo
)

    ' For fields that may be null, call a function to
determine
    ' the string to be appended to the insert statement
    strInsert = strInsert & ", " &
mFieldValue.MakeStringFieldValid(mstrStepLabel)
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   strInsert = strInsert & ", " &
mFieldValue.MakeStringFieldValid(mstrSte
pTextFile)
   strInsert = strInsert & ", " &
mFieldValue.MakeStringFieldValid(mstrSte
pText)
   strInsert = strInsert & ", " &
mFieldValue.MakeStringFieldValid(mstrSta
rtDir)

   strInsert = strInsert & ", " &
Str(mlngParentStepId) & _
       ", " &
mFieldValue.MakeStringFieldValid(mstrPar
entVersionNo) & _
       ", " & Str(mintSequenceNo) & _
       ", " & Str(mblnEnabledFlag) & ", " &
Str(mintStepLevel)

   strInsert = strInsert & ", " &
mFieldValue.MakeStringFieldValid(mstrDe
greeParallelism)
   strInsert = strInsert & ", " &
Str(mintExecutionMechanism)

   strInsert = strInsert & ", " &
mFieldValue.MakeStringFieldValid(mstrFai
lureDetails) & _
       ", " & Str(mintContinuationCriteria) &
_
       ", " & Str(mblnGlobalFlag) & _
       ", " & Str(mblnArchivedFlag)

   strInsert = strInsert & ", " &
mFieldValue.MakeStringFieldValid(mstrOu
tputFile)
'   strInsert = strInsert & ", " &
mFieldValue.MakeStringFieldValid(mstrLo
gFile)
   strInsert = strInsert & ", " &
mFieldValue.MakeStringFieldValid(mstrErr
orFile)
   strInsert = strInsert & ", " &
mFieldValue.MakeStringFieldValid(mstrIte
ratorName)

   strInsert = strInsert & " ) "

   BugMessage strInsert
   mdbsDatabase.Execute strInsert,
dbFailOnError

#End If

   Exit Sub

InsertStepRecErr:
   mstrSource = mstrModuleName &
"InsertStepRec"
   LogErrors Errors
   On Error GoTo 0
   Err.Raise vbObjectError +
errInsertStepFailed, _
         mstrSource,
LoadResString(errInsertStepFailed)
End Sub
Private Sub UpdOldVersionsArchFlg()
   ' Updates the archived flag on all old
version for the step to True

   Dim sUpdate As String
   Dim qy As DAO.QueryDef

   On Error GoTo
UpdOldVersionsArchFlgErr
   mstrSource = mstrModuleName &
"UpdOldVersionsArchFlg"

#If USE_JET Then

   sUpdate = "update att_steps " & _
      " set archived_flag = True "

   ' Append the Where clause
   sUpdate = sUpdate & " where step_id =
[s_id] " & _
      " and version_no <> [ver_no]"

   Set qy =
mdbsDatabase.CreateQueryDef(gstrEmptyStri
ng, sUpdate)

   ' Call a procedure to execute the Querydef
object
   Call AssignParameters(qy)
   qy.Execute dbFailOnError

   If qy.RecordsAffected = 0 Then
      On Error GoTo 0
      Err.Raise vbObjectError +
errModifyStepFailed, _
            mstrSource,
LoadResString(errModifyStepFailed)
   End If

   qy.Close

#Else

   sUpdate = "update att_steps " & _
      " set archived_flag = True "

   sUpdate = sUpdate & " where step_id = " &
Str(mlngStepId) & _
      " and version_no <> " &
mFieldValue.MakeStringFieldValid(mstrVersi
onNo)

   BugMessage sUpdate
   mdbsDatabase.Execute sUpdate,
dbFailOnError
#End If

   Exit Sub

UpdOldVersionsArchFlgErr:
   mstrSource = mstrModuleName &
"UpdOldVersionsArchFlg"
   LogErrors Errors
   On Error GoTo 0
   Err.Raise vbObjectError +
errModifyStepFailed, _
         mstrSource,
LoadResString(errModifyStepFailed)
End Sub
Public Sub InsertIterator(cItRecord As
cIterator)
   ' Inserts the iterator record into the database

   Call cItRecord.Add(mlngStepId,
mstrVersionNo)

End Sub
Public Sub UpdateIterator(cItRecord As
cIterator)
   ' Updates the iterator record in the database

   Call cItRecord.Update(mlngStepId,
mstrVersionNo)
End Sub
Public Sub UpdateIteratorVersion()
   ' Updates the iterator record in the database

   Dim lngIndex As Long
   Dim cTempIt As cIterator

   On Error GoTo UpdateIteratorVersionErr

   For lngIndex = 0 To mcIterators.Count - 1
      ' Increase the array dimension and add the
constraint
      ' to it
      Set cTempIt = mcIterators(lngIndex)

      If cTempIt.IndOperation <> DeleteOp Then
         ' Set the operation to indicate an insert
         cTempIt.IndOperation = InsertOp
      End If

   Next lngIndex

   Exit Sub

UpdateIteratorVersionErr:
   mstrSource = mstrModuleName &
"UpdateIteratorVersion"
   LogErrors Errors
   On Error GoTo 0
   Err.Raise vbObjectError + errUpdateFailed, _
         mstrSource,
LoadResString(errUpdateFailed)

End Sub
Public Sub AddIterator(cItRecord As cIterator)
   ' Adds the iterator record to the collection of
iterators
   ' for the step

   Call mcIterators.Add(cItRecord)

End Sub
Public Sub AddAllIterators()
   ' Sets the indicator variable for all iterators to
insert

   Dim lngIndex As Long

   For lngIndex = 0 To mcIterators.Count - 1
      mcIterators(lngIndex).Validate
      mcIterators(lngIndex).IndOperation = InsertOp
   Next lngIndex

End Sub

Public Sub LoadIterator(cItRecord As cIterator)
   ' Adds the iterator record to the collection of
iterators
   ' for the step

   Call mcIterators.Load(cItRecord)

End Sub
Public Sub UnloadIterators()
   ' Unloads all iterator records for the step

   Dim lngIndex As Long

   For lngIndex = mcIterators.Count - 1 To 0 Step -
1
      ' Calls the collection method to unload the
node
      ' from the array
      mcIterators.Unload lngIndex
   Next lngIndex
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
End Sub
Public Sub ModifyIterator(cItRecord As
cIterator)
   ' Modifies the iterator record in the
collection

   Call mcIterators.Modify(cItRecord)

End Sub
Public Sub DeleteIterator(cItRecord As
cIterator)
   ' Deletes the iterator record from the
database

   Call cItRecord.Delete(mlngStepId,
mstrVersionNo)

End Sub
Public Sub RemoveIterator(cItRecord As
cIterator)
   ' Marks the iterator record in the
collection to
   ' indicate a delete

   Call
mcIterators.Delete(cItRecord.Position)

End Sub


Private Sub AssignParameters(qyExec As
DAO.QueryDef)
   ' Assigns values to the parameters in the
querydef object
   ' The parameter names are cryptic to make
them different
   ' from the actual field names. When the
parameter names
   ' are the same as the field names,
parameters in the
   ' where clause do not get created.

   Dim prmParam As DAO.Parameter

   On Error GoTo AssignParametersErr
   mstrSource = mstrModuleName &
"AssignParameters"

   For Each prmParam In
qyExec.Parameters
      Select Case prmParam.Name
         Case "[w_id]"
            prmParam.Value =
mlngWorkspaceId
         Case "[s_id]"
            prmParam.Value = mlngStepId
         Case "[ver_no]"
            prmParam.Value =
mstrVersionNo
         Case "[s_label]"
            prmParam.Value = mstrStepLabel
         Case "[s_file_name]"
            prmParam.Value =
mstrStepTextFile
         Case "[s_text]"
            prmParam.Value = mstrStepText
         Case "[s_start_dir]"
            prmParam.Value = mstrStartDir
         Case "[p_step_id]"
            prmParam.Value =
mlngParentStepId
         Case "[p_version_no]"
            prmParam.Value =
mstrParentVersionNo
         Case "[seq_no]"
            prmParam.Value = mintSequenceNo
         Case "[enabled]"
            prmParam.Value = mblnEnabledFlag
         Case "[s_level]"
            prmParam.Value = mintStepLevel
         Case "[deg_parallelism]"
            prmParam.Value =
mstrDegreeParallelism
         Case "[exec_mechanism]"
            prmParam.Value =
mintExecutionMechanism
         Case "[fail_dtls]"
            prmParam.Value =
mstrFailureDetails
         Case "[cont_criteria]"
            prmParam.Value =
mintContinuationCriteria
         Case "[global]"
            prmParam.Value = mblnGlobalFlag
         Case "[archived]"
            prmParam.Value =
mblnArchivedFlag
         Case "[output_file]"
            prmParam.Value = mstrOutputFile
'         Case "[log_file]"
'            prmParam.Value = mstrLogFile
         Case "[error_file]"
            prmParam.Value = mstrErrorFile
         Case "[it_name]"
            prmParam.Value = mstrIteratorName
         Case Else
            ' Write the parameter name that is
faulty
            WriteError errInvalidParameter,
mstrSource, _
                  prmParam.Name
            On Error GoTo 0
            Err.Raise errInvalidParameter,
mstrSource, _

LoadResString(errInvalidParameter)
      End Select
   Next prmParam

   If qyExec.Parameters("s_id") = 0 Or
StringEmpty(qyExec.Parameters("ver_no"))
Then
      WriteError errInvalidParameter,
mstrSource
      On Error GoTo 0
      Err.Raise errInvalidParameter,
mstrSource,
LoadResString(errInvalidParameter)
   End If

   Exit Sub

AssignParametersErr:

   mstrSource = mstrModuleName &
"AssignParameters"
   Call LogErrors(Errors)
   On Error GoTo 0
   Err.Raise vbObjectError +
errAssignParametersFailed, _
      mstrSource,
LoadResString(errAssignParametersFailed)

End Sub

Public Sub Modify()

   Dim strUpdate As String
   Dim qy As QueryDef

   On Error GoTo ModifyErr
   mstrSource = mstrModuleName & "Modify"

   ' Check if the database object is valid
   Call CheckDB

   ' Check if the step record is valid
   Call Validate

   ' The step_id and version_no will never be
updated -
   ' whenever a step is modified a copy of the old
step will
   ' be created with an incremented version_no

#If USE_JET Then

   strUpdate = "update att_steps " & _
      " set step_label = [s_label] " & _
      " , step_file_name = [s_file_name] " & _
      " , step_text = [s_text] " & _
      " , start_directory = [s_start_dir] " & _
      " , workspace_id = [w_id] " & _
      " , parent_step_id = [p_step_id] " & _
      " , parent_version_no = [p_version_no] " & _
      " , sequence_no = [seq_no] " & _
      " , step_level = [s_level] " & _
      " , enabled_flag = [enabled] " & _
      " , degree_parallelism = [deg_parallelism] " &
_
      " , execution_mechanism = [exec_mechanism]
" & _
      " , failure_details = [fail_dtls] " & _
      " , continuation_criteria = [cont_criteria] " & _
      " , global_flag = [global] " & _
      " , archived_flag = [archived] " & _
      " , output_file_name = [output_file] " & _
      " , error_file_name = [error_file] " & _
      " , iterator_name = [it_name] "

      ' " , log_file_name = [log_file] " & _

   ' Append the Where clause
   strUpdate = strUpdate & " where step_id = [s_id]
" & _
      " and version_no = [ver_no]"

   Set qy =
mdbsDatabase.CreateQueryDef(gstrEmptyString,
strUpdate)

   ' Call a procedure to execute the Querydef object
   Call AssignParameters(qy)
   qy.Execute dbFailOnError

   If qy.RecordsAffected = 0 Then
      On Error GoTo 0
      Err.Raise vbObjectError +
errModifyStepFailed, _
            mstrSource,
LoadResString(errModifyStepFailed)
   End If

   qy.Close

#Else

   strUpdate = "update att_steps " & _
      " set step_label = "

   ' For fields that may be null, call a function to
determine
   ' the string to be appended to the update statement
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   strUpdate = strUpdate &
mFieldValue.MakeStringFieldValid(mstrSte
pLabel)

   strUpdate = strUpdate & " ,
step_file_name = " &
mFieldValue.MakeStringFieldValid(mstrSte
pTextFile)
   strUpdate = strUpdate & " , step_text = "
&
mFieldValue.MakeStringFieldValid(mstrSte
pText)
   strUpdate = strUpdate & " ,
start_directory = " &
mFieldValue.MakeStringFieldValid(mstrSta
rtDir)

   strUpdate = strUpdate & " , workspace_id
= " & Str(mlngWorkspaceId) & _
       " , parent_step_id = " &
Str(mlngParentStepId) & _
       " , parent_version_no = " &
mFieldValue.MakeStringFieldValid(mstrPar
entVersionNo) & _
       " , sequence_no = " &
Str(mintSequenceNo) & _
       " , step_level = " & Str(mintStepLevel)
& _
       " , enabled_flag = " &
Str(mblnEnabledFlag) & _
       " , degree_parallelism = " &
mFieldValue.MakeStringFieldValid(mstrDe
greeParallelism) & _
       " , execution_mechanism = " &
Str(mintExecutionMechanism) & _
       " , failure_details = " &
mFieldValue.MakeStringFieldValid(mstrFai
lureDetails) & _
       " , continuation_criteria = " &
Str(mintContinuationCriteria) & _
       " , global_flag = " &
Str(mblnGlobalFlag) & _
       " , archived_flag = " &
Str(mblnArchivedFlag) & _
       " , output_file_name = " &
mFieldValue.MakeStringFieldValid(mstrOu
tputFile) & _
       " , error_file_name = " &
mFieldValue.MakeStringFieldValid(mstrErr
orFile) & _
       " , iterator_name = " &
mFieldValue.MakeStringFieldValid(mstrIter
atorName)

'      " , log_file_name = " &
mFieldValue.MakeStringFieldValid(mstrLo
gFile) & _

   strUpdate = strUpdate & " where step_id
= " & Str(mlngStepId) & _
       " and version_no = " &
mFieldValue.MakeStringFieldValid(mstrVe
rsionNo)

   BugMessage strUpdate
   mdbsDatabase.Execute strUpdate,
dbFailOnError
#End If

   Exit Sub

ModifyErr:
   LogErrors Errors
   mstrSource = mstrModuleName &
"Modify"
   On Error GoTo 0
   Err.Raise vbObjectError +
errModifyStepFailed, _
         mstrSource,
LoadResString(errModifyStepFailed)
End Sub
Private Sub CheckDB()
   ' Check if the database object has been
initialized

   If mdbsDatabase Is Nothing Then
      ShowError errInvalidDB
      On Error GoTo 0
      Err.Raise vbObjectError + errInvalidDB,
_
         mstrModuleName,
LoadResString(errInvalidDB)
   End If

End Sub

Public Sub Delete()

   Dim strDelete As String
   Dim qy As DAO.QueryDef

   On Error GoTo DeleteErr

   Call CheckDB

   strDelete = "delete from att_steps " & _
         " where step_id = [s_id] " & _
         " and version_no = [ver_no] "
'   mdbsDatabase.Execute strDelete,
dbFailOnError
   Set qy =
mdbsDatabase.CreateQueryDef(gstrEmptyStri
ng, strDelete)

   Call AssignParameters(qy)
   qy.Execute dbFailOnError

   qy.Close

   Exit Sub

DeleteErr:
   LogErrors Errors
   On Error GoTo 0
   Err.Raise vbObjectError +
errDeleteStepFailed, _
         mstrModuleName & "Delete",
LoadResString(errDeleteStepFailed)
End Sub
Public Property Get DegreeParallelism() As
String

   DegreeParallelism = mstrDegreeParallelism

End Property
Public Property Get Position() As Long

   Position = mlngPosition

End Property

Public Property Let DegreeParallelism(ByVal
vdata As String)

   ' The degree of parallelism must be zero for
all global steps
   ' This check must be made by the global step
class. Only
   ' generic step validations will be carried out
by this
   ' class
   mstrDegreeParallelism = vdata

End Property

Public Property Let ExecutionMechanism(ByVal
vdata As ExecutionMethod)

   BugAssert vdata = gintExecuteODBC Or vdata =
gintExecuteShell Or vdata = gintNoOption, _
         "Execution mechanism invalid"
   mintExecutionMechanism = vdata

End Property

Public Property Let FailureDetails(ByVal vdata As
String)

   mstrFailureDetails = vdata

End Property
Public Property Let SequenceNo(ByVal vdata As
Integer)
   mintSequenceNo = vdata
End Property

Public Property Let Position(ByVal vdata As Long)
   mlngPosition = vdata
End Property

Public Property Let ParentStepId(ByVal vdata As
Long)
   mlngParentStepId = vdata
End Property

Public Property Get SequenceNo() As Integer

   SequenceNo = mintSequenceNo

End Property

Public Property Get StepLevel() As Integer
   StepLevel = mintStepLevel
End Property

Public Property Get ParentVersionNo() As String
   ParentVersionNo = mstrParentVersionNo
End Property

Public Property Let ParentVersionNo(ByVal vdata
As String)
   mstrParentVersionNo = vdata
End Property

Public Property Get ParentStepId() As Long
   ParentStepId = mlngParentStepId
End Property

Public Property Let WorkspaceId(ByVal vdata As
Long)
   mlngWorkspaceId = vdata
End Property

Public Property Let VersionNo(ByVal vdata As
String)
   ' The version number of a step is stored in the x.y
format where
   ' x represents a change to the step as a result of
modifications
   ' to any of the step properties
   ' y represents a change to the step as a result of
modifications
   ' to the sub-steps associated with it. Hence the y-
component
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
    ' of the version will be incremented when
a sub-step is added,
    ' modified or deleted
    ' x will be referred to throughout this code
as the parent
    ' component of the version and y will be
referred to as the
    ' child component of the version
    ' The version information for a step is
maintained by the
    ' calling function

    mstrVersionNo = vdata

End Property

Public Property Get StepType() As
gintStepType

    On Error GoTo StepTypeErr

    If mintStepType = 0 Then
        ' The step type variable has not been
initialized -
        If mblnGlobalFlag Then
            mintStepType = gintGlobalStep
        ElseIf IsStringEmpty(mstrStepText)
And _
                IsStringEmpty(mstrStepTextFile)
Then
            mintStepType = gintManagerStep
        Else
            mintStepType = gintWorkerStep
        End If
    End If

    StepType = mintStepType

    Exit Property

StepTypeErr:
    LogErrors Errors
    mstrSource = mstrModuleName &
"StepType"
    On Error GoTo 0
    Err.Raise vbObjectError +
errGetStepTypeFailed, _
            mstrSource, _

LoadResString(errGetStepTypeFailed)

End Property

Public Property Let StepType(vdata As
gintStepType)

    On Error GoTo StepTypeErr

    Select Case vdata
        Case gintGlobalStep, gintManagerStep,
gintWorkerStep
            mintStepType = vdata

        Case Else
            On Error GoTo 0
            Err.Raise vbObjectError +
errStepTypeInvalid, _
                    mstrModuleName &
"StepType",
LoadResString(errStepTypeInvalid)
    End Select
    Exit Property

StepTypeErr:
    LogErrors Errors
    mstrSource = mstrModuleName &
"StepType"
    On Error GoTo 0
    Err.Raise vbObjectError +
errLetStepTypeFailed, _
            mstrSource, _
            LoadResString(errLetStepTypeFailed)

End Property

Public Property Get WorkspaceId() As Long
    WorkspaceId = mlngWorkspaceId
End Property

Public Property Get ContinuationCriteria() As
ContinuationCriteria

    ContinuationCriteria =
mintContinuationCriteria

End Property

Public Property Let
ContinuationCriteria(ByVal vdata As
ContinuationCriteria)

    ' The Continuation criteria must be null for
all global steps
    ' and non-null for all manager and worker
steps
    ' These checks will have to be made by the
corresponding
    ' classes - only generic step validations will
be made
    ' by this class
    BugAssert vdata =
gintOnFailureAbortSiblings Or vdata =
gintOnFailureCompleteSiblings _
                Or vdata =
gintOnFailureSkipSiblings Or vdata =
gintOnFailureAbort _
                Or vdata = gintOnFailureContinue
Or vdata = gintOnFailureAsk _
                Or vdata = gintNoOption, _
                "Invalid continuation criteria"
    mintContinuationCriteria = vdata

End Property
Public Property Get ExecutionMechanism() As
ExecutionMethod

    ExecutionMechanism =
mintExecutionMechanism

End Property


Public Property Get FailureDetails() As String

    FailureDetails = mstrFailureDetails

End Property

Public Property Let StepText(ByVal vdata As
String)
    ' Has to be null for manager steps
    ' The check will have to be made by the user
interface or
    ' by the manager step class
    mstrStepText = vdata
End Property
Public Property Let StepLevel(ByVal vdata As
Integer)
    ' The step level must be zero for all global steps
    ' This check must be made in the global step class
    mintStepLevel = vdata

End Property
Public Property Get StepText() As String
    StepText = mstrStepText
End Property

Public Property Let StepTextFile(ByVal vdata As
String)
    ' Has to be null for manager steps
    ' The check will have to be made by the user
interface and
    ' by the manager step class
    mstrStepTextFile = vdata
End Property

Public Property Get StepTextFile() As String
    StepTextFile = mstrStepTextFile
End Property

Public Property Let StepLabel(ByVal vdata As
String)
    ' Cannot be null for manager steps
    ' But this check cannot be made here since we do
not know
    ' at this point if the step being created is a
manager
    ' or a worker step
    ' The check will have to be made by the user
interface and
    ' by the manager step class
    mstrStepLabel = vdata
End Property

Public Property Get StepLabel() As String
    StepLabel = mstrStepLabel
End Property

Public Property Let StartDir(ByVal vdata As
String)
    mstrStartDir = vdata
End Property

Public Property Get StartDir() As String
    StartDir = mstrStartDir
End Property

Public Property Get VersionNo() As String
    ' The version number of a step is stored in the x.y
format where
    ' x represents a change to the step as a result of
modifications
    ' to any of the step properties
    ' y represents a change to the step as a result of
modifications
    ' to the sub-steps associated with it. Hence the y-
component
    ' of the version will be incremented when a sub-
step is added,
    ' modified or deleted
    ' x will be referred to throughout this code as the
parent
    ' component of the version and y will be referred
to as the
    ' child component of the version
    ' The version information for a step is maintained
by the
    ' calling function

    VersionNo = mstrVersionNo

End Property
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
Public Property Get StepId() As Long

   StepId = mlngStepId

End Property
Public Property Get NextStepId() As Long

   Dim lngNextId As Long

   On Error GoTo NextStepIdErr

   ' First check if the database object is valid
   Call CheckDB

   ' Retrieve the next identifier using the
sequence class
   Set mStepSeq = New cSequence
   Set mStepSeq.IdDatabase =
mdbsDatabase
   mStepSeq.IdentifierColumn = "step_id"
   lngNextId = mStepSeq.Identifier
   Set mStepSeq = Nothing

   NextStepId = lngNextId
   Exit Property

NextStepIdErr:
   LogErrors Errors
   mstrSource = mstrModuleName &
"NextStepId"
   On Error GoTo 0
   Err.Raise vbObjectError +
errStepIdGetFailed, _
       mstrSource,
LoadResString(errStepIdGetFailed)

End Property
Public Property Let StepId(ByVal vdata As
Long)

   mlngStepId = vdata

End Property

Private Sub Class_Initialize()

   ' Initialize the operation indicator variable
to Query
   ' It will be modified later by the collection
class when
   ' inserts, updates or deletes are performed
   mintOperation = QueryOp
   mbIsNewVersion = False
   msOldVersion = gstrEmptyString

   Set mFieldValue = New cStringSM
   Set mcIterators = New cNodeCollections

End Sub

Private Sub Class_Terminate()

   Set mFieldValue = Nothing
   Set mcIterators = Nothing

End Sub


VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cStepTree"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:       cStepTree.cls
'              Microsoft TPC-H Kit Ver. 1.00
'              Copyright Microsoft, 1999
'              All Rights Reserved
'
'
'  PURPOSE:    Implements step navigation
functions such as determining
'              the child of a step and so on.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

' Used to indicate the source module name
when errors
' are raised by this class
Private Const mstrModuleName As String =
"cStepTree."
Private mstrSource As String

Public StepRecords As cArrSteps
Public Property Get HasChild(Optional ByVal
StepKey As String, _
       Optional ByVal StepId As Long = 0) As
Boolean

   Dim lTemp As Long

   HasChild = False
   StepId = GetStepId(StepKey, StepId)

   For lTemp = 0 To StepRecords.StepCount -
1
       If StepRecords(lTemp).StepType <>
gintGlobalStep And
StepRecords(lTemp).ParentStepId = StepId
Then
           HasChild = True
           Exit For
       End If
   Next lTemp

End Property
Public Property Get ChildStep(Optional ByVal
StepKey As String, _
       Optional ByVal StepId As Long = 0) As
cStep

   Dim lTemp As Long

   Set ChildStep = Nothing
   StepId = GetStepId(StepKey, StepId)

   For lTemp = 0 To StepRecords.StepCount -
1
       If StepRecords(lTemp).StepType <>
gintGlobalStep And
StepRecords(lTemp).ParentStepId = StepId
And _
               StepRecords(lTemp).SequenceNo =
gintMinSequenceNo Then
           Set ChildStep = StepRecords(lTemp)
           Exit For
       End If
   Next lTemp

End Property
Public Property Get NextStep(Optional ByVal
StepKey As String, _
       Optional ByVal StepId As Long = 0) As
cStep

   Dim lTemp As Long
   Dim cChildStep As cStep

   Set NextStep = Nothing
   StepId = GetStepId(StepKey, StepId)
   Set cChildStep = StepRecords.QueryStep(StepId)

   For lTemp = 0 To StepRecords.StepCount - 1
       If StepRecords(lTemp).StepType <>
gintGlobalStep And _
               StepRecords(lTemp).ParentStepId =
cChildStep.ParentStepId And _
               StepRecords(lTemp).SequenceNo =
cChildStep.SequenceNo + 1 Then
           Set NextStep = StepRecords(lTemp)
           Exit For
       End If
   Next lTemp

End Property
Private Function GetStepId(Optional ByVal
StepKey As String, _
       Optional ByVal StepId As Long = 0) As Long
   If StepId = 0 Then
       If StringEmpty(StepKey) Then
           Err.Raise vbObjectError +
errMandatoryParameterMissing, _
                   mstrModuleName & "GetStepId",
LoadResString(errMandatoryParameterMissing)
       Else
           GetStepId = IIf(IsLabel(StepKey), 0,
MakeIdentifierValid(StepKey))
       End If
   Else
       GetStepId = StepId
   End If
End Function
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cStringSM"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:       cStringSM.cls
'              Microsoft TPC-H Kit Ver. 1.00
'              Copyright Microsoft, 1999
'              All Rights Reserved
'
'
'  PURPOSE:    This module contains common
procedures that can be used
'              to manipulate strings
'              It is called StringSM, since String is a
Visual Basic keyword
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

' Used to indicate the source module name when
errors
' are raised by this class
Private mstrSource As String
Private Const mstrModuleName As String =
"cStringSM."

Private mstrText As String

Private Const mstrNullValue = "null"
Private Const mstrSQ = """"
Private Const mstrEnvVarSeparator = "%"
Public Function InsertEnvVariables( _
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
      Optional ByVal strComString As
String) As String
   ' This function replaces all environment
variables in
   ' the passed in string with their values -
they are
   ' enclosed by "%"

   Dim intPos As Integer
   Dim intEndPos As Integer
   Dim strEnvVariable As String
   Dim strValue As String
   Dim strCommand As String

   On Error GoTo InsertEnvVariablesErr
   mstrSource = mstrModuleName &
"InsertEnvVariables"

   ' Initialize the return value of the function
to the
   ' passed in command
   If IsStringEmpty(strComString) Then
      strCommand = mstrText
   Else
      strCommand = strComString
   End If

   intPos = InStr(strCommand,
mstrEnvVarSeparator)
   Do While intPos <> 0
      ' Extract the environment variable from
the passed
      ' in string
      intEndPos = InStr(intPos + 1,
strCommand, mstrEnvVarSeparator)
      strEnvVariable = Mid(strCommand,
intPos + 1, intEndPos - intPos - 1)

      ' Get the value of the variable and call a
function
      ' to replace the variable with it's value
      strValue = Environ$(strEnvVariable)
      strCommand =
ReplaceSubString(strCommand, _
            mstrEnvVarSeparator &
strEnvVariable & mstrEnvVarSeparator, _
            strValue)

      intPos = InStr(strCommand,
mstrEnvVarSeparator)
   Loop

   InsertEnvVariables = strCommand
   Exit Function

InsertEnvVariablesErr:
   ' Log the error code raised by Visual
Basic
   Call LogErrors(Errors)
   ' Return an empty string
   InsertEnvVariables = gstrEmptyString

End Function
Public Function MakeStringFieldValid( _
      Optional strField As String =
gstrEmptyString) As String
   ' Returns a string that can be appended to
any insert
   ' or modify (sql) statement
   ' If an argument is not passed to this
function, the
   ' default text property is used

   Dim strTemp As String
   On Error GoTo MakeStringFieldValidErr

   If IsStringEmpty(strField) Then
      strTemp = mstrText
   Else
      strTemp = strField
   End If

   ' It checks whether the text is empty
   ' If so, it returns the string, "null"
   If IsStringEmpty(strTemp) Then
      MakeStringFieldValid = mstrNullValue
   Else
      ' Single-quotes have to be replaced by two
single-quotes,
      ' since a single-quote is the identifier
delimiter
      ' character - call a procedure to do the
replace
      strTemp = ReplaceSubString(strTemp,
mstrSQ, mstrSQ & mstrSQ)

      ' Replace pipe characters with the
corresponding chr function
      strTemp = ReplaceSubString(strTemp, "|",
"' & Chr(124) & "')

      ' Enclose the string in single quotes
      MakeStringFieldValid = mstrSQ &
strTemp & mstrSQ

   End If

   Exit Function

MakeStringFieldValidErr:
   mstrSource = mstrModuleName &
"MakeStringFieldValid"
   LogErrors Errors
   On Error GoTo 0
   Err.Raise vbObjectError +
errMakeFieldValidFailed, _
         mstrSource,
LoadResString(errMakeFieldValidFailed)

End Function
Public Function MakeDateFieldValid( _
      Optional dtmField As Date = gdtmEmpty)
As String
   ' Returns a string that can be appended to
any insert
   ' or modify (sql) statement

   ' Enclose the date in single quotes
   MakeDateFieldValid = mstrSQ & dtmField
& mstrSQ

End Function

Private Function IsStringEmpty(strToCheck
As String) As Boolean

   If strToCheck = gstrEmptyString Then
      IsStringEmpty = True
   Else
      IsStringEmpty = False
   End If

End Function
Public Function ReplaceSubString(ByVal
MainString As String, _
      ByVal ReplaceString As String, _
      ByVal ReplaceWith As String) As String

   ' Replaces all occurrences of ReplaceString in
MainString with ReplaceWith

   Dim intPos As Integer
   Dim strTemp As String

   On Error GoTo ReplaceSubStringErr

   strTemp = MainString

   intPos = InStr(strTemp, ReplaceString)
   Do While intPos <> 0
      strTemp = Left(strTemp, intPos - 1) &
ReplaceWith & _
            Mid(strTemp, intPos +
Len(ReplaceString))
      intPos = InStr(intPos + Len(ReplaceString) +
1, strTemp, ReplaceString)
   Loop
   ReplaceSubString = strTemp

   Exit Function

ReplaceSubStringErr:
   Call LogErrors(Errors)
   mstrSource = mstrModuleName &
"ReplaceSubString"
   On Error GoTo 0
   Err.Raise vbObjectError + errParseStringFailed,
_
      mstrSource, _
      LoadResString(errParseStringFailed)

End Function

Public Property Get Text() As String
Attribute Text.VB_UserMemId = 0
   Text = mstrText
End Property

Public Property Let Text(ByVal vdata As String)
   mstrText = vdata
End Property

VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cSubStep"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:       cSubStep.cls
'             Microsoft TPC-H Kit Ver. 1.00
'             Copyright Microsoft, 1999
'             All Rights Reserved
'
'
'  PURPOSE:    This module encapsulates the
properties of sub-steps
'             that are used during the execution of a
workspace.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

' Used to indicate the source module name when
errors
' are raised by this class
Private Const mstrModuleName As String =
"cSubStep"

Private mlngStepId As Long
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
Private mintRunning As Integer ' Number of
running tasks
Private mintComplete As Integer ' Number
of completed tasks
' The last iterator for this sub-step
Private mcLastIterator As cRunItDetails

Public Function NewIteration(cStepRec As
cStep) As cIterator
   ' Calls a procedure to determine the next
iterator value
   ' for the passed in step - returns the value
to be used
   ' in the iteration.
   ' It updates the instance node with the new
iteration
   ' for the step.

   Dim cItRec As cIterator

   On Error GoTo NewIterationErr

   ' Call a function that will populate an
iterator record
   ' with the iterator values
   Set cItRec = NextIteration(cStepRec)

   ' Initialize the run node with the new
iterator
   ' values
   If Not mcLastIterator Is Nothing Then
      If cItRec Is Nothing Then
         mcLastIterator.Value =
gstrEmptyString
      Else
         mcLastIterator.Value = cItRec.Value

         ' And if the iterator is a list of values,
then update
         ' the sequence number as well
         If mcLastIterator.IteratorType =
gintValue Then
            mcLastIterator.Sequence =
cItRec.SequenceNo
         End If
      End If
   End If

   Set NewIteration = cItRec
   Set cItRec = Nothing

   Exit Function

NewIterationErr:
   ' Log the error code raised by Visual
Basic
   Call LogErrors(Errors)
   On Error GoTo 0
   Err.Raise vbObjectError +
errIterateFailed, mstrModuleName, _
      LoadResString(errIterateFailed)

End Function
Public Function NextIteration(cStepRec As
cStep) As cIterator

   ' Retrieves the next iterator value for the
passed in step -
   ' returns an iterator record with the new
iterator values

   Dim cItRec As cIterator
   Dim vntIterators As Variant
   Dim lngValue As String

   On Error GoTo NextIterationErr

   vntIterators = cStepRec.Iterators

   If Not mcLastIterator Is Nothing Then
      ' The run node contains the iterator details
      ' Get the next value for the iterator
      If mcLastIterator.IteratorType = gintValue
Then
         ' Find the next iterator that appears in
the list of
         ' iterator values
         Set cItRec =
NextInSequence(vntIterators,
mcLastIterator.Sequence)
      Else
         lngValue =
CLng(Trim$(mcLastIterator.Value))
         ' Determine whether the new iterator
value falls in the
         ' range between From and To
         If (mcLastIterator.RangeStep > 0 And _
               (mcLastIterator.RangeFrom <=
mcLastIterator.RangeTo) And _
               (mcLastIterator.RangeStep +
lngValue) <= mcLastIterator.RangeTo) Or _
               (mcLastIterator.RangeStep < 0
And _
               (mcLastIterator.RangeFrom >=
mcLastIterator.RangeTo) And _
               (mcLastIterator.RangeStep +
lngValue) >= mcLastIterator.RangeTo) Then
            Set cItRec = New cIterator
            cItRec.Value =
Trim$(CStr(mcLastIterator.RangeStep +
lngValue))
         Else
            Set cItRec = Nothing
         End If
      End If
   Else
      Set cItRec = Nothing
   End If

   Set NextIteration = cItRec
   Exit Function

NextIterationErr:
   ' Log the error code raised by Visual Basic
   Call LogErrors(Errors)
   On Error GoTo 0
   Err.Raise vbObjectError + errIterateFailed,
mstrModuleName, _
      LoadResString(errIterateFailed)

End Function
Public Sub InitializeIt(cPendingStep As cStep,
_
      ColParameters As cArrParameters, _
      Optional vntIterators As Variant)

   ' Initializes the LastIteration structure with
the iterator details for the
   ' passed in step

   On Error GoTo InitializeItErr

   If IsMissing(vntIterators) Then
      vntIterators = cPendingStep.Iterators
   End If

   If IsArray(vntIterators) And Not
IsEmpty(vntIterators) Then
      mcLastIterator.IteratorName =
cPendingStep.IteratorName
      If
vntIterators(LBound(vntIterators)).IteratorType = _
            gintValue Then
         mcLastIterator.IteratorType = gintValue
         ' Since the sequence numbers begin at 0
         mcLastIterator.Sequence =
gintMinIteratorSequence - 1
      Else
         mcLastIterator.IteratorType = gintFrom
         Call InitializeItRange(vntIterators,
cPendingStep.WorkspaceId, _
               ColParameters)
      End If
   Else
      Set mcLastIterator = Nothing
   End If

   Exit Sub

InitializeItErr:
   ' Log the error code raised by Visual Basic
   Call LogErrors(Errors)
   On Error GoTo 0
   Err.Raise vbObjectError + errIterateFailed,
mstrModuleName, _
      LoadResString(errIterateFailed)

End Sub

Private Sub InitializeItRange(vntIterators As
Variant, ByVal lWorkspace As Long, _
      ColParameters As cArrParameters)

   ' Initializes the LastIteration structure for range
iterators from the
   ' passed in variant containing the iterator records

   Dim lngIndex As Long
   Dim cItRec As cIterator

   On Error GoTo InitializeItRangeErr

   If IsArray(vntIterators) And Not
IsEmpty(vntIterators) Then

      ' Check if the iterator range has been
completely initialized
      RangeComplete (vntIterators)

      ' Initialize the Run node with the values for the
From,
      ' To and Step boundaries
      For lngIndex = LBound(vntIterators) To
UBound(vntIterators)
         Set cItRec = vntIterators(lngIndex)
         Select Case cItRec.IteratorType
            Case gintFrom
               mcLastIterator.RangeFrom =
SubstituteParameters(cItRec.Value, lWorkspace,
WspParameters:=ColParameters)
            Case gintTo
               mcLastIterator.RangeTo =
SubstituteParameters(cItRec.Value, lWorkspace,
WspParameters:=ColParameters)
            Case gintStep
               mcLastIterator.RangeStep =
SubstituteParameters(cItRec.Value, lWorkspace,
WspParameters:=ColParameters)
            Case Else
               On Error GoTo 0
               Err.Raise vbObjectError +
errTypeInvalid, mstrModuleName, _
                  LoadResString(errTypeInvalid)
         End Select
      Next lngIndex
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
      mcLastIterator.Value =
Trim$(CStr(mcLastIterator.RangeFrom -
mcLastIterator.RangeStep))
   End If

   Exit Sub

InitializeItRangeErr:
   ' Log the error code raised by Visual
Basic
   Call LogErrors(Errors)
   On Error GoTo 0
   Err.Raise vbObjectError +
errIterateFailed, mstrModuleName, _
      LoadResString(errIterateFailed)

End Sub
Private Function
NextInSequence(vntIterators As Variant, _
      lngOldSequence As Long) As cIterator

   Dim lngIndex As Long
   Dim cItRec As cIterator

   On Error GoTo NextInSequenceErr

   If IsArray(vntIterators) And Not
IsEmpty(vntIterators) Then
      For lngIndex = LBound(vntIterators)
To UBound(vntIterators)
         Set cItRec = vntIterators(lngIndex)
         If cItRec.IteratorType <> gintValue
Then
            On Error GoTo 0
            Err.Raise vbObjectError +
errTypeInvalid, mstrModuleName, _

LoadResString(errTypeInvalid)
         End If
         If cItRec.SequenceNo =
lngOldSequence + 1 Then
            Exit For
         End If

      Next lngIndex

      If cItRec.SequenceNo <>
lngOldSequence + 1 Then
         Set cItRec = Nothing
      End If
   Else
      Set cItRec = Nothing
   End If

   Set NextInSequence = cItRec

   Exit Function

NextInSequenceErr:
   ' Log the error code raised by Visual
Basic
   Call LogErrors(Errors)
   On Error GoTo 0
   Err.Raise vbObjectError +
errIterateFailed, mstrModuleName, _
      LoadResString(errIterateFailed)

End Function

Public Property Get LastIterator() As
cRunItDetails

   Set LastIterator = mcLastIterator
End Property
Public Property Set LastIterator(vdata As
cRunItDetails)

   Set mcLastIterator = vdata

End Property

Public Property Get TasksRunning() As
Integer

   TasksRunning = mintRunning

End Property

Public Property Let TasksRunning(ByVal
vdata As Integer)

   mintRunning = vdata

End Property

Public Property Get TasksComplete() As
Integer

   TasksComplete = mintComplete

End Property
Public Property Let TasksComplete(ByVal
vdata As Integer)

   mintComplete = vdata

End Property
Public Property Get StepId() As Long

   StepId = mlngStepId

End Property
Public Property Let StepId(ByVal vdata As
Long)

   mlngStepId = vdata

End Property
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cSubSteps"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:       cSubSteps.cls
'             Microsoft TPC-H Kit Ver. 1.00
'             Copyright Microsoft, 1999
'             All Rights Reserved
'
'
'  PURPOSE:    This module provides a type-
safe wrapper around cVector to
'             implement a collection of cSubStep
objects.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

Private mcSubSteps As cVector

Public Sub Add(ByVal objItem As cSubStep)

   mcSubSteps.Add objItem
End Sub


Public Sub Clear()

   mcSubSteps.Clear

End Sub


Public Function Count() As Long

   Count = mcSubSteps.Count

End Function


Public Function Delete(ByVal lngDelete As Long)
As cSubStep

   Set Delete = mcSubSteps.Delete(lngDelete)

End Function


Public Property Get Item(ByVal Position As Long)
As cSubStep
Attribute Item.VB_UserMemId = 0

   Set Item = mcSubSteps.Item(Position)

End Property

Private Sub Class_Initialize()

   Set mcSubSteps = New cVector

End Sub

Private Sub Class_Terminate()

   Set mcSubSteps = Nothing

End Sub


VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cTermProcess"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:       cTermProcess.cls
'             Microsoft TPC-H Kit Ver. 1.00
'             Copyright Microsoft, 1999
'             All Rights Reserved
'
'
'  PURPOSE:    This module raises an event if a
completed step exists.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

Private WithEvents moTimer As cTimerSM
Attribute moTimer.VB_VarHelpID = -1
Private bTermProcessExists As Boolean
Public Event TermProcessExists()

Public Sub ProcessTerminated()
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   bTermProcessExists = True
   moTimer.Enabled = True

End Sub

Private Sub Class_Initialize()

   bTermProcessExists = False

   Set moTimer = New cTimerSM
   moTimer.Enabled = False

End Sub

Private Sub Class_Terminate()

   Set moTimer = Nothing

End Sub

Private Sub moTimer_Timer()

   On Error GoTo moTimer_TimerErr

   If bTermProcessExists Then
      RaiseEvent TermProcessExists
   End If

   moTimer.Enabled = False
   bTermProcessExists = False

   Exit Sub

moTimer_TimerErr:
   LogErrors Errors

End Sub

VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cTermStep"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:       cTermStep.cls
'             Microsoft TPC-H Kit Ver. 1.00
'             Copyright Microsoft, 1999
'             All Rights Reserved
'
'
'  PURPOSE:    This module encapsulates
the properties of steps that
'             have completed execution such as
status and time of completion.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

Public TimeComplete As Currency
Public Index As Long
Public InstanceId As Long
Public ExecutionStatus As InstanceStatus

VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cTermSteps"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:       cTermSteps.cls
'             Microsoft TPC-H Kit Ver. 1.00
'             Copyright Microsoft, 1999
'             All Rights Reserved
'
'
'  PURPOSE:    This module provides a type-
safe wrapper around cVector to
'             implement a collection of cTermStep
objects. Raises an
'             event if a step that has completed
execution exists.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

Private mcTermSteps As cVector
Private WithEvents moTimer As cTimerSM
Attribute moTimer.VB_VarHelpID = -1
Public Event TermStepExists(cStepDetails As
cTermStep)

Public Sub Add(ByVal citem As cTermStep)

   Call mcTermSteps.Add(citem)
   moTimer.Enabled = True

End Sub

Public Sub Clear()

   mcTermSteps.Clear

End Sub

Public Function Delete()

   Call mcTermSteps.Delete(0)
   ' Disable the timer if there are no more
pending events
   If mcTermSteps.Count = 0 Then
moTimer.Enabled = False

End Function

Public Property Get Item(ByVal Position As
Long) As cTermStep

   Set Item = mcTermSteps(Position)

End Property

Public Function Count() As Long

   Count = mcTermSteps.Count

End Function

Private Sub Class_Initialize()

   Set mcTermSteps = New cVector

   Set moTimer = New cTimerSM
   moTimer.Enabled = False

End Sub

Private Sub Class_Terminate()

   Set mcTermSteps = Nothing
   Set moTimer = Nothing

End Sub

Private Sub moTimer_Timer()

   On Error GoTo moTimer_TimerErr

   If mcTermSteps.Count > 0 Then
      ' Since items are appended to the end of the
array
      RaiseEvent TermStepExists(mcTermSteps(0))
   Else
      moTimer.Enabled = False
   End If
   Exit Sub

moTimer_TimerErr:
   LogErrors Errors

End Sub
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cTimerSM"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:       cTimer.cls
'             Microsoft TPC-H Kit Ver. 1.00
'             Copyright Microsoft, 1999
'             All Rights Reserved
'
'
'  PURPOSE:    This module implements a timer.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'

Option Explicit

Public Event Timer()

Private Const mnDefaultInterval As Long = 1

Private mnTimerID As Long
Private mnInterval As Long
Private mfEnabled As Boolean

Public Property Get Interval() As Long
   Interval = mnInterval
End Property
Public Property Let Interval(Value As Long)
   If mnInterval <> Value Then
      mnInterval = Value
      If mfEnabled Then
         SetInterval mnInterval, mnTimerID
      End If
   End If
End Property

Public Property Get Enabled() As Boolean
   Enabled = mfEnabled
End Property
Public Property Let Enabled(Value As Boolean)
   If mfEnabled <> Value Then
      If Value Then
         mnTimerID = StartTimer(mnInterval)
         If mnTimerID <> 0 Then
            mfEnabled = True
            'Storing Me in the global would add a
reference to Me, which
            '  would prevent Me from being released,
which in turn would
            '  prevent my Class_Terminate code from
running. To prevent
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
            '   this, I store a "soft reference" -
the collection holds a
            '   pointer to me without
incrementing my reference count.
            gcTimerObjects.Add ObjPtr(Me),
Str$(mnTimerID)
        End If
      Else
        StopTimer mnTimerID
        mfEnabled = False
        gcTimerObjects.Remove
Str$(mnTimerID)
      End If
    End If
End Property

Private Sub Class_Initialize()
    If gcTimerObjects Is Nothing Then Set
gcTimerObjects = New Collection
    mnInterval = mnDefaultInterval
End Sub

Private Sub Class_Terminate()
    Enabled = False
End Sub

Friend Sub Tick()
    RaiseEvent Timer
End Sub
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cVBErrorsSM"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
Attribute VB_Ext_KEY =
"SavedWithClassBuilder" ,"Yes"
Attribute VB_Ext_KEY = "Top_Level"
,"Yes"
'  FILE:       cVBErrors.cls
'              Microsoft TPC-H Kit Ver. 1.00
'              Copyright Microsoft, 1999
'              All Rights Reserved
'
'
'  PURPOSE:    This module encapsulates
the handling of Visual Basic errors.
'              This module does not do any error
handling - any error handler
'              will erase the errors object!
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

' The Execute class exposes a method,
WriteError through which we can write to
the
' error log that is currently being used by the
Execute object. Store a reference to
' Execute object locally.
Private mcExecObjRef As
EXECUTEDLLLib.Execute
Public Sub WriteError(ByVal ErrorCode As
errErrorConstants, _
      Optional ByVal ErrorSource As String
= gstrEmptyString, _
      Optional ByVal OptArgs As String =
gstrEmptyString)

    Dim sError As String

    sError = "StepMaster Error:" & ErrorCode
& vbCrLf & LoadResString(ErrorCode) &
vbCrLf

    If Not StringEmpty(ErrorSource) Then
        sError = sError & "(Source: " &
ErrorSource & ")" & vbCrLf
    End If
    sError = sError & OptArgs

    Call LogMessage(sError)

End Sub
Private Function InitErrorString() As String
    ' Initializes a string with all the properties of
the
    ' Err object

    Dim strError As String
    Dim errCode As Long

    If Err.Number = 0 Then
        InitErrorString = gstrEmptyString
    Else
    With Err
        If Err.Number > vbObjectError And
Err.Number < (vbObjectError + 65536) Then
            errCode = .Number - vbObjectError
        Else
            errCode = .Number
        End If
        strError = "Error #: " & errCode &
vbCrLf
        strError = strError & "Description: " &
.Description & vbCrLf
        strError = strError & "Source: " &
Err.Source & vbCrLf
    End With

    Debug.Print strError
    InitErrorString = strError
    End If

End Function
Public Sub LogVBErrors()

    Dim strErr As String

    strErr = InitErrorString

    On Error GoTo LogVBErrorsErr

    If Not StringEmpty(strErr) Then
        ' Write an error using the WriteError
method of the Execute object.
        If Not mcExecObjRef Is Nothing Then
            mcExecObjRef.WriteError strErr
        Else
            WriteMessage strErr
        End If
    End If

    Err.Clear

    Exit Sub

LogVBErrorsErr:
    Call LogErrors(Errors)
    ' Since write to the error file for the step has
failed, write to the project log
    Call WriteMessage(strErr)

End Sub
Public Sub DisplayErrors()

    Dim strErr As String

    strErr = InitErrorString

    If Not StringEmpty(strErr) Then
        ' Display the error message
        MsgBox strErr
    End If

    Err.Clear

End Sub
Public Sub LogMessage(strMsg As String)

    On Error GoTo LogMessageErr

    ' Write an error using the WriteError method of
the Execute object.
    If Not mcExecObjRef Is Nothing Then
        mcExecObjRef.WriteError strMsg
    Else
        WriteMessage strMsg
    End If

    Exit Sub

LogMessageErr:
    Call LogErrors(Errors)
    ' Since write to the error file for the step has
failed, write to the project log
    Call WriteMessage(strMsg)

End Sub
Public Property Set ErrorFile(vdata As
EXECUTEDLLLib.Execute)

    Set mcExecObjRef = vdata

End Property
Private Sub Class_Terminate()

    Set mcExecObjRef = Nothing

End Sub
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cVector"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:       cVector.cls
'              Microsoft TPC-H Kit Ver. 1.00
'              Copyright Microsoft, 1999
'              All Rights Reserved
'
'
'  PURPOSE:    This class implements an array of
objects.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

' Used to indicate the source module name when
errors
' are raised by this class
Private mstrSource As String
Private Const mstrModuleName As String =
"cVector."

' Array counter
Private mlngCount As Long
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
Private mcarrItems() As Object

Public Sub Add(ByVal objItem As Object)
   ' Adds the passed in Object variable to the
array

   On Error GoTo AddErr

   ReDim Preserve mcarrItems(mlngCount)

   ' Set the newly added element in the array
to the
   ' passed in variable
   Set mcarrItems(mlngCount) = objItem
   mlngCount = mlngCount + 1

   Exit Sub

AddErr:
   LogErrors Errors
   gstrSource = mstrModuleName & "Add"
   On Error GoTo 0
   Err.Raise vbObjectError +
errLoadInArrayFailed, _
         mstrSource, _

LoadResString(errLoadInArrayFailed)

End Sub
Public Sub Clear()

   ' Clear the array
   ReDim mcarrItems(0)
   mlngCount = 0

End Sub

Public Function Delete(ByVal lngDelete As
Long) As Object

   Dim lngIndex As Long

   On Error GoTo DeleteErr

   If lngDelete < (mlngCount - 1) Then

      ' We want to maintain the order of all
items in the
      ' array - so move all remaining
elements in the array
      ' up by 1
      For lngIndex = lngDelete To
mlngCount - 2
         MoveDown lngIndex
      Next lngIndex

   End If

   ' Return the deleted node
   Set Delete = mcarrItems(mlngCount - 1)

   ' Delete the last Node from the array
   mlngCount = mlngCount - 1
   If mlngCount > 0 Then
      ReDim Preserve mcarrItems(0 To
mlngCount - 1)
   Else
      ReDim mcarrItems(0)
   End If

   Exit Function

DeleteErr:
   LogErrors Errors
   mstrSource = mstrModuleName & "Delete"
   On Error GoTo 0
   Err.Raise vbObjectError +
errDeleteArrayElementFailed, _
         mstrSource, _

LoadResString(errDeleteArrayElementFailed)

End Function
Public Property Get Item(ByVal Position As
Long) As Object
Attribute Item.VB_UserMemId = 0

   ' Returns the element at the passed in
position in the array
   If Position >= 0 And Position < mlngCount
Then
      Set Item = mcarrItems(Position)
   Else
      On Error GoTo 0
      Err.Raise vbObjectError +
errItemDoesNotExist, mstrSource, _
         LoadResString(errItemDoesNotExist)
   End If

End Property
Public Property Set Item(ByVal Position As
Long, _
      ByVal Value As Object)

   ' Returns the element at the passed in
position in the array
   If Position >= 0 Then
      ' If the passed in position is outside the
array
      ' bounds, then resize the array
      If Position >= mlngCount Then
         ReDim Preserve mcarrItems(Position)
         mlngCount = Position + 1
      End If

      ' Set the newly added element in the array
to the
      ' passed in variable
      Set mcarrItems(Position) = Value
   Else
      On Error GoTo 0
      Err.Raise vbObjectError +
errItemDoesNotExist, mstrSource, _
         LoadResString(errItemDoesNotExist)
   End If

End Property
Public Sub MoveUp(ByVal Position As Long)
   ' Moves the element at the passed in position
up by 1

   Dim cTemp As Object

   If Position > 0 And Position < mlngCount
Then
      Set cTemp = mcarrItems(Position)

      Set mcarrItems(Position) =
mcarrItems(Position - 1)
      Set mcarrItems(Position - 1) = cTemp
   End If

End Sub
Public Sub MoveDown(ByVal Position As
Long)
   ' Moves the element at the passed in position
down by 1

   Dim cTemp As Object

   If Position >= 0 And Position < mlngCount - 1
Then
      Set cTemp = mcarrItems(Position)

      Set mcarrItems(Position) =
mcarrItems(Position + 1)
      Set mcarrItems(Position + 1) = cTemp
   End If

End Sub

Public Function Count() As Long

   Count = mlngCount

End Function


Private Sub Class_Initialize()

   mlngCount = 0

End Sub


Private Sub Class_Terminate()

   Call Clear

End Sub

VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cVectorLng"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:       cVectorLng.cls
'              Microsoft TPC-H Kit Ver. 1.00
'              Copyright Microsoft, 1999
'              All Rights Reserved
'
'
'  PURPOSE:    This class implements an array of
longs.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

' Used to indicate the source module name when
errors
' are raised by this class
Private mstrSource As String
Private Const mstrModuleName As String =
"cVectorLng."

' Array counter
Private mlngCount As Long
Private mcarrItems() As Long

Public Sub Add(ByVal lngItem As Long)
   ' Adds the passed in long variable to the array

   On Error GoTo AddErr

   ReDim Preserve mcarrItems(mlngCount)

   ' Set the newly added element in the array to the
   ' passed in variable
   mcarrItems(mlngCount) = lngItem
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   mlngCount = mlngCount + 1

   Exit Sub

AddErr:
   LogErrors Errors
   gstrSource = mstrModuleName & "Add"
   On Error GoTo 0
   Err.Raise vbObjectError +
errLoadInArrayFailed, _
         mstrSource, _

LoadResString(errLoadInArrayFailed)

End Sub
Public Sub Clear()

   ' Clear the array
   ReDim mcarrItems(0)

End Sub

Public Sub Delete(Optional ByVal Position
As Long = -1, _
      Optional ByVal Item As Long = -1)
   ' The user can opt to delete either a
specific item in
   ' the list or the item at a specified position.
If no
   ' parameters are passed in, we delete the
element at
   ' position 0!

   Dim lngDelete As Long
   Dim lngIndex As Long

   On Error GoTo DeleteErr

   If Position = -1 Then
      ' Since we can never store an element at
position -1,
      ' we can be sure that the user is trying
to delete
      ' a given item
      lngDelete = Find(Item)
   Else
      lngDelete = Position
   End If

   If lngDelete < (mlngCount - 1) Then

      ' We want to maintain the order of all
items in the
      ' array - so move all remaining
elements in the array
      ' up by 1
      For lngIndex = lngDelete To
mlngCount - 2
         MoveDown lngIndex
      Next lngIndex

   End If

   ' Delete the last Node from the array
   mlngCount = mlngCount - 1
   If mlngCount > 0 Then
      ReDim Preserve mcarrItems(0 To
mlngCount - 1)
   Else
      ReDim mcarrItems(0)
   End If

   Exit Sub

DeleteErr:
   LogErrors Errors
   mstrSource = mstrModuleName & "Delete"
   On Error GoTo 0
   Err.Raise vbObjectError +
errDeleteArrayElementFailed, _
         mstrSource, _

LoadResString(errDeleteArrayElementFailed)

End Sub
Public Function Find(ByVal Item As Long) As
Long

   ' Returns the position at which the passed in
value occurs
   ' in the array

   Dim lngIndex As Long

   On Error GoTo FindErr

   ' Find the element in the array to be deleted
   For lngIndex = 0 To mlngCount - 1

      If mcarrItems(lngIndex) = Item Then
         Find = lngIndex
         Exit Function
      End If

   Next lngIndex

   Find = -1

   Exit Function

FindErr:
   LogErrors Errors
   mstrSource = mstrModuleName & "Find"
   On Error GoTo 0
   Err.Raise vbObjectError +
errItemNotFound, mstrSource, _
      LoadResString(errItemNotFound)

End Function
Public Property Get Item(ByVal Position As
Long) As Long
Attribute Item.VB_UserMemId = 0

   ' Returns the element at the passed in
position in the array
   If Position >= 0 And Position < mlngCount
Then
      Item = mcarrItems(Position)
   Else
      On Error GoTo 0
      Err.Raise vbObjectError +
errItemDoesNotExist, mstrSource, _
         LoadResString(errItemDoesNotExist)
   End If

End Property
Public Property Let Item(ByVal Position As
Long, _
      ByVal Value As Long)

   ' Returns the element at the passed in
position in the array
   If Position >= 0 Then
      ' If the passed in position is outside the
array
      ' bounds, then resize the array
      If Position >= mlngCount Then
         ReDim Preserve mcarrItems(Position)
         mlngCount = Position + 1
      End If

      ' Set the newly added element in the array to
the
      ' passed in variable
      mcarrItems(Position) = Value
   Else
      On Error GoTo 0
      Err.Raise vbObjectError +
errItemDoesNotExist, mstrSource, _
         LoadResString(errItemDoesNotExist)
   End If

End Property
Public Sub MoveUp(ByVal Position As Long)
   ' Moves the element at the passed in position up
by 1

   Dim lngTemp As Long

   If Position > 0 And Position < mlngCount Then
      lngTemp = mcarrItems(Position)

      mcarrItems(Position) = mcarrItems(Position -
1)
      mcarrItems(Position - 1) = lngTemp
   End If

End Sub
Public Sub MoveDown(ByVal Position As Long)
   ' Moves the element at the passed in position
down by 1

   Dim lngTemp As Long

   If Position >= 0 And Position < mlngCount - 1
Then
      lngTemp = mcarrItems(Position)

      mcarrItems(Position) = mcarrItems(Position +
1)
      mcarrItems(Position + 1) = lngTemp
   End If

End Sub

Public Function Count() As Long

   Count = mlngCount

End Function


Private Sub Class_Initialize()

   mlngCount = 0

End Sub


Private Sub Class_Terminate()

   Call Clear

End Sub

VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cVectorStr"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:       cVectorStr.cls
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
'           Microsoft TPC-H Kit Ver. 1.00
'           Copyright Microsoft, 1999
'           All Rights Reserved
'
'
'  PURPOSE:    This class implements an
array of strings.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

' Used to indicate the source module name
when errors
' are raised by this class
Private mstrSource As String
Private Const mstrModuleName As String =
"cVectorStr."

' Array counter
Private mlngCount As Long
Private mcarrItems() As String

Public Sub Add(ByVal strItem As String)
   ' Adds the passed in string variable to the
array

   On Error GoTo AddErr

   ReDim Preserve mcarrItems(mlngCount)

   ' Set the newly added element in the array
to the
   ' passed in variable
   mcarrItems(mlngCount) = strItem
   mlngCount = mlngCount + 1

   Exit Sub

AddErr:
   Call LogErrors(Errors)
   gstrSource = mstrModuleName & "Add"
   On Error GoTo 0
   Err.Raise vbObjectError +
errLoadInArrayFailed, _
         mstrSource, _

LoadResString(errLoadInArrayFailed)

End Sub
Public Sub Clear()

   ' Clear the array
   ReDim mcarrItems(0)

End Sub

Public Sub Delete(Optional ByVal Position
As Long = -1, _
      Optional ByVal Item As String = -1)
   ' The user can opt to delete either a
specific item in
   ' the list or the item at a specified position.
If no
   ' parameters are passed in, we delete the
element at
   ' position 0!

   Dim lngDelete As Long
   Dim lngIndex As Long

   On Error GoTo DeleteErr
   mstrSource = mstrModuleName &
"Delete"
   If Position = -1 Then
      ' Since we can never store an element at
position -1,
      ' we can be sure that the user is trying to
delete
      ' a given item
      lngDelete = Find(Item)
   Else
      lngDelete = Position
   End If

   If lngDelete < (mlngCount - 1) Then

      ' We want to maintain the order of all
items in the
      ' array - so move all remaining elements in
the array
      ' up by 1
      For lngIndex = lngDelete To mlngCount -
2
         MoveDown lngIndex
      Next lngIndex

   End If

   ' Delete the last Node from the array
   mlngCount = mlngCount - 1
   If mlngCount > 0 Then
      ReDim Preserve mcarrItems(0 To
mlngCount - 1)
   Else
      ReDim mcarrItems(0)
   End If

   Exit Sub

DeleteErr:
   Call LogErrors(Errors)
   mstrSource = mstrModuleName & "Delete"
   On Error GoTo 0
   Err.Raise vbObjectError +
errDeleteArrayElementFailed, _
         mstrSource, _

LoadResString(errDeleteArrayElementFailed)

End Sub
Public Function Find(ByVal Item As String)
As Long

   ' Returns the position at which the passed in
value occurs
   ' in the array

   Dim lngIndex As Long

   On Error GoTo FindErr
   mstrSource = mstrModuleName & "Find"

   ' Find the element in the array to be deleted
   For lngIndex = 0 To mlngCount - 1

      If mcarrItems(lngIndex) = Item Then
         Find = lngIndex
         Exit Function
      End If

   Next lngIndex

   Find = -1

   Exit Function

FindErr:
   Call LogErrors(Errors)
   mstrSource = mstrModuleName & "Find"
   On Error GoTo 0
   Err.Raise vbObjectError + errItemNotFound,
mstrSource, _
      LoadResString(errItemNotFound)

End Function
Public Property Get Item(ByVal Position As Long)
As String
Attribute Item.VB_UserMemId = 0

   ' Returns the element at the passed in position in
the array
   If Position >= 0 And Position < mlngCount Then
      Item = mcarrItems(Position)
   Else
      On Error GoTo 0
      Err.Raise vbObjectError +
errItemDoesNotExist, mstrSource, _
         LoadResString(errItemDoesNotExist)
   End If

End Property
Public Property Let Item(ByVal Position As Long,
_
      ByVal Value As String)

   ' Returns the element at the passed in position in
the array
   If Position >= 0 Then
      ' If the passed in position is outside the array
      ' bounds, then resize the array
      If Position >= mlngCount Then
         ReDim Preserve mcarrItems(Position)
         mlngCount = Position + 1
      End If

      ' Set the newly added element in the array to
the
      ' passed in variable
      mcarrItems(Position) = Value
   Else
      On Error GoTo 0
      Err.Raise vbObjectError +
errItemDoesNotExist, mstrSource, _
         LoadResString(errItemDoesNotExist)
   End If

End Property
Public Sub MoveUp(ByVal Position As Long)
   ' Moves the element at the passed in position up
by 1

   Dim strTemp As String

   If Position > 0 And Position < mlngCount Then
      strTemp = mcarrItems(Position)

      mcarrItems(Position) = mcarrItems(Position -
1)
      mcarrItems(Position - 1) = strTemp
   End If

End Sub
Public Sub MoveDown(ByVal Position As Long)
   ' Moves the element at the passed in position
down by 1

   Dim strTemp As String

   If Position >= 0 And Position < mlngCount - 1
Then
      strTemp = mcarrItems(Position)
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
      mcarrItems(Position) =
mcarrItems(Position + 1)
      mcarrItems(Position + 1) = strTemp
   End If

End Sub

Public Function Count() As Long

   Count = mlngCount

End Function


Private Sub Class_Initialize()

   mlngCount = 0

End Sub
Private Sub Class_Terminate()

   Call Clear

End Sub


VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cWorker"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:       cWorker.cls
'              Microsoft TPC-H Kit Ver. 1.00
'              Copyright Microsoft, 1999
'              All Rights Reserved
'
'
'  PURPOSE:    Encapsulates the properties
and methods of a worker step.
'              Implements the cStep class -
carries out initializations
'              and validations that are specific to
worker steps.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

Implements cStep

' Object variable to keep the step reference
in
Private mcStep As cStep

' Used to indicate the source module name
when errors
' are raised by this class
Private mstrSource As String
Private Const mstrModuleName As String =
"cWorker."
Private Sub cStep_AddAllIterators()

   Call mcStep.AddAllIterators

End Sub

Private Property Let cStep_StartDir(ByVal
RHS As String)

   mcStep.StartDir = RHS

End Property

Private Property Get cStep_StartDir() As
String

   cStep_StartDir = mcStep.StartDir

End Property

Private Property Set cStep_NodeDB(RHS As
DAO.Database)

   Set mcStep.NodeDB = RHS

End Property

Private Property Get cStep_NodeDB() As
DAO.Database

   Set cStep_NodeDB = mcStep.NodeDB

End Property

Private Function cStep_IncVersionY() As
String

   cStep_IncVersionY = mcStep.IncVersionY

End Function
Private Function cStep_IsNewVersion() As
Boolean
   cStep_IsNewVersion =
mcStep.IsNewVersion
End Function
Private Function cStep_OldVersionNo() As
String
   cStep_OldVersionNo =
mcStep.OldVersionNo
End Function

Private Function cStep_IncVersionX() As
String

   cStep_IncVersionX = mcStep.IncVersionX

End Function
Private Sub cStep_UpdateIteratorVersion()

   Call mcStep.UpdateIteratorVersion

End Sub

Private Function cStep_IteratorCount() As
Long

   cStep_IteratorCount = mcStep.IteratorCount

End Function

Private Sub cStep_UnloadIterators()

   Call mcStep.UnloadIterators

End Sub

Private Sub cStep_SaveIterators()

   Call mcStep.SaveIterators

End Sub
Private Property Get cStep_IteratorName() As
String

   cStep_IteratorName = mcStep.IteratorName

End Property
Private Property Let cStep_IteratorName(ByVal
RHS As String)

   mcStep.IteratorName = RHS

End Property

Private Sub cStep_LoadIterator(cItRecord As
cIterator)

   Call mcStep.LoadIterator(cItRecord)

End Sub
Private Sub cStep_DeleteIterator(cItRecord As
cIterator)

   Call mcStep.DeleteIterator(cItRecord)

End Sub

Private Sub cStep_InsertIterator(cItRecord As
cIterator)

   Call mcStep.InsertIterator(cItRecord)

End Sub
Private Function cStep_Iterators() As Variant

   cStep_Iterators = mcStep.Iterators

End Function
Private Sub cStep_ModifyIterator(cItRecord As
cIterator)

   Call mcStep.ModifyIterator(cItRecord)

End Sub
Private Sub cStep_RemoveIterator(cItRecord As
cIterator)

   Call mcStep.RemoveIterator(cItRecord)

End Sub
Private Sub cStep_UpdateIterator(cItRecord As
cIterator)

   Call mcStep.UpdateIterator(cItRecord)

End Sub
Private Sub cStep_AddIterator(cItRecord As
cIterator)

   Call mcStep.AddIterator(cItRecord)

End Sub

Private Property Let cStep_Position(ByVal RHS As
Long)

   mcStep.Position = RHS

End Property

Private Property Get cStep_Position() As Long

   cStep_Position = mcStep.Position

End Property

Private Function cStep_Clone(Optional cCloneStep
As cStep) As cStep

   Dim cNewWorker As cWorker
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   Set cNewWorker = New cWorker
   Set cStep_Clone =
mcStep.Clone(cNewWorker)

End Function

Private Sub StepTextOrFileEntered()
   ' Checks if either the step text or the name
of the file containing
   ' the text has been entered
   ' If both of them are null or both of them
are not null,
   ' the worker step is invalid and an error is
raised
   If StringEmpty(mcStep.StepText) And
StringEmpty(mcStep.StepTextFile) Then
      ShowError errStepTextAndFileNull
      On Error GoTo 0
      Err.Raise vbObjectError +
errStepTextAndFileNull, _
         mstrSource,
LoadResString(errStepTextAndFileNull)
   End If

End Sub

Private Property Get cStep_IndOperation()
As Operation

   cStep_IndOperation =
mcStep.IndOperation

End Property

Private Property Let
cStep_IndOperation(ByVal RHS As
Operation)

   mcStep.IndOperation = RHS

End Property

Private Property Get cStep_NextStepId() As
Long

   cStep_NextStepId = mcStep.NextStepId

End Property

Private Property Let
cStep_OutputFile(ByVal RHS As String)

   mcStep.OutputFile = RHS

End Property

Private Property Get cStep_OutputFile() As
String

   cStep_OutputFile = mcStep.OutputFile

End Property

Private Property Let cStep_ErrorFile(ByVal
RHS As String)

   mcStep.ErrorFile = RHS

End Property

Private Property Get cStep_ErrorFile() As
String

   cStep_ErrorFile = mcStep.ErrorFile

End Property
'Private Property Let cStep_LogFile(ByVal
RHS As String)
'
'   mcStep.LogFile = RHS
'
'End Property
'
'Private Property Get cStep_LogFile() As
String
'
'   cStep_LogFile = mcStep.LogFile
'
'End Property

Private Property Let
cStep_ArchivedFlag(ByVal RHS As Boolean)

   mcStep.ArchivedFlag = RHS

End Property

Private Property Get cStep_ArchivedFlag() As
Boolean

   cStep_ArchivedFlag = mcStep.ArchivedFlag

End Property

Private Sub Class_Initialize()

   ' Create the object
   Set mcStep = New cStep

   ' Initialize the object with valid values for a
Worker step
   ' The global flag should be the first field to
be initialized
   ' since subsequent validations might try to
check if the
   ' step being created is global
   mcStep.GlobalFlag = False
'   mcStep.GlobalRunMethod = gintNoOption
   mcStep.StepType = gintWorkerStep

End Sub
Private Sub Class_Terminate()

   ' Remove the step object
   Set mcStep = Nothing

End Sub
Private Sub cStep_Add()

   ' Call a private procedure to see if the step
text has been
   ' entered - since a worker step actually
executes a step, entry
   ' of the text is mandatory
   Call StepTextOrFileEntered

   ' Call the Add method of the step class to
carry out the insert
   mcStep.Add

End Sub

Private Property Get
cStep_ContinuationCriteria() As
ContinuationCriteria

   cStep_ContinuationCriteria =
mcStep.ContinuationCriteria

End Property

Private Property Let
cStep_ContinuationCriteria(ByVal RHS As
ContinuationCriteria)

   ' The Continuation criteria must be non-null for
all worker steps.
   ' Check if the Continuation Criteria is valid
   Select Case RHS
      Case gintOnFailureAbortSiblings,
gintOnFailureCompleteSiblings, _
            gintOnFailureSkipSiblings,
gintOnFailureAbort, _
            gintOnFailureContinue,
gintOnFailureAsk
         mcStep.ContinuationCriteria = RHS

      Case Else
         On Error GoTo 0
         Err.Raise vbObjectError +
errContCriteriaInvalid, _
               mstrModuleName,
LoadResString(errContCriteriaInvalid)
   End Select

End Property

Private Property Let
cStep_DegreeParallelism(ByVal RHS As String)

   mcStep.DegreeParallelism = RHS

End Property

Private Property Get cStep_DegreeParallelism() As
String

   cStep_DegreeParallelism =
mcStep.DegreeParallelism

End Property

Private Sub cStep_Delete()

   mcStep.Delete

End Sub

Private Property Get cStep_EnabledFlag() As
Boolean

   cStep_EnabledFlag = mcStep.EnabledFlag

End Property

Private Property Let cStep_EnabledFlag(ByVal
RHS As Boolean)

   mcStep.EnabledFlag = RHS

End Property

Private Property Let
cStep_ExecutionMechanism(ByVal RHS As
ExecutionMethod)

   On Error GoTo ExecutionMechanismErr
   mstrSource = mstrModuleName &
"cStep_ExecutionMechanism"

   Select Case RHS
      Case gintExecuteShell, gintExecuteODBC
         mcStep.ExecutionMechanism = RHS

      Case Else
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
        On Error GoTo 0
        Err.Raise vbObjectError +
errExecutionMechanismInvalid, _
                mstrSource,
LoadResString(errExecutionMechanismInva
lid)
    End Select

    Exit Property

ExecutionMechanismErr:
    LogErrors Errors
    mstrSource = mstrModuleName &
"cStep_ExecutionMechanism"
    On Error GoTo 0
    Err.Raise vbObjectError +
errExecutionMechanismLetFailed, _
        mstrSource,
LoadResString(errExecutionMechanismLet
Failed)

End Property

Private Property Get
cStep_ExecutionMechanism() As
ExecutionMethod

    cStep_ExecutionMechanism =
mcStep.ExecutionMechanism

End Property

Private Property Let
cStep_FailureDetails(ByVal RHS As String)

    mcStep.FailureDetails = RHS

End Property

Private Property Get cStep_FailureDetails()
As String

    cStep_FailureDetails =
mcStep.FailureDetails

End Property

Private Property Get cStep_GlobalFlag() As
Boolean

    cStep_GlobalFlag = mcStep.GlobalFlag

End Property

Private Property Let
cStep_GlobalFlag(ByVal RHS As Boolean)

    ' Set the global flag to false - this flag is
initialized when
    ' an instance of the class is created. Just
making sure that
    ' nobody changes the value inadvertently
    mcStep.GlobalFlag = False

End Property
Private Sub cStep_Modify()

    ' Call a private procedure to see if the step
text has been
    ' entered - since a worker step actually
executes a step, entry
    ' of the text is mandatory
    Call StepTextOrFileEntered

    ' Call the Modify method of the step class to
carry out the update
    mcStep.Modify

End Sub

Private Property Let
cStep_ParentStepId(ByVal RHS As Long)

    mcStep.ParentStepId = RHS

End Property

Private Property Get cStep_ParentStepId() As
Long

    cStep_ParentStepId = mcStep.ParentStepId

End Property

Private Property Let
cStep_ParentVersionNo(ByVal RHS As
String)

    mcStep.ParentVersionNo = RHS

End Property

Private Property Get cStep_ParentVersionNo()
As String

    cStep_ParentVersionNo =
mcStep.ParentVersionNo

End Property

Private Property Let
cStep_SequenceNo(ByVal RHS As Integer)

    mcStep.SequenceNo = RHS

End Property

Private Property Get cStep_SequenceNo() As
Integer

    cStep_SequenceNo = mcStep.SequenceNo

End Property

Private Property Let cStep_StepId(ByVal RHS
As Long)

    mcStep.StepId = RHS

End Property

Private Property Get cStep_StepId() As Long

    cStep_StepId = mcStep.StepId

End Property


Private Property Let cStep_StepLabel(ByVal
RHS As String)

    mcStep.StepLabel = RHS

End Property

Private Property Get cStep_StepLabel() As
String

    cStep_StepLabel = mcStep.StepLabel

End Property


Private Property Let cStep_StepLevel(ByVal RHS
As Integer)

    mcStep.StepLevel = RHS

End Property

Private Property Get cStep_StepLevel() As Integer

    cStep_StepLevel = mcStep.StepLevel

End Property

Private Property Let cStep_StepText(ByVal RHS
As String)

    mcStep.StepText = RHS

End Property

Private Property Get cStep_StepText() As String

    cStep_StepText = mcStep.StepText

End Property

Private Property Let cStep_StepTextFile(ByVal
RHS As String)

    mcStep.StepTextFile = RHS

End Property

Private Property Get cStep_StepTextFile() As
String

    cStep_StepTextFile = mcStep.StepTextFile

End Property

Private Property Let cStep_StepType(RHS As
gintStepType)

    mcStep.StepType = gintWorkerStep

End Property

Private Property Get cStep_StepType() As
gintStepType

    cStep_StepType = mcStep.StepType

End Property

Private Sub cStep_Validate()
    ' The validate routines for each of the steps will
    ' carry out the specific validations for the type
and
    ' call the generic validation routine

    On Error GoTo cStep_ValidateErr

    ' Validations specific to worker steps

    ' Check if the step text or a file name has been
    ' specified
    Call StepTextOrFileEntered

    mcStep.Validate

    Exit Sub
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
cStep_ValidateErr:
   LogErrors Errors
   mstrSource = mstrModuleName &
"cStep_Validate"
   On Error GoTo 0
   Err.Raise vbObjectError +
errValidateFailed, _
         mstrSource, _
         LoadResString(errValidateFailed)
End Sub

Private Property Let
cStep_VersionNo(ByVal RHS As String)

   mcStep.VersionNo = RHS

End Property

Private Property Get cStep_VersionNo() As
String

   cStep_VersionNo = mcStep.VersionNo

End Property

Private Property Let
cStep_WorkspaceId(ByVal RHS As Long)

   mcStep.WorkspaceId = RHS

End Property

Private Property Get cStep_WorkspaceId()
As Long

   cStep_WorkspaceId =
mcStep.WorkspaceId

End Property
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cWorkspace"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:       cWorkspace.cls
'             Microsoft TPC-H Kit Ver. 1.00
'             Copyright Microsoft, 1999
'             All Rights Reserved
'
'
'  PURPOSE:    Encapsulates the properties
and methods of a workspace.
'             Contains functions to insert,
update and delete
'             att_workspaces records from the
database.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

' Local variable(s) to hold property value(s)
Private mlngWorkspaceId As Long
Private mstrWorkspaceName As String
Private mblnArchivedFlag As Boolean
Private mdbsStepMaster As Database

' Used to indicate the source module name
when errors
' are raised by this class
Private mstrSource As String
Private Const mstrModuleName As String =
"cWorkspace."

' The cSequence class is used to generate
unique workspace identifiers
Private mWorkspaceSeq As cSequence

' The StringSM class is used to carry out string
operations
Private mFieldValue As cStringSM

Public Function Clone() As cWorkspace

   ' Creates a copy of a given workspace

   Dim cCloneWsp As cWorkspace

   On Error GoTo CloneErr

   Set cCloneWsp = New cWorkspace

   ' Copy all the workspace properties to the
newly
   ' created workspace
   cCloneWsp.WorkspaceId =
mlngWorkspaceId
   cCloneWsp.WorkspaceName =
mstrWorkspaceName
   cCloneWsp.ArchivedFlag =
mblnArchivedFlag

   ' And set the return value to the newly
created workspace
   Set Clone = cCloneWsp

   Exit Function

CloneErr:
   LogErrors Errors
   mstrSource = mstrModuleName & "Clone"
   On Error GoTo 0
   Err.Raise vbObjectError + errCloneFailed, _
         mstrSource,
LoadResString(errCloneFailed)

End Function

Public Property Let ArchivedFlag(ByVal vdata
As Boolean)

   mblnArchivedFlag = vdata

End Property

Public Property Get ArchivedFlag() As
Boolean

   ArchivedFlag = mblnArchivedFlag

End Property

Public Property Set WorkDatabase(vdata As
Database)

   Set mdbsStepMaster = vdata

End Property

Private Sub WorkspaceNameDuplicate()
   ' Check if the workspace name already exists
in the workspace

   Dim rstWorkspace As Recordset
   Dim strSql As String
   Dim qy As DAO.QueryDef

   On Error GoTo WorkspaceNameDuplicateErr
   mstrSource = mstrModuleName &
"WorkspaceNameDuplicate"

   ' Create a recordset to retrieve the count of
records
   ' having the same workspace name
   strSql = " Select count(*) as workspace_count "
& _
      " from att_workspaces " & _
      " where workspace_name = [w_name] " & _
      " and workspace_id <> [w_id] "
   Set qy =
mdbsStepMaster.CreateQueryDef(gstrEmptyString,
strSql)

   ' Call a procedure to assign the parameter values
   Call AssignParameters(qy)

   Set rstWorkspace =
qy.OpenRecordset(dbOpenForwardOnly)

'      mFieldValue.MakeStringFieldValid
(mstrWorkspaceName) & _
'       " and workspace_id <> " & _
'       Str(mlngWorkspaceId)
'
'   Set rstWorkspace =
mdbsStepMaster.OpenRecordset( _
'       strSQL, dbOpenForwardOnly)

   If rstWorkspace![workspace_count] > 0 Then
      rstWorkspace.Close
      qy.Close
      ShowError errDuplicateWorkspaceName
      On Error GoTo 0
      Err.Raise vbObjectError +
errDuplicateWorkspaceName, _
         mstrSource,
LoadResString(errDuplicateWorkspaceName)
   End If
   rstWorkspace.Close
   qy.Close

   Exit Sub

WorkspaceNameDuplicateErr:
   Call LogErrors(Errors)
   mstrSource = mstrModuleName &
"WorkspaceNameDuplicate"
   On Error GoTo 0
   Err.Raise vbObjectError +
errWorkspaceNameDuplicateFailed, _
      mstrSource,
LoadResString(errWorkspaceNameDuplicateFailed
)

End Sub
Public Property Let WorkspaceName(vdata As
String)

   On Error GoTo WorkspaceNameErr
   mstrSource = mstrModuleName &
"WorkspaceName"

   If vdata = gstrEmptyString Then

      On Error GoTo 0
      ' Propogate this error back to the caller
      Err.Raise vbObjectError +
errWorkspaceNameMandatory, _
         mstrSource,
LoadResString(errWorkspaceNameMandatory)
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   Else
      mstrWorkspaceName = vdata
   End If
   Exit Property

WorkspaceNameErr:
   LogErrors Errors
   mstrSource = mstrModuleName &
"WorkspaceName"
   On Error GoTo 0
   Err.Raise vbObjectError +
errWorkspaceNameSetFailed, _
      mstrSource,
LoadResString(errWorkspaceNameSetFaile
d)

End Property

Public Property Let WorkspaceId(vdata As
Long)

   On Error GoTo WorkspaceIdErr
   mstrSource = mstrModuleName &
"WorkspaceId"

   If (vdata > 0) Then
      mlngWorkspaceId = vdata
   Else
      ' Propogate this error back to the caller
      On Error GoTo 0
      Err.Raise vbObjectError +
errWorkspaceIdInvalid, _
         mstrSource,
LoadResString(errWorkspaceIdInvalid)
   End If

   Exit Property

WorkspaceIdErr:
   LogErrors Errors
   mstrSource = mstrModuleName &
"WorkspaceId"
   On Error GoTo 0
   Err.Raise vbObjectError +
errWorkspaceIdSetFailed, _
      mstrSource,
LoadResString(errWorkspaceIdSetFailed)

End Property

Public Sub AddWorkspace()

   Dim strInsert As String
   Dim qy As DAO.QueryDef

   On Error GoTo AddWorkspaceErr

   ' Retrieve the next identifier using the
sequence class
   Set mWorkspaceSeq = New cSequence
   Set mWorkspaceSeq.IdDatabase =
mdbsStepMaster
   mWorkspaceSeq.IdentifierColumn =
FLD_ID_WORKSPACE
   mlngWorkspaceId =
mWorkspaceSeq.Identifier
   Set mWorkspaceSeq = Nothing

   ' Call procedure to raise an error if the
Workspace name
   ' already exists in the db
   Call WorkspaceNameDuplicate

   ' A new record will have the
archived_flag turned off
   mblnArchivedFlag = False

   ' Create a temporary querydef object
   strInsert = "insert into att_workspaces " & _
         "(  workspace_id, workspace_name, " &
_
         " archived_flag ) " & _
         " values ( [w_id], [w_name], [archived]
) "
   Set qy =
mdbsStepMaster.CreateQueryDef(gstrEmptySt
ring, strInsert)

   ' Call a procedure to assign the parameter
values
   Call AssignParameters(qy)

   qy.Execute dbFailOnError
   qy.Close

'    strInsert = "insert into att_workspaces " & _
'          "(  workspace_id, workspace_name, "
& _
'          " archived_flag ) " & _
'          " values ( " & _
'          Str(mlngWorkspaceId) & _
'          ", " &
mFieldValue.MakeStringFieldValid(mstrWork
spaceName) & _
'          ", " & Str(mblnArchivedFlag) & _
'          " ) "
'    mdbsStepMaster.Execute strInsert,
dbFailOnError
'
   Exit Sub

AddWorkspaceErr:

   Call LogErrors(Errors)
   mstrSource = mstrModuleName &
"AddWorkspace"
   On Error GoTo 0
   Err.Raise vbObjectError +
errWorkspaceInsertFailed, _
      mstrSource,
LoadResString(errWorkspaceInsertFailed)

End Sub
Private Sub AssignParameters(qyExec As
DAO.QueryDef)
   ' Assigns values to the parameters in the
querydef object
   ' The parameter names are cryptic to make
them different
   ' from the field names. When the parameter
names are
   ' the same as the field names, parameters in
the where
   ' clause do not get created.

   Dim prmParam As DAO.Parameter

   On Error GoTo AssignParametersErr
   mstrSource = mstrModuleName &
"AssignParameters"

   For Each prmParam In qyExec.Parameters
      Select Case prmParam.Name
         Case "[w_id]"
            prmParam.Value =
mlngWorkspaceId

         Case "[w_name]"
            prmParam.Value =
mstrWorkspaceName

         Case "[archived]"
            prmParam.Value = mblnArchivedFlag

         Case Else
            ' Write the parameter name that is faulty
            WriteError errInvalidParameter,
mstrSource, _
               prmParam.Name
            On Error GoTo 0
            Err.Raise errInvalidParameter,
mstrSource, _
               LoadResString(errInvalidParameter)
      End Select
   Next prmParam

   Exit Sub

AssignParametersErr:

   mstrSource = mstrModuleName &
"AssignParameters"
   Call LogErrors(Errors)
   On Error GoTo 0
   Err.Raise vbObjectError +
errAssignParametersFailed, _
      mstrSource,
LoadResString(errAssignParametersFailed)

End Sub
Public Sub DeleteWorkspace()

   Dim strDelete As String
   Dim qy As DAO.QueryDef

   On Error GoTo DeleteWorkspaceErr

   strDelete = "delete from att_workspaces " & _
      " where workspace_id = [w_id]"
   Set qy =
mdbsStepMaster.CreateQueryDef(gstrEmptyString,
strDelete)

   ' Call a procedure to assign the parameter values
   Call AssignParameters(qy)

   qy.Execute dbFailOnError
   qy.Close

'    mdbsStepMaster.Execute strDelete,
dbFailOnError
'       " where workspace_id = " & _
'       Str(mlngWorkspaceId)

   Exit Sub

DeleteWorkspaceErr:
   Call LogErrors(Errors)
   mstrSource = mstrModuleName &
"DeleteWorkspace"
   On Error GoTo 0
   Err.Raise vbObjectError +
errWorkspaceDeleteFailed, _
      mstrSource,
LoadResString(errWorkspaceDeleteFailed)
End Sub

Public Sub ModifyWorkspace()

   Dim strUpdate As String
   Dim qy As DAO.QueryDef

   On Error GoTo ModifyWorkspaceErr
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   ' Call procedure to raise an error if the
Workspace name
   ' already exists in the db
   Call WorkspaceNameDuplicate

   strUpdate = "update att_workspaces " & _
       " set workspace_name = [w_name] " &
_
       ", archived_flag = [archived] " & _
       " where workspace_id = [w_id] "
   Set qy =
mdbsStepMaster.CreateQueryDef(gstrEmpt
yString, strUpdate)

   ' Call a procedure to assign the parameter
values
   Call AssignParameters(qy)

   qy.Execute dbFailOnError
   qy.Close

'   strUpdate = "update att_workspaces " &
_
'       " set workspace_name = " & _
'
mFieldValue.MakeStringFieldValid(mstrW
orkspaceName) & _
'       ", archived_flag = " & _
'       Str(mblnArchivedFlag) & _
'       " where workspace_id = " & _
'       Str(mlngWorkspaceId)
'
'   mdbsStepMaster.Execute strUpdate,
dbFailOnError
'
   Exit Sub

ModifyWorkspaceErr:

   Call LogErrors(Errors)
   mstrSource = mstrModuleName &
"ModifyWorkspace"
   On Error GoTo 0
   Err.Raise vbObjectError +
errWorkspaceUpdateFailed, _
       mstrSource,
LoadResString(errWorkspaceUpdateFailed)

End Sub
Public Property Get WorkspaceName() As
String

   WorkspaceName = mstrWorkspaceName

End Property

Public Property Get WorkspaceId() As Long

   WorkspaceId = mlngWorkspaceId

End Property

Private Sub Class_Initialize()

   ' Each function will append it's own name
to this
   ' variable
   mstrSource = "cWorkspace."

   Set mFieldValue = New cStringSM

End Sub

Private Sub Class_Terminate()

   Set mdbsStepMaster = Nothing
   Set mFieldValue = Nothing

End Sub
Attribute VB_Name = "DatabaseSM"
'  FILE:      DatabaseSM.bas
'             Microsoft TPC-H Kit Ver. 1.00
'             Copyright Microsoft, 1999
'             All Rights Reserved
'
'
'  PURPOSE:    Contains all the database
initialization/cleanup
'             procedures for the project. Also
contains upgrade
'             database upgrade functions.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
' This module is called DatabaseSM, since
Database is a standard
' Visual Basic object and we want to avoid any
confusion with it.

Option Explicit

Public wrkJet As Workspace
Public dbsAttTool As Database
Public gblnDbOpen As Boolean
Public gRunEngine As rdoEngine

' Used to indicate the source module name
when errors
' are raised by this module
Private Const mstrModuleName As String =
"DatabaseSM."
Public Const gsDefDBFileExt As String =
".stp"
Private Const msDefDBFile As String =
"\SMData" & gsDefDBFileExt

Private Const merrFileNotFound As Integer =
3024
Private Const merrDaoTableMissing As
Integer = 3078

Private Const
STEPMASTER_SETTINGS_VAL_NAME_D
BFILE As String = "WorkspaceFile"

Public Const DEF_NO_COUNT_DISPLAY
As Boolean = False
Public Const DEF_NO_EXECUTE
As Boolean = False
Public Const DEF_PARSE_QUERY_ONLY
As Boolean = False
Public Const
DEF_ANSI_QUOTED_IDENTIFIERS   As
Boolean = False
Public Const DEF_ANSI_NULLS          As
Boolean = True
Public Const DEF_SHOW_QUERY_PLAN
As Boolean = False
Public Const DEF_SHOW_STATS_TIME
As Boolean = False
Public Const DEF_SHOW_STATS_IO
As Boolean = False
Public Const
DEF_PARSE_ODBC_MSG_PREFIXES   As
Boolean = True
Public Const DEF_ROW_COUNT
As Long = 0
Public Const
DEF_TSQL_BATCH_SEPARATOR     As
String = "GO"
Public Const DEF_QUERY_TIME_OUT          As
Long = 0
Public Const DEF_SERVER_LANGUAGE
As String = "(Default)"
Public Const
DEF_CHARACTER_TRANSLATION     As
Boolean = True
Public Const DEF_REGIONAL_SETTINGS
As Boolean = False

Public Const PARAM_DEFAULT_DIR           As
String = "DEFAULT_DIR"
Public Const PARAM_DEFAULT_DIR_DESC
As String = "Default destination directory " & _
            "for all output and error files. If it is
blank, the StepMaster installation directory will be
used."

Public Const PARAM_RUN_ID                As
String = "RUN_ID"
Public Const PARAM_RUN_ID_DESC           As
String = "The run identifier for a run. " & _
            "Any modifications will be overwritten
before each run."

Public Const PARAM_OUTPUT_DIR            As
String = "OUTPUT_DIR"
Public Const PARAM_OUTPUT_DIR_DESC
As String = "The output directory for a run. " & _
            "Any modifications will be overwritten
before each run."

Public Const
CONNECTION_STRINGS_TO_NAME_SUFFIX
As String = "_NAME"

Private Const TBL_RUN_STEP_HDR As String =
"run_header"
Private Const TBL_RUN_STEP_DTLS As String =
"run_step_details"
Public Const TBL_CONNECTION_DTLS As
String = "connection_dtls"
Public Const TBL_CONNECTION_STRINGS As
String = "workspace_connections"
Public Const TBL_STEPS As String = "att_steps"

Public Const FLD_ID_CONN_NAME As String =
"connection_name_id"
Public Const FLD_ID_WORKSPACE As String =
"workspace_id"
Public Const FLD_ID_STEP As String = "step_id"
Public Const FLD_ID_PARAMETER As String =
"parameter_id"

Public Const
FLD_CONN_DTL_CONNECTION_NAME As
String = "connection_name"
Public Const
FLD_CONN_DTL_CONNECTION_STRING As
String = "connection_string_name"
Public Const
FLD_CONN_DTL_CONNECTION_TYPE As
String = "connection_type"

Public Const
FLD_CONN_STR_CONNECTION_NAME As
String = "connection_name"

Public Const FLD_STEPS_EXEC_MECHANISM
As String = "execution_mechanism"
Public Const FLD_STEPS_EXEC_DTL As String
= "start_directory"
Public Const FLD_STEPS_VERSION_NO As
String = "version_no"
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
Public Const DATA_TYPE_CURRENCY
As String = "CURRENCY"
Public Const DATA_TYPE_LONG As
String = "Long"
Public Const DATA_TYPE_INTEGER As
String = "INTEGER"
Public Const DATA_TYPE_TEXT255 As
String = "Text(255)"

Private Sub InsertBuiltInParameter(dbFile
As Database, sParamName As String, _
      sParamValue As String, sParamDesc
As String)

   Dim sBuf As String
   Dim cTempStr As New cStringSM
   Dim lId As Long
   Dim rTemp As DAO.Recordset
   Dim rParam As DAO.Recordset
   Dim cTempSeq As cSequence

   ' Create the passed in built-in parameter,
for each workspace in the db
   Set cTempSeq = New cSequence
   Set cTempSeq.IdDatabase = dbFile
   cTempSeq.IdentifierColumn =
FLD_ID_PARAMETER

   sBuf = "select * from att_workspaces "
   Set rTemp = dbFile.OpenRecordset(sBuf,
dbOpenSnapshot)
   If rTemp.RecordCount <> 0 Then
      rTemp.MoveFirst

      While Not rTemp.EOF
         sBuf = "select * from
workspace_parameters " & _
                " where workspace_id = " &
Str(rTemp!workspace_id) & _
                " and parameter_name = " &
cTempStr.MakeStringFieldValid(sParamNa
me)
         Set rParam =
dbFile.OpenRecordset(sBuf,
dbOpenSnapshot)
         If rParam.RecordCount <> 0 Then
            rParam.MoveFirst
            ' Since the parameter already
exists, change it to a built-in type
            sBuf = "update
workspace_parameters " & _
                  " set parameter_type = " &
CStr(gintParameterBuiltIn) & _
                  ", description = " &
cTempStr.MakeStringFieldValid(sParamDe
sc) & _
                  " where workspace_id = " &
Str(rTemp!workspace_id) & _
                  " and parameter_id = " &
Str(rParam!parameter_id)
         Else
            ' Else, insert a parameter record
            lId = cTempSeq.Identifier
            sBuf = "insert into
workspace_parameters " & _
                  "( workspace_id,
parameter_id, " & _
                  " parameter_name,
parameter_value, " & _
                  " description, parameter_type
) " & _
                  " values ( " & _
                  Str(rTemp!workspace_id) &
", " & Str(lId) & ", " & _
cTempStr.MakeStringFieldValid(sParamName
) & ", " & _

cTempStr.MakeStringFieldValid(sParamValue
) & ", " & _

cTempStr.MakeStringFieldValid(sParamDesc)
& ", " & _
                  CStr(gintParameterBuiltIn) & _
                  " ) "
         End If
         dbFile.Execute sBuf, dbFailOnError
         rParam.Close

         rTemp.MoveNext
      Wend
   End If
   rTemp.Close

End Sub
Public Sub InitRunEngine()

   Set gRunEngine = New rdoEngine
   gRunEngine.rdoDefaultCursorDriver =
rdUseServer

End Sub

Public Function DefaultDBFile() As String
   DefaultDBFile = GetSetting(App.Title,
"Settings",
STEPMASTER_SETTINGS_VAL_NAME_D
BFILE, App.Path & msDefDBFile)
End Function

Public Sub CloseDatabase()

   Dim dbsInstance As Database
   Dim recInstance As Recordset

   On Error GoTo CloseDatabaseErr

   ' Close all open recordsets and databases in
the workspace
   For Each dbsInstance In wrkJet.Databases

      For Each recInstance In
dbsAttTool.Recordsets
         recInstance.Close
      Next recInstance
      dbsInstance.Close

   Next dbsInstance

   Set dbsAttTool = Nothing

   gblnDbOpen = False
   wrkJet.Close

   Exit Sub

CloseDatabaseErr:

   Call LogErrors(Errors)
   Resume Next

End Sub

Private Function NoDbChanges(sVerTo As
String, sVerFrom As String) As Boolean

   If sVerTo = gsVersion242 And sVerFrom =
gsVersion241 Then
      NoDbChanges = True
   ElseIf sVerTo = gsVersion242 And sVerFrom =
gsVersion24 Then
      NoDbChanges = True
   ElseIf sVerTo = gsVersion253 And sVerFrom =
gsVersion251 Then
      NoDbChanges = True
   ElseIf sVerTo = gsVersion255 And sVerFrom =
gsVersion251 Then
      NoDbChanges = True
   Else
      NoDbChanges = False
   End If

End Function

Public Function SMOpenDatabase(Optional
strDbName As String = gstrEmptyString) As
Boolean
   Dim sVersion As String
   Dim bOpeningDb As Boolean ' This flag is used
to check if OpenDatabase failed

   On Error GoTo OpenDatabaseErr

   bOpeningDb = False
   SMOpenDatabase = False

   ' Create Microsoft Jet Workspace object.
   If Not gblnDbOpen Then
      Set wrkJet =
CreateWorkspace("att_tool_workspace_setup",
"admin", gstrEmptyString, dbUseJet)
   End If

   ' Prompt the user for the database file if it is not
passed in
   If StringEmpty(strDbName) Then
      strDbName = BrowseDBFile
      If StringEmpty(strDbName) Then
         Exit Function
      End If
   End If

   Do
      If gblnDbOpen Then
#If Not RUN_ONLY Then
         CloseOpenWorkspaces
#End If
         Set wrkJet =
CreateWorkspace("att_tool_workspace_setup",
"admin", gstrEmptyString, dbUseJet)
      End If

      ' Toggle the bOpeningDb flag around the
OpenDatabase method - the value
      ' of this flag will be checked by the error
handler to determine if it is
      ' the OpenDatabase that failed.
      BugMessage "DB File: " & strDbName

      bOpeningDb = True
      ' Open the database for exclusive use
      Set dbsAttTool =
wrkJet.OpenDatabase(strDbName, Options:=True)
      bOpeningDb = False

      If dbsAttTool Is Nothing Then
         ' If the file is not present in the directory,
display
         ' an error and ask the user to enter a new
path
         Call ShowError(errOpenDbFailed,
OptArgs:=strDbName)

         strDbName = BrowseDBFile
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
      Else
         sVersion = DBVersion(dbsAttTool)

         ' Make sure the application and db
version numbers match
         If sVersion = gsVersion Then
            Call InitializeData(strDbName)
            gblnDbOpen = True
            SMOpenDatabase = True
         Else
            If UpgradeDb(wrkJet, dbsAttTool,
gsVersion, sVersion) Then
               Call InitializeData(strDbName)
               gblnDbOpen = True
               SMOpenDatabase = True
            Else
               dbsAttTool.Close
               Set dbsAttTool = Nothing

               ShowError
errVersionMismatch, _
                  OptArgs:=" Please install
Version "' & gsVersion & "' of the
workspace definition file."
               strDbName = BrowseDBFile
            End If
         End If
      End If
   Loop While gblnDbOpen = False And
Not StringEmpty(strDbName)

   Exit Function

OpenDatabaseErr:
   Call DisplayErrors(Errors)

   ' If the OpenDatabase failed, continue
   If bOpeningDb Then
      Resume Next
   End If

   Call ShowError(errOpenDbFailed,
OptArgs:=strDbName)

End Function
Private Sub InitializeData(sDb As String)

   Set gcParameters = New cArrParameters
   Set gcParameters.ParamDatabase =
dbsAttTool

   Set gcSteps = New cArrSteps
   Set gcSteps.StepDB = dbsAttTool

   Set gcConstraints = New cArrConstraints
   Set gcConstraints.ConstraintDB =
dbsAttTool

   Set gcConnections = New cConnections
   Set gcConnections.ConnDb = dbsAttTool

   Set gcConnDtls = New cConnDtls
   Set gcConnDtls.ConnDb = dbsAttTool

   ' Disable the error handler since this is not
a critical step
   On Error GoTo 0
   SaveSetting App.Title, "Settings",
STEPMASTER_SETTINGS_VAL_NAME
_DBFILE, sDb
End Sub
Private Sub
UpdateContinuationCriteria(dbFile As
DAO.Database)

   Dim qyTemp As DAO.QueryDef
   Dim sBuf As String

   On Error GoTo
UpdateContinuationCriteriaErr

   sBuf = "Since this version of the executable
incorporates failure processing, " & _
         "the upgrade will update the On Failure
field for each of the steps " & _
         "to 'Continue' to be compatible with the
existing behaviour. " & _
         "Proceed?"
   If Not Confirm(Buttons:=vbYesNo,
strMessage:=sBuf, strTitle:="Upgrade
database") Then
      Exit Sub
   End If

   ' Create a recordset object to retrieve all
steps for
   ' the given workspace
   sBuf = " update att_steps a " & _
      " set continuation_criteria = " &
CStr(gintOnFailureContinue) & _
      " where archived_flag = [archived] "

   ' Find the highest X-component of the
version number
   sBuf = sBuf & " AND cint( mid(
version_no, 1, instr( version_no, " & gstrDQ &
gstrVerSeparator & gstrDQ & " ) - 1 ) ) = " &
_
      " ( select max( cint( mid( version_no, 1,
instr( version_no, " & gstrDQ &
gstrVerSeparator & gstrDQ & " ) - 1 ) ) ) " & _
      " from att_steps AS d " & _
      " WHERE a.step_id = d.step_id ) "

   ' Find the highest Y-component of the
version number for the highest X-component
   sBuf = sBuf & " AND cint( mid(
version_no, instr( version_no, " & gstrDQ &
gstrVerSeparator & gstrDQ & " ) + 1 ) ) = " &
_
      " ( select max( cint( mid( version_no,
instr( version_no, " & gstrDQ &
gstrVerSeparator & gstrDQ & " ) + 1 ) ) ) " &
_
      " from att_steps AS b " & _
      " Where a.step_id = b.step_id " & _
      " AND cint( mid( version_no, 1, instr(
version_no, " & gstrDQ & gstrVerSeparator &
gstrDQ & " ) - 1 ) ) = " & _
      " ( select max( cint( mid( version_no, 1,
instr( version_no, " & gstrDQ &
gstrVerSeparator & gstrDQ & " ) - 1 ) ) ) " & _
      " from att_steps AS c " & _
      " WHERE a.step_id = c.step_id ) ) "

   ' Create a temporary Querydef object
   Set qyTemp =
dbFile.CreateQueryDef(gstrEmptyString,
sBuf)
   qyTemp.Parameters("archived").Value =
False

   qyTemp.Execute dbFailOnError
   qyTemp.Close

   Exit Sub

UpdateContinuationCriteriaErr:
   Call LogErrors(Errors)
   Err.Raise vbObjectError + errModifyStepFailed,
mstrModuleName, _
         LoadResString(errModifyStepFailed)

End Sub

Private Sub UpdateDbDtls(dbFile As Database,
sNewVersion As String)

   Dim sSql As String
   Dim cTemp As New cStringSM

   On Error GoTo UpdateDbDtlsErr

   sSql = "update db_details " & _
         " set db_version = " &
cTemp.MakeStringFieldValid(sNewVersion)

   dbFile.Execute sSql, dbFailOnError

   Exit Sub

UpdateDbDtlsErr:
   Call LogErrors(Errors)
   Err.Raise vbObjectError + errUpgradeFailed,
mstrModuleName, _
         LoadResString(errUpgradeFailed)

End Sub

Private Sub Upgrade10to21(UpgradeWsp As
DAO.Workspace, dbFile As Database, sVersion As
String)

   Dim sSql As String

   On Error GoTo Upgrade10to21Err

   Call UpdateDbDtls(dbFile, sVersion)

   Call UpdateContinuationCriteria(dbFile)

   Exit Sub

Upgrade10to21Err:
   UpgradeWsp.Rollback
   Call LogErrors(Errors)
   Err.Raise vbObjectError + errUpgradeFailed,
mstrModuleName, _
         LoadResString(errUpgradeFailed)

End Sub
Private Sub Upgrade21to23(UpgradeWsp As
DAO.Workspace, dbFile As Database, sVersion As
String)

   Dim sBuf As String
   Dim cTempStr As New cStringSM

   On Error GoTo Upgrade21to23Err

   ' Add a parameter type field and a description
field to the parameter table
   sBuf = "alter table workspace_parameters " & _
         " add column description TEXT(255) "
   dbFile.Execute sBuf, dbFailOnError

   sBuf = "alter table workspace_parameters " & _
         " add column parameter_type INTEGER "
   dbFile.Execute sBuf, dbFailOnError

   ' Initialize the parameter type on all parameters to
indicate generic parameters
   sBuf = "update workspace_parameters " & _
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
        " set parameter_type = " &
CStr(gintParameterGeneric)
   dbFile.Execute sBuf, dbFailOnError

   sBuf = "Release 2.3 onwards, connection
string parameters will be " & _
        "displayed in a separate node. After
this upgrade, all connection " & _
        "string parameters will appear under
the Globals/Connection Strings " & _
        "node in the workspace. "
   Call MsgBox(sBuf, vbOKOnly +
vbApplicationModal, "Upgrade database")

   ' Update the parameter type on all
parameters that look like db connection
strings
   sBuf = "update workspace_parameters "
& _
        " set parameter_type = " &
CStr(gintParameterConnect) & _
        " where UCase(parameter_value)
like '*DRIVER*' " & _
        " or UCase(parameter_value) like
'*DSN*'"
   dbFile.Execute sBuf, dbFailOnError

   ' Add an elapsed time field to the
run_step_details table - this field is
   ' needed to store the elapsed time in
milliseconds.
   sBuf = "alter table run_step_details " & _
        " add column elapsed_time LONG "
   dbFile.Execute sBuf, dbFailOnError

   ' The failure_details field has some data
for the case when an ODBC failure
   ' threshold was specified. Since that's no
longer relevant, update the failure_details
   ' field for records with failure_criteria =
gintFailureODBC to empty.
   ' failure_criteria = gintFailureODBC = 1
   sBuf = "update att_steps " & _
        " set failure_details = " &
cTempStr.MakeStringFieldValid(gstrEmpty
String) & _
        " where failure_criteria = '1'"
   dbFile.Execute sBuf, dbFailOnError

   Call UpdateDbDtls(dbFile, sVersion)

   UpgradeWsp.CommitTrans

   On Error GoTo DropColumnErr

   UpgradeWsp.BeginTrans

   ' This ddl cannot be in the same
transaction as the failure_details update
   ' But we can do this in a separate
transaction since we do not expect this
   ' statement to fail - AND, it doesn't matter
if this transaction fails
   ' Drop the failure_criteria column from
the att_steps table
   sBuf = "alter table att_steps " & _
        " drop column failure_criteria "
   dbFile.Execute sBuf, dbFailOnError

   Exit Sub

DropColumnErr:
   Call LogErrors(Errors)
   ShowError errDeleteColumnFailed
   Exit Sub

Upgrade21to23Err:
   UpgradeWsp.Rollback
   Call LogErrors(Errors)
   Err.Raise vbObjectError +
errUpgradeFailed, mstrModuleName, _
        LoadResString(errUpgradeFailed)

End Sub
Private Sub Upgrade23to24(UpgradeWsp As
DAO.Workspace, dbFile As Database,
sVersion As String)

   Dim sBuf As String
   Dim lId As Long
   Dim rTemp As DAO.Recordset
   Dim cTempStr As New cStringSM

   On Error GoTo Upgrade23to24Err

   ' Add a new table for connection properties
   sBuf = CreateConnectionsTableScript()
   ' TODO: Not sure of column sizes for row
count, tsql_batch_separator and
server_language
   dbFile.Execute sBuf, dbFailOnError

   ' Move all connection parameters from the
parameter table to the connections tables
   ' Insert default values for the newly added
connection properties
   sBuf = "select * from workspace_parameters
" & _
        "where parameter_type = " &
CStr(gintParameterConnect)
   Set rTemp = dbFile.OpenRecordset(sBuf,
dbOpenSnapshot)
   lId = 1
   If rTemp.RecordCount <> 0 Then
      rTemp.MoveFirst

      While Not rTemp.EOF
         sBuf = "insert into
workspace_connections " & _
         "( workspace_id, connection_id, " & _
         "connection_name, connection_value,
" & _
         "description, no_count_display, " & _
         "no_execute, parse_query_only, " & _
         "ANSI_quoted_identifiers,
ANSI_nulls, " & _
         "show_query_plan, show_stats_time, "
& _
         "show_stats_io,
parse_odbc_msg_prefixes, " & _
         "row_count, tsql_batch_separator, " &
_
         "query_time_out, server_language, " &
_
         "character_translation,
regional_settings ) " & _
         " values ( " & _
         Str(rTemp!workspace_id) & ", " &
Str(lId) & ", " & _
         cTempStr.MakeStringFieldValid("'" &
rTemp!parameter_name) & ", " & _
         cTempStr.MakeStringFieldValid("'" &
rTemp!parameter_value) & ", " & _
         cTempStr.MakeStringFieldValid("'" &
rTemp!Description) & ", " & _
         Str(DEF_NO_COUNT_DISPLAY) &
", " & _
         Str(DEF_NO_EXECUTE) & ", " &
Str(DEF_PARSE_QUERY_ONLY) & ", " & _
         Str(DEF_ANSI_QUOTED_IDENTIFIERS)
& ", " & Str(DEF_ANSI_NULLS) & ", " & _
         Str(DEF_SHOW_QUERY_PLAN) & ", " &
Str(DEF_SHOW_STATS_TIME) & ", " & _
         Str(DEF_SHOW_STATS_IO) & ", " &
Str(DEF_PARSE_ODBC_MSG_PREFIXES) & ",
" & _
         Str(DEF_ROW_COUNT) & ", " &
cTempStr.MakeStringFieldValid(DEF_TSQL_BAT
CH_SEPARATOR) & ", " & _
         Str(DEF_QUERY_TIME_OUT) & ", " &
cTempStr.MakeStringFieldValid(DEF_SERVER_L
ANGUAGE) & ", " & _
         Str(DEF_CHARACTER_TRANSLATION)
& ", " & Str(DEF_REGIONAL_SETTINGS) & _
         " ) "
         dbFile.Execute sBuf, dbFailOnError

         lId = lId + 1
         rTemp.MoveNext
      Wend
   End If
   rTemp.Close

   ' Add an identifier column for the connection_id
field
   sBuf = "alter table att_identifiers " & _
        " add column connection_id long "
   dbFile.Execute sBuf, dbFailOnError

   ' Initialize the value of the connection identifier,
initialized above
   sBuf = "update att_identifiers " & _
        " set connection_id = " & Str(lId)
   dbFile.Execute sBuf, dbFailOnError

   ' Delete all connection strings from the parameter
table
   sBuf = "delete from workspace_parameters " & _
         "where parameter_type = " &
CStr(gintParameterConnect)
   dbFile.Execute sBuf, dbFailOnError

   ' Create the built-in parameter, default directory,
for each workspace in the db
   Call InsertBuiltInParameter(dbFile,
PARAM_DEFAULT_DIR, gstrEmptyString,
PARAM_DEFAULT_DIR_DESC)

   Call UpdateDbDtls(dbFile, sVersion)

   Exit Sub

Upgrade23to24Err:
   UpgradeWsp.Rollback
   Call LogErrors(Errors)
   Err.Raise vbObjectError + errUpgradeFailed,
mstrModuleName, _
        LoadResString(errUpgradeFailed)

End Sub
Private Sub Upgrade243to25(UpgradeWsp As
DAO.Workspace, dbFile As Database, sVersion As
String)

   Dim sBuf As String
   Dim qy As DAO.QueryDef
   Dim rTemp As DAO.Recordset
   Dim lId As Long
   Dim cTempStr As New cStringSM

   On Error GoTo Upgrade243to25Err

   sBuf = "Release " & gsVersion25 & " onwards,
new 'Connections' must be created for all " & _
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
        "connection strings. " & vbCrLf &
vbCrLf & _
        "Connections will appear under the
Globals/Connections " & _
        "node in the workspace. " & vbCrLf
& _
        "A list of all 'Connections' (instead
of 'Connection Strings') " & _
        "in the workspace will be displayed
in the 'Connections' field for " & _
        "ODBC steps on the Step definition
screen. " & vbCrLf & vbCrLf & _
        "Each Connection can be marked as
static or dynamic. " & vbCrLf & _
        "Dynamic connections will be
created when a step starts execution and " &
_
        "closed once the step completes. " &
vbCrLf & _
        "Static connections will be kept open
till the run completes." & vbCrLf & vbCrLf
& _
        "Currently dynamic 'Connections'
have been created for all existing
'Connection Strings' " & _
        "with the suffix " &
CONNECTION_STRINGS_TO_NAME_S
UFFIX
   Call MsgBox(sBuf, vbOKOnly +
vbApplicationModal, "Upgrade database")

   ' Add a new table for the connection name
entity
   ' This table has been added in order to
satisfy the TPC-H requirement that
   ' all the queries in a stream need to be
executed on a single connection.
   sBuf =
CreateConnectionDtlsTableScript()
   dbFile.Execute sBuf, dbFailOnError

   ' Add an identifier column for the
connection_name_id field
   sBuf = "alter table att_identifiers " & _
        " add column " &
FLD_ID_CONN_NAME & " long "
   dbFile.Execute sBuf, dbFailOnError

   Call UpdateDbDtls(dbFile, sVersion)

   ' insert connection_dtl records for each of
the connection strings
   sBuf = "select * from " &
TBL_CONNECTION_STRINGS
   Set rTemp = dbFile.OpenRecordset(sBuf,
dbOpenSnapshot)

   sBuf = "insert into " &
TBL_CONNECTION_DTLS & _
         "( " & FLD_ID_WORKSPACE & _
         ", " & FLD_ID_CONN_NAME & _
         ", " &
FLD_CONN_DTL_CONNECTION_NAM
E & _
         ", " &
FLD_CONN_DTL_CONNECTION_STRI
NG & _
         ", " &
FLD_CONN_DTL_CONNECTION_TYPE
& " ) " & _
         " values ( [w_id], [c_id], [c_name],
[c_str], [c_type] ) "
   Set qy = dbFile.CreateQueryDef("", sBuf)

   lId = glMinId
   If rTemp.RecordCount <> 0 Then
      rTemp.MoveFirst

      While Not rTemp.EOF
         qy.Parameters("w_id").Value =
rTemp.Fields(FLD_ID_WORKSPACE)
         qy.Parameters("c_id").Value = lId
         qy.Parameters("c_name").Value =
rTemp.Fields(FLD_CONN_STR_CONNECTI
ON_NAME) &
CONNECTION_STRINGS_TO_NAME_SUF
FIX
         qy.Parameters("c_str").Value =
rTemp.Fields(FLD_CONN_STR_CONNECTI
ON_NAME)
         qy.Parameters("c_type").Value =
ConnTypeDynamic

         qy.Execute dbFailOnError

         lId = lId + 1
         rTemp.MoveNext
      Wend
   End If
   qy.Close
   rTemp.Close

   ' Initialize the value of the
connection_name_id
   sBuf = "update att_identifiers " & _
         " set " & FLD_ID_CONN_NAME & "
= " & Str(lId)
   dbFile.Execute sBuf, dbFailOnError

   ' Update the start_directory field in att_steps
to point to the newly
   ' created connections
   Call ReadStepsInWorkspace(rTemp, qy,
glInvalidId, dbLoad:=dbFile, _
         bSelectArchivedRecords:=False)

   sBuf = "update " & TBL_STEPS & _
      " set " & FLD_STEPS_EXEC_DTL & " =
[c_name] " & _
      " where " & FLD_ID_STEP & " = [s_id] "
& _
      " and " & FLD_STEPS_VERSION_NO
& " = [ver_no] "
   Set qy = dbFile.CreateQueryDef("", sBuf)

   If rTemp.RecordCount <> 0 Then
      rTemp.MoveFirst

      While Not rTemp.EOF
         If
rTemp.Fields(FLD_STEPS_EXEC_MECHAN
ISM).Value = gintExecuteODBC Then
            If Not (StringEmpty("" &
rTemp.Fields(FLD_STEPS_EXEC_DTL)))
Then
               sBuf =
rTemp.Fields(FLD_STEPS_EXEC_DTL)
               ' Strip the enclosing "%"
characters
               sBuf = Mid(sBuf, 2, Len(sBuf) -
2) &
CONNECTION_STRINGS_TO_NAME_SUF
FIX

               qy.Parameters("c_name").Value =
sBuf
               qy.Parameters("s_id").Value =
rTemp.Fields(FLD_ID_STEP)
               qy.Parameters("ver_no").Value =
rTemp.Fields(FLD_STEPS_VERSION_NO)
               qy.Execute dbFailOnError
            End If
         End If
         rTemp.MoveNext
      Wend
   End If

   qy.Close
   rTemp.Close

   Exit Sub

Upgrade243to25Err:
   UpgradeWsp.Rollback
   Call LogErrors(Errors)
   Err.Raise vbObjectError + errUpgradeFailed,
mstrModuleName, _
         LoadResString(errUpgradeFailed)

End Sub
Private Sub Upgrade25to251(UpgradeWsp As
DAO.Workspace, dbFile As Database, sVersion As
String)

   On Error GoTo Upgrade25to251Err

   ' Create the built-in parameters, run_id and
output_dir, for each workspace in the db
   Call InsertBuiltInParameter(dbFile,
PARAM_RUN_ID, gstrEmptyString,
PARAM_RUN_ID_DESC)
   Call InsertBuiltInParameter(dbFile,
PARAM_OUTPUT_DIR, gstrEmptyString,
PARAM_OUTPUT_DIR_DESC)

   Call UpdateDbDtls(dbFile, sVersion)

   Exit Sub

Upgrade25to251Err:
   UpgradeWsp.Rollback
   Call LogErrors(Errors)
   Err.Raise vbObjectError + errUpgradeFailed,
mstrModuleName, _
         LoadResString(errUpgradeFailed)

End Sub

Private Sub Upgrade242to243(UpgradeWsp As
DAO.Workspace, dbFile As Database, sVersion As
String)

   Dim sBuf As String
   Dim cTempStr As New cStringSM
   Dim iResponse As Integer

   On Error GoTo DeleteHistoryErr

   Call DeleteRunHistory(dbFile)

   On Error GoTo Upgrade242to243Err

   UpgradeWsp.CommitTrans

   UpgradeWsp.BeginTrans

   ' Add a parameter type field and a description
field to the parameter table
   sBuf = "alter table run_step_details " & _
         " add column parent_instance_id LONG "

   dbFile.Execute sBuf, dbFailOnError

   sBuf = "alter table run_step_details " & _
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
        " add column iterator_value
TEXT(255) "

   dbFile.Execute sBuf, dbFailOnError

   Call AlterFieldType(dbFile,
TBL_RUN_STEP_DTLS, "start_time",
DATA_TYPE_CURRENCY)
   Call AlterFieldType(dbFile,
TBL_RUN_STEP_DTLS, "end_time",
DATA_TYPE_CURRENCY)
   Call AlterFieldType(dbFile,
TBL_RUN_STEP_HDR, "start_time",
DATA_TYPE_CURRENCY)
   Call AlterFieldType(dbFile,
TBL_RUN_STEP_HDR, "end_time",
DATA_TYPE_CURRENCY)

   Call UpdateDbDtls(dbFile, sVersion)

   Exit Sub

DeleteHistoryErr:
   ' This is not a critical error - continue with
upgrade
   Call LogErrors(Errors)
   Resume Next

Upgrade242to243Err:
   UpgradeWsp.Rollback
   Call LogErrors(Errors)
   Err.Raise vbObjectError +
errUpgradeFailed, mstrModuleName, _
        LoadResString(errUpgradeFailed)

End Sub
'*************************************
******************************
' The AlterFieldType Sub procedure requires
three string
' parameters. The first string specifies the
name of the table
' containing the field to be changed. The
second string specifies
' the name of the field to be changed. The
third string specifies
' the new data type for the field.
'*************************************
******************************

Private Sub AlterFieldType(dbFile As
Database, TblName As String, FieldName
As String, _
              NewDataType As String)
   Dim qdf As DAO.QueryDef
   Dim sSql As String

   ' Add a temporary field to the table.
   sSql = "ALTER TABLE [" & TblName &
_
        "] ADD COLUMN AlterTempField
" & NewDataType
   Set qdf = dbFile.CreateQueryDef("",
sSql)
   qdf.Execute

   ' Copy the data from old field into the new
field.
   qdf.SQL = "UPDATE DISTINCTROW
[" & TblName & "] SET AlterTempField =
[" & FieldName & "]"
   qdf.Execute

   ' Delete the old field.
   qdf.SQL = "ALTER TABLE [" & TblName
& "] DROP COLUMN [" & FieldName & "]"
   qdf.Execute

   ' Rename the temporary field to the old
field's name.
   dbFile.TableDefs("[" & TblName &
"]").Fields("AlterTempField").Name =
FieldName
   dbFile.TableDefs.Refresh

   ' Clean up.
End Sub
Private Sub Upgrade01to21(UpgradeWsp As
DAO.Workspace, dbFile As DAO.Database,
sVersion As String)
   Dim sSql As String

   On Error GoTo Upgrade01to21Err

   sSql = "Create table db_details (" & _
        "db_version          Text(50) " & _
   ");"

   dbFile.Execute sSql, dbFailOnError

   sSql = "insert into db_details " & _
        "( db_version ) values ( '" & sVersion
& "' ) "

   dbFile.Execute sSql, dbFailOnError

   Call UpdateContinuationCriteria(dbFile)

   Exit Sub

Upgrade01to21Err:
   Call LogErrors(Errors)
   UpgradeWsp.Rollback
   Err.Raise vbObjectError +
errUpgradeFailed, mstrModuleName, _
        LoadResString(errUpgradeFailed)

End Sub
Private Function UpgradeDb(UpgradeWsp As
DAO.Workspace, dbFile As Database, _
      sVerTo As String, sVerFrom As String)
As Boolean

   Dim sMsg As String

   On Error GoTo UpgradeDbErr

   UpgradeDb = False
   If Not ValidUpgrade(sVerTo, sVerFrom)
Then Exit Function

   If NoDbChanges(sVerTo, sVerFrom) Then
      UpgradeDb = True
      Exit Function
   End If

   sMsg = "The database needs to be upgraded
from Version " & sVerFrom & _
        " to Version " & sVerTo & "." &
vbCrLf & _
        "Proceed?"
   If Not Confirm(Buttons:=vbYesNo,
strMessage:=sMsg, strTitle:="Upgrade
database") Then
      Exit Function
   End If

   UpgradeWsp.BeginTrans

   Select Case sVerFrom
      Case gsVersion25
         Call Upgrade25to251(UpgradeWsp, dbFile,
gsVersion251)

      Case gsVersion243
         Call Upgrade243to25(UpgradeWsp, dbFile,
gsVersion25)
         Call Upgrade25to251(UpgradeWsp, dbFile,
gsVersion251)

      Case gsVersion24, gsVersion241,
gsVersion242
         sMsg = "After this upgrade, the run history
for previous runs will no longer be available. " & _
              "Continue?"
         If Not Confirm(Buttons:=vbYesNo,
strMessage:=sMsg, strTitle:="Upgrade database")
Then
            UpgradeWsp.CommitTrans
            Exit Function
         End If
         Call Upgrade242to243(UpgradeWsp,
dbFile, gsVersion243)
         Call Upgrade243to25(UpgradeWsp, dbFile,
gsVersion25)
         Call Upgrade25to251(UpgradeWsp, dbFile,
gsVersion251)

      Case gsVersion23
         Call Upgrade23to24(UpgradeWsp, dbFile,
gsVersion24)
         Call Upgrade242to243(UpgradeWsp,
dbFile, gsVersion242)
         Call Upgrade243to25(UpgradeWsp, dbFile,
gsVersion25)
         Call Upgrade25to251(UpgradeWsp, dbFile,
gsVersion251)

      Case gsVersion21
         Call Upgrade21to23(UpgradeWsp, dbFile,
gsVersion23)
         Call Upgrade23to24(UpgradeWsp, dbFile,
gsVersion24)
         Call Upgrade242to243(UpgradeWsp,
dbFile, gsVersion242)
         Call Upgrade243to25(UpgradeWsp, dbFile,
gsVersion25)
         Call Upgrade25to251(UpgradeWsp, dbFile,
gsVersion251)

      Case gsVersion10
         Call Upgrade10to21(UpgradeWsp, dbFile,
gsVersion21)
         Call Upgrade21to23(UpgradeWsp, dbFile,
gsVersion23)
         Call Upgrade23to24(UpgradeWsp, dbFile,
gsVersion24)
         Call Upgrade242to243(UpgradeWsp,
dbFile, gsVersion242)
         Call Upgrade243to25(UpgradeWsp, dbFile,
gsVersion25)
         Call Upgrade25to251(UpgradeWsp, dbFile,
gsVersion251)

      Case gsVersion01
         Call Upgrade01to21(UpgradeWsp, dbFile,
gsVersion21)
         Call Upgrade21to23(UpgradeWsp, dbFile,
gsVersion23)
         Call Upgrade23to24(UpgradeWsp, dbFile,
gsVersion24)
         Call Upgrade242to243(UpgradeWsp,
dbFile, gsVersion242)
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
        Call Upgrade243to25(UpgradeWsp,
dbFile, gsVersion25)
        Call Upgrade25to251(UpgradeWsp,
dbFile, gsVersion251)

    End Select

    UpgradeWsp.CommitTrans

    UpgradeDb = True
    Exit Function

UpgradeDbErr:
    Call LogErrors(Errors)
    ShowError errUpgradeFailed

End Function
Private Function DBVersion(TestDb As
Database) As String
    ' Retrieves the database version
    Dim rVersion As Recordset

    On Error GoTo DBVersionErr

    Set rVersion =
TestDb.OpenRecordset("Select db_version
from db_details ", _
        dbOpenForwardOnly)

    BugAssert rVersion.RecordCount <> 0
    DBVersion = rVersion!db_version

    rVersion.Close
    Exit Function

DBVersionErr:
    If Err.Number = merrDaoTableMissing
Then
        DBVersion = gsVersion01
    Else
        LogErrors Errors
        Err.Raise vbObjectError +
errUpgradeFailed, mstrModuleName, _
            LoadResString(errUpgradeFailed)
    End If

End Function

Private Function ValidUpgrade(sVerTo As
String, sVerFrom As String) As Boolean

    If sVerTo = gsVersion And sVerFrom =
gsVersion251 Then
        ValidUpgrade = True
    ElseIf sVerTo = gsVersion And
sVerFrom = gsVersion25 Then
        ValidUpgrade = True
    ElseIf sVerTo = gsVersion And
sVerFrom = gsVersion243 Then
        ValidUpgrade = True
    ElseIf sVerTo = gsVersion And
sVerFrom = gsVersion242 Then
        ValidUpgrade = True
    ElseIf sVerTo = gsVersion And
sVerFrom = gsVersion241 Then
        ValidUpgrade = True
    ElseIf sVerTo = gsVersion And
sVerFrom = gsVersion24 Then
        ValidUpgrade = True
    ElseIf sVerTo = gsVersion And
sVerFrom = gsVersion23 Then
        ValidUpgrade = True
    ElseIf sVerTo = gsVersion And
sVerFrom = gsVersion21 Then
        ValidUpgrade = True
    ElseIf sVerTo = gsVersion And sVerFrom =
gsVersion10 Then
        ValidUpgrade = True
    ElseIf sVerTo = gsVersion And sVerFrom =
gsVersion01 Then
        ValidUpgrade = True
    Else
        ValidUpgrade = False
    End If

End Function

Attribute VB_Name = "DebugSM"
'  FILE:      DebugSM.bas
'             Microsoft TPC-H Kit Ver. 1.00
'             Copyright Microsoft, 1999
'             All Rights Reserved
'
'
'  PURPOSE:    Contains all the functions that
carry out error/debug
'             processing for the project.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
' Most of the functions in this module that
manipulate the
' error object do not have an On Error GoTo
statement - this
' is because it will clear the passed in error
object - let
' the calling functions handle the errors raised
by this
' module, if any
Option Explicit

' Used to indicate the source module name
when errors
' are raised by this module
Private Const mstrModuleName As String =
"DebugSM."

Private mcLogFile As cFileSM
Private mcErrorFile As cFileSM

Private Const
FORMAT_MESSAGE_FROM_SYSTEM =
&H1000
Private Const
FORMAT_MESSAGE_IGNORE_INSERTS =
&H200
Private Const pNull = 0

Declare Function FormatMessage Lib
"kernel32" Alias "FormatMessageA" (ByVal
dwFlags As Long, lpSource As Any, ByVal
dwMessageId As Long, ByVal dwLanguageId
As Long, ByVal lpbuffer As String, ByVal
nSize As Long, Arguments As Long) As Long
Public Function Confirm(Optional
lngMessageCode As conConfirmMsgCodes, _
        Optional lngTitleCode As
conConfirmMsgTitleCodes, _
        Optional TitleParameter As String, _
        Optional ByVal Buttons As Integer = -1, _
        Optional strMessage As String =
gstrEmptyString, _
        Optional strTitle As String =
gstrEmptyString) _
        As Boolean
    ' Displays a confirmation message
corresponding to the
    ' passed in message code. Returns True if the
user says
    ' Ok and False otherwise

    Dim intResponse As Integer
    Dim intButtonStyle As Integer

    On Error GoTo ConfirmErr

    Confirm = False

    ' If the buttons style hasn't been specified, set the
    ' default style to display OK and Cancel buttons
    If Buttons = -1 Then
        intButtonStyle = vbOKCancel
    Else
        intButtonStyle = Buttons
    End If

    ' Find the message string for the passed in code
    If StringEmpty(strMessage) Then
        strMessage =
Trim$(LoadResString(lngMessageCode))
    End If

    If StringEmpty(strTitle) Then
        strTitle =
Trim$(LoadResString(lngTitleCode))
    End If

    If Not StringEmpty(TitleParameter) Then
        strTitle = strTitle & Chr$(vbKeySpace) & _
            gstrSQ & TitleParameter & gstrSQ
    End If

    ' Display the confirmation message with the
Cancel button
    ' set to the default - assume that we are
confirming
    ' potentially dangerous operations!
    intResponse = MsgBox(strMessage, _
        intButtonStyle + vbQuestion +
vbApplicationModal, _
        strTitle)

    ' Translate the user response into a True/False
return code
    If intButtonStyle = vbOKCancel Then
        If intResponse = vbOK Then
            Confirm = True
        Else
            Confirm = False
        End If
    Else
        If intResponse = vbYes Then
            Confirm = True
        Else
            Confirm = False
        End If
    End If

    Exit Function

ConfirmErr:
    ' Log the error code raised by Visual Basic
    Call LogErrors(Errors)
    On Error GoTo 0
    gstrSource = mstrModuleName & "Confirm"
    Err.Raise vbObjectError + errConfirmFailed, _
        gstrSource, _
        LoadResString(errConfirmFailed)

End Function
Public Sub LogSystemError()
    Dim eErrCode As Long

    eErrCode = GetLastError()
    If eErrCode <> 0 Then
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
      WriteToFile "System Error: " &
eErrCode & vbCrLf & ApiError(eErrCode),
_
            blnError:=True
   End If

End Sub
Public Function ApiError(ByVal e As Long)
As String

   Dim s As String
   Dim c As Long

   s = String(256, 0)
   c =
FormatMessage(FORMAT_MESSAGE_FR
OM_SYSTEM Or _

FORMAT_MESSAGE_IGNORE_INSERT
S, _
         pNull, e, 0&, s, Len(s), ByVal pNull)
   If c Then ApiError = e & ": " & Left$(s,
c)

End Function

' Output flags determine output destination
of BugAsserts and messages
#Const afLogfile = 1
#Const afMsgBox = 2
#Const afDebugWin = 4
#Const afAppLog = 8

' Display appropriate error message, and
then stop
' program.  These errors should NOT be
possible in
' shipping product.
Sub BugAssert(ByVal fExpression As
Boolean, _
           Optional sExpression As String)
#If afDebug Then
   If fExpression Then Exit Sub
   BugMessage "BugAssert failed: " &
sExpression
   Stop
#End If
End Sub

Sub BugMessage(sMsg As String)

#If afDebug And afLogfile Then
   ' Since we are writing log messages, the
error flag is turned off
   Call WriteToFile(sMsg, False)
#End If
#If afDebug And afMsgBox Then
   MsgBox sMsg
#End If
#If afDebug And afDebugWin Then
   Debug.Print sMsg
#End If
#If afDebug And afAppLog Then
    App.LogEvent sMsg
#End If

End Sub
Public Function ProjectLogFile() As String

   ProjectLogFile = mcLogFile.FileName

End Function
Public Function ProjectErrorFile() As String

   ProjectErrorFile = mcErrorFile.FileName

End Function

Private Sub WriteToFile(sMsg As String,
Optional ByVal blnError As Boolean)

   ' Calls procedures to write the passed in
message to the log -
   ' The blnError flag is used to indicate that
the message
   ' should be logged to the error file - by
default the log
   ' file is used

   Dim mcFileObj As cFileSM
   Dim strFileName As String
   Dim strFileHdr As String

   On Error GoTo WriteToFileErr

   If blnError Then
      If mcErrorFile Is Nothing Then
         Set mcErrorFile = New cFileSM
      End If
      Set mcFileObj = mcErrorFile
   Else
      If mcLogFile Is Nothing Then
         Set mcLogFile = New cFileSM
      End If
      Set mcFileObj = mcLogFile
   End If

   If StringEmpty(mcFileObj.FileName) Then
      If blnError Then
         strFileName = gstrProjectPath & "\" &
App.EXEName & ".ERR"
         strFileHdr = "Stepmaster Errors"
      Else
         strFileName = gstrProjectPath & "\" &
App.EXEName & ".DBG"
         strFileHdr = "Stepmaster Log"
      End If

      mcFileObj.FileName = strFileName
      mcFileObj.WriteLine strFileHdr
      mcFileObj.WriteLine "Log start time : "
& Now
   End If

   mcFileObj.WriteLine sMsg

   Exit Sub

WriteToFileErr:
   ' Display the error code raised by Visual
Basic
   Call DisplayErrors(Errors)
   ' An error message would've been displayed
by the called
   ' procedures

End Sub
Public Sub WriteMessage(sMsg As String)

   Call WriteToFile(sMsg, True)

End Sub

Sub BugTerm()
#If afDebug And afLogfile Then
   ' Close log file
   mcLogFile.CloseFile
#End If
End Sub
Public Sub ShowError(ByVal ErrorCode As
errErrorConstants, _
      Optional ByVal ErrorSource As String =
gstrEmptyString, _
      Optional ByVal OptArgs As String =
gstrEmptyString, _
      Optional ByVal DoWriteError As Boolean =
True)

   If DoWriteError Then
      ' Call a procedure to write the error to a log file
      Call WriteError(ErrorCode, ErrorSource,
OptArgs)
   End If

   ' Re-initialize the values of the Error object
before
   ' displaying the error to the user
   Call InitErrObject(ErrorCode, ErrorSource,
OptArgs)

   Call DisplayErrors(Errors)

   Err.Clear

End Sub
Public Sub WriteError(ByVal ErrorCode As
errErrorConstants, _
      Optional ByVal ErrorSource As String =
gstrEmptyString, _
      Optional ByVal OptArgs As String =
gstrEmptyString)

   ' Initialize the values of the Error object before
   ' calling the log function
   Call InitErrObject(ErrorCode, ErrorSource,
OptArgs)

   Call LogErrors(Errors)

   Err.Clear

End Sub
Private Sub InitErrObject(ByVal ErrorCode As
errErrorConstants, _
      Optional ByVal ErrorSource As String =
gstrEmptyString, _
      Optional ByVal OptArgs As String =
gstrEmptyString)

   Dim lngError As Long

   lngError = IIf(ErrorCode > vbObjectError And
ErrorCode < vbObjectError + 65535, _
         ErrorCode - vbObjectError, ErrorCode)
   Err.Number = lngError + vbObjectError
   Err.Description = LoadResString(lngError) &
OptArgs
   Err.Source = App.EXEName & ErrorSource

End Sub
Public Sub ShowMessage(ByVal MessageCode As
errErrorConstants, _
      Optional ByVal OptArgs As String)

   Dim strMessage As String

   On Error GoTo ShowMessageErr

   strMessage = LoadResString(MessageCode) &
OptArgs

   ' Write the error to a log file
   BugMessage strMessage
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   MsgBox strMessage, vbOKOnly

   Exit Sub

ShowMessageErr:
   ' Log the error and exit
   Call DisplayErrors(Errors)

End Sub
Public Sub ShowMessageStr(sMessage As
String)

   ' Write the error to a log file
   BugMessage sMessage

   MsgBox sMessage, vbOKOnly

End Sub

Public Sub DisplayErrors(myErrCollection
As Errors)
   Dim strError As String
   Dim errLoop As Error
   Dim errCode As Long

   ' Enumerate Errors collection and display
properties of
   ' each Error object.
   If Err.Number <> 0 Then
      If Err.Number > vbObjectError And
Err.Number < (vbObjectError + 65536)
Then
         errCode = Err.Number -
vbObjectError
      Else
         errCode = Err.Number
      End If
      strError = "Error # " & Str(errCode) &
" was generated by " _
         & Err.Source & Chr(13) &
Err.Description
      MsgBox strError, , "Error",
Err.HelpFile, Err.HelpContext
   Else
      For Each errLoop In myErrCollection
         With errLoop
            If Err.Number > vbObjectError
And Err.Number < (vbObjectError + 65536)
Then
               errCode = .Number -
vbObjectError
            Else
               errCode = .Number
            End If
            strError = "Error #" & errCode &
vbCrLf
            strError = strError & "    " &
.Description & vbCrLf
            strError = strError & _
               "    (Source: " & .Source & ")"
& vbCrLf
'           strError = strError & _
'              "Press F1 to see topic " &
.HelpContext & vbCrLf
'           strError = strError & _
'              "    in the file " & .HelpFile &
"."
         End With

         MsgBox strError
      Next
   End If

End Sub
Public Sub LogErrors(myErrCollection As
Errors)
   Dim cColErrors As cVectorStr
   Dim strError As String
   Dim errLoop As Error
   Dim errCode As Long
   Dim lngIndex As Long

   Set cColErrors = New cVectorStr

   ' Enumerate Errors collection and display
properties of
   ' each Error object.
   If Err.Number <> 0 Then
      If Err.Number > vbObjectError And
Err.Number < (vbObjectError + 65536) Then
         errCode = Err.Number - vbObjectError
      Else
         errCode = Err.Number
      End If
      strError = "Error # " & Str(errCode) & "
was generated by " _
         & Err.Source & vbCrLf &
Err.Description

      cColErrors.Add strError
   End If

   ' Log all database errors, if any
   For Each errLoop In myErrCollection
      With errLoop
         If Err.Number > vbObjectError And
Err.Number < (vbObjectError + 65536) Then
            errCode = .Number - vbObjectError
         Else
            errCode = .Number
         End If
         strError = "Error #" & errCode &
vbCrLf
         strError = strError & "    " &
.Description & vbCrLf
         strError = strError & _
            "    (Source: " & .Source & ")" &
vbCrLf
      End With

      cColErrors.Add strError
   Next

   ' We can have a error handler now that we
have stored all
   ' errors away safely! - having an error
handler before
   ' enumerating all the errors would have
cleared the error
   ' collection
   On Error GoTo LogErrorsErr
   gstrSource = mstrModuleName &
"LogErrors"

   For lngIndex = 0 To cColErrors.Count - 1
      strError = cColErrors(lngIndex)
      Debug.Print strError
      Call WriteToFile(strError, True)
   Next lngIndex

   Set cColErrors = Nothing

   Exit Sub

LogErrorsErr:
   ' Display the error code raised by Visual
Basic
   DisplayErrors Errors
   On Error GoTo 0
   ShowError errUnableToWriteError,
DoWriteError:=False

End Sub
Attribute VB_Name = "FileCommon"
'  FILE:      FileCommon.bas
'             Microsoft TPC-H Kit Ver. 1.00
'             Copyright Microsoft, 1999
'             All Rights Reserved
'
'
'  PURPOSE:   This module contains common
functionality to display
'             the File Open dialog.
'  Contact:   Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

' Used to indicate the source module name when
errors
' are raised by this module
Private Const mstrModuleName As String =
"FileCommon."

Private Enum EOpenFile
   OFN_OVERWRITEPROMPT = &H2
   OFN_HIDEREADONLY = &H4
   OFN_FILEMUSTEXIST = &H1000
   OFN_EXPLORER = &H80000
End Enum

' The locations for the different output files are
presented to
' the user in a list box. These constants are used
while loading the
' data and while reading the data from the list box.
' These constants also represent the different file
types that are
' displayed to the user in File Open dialogs
Public Enum gFileTypes
   gintOutputFile = 0
'  gintLogFile = 1
   gintErrorFile
   gintStepTextFile
   gintOutputCompareFile
   gintDBFile
   gintDBFileNew
   gintImportFile
   gintExportFile
End Enum

Public Const gsSqlFileSuffix = ".sql"
Public Const gsCmdFileSuffix = ".cmd"

Public Const gsOutputFileSuffix = ".out"
Public Const gstrLogFileSuffix = ".log"
Public Const gsErrorFileSuffix = ".err"
Public Function BrowseDBFile() As String
   ' Prompts the user for a database file with the
workspace information
   ' Call CallFileDialog to display the open file
dialog
   BrowseDBFile = CallFileDialog(gintDBFile)

End Function
Public Function CallFileDialog(intFileType As
Integer, _
      Optional ByVal strDefaultFile As String =
gstrEmptyString) As String
   ' This function initializes the values of the filter
property,
   ' the dialog title and flags for the File Open dialog
depending
   ' on the FileType passed in
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
    ' It then calls ShowFileOpenDialog to set
these properties and
    ' display the File Open dialog to the user

    ' All the properties used by the File Open
dialog are defined
    ' as constants in this function and passed
to ShowFileOpenDialog
    ' as parameters. So if any of the dialog
properties need to be
    ' modified, these constants are what need
to be changed
    Const s_DLG_TITLE_OPEN = "Open"
    Const s_DLG_TITLE_NEW = "New"
    Const s_DLG_TITLE_IMPORT =
"Import From"
    Const s_DLG_TITLE_EXPORT =
"Export To"

    Const mlng_FILE_STEP_TEXT_FLAGS
= OFN_EXPLORER Or
OFN_FILEMUSTEXIST Or
OFN_HIDEREADONLY
    Const
mlng_FILE_OUTPUT_COMPARE_FLAG
S = mlng_FILE_STEP_TEXT_FLAGS
    Const mlng_FILE_DB_FLAGS =
mlng_FILE_STEP_TEXT_FLAGS
    Const mlng_FILE_OUTPUT_FLAGS =
OFN_EXPLORER Or
OFN_HIDEREADONLY Or
OFN_OVERWRITEPROMPT
    Const mlng_FILE_LOG_FLAGS =
mlng_FILE_OUTPUT_FLAGS
    Const mlng_FILE_ERROR_FLAGS =
mlng_FILE_OUTPUT_FLAGS
    Const mlng_FILE_DB_NEW_FLAGS =
mlng_FILE_OUTPUT_FLAGS

    Const mstr_FILE_ALL_FILTER = "|All
Files (*.*)|*.*"
    Const mstr_FILE_STEP_TEXT_FILTER
= "Query Files (*" & gsSqlFileSuffix & _
          ")|*" & gsSqlFileSuffix &
"|Command Script Files (*" &
gsCmdFileSuffix & _
          ")|*" & gsCmdFileSuffix
    Const
mstr_FILE_OUTPUT_COMPARE_FILTE
R = "Text Files (*.txt)|*.txt"
    Const mstr_FILE_OUTPUT_FILTER =
"Output Files (*.out)|*.out"
    Const mstr_FILE_LOG_FILTER = "Log
Files (*.log)|*.log"
    Const mstr_FILE_ERROR_FILTER =
"Error Files (*.err)|*.err"
    Const mstr_FILE_DB_FILTER =
"Stepmaster Workspace Files (*" &
gsDefDBFileExt & ")|*" & gsDefDBFileExt

    Dim strFileName As String

    On Error GoTo CallFileDialogErr

    Select Case intFileType
       Case gintStepTextFile
          strFileName =
ShowFileOpenDialog( _

mstr_FILE_STEP_TEXT_FILTER &
mstr_FILE_ALL_FILTER, _
             s_DLG_TITLE_OPEN, _

mlng_FILE_STEP_TEXT_FLAGS, _
             strDefaultFile)

       Case gintOutputCompareFile
          strFileName = ShowFileOpenDialog( _

mstr_FILE_OUTPUT_COMPARE_FILTER
& mstr_FILE_ALL_FILTER, _
             s_DLG_TITLE_OPEN, _

mlng_FILE_OUTPUT_COMPARE_FLAGS,
_
             strDefaultFile)

       Case gintOutputFile
          strFileName = ShowFileOpenDialog( _
             mstr_FILE_OUTPUT_FILTER &
mstr_FILE_ALL_FILTER, _
             s_DLG_TITLE_OPEN, _
             mlng_FILE_OUTPUT_FLAGS, _
             strDefaultFile)

'       Case gintLogFile
'          strFileName = ShowFileOpenDialog( _
'             mstr_FILE_LOG_FILTER &
mstr_FILE_ALL_FILTER, _
'             s_DLG_TITLE_OPEN, _
'             mlng_FILE_LOG_FLAGS, _
'             strDefaultFile)
'
       Case gintErrorFile
          strFileName = ShowFileOpenDialog( _
             mstr_FILE_ERROR_FILTER &
mstr_FILE_ALL_FILTER, _
             s_DLG_TITLE_OPEN, _
             mlng_FILE_ERROR_FLAGS, _
             strDefaultFile)

       Case gintDBFile
          strFileName = ShowFileOpenDialog( _
             mstr_FILE_DB_FILTER &
mstr_FILE_ALL_FILTER, _
             s_DLG_TITLE_OPEN, _
             mlng_FILE_DB_FLAGS, _
             strDefaultFile)

       Case gintDBFileNew
          strFileName = ShowFileOpenDialog( _
             mstr_FILE_DB_FILTER &
mstr_FILE_ALL_FILTER, _
             s_DLG_TITLE_NEW, _
             mlng_FILE_DB_NEW_FLAGS, _
             strDefaultFile)

       Case gintImportFile
          strFileName = ShowFileOpenDialog( _
             mstr_FILE_DB_FILTER &
mstr_FILE_ALL_FILTER, _
             s_DLG_TITLE_IMPORT, _
             mlng_FILE_DB_FLAGS, _
             strDefaultFile)

       Case gintExportFile
          strFileName = ShowFileOpenDialog( _
             mstr_FILE_DB_FILTER &
mstr_FILE_ALL_FILTER, _
             s_DLG_TITLE_EXPORT, _
             mlng_FILE_DB_FLAGS, _
             strDefaultFile)

       Case Else
          BugAssert True, "Incorrect file type
passed in."
          ' Default processing will be for the
output file
          strFileName = ShowFileOpenDialog( _
             mstr_FILE_OUTPUT_FILTER &
mstr_FILE_ALL_FILTER, _
             s_DLG_TITLE_OPEN, _
             mlng_FILE_OUTPUT_FLAGS, _
             strDefaultFile)

    End Select
    CallFileDialog = strFileName

    Exit Function

CallFileDialogErr:
    CallFileDialog = gstrEmptyString
    ' Log the error code raised by Visual Basic
    Call LogErrors(Errors)
    gstrSource = mstrModuleName &
"CallFileDialog"
    Call ShowError(errBrowseFailed)

End Function

VERSION 5.00
Begin VB.Form frmSplash
   BorderStyle    =   3  'Fixed Dialog
   ClientHeight   =   4710
   ClientLeft     =   45
   ClientTop      =   45
   ClientWidth    =   7455
   ControlBox     =   0  'False
   Icon           =   "frmSplash.frx":0000
   LinkTopic      =   "Form1"
   LockControls   =   -1  'True
   MaxButton      =   0  'False
   MinButton      =   0  'False
   ScaleHeight    =   4710
   ScaleWidth     =   7455
   ShowInTaskbar  =   0  'False
   StartUpPosition =  2  'CenterScreen
   Visible        =   0  'False
   Begin VB.Frame fraMainFrame
      Height         =   4590
      Left           =   45
      TabIndex       =   0
      Top            =   -15
      Width          =   7380
      Begin VB.PictureBox picLogo
         Height         =   2385
         Left           =   510
         Picture        =   "frmSplash.frx":0442
         ScaleHeight    =   2325
         ScaleWidth     =   1755
         TabIndex       =   1
         Top            =   855
         Width          =   1815
      End
      Begin VB.Label lblReserved
         AutoSize       =   -1  'True
         Caption        =   "All Rights Reserved."
         Height         =   195
         Left           =   5520
         TabIndex       =   8
         Top            =   4080
         Width          =   1440
      End
      Begin VB.Label lblCopyright
         AutoSize       =   -1  'True
         Caption        =   "Copyright © 1998"
         BeginProperty Font
            Name           =   "Arial"
            Size           =   8.25
            Charset        =   0
            Weight         =   400
            Underline      =   0  'False
            Italic         =   0  'False
            Strikethrough  =   0  'False
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
      EndProperty
      Height        =   210
      Left          =   5550
      TabIndex      =   7
      Tag           =   "Copyright"
      Top           =   3850
      Width         =   1380
      WordWrap      =   -1  'True
    End
    Begin VB.Label lblCompany
      AutoSize      =   -1  'True
      Caption       =   "Microsoft
Corporation"
      BeginProperty Font
        Name          =   "Arial"
        Size          =   8.25
        Charset       =   0
        Weight        =   400
        Underline     =   0   'False
        Italic        =   0   'False
        Strikethrough =   0   'False
      EndProperty
      Height        =   210
      Left          =   5460
      TabIndex      =   6
      Tag           =   "Company"
      Top           =   3640
      Width         =   1560
    End
    Begin VB.Label lblRestrictions
      AutoSize      =   -1  'True
      Caption       =   "See License
Agreement for Restrictions"
      Height        =   195
      Left          =   460
      TabIndex      =   5
      Top           =   4080
      Width         =   2790
    End
    Begin VB.Label lblProductName
      AutoSize      =   -1  'True
      Caption       =   "StepMaster"
      BeginProperty Font
        Name          =   "Arial"
        Size          =   32.25
        Charset       =   0
        Weight        =   700
        Underline     =   0   'False
        Italic        =   0   'False
        Strikethrough =   0   'False
      EndProperty
      Height        =   765
      Left          =   2670
      TabIndex      =   4
      Tag           =   "Product"
      Top           =   1200
      Width         =   3495
    End
    Begin VB.Label lblVersion
      Alignment     =   1  'Right Justify
      AutoSize      =   -1  'True
      Caption       =   "Version 2.4"
      BeginProperty Font
        Name          =   "Arial"
        Size          =   12
        Charset       =   0
        Weight        =   700
        Underline     =   0   'False
        Italic        =   0   'False
        Strikethrough =   0   'False
      EndProperty
      Height        =   285
      Left          =   4800
      TabIndex      =   3
      Tag           =   "Version"
      Top           =   2040
      Width         =   1275
    End
    Begin VB.Label lblWarning
      AutoSize      =   -1  'True
      Caption       =   "Do Not Redistribute"
      BeginProperty Font
        Name          =   "Arial"
        Size          =   8.25
        Charset       =   0
        Weight        =   400
        Underline     =   0   'False
        Italic        =   0   'False
        Strikethrough =   0   'False
      EndProperty
      Height        =   210
      Left          =   480
      TabIndex      =   2
      Tag           =   "Warning"
      Top           =   3850
      Width         =   1380
    End
  End
End
Attribute VB_Name = "frmSplash"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
'  FILE:      frmSplash.frm
'             Microsoft TPC-H Kit Ver. 1.00
'             Copyright Microsoft, 1999
'             All Rights Reserved
'
'
'  PURPOSE:    Splash screen for StepMaster
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit


Private Sub Form_Load()
'    lblVersion.Caption = "Version " &
App.Major & "." & App.Minor & "." &
App.Revision
    lblProductName.Caption = App.Title
    lblVersion.Caption = GetVersionString

End Sub



VERSION 5.00
Begin VB.Form frmWorkspaceOpen
  BorderStyle     =   3  'Fixed Dialog
  Caption         =   "Open Workspace"
  ClientHeight    =   2550
  ClientLeft      =   45
  ClientTop       =   330
  ClientWidth     =   4695
  LinkTopic       =   "Form1"
  LockControls    =   -1  'True
  MaxButton       =   0   'False
  MinButton       =   0   'False
  ScaleHeight     =   2550
  ScaleWidth      =   4695
  ShowInTaskbar   =   0   'False
  StartUpPosition =   1  'CenterOwner
  Begin VB.CommandButton cmdOK
    Caption         =   "OK"
    Default         =   -1  'True
    Height          =   375
    Left            =   2160
    TabIndex        =   2
    Top             =   2040
    Width           =   1095
  End
  Begin VB.CommandButton cmdCancel
    Cancel          =   -1  'True
    Caption         =   "Cancel"
    Height          =   375
    Left            =   3360
    TabIndex        =   3
    Top             =   2040
    Width           =   1095
  End
  Begin VB.ListBox lstWorkspaces
    Height          =   1425
    ItemData        =
"frmWorkspaceOpen.frx":0000
    Left            =   240
    List            =   "frmWorkspaceOpen.frx":0002
    MultiSelect     =   2  'Extended
    TabIndex        =   1
    Top             =   480
    Width           =   4215
  End
  Begin VB.Label lblWorkspaces
    AutoSize        =   -1  'True
    Caption         =   "Workspace"
    Height          =   195
    Left            =   240
    TabIndex        =   0
    Top             =   120
    Width           =   825
  End
End
Attribute VB_Name = "frmWorkspaceOpen"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
'  FILE:      frmWorkspaceOpen.frm
'             Microsoft TPC-H Kit Ver. 1.00
'             Copyright Microsoft, 1999
'             All Rights Reserved
'
'
'  PURPOSE:    Used to display a list of all
workspaces in a
'             workspace definition file for Open, Import,
Export
'             and Run (in SMRunOnly) workspace
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit
#If RUN_ONLY Then
    Private WithEvents cRunOnlyInst As cRunOnly
Attribute cRunOnlyInst.VB_VarHelpID = -1

Private Sub cRunOnlyInst_Done()
    ShowFree
End Sub

Private Sub Run()
    Dim iIndex As Integer

    Set cRunOnlyInst = New cRunOnly

    For iIndex = 0 To Me.lstWorkspaces.ListCount -
1
        If Me.lstWorkspaces.Selected(iIndex) Then
            ShowBusy
            cRunOnlyInst.WorkspaceId =
Me.lstWorkspaces.ItemData(Me.lstWorkspaces.List
Index)
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
        cRunOnlyInst.WspName =
Me.lstWorkspaces.List(Me.lstWorkspaces.L
istIndex)
        cRunOnlyInst.RunWsp
      End If
   Next iIndex

End Sub

#End If

Private Sub cmdCancel_Click()

   Unload Me

End Sub

Private Sub cmdOK_Click()

#If RUN_ONLY Then
   Call Run
#Else
   Call WorkspaceOpenOk
#End If

End Sub


Private Sub lstWorkspaces_DblClick()

   ' A double click on the workspaces list
box is considered
   ' equivalent to selecting an item in the list
and then selecting
   ' the OK command
   Call cmdOK_Click

End Sub

Attribute VB_Name = "IteratorCommon"
'  FILE:      IteratorCommon.bas
'             Microsoft TPC-H Kit Ver. 1.00
'             Copyright Microsoft, 1999
'             All Rights Reserved
'
'
'  PURPOSE:   Contains functionality
common across StepMaster and
'             SMRunOnly, pertaining to iterators
'             Specifically, functions to read
iterators records
'             in the workspace, load them in an
array and so on.
'  Contact:   Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

' Used to indicate the source module name
when errors
' are raised by this module
Private Const mstrModuleName As String =
"IteratorCommon."

Public Const gintMinIteratorSequence As
Integer = 0

Public Sub RangeComplete(vntIterators As
Variant)
   ' This is a debug procedure
   ' Checks if the from, to and step values
are present in
   ' the array

   Dim bReset As Byte
   Dim bShift As Byte
   Dim lngIndex As Long

   ' Set the three lowest order bits to 1
   bReset = 7

   BugAssert IsArray(vntIterators) And Not
IsEmpty(vntIterators), _
         "Iterators not specified!"

   For lngIndex = LBound(vntIterators) To _
         UBound(vntIterators)
      bShift = 1
      bShift = bShift * (2 ^
(vntIterators(lngIndex).IteratorType - 1))

      bReset = bReset Xor bShift
   Next lngIndex

   ' Assert that all the elements are present
   BugAssert bReset = 0, "Range not
completely specified!"

End Sub
Public Sub LoadIteratorsForWsp(cStepsCol As
cArrSteps, _
      ByVal lngWorkspaceId As Long,
rstStepsInWsp As Recordset)
   ' Initializes the step records in with all the
iterator
   ' values for each step

   Dim recIterators As Recordset

   On Error GoTo LoadIteratorsForWspErr

#If QUERY_ALL Then
   Dim dtStart As Date

   dtStart = Now
   Set recIterators =
ReadWspIterators(lngWorkspaceId)

   Call LoadIteratorsArray(cStepsCol,
recIterators)

   recIterators.Close

   BugMessage "QueryAll Read + load took: "
& CStr(DateDiff("s", dtStart, Now))

#Else
   Dim dtStart As Date
   Dim qyIt As DAO.QueryDef
   Dim sSql As String

   dtStart = Now
   If rstStepsInWsp.RecordCount = 0 Then
      Exit Sub
   End If

   ' This method has the advantage that if the
steps are queried right, everything else follows
   sSql = "Select step_id, version_no, type,
iterator_value, " & _
      " sequence_no " & _
      " from iterator_values " & _
      " where step_id = [s_id] " & _
      " and version_no = [ver_no] "

   ' Order the iterators by sequence within a
step
   sSql = sSql & " order by sequence_no "

   Set qyIt =
dbsAttTool.CreateQueryDef(gstrEmptyString, sSql)
   rstStepsInWsp.MoveFirst

   While Not rstStepsInWsp.EOF

      qyIt.Parameters("s_id").Value =
rstStepsInWsp!step_id
      qyIt.Parameters("ver_no").Value =
rstStepsInWsp!version_no

      Set recIterators =
qyIt.OpenRecordset(dbOpenSnapshot)

      Call LoadIteratorsArray(cStepsCol,
recIterators)
      recIterators.Close

      rstStepsInWsp.MoveNext
   Wend

   qyIt.Close

   BugMessage "Query step at a time Read + load
took: " & CStr(DateDiff("s", dtStart, Now))

#End If

   Exit Sub

LoadIteratorsForWspErr:
   LogErrors Errors
   gstrSource = mstrModuleName &
"LoadIteratorsForWsp"
   On Error GoTo 0
   Err.Raise vbObjectError +
errLoadRsInArrayFailed, _
         gstrSource, _
         LoadResString(errLoadRsInArrayFailed)

End Sub
Private Function ReadWspIterators(ByVal
lngWorkspaceId As Long) As Recordset

   ' This function will return a recordset that is
populated
   ' with the iterators for all the steps in a given
workspace

   Dim recIterators As Recordset
   Dim qyIt As DAO.QueryDef
   Dim strSql As String

   On Error GoTo ReadWspIteratorsErr
   gstrSource = mstrModuleName &
"ReadWspIterators"

   strSql = "Select i.step_id, i.version_no, " & _
      " i.type, i.iterator_value, " & _
      " i.sequence_no " & _
      " from iterator_values i, att_steps a " & _
      " where i.step_id = a.step_id " & _
      " and i.version_no = a.version_no " & _
      " and a.workspace_id = [w_id] " & _
      " and a.archived_flag = [archived] "

   ' Find the highest X-component of the version
number
   strSql = strSql & " AND cint( mid( a.version_no,
1, instr( a.version_no, " & gstrDQ &
gstrVerSeparator & gstrDQ & " ) - 1 ) ) = " & _
      " ( select max( cint( mid( version_no, 1, instr(
version_no, " & gstrDQ & gstrVerSeparator &
gstrDQ & " ) - 1 ) ) ) " & _
      " from att_steps AS d " & _
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
      " WHERE a.step_id = d.step_id ) "

   ' Find the highest Y-component of the
version number for the highest X-
component
   strSql = strSql & " AND cint( mid(
a.version_no, instr( a.version_no, " &
gstrDQ & gstrVerSeparator & gstrDQ & " )
+ 1 ) ) = " & _
      " ( select max( cint( mid( version_no,
instr( version_no, " & gstrDQ &
gstrVerSeparator & gstrDQ & " ) + 1 ) ) ) "
& _
      " from att_steps AS b " & _
      " Where a.step_id = b.step_id " & _
      " AND cint( mid( version_no, 1, instr(
version_no, " & gstrDQ & gstrVerSeparator
& gstrDQ & " ) - 1 ) ) = " & _
      " ( select max( cint( mid( version_no, 1,
instr( version_no, " & gstrDQ &
gstrVerSeparator & gstrDQ & " ) - 1 ) ) ) "
& _
      " from att_steps AS c " & _
      " WHERE a.step_id = c.step_id ) ) "


   ' Order the iterators by sequence within a
step
   strSql = strSql & " order by i.step_id,
i.sequence_no "

   Set qyIt =
dbsAttTool.CreateQueryDef(gstrEmptyStrin
g, strSql)
   qyIt.Parameters("w_id").Value =
lngWorkspaceId
   qyIt.Parameters("archived").Value =
False

   Set recIterators =
qyIt.OpenRecordset(dbOpenSnapshot)

   qyIt.Close
   Set ReadWspIterators = recIterators

   Exit Function

ReadWspIteratorsErr:
   ' Log the error code raised by Visual
Basic
   Call LogErrors(Errors)
   On Error GoTo 0
   Err.Raise vbObjectError +
errReadDataFailed, _
         gstrSource,
LoadResString(errReadDataFailed)

End Function
Private Sub LoadIteratorsArray(cStepsCol
As cArrSteps, _
      recIterators As Recordset)
   ' Initializes the step records with the
iterators for
   ' the step

   Dim cNewIt As cIterator
   Dim cStepRec As cStep
   Dim lngStepId As Long

   On Error GoTo LoadIteratorsArrayErr
   gstrSource = mstrModuleName &
"LoadIteratorsArray"

   If recIterators.RecordCount = 0 Then
      Exit Sub
   End If

   recIterators.MoveFirst
   While Not recIterators.EOF
      Set cNewIt = New cIterator

      lngStepId =
CLng(ErrorOnNullField(recIterators,
"step_id"))
      If Not cStepRec Is Nothing Then
         If cStepRec.StepId <> lngStepId Then
            Set cStepRec =
cStepsCol.QueryStep(lngStepId)
         End If
      Else
         Set cStepRec =
cStepsCol.QueryStep(lngStepId)
      End If

      ' Initialize iterator values
      cNewIt.IteratorType =
CInt(ErrorOnNullField(recIterators, "type"))
      cNewIt.Value =
CStr(ErrorOnNullField(recIterators,
"iterator_value"))
      cNewIt.SequenceNo =
CInt(ErrorOnNullField(recIterators,
"sequence_no"))

      ' Add this record to the array of iterators
      cStepRec.LoadIterator cNewIt

      Set cNewIt = Nothing
      recIterators.MoveNext
   Wend

   Exit Sub

LoadIteratorsArrayErr:
   LogErrors Errors
   gstrSource = mstrModuleName &
"LoadIteratorsArray"
   On Error GoTo 0
   Err.Raise vbObjectError +
errLoadRsInArrayFailed, _
         gstrSource, _

LoadResString(errLoadRsInArrayFailed)

End Sub


Attribute VB_Name = "MsgConfirm"
'  FILE:      MsgConfirm.bas
'            Microsoft TPC-H Kit Ver. 1.00
'            Copyright Microsoft, 1999
'            All Rights Reserved
'
'
'  PURPOSE:    Contains constants for
confirmation messages that
'            will be displayed by StepMaster
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

' A public enum containing the codes for all the
confirmation
' messages that will be used by the project -
each of the codes
' has the prefix, con
Public Enum conConfirmMsgCodes
   conWspDelete = 2000
   conSave
   conStopRun
   conSaveConnect
   conSaveDB
End Enum

' A public enum containing the titles for all the
confirmation
' messages that will be used by the project - each of
the codes
' has the prefix, cont - most confirmation message
codes will
' have a corresponding title code in here
Public Enum conConfirmMsgTitleCodes
   contWspDelete = 3000
   contSave
   contStopRun
   contSaveConnect
   contSaveDB
End Enum

Attribute VB_Name = "OpenFiles"
'  FILE:      OpenFiles.bas
'            Microsoft TPC-H Kit Ver. 1.00
'            Copyright Microsoft, 1999
'            All Rights Reserved
'
'
'  PURPOSE:    This module holds a list of all files
that have been
'            opened by the project. This module is
needed since there
'            is no way to share static data between
different instances
'            of a class.
'            Many procedure in this module do not do
any error handling -
'            this is 'coz it is also used by procedures
that log error
'            messages and any error handler will erase
the collection
'            of errors!

'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit


' Used to indicate the source module name when
errors
' are raised by this class
Private Const mstrModuleName As String =
".OpenFiles."

Private mOpenFiles As cNodeCollections

Private Const mstrTempDir As String = "\Temp\"

' The maximum number of temporary files that we
can create in a
' session
Private Const mlngMaxFileIndex As Long =
999999
Private Const mstrFileIndexFormat As String =
"000000"
Private Const mstrTempFilePrefix As String =
"SM"
Private Const mstrTempFileSuffix As String =
".cmd"

Private Const merrFileNotFound As Long = 76
Private Function GetFileHandle(strFileName) As
cFileInfo

   Dim lngIndex As Long
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   Dim blnFileOpen As Boolean

   If Not mOpenFiles Is Nothing Then

      blnFileOpen = False
      For lngIndex = 0 To mOpenFiles.Count
- 1
         If mOpenFiles(lngIndex).FileName
= strFileName Then
            blnFileOpen = True
            Exit For
         End If
      Next lngIndex

      If blnFileOpen Then
         Set GetFileHandle =
mOpenFiles(lngIndex)
      Else
         Set GetFileHandle = Nothing
      End If
   Else
      Set GetFileHandle = Nothing
   End If

End Function

Private Function GetTempFileDir() As
String

   Dim strTempFileDir As String

   On Error GoTo GetTempFileDirErr

   strTempFileDir = gstrProjectPath &
mstrTempDir

   If StringEmpty(Dir$(strTempFileDir,
vbDirectory)) Then
      MkDir strTempFileDir
   End If

   GetTempFileDir = strTempFileDir

   Exit Function

GetTempFileDirErr:
   ' Log the error code raised by Visual
Basic
   Call LogErrors(Errors)
   gstrSource = mstrModuleName &
"GetTempFileDir"
   On Error GoTo 0
   Err.Raise vbObjectError +
errProgramError, gstrSource, _
         LoadResString(errProgramError)

End Function
Public Function
MakePathValid(strFileName As String) As
String
   ' Checks if the passed in file path is valid

   Dim strFileDir As String
   Dim strTempDir As String
   Dim strTempFile As String
   Dim intPos As Integer
   Dim intStart As Integer

   On Error GoTo MakePathValidErr
   gstrSource = mstrModuleName &
"MakePathValid"

   strTempFile = strFileName
   intPos = InstrR(strFileName,
gstrFileSeparator)

   If intPos > 0 Then
      strFileDir = Left$(strTempFile, intPos - 1)
      If StringEmpty(Dir$(strFileDir,
vbDirectory)) Then
         ' Loop through the entire path starting
at the root
         ' since Mkdir can create only one level
of sub-directory
         ' at a time
         intStart = InStr(strFileDir,
gstrFileSeparator)

         Do While strTempDir <> strFileDir

            If intStart > 0 Then
               strTempDir = Left$(strFileDir,
intStart - 1)
            Else
               strTempDir = strFileDir
            End If

            If StringEmpty(Dir$(strTempDir,
vbDirectory)) Then
               ' If the specified directory doesn't
exist, try to
               ' create it.
               MkDir strTempDir
            Else
               ' The directory exists - go to it's
sub-directory
            End If
            intStart = InStr(intStart + 1,
strFileDir, gstrFileSeparator)
         Loop

         ' Sanity check
         If StringEmpty(Dir$(strFileDir,
vbDirectory)) Then
            ' We were unable to create the file
directory
            ShowError errCreateDirectoryFailed,
gstrSource, _
                  strFileDir, DoWriteError:=False
            MakePathValid = gstrEmptyString
         Else
            MakePathValid = strTempFile
         End If
      Else
         ' The specified directory exists - we
should be able
         ' to create the output file in it
         MakePathValid = strTempFile
      End If
   Else
      ' The user has only specified a filename -
VB will try
      ' to create it in the current directory
      MakePathValid = strTempFile
   End If

   Exit Function

MakePathValidErr:
   ' Log the error code raised by Visual Basic
   Call LogErrors(Errors)
   gstrSource = mstrModuleName &
"MakePathValid"
   ' Log the filename for debug
   Call WriteError(errInvalidFile, gstrSource,
strTempFile)
   On Error GoTo 0
   Err.Raise vbObjectError + errProgramError,
gstrSource, _
         LoadResString(errProgramError)

End Function
Public Function OpenFileSM(strFileName As
String) As Integer
   Dim intHFile As Integer
   Dim NewFileInfo As cFileInfo

   On Error GoTo OpenFileSMErr
   gstrSource = mstrModuleName & "OpenFileSM"

   If StringEmpty(strFileName) Then
      On Error GoTo 0
      Err.Raise vbObjectError + errInvalidFile,
gstrSource, _
            LoadResString(errInvalidFile)
   End If

   If mOpenFiles Is Nothing Then
      Set mOpenFiles = New cNodeCollections
   End If

   Set NewFileInfo = GetFileHandle(strFileName)

   If NewFileInfo Is Nothing Then
      ' The file has not been opened yet

      ' If the filename has not been initialized, do not
      ' attempt to open it
      strFileName = MakePathValid(strFileName)

      If strFileName <> gstrEmptyString Then
         intHFile = FreeFile
         Open strFileName For Output Shared As
intHFile

         Set NewFileInfo = New cFileInfo
         NewFileInfo.FileHandle = intHFile
         NewFileInfo.FileName = strFileName
         mOpenFiles.Load NewFileInfo
      Else
         ' Either the directory was invalid or s'thing
failed
         ' Display the error to the user instead of
trying
         ' to log to the file
         ShowError errInvalidFile, gstrSource,
strFileName, _
               DoWriteError:=False
         intHFile = 0
      End If
   Else
      intHFile = NewFileInfo.FileHandle
   End If

   OpenFileSM = intHFile

   Exit Function

OpenFileSMErr:
   ' Log the error code raised by Visual Basic
   Call LogErrors(Errors)
   ' The Open command failed for some reason -
write an error
   ' and let the calling function handle the error
   ShowError errInvalidFile, gstrSource,
strFileName, _
         DoWriteError:=False
   OpenFileSM = 0

End Function
Public Function CreateTempFile() As String

   Dim strTempFileDir As String
   Dim strTempFileName As String
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   Static lngLastFileIndex As Long

   On Error GoTo CreateTempFileErr

   strTempFileDir = GetTempFileDir()

   Do
      If lngLastFileIndex =
mlngMaxFileIndex Then
         On Error GoTo 0
         Err.Raise vbObjectError +
errMaxTempFiles, gstrSource, _

LoadResString(errMaxTempFiles)
      End If

      lngLastFileIndex = lngLastFileIndex +
1
      strTempFileName =
mstrTempFilePrefix & _
            Format$(lngLastFileIndex,
mstrFileIndexFormat) & _
            mstrTempFileSuffix

      If Not
StringEmpty(Dir$(strTempFileDir &
strTempFileName)) Then
         ' Remove any files left over from a
previous run,
         ' if they still exist
         Kill strTempFileDir &
strTempFileName
      End If

   ' Looping in case the file delete doesn't go
through for
   ' some reason
   Loop While Not
StringEmpty(Dir$(strTempFileDir &
strTempFileName))

   CreateTempFile =
GetShortName(strTempFileDir)
   CreateTempFile = CreateTempFile &
strTempFileName

   Exit Function

CreateTempFileErr:
   ' Log the error code raised by Visual
Basic
   Call LogErrors(Errors)
   gstrSource = gstrSource &
"CreateTempFile"
   On Error GoTo 0
   Err.Raise vbObjectError +
errProgramError, gstrSource, _
         LoadResString(errProgramError)

End Function
Public Sub CloseFileSM(strFileName As
String)
   Dim FileToClose As cFileInfo

   If Not mOpenFiles Is Nothing Then

      ' Get the handle to the open file, if it
exists
      Set FileToClose =
GetFileHandle(strFileName)

      If Not FileToClose Is Nothing Then
         Close FileToClose.FileHandle
         ' Remove the file info from the
collection of open files
         mOpenFiles.Unload
FileToClose.Position
      End If
   End If

End Sub
Public Sub CloseOpenFiles()
   Dim lIndex As Long

   If Not mOpenFiles Is Nothing Then
      For lIndex = mOpenFiles.Count - 1 To 0
         CloseFileSM
(mOpenFiles(lIndex).FileName)
      Next lIndex
   End If

End Sub

Attribute VB_Name = "ParameterCommon"
'  FILE:      ParameterCommon.bas
'             Microsoft TPC-H Kit Ver. 1.00
'             Copyright Microsoft, 1999
'             All Rights Reserved
'
'
'  PURPOSE:    Contains functionality common
across StepMaster and
'             SMRunOnly, pertaining to parameters
'             Specifically, functions to load
parameter records
'             in an array.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

' Used to indicate the source module name
when errors
' are raised by this module
Private Const mstrModuleName As String =
"ParameterCommon."

Public Sub
LoadRecordsetInParameterArray(rstWorkSpac
eParameters As Recordset, _
   cParamCol As cArrParameters)

   Dim cNewParameter As cParameter

   On Error GoTo
LoadRecordsetInParameterArrayErr

   If rstWorkSpaceParameters.RecordCount =
0 Then
      Exit Sub
   End If

   rstWorkSpaceParameters.MoveFirst
   While Not rstWorkSpaceParameters.EOF

      Set cNewParameter = New cParameter

      ' Initialize parameter values
      cNewParameter.ParameterId =
rstWorkSpaceParameters.Fields(0)

      ' Call a procedure to raise an error if
mandatory fields are
      ' null.
      cNewParameter.ParameterName = CStr( _

ErrorOnNullField(rstWorkSpaceParameters,
"parameter_name"))
      cNewParameter.ParameterValue =
CheckForNullField( _
            rstWorkSpaceParameters,
"parameter_value")
      cNewParameter.WorkspaceId = CStr( _

ErrorOnNullField(rstWorkSpaceParameters,
FLD_ID_WORKSPACE))
      cNewParameter.ParameterType = CStr( _

ErrorOnNullField(rstWorkSpaceParameters,
"parameter_type"))
      cNewParameter.Description =
CheckForNullField( _
            rstWorkSpaceParameters, "description")

      cParamCol.Load cNewParameter

      Set cNewParameter = Nothing
      rstWorkSpaceParameters.MoveNext
   Wend

   Exit Sub

LoadRecordsetInParameterArrayErr:
   LogErrors Errors
   gstrSource = mstrModuleName &
"LoadRecordsetInParameterArray"
   On Error GoTo 0
   Err.Raise vbObjectError +
errLoadRsInArrayFailed, gstrSource, _
         LoadResString(errLoadRsInArrayFailed)
End Sub
Attribute VB_Name = "Public"
'  FILE:      Public.bas
'             Microsoft TPC-H Kit Ver. 1.00
'             Copyright Microsoft, 1999
'             All Rights Reserved
'
'
'  PURPOSE:    This module contains all the public
constants for this project
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

Public Const gsVersion01 As String = "0.1"
Public Const gsVersion10 As String = "1.0"
Public Const gsVersion21 As String = "2.1"
Public Const gsVersion23 As String = "2.3"
Public Const gsVersion24 As String = "2.4"
Public Const gsVersion241 As String = "2.4.1"
Public Const gsVersion242 As String = "2.4.2"
Public Const gsVersion243 As String = "2.4.3"
Public Const gsVersion25 As String = "2.5"
Public Const gsVersion251 As String = "2.5.1"
Public Const gsVersion253 As String = "2.5.3"
Public Const gsVersion254 As String = "2.5.4"
Public Const gsVersion255 As String = "2.5.5"
Public Const gsVersion As String = gsVersion255

' The same form is used for the creation of new
nodes and
' updates to existing nodes (where each node can be
a parameter,
' global step, etc.) A tag is set on each flag is used to
indicate
' whether it is being called in the insert or update
mode. The
' constants for these modes are defined below
Public Const gstrInsertMode = "Insert"
Public Const gstrUpdateMode = "Update"
Public Const gstrPropertiesMode = "View"
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
Public Const gstrEmptyString = ""
Public Const gstrSQ = "'"
Public Const gstrDQ = """"
Public Const gstrVerSeparator = "."
Public Const gstrBlank = " "

' Constants used to indicate type of node
being processed
' The constants for the different objects
correspond to the
' indexes in the menu control arrays (for
both the main and popup
' menus) that are used to create new objects.
That way we can
' use the index passed in by the click event
to determine the
' type of node being processed
Public Const gintWorkspace = 1

' Decided to leave it here after some debate
over whether it
' actually belongs in the cStep class
definition
Public Enum gintStepType
   gintGlobalStep = 3
   gintManagerStep
   gintWorkerStep
End Enum

Public Const gintRunManager = 6
Public Const gintRunWorker = 7

Public Enum gintParameterNodeType
   gintParameter = 8
   gintNodeParamConnection
   gintNodeParamExtension
   gintNodeParamBuiltIn
End Enum

' Leave some constants free for newer types
of parameters (?)
Public Const gintConnectionDtl = 15

Public Enum gintLabelNodeType
   gintGlobalsLabel = 21
   gintParameterLabel
   gintParamConnectionLabel
   gintParamExtensionLabel
   gintParamBuiltInLabel
   gintConnDtlLabel
   gintGlobalStepLabel
   gintStepLabel
End Enum

Public Enum ConnectionType
   ConnTypeStatic = 1
   ConnTypeDynamic
End Enum

Public Const giDefaultConnType As Integer
= ConnTypeStatic

' The constants defined below are used to
identify the different
' tabs. If any more step properties and
thereby tabs are added
' to the tabbed dialog on the Step Properties
form, they should
' be defined here and accessed in the code
only using these
' pre-defined constants
' Note: These constants will mainly be used
by the functions that
' initialize, customize and display the Step
Properties form
Public Const gintDefinition = 0
Public Const gintExecution = 1
Public Const gintMgrDefinition = 2
Public Const gintPreExecutionSteps = 3
Public Const gintPostExecutionSteps = 4
Public Const gintFileLocations = 5

' These constants correspond to the index
values in the imagelist
' associated with the tree view control. The
imagelist contains
' the icons that will be displayed for each node.
Public Enum TreeImages
   gintImageWorkspaceClosed = 1
   gintImageWorkspaceOpen
   gintImageLabelClosed
   gintImageLabelOpen
   gintImageManagerClosedDis
   gintImageManagerClosedEn
   gintImageManagerOpenDis
   gintImageManagerOpenEn
   gintImageWorkerDis
   gintImageWorkerEn
   gintImageGlobalClosed
   gintImageGlobalOpen
   gintImageParameter
   gintImageRun
   gintImagePending
   gintImageStop
   gintImageDisabled
   gintImageAborted
   gintImageFailed
End Enum

' Public variable used to indicate the name of
the function
' that raises an error
Public gstrSource As String

' Public instances of the different collections
Public gcParameters As cArrParameters
Public gcSteps As cArrSteps
Public gcConstraints As cArrConstraints
Public gcConnections As cConnections
Public gcConnDtls As cConnDtls

' Public constants for the index values of the
different toolbar
' options. Will be used while dynamically
enabling/disabling
' these options.
Public Const tbNew = 1
Public Const tbOpen = 2
Public Const tbSave = 3

Public Const tbCut = 5
Public Const tbCopy = 6
Public Const tbPaste = 7
Public Const tbDelete = 8

Public Const tbProperties = 10
Public Const tbRun = 11
Public Const tbStop = 12

' The initial version #
Public Const gstrMinVersion As String = "0.0"
Public Const gstrGlobalParallelism As String =
"0"
Public Const gintMinParallelism As Integer =
1
Public Const gintMaxParallelism As Integer =
100

' Constant for the minimum identifier, used for
all identifier, viz.
' step, workspace, etc.
Public Const glMinId As Long = 1
Public Const glInvalidId As Long = -1

' A parameter that has a special meaning to
Stepmaster
' The system time will be substituted wherever it
occurs
' (typically as a part of the error, log ... file names
Public Const gstrTimeStamp As String =
"TIMESTAMP"
Public Const gstrEnvVarSeparator = "%"
Public Const gstrFileSeparator = "\"
Public Const gstrUnderscore = "_"

' Constants used by date and time formatting
functions
Public Const gsTimeSeparator = ":"
Public Const gsDateSeparator = "-"
Public Const gsMsSeparator = "."
Public Const gsDtFormat = "00"
Public Const gsYearFormat = "0000"
Public Const gsTmFormat = "00"
Public Const gsMSecondFormat = "000"

' Default nothing value for a date variable
Public Const gdtmEmpty As Currency = 0

Public Const FMT_WSP_LOG_FILE As String =
"yyyymmdd-hhnnss"

Public gsContCriteria() As String
' Note: Update the initialization of
gsExecutionStatus in Initialize() if the
' InstanceStatus values are modified - also the
boundary checks
Public gsExecutionStatus() As String

Public Const gsConnTypeStatic As String =
"Static"
Public Const gsConnTypeDynamic As String =
"Dynamic"

#If RUN_ONLY Then
Public Const gsCaptionRunWsp As String = "Run
Workspace"
#End If

' Valid operations on a cNode object
Public Enum Operation
   QueryOp = 1
   InsertOp = 2
   UpdateOp = 3
   DeleteOp = 4
End Enum

Attribute VB_Name = "RunCommon"
'  FILE:      RunCommon.bas
'             Microsoft TPC-H Kit Ver. 1.00
'             Copyright Microsoft, 1999
'             All Rights Reserved
'
'
'  PURPOSE:    Contains common functions that are
used during the execution
'             of a workspace.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

' Used to indicate the source module name when
errors
' are raised by this class
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
Private Const mstrModuleName As String =
".RunCommon."


Public Function
GetInstanceItValue(cInstanceRec As
cInstance) As String

   ' Returns the iterator value for the
instance, if an
   ' iterator has been defined for it
   Dim cStepIt As cRunColIt
   Dim cRunIterator As cRunItNode

   On Error GoTo GetInstanceItValueErr

   ' Since we create a dummy instance for
Disabled and Pending steps,
   ' doesn't make sense to look at their
iterators
   If cInstanceRec.Status <> gintDisabled
And cInstanceRec.Status <> gintPending
Then
      Set cStepIt = cInstanceRec.Iterators

      If Not
StringEmpty(cInstanceRec.Step.IteratorNam
e) Then
         If cStepIt.Count > 0 Then
            Set cRunIterator = cStepIt(0)
            BugAssert
cRunIterator.IteratorName =
cInstanceRec.Step.IteratorName, _
                  "The first iterator in the
collection is the " & _
                  "one that has been defined for
the step."
            If cRunIterator.IteratorName =
cInstanceRec.Step.IteratorName Then
               GetInstanceItValue =
cRunIterator.Value
            Else
               GetInstanceItValue =
gstrEmptyString
            End If
         Else
            GetInstanceItValue =
gstrEmptyString
         End If
      Else
         GetInstanceItValue =
gstrEmptyString
      End If
   Else
      GetInstanceItValue = gstrEmptyString
   End If

   Exit Function

GetInstanceItValueErr:
   ' Log the error code raised by Visual
Basic
   Call LogErrors(Errors)
   On Error GoTo 0
   gstrSource = mstrModuleName &
"GetInstanceItValue"
   Err.Raise vbObjectError +
errProgramError, gstrSource, _
         LoadResString(errProgramError)

End Function


Attribute VB_Name = "RunInstHelper"
'  FILE:      RunInstHelper.bas
'            Microsoft TPC-H Kit Ver. 1.00
'            Copyright Microsoft, 1999
'            All Rights Reserved
'
'
'  PURPOSE:    This module contains helper
procedures that are called by
'            cRunInst.cls
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

' Used to indicate the source module name
when errors
' are raised by this class
Private Const mstrModuleName As String =
"RunInstHelper."

' Should be equal to the number of steps
defined in cRunInst.cls
Public Const glngNumConcurrentProcesses As
Long = 99
Public Const gintBitsPerByte = 8
Public Function AnyStepRunning(cFreeSteps
As cVectorLng, arrFree() As Byte) As Boolean

   Dim lngIndex As Long
   Dim intPosInByte As Integer
   Dim lngTemp As Long

   ' Check if there are any running instances to
wait for
   If cFreeSteps.Count <>
glngNumConcurrentProcesses Then

      ' For every free step, reset the
corresponding element
      ' in the byte array to 0
      For lngIndex = 0 To cFreeSteps.Count - 1

         lngTemp = cFreeSteps(lngIndex) \
gintBitsPerByte
         intPosInByte = cFreeSteps(lngIndex)
Mod gintBitsPerByte

         arrFree(lngTemp) = arrFree(lngTemp)
Xor 2 ^ intPosInByte
      Next lngIndex

      AnyStepRunning = False

      ' Check if we have a non-zero bit in the
byte array
      For lngIndex = LBound(arrFree) To
UBound(arrFree) Step 1
         If arrFree(lngIndex) <> 0 Then
            ' We are waiting for a step to
complete
            AnyStepRunning = True
            Exit For
         End If
      Next lngIndex

   Else
      AnyStepRunning = False
   End If

End Function


Public Function
OrderConstraints(vntTempCons() As Variant,
_
      intConsType As ConstraintType) As Variant
   ' Returns a variant containing all the constraint
records in the order
   ' in which they should be executed

   Dim vntTemp As Variant
   Dim lngOuter As Long
   Dim lngInner As Long
   Dim cTempConstraint As cConstraint
   Dim cConstraints() As cConstraint
   Dim lngConsCount As Long
   Dim lngLbound As Long
   Dim lngUbound As Long
   Dim lngStep As Long

   On Error GoTo OrderConstraintsErr

   If intConsType = gintPreStep Then
      ' Since we are travelling up and we need to
execute the constraints
      ' for the top-level steps first, reverse the order
that they
      ' have been stored in the array
      lngLbound = UBound(vntTempCons)
      lngUbound = LBound(vntTempCons)
      lngStep = -1
   Else
      lngLbound = LBound(vntTempCons)
      lngUbound = UBound(vntTempCons)
      lngStep = 1
   End If

   lngConsCount = 0

   For lngOuter = lngLbound To lngUbound Step
lngStep
      vntTemp = vntTempCons(lngOuter)

      If Not IsEmpty(vntTemp) Then
         ' Each of the elements is an array
         For lngInner = LBound(vntTemp) To
UBound(vntTemp) Step 1
            If Not IsEmpty(vntTemp(lngInner)) Then
               Set cTempConstraint =
vntTemp(lngInner)

               If Not cTempConstraint Is Nothing
Then
                  ReDim Preserve
cConstraints(lngConsCount)
                  Set cConstraints(lngConsCount) =
cTempConstraint
                  lngConsCount = lngConsCount + 1
               End If
            End If
         Next lngInner
      End If
   Next lngOuter

   ' Set the return value of the function to the array
of
   ' constraints that has been built above
   If lngConsCount = 0 Then
      OrderConstraints = Empty
   Else
      OrderConstraints = cConstraints()
   End If

   Exit Function

OrderConstraintsErr:
   ' Log the error code raised by Visual Basic
   Call LogErrors(Errors)
   On Error GoTo 0
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   Err.Raise vbObjectError +
errExecInstanceFailed, _
         mstrModuleName,
LoadResString(errExecInstanceFailed)

End Function
Attribute VB_Name = "ShellSM"
'  FILE:      ShellSM.bas
'            Microsoft TPC-H Kit Ver. 1.00
'            Copyright Microsoft, 1999
'            All Rights Reserved
'
'
'  PURPOSE:    This module contains a
function that creates a process and
'            waits for it to complete.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

Public Function SyncShell(CommandLine
As String, Optional Timeout As Long, _
   Optional WaitForInputIdle As Boolean)
As Boolean

   Dim proc As
PROCESS_INFORMATION
   Dim Start As STARTUPINFO
   Dim ret As Long
   Dim nMilliseconds As Long

   BugMessage "Executing: " &
CommandLine
   If Timeout > 0 Then
      nMilliseconds = Timeout
   Else
      nMilliseconds = INFINITE
   End If

   'Initialize the STARTUPINFO structure:
   Start.cb = Len(Start)
   Start.dwFlags =
STARTF_USESHOWWINDOW
   Start.wShowWindow =
SW_SHOWMINNOACTIVE

   'Start the shelled application:
   CreateProcessA 0&, CommandLine, 0&,
0&, 1&, _
      NORMAL_PRIORITY_CLASS, 0&,
0&, Start, proc

   If WaitForInputIdle Then
      'Wait for the shelled application to
finish setting up its UI:
      ret = InputIdle(proc.hProcess,
nMilliseconds)
   Else
      'Wait for the shelled application to
terminate:
      ret =
WaitForSingleObject(proc.hProcess,
nMilliseconds)
   End If

   CloseHandle proc.hProcess

   'Return True if the application finished.
Otherwise it timed out or erred.
   SyncShell = (ret = WAIT_OBJECT_0)
End Function

VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "cArrConstraints"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
'  FILE:Attribute VB_Name = "SMErr"
'  FILE:      SMErr.bas
'            Microsoft TPC-H Kit Ver. 1.00
'            Copyright Microsoft, 1999
'            All Rights Reserved
'
'
'  PURPOSE:    This module contains error
code for all the errors that are
'            raised by StepMaster.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

' A public enum containing the codes for all the
error
' messages that will be displayed by the project
- each
' of the codes has the prefix, err
Public Enum errErrorConstants
   errParameterIdInvalid = 1000
   errParameterNameMandatory
   errParameterInsertFailed
   errStepLabelOrTextOrFileRequired
   errMandatoryNodeTextMissing
   errParameterUpdateFailed
   errDupConnDtlName
   errDummy14
   errContCriteriaMandatory
   errContCriteriaNullForGlobal
   errContCriteriaInvalid = 1010
   errParamSeparatorMissing
   errStepTextOrTextFileMandatory
   errStepTextOrFile
   errEnabledFlagFalseForGlobal
   errEnabledFlagLetFailed
   errDegParallelismNullForGlobal
   errInvalidDegParallelism
   errExecutionMechanismInvalid
   errExecutionMechanismLetFailed
   errStepLevelNull = 1020
   errStepLevelZeroForGlobal
   errStepLevelLetFailed
   errRowCountNumeric
   errConnNameInvalid
   errTimeoutNumeric
   errResetConnPropertiesFailed
   errFailureThresholdNumeric
   errConnectionUpdateFailed
   errGlobalRunMethodMandatory
   errGlobalRunMethodNull = 1030
   errGlobalRunMethodInvalid
   errGlobalRunMethodLetFailed
   errNoTrueOption
   errGetOptionFailed
   errSetEnabled
   errStepLabelTextAndFileNull
   errInvalidNodeType
   errSetOptionFailed
   errQueryParameterFailed
   errParamNotFound = 1040
   errQueryStepFailed
   errStepNotFound
   errReadFromScreenFailed
   errCopyPropertiesToFormFailed
   errMandatoryFieldNull
   errUnableToCheckNull
   errUpgradeFailed
   errFindStepSequenceFailed
   errFindParentStepIdFailed
   errCircularReference = 1050
   errAddStepFailed
   errModifyStepFailed
   errDeleteColumnFailed
   errFindPositionFailed
   errDeleteStepFailed
   errRunExistsForStepFailed
   errCreateNewParentFailed
   errInsertNewStepVersionFailed
   errCreateNewStepFailed
   errDuplicateParameterName = 1060
   errCheckDupParameterNameFailed
   errConstraintTypeInvalid
   errConstraintTypeLetFailed
   errAddConstraintFailed
   errWorkspaceIdMandatory
   errInvalidWorkspaceData
   errGetWorkspaceDetailsFailed
   errNoWorkspaceLoaded
   errWorkspaceAlreadyOpen
   errDuplicateWorkspaceName = 1070
   errWorkspaceNameMandatory
   errWorkspaceNameSetFailed
   errWorkspaceIdInvalid
   errWorkspaceIdSetFailed
   errWorkspaceInsertFailed
   errWorkspaceDeleteFailed
   errWorkspaceUpdateFailed
   errInvalidFile
   errCheckWorkspaceOpenFailed
   errWriteFailed = 1080
   errDeleteParameterRecordFailed
   errUnableToLogOutput
   errDeleteDBRecordFailed
   errRunExistsForWorkspaceFailed
   errClearHistoryFailed
   errCreateDBFailed
   errImportWspFailed
   errStepModifyFailed
   errStepDeleteFailed
   errDummy3 = 1090
   errUnableToGetWorkspace
   errInvalidNode
   errUnableToRemoveSubtree
   errSetFileNameFailed
   errStepTypeInvalid
   errObjectMandatory
   errBuiltInUpdateOnly
   errInvalidStep
   errTypeOfStepFailed
   errGetParentKeyFailed = 1100
   errLabelTextAndFileCheckFailed
   errStepTextAndFileNull
   errTextOrFileCheckFailed
   errParentStepManager
   errDeleteSubStepsFailed
   errWorkspaceNameDuplicateFailed
   errNewConstraintVersionFailed
   errDeleteStepConstraintsFailed
   errOldVersionMandatory
   errLoadConstraintsInListFailed = 1110
   errLoadGlobalStepsFailed
   errDeleteConstraintFailed
   errUpdateConstraintFailed
   errConstraintIdInvalid
   errConstraintIdSetFailed
   errGlobalStepIdInvalid
   errGlobalStepIdSetFailed
   errUpdateVersionFailed
   errQueryAdjacentConsFailed
   errConstraintNotFound = 1120
   errQueryConstraintFailed
```


© 2009 Hewlett-Packard Company. All rights reserved.

errSetDBBeforeLoad
errLoadDataFailed
errLoadRsInArrayFailed
errConstraintsForStepFailed
errPreConstraintsForStepFailed
errPostConstraintsForStepFailed
errExecuteConstraintMethodFailed
errIdOrKeyMandatory
errInListFailed = 1130
errUnableToWriteChanges
errQuickSortFailed
errCheckParentValidFailed
errLogErrorFailed
errCopyListFailed
errConnected
errVersionMismatch
errStepNodeFailed
errWorkspaceSelectedFailed
errIdentifierSelectedFailed = 1140
errCheckForNullFieldFailed
errInstanceInUse
errSetVisiblePropertyFailed
errExportWspFailed
errMakeKeyValidFailed
errDummy16
errRunApplicationFailed
errStepLabelUnique
errDeleteSingleFile
errMakeIdentifierValidFailed = 1150
errTypeOfNodeFailed
errConstraintCommandFailed
errOpenDbFailed
errInsertNewConstraintsFailed
errLoadPostExecutionStepsFailed
errLoadPreExecutionStepsFailed
errCreateNewNodeFailed
errDeleteNodeFailed
errDisplayPopupFailed
errDisplayPropertiesFailed = 1160
errUnableToCreateNewObject
errDiffFailed
errLoadWorkspaceFailed
errTerminateProcessFailed
errCompareFailed
errCreateConnectionFailed
errShowFormFailed
errAbortFailed
errDeleteParameterFailed
errUpdateViewFailed = 1170
errParameterNewFailed
errCopyNodeFailed
errCutNodeFailed
errCheckObjectValidFailed
errDeleteViewNodeFailed
errMainFailed
errNewStepFailed
errProcessStepModifyFailed
errCustomizeStepFormFailed
errInitializeStepFormFailed = 1180
errInsertStepFailed
errIncVersionYFailed
errIncVersionXFailed
errShowCreateStepFormFailed
errShowStepFormFailed
errStepNewFailed
errUnableToApplyChanges
errUnableToCommitChanges
errGetStepNodeTextFailed
errSelectGlobalRunMethodFailed = 1190
errConnectionNameMandatory
errUpdateStepFailed
errBrowseFailed
errDummy4
errDummy1
errDummy2

errDummy
errUnableToPreviewFile
errCopyWorkspaceFailed
errCopyParameterFailed = 1200
errGetStepTypeAndPositionFailed
errCopyStepFailed
errMandatoryParameterMissing
errDeleteWorkspaceRecordsFailed
errCreateDirectoryFailed
errConfirmDeleteOrMoveFailed
errCreateWorkspaceFailed
errTypeOfObjectFailed
errCreateNodeFailed
errCreateParameterFailed = 1210
errInsertParameterFailed
errCreateStepFailed
errNoConstraintsCreated
errCopyFailed
errCloneFailed
errCloneGlobalFailed
errCloneWorkerFailed
errCloneManagerFailed
errLetStepTypeFailed
errUnableToCloseWorkspace = 1220
errUnableToModifyWorkspace
errUnableToCreateWorkspace
errAddArrayElementFailed
errUpdateSequenceFailed
errCannotCopySubSteps
errSubStepsFailed
errModifyInArrayFailed
errUpdateParentVersionFailed
errGetNodeTextFailed
errAddToArrayFailed = 1230
errDeleteFromArrayFailed
errQueryIndexFailed
errCreateNewConstraintVersionFailed
errGetRootNodeFailed
errPopulateWspDetailsFailed
errLoadRsInTreeFailed
errAddNodeToTreeFailed
errMaxTempFiles
errMoveFailed
errRootNodeKeyInvalid = 1240
errNextNodeFailed
errBranchWillMove
errMoveBranchInvalid
errCreateIdRecordsetFailed
errIdentifierColumnFailed
errGetIdentifierFailed
errGetStepTypeFailed
errUpdateConstraintSeqFailed
errDelParamsInWspFailed
errDuplicateConnectionName = 1250
errOpenWorkspaceFailed
errShowWorkspaceNewFailed
errShowWorkspaceModifyFailed
errPopulateListFailed
errExploreNodeFailed
errInitializeListNodeFailed
errMakeListColumnsFailed
errRefreshViewFailed
errExploreFailed
errCollapseNodeFailed = 1260
errUnableToProcessListViewClick
errSetEnabledForStepFailed
errDisplayStepFormFailed
errSetEnabledPropertyFailed
errInvalidDB
errDeleteConnectionFailed
errInvalidOperation
errLetOperationFailed
errIdGetFailed
errCommitFailed = 1270
errSaveParametersInWspFailed

errDeleteArrayElementFailed
errSaveWorkspaceFailed
errInitializeFailed
errLoadInArrayFailed
errSaveStepsInWspFailed
errCommitStepFailed
errStepIdGetFailed
errUnloadFromArrayFailed
errValidateFailed = 1280
errTextEnteredFailed
errStepLabelMandatory
errTextAndFileNullForManager
errFailureDetailsNullForMgr
errSetTabOrderFailed
errSaveWspConstraintsFailed
errCommitConstraintFailed
errUnloadStepConstraintsFailed
errUnableToModifyMenu
errConfirmFailed = 1290
errInitSubItemsFailed
errUpdateListNodeFailed
errAddNodeFailed
errLoadListNodeFailed
errAddListNodeFailed
errExecutionFailed
errSetListViewStyleFailed
errSetCheckedFailed
errGetCheckedFailed
errUnableToProcessListViewDblClick = 1300
errDefaultPosition
errShellFailed
errOpenFileFailed
errSetTBar97Failed
errConnectFailed
errApiFailed
errRegEntryInvalid
errParseStringFailed
errConstraintsForWspFailed
errPostConstraintsForWspFailed = 1310
errPreConstraintsForWspFailed
errLoadWspPostExecStepsFailed
errLoadWspPreExecStepsFailed
errLoadConstraintsOnFormFailed
errQueryFailed
errPasteNodeFailed
errShowAllWorkspacesFailed
errMakeFieldValidFailed
errInitializeTree
errRootNodeFailed = 1320
errDirectionInvalid
errUnableToDetListProperty
errUnableToGetListData
errItemNotFound
errItemDoesNotExist
errParamNameInvalid
errGetParamValueFailed
errSubValuesFailed
errStringOpFailed
errReadWorkspaceDataFailed = 1330
errUpdateRunDataFailed
errProgramError
errUnableToOpenFile
errLoadRunDataFailed
errExecuteODBCCommandFailed
errRunWorkspaceFailed
errExecuteStepFailed
errUnableToWriteError
errRunStepFailed
errSaveChanges = 1340
errDragDropFailed
errInvalidParameter
errAssignParametersFailed
errLoadLabelsInTreeFailed
errInstrRFailed
errInsertIteratorFailed

© 2009 Hewlett-Packard Company. All rights reserved.

```
    errDeleteIteratorFailed
    errTypeInvalid
    errLoadFailed
    errDeleteFailed = 1350
    errModifyFailed
    errIteratorsFailed
    errInsertFailed
    errUpdateFailed
    errDuplicateIterator
    errSaveFailed
    errReadDataFailed
    errUnloadFailed
    errAddFailed
    errExecuteBranchFailed = 1360
    errRangeNumeric
    errRangeInvalid
    errNextStepFailed
    errUpdateDisplayFailed
    errDateToStringFailed
    errGetElapsedTimeFailed
    errMaxProcessesExceeded
    errInvalidProperty
    errInvalidChild
    errCreateInstanceFailed = 1370
    errInvalidForWorker
    errInstanceOpFailed
    errNavInstancesFailed
    errIterateFailed
    errExecInstanceFailed
    errDupIterator
End Enum
Attribute VB_Name = "SortSM"
'  FILE:       SortSM.bas
'            Microsoft TPC-H Kit Ver. 1.00
'            Copyright Microsoft, 1999
'            All Rights Reserved
'
'
'  PURPOSE:    This module contains an
implementation of QuickSort.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

' Comment out for case-sensitive sorts
Option Compare Text

' Used to indicate the source module name
when errors
' are raised by this module
Private Const mstrModuleName As String =
"SortSM."

Private Function Compare(ByVal
vntToCompare1 As Variant, _
        ByVal vntToCompare2 As Variant) As
Integer

    On Error GoTo CompareErr

    Compare = 0

    If vntToCompare1.SequenceNo <
vntToCompare2.SequenceNo Then
        Compare = -1
    ElseIf vntToCompare1.SequenceNo >
vntToCompare2.SequenceNo Then
        Compare = 1
    End If

    Exit Function

CompareErr:
    ' Log the error code raised by Visual Basic
    Call LogErrors(Errors)
    On Error GoTo 0
    Err.Raise vbObjectError +
errCompareFailed, _
            gstrSource, _
            LoadResString(errCompareFailed)

End Function

Private Sub Swap(ByRef vntToSwap1 As
Variant, _
        ByRef vntToSwap2 As Variant)

    Dim vntTemp As Variant

    On Error GoTo SwapErr

    If IsObject(vntToSwap1) And
IsObject(vntToSwap2) Then
        Set vntTemp = vntToSwap1
        Set vntToSwap1 = vntToSwap2
        Set vntToSwap2 = vntTemp
    Else
        vntTemp = vntToSwap1
        vntToSwap1 = vntToSwap2
        vntToSwap2 = vntTemp
    End If

    Exit Sub

SwapErr:
    ' Log the error code raised by Visual Basic
    Call LogErrors(Errors)
    On Error GoTo 0
    Err.Raise vbObjectError +
errQuickSortFailed, mstrModuleName &
"Swap", _
        LoadResString(errQuickSortFailed)

End Sub

Public Sub QuickSort(vntArray As Variant, _
        Optional ByVal intLBound As Integer, _
        Optional ByVal intUBound As Integer)
    ' Sorts a variant array using Quicksort

    Dim i As Integer
    Dim j As Integer
    Dim vntMid As Variant

    On Error GoTo QuickSortErr

    If IsEmpty(vntArray) Or _
            Not IsArray(vntArray) Then
        Exit Sub
    End If

    ' Set default boundary values for first time
through
    If intLBound = 0 And intUBound = 0 Then
        intLBound = LBound(vntArray)
        intUBound = UBound(vntArray)
    End If

    ' BugMessage "Sorting elements " &
Str(intLBound) & " and " & Str(intUBound)

    If intLBound > intUBound Then
        Exit Sub
    End If

    ' Only two elements in this subdivision;
exchange if they
    ' are out of order and end recursive calls
    If (intUBound - intLBound) = 1 Then
        If Compare(vntArray(intLBound),
vntArray(intUBound)) > 0 Then
            Call Swap(vntArray(intLBound),
vntArray(intUBound))
        End If
        Exit Sub
    End If

    ' Set the pivot point
    Set vntMid = vntArray(intUBound)
    i = intLBound
    j = intUBound

    Do
        ' Move in from both sides towards pivot
element
        Do While (i < j) And Compare(vntArray(i),
vntMid) <= 0
            i = i + 1
        Loop

        Do While (j > i) And Compare(vntArray(j),
vntMid) >= 0
            j = j - 1
        Loop

        If i < j Then
            Call Swap(vntArray(i), vntArray(j))
        End If
    Loop While i < j

    ' Since i has been adjusted, swap element i with
element,
    ' intUBound
    Call Swap(vntArray(i), vntArray(intUBound))

    ' Recursively call sort array - pass smaller
subdivision
    ' first to conserve stack space
    If (i - intLBound) < (intUBound - 1) Then
        ' Recursively sort with adjusted values for
upper and
        ' lower bounds
        Call QuickSort(vntArray, intLBound, i - 1)
        Call QuickSort(vntArray, i + 1, intUBound)
    Else
        Call QuickSort(vntArray, i + 1, intUBound)
        Call QuickSort(vntArray, intLBound, i - 1)
    End If
    Exit Sub

QuickSortErr:
    Call LogErrors(Errors)
    On Error GoTo 0
    Err.Raise vbObjectError + errQuickSortFailed,
mstrModuleName & "QuickSort", _
        LoadResString(errQuickSortFailed)

End Sub
Attribute VB_Name = "Startup"
'  FILE:       Startup.bas
'            Microsoft TPC-H Kit Ver. 1.00
'            Copyright Microsoft, 1999
'            All Rights Reserved
'
'
'  PURPOSE:    This module contains startup and
cleanup functions for the project.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

Public Const LISTVIEW_BUTTON = 14
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
Public gstrProjectPath As String

' Used to indicate the source module name
when errors
' are raised by this module
Private Const mstrModuleName As String =
"Startup."

Private Sub Initialize()

   On Error GoTo InitializeErr

   ReDim
gsContCriteria(gintOnFailureAbort To
gintOnFailureAsk) As String
   gsContCriteria(gintOnFailureAbort) =
"Abort"
   gsContCriteria(gintOnFailureContinue) =
"Continue"

gsContCriteria(gintOnFailureCompleteSibli
ngs) = "Execute sibling steps and stop"

gsContCriteria(gintOnFailureAbortSiblings)
= "Abort sibling steps and execute next
parent"

gsContCriteria(gintOnFailureSkipSiblings)
= "Skip sibling steps and execute next
parent"
   gsContCriteria(gintOnFailureAsk) =
"Ask"

   ReDim gsExecutionStatus(gintDisabled
To gintAborted) As String
   gsExecutionStatus(gintDisabled) =
"Disabled"
   gsExecutionStatus(gintPending) =
"Pending"
   gsExecutionStatus(gintRunning) =
"Running"
   gsExecutionStatus(gintComplete) =
"Complete"
   gsExecutionStatus(gintFailed) = "Failed"
   gsExecutionStatus(gintAborted) =
"Stopped"

#If Not RUN_ONLY Then
   ' Call a procedure to change the style of
the toolbar
   ' on the Step Properties form
   Call
SetTBar97(frmSteps.tblConstraintCommand
s)

#End If

   Call InitRunEngine

   Exit Sub

InitializeErr:
   ' Log the error code raised by Visual
Basic
   Call LogErrors(Errors)
   gstrSource = mstrModuleName &
"Initialize"
   Call ShowError(errInitializeFailed)

End Sub
Sub Main()

   On Error GoTo MainErr

   ' Mousepointer should indicate busy
   Call ShowBusy

   ' Display the Splash screen while we carry
out some initialization
   frmSplash.Show
   frmSplash.Refresh

   gstrProjectPath = App.Path

   ' Open the database
   If OpenDBFile() = False Then
      Unload frmSplash
      Exit Sub
   End If

#If Not RUN_ONLY Then
   Load frmMain

   ' Enable the Stop Run menu options only
when a workspace is
   ' actually running
   Call EnableStop(False)

   ' Clear all application extension menu items
   Call ClearToolsMenu
#End If

   Call Initialize

   ' Mousepointer - ready to accept user input
   Call ShowFree

   ' Unload the Splash screen and display the
main form
   Unload frmSplash

#If RUN_ONLY Then
   frmWorkspaceOpen.Caption =
gsCaptionRunWsp

   Call ShowWorkspacesInDb(dbsAttTool)
#Else
   frmMain.Show
#End If

   Exit Sub

MainErr:
   ' Log the error code raised by Visual Basic
   Call LogErrors(Errors)
   Call ShowFree
   Call ShowError(errMainFailed)

End Sub
Private Function OpenDBFile() As Boolean
   Dim sDb As String

   On Error GoTo OpenDBFileErr

#If RUN_ONLY Then
   ' Always use the registry setting for the
run_only mode
   sDb = DefaultDBFile()
#Else
   ' Check if the user has specified the
workspace defn. file to open on the command
line
   ' Else, use the registry setting
   sDb = IIf(StringEmpty(Command),
DefaultDBFile(), Command)

   If Len(sDb) > 0 Then
      ' Trim off the enclosing double-quotes if
any
      If Mid(sDb, 1, 1) = gstrDQ Then
         If Len(sDb) > 1 Then
            sDb = Mid(sDb, 2)
         Else
            sDb = gstrEmptyString
         End If
      End If
   End If

   If Len(sDb) > 0 Then
      If Mid(sDb, Len(sDb), 1) = gstrDQ Then
         If Len(sDb) > 1 Then
            sDb = Mid(sDb, 1, Len(sDb) - 1)
         Else
            sDb = gstrEmptyString
         End If
      End If
   End If
#End If

   ' Open the database
   OpenDBFile = SMOpenDatabase(sDb)

   Exit Function

OpenDBFileErr:
   Call LogErrors(Errors)
   OpenDBFile = False

End Function
Public Sub Cleanup()

   On Error GoTo CleanupErr

   ' Set the mousepointer to indicate Busy
   Call ShowBusy

#If Not RUN_ONLY Then
   ' Close all open workspaces - will also prompt for
unsaved
   ' changes
   Call CloseOpenWorkspaces
#End If

   ' Close all open files
   Call CloseOpenFiles

   ' Reset the mousepointer
   Call ShowFree

   Exit Sub

CleanupErr:
   ' Log the error code raised by Visual Basic
   Call LogErrors(Errors)
   Resume Next

End Sub
Attribute VB_Name = "StepCommon"
'  FILE:      StepCommon.bas
'            Microsoft TPC-H Kit Ver. 1.00
'            Copyright Microsoft, 1999
'            All Rights Reserved
'
'
'  PURPOSE:    Contains functionality common
across StepMaster and
'            SMRunOnly, pertaining to steps
'            Specifically, functions to load iterators
records
'            in an array, determine the type of step, etc.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
' Used to indicate the source module name
when errors
' are raised by this module
Private Const mstrModuleName As String =
"StepCommon."

' Step property constants
Private Const mintMinFailureThreshold As
Integer = 1
Public Const gintMinSequenceNo As
Integer = 1
Public Const gintMinLevel As Integer = 0
Public Function
ValidateParallelism(sParallelism As String,
lWorkspace As Long, _
      Optional ParamsInWsp As
cArrParameters = Nothing) As String
   ' Returns the degree of parallelism for the
step if the user input is valid
   Dim sTemp As String

   On Error GoTo ValidateParallelismErr
   gstrSource = mstrModuleName &
"ValidateParallelism"

   sTemp =
SubstituteParameters(Trim$(sParallelism),
lWorkspace,
WspParameters:=ParamsInWsp)

   If Not IsNumeric(sTemp) Then
      ShowError errInvalidDegParallelism
      On Error GoTo 0
      Err.Raise vbObjectError +
errInvalidDegParallelism, gstrSource, _

LoadResString(errInvalidDegParallelism)
   Else
      If (CInt(sTemp) < gintMinParallelism)
Or (CInt(sTemp) > gintMaxParallelism)
Then
         ShowError errInvalidDegParallelism
         On Error GoTo 0
         Err.Raise vbObjectError +
errInvalidDegParallelism, gstrSource, _

LoadResString(errInvalidDegParallelism)
      Else
         ValidateParallelism =
Trim$(sParallelism)
      End If
   End If

   Exit Function

ValidateParallelismErr:
   ' Log the error code raised by Visual
Basic
   gstrSource = mstrModuleName &
"ValidateParallelism"
   If Err.Number = vbObjectError +
errSubValuesFailed Then
      ShowError errInvalidDegParallelism
   End If

   Call LogErrors(Errors)
   On Error GoTo 0
   Err.Raise vbObjectError +
errInvalidDegParallelism, gstrSource, _

LoadResString(errInvalidDegParallelism)

End Function
Public Function IsGlobal( _
      Optional ByVal StepClass As cStep =
Nothing, _
      Optional ByVal StepRecord As Recordset
= Nothing, _
      Optional ByVal StepKey As String =
gstrEmptyString, _
      Optional ByVal StepId As Long = 0, _
      Optional StepForm As Form = Nothing)
As Boolean

   ' This function contains all the possible
checks for whether
   ' a step is global - The check that will be
made depends on
   ' the parameter passed in

   Dim cStepRecord As cStep

   If Not StepClass Is Nothing Then
      IsGlobal = StepClass.GlobalFlag
      Exit Function
   End If

   If Not StepRecord Is Nothing Then
      IsGlobal = StepRecord![global_flag]
      Exit Function
   End If

   If Not StringEmpty(StepKey) Then
      IsGlobal = InStr(StepKey,
gstrGlobalStepPrefix) > 0
      Exit Function
   End If

   If StepId <> 0 Then
      Set cStepRecord =
gcSteps.QueryStep(StepId)
      IsGlobal = cStepRecord.GlobalFlag
      Set cStepRecord = Nothing
      Exit Function
   End If

   If Not StepForm Is Nothing Then
      IsGlobal =
(StepForm.lblStepType.Caption =
Str(gintGlobalStep))
      Exit Function
   End If

   ' Not a single object was passed in! - raise an
error
   On Error GoTo 0
   Err.Raise vbObjectError +
errObjectMandatory, _
         mstrModuleName & "IsGlobal", _
         LoadResString(errObjectMandatory)

End Function
Public Function TypeOfStep(Optional ByVal
StepClass As cStep = Nothing, _
      Optional ByVal StepRecord As Recordset
= Nothing, _
      Optional ByVal StepKey As String =
gstrEmptyString, _
      Optional ByVal StepId As Long = 0, _
      Optional StepForm As Form = Nothing)
As Integer
   ' Calls functions to determine the type of
step
   ' The check that will be made depends on the
parameter passed in

   On Error GoTo TypeOfStepErr

   ' Make the check whether a step is global first -
both
   ' worker and global steps have the step text or file
name
   ' not null - but only the global step will have the
global
   ' flag set
   If IsGlobal(StepClass, StepRecord, StepKey,
StepId, StepForm) Then
      TypeOfStep = gintGlobalStep
   ElseIf IsManager(StepClass, StepRecord,
StepKey, StepId, StepForm) Then
      TypeOfStep = gintManagerStep
   ElseIf IsWorker(StepClass, StepRecord,
StepKey, StepId, StepForm) Then
      TypeOfStep = gintWorkerStep
   Else
      On Error GoTo 0
      Err.Raise vbObjectError + errInvalidStep, _
            mstrModuleName & "TypeOfStep", _
            LoadResString(errInvalidStep)
   End If

   Exit Function

TypeOfStepErr:
   ' Log the error code raised by Visual Basic
   Call LogErrors(Errors)
   On Error GoTo 0
   Err.Raise vbObjectError + errTypeOfStepFailed,
_
         mstrModuleName & "TypeOfStep", _
         LoadResString(errTypeOfStepFailed)

End Function

Public Function IsStep(intNodeType As Integer) As
Boolean
   ' Returns true if the node type corresponds to a
global, manager
   ' or worker step
   IsStep = (intNodeType = gintGlobalStep) Or
(intNodeType = gintManagerStep) Or _
      (intNodeType = gintWorkerStep)

End Function

Public Function IsManager(Optional ByVal
StepClass As cStep = Nothing, _
      Optional ByVal StepRecord As Recordset =
Nothing, _
      Optional ByVal StepKey As String =
gstrEmptyString, _
      Optional ByVal StepId As Long = 0, _
      Optional StepForm As Form = Nothing) As
Boolean

   ' This function contains all the possible checks
for whether
   ' a step is a manager step - The check that will be
made depends
   ' on the parameter passed in

   Dim cStepRecord As cStep

   If Not StepClass Is Nothing Then
      IsManager = (StepClass.StepType =
gintManagerStep)
      Exit Function
   End If

   If Not StepRecord Is Nothing Then
      IsManager = (IsNull(StepRecord![step_text])
And IsNull(StepRecord![step_file_name]))
      Exit Function
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
    End If

    If Not StringEmpty(StepKey) Then
        IsManager = (InStr(StepKey,
gstrManagerStepPrefix) > 0)
        Exit Function
    End If

    If StepId <> 0 Then
        Set cStepRecord =
gcSteps.QueryStep(StepId)
        IsManager = (cStepRecord.StepType =
gintManagerStep)
        Set cStepRecord = Nothing
        Exit Function
    End If

    If Not StepForm Is Nothing Then
        IsManager =
(StepForm.lblStepType.Caption =
Str(gintManagerStep))
        Exit Function
    End If

    ' Not a single object was passed in! - raise
an error
    On Error GoTo 0
    Err.Raise vbObjectError +
errObjectMandatory, _
            "Step.IsManager", _
            LoadResString(errObjectMandatory)

End Function
Public Function IsWorker( _
        Optional ByVal StepClass As cStep =
Nothing, _
        Optional ByVal StepRecord As
Recordset = Nothing, _
        Optional ByVal StepKey As String =
gstrEmptyString, _
        Optional ByVal StepId As Long = 0, _
        Optional StepForm As Form =
Nothing) As Boolean

    ' This function contains all the possible
checks for whether
    ' a step is a Worker step - The check that
will be made depends
    ' on the parameter passed in

    Dim cStepRecord As cStep

    If Not StepClass Is Nothing Then
        IsWorker = (StepClass.StepType =
gintWorkerStep)
        Exit Function
    End If

    If Not StepRecord Is Nothing Then
        IsWorker = (Not
StepRecord![global_flag] And _
                (Not
IsNull(StepRecord![step_text]) Or Not
IsNull(StepRecord![step_file_name])))
        Exit Function
    End If

    If Not StringEmpty(StepKey) Then
        IsWorker = InStr(StepKey,
gstrWorkerStepPrefix) > 0
        Exit Function
    End If

    If StepId <> 0 Then
        Set cStepRecord =
gcSteps.QueryStep(StepId)
        IsWorker = (cStepRecord.StepType =
gintWorkerStep)
        Set cStepRecord = Nothing
        Exit Function
    End If

    If Not StepForm Is Nothing Then
        IsWorker =
(StepForm.lblStepType.Caption =
Str(gintWorkerStep))
        Exit Function
    End If

    ' Not a single object was passed in! - raise an
error
    On Error GoTo 0
    Err.Raise vbObjectError +
errObjectMandatory, _
            "Step.IsWorker", _
            LoadResString(errObjectMandatory)

End Function
Public Function GetStepNodeText(ByVal
cStepNode As cStep) As String

    On Error GoTo GetStepNodeTextErr

    ' Returns the string that will be displayed as
the text
    ' in the tree view node to the user
    If StringEmpty(cStepNode.StepLabel) Then

        If StringEmpty(cStepNode.StepTextFile)
Then

            If StringEmpty(cStepNode.StepText)
Then
                ' This should never happen
                On Error GoTo 0
                Err.Raise vbObjectError +
errStepLabelTextAndFileNull, _
                    gstrSource, _

LoadResString(errStepLabelTextAndFileNull)
            Else
                GetStepNodeText =
cStepNode.StepText
            End If
        Else
            GetStepNodeText =
cStepNode.StepTextFile
        End If
    Else
        GetStepNodeText = cStepNode.StepLabel
    End If

    Exit Function

GetStepNodeTextErr:
    ' Log the error code raised by Visual Basic
    Call LogErrors(Errors)
    On Error GoTo 0
    Err.Raise vbObjectError +
errGetStepNodeTextFailed, _
            gstrSource, _

LoadResString(errGetStepNodeTextFailed)

End Function

Public Function
LoadRecordsetInStepsArray(rstSteps As
Recordset, _
    cStepCol As cArrSteps) As Boolean

    Dim cNewStep As cStep
    Dim cNewGlobal As cGlobalStep
    Dim cNewManager As cManager
    Dim cNewWorker As cWorker

    On Error GoTo LoadRecordsetInStepsArrayErr

    If rstSteps.RecordCount = 0 Then
        Exit Function
    End If

    rstSteps.MoveFirst
    While Not rstSteps.EOF
        ' For fields that should not be null, a procedure
is first
        ' called to raise an error if the field is null

        Set cNewStep = New cStep

        cNewStep.StepType =
TypeOfStep(StepRecord:=rstSteps)

        If cNewStep.StepType = gintGlobalStep Then
            Set cNewGlobal = New cGlobalStep
            Set cNewStep = cNewGlobal
        ElseIf cNewStep.StepType = gintManagerStep
Then
            Set cNewManager = New cManager
            Set cNewStep = cNewManager
        Else
            Set cNewWorker = New cWorker
            Set cNewStep = cNewWorker
        End If

        ' Initialize the global flag first, since
subsequent
        ' validations might depend on whether the step
is global
        cNewStep.GlobalFlag =
CBool(ErrorOnNullField(rstSteps, "global_flag"))

        ' Initialize step values
        cNewStep.StepId =
CLng(ErrorOnNullField(rstSteps, "step_id"))
        cNewStep.VersionNo =
CStr(ErrorOnNullField(rstSteps, "version_no"))

        cNewStep.StepLabel =
CheckForNullField(rstSteps, "step_label")
        cNewStep.StepTextFile =
CheckForNullField(rstSteps, "step_file_name")
        cNewStep.StepText =
CheckForNullField(rstSteps, "step_text")
        cNewStep.StartDir =
CheckForNullField(rstSteps, "start_directory")

        cNewStep.WorkspaceId =
CLng(ErrorOnNullField(rstSteps,
FLD_ID_WORKSPACE))
        cNewStep.ParentStepId =
CLng(ErrorOnNullField(rstSteps,
"parent_step_id"))
        cNewStep.ParentVersionNo =
CStr(ErrorOnNullField(rstSteps,
"parent_version_no"))

        cNewStep.SequenceNo =
CInt(ErrorOnNullField(rstSteps, "sequence_no"))
        cNewStep.StepLevel =
CInt(ErrorOnNullField(rstSteps, "step_level"))
        cNewStep.EnabledFlag =
CBool(ErrorOnNullField(rstSteps, "enabled_flag"))
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
        ' Initialize the execution details for the
step
        cNewStep.DegreeParallelism =
CheckForNullField(rstSteps,
"degree_parallelism")
        cNewStep.ExecutionMechanism =
CInt(ErrorOnNullField(rstSteps,
"execution_mechanism"))
        cNewStep.FailureDetails =
CheckForNullField(rstSteps,
"failure_details")
        cNewStep.ContinuationCriteria =
CInt(ErrorOnNullField(rstSteps,
"continuation_criteria"))

        ' Initialize the output file locations for
the step
        cNewStep.OutputFile =
CheckForNullField(rstSteps,
"output_file_name")
        ' cNewStep.LogFile =
CheckForNullField(rstSteps,
"log_file_name")
        cNewStep.ErrorFile =
CheckForNullField(rstSteps,
"error_file_name")

        ' Initialize the iterator name for the
step, if any
        cNewStep.IteratorName =
CheckForNullField(rstSteps,
"iterator_name")

        ' Add this record to the array of steps
        cStepCol.Load cNewStep

        Set cNewStep = Nothing
        rstSteps.MoveNext
    Wend

    Exit Function

LoadRecordsetInStepsArrayErr:

    LogErrors Errors
    gstrSource = mstrModuleName &
"LoadRecordsetInStepsArray"
    On Error GoTo 0
    Err.Raise vbObjectError +
errLoadRsInArrayFailed, gstrSource, _

LoadResString(errLoadRsInArrayFailed)

End Function


Attribute VB_Name = "TimerSM"
'  FILE:      TimerSM.bas
'             Microsoft TPC-H Kit Ver. 1.00
'             Copyright Microsoft, 1999
'             All Rights Reserved
'
'
'  PURPOSE:    This module contains
wrapper functions for Timer APIs.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'

Option Explicit

Private Declare Function SetTimer Lib
"user32" (ByVal hWnd As Long, _
    ByVal nIDEvent As Long, ByVal uElapse
As Long, ByVal lpTimerFunc As Long) _
    As Long
Private Declare Function KillTimer Lib
"user32" (ByVal hWnd As Long, _
    ByVal nIDEvent As Long) As Long
Private Declare Sub CopyMemory Lib
"kernel32" Alias "RtlMoveMemory" ( _
    pDest As Any, pSource As Any, ByVal
ByteLen As Long)

Public gcTimerObjects As Collection

Private Sub TimerProc(ByVal lHwnd As Long,
ByVal lMsg As Long, _
    ByVal lTimerID As Long, ByVal lTime As
Long)

    Dim nPtr As Long
    Dim oTimerObject As cTimerSM

    'Create a Timer object from the pointer
    nPtr = gcTimerObjects.Item(Str$(lTimerID))
    CopyMemory oTimerObject, nPtr, 4
    'Call a method which will fire the Timer
event
    oTimerObject.Tick
    'Get rid of the Timer object so that VB will
not try to release it
    CopyMemory oTimerObject, 0&, 4
End Sub

Public Function StartTimer(lInterval As Long)
As Long
    StartTimer = SetTimer(0, 0, lInterval,
AddressOf TimerProc)
End Function

Public Sub StopTimer(lTimerID As Long)
    KillTimer 0, lTimerID
End Sub

Public Sub SetInterval(lInterval As Long,
lTimerID As Long)
    SetTimer 0, lTimerID, lInterval, AddressOf
TimerProc
End Sub
Attribute VB_Name = "ToolsCommon"
'  FILE:      ToolsCommon.bas
'             Microsoft TPC-H Kit Ver. 1.00
'             Copyright Microsoft, 1999
'             All Rights Reserved
'
'
'  PURPOSE:    Contains functions to remove
run history and initialize
'              table creation scripts
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

Public sCreateTables() As String

Public Const gsExtSeparator As String = "."

Public Sub DeleteRunHistory(dbFile As
DAO.Database)
    ' Delete all run history records from the
database, viz. the records in
    ' run_header, run_step_details and
run_parameters

    Dim sDelete As String

    On Error GoTo DeleteRunHistoryErr

    sDelete = "delete from run_header "
    dbFile.Execute sDelete, dbFailOnError

    sDelete = "delete from run_step_details "
    dbFile.Execute sDelete, dbFailOnError

    sDelete = "delete from run_parameters "
    dbFile.Execute sDelete, dbFailOnError

    sDelete = "update att_identifiers " & _
        " set run_id = " & CStr(glMinId)
    dbFile.Execute sDelete, dbFailOnError

    Exit Sub

DeleteRunHistoryErr:
    LogErrors Errors
    Err.Raise vbObjectError +
errDeleteDBRecordFailed, "DeleteRunHistory", _
          LoadResString(errDeleteDBRecordFailed)

End Sub

Public Function CreateConnectionsTableScript() As
String
    ' Returns the table creation script for the
workspace_connections table

    Call InitCreateSQLArray
    CreateConnectionsTableScript =
sCreateTables(10)
    ReDim sCreateTables(0)

End Function
Public Function CreateConnectionDtlsTableScript()
As String
    ' Returns the table creation script for the
connection_dtls table

    Call InitCreateSQLArray
    CreateConnectionDtlsTableScript =
sCreateTables(11)
    ReDim sCreateTables(0)

End Function

Public Sub InitCreateSQLArray()

ReDim sCreateTables(0 To 11)

sCreateTables(0) = "Create table att_identifiers (" &
_
        "workspace_id            Long, " & _
        "parameter_id            Long, " & _
        "step_id                 Long, " & _
        "constraint_id           Long, " & _
        "run_id                  Long, " & _
        "connection_id           Long " & _
        ", " & FLD_ID_CONN_NAME & gstrBlank
& DATA_TYPE_LONG & _
");"

sCreateTables(1) = "Create table att_steps (step_id
Long, " & _
        "version_no              Text(255), " & _
        "step_label              Text(255), " & _
        "step_file_name          Text(255), " & _
        "step_text               Memo, " & _
        "start_directory         Text(255), " & _
        "workspace_id            Long, " & _
        "parent_step_id          Long, " & _
        "parent_version_no       Text(255), " & _
        "step_level              Long, " & _
        "sequence_no             Integer, " & _
        "enabled_flag            Bit, " & _
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
        "degree_parallelism        Text(255), "
& _
        "execution_mechanism       Text(50),
" & _
        "failure_details           Text(255), " &
_
        "continuation_criteria     Text(50), " &
_
        "global_flag               Long, " & _
        "archived_flag             Bit, " & _
        "output_file_name          Text(255), "
& _
        "error_file_name           Text(255), "
& _
        "iterator_name             Text(255), " &
_
        "CONSTRAINT pk_steps PRIMARY
KEY (step_id, version_no) " & _
");"

'       "log_file_name             Text(255), "
& _

sCreateTables(2) = "Create table
att_workspaces (" & _
        "workspace_id              Long, " & _
        "workspace_name            Text(255), "
& _
        "archived_flag             Bit, " & _
        "CONSTRAINT pk_workspaces
PRIMARY KEY (workspace_id) " & _
");"

sCreateTables(3) = "Create table
iterator_values (" & _
        "step_id                   Long, " & _
        "version_no                Text(255), " &
_
        "type                      Integer, " & _
        "iterator_value            Text(255), " &
_
        "sequence_no               Integer " & _
");"

sCreateTables(4) = "Create table run_header
(" & _
        "run_id                    Long, " & _
        "workspace_id              Long, " & _
        "start_time                Currency, " & _
        "end_time                  Currency, " & _
        "CONSTRAINT pk_run_header
PRIMARY KEY (run_id) " & _
");"

sCreateTables(5) = "Create table
run_parameters (" & _
        "run_id                    Long, " & _
        "parameter_name            Text(255), "
& _
        "parameter_value           Text(255) "
& _
");"

sCreateTables(6) = "Create table
run_step_details (" & _
        "run_id                    Long, " & _
        "step_id                   Long, " & _
        "version_no                Text(255), " &
_
        "instance_id               Long, " & _
        "parent_instance_id        Long, " & _
        "command                   Memo, " & _
        "iterator_value            Text(255), " &
_
        "start_time                Currency, " & _
        "end_time                  Currency, " & _
        "elapsed_time              Long " & _
");"

sCreateTables(7) = "Create table
step_constraints (" & _
        "constraint_id             Long, " & _
        "step_id                   Long, " & _
        "version_no                Text(255), " & _
        "constraint_type           Integer, " & _
        "global_step_id            Long, " & _
        "global_version_no         Text(255), " &
_
        "sequence_no               Integer " & _
");"

sCreateTables(8) = "Create table
workspace_parameters (" & _
        "workspace_id              Long, " & _
        "parameter_id              Long, " & _
        "parameter_name            Text(255), " &
_
        "parameter_value           Text(255), " & _
        "description               Text(255), " & _
        "parameter_type            Integer, " & _
        "CONSTRAINT pk_parameters
PRIMARY KEY (parameter_id) " & _
");"

sCreateTables(9) = "Create table db_details ("
& _
        "db_version                Text(50) " & _
");"

sCreateTables(10) = "Create table " &
TBL_CONNECTION_STRINGS & " (" & _
        "workspace_id              Long, " & _
        "connection_id             Long, " & _
        "connection_name           Text(255), " &
_
        "connection_value          Text(255), " & _
        "description               Text(255), " & _
        "no_count_display          Bit, " & _
        "no_execute                Bit, " & _
        "parse_query_only          Bit, " & _
        "ANSI_quoted_identifiers   Bit, " & _
        "ANSI_nulls                Bit, " & _
        "show_query_plan           Bit, " & _
        "show_stats_time           Bit, " & _
        "show_stats_io             Bit, " & _
        "parse_odbc_msg_prefixes   Bit, " & _
        "row_count                 long, " & _
        "tsql_batch_separator      Text(255), " &
_
        "query_time_out            long, " & _
        "server_language           Text(255), " & _
        "character_translation     Bit, " & _
        "regional_settings         Bit, " & _
        "CONSTRAINT pk_connections
PRIMARY KEY (connection_id) " & _
");"

' This table has been added in order to satisfy
the TPC-H requirement that
' all the queries in a stream need to be executed
on a single connection.
' Specify a connection for each odbc step. If the
connection is of type,
' static, it should be kept open till the step
execution is complete.
sCreateTables(11) = "Create table " &
TBL_CONNECTION_DTLS & " (" & _
        FLD_ID_WORKSPACE & gstrBlank &
DATA_TYPE_LONG & ", " & _
        FLD_ID_CONN_NAME & gstrBlank &
DATA_TYPE_LONG & ", " & _
        FLD_CONN_DTL_CONNECTION_NAME
& gstrBlank & DATA_TYPE_TEXT255 & ", " & _

FLD_CONN_DTL_CONNECTION_STRING &
gstrBlank & DATA_TYPE_TEXT255 & ", " & _
        FLD_CONN_DTL_CONNECTION_TYPE &
gstrBlank & DATA_TYPE_INTEGER & ", " & _
        "CONSTRAINT pk_connection_name
PRIMARY KEY (" & FLD_ID_CONN_NAME &
") " & _
");"

End Sub
Attribute VB_Name = "WindowsApiCommon"
'  FILE:      WindowsApiCommon.bas
'             Microsoft TPC-H Kit Ver. 1.00
'             Copyright Microsoft, 1999
'             All Rights Reserved
'
'
'  PURPOSE:   This module contains functions that
are wrappers around the
'             Windows API and are used by both
StepMaster and SMRunOnly.
'  Contact:   Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

' Used to indicate the source module name when
errors
' are raised by this module
Private Const mstrModuleName As String =
"WindowsApiCommon."

Public Type PROCESS_INFORMATION
   hProcess    As Long
   hThread     As Long
   dwProcessID As Long
   dwThreadID  As Long
End Type

' Used by GetShortName to return the short file
name for a given file
Private Declare Function GetShortPathName Lib
"kernel32" _
     Alias "GetShortPathNameA" (ByVal
lpszLongPath As String, _
     ByVal lpszShortPath As String, ByVal
cchBuffer As Long) As Long

Public Declare Function GetExitCodeProcess Lib
"kernel32" ( _
   ByVal hProcess As Long, lpExitCode As Long)
As Long
Public Declare Function TerminateProcess Lib
"kernel32" ( _
      hProcess As Long, uExitCode As Long) As
Long
Public Declare Function CloseHandle Lib
"kernel32" ( _
   ByVal hObject As Long) As Long

Public Const NORMAL_PRIORITY_CLASS
As Long = &H20&
Public Const INFINITE                As Long = -1&

Public Const STATUS_WAIT_0           As Long
= &H0
Public Const STATUS_ABANDONED_WAIT_0
As Long = &H80
Public Const STATUS_USER_APC         As
Long = &HC0
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
Public Const STATUS_TIMEOUT
As Long = &H102
Public Const STATUS_PENDING
As Long = &H103

Public Const WAIT_FAILED              As
Long = &HFFFFFFFF
Public Const WAIT_OBJECT_0            As
Long = STATUS_WAIT_0
Public Const WAIT_TIMEOUT             As
Long = STATUS_TIMEOUT

Public Const WAIT_ABANDONED
As Long =
STATUS_ABANDONED_WAIT_0
Public Const WAIT_ABANDONED_0
As Long =
STATUS_ABANDONED_WAIT_0

Public Const WAIT_IO_COMPLETION
As Long = STATUS_USER_APC
Public Const STILL_ACTIVE             As
Long = STATUS_PENDING

Public Const
PROCESS_QUERY_INFORMATION  As
Long = &H400
Public Const
STANDARD_RIGHTS_REQUIRED   As
Long = &HF0000

'-------------------------------------------------------
--------------------------
'Declarations for shelling:

Public Type STARTUPINFO
   cb             As Long
   lpReserved     As String
   lpDesktop      As String
   lpTitle        As String
   dwX            As Long
   dwY            As Long
   dwXSize        As Long
   dwYSize        As Long
   dwXCountChars  As Long
   dwYCountChars  As Long
   dwFillAttribute As Long
   dwFlags        As Long
   wShowWindow    As Integer
   cbReserved2    As Integer
   lpReserved2    As Long
   hStdInput      As Long
   hStdOutput     As Long
   hStdError      As Long
End Type

Public Declare Function
WaitForSingleObject Lib "kernel32" ( _
   ByVal hProcess As Long, ByVal
dwMilliseconds As Long) As Long

Public Declare Function InputIdle Lib
"user32" Alias "WaitForInputIdle" ( _
   ByVal hProcess As Long, ByVal
dwMilliseconds As Long) As Long

Public Declare Function CreateProcessA Lib
"kernel32" ( _
   ByVal lpApplicationName As Long,
ByVal lpCommandLine As String, _
   ByVal lpProcessAttributes As Long,
ByVal lpThreadAttributes As Long, _
   ByVal bInheritHandles As Long, ByVal
dwCreationFlags As Long, _
   ByVal lpEnvironment As Long, ByVal
lpCurrentDirectory As Long, _
   lpStartupInfo As STARTUPINFO,
lpProcessInformation As _
   PROCESS_INFORMATION) As Long

Public Declare Function GetLastError Lib
"kernel32" () As Long

Private Type OPENFILENAME
   lStructSize As Long
   hwndOwner As Long
   hInstance As Long
   lpstrFilter As String
   lpstrCustomFilter As String
   nMaxCustFilter As Long
   nFilterIndex As Long
   lpstrFile As String
   nMaxFile As Long
   lpstrFileTitle As String
   nMaxFileTitle As Long
   lpstrInitialDir As String
   lpstrTitle As String
   Flags As Long
   nFileOffset As Integer
   nFileExtension As Integer
   lpstrDefExt As String
   lCustData As Long
   lpfnHook As Long
   lpTemplateName As Long
End Type

Private Declare Function GetOpenFileName
Lib "COMDLG32" _
      Alias "GetOpenFileNameA" (file As
OPENFILENAME) As Long

Private Declare Function lstrlen Lib "kernel32"
(lpsz As String) As Long

Public Const MAX_PATH = 255

' Used when creating a process
Public Const SW_SHOWMINNOACTIVE = 7
Public Const STARTF_USESHOWWINDOW
= &H1

Public Const MB_YESNOCANCEL = &H3&
Public Const MB_ABORTRETRYIGNORE =
&H2&
Public Const MB_OK = &H0&

Public Const MB_APPLMODAL = &H0&

Public Const MB_ICONQUESTION =
&H20&
Public Const MB_ICONEXCLAMATION =
&H30&

Public Const IDABORT = 3
Public Const IDRETRY = 4
Public Const IDIGNORE = 5
Public Const IDYES = 6
Public Const IDNO = 7
Public Const IDCANCEL = 2

Private Declare Function MessageBox Lib
"user32" Alias "MessageBoxA" ( _
      ByVal hWnd As Long, ByVal lpText As
String, _
      ByVal lpCaption As String, ByVal wType
As Long) As Long

Private Type SYSTEMTIME
      wYear As Integer
      wMonth As Integer
      wDayOfWeek As Integer
      wDay As Integer
      wHour As Integer
      wMinute As Integer
      wSecond As Integer
      wMilliseconds As Integer
End Type

Private Declare Function Get64BitTime Lib
"smtime.dll" ( _
      ByVal lpInitTime As Any) As Currency

Public Function ShowMessageBox(hWnd As Long,
strText As String, _
      strTitle As String, wType As Integer) As Long
   ' Using the Windows MessageBox Api since the
VB Msgbox function suppresses
   ' all events
   ShowMessageBox = MessageBox(hWnd, ByVal
strText, ByVal strTitle, wType)

   If ShowMessageBox = 0 Then
      LogSystemError
      Err.Raise vbObjectError + errConfirmFailed,
App.EXEName, _
            LoadResString(errConfirmFailed)
   End If

End Function
Public Function ShowFileOpenDialog(ByVal
strFilter As String, _
      ByVal strDialogTitle As String, ByVal
lngFlags As Long, _
      Optional ByVal strOldFile As String =
gstrEmptyString) As String
   ' Returns the file name selected by the user
   Dim strInitDir As String
   Dim intPos As Integer
   Dim opfile As OPENFILENAME
   Dim sFile As String

   On Error GoTo ShowFileOpenDialogErr

   If Not StringEmpty(strOldFile) Then
      intPos = InstrR(strOldFile, gstrFileSeparator)
      If intPos > 0 Then
         strInitDir = Left$(strOldFile, intPos - 1)
      End If
   End If

   With opfile
      .lStructSize = Len(opfile)
      .Flags = lngFlags
      .lpstrInitialDir = strInitDir
      .lpstrTitle = strDialogTitle
      .lpstrFilter = MakeWindowsFilter(strFilter)
      sFile = strOldFile & String$(MAX_PATH -
Len(strOldFile), 0)
      .lpstrFile = sFile
      .nMaxFile = MAX_PATH
   End With

   If GetOpenFileName(opfile) Then
      ShowFileOpenDialog = Left$(opfile.lpstrFile,
InStr(opfile.lpstrFile, vbNullChar) - 1)
   Else
      ShowFileOpenDialog = strOldFile
   End If

   Exit Function

ShowFileOpenDialogErr:
   Call LogErrors(Errors)
   ' Reset the selection to the passed in file, if any
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   ShowFileOpenDialog = strOldFile

End Function

Private Function
MakeWindowsFilter(sFilter As String) As
String

   Dim s As String, ch As String, iTemp As
Integer

   On Error GoTo MakeWindowsFilterErr

   ' To make Windows-style filter, replace |
and : with nulls
   For iTemp = 1 To Len(sFilter)
      ch = Mid$(sFilter, iTemp, 1)
      If ch = "|" Then
         s = s & vbNullChar
      Else
         s = s & ch
      End If
   Next iTemp

   ' Put double null at end
   s = s & vbNullChar & vbNullChar
   MakeWindowsFilter = s

   Exit Function

MakeWindowsFilterErr:
   Call LogErrors(Errors)
   gstrSource = mstrModuleName &
"MakeWindowsFilter"
   On Error GoTo 0
   Err.Raise vbObjectError + errApiFailed,
gstrSource, _
         LoadResString(errApiFailed)

End Function


Public Function GetShortName(ByVal
sLongFileName As String) As String
   ' Returns the short name for the passed in
file - will only work
   ' if the passed in path/file exists

   Dim lRetVal As Long, sShortPathName
As String, iLen As Integer
   Dim sLongFile As String
   Dim sDir As String
   Dim sFile As String
   Dim intPos As Integer

   On Error GoTo GetShortNameErr

   sFile = gstrEmptyString
   sLongFile =
MakePathValid(sLongFileName)
   If StringEmpty(Dir$(sLongFile,
vbNormal + vbDirectory)) Then
      ' The passed in path is a file that does
not exist - since
      ' the GetShortPathName api does not
work on non-existent files
      ' on Win2K, use the directory as an
argument to the api and
      ' then append the file
      intPos = InstrR(sLongFile,
gstrFileSeparator)
      sDir = Mid$(sLongFile, 1, intPos - 1)
      sFile = Right(sLongFile,
Len(sLongFile) - intPos + 1)
      sLongFile = sDir
   End If

   'Set up buffer area for API function call
return
   sShortPathName = Space(MAX_PATH)
   iLen = Len(sShortPathName)

   'Call the function
   lRetVal = GetShortPathName(sLongFile,
sShortPathName, iLen)
   If lRetVal = 0 Then
      Call LogSystemError
   End If

   GetShortName = IIf(lRetVal = 0, sLongFile,
Left(sShortPathName, lRetVal))
   If Not StringEmpty(sFile) Then
      GetShortName = GetShortName & sFile
   End If

   Exit Function

GetShortNameErr:
   Call LogErrors(Errors)
   gstrSource = mstrModuleName &
"GetShortName"
   On Error GoTo 0
   Err.Raise vbObjectError + errApiFailed,
gstrSource, _
         LoadResString(errApiFailed)

End Function
Public Function Determine64BitTime() As
Currency

   Determine64BitTime =
Get64BitTime(ByVal 0&)

End Function


Attribute VB_Name = "WorkspaceCommon"
'  FILE:      WorkspaceCommon.bas
'             Microsoft TPC-H Kit Ver. 1.00
'             Copyright Microsoft, 1999
'             All Rights Reserved
'
'
'  PURPOSE:    Contains functionality common
across StepMaster and
'             SMRunOnly, pertaining to
workspaces
'             Specifically, functions to read
workspace records from
'             the database and so on.
'  Contact:    Reshma Tharamal
(reshmat@microsoft.com)
'
Option Explicit

' Used to indicate the source module name
when errors
' are raised by this module
Private Const mstrModuleName As String =
"WorkspaceCommon."


Public Function GetWorkspaceDetails( _
   Optional ByVal WorkspaceId As Long, _
   Optional WorkspaceName As String =
gstrEmptyString _
   ) As Variant
   ' Depending on the passed in parameter, it
returns
   ' either the workspace name or the workspace
identifier
   ' in a variant. The calling function must convert
the
   ' return value to the appropriate type

   Dim rstWorkspace As Recordset
   Dim qyWsp As DAO.QueryDef
   Dim strSql As String
   Dim cTempStr As cStringSM

   On Error GoTo GetWorkspaceDetailsErr
   gstrSource = mstrModuleName &
"GetWorkspaceDetails"

   If WorkspaceId = 0 And _
         WorkspaceName = gstrEmptyString Then
      On Error GoTo 0
      Err.Raise vbObjectError +
errMandatoryParameterMissing, _
            gstrSource, _

LoadResString(errMandatoryParameterMissing)
   End If

   Set cTempStr = New cStringSM

   If WorkspaceId = 0 Then
      strSql = " Select workspace_id from
att_workspaces " & _
            " where workspace_name = [w_name] "
      Set qyWsp =
dbsAttTool.CreateQueryDef(gstrEmptyString,
strSql)
      qyWsp.Parameters("w_name").Value =
WorkspaceName
   Else
      strSql = " Select workspace_name from
att_workspaces " & _
            " where workspace_id = [w_id] "
      Set qyWsp =
dbsAttTool.CreateQueryDef(gstrEmptyString,
strSql)
      qyWsp.Parameters("w_id").Value =
WorkspaceId
   End If

   Set cTempStr = Nothing

   Set rstWorkspace =
qyWsp.OpenRecordset(dbOpenForwardOnly)

   If rstWorkspace.RecordCount <> 0 Then
      GetWorkspaceDetails =
rstWorkspace.Fields(0)
   Else
      rstWorkspace.Close
      qyWsp.Close
      On Error GoTo 0
      Err.Raise vbObjectError +
errInvalidWorkspaceData, _
            gstrSource, _

LoadResString(errInvalidWorkspaceData)
   End If

   rstWorkspace.Close
   qyWsp.Close
   Exit Function

GetWorkspaceDetailsErr:
   Call LogErrors(Errors)
   gstrSource = mstrModuleName &
"GetWorkspaceDetails"
   On Error GoTo 0
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   Err.Raise vbObjectError +
errGetWorkspaceDetailsFailed, _
         gstrSource, _

LoadResString(errGetWorkspaceDetailsFail
ed)

End Function

Public Sub
ReadStepsInWorkspace(rstStepsInWorkSpa
ce As Recordset, _
      qySteps As DAO.QueryDef, _
      Optional lngWorkspaceId As Long =
glInvalidId, _
      Optional dbLoad As DAO.Database =
Nothing, _
      Optional ByVal
bSelectArchivedRecords As Boolean =
False)

   ' This function will populate the passed in
recordset with
   ' all the steps for a given workspace (if
one is passed in, else all workspaces)

   Dim strSql As String

   On Error GoTo
ReadStepsInWorkspaceErr

   ' Create a recordset object to retrieve all
steps for
   ' the given workspace
   strSql = "Select step_id, step_label,
step_file_name, step_text, " & _
      " start_directory, version_no,
workspace_id, " & _
      " parent_step_id, parent_version_no, "
& _
      " sequence_no, step_level, " & _
      " enabled_flag, degree_parallelism, " &
_
      " execution_mechanism, " & _
      " failure_details, continuation_criteria,
" & _
      " global_flag, archived_flag, " & _
      " output_file_name, " & _
      " error_file_name, iterator_name " & _
      " from att_steps a " & _
      " where "

      ' log_file_name,

   If lngWorkspaceId <> glInvalidId Then
      strSql = strSql & " workspace_id =
[w_id] AND "
   End If

   If Not bSelectArchivedRecords Then
      strSql = strSql & " archived_flag =
[archived] AND "
   End If

   ' Find the highest X-component of the
version number
   strSql = strSql & " cint( mid( version_no,
1, instr( version_no, " & gstrDQ &
gstrVerSeparator & gstrDQ & " ) - 1 ) ) = "
& _
      " ( select max( cint( mid( version_no, 1,
instr( version_no, " & gstrDQ &
gstrVerSeparator & gstrDQ & " ) - 1 ) ) ) "
& _
      " from att_steps AS d " & _
      " WHERE a.step_id = d.step_id ) "

   ' Find the highest Y-component of the
version number for the highest X-component
   strSql = strSql & " AND cint( mid(
version_no, instr( version_no, " & gstrDQ &
gstrVerSeparator & gstrDQ & " ) + 1 ) ) = " &
_
      " ( select max( cint( mid( version_no,
instr( version_no, " & gstrDQ &
gstrVerSeparator & gstrDQ & " ) + 1 ) ) ) " &
_
      " from att_steps AS b " & _
      " Where a.step_id = b.step_id " & _
      " AND cint( mid( version_no, 1, instr(
version_no, " & gstrDQ & gstrVerSeparator &
gstrDQ & " ) - 1 ) ) = " & _
      " ( select max( cint( mid( version_no, 1,
instr( version_no, " & gstrDQ &
gstrVerSeparator & gstrDQ & " ) - 1 ) ) ) " & _
      " from att_steps AS c " & _
      " WHERE a.step_id = c.step_id ) ) "

   ' Append the order clause as follows
   ' First, separate all global/non-global steps
   ' Order the worker and manager steps by
step_level to
   ' ensure that the parent steps are populated
before
   ' any sub-steps within it
   ' Further ordering by parent_step_id and
sequence_no
   ' ensures that all the children within a parent
are
   ' selected in the necessary order
   strSql = strSql & " order by global_flag,
step_level, " & _
      " parent_step_id, sequence_no "

   If dbLoad Is Nothing Then Set dbLoad =
dbsAttTool

   ' Create a temporary Querydef object
   Set qySteps =
dbLoad.CreateQueryDef(gstrEmptyString,
strSql)

   ' Initialize the parameter values
   If lngWorkspaceId <> glInvalidId Then
      qySteps.Parameters("w_id").Value =
lngWorkspaceId
   End If

   If Not bSelectArchivedRecords Then
      qySteps.Parameters("archived").Value =
False
   End If

   Set rstStepsInWorkSpace =
qySteps.OpenRecordset(dbOpenSnapshot)

   Exit Sub

ReadStepsInWorkspaceErr:

   LogErrors Errors
   gstrSource = mstrModuleName &
"ReadStepsInWorkspace"
   On Error GoTo 0
   Err.Raise vbObjectError +
errReadWorkspaceDataFailed, _
         gstrSource, _

LoadResString(errReadWorkspaceDataFailed)

End Sub
Public Sub ReadWorkspaces(dbLoad As Database,
rstWsp As Recordset, _
      qyWsp As DAO.QueryDef, _
      Optional ByVal bSelectArchivedRecords As
Boolean = False)

   ' This function will populate the passed in
recordset with all workspace records

   Dim strSql As String

   On Error GoTo ReadWorkspacesErr

   ' Create a recordset object containing all the
workspaces
   ' (that haven't been archived) in the database
   strSql = " Select workspace_id,
workspace_name, archived_flag " & _
         " from att_workspaces "

   If Not bSelectArchivedRecords Then
      strSql = strSql & " where archived_flag =
[archived]"
   End If
   strSql = strSql & " order by workspace_name"

   Set qyWsp =
dbLoad.CreateQueryDef(gstrEmptyString, strSql)
   If Not bSelectArchivedRecords Then
      qyWsp.Parameters("archived").Value = False
   End If

   Set rstWsp =
qyWsp.OpenRecordset(dbOpenForwardOnly)

   Exit Sub

ReadWorkspacesErr:

   LogErrors Errors
   gstrSource = mstrModuleName &
"ReadWorkspaces"
   On Error GoTo 0
   Err.Raise vbObjectError +
errReadWorkspaceDataFailed, _
         gstrSource, _

LoadResString(errReadWorkspaceDataFailed)

End Sub
Public Sub ShowWorkspacesInDb(dbLoad As
Database)

   Dim recWorkspaces As Recordset
   Dim qryAllWsp As QueryDef

   On Error GoTo ShowWorkspacesInDbErr

   ' Set the mousepointer to indicate Busy
   Call ShowBusy

   Load frmWorkspaceOpen

   Call ReadWorkspaces(dbLoad, recWorkspaces,
qryAllWsp)

   frmWorkspaceOpen.lstWorkspaces.Clear

   ' Load all the workspaces into the listbox
   If recWorkspaces.RecordCount <> 0 Then
      Do
         ' Add the workspace name to the list and
store
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
        ' the corresponding workspace id as
the ItemData
        ' property of the item.
        ' The workspace id will be used for
all further
        ' processing of the workspace

frmWorkspaceOpen.lstWorkspaces.AddItem
recWorkspaces![workspace_name]

frmWorkspaceOpen.lstWorkspaces.ItemDat
a(frmWorkspaceOpen.lstWorkspaces.NewIn
dex) = _
            recWorkspaces![workspace_id]
        recWorkspaces.MoveNext

      Loop Until recWorkspaces.EOF
   End If
   recWorkspaces.Close
   qryAllWsp.Close

   ' Reset the mousepointer
   ShowFree

   #If RUN_ONLY Then
      frmWorkspaceOpen.Show vbModal
   #Else
      frmWorkspaceOpen.Show vbModal,
frmMain
   #End If

   Exit Sub

ShowWorkspacesInDbErr:
   LogErrors Errors
   Call ShowFree
   Err.Raise vbObjectError +
errProgramError, mstrModuleName &
"ShowWorkspacesInDb", _
        LoadResString(errProgramError)

End Sub
Private Sub
ReadWorkspaceParameters(lngWorkspaceId
As Long, _
   rstWorkSpaceParameters As Recordset, _
   qyWspParams As DAO.QueryDef)

   ' Will populate the recordset with all the
parameters for
   ' a given workspace

   Dim strSql As String

   On Error GoTo
ReadWorkspaceParametersErr

   strSql = "Select parameter_id,
parameter_name, " & _
        " parameter_value, workspace_id,
parameter_type, description " & _
        " from workspace_parameters " & _
        " where workspace_id = [w_id] " &
_
        " order by parameter_name,
parameter_value "

   ' Create a temporary Querydef object and
initialize
   ' it's parameter values
   Set qyWspParams =
dbsAttTool.CreateQueryDef(gstrEmptyStrin
g, strSql)
   qyWspParams.Parameters("w_id").Value
= lngWorkspaceId

   Set rstWorkSpaceParameters =
qyWspParams.OpenRecordset(dbOpenSnapsh
ot)

   Exit Sub

ReadWorkspaceParametersErr:

   LogErrors Errors
   gstrSource = mstrModuleName &
"ReadWorkspaceParameters"
   On Error GoTo 0
   Err.Raise vbObjectError +
errReadWorkspaceDataFailed, _
        gstrSource, _

LoadResString(errReadWorkspaceDataFailed)

End Sub
Private Sub ReadConnections(lngWorkspaceId
As Long, rstConns As Recordset, _
   qyConns As DAO.QueryDef)

   ' Will populate the recordset with all the
parameters for
   ' a given workspace

   Dim strSql As String

   On Error GoTo
ReadWorkspaceParametersErr

   strSql = "Select connection_id, " & _
        " connection_name, connection_value,
workspace_id, description, " & _
        " no_count_display, no_execute,
parse_query_only, ANSI_quoted_identifiers, "
& _
        " ANSI_nulls, show_query_plan,
show_stats_time, show_stats_io, " & _
        " parse_odbc_msg_prefixes,
row_count, tsql_batch_separator,
query_time_out, " & _
        " server_language,
character_translation, regional_settings " & _
        " from workspace_connections " & _
        " where workspace_id = [w_id] " & _
        " order by connection_name,
connection_value "

   ' Create a temporary Querydef object and
initialize
   ' it's parameter values
   Set qyConns =
dbsAttTool.CreateQueryDef(gstrEmptyString,
strSql)
   qyConns.Parameters("w_id").Value =
lngWorkspaceId

   Set rstConns =
qyConns.OpenRecordset(dbOpenSnapshot)

   Exit Sub

ReadWorkspaceParametersErr:

   LogErrors Errors
   On Error GoTo 0
   Err.Raise vbObjectError +
errReadWorkspaceDataFailed, _
        mstrModuleName &
"ReadConnections",
LoadResString(errReadWorkspaceDataFailed)

End Sub
Private Sub ReadConnectionDtls(lngWorkspaceId
As Long, rstConns As Recordset, _
   qyConns As DAO.QueryDef)

   ' Will populate the recordset with all the
connection_dtls records for
   ' a given workspace

   Dim strSql As String

   On Error GoTo ReadWorkspaceParametersErr

   strSql = "Select " & FLD_ID_CONN_NAME &
", " & _

FLD_CONN_DTL_CONNECTION_NAME & ", "
& _

FLD_CONN_DTL_CONNECTION_STRING & ",
" & _
        FLD_ID_WORKSPACE & ", " & _
        FLD_CONN_DTL_CONNECTION_TYPE
& _
        " from " & TBL_CONNECTION_DTLS &
_
        " where " & FLD_ID_WORKSPACE & " =
[w_id] " & _
        " order by " &
FLD_CONN_DTL_CONNECTION_NAME

   ' Create a temporary Querydef object and
initialize
   ' it's parameter values
   Set qyConns =
dbsAttTool.CreateQueryDef(gstrEmptyString,
strSql)
   qyConns.Parameters("w_id").Value =
lngWorkspaceId
   Set rstConns =
qyConns.OpenRecordset(dbOpenSnapshot)
   Exit Sub

ReadWorkspaceParametersErr:

   LogErrors Errors
   On Error GoTo 0
   Err.Raise vbObjectError +
errReadWorkspaceDataFailed, _
        mstrModuleName & "ReadConnectionDtls",
LoadResString(errReadWorkspaceDataFailed)

End Sub

Public Sub ReadWorkspaceData(lngWorkspaceId
As Long, _
      cStepsCol As cArrSteps, _
      cParamsCol As cArrParameters, _
      cConsCol As cArrConstraints, _
      cConns As cConnections, _
      cConnDetails As cConnDtls, _
      rstStepsInWsp As Recordset, _
      qyStepsInWsp As DAO.QueryDef, _
      rstParamsInWsp As Recordset, _
      qyParamsInWsp As DAO.QueryDef, _
      rstConns As Recordset, _
      qyConns As DAO.QueryDef, _
      rstConnDtls As Recordset, _
      qyConnDtls As DAO.QueryDef)
   ' Loads the passed in structures with all the data
for
   ' the workspace. It also initializes the recordsets
   ' with the step and parameter records for the
workspace.
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   On Error GoTo ReadWorkspaceDataErr

   ShowBusy

   Call
ReadStepsInWorkspace(rstStepsInWsp,
qyStepsInWsp, lngWorkspaceId)

   ' Load all the steps in the array
   LoadRecordsetInStepsArray
rstStepsInWsp, cStepsCol

   ' Initialize the steps with all the iterator
   ' records for each step
   Call LoadIteratorsForWsp(cStepsCol,
lngWorkspaceId, rstStepsInWsp)
      ReadWorkspaceParameters
lngWorkspaceId, rstParamsInWsp,
qyParamsInWsp

      ' Load all the workspace parameters in the
array
   LoadRecordsetInParameterArray
rstParamsInWsp, cParamsCol
   ' Read and load connection strings
   ReadConnections lngWorkspaceId,
rstConns, qyConns
   LoadRecordsetInConnectionArray rstConns,
cConns
   ' Read and load connection information
   ReadConnectionDtls lngWorkspaceId,
rstConnDtls, qyConnDtls
   LoadRSInConnDtlArray rstConnDtls,
cConnDetails
   ' Finally, load the step constraints collection
class with
   ' all the constraints for the steps in the
workspace
   cConsCol.LoadConstraints lngWorkspaceId,
rstStepsInWsp

   ShowFree
   Exit Sub

ReadWorkspaceDataErr:
   ' Log the error code raised by Visual Basic
   ShowFree
   Call LogErrors(Errors)
   On Error GoTo 0
   gstrSource = mstrModuleName &
"ReadWorkspaceData"
   Err.Raise vbObjectError +
errReadWorkspaceDataFailed, _
         gstrSource, _

LoadResString(errReadWorkspaceDataFailed)

End Sub
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
//  FILE:      LogWriter.cpp
//             Microsoft TPC-H Kit Ver. 1.00
//             Copyright Microsoft, 1999
//             All Rights Reserved
//
//  PURPOSE:    Implementation of DLL Exports.
//  Contact:    Reshma Tharamal (reshmat@microsoft.com)
//
// LogWriter.cpp : Implementation of DLL Exports.

// Note: Proxy/Stub Information
//      To build a separate proxy/stub DLL,
//      run nmake -f LogWriterps.mk in the project directory.

#include "stdafx.h"
#include "resource.h"
#include <initguid.h>
#include "LogWriter.h"

#include "LogWriter_i.c"
#include "SMLog.h"

CComModule _Module;

BEGIN_OBJECT_MAP(ObjectMap)
OBJECT_ENTRY(CLSID_SMLog, CSMLog)
END_OBJECT_MAP()

/////////////////////////////////////////////////////////////////////////////
// DLL Entry Point

extern "C"
BOOL WINAPI DllMain(HINSTANCE hInstance, DWORD
dwReason, LPVOID /*lpReserved*/)
{
   if (dwReason == DLL_PROCESS_ATTACH)
   {
      _Module.Init(ObjectMap, hInstance,
&LIBID_LOGWRITERLib);
      DisableThreadLibraryCalls(hInstance);
   }
   else if (dwReason == DLL_PROCESS_DETACH)
      _Module.Term();
   return TRUE;    // ok
}
/////////////////////////////////////////////////////////////////////////////
// Used to determine whether the DLL can be unloaded by OLE

STDAPI DllCanUnloadNow(void)
{
   return (_Module.GetLockCount()==0) ? S_OK : S_FALSE;
}

/////////////////////////////////////////////////////////////////////////////
// Returns a class factory to create an object of the requested type

STDAPI DllGetClassObject(REFCLSID rclsid, REFIID riid,
LPVOID* ppv)
{
   return _Module.GetClassObject(rclsid, riid, ppv);
}
/////////////////////////////////////////////////////////////////////////////
// DllRegisterServer - Adds entries to the system registry

STDAPI DllRegisterServer(void)
{
   // registers object, typelib and all interfaces in typelib
   return _Module.RegisterServer(TRUE);
}
/////////////////////////////////////////////////////////////////////////////
// DllUnregisterServer - Removes entries from the system
registry

STDAPI DllUnregisterServer(void)
{
   return _Module.UnregisterServer(TRUE);
}


/* this ALWAYS GENERATED file contains the definitions for
the interfaces */


/* File created by MIDL compiler version 5.01.0164 */
/* at Thu Sep 19 17:49:50 2002
 */
/* Compiler settings for
C:\charles\Stepmaster\LogWriter\LogWriter.idl:
   Oicf (OptLev=i2), W1, Zp8, env=Win32, ms_ext, c_ext
   error checks: allocation ref bounds_check enum stub_data
*/
//@@MIDL_FILE_HEADING(  )


/* verify that the <rpcndr.h> version is high enough to compile
this file*/
#ifndef __REQUIRED_RPCNDR_H_VERSION__
#define __REQUIRED_RPCNDR_H_VERSION__ 440
#endif

#include "rpc.h"
#include "rpcndr.h"

#ifndef __RPCNDR_H_VERSION__
#error this stub requires an updated version of <rpcndr.h>
#endif // __RPCNDR_H_VERSION__

#ifndef COM_NO_WINDOWS_H
#include "windows.h"
#include "ole2.h"
#endif /*COM_NO_WINDOWS_H*/

#ifndef __LogWriter_h__
#define __LogWriter_h__

#ifdef __cplusplus
extern "C"{
#endif

/* Forward Declarations */

#ifndef __ISMLog_FWD_DEFINED__
#define __ISMLog_FWD_DEFINED__
typedef interface ISMLog ISMLog;
#endif    /* __ISMLog_FWD_DEFINED__ */


#ifndef ___ISMLogEvents_FWD_DEFINED__
#define ___ISMLogEvents_FWD_DEFINED__
typedef interface _ISMLogEvents _ISMLogEvents;
#endif    /* ___ISMLogEvents_FWD_DEFINED__ */


#ifndef __SMLog_FWD_DEFINED__
#define __SMLog_FWD_DEFINED__
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
#ifdef __cplusplus
typedef class SMLog SMLog;
#else
typedef struct SMLog SMLog;
#endif /* __cplusplus */

#endif    /* __SMLog_FWD_DEFINED__ */

/* header files for imported files */
#include "oaidl.h"
#include "ocidl.h"

void __RPC_FAR * __RPC_USER MIDL_user_allocate(size_t);
void __RPC_USER MIDL_user_free( void __RPC_FAR * );

#ifndef __ISMLog_INTERFACE_DEFINED__
#define __ISMLog_INTERFACE_DEFINED__

/* interface ISMLog */
/* [unique][helpstring][dual][uuid][object] */

EXTERN_C const IID IID_ISMLog;

#if defined(__cplusplus) && !defined(CINTERFACE)

   MIDL_INTERFACE("5AC75DAD-1936-11D3-BC2D-
00A0C90D2CA5")
   ISMLog : public IDispatch
   {
   public:
      virtual /* [helpstring][id][propput] */ HRESULT
STDMETHODCALLTYPE put_FileHeader(
         /* [in] */ BSTR newVal) = 0;

      virtual /* [helpstring][id] */ HRESULT
STDMETHODCALLTYPE WriteLine(
         BSTR szMsg) = 0;

      virtual /* [helpstring][id] */ HRESULT
STDMETHODCALLTYPE WriteField(
         BSTR szMsg) = 0;

      virtual /* [helpstring][id][propput] */ HRESULT
STDMETHODCALLTYPE put_FileName(
         /* [in] */ BSTR newVal) = 0;

      virtual /* [helpstring][id][propput] */ HRESULT
STDMETHODCALLTYPE put_Append(
         /* [in] */ BOOL newVal) = 0;

   };
#else      /* C style interface */

   typedef struct ISMLogVtbl
   {
      BEGIN_INTERFACE

      HRESULT ( STDMETHODCALLTYPE __RPC_FAR
*QueryInterface )(
         ISMLog __RPC_FAR * This,
         /* [in] */ REFIID riid,
         /* [iid_is][out] */ void __RPC_FAR *__RPC_FAR
*ppvObject);

      ULONG ( STDMETHODCALLTYPE __RPC_FAR
*AddRef )(
         ISMLog __RPC_FAR * This);

      ULONG ( STDMETHODCALLTYPE __RPC_FAR
*Release )(
         ISMLog __RPC_FAR * This);

      HRESULT ( STDMETHODCALLTYPE __RPC_FAR
*GetTypeInfoCount )(
         ISMLog __RPC_FAR * This,
         /* [out] */ UINT __RPC_FAR *pctinfo);

      HRESULT ( STDMETHODCALLTYPE __RPC_FAR
*GetTypeInfo )(
         ISMLog __RPC_FAR * This,
         /* [in] */ UINT iTInfo,
         /* [in] */ LCID lcid,
         /* [out] */ ITypeInfo __RPC_FAR *__RPC_FAR
*ppTInfo);

      HRESULT ( STDMETHODCALLTYPE __RPC_FAR
*GetIDsOfNames )(
         ISMLog __RPC_FAR * This,
         /* [in] */ REFIID riid,
         /* [size_is][in] */ LPOLESTR __RPC_FAR *rgszNames,
         /* [in] */ UINT cNames,
         /* [in] */ LCID lcid,
         /* [size_is][out] */ DISPID __RPC_FAR *rgDispId);

      /* [local] */ HRESULT ( STDMETHODCALLTYPE
__RPC_FAR *Invoke )(
         ISMLog __RPC_FAR * This,
         /* [in] */ DISPID dispIdMember,
         /* [in] */ REFIID riid,
         /* [in] */ LCID lcid,
         /* [in] */ WORD wFlags,
         /* [out][in] */ DISPPARAMS __RPC_FAR
*pDispParams,
         /* [out] */ VARIANT __RPC_FAR *pVarResult,
         /* [out] */ EXCEPINFO __RPC_FAR *pExcepInfo,
         /* [out] */ UINT __RPC_FAR *puArgErr);

      /* [helpstring][id][propput] */ HRESULT (
STDMETHODCALLTYPE __RPC_FAR *put_FileHeader )(
         ISMLog __RPC_FAR * This,
         /* [in] */ BSTR newVal);

      /* [helpstring][id] */ HRESULT (
STDMETHODCALLTYPE __RPC_FAR *WriteLine )(
         ISMLog __RPC_FAR * This,
         BSTR szMsg);

      /* [helpstring][id] */ HRESULT (
STDMETHODCALLTYPE __RPC_FAR *WriteField )(
         ISMLog __RPC_FAR * This,
         BSTR szMsg);

      /* [helpstring][id][propput] */ HRESULT (
STDMETHODCALLTYPE __RPC_FAR *put_FileName )(
         ISMLog __RPC_FAR * This,
         /* [in] */ BSTR newVal);

      /* [helpstring][id][propput] */ HRESULT (
STDMETHODCALLTYPE __RPC_FAR *put_Append )(
         ISMLog __RPC_FAR * This,
         /* [in] */ BOOL newVal);

      END_INTERFACE
   } ISMLogVtbl;
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
    interface ISMLog
    {
        CONST_VTBL struct ISMLogVtbl __RPC_FAR *lpVtbl;
    };



#ifdef COBJMACROS


#define ISMLog_QueryInterface(This,riid,ppvObject)      \
    (This)->lpVtbl -> QueryInterface(This,riid,ppvObject)

#define ISMLog_AddRef(This)      \
    (This)->lpVtbl -> AddRef(This)

#define ISMLog_Release(This)      \
    (This)->lpVtbl -> Release(This)

#define ISMLog_GetTypeInfoCount(This,pctinfo)     \
    (This)->lpVtbl -> GetTypeInfoCount(This,pctinfo)

#define ISMLog_GetTypeInfo(This,iTInfo,lcid,ppTInfo)  \
    (This)->lpVtbl -> GetTypeInfo(This,iTInfo,lcid,ppTInfo)

#define
ISMLog_GetIDsOfNames(This,riid,rgszNames,cNames,lcid,rgD
ispId)     \
    (This)->lpVtbl ->
GetIDsOfNames(This,riid,rgszNames,cNames,lcid,rgDispId)

#define
ISMLog_Invoke(This,dispIdMember,riid,lcid,wFlags,pDispPara
ms,pVarResult,pExcepInfo,puArgErr)    \
    (This)->lpVtbl ->
Invoke(This,dispIdMember,riid,lcid,wFlags,pDispParams,pVarR
esult,pExcepInfo,puArgErr)


#define ISMLog_put_FileHeader(This,newVal)   \
    (This)->lpVtbl -> put_FileHeader(This,newVal)

#define ISMLog_WriteLine(This,szMsg)      \
    (This)->lpVtbl -> WriteLine(This,szMsg)

#define ISMLog_WriteField(This,szMsg)     \
    (This)->lpVtbl -> WriteField(This,szMsg)

#define ISMLog_put_FileName(This,newVal)     \
    (This)->lpVtbl -> put_FileName(This,newVal)

#define ISMLog_put_Append(This,newVal)\
    (This)->lpVtbl -> put_Append(This,newVal)

#endif /* COBJMACROS */


#endif    /* C style interface */


/* [helpstring][id][propput] */ HRESULT
STDMETHODCALLTYPE ISMLog_put_FileHeader_Proxy(
    ISMLog __RPC_FAR * This,
    /* [in] */ BSTR newVal);
```

```
void __RPC_STUB ISMLog_put_FileHeader_Stub(
    IRpcStubBuffer *This,
    IRpcChannelBuffer *_pRpcChannelBuffer,
    PRPC_MESSAGE _pRpcMessage,
    DWORD *_pdwStubPhase);


/* [helpstring][id] */ HRESULT STDMETHODCALLTYPE
ISMLog_WriteLine_Proxy(
    ISMLog __RPC_FAR * This,
    BSTR szMsg);


void __RPC_STUB ISMLog_WriteLine_Stub(
    IRpcStubBuffer *This,
    IRpcChannelBuffer *_pRpcChannelBuffer,
    PRPC_MESSAGE _pRpcMessage,
    DWORD *_pdwStubPhase);


/* [helpstring][id] */ HRESULT STDMETHODCALLTYPE
ISMLog_WriteField_Proxy(
    ISMLog __RPC_FAR * This,
    BSTR szMsg);


void __RPC_STUB ISMLog_WriteField_Stub(
    IRpcStubBuffer *This,
    IRpcChannelBuffer *_pRpcChannelBuffer,
    PRPC_MESSAGE _pRpcMessage,
    DWORD *_pdwStubPhase);


/* [helpstring][id][propput] */ HRESULT
STDMETHODCALLTYPE ISMLog_put_FileName_Proxy(
    ISMLog __RPC_FAR * This,
    /* [in] */ BSTR newVal);

void __RPC_STUB ISMLog_put_FileName_Stub(
    IRpcStubBuffer *This,
    IRpcChannelBuffer *_pRpcChannelBuffer,
    PRPC_MESSAGE _pRpcMessage,
    DWORD *_pdwStubPhase);


/* [helpstring][id][propput] */ HRESULT
STDMETHODCALLTYPE ISMLog_put_Append_Proxy(
    ISMLog __RPC_FAR * This,
    /* [in] */ BOOL newVal);

void __RPC_STUB ISMLog_put_Append_Stub(
    IRpcStubBuffer *This,
    IRpcChannelBuffer *_pRpcChannelBuffer,
    PRPC_MESSAGE _pRpcMessage,
    DWORD *_pdwStubPhase);

#endif    /* __ISMLog_INTERFACE_DEFINED__ */

#ifndef __LOGWRITERLib_LIBRARY_DEFINED__
#define __LOGWRITERLib_LIBRARY_DEFINED__
EXTERN_C const IID LIBID_LOGWRITERLib;

#ifndef ___ISMLogEvents_DISPINTERFACE_DEFINED__
#define ___ISMLogEvents_DISPINTERFACE_DEFINED__
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
EXTERN_C const IID DIID__ISMLogEvents;

#if defined(__cplusplus) && !defined(CINTERFACE)

   MIDL_INTERFACE("5AC75DB1-1936-11D3-BC2D-
00A0C90D2CA5")
   _ISMLogEvents : public IDispatch
   {
   };

#else      /* C style interface */

   typedef struct _ISMLogEventsVtbl
   {
      BEGIN_INTERFACE

      HRESULT ( STDMETHODCALLTYPE __RPC_FAR
*QueryInterface )(
         _ISMLogEvents __RPC_FAR * This,
         /* [in] */ REFIID riid,
         /* [iid_is][out] */ void __RPC_FAR *__RPC_FAR
*ppvObject);

      ULONG ( STDMETHODCALLTYPE __RPC_FAR
*AddRef )(
         _ISMLogEvents __RPC_FAR * This);

      ULONG ( STDMETHODCALLTYPE __RPC_FAR
*Release )(
         _ISMLogEvents __RPC_FAR * This);

      HRESULT ( STDMETHODCALLTYPE __RPC_FAR
*GetTypeInfoCount )(
         _ISMLogEvents __RPC_FAR * This,
         /* [out] */ UINT __RPC_FAR *pctinfo);

      HRESULT ( STDMETHODCALLTYPE __RPC_FAR
*GetTypeInfo )(
         _ISMLogEvents __RPC_FAR * This,
         /* [in] */ UINT iTInfo,
         /* [in] */ LCID lcid,
         /* [out] */ ITypeInfo __RPC_FAR *__RPC_FAR
*ppTInfo);

      HRESULT ( STDMETHODCALLTYPE __RPC_FAR
*GetIDsOfNames )(
         _ISMLogEvents __RPC_FAR * This,
         /* [in] */ REFIID riid,
         /* [size_is][in] */ LPOLESTR __RPC_FAR *rgszNames,
         /* [in] */ UINT cNames,
         /* [in] */ LCID lcid,
         /* [size_is][out] */ DISPID __RPC_FAR *rgDispId);

      /* [local] */ HRESULT ( STDMETHODCALLTYPE
__RPC_FAR *Invoke )(
         _ISMLogEvents __RPC_FAR * This,
         /* [in] */ DISPID dispIdMember,
         /* [in] */ REFIID riid,
         /* [in] */ LCID lcid,
         /* [in] */ WORD wFlags,
         /* [out][in] */ DISPPARAMS __RPC_FAR
*pDispParams,
         /* [out] */ VARIANT __RPC_FAR *pVarResult,
         /* [out] */ EXCEPINFO __RPC_FAR *pExcepInfo,
         /* [out] */ UINT __RPC_FAR *puArgErr);

      END_INTERFACE
   } _ISMLogEventsVtbl;


   interface _ISMLogEvents
   {
      CONST_VTBL struct _ISMLogEventsVtbl __RPC_FAR
*lpVtbl;
   };



#ifdef COBJMACROS


#define _ISMLogEvents_QueryInterface(This,riid,ppvObject) \
   (This)->lpVtbl -> QueryInterface(This,riid,ppvObject)

#define _ISMLogEvents_AddRef(This) \
   (This)->lpVtbl -> AddRef(This)

#define _ISMLogEvents_Release(This) \
   (This)->lpVtbl -> Release(This)


#define _ISMLogEvents_GetTypeInfoCount(This,pctinfo)      \
   (This)->lpVtbl -> GetTypeInfoCount(This,pctinfo)

#define _ISMLogEvents_GetTypeInfo(This,iTInfo,lcid,ppTInfo)
      \
   (This)->lpVtbl -> GetTypeInfo(This,iTInfo,lcid,ppTInfo)

#define
_ISMLogEvents_GetIDsOfNames(This,riid,rgszNames,cNames,l
cid,rgDispId)\
   (This)->lpVtbl ->
GetIDsOfNames(This,riid,rgszNames,cNames,lcid,rgDispId)

#define
_ISMLogEvents_Invoke(This,dispIdMember,riid,lcid,wFlags,pD
ispParams,pVarResult,pExcepInfo,puArgErr)      \
   (This)->lpVtbl ->
Invoke(This,dispIdMember,riid,lcid,wFlags,pDispParams,pVarR
esult,pExcepInfo,puArgErr)

#endif /* COBJMACROS */


#endif    /* C style interface */


#endif    /* ___ISMLogEvents_DISPINTERFACE_DEFINED__
*/


EXTERN_C const CLSID CLSID_SMLog;

#ifdef __cplusplus

class DECLSPEC_UUID("5AC75DB0-1936-11D3-BC2D-
00A0C90D2CA5")
SMLog;
#endif
#endif /* __LOGWRITERLib_LIBRARY_DEFINED__ */

/* Additional Prototypes for ALL interfaces */
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
unsigned long __RPC_USER  BSTR_UserSize(unsigned long
__RPC_FAR *, unsigned long , BSTR __RPC_FAR * );
unsigned char __RPC_FAR * __RPC_USER
BSTR_UserMarshal(unsigned long __RPC_FAR *,
unsigned char __RPC_FAR *, BSTR __RPC_FAR * );
unsigned char __RPC_FAR * __RPC_USER
BSTR_UserUnmarshal(unsigned long __RPC_FAR *,
unsigned char __RPC_FAR *, BSTR __RPC_FAR * );
void     __RPC_USER  BSTR_UserFree(     unsigned long
__RPC_FAR *, BSTR __RPC_FAR * );

/* end of Additional Prototypes */

#ifdef __cplusplus
}
#endif

#endif

//  FILE:       LogWriter.cpp
//            Microsoft TPC-H Kit Ver. 1.00
//            Copyright Microsoft, 1999
//            All Rights Reserved
//
//
//  PURPOSE:    Implementation of DLL Exports.
//  Contact:    Reshma Tharamal (reshmat@microsoft.com)
//
// LogWriter.cpp : Implementation of DLL Exports.


// Note: Proxy/Stub Information
//      To build a separate proxy/stub DLL,
//      run nmake -f LogWriterps.mk in the project directory.

#include "stdafx.h"
#include "resource.h"
#include <initguid.h>
#include "LogWriter.h"

#include "LogWriter_i.c"
#include "SMLog.h"


CComModule _Module;

BEGIN_OBJECT_MAP(ObjectMap)
OBJECT_ENTRY(CLSID_SMLog, CSMLog)
END_OBJECT_MAP()

/////////////////////////////////////////////////////////////////////////////
// DLL Entry Point

extern "C"
BOOL WINAPI DllMain(HINSTANCE hInstance, DWORD
dwReason, LPVOID /*lpReserved*/)
{
   if (dwReason == DLL_PROCESS_ATTACH)
   {
      _Module.Init(ObjectMap, hInstance,
&LIBID_LOGWRITERLib);
      DisableThreadLibraryCalls(hInstance);
   }
   else if (dwReason == DLL_PROCESS_DETACH)
      _Module.Term();
   return TRUE;    // ok
}
/////////////////////////////////////////////////////////////////////////////
// Used to determine whether the DLL can be unloaded by OLE

STDAPI DllCanUnloadNow(void)
{
   return (_Module.GetLockCount()==0) ? S_OK : S_FALSE;
}

/////////////////////////////////////////////////////////////////////////////
// Returns a class factory to create an object of the requested type

STDAPI DllGetClassObject(REFCLSID rclsid, REFIID riid,
LPVOID* ppv)
{
   return _Module.GetClassObject(rclsid, riid, ppv);
}

/////////////////////////////////////////////////////////////////////////////
// DllRegisterServer - Adds entries to the system registry

STDAPI DllRegisterServer(void)
{
   // registers object, typelib and all interfaces in typelib
   return _Module.RegisterServer(TRUE);
}

/////////////////////////////////////////////////////////////////////////////
// DllUnregisterServer - Removes entries from the system
registry

STDAPI DllUnregisterServer(void)
{
   return _Module.UnregisterServer(TRUE);
}


/* this ALWAYS GENERATED file contains the definitions for
the interfaces */


/* File created by MIDL compiler version 5.01.0164 */
/* at Thu Sep 19 17:49:50 2002
 */
/* Compiler settings for
C:\charles\Stepmaster\LogWriter\LogWriter.idl:
   Oicf (OptLev=i2), W1, Zp8, env=Win32, ms_ext, c_ext
   error checks: allocation ref bounds_check enum stub_data
*/
//@@MIDL_FILE_HEADING(  )


/* verify that the <rpcndr.h> version is high enough to compile
this file*/
#ifndef __REQUIRED_RPCNDR_H_VERSION__
#define __REQUIRED_RPCNDR_H_VERSION__ 440
#endif

#include "rpc.h"
#include "rpcndr.h"

#ifndef __RPCNDR_H_VERSION__
#error this stub requires an updated version of <rpcndr.h>
#endif // __RPCNDR_H_VERSION__
#ifndef COM_NO_WINDOWS_H
#include "windows.h"
#include "ole2.h"
#endif /*COM_NO_WINDOWS_H*/
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
#ifndef __LogWriter_h__
#define __LogWriter_h__

#ifdef __cplusplus
extern "C"{
#endif

/* Forward Declarations */

#ifndef __ISMLog_FWD_DEFINED__
#define __ISMLog_FWD_DEFINED__
typedef interface ISMLog ISMLog;
#endif    /* __ISMLog_FWD_DEFINED__ */

#ifndef ___ISMLogEvents_FWD_DEFINED__
#define ___ISMLogEvents_FWD_DEFINED__
typedef interface _ISMLogEvents _ISMLogEvents;
#endif    /* ___ISMLogEvents_FWD_DEFINED__ */


#ifndef __SMLog_FWD_DEFINED__
#define __SMLog_FWD_DEFINED__

#ifdef __cplusplus
typedef class SMLog SMLog;
#else
typedef struct SMLog SMLog;
#endif /* __cplusplus */
#endif    /* __SMLog_FWD_DEFINED__ */

/* header files for imported files */
#include "oaidl.h"
#include "ocidl.h"

void __RPC_FAR * __RPC_USER MIDL_user_allocate(size_t);
void __RPC_USER MIDL_user_free( void __RPC_FAR * );

#ifndef __ISMLog_INTERFACE_DEFINED__
#define __ISMLog_INTERFACE_DEFINED__
/* interface ISMLog */
/* [unique][helpstring][dual][uuid][object] */

EXTERN_C const IID IID_ISMLog;

#if defined(__cplusplus) && !defined(CINTERFACE)

   MIDL_INTERFACE("5AC75DAD-1936-11D3-BC2D-
00A0C90D2CA5")
   ISMLog : public IDispatch
   {
   public:
      virtual /* [helpstring][id][propput] */ HRESULT
STDMETHODCALLTYPE put_FileHeader(
         /* [in] */ BSTR newVal) = 0;
      virtual /* [helpstring][id] */ HRESULT
STDMETHODCALLTYPE WriteLine(
         BSTR szMsg) = 0;
      virtual /* [helpstring][id] */ HRESULT
STDMETHODCALLTYPE WriteField(
         BSTR szMsg) = 0;
      virtual /* [helpstring][id][propput] */ HRESULT
STDMETHODCALLTYPE put_FileName(
         /* [in] */ BSTR newVal) = 0;
      virtual /* [helpstring][id][propput] */ HRESULT
STDMETHODCALLTYPE put_Append(
         /* [in] */ BOOL newVal) = 0;
   };
#else     /* C style interface */
   typedef struct ISMLogVtbl
   {
      BEGIN_INTERFACE
      HRESULT ( STDMETHODCALLTYPE __RPC_FAR
*QueryInterface )(
         ISMLog __RPC_FAR * This,
         /* [in] */ REFIID riid,
         /* [iid_is][out] */ void __RPC_FAR *__RPC_FAR
*ppvObject);
      ULONG ( STDMETHODCALLTYPE __RPC_FAR
*AddRef )(
         ISMLog __RPC_FAR * This);
      ULONG ( STDMETHODCALLTYPE __RPC_FAR
*Release )(
         ISMLog __RPC_FAR * This);
      HRESULT ( STDMETHODCALLTYPE __RPC_FAR
*GetTypeInfoCount )(
         ISMLog __RPC_FAR * This,
         /* [out] */ UINT __RPC_FAR *pctinfo);
      HRESULT ( STDMETHODCALLTYPE __RPC_FAR
*GetTypeInfo )(
         ISMLog __RPC_FAR * This,
         /* [in] */ UINT iTInfo,
         /* [in] */ LCID lcid,
         /* [out] */ ITypeInfo __RPC_FAR *__RPC_FAR
*ppTInfo);
      HRESULT ( STDMETHODCALLTYPE __RPC_FAR
*GetIDsOfNames )(
         ISMLog __RPC_FAR * This,
         /* [in] */ REFIID riid,
         /* [size_is][in] */ LPOLESTR __RPC_FAR *rgszNames,
         /* [in] */ UINT cNames,
         /* [in] */ LCID lcid,
         /* [size_is][out] */ DISPID __RPC_FAR *rgDispId);
      /* [local] */ HRESULT ( STDMETHODCALLTYPE
__RPC_FAR *Invoke )(
         ISMLog __RPC_FAR * This,
         /* [in] */ DISPID dispIdMember,
         /* [in] */ REFIID riid,
         /* [in] */ LCID lcid,
         /* [in] */ WORD wFlags,
         /* [out][in] */ DISPPARAMS __RPC_FAR
*pDispParams,
         /* [out] */ VARIANT __RPC_FAR *pVarResult,
         /* [out] */ EXCEPINFO __RPC_FAR *pExcepInfo,
         /* [out] */ UINT __RPC_FAR *puArgErr);
      /* [helpstring][id][propput] */ HRESULT (
STDMETHODCALLTYPE __RPC_FAR *put_FileHeader )(
         ISMLog __RPC_FAR * This,
         /* [in] */ BSTR newVal);
      /* [helpstring][id] */ HRESULT (
STDMETHODCALLTYPE __RPC_FAR *WriteLine )(
         ISMLog __RPC_FAR * This,
         BSTR szMsg);
      /* [helpstring][id] */ HRESULT (
STDMETHODCALLTYPE __RPC_FAR *WriteField )(
         ISMLog __RPC_FAR * This,
         BSTR szMsg);
      /* [helpstring][id][propput] */ HRESULT (
STDMETHODCALLTYPE __RPC_FAR *put_FileName )(
         ISMLog __RPC_FAR * This,
         /* [in] */ BSTR newVal);
      /* [helpstring][id][propput] */ HRESULT (
STDMETHODCALLTYPE __RPC_FAR *put_Append )(
         ISMLog __RPC_FAR * This,
         /* [in] */ BOOL newVal);
      END_INTERFACE
   } ISMLogVtbl;
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
interface ISMLog
{
   CONST_VTBL struct ISMLogVtbl __RPC_FAR *lpVtbl;
};



#ifdef COBJMACROS


#define ISMLog_QueryInterface(This,riid,ppvObject)      \
   (This)->lpVtbl -> QueryInterface(This,riid,ppvObject)

#define ISMLog_AddRef(This)      \
   (This)->lpVtbl -> AddRef(This)

#define ISMLog_Release(This)      \
   (This)->lpVtbl -> Release(This)


#define ISMLog_GetTypeInfoCount(This,pctinfo)     \
   (This)->lpVtbl -> GetTypeInfoCount(This,pctinfo)

#define ISMLog_GetTypeInfo(This,iTInfo,lcid,ppTInfo)  \
   (This)->lpVtbl -> GetTypeInfo(This,iTInfo,lcid,ppTInfo)

#define
ISMLog_GetIDsOfNames(This,riid,rgszNames,cNames,lcid,rgD
ispId)     \
   (This)->lpVtbl ->
GetIDsOfNames(This,riid,rgszNames,cNames,lcid,rgDispId)

#define
ISMLog_Invoke(This,dispIdMember,riid,lcid,wFlags,pDispPara
ms,pVarResult,pExcepInfo,puArgErr)   \
   (This)->lpVtbl ->
Invoke(This,dispIdMember,riid,lcid,wFlags,pDispParams,pVarR
esult,pExcepInfo,puArgErr)


#define ISMLog_put_FileHeader(This,newVal)  \
   (This)->lpVtbl -> put_FileHeader(This,newVal)

#define ISMLog_WriteLine(This,szMsg)     \
   (This)->lpVtbl -> WriteLine(This,szMsg)

#define ISMLog_WriteField(This,szMsg)    \
   (This)->lpVtbl -> WriteField(This,szMsg)

#define ISMLog_put_FileName(This,newVal)    \
   (This)->lpVtbl -> put_FileName(This,newVal)

#define ISMLog_put_Append(This,newVal)\
   (This)->lpVtbl -> put_Append(This,newVal)

#endif /* COBJMACROS */

#endif    /* C style interface */

/* [helpstring][id][propput] */ HRESULT
STDMETHODCALLTYPE ISMLog_put_FileHeader_Proxy(
   ISMLog __RPC_FAR * This,
   /* [in] */ BSTR newVal);
```

```
void __RPC_STUB ISMLog_put_FileHeader_Stub(
   IRpcStubBuffer *This,
   IRpcChannelBuffer *_pRpcChannelBuffer,
   PRPC_MESSAGE _pRpcMessage,
   DWORD *_pdwStubPhase);

/* [helpstring][id] */ HRESULT STDMETHODCALLTYPE
ISMLog_WriteLine_Proxy(
   ISMLog __RPC_FAR * This,
   BSTR szMsg);


void __RPC_STUB ISMLog_WriteLine_Stub(
   IRpcStubBuffer *This,
   IRpcChannelBuffer *_pRpcChannelBuffer,
   PRPC_MESSAGE _pRpcMessage,
   DWORD *_pdwStubPhase);


/* [helpstring][id] */ HRESULT STDMETHODCALLTYPE
ISMLog_WriteField_Proxy(
   ISMLog __RPC_FAR * This,
   BSTR szMsg);


void __RPC_STUB ISMLog_WriteField_Stub(
   IRpcStubBuffer *This,
   IRpcChannelBuffer *_pRpcChannelBuffer,
   PRPC_MESSAGE _pRpcMessage,
   DWORD *_pdwStubPhase);


/* [helpstring][id][propput] */ HRESULT
STDMETHODCALLTYPE ISMLog_put_FileName_Proxy(
   ISMLog __RPC_FAR * This,
   /* [in] */ BSTR newVal);


void __RPC_STUB ISMLog_put_FileName_Stub(
   IRpcStubBuffer *This,
   IRpcChannelBuffer *_pRpcChannelBuffer,
   PRPC_MESSAGE _pRpcMessage,
   DWORD *_pdwStubPhase);


/* [helpstring][id][propput] */ HRESULT
STDMETHODCALLTYPE ISMLog_put_Append_Proxy(
   ISMLog __RPC_FAR * This,
   /* [in] */ BOOL newVal);


void __RPC_STUB ISMLog_put_Append_Stub(
   IRpcStubBuffer *This,
   IRpcChannelBuffer *_pRpcChannelBuffer,
   PRPC_MESSAGE _pRpcMessage,
   DWORD *_pdwStubPhase);

#endif    /* __ISMLog_INTERFACE_DEFINED__ */



#ifndef __LOGWRITERLib_LIBRARY_DEFINED__
#define __LOGWRITERLib_LIBRARY_DEFINED__
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
/* library LOGWRITERLib */
/* [helpstring][version][uuid] */


EXTERN_C const IID LIBID_LOGWRITERLib;

#ifndef ___ISMLogEvents_DISPINTERFACE_DEFINED__
#define ___ISMLogEvents_DISPINTERFACE_DEFINED__

/* dispinterface _ISMLogEvents */
/* [helpstring][uuid] */


EXTERN_C const IID DIID__ISMLogEvents;

#if defined(__cplusplus) && !defined(CINTERFACE)

   MIDL_INTERFACE("5AC75DB1-1936-11D3-BC2D-
00A0C90D2CA5")
   _ISMLogEvents : public IDispatch
   {
   };

#else      /* C style interface */

   typedef struct _ISMLogEventsVtbl
   {
      BEGIN_INTERFACE

      HRESULT ( STDMETHODCALLTYPE __RPC_FAR
*QueryInterface )(
         _ISMLogEvents __RPC_FAR * This,
         /* [in] */ REFIID riid,
         /* [iid_is][out] */ void __RPC_FAR *__RPC_FAR
*ppvObject);

      ULONG ( STDMETHODCALLTYPE __RPC_FAR
*AddRef )(
         _ISMLogEvents __RPC_FAR * This);

      ULONG ( STDMETHODCALLTYPE __RPC_FAR
*Release )(
         _ISMLogEvents __RPC_FAR * This);

      HRESULT ( STDMETHODCALLTYPE __RPC_FAR
*GetTypeInfoCount )(
         _ISMLogEvents __RPC_FAR * This,
         /* [out] */ UINT __RPC_FAR *pctinfo);

      HRESULT ( STDMETHODCALLTYPE __RPC_FAR
*GetTypeInfo )(
         _ISMLogEvents __RPC_FAR * This,
         /* [in] */ UINT iTInfo,
         /* [in] */ LCID lcid,
         /* [out] */ ITypeInfo __RPC_FAR *__RPC_FAR
*ppTInfo);

      HRESULT ( STDMETHODCALLTYPE __RPC_FAR
*GetIDsOfNames )(
         _ISMLogEvents __RPC_FAR * This,
         /* [in] */ REFIID riid,
         /* [size_is][in] */ LPOLESTR __RPC_FAR *rgszNames,
         /* [in] */ UINT cNames,
         /* [in] */ LCID lcid,
         /* [size_is][out] */ DISPID __RPC_FAR *rgDispId);

      /* [local] */ HRESULT ( STDMETHODCALLTYPE
__RPC_FAR *Invoke )(
         _ISMLogEvents __RPC_FAR * This,
         /* [in] */ DISPID dispIdMember,
         /* [in] */ REFIID riid,
         /* [in] */ LCID lcid,
         /* [in] */ WORD wFlags,
         /* [out][in] */ DISPPARAMS __RPC_FAR
*pDispParams,
         /* [out] */ VARIANT __RPC_FAR *pVarResult,
         /* [out] */ EXCEPINFO __RPC_FAR *pExcepInfo,
         /* [out] */ UINT __RPC_FAR *puArgErr);

      END_INTERFACE
   } _ISMLogEventsVtbl;

   interface _ISMLogEvents
   {
      CONST_VTBL struct _ISMLogEventsVtbl __RPC_FAR
*lpVtbl;
   };



#ifdef COBJMACROS


#define _ISMLogEvents_QueryInterface(This,riid,ppvObject) \
   (This)->lpVtbl -> QueryInterface(This,riid,ppvObject)

#define _ISMLogEvents_AddRef(This) \
   (This)->lpVtbl -> AddRef(This)

#define _ISMLogEvents_Release(This) \
   (This)->lpVtbl -> Release(This)


#define _ISMLogEvents_GetTypeInfoCount(This,pctinfo)      \
   (This)->lpVtbl -> GetTypeInfoCount(This,pctinfo)

#define _ISMLogEvents_GetTypeInfo(This,iTInfo,lcid,ppTInfo)
      \
   (This)->lpVtbl -> GetTypeInfo(This,iTInfo,lcid,ppTInfo)

#define
_ISMLogEvents_GetIDsOfNames(This,riid,rgszNames,cNames,l
cid,rgDispId)\
   (This)->lpVtbl ->
GetIDsOfNames(This,riid,rgszNames,cNames,lcid,rgDispId)

#define
_ISMLogEvents_Invoke(This,dispIdMember,riid,lcid,wFlags,pD
ispParams,pVarResult,pExcepInfo,puArgErr)      \
   (This)->lpVtbl ->
Invoke(This,dispIdMember,riid,lcid,wFlags,pDispParams,pVarR
esult,pExcepInfo,puArgErr)

#endif /* COBJMACROS */


#endif    /* C style interface */


#endif    /* ___ISMLogEvents_DISPINTERFACE_DEFINED__
*/
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
EXTERN_C const CLSID CLSID_SMLog;

#ifdef __cplusplus
class DECLSPEC_UUID("5AC75DB0-1936-11D3-BC2D-
00A0C90D2CA5")
SMLog;
#endif
#endif /* __LOGWRITERLib_LIBRARY_DEFINED__ */

unsigned long __RPC_USER  BSTR_UserSize(unsigned long
__RPC_FAR *,
unsigned long   , BSTR __RPC_FAR * );
unsigned char __RPC_FAR * __RPC_USER
BSTR_UserMarshal(  unsigned long __RPC_FAR *,
unsigned char __RPC_FAR *, BSTR __RPC_FAR * );
unsigned char __RPC_FAR * __RPC_USER
BSTR_UserUnmarshal(unsigned long __RPC_FAR *,
unsigned char __RPC_FAR *, BSTR __RPC_FAR * );
void     __RPC_USER  BSTR_UserFree(     unsigned long
__RPC_FAR *, BSTR __RPC_FAR * );

/* end of Additional Prototypes */

#ifdef __cplusplus
}
#endif
#endif
//  FILE:      LogWriter.cpp
///  FILE:      LogWriterCP.h
//            Microsoft TPC-H Kit Ver. 1.00
//            Copyright Microsoft, 1999
//            All Rights Reserved
//  PURPOSE:    Connection point implementation
//  Contact:    Reshma Tharamal (reshmat@microsoft.com)
#ifndef _LOGWRITERCP_H_
#define _LOGWRITERCP_H_

template <class T>
class CProxy_ISMLogEvents : public IConnectionPointImpl<T,
&DIID__ISMLogEvents, CComDynamicUnkArray>
{
      //Warning this class may be recreated by the wizard.
public:
};
#endif
//  FILE:      resource.h
//            Microsoft TPC-H Kit Ver. 1.00
//            Copyright Microsoft, 1999
//            All Rights Reserved
//  PURPOSE:    Resource file
//  Contact:    Reshma Tharamal (reshmat@microsoft.com)
//
//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.
// Used by LogWriter.rc
//
#define IDS_PROJNAME                    100
#define IDR_SMLOG                       101
// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE        201
#define _APS_NEXT_COMMAND_VALUE         32768
#define _APS_NEXT_CONTROL_VALUE         201
#define _APS_NEXT_SYMED_VALUE           102
#endif
#endif
```

```
//  FILE:      SMLog.cpp
//            Microsoft TPC-H Kit Ver. 1.00
//            Copyright Microsoft, 1999
//            All Rights Reserved
//
//
//  PURPOSE:    Implementation of CSMLog
//  Contact:    Reshma Tharamal (reshmat@microsoft.com)
//
// SMLog.cpp : Implementation of CSMLog
#include "stdafx.h"
#include <stdio.h>
#include "LogWriter.h"
#include "SMLog.h"

/////////////////////////////////////////////////////////////////////////////
// CSMLog

STDMETHODIMP
CSMLog::InterfaceSupportsErrorInfo(REFIID riid)
{
static const IID* arr[] =
{
&IID_ISMLog
};
for (int i=0; i < sizeof(arr) / sizeof(arr[0]); i++)
{
if (InlineIsEqualGUID(*arr[i],riid))
return S_OK;
}
return S_FALSE;
}

STDMETHODIMP CSMLog::WriteToFile(BSTR szMsg)
{
   // Writes the passed in string to the file
      _bstr_t szTempMsg(szMsg);
      return(Write((PBYTE)(LPSTR)szTempMsg,
SysStringLen(szMsg)));
}

HRESULT CSMLog::Init()
{
charszDrive[256];
charszDir[256];
charszLogDir[256];
HANDLE     hLogThread;
DWORD      dwThreadID;
_bstr_t szFile(m_szFile);
DWORD      lDisposition;

//create transaction log directory
_splitpath((LPCTSTR)szFile, szDrive, szDir, NULL, NULL);
_makepath(szLogDir, szDrive, szDir, NULL, NULL);
CreateDirectory(szLogDir, NULL);
   iBufferSize = WRITE_BUFFER_SIZE;
iBytesFreeInBuffer     = iBufferSize;

// use VirtualAlloc to get page aligned buffers //
for (int i=0;i<MAX_NUM_BUFFERS;i++)
{
// use VirtualAlloc to get page aligned buffers //
pBuffer[i] = (BYTE *)VirtualAlloc(NULL, iBufferSize,
MEM_COMMIT, PAGE_READWRITE );
if (pBuffer[i] == NULL)
return RaiseSystemError();
}
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
	iActiveBuffer = 0;
	pCurrent = pBuffer[iActiveBuffer];

	lDisposition = m_bAppend ? OPEN_ALWAYS :
CREATE_ALWAYS;
	m_hTxnFile = CreateFile((LPCTSTR)szFile,
GENERIC_WRITE, FILE_SHARE_READ,
		NULL, lDisposition, FILE_ATTRIBUTE_NORMAL,
NULL);
	if ( m_hTxnFile == INVALID_HANDLE_VALUE )
		return (RaiseSystemError());

	if (m_bAppend)
		if ( SetFilePointer(m_hTxnFile, 0, NULL, FILE_END)
== ERR_SET_FILE_POINTER )
			return (RaiseSystemError());

	hIoComplete  = CreateEvent(NULL, TRUE, TRUE, NULL);
	if ( hIoComplete == NULL)
		return RaiseSystemError();

	hLogFileIo = CreateEvent(NULL, FALSE, FALSE, NULL);
	if ( hLogFileIo == NULL)
		return RaiseSystemError();

	hLogThread = CreateThread( NULL, 0,
(LPTHREAD_START_ROUTINE)LogFileIO, this, 0,
&dwThreadID );
	if (hLogThread == NULL)
		return RaiseSystemError();

	if (m_szHeader != NULL)
		WriteLine(m_szHeader);

	return S_OK;
}

void CSMLog::LogFileIO(void *ptr)
{
	unsigned long	BytesWritten;
	CSMLog *p=(CSMLog *)ptr;

	while( TRUE )
	{
	WaitForSingleObject(p->hLogFileIo, INFINITE);
	if ( p->m_hTxnFile == INVALID_HANDLE_VALUE )
		break;

	// do synchronous (blocking) write to log file
	if ( !WriteFile(p->m_hTxnFile, p->pBuffer[p->iIoBuffer],p-
>iWriteSize,&BytesWritten,NULL) )
	{
	// set error code in this thread, but don't throw an exception
	// because no one will catch it.
	p->dwError = GetLastError();

	}
	SetEvent(p->hIoComplete);
	}

	SetEvent(p->hIoComplete);
}
```

```
HRESULT CSMLog::Write(BYTE *ptr, DWORD iSize)
{
	int		StartPos, Remainder;
	int		dwErrorLocal = 0;

	if (!m_bInitialized)
	{
		HRESULT hr = Init();
		m_bInitialized = TRUE;

		if (FAILED(hr))
			return hr;
	}

	if ( m_hTxnFile == INVALID_HANDLE_VALUE )
		return S_OK;

	if ( iBytesFreeInBuffer >= iSize )
	{
		memcpy(pCurrent, ptr, iSize);
		pCurrent += iSize;
		iBytesFreeInBuffer -= iSize;
	}
	else
	{
		// We don't expect to ever have to wait here, but just in
case...
		WaitForSingleObject(hIoComplete, INFINITE);

		// check for an error from the log writer thread
		if (dwError != 0)
		{
			SetLastError(dwError);
			return RaiseSystemError();
		}

		assert( iSize <= iBufferSize );
		memcpy(pCurrent, ptr, iBytesFreeInBuffer);
		StartPos = iBytesFreeInBuffer;
		Remainder = iSize - iBytesFreeInBuffer;

		// trigger an IO on the current buffer and roll to the next
buffer
		iIoBuffer = iActiveBuffer;
		iWriteSize = iBufferSize;
		ResetEvent(hIoComplete);
		SetEvent( hLogFileIo );		// wake up IO writer

		iActiveBuffer = (iActiveBuffer+1) %
MAX_NUM_BUFFERS;
		pCurrent = pBuffer[iActiveBuffer];

		memcpy(pCurrent, ((BYTE *)ptr+StartPos),
Remainder);
		pCurrent += Remainder;
		iBytesFreeInBuffer = iBufferSize - Remainder;
	}

	return S_OK;
}
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
void CSMLog::CloseLogFile(void)
{

if ( m_hTxnFile != INVALID_HANDLE_VALUE )
{
	if ( iBytesFreeInBuffer < iBufferSize )
	{
	WaitForSingleObject(hIoComplete, INFINITE);
	ResetEvent(hIoComplete);

	// check for an error from the log writer thread
	if (dwError != 0)
	{
	SetLastError( dwError );
	goto exit_SpinLock;
	}

	//zero fill remainder of buffer
	ZeroMemory(pCurrent, iBytesFreeInBuffer);

	iIoBuffer = iActiveBuffer;
	iWriteSize = iBufferSize - iBytesFreeInBuffer;
	SetEvent(hLogFileIo); // wake up IO writer
	}

	WaitForSingleObject(hIoComplete, INFINITE);
// check for an error from the log writer thread if (dwError != 0)
	goto exit_SpinLock;

	pCurrent = pBuffer[iActiveBuffer];
	ZeroMemory(pCurrent, iBufferSize);
	iIoBuffer = iActiveBuffer;

	CloseHandle(m_hTxnFile);
	m_hTxnFile = INVALID_HANDLE_VALUE;	//handle to
open transaction log file

// wake up IO writer one more time for it to terminate
	ResetEvent(hIoComplete);
	SetEvent(hLogFileIo);	// wake up IO writer
		WaitForSingleObject(hIoComplete, INFINITE);
	}

exit_SpinLock:

	if (dwError != 0)
	{
		if (m_hTxnFile != INVALID_HANDLE_VALUE)
	{
	CloseHandle(m_hTxnFile);
	m_hTxnFile = INVALID_HANDLE_VALUE;
	}

	SetLastError( dwError );
// TODO: Don't know yet what to do with an error on the file
close, since this function is called by the desctructor (which does
 not return a value)
	//throw new CSystemErr( CSystemErr::eWriteFile,
"CTxnLog::CloseTransactionLogFile" );
}

}
```

```
// Wrapper function that raises an error if a Windows Api fails
STDMETHODIMP CSMLog::RaiseSystemError(void)
{
	char s[ERR_BUFFER_SIZE];
	long c;
	DWORD e;

	e = GetLastError();

	c = sprintf(s, "Error code: %ld. ", e);
	c =
FormatMessage(FORMAT_MESSAGE_FROM_SYSTEM |
FORMAT_MESSAGE_IGNORE_INSERTS,
		NULL, e, 0, s + c, sizeof(s) - c, NULL);

	return Error(s, 0, NULL, GUID_NULL);
}

//DEL STDMETHODIMP CSMLog::get_FileName(BSTR
*pVal)
//DEL {
//DEL	*pVal = m_szFile;
//DEL	return S_OK;
//DEL }

STDMETHODIMP CSMLog::put_FileName(BSTR newVal)
{
	m_szFile = SysAllocString(newVal);

	return S_OK;
}

//DEL STDMETHODIMP CSMLog::get_FileHeader(BSTR
*pVal)
//DEL {
//DEL	*pVal = m_szHeader;
//DEL	return S_OK;
//DEL }

STDMETHODIMP CSMLog::put_FileHeader(BSTR newVal)
{
	m_szHeader = SysAllocString(newVal);
	return S_OK;
}

STDMETHODIMP CSMLog::WriteLine(BSTR szMsg)
{
	_bstr_t szTmp(szMsg);

	szTmp += "\r";
	szTmp += "\n";
	return(WriteToFile(szTmp));
}

STDMETHODIMP CSMLog::WriteField(BSTR szMsg)
{
	return(WriteToFile(szMsg));
}

//DEL STDMETHODIMP CSMLog::get_Append(BOOL *pVal)
//DEL {
//DEL	*pVal = m_bAppend;
//DEL	return S_OK;
//DEL }
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
STDMETHODIMP CSMLog::put_Append(BOOL newVal)
{
	m_bAppend = newVal;
	return S_OK;
}
//  FILE:	SMExecute.h
//		Microsoft TPC-H Kit Ver. 1.00
//		Copyright Microsoft, 1999
//		All Rights Reserved
//
//  PURPOSE:	Declaration of the CSMLog
//  Contact:	Reshma Tharamal (reshmat@microsoft.com)
//
// SMLog.h : Declaration of the CSMLog

#ifndef __SMLOG_H_
#define __SMLOG_H_
#include "resource.h"	// main symbols
#include "assert.h"
#include "comdef.h"
#include "LogWriterCP.h"
#define  WRITE_BUFFER_SIZE	8*1024
#define  MAX_NUM_BUFFERS	2
#define ERR_BUFFER_SIZE		512
#define ERR_SET_FILE_POINTER	0xFFFFFFFF

/////////////////////////////////////////////////////////////////////////////
// CSMLog
class ATL_NO_VTABLE CSMLog :
public CComObjectRootEx<CComSingleThreadModel>,
public CComCoClass<CSMLog, &CLSID_SMLog>,
public ISupportErrorInfo,
public IConnectionPointContainerImpl<CSMLog>,
public IDispatchImpl<ISMLog, &IID_ISMLog,
&LIBID_LOGWRITERLib>,
public CProxy_ISMLogEvents< CSMLog >
{
public:
CSMLog()
{
	m_bAppend = FALSE;
	m_bInitialized = FALSE;
	m_hTxnFile = INVALID_HANDLE_VALUE;
	hIoComplete = NULL;
	hLogFileIo = NULL;
for (int i=0;i<MAX_NUM_BUFFERS;i++)
	pBuffer[i] = NULL;
	pCurrent = NULL;
	m_szFile = NULL;
	m_szHeader = NULL;
	dwError = 0;
	}
private:
BOOL	m_bAppend;
BOOL	m_bInitialized;
HANDLE m_hTxnFile; //handle to log file
HANDLE hIoComplete; //event to signify that there are no
pending IOs
HANDLE hLogFileIo; //event to signal the IO thread to write the
inactive buffer
BYTE	*pBuffer[MAX_NUM_BUFFERS];
BYTE	*pCurrent;  //ptr to current buffer
int	iActiveBuffer;//indicates which buffer is active: 0 or 1int
	iIoBuffer;	//buffer for any pending IO operation
DWORD	dwError;
DWORD	iBufferSize; //buffer allocated size
DWORD	iBytesFreeInBuffer; // bytes available for use in
buffer
DWORD	iWriteSize;	//bytes to write to file
BSTR  m_szFile;		//name of the file to write to
BSTR  m_szHeader; //header to be printed to the file (optional)
private:
static void		LogFileIO(void *p);
HRESULT	Write(BYTE *ptr, DWORD iSize);
STDMETHODIMP	WriteToFile(BSTR szMsg);
voidCloseLogFile(void);
STDMETHODIMP	RaiseSystemError(void);
};

#endif //__SMLOG_H_
//  FILE:	stdafx.cpp
//		Microsoft TPC-H Kit Ver. 1.00
//		Copyright Microsoft, 1999
//		All Rights Reserved
//  PURPOSE:	source file that includes just the standard
includes
//  Contact:	Reshma Tharamal (reshmat@microsoft.com)
// stdafx.cpp : source file that includes just the standard includes
//  stdafx.pch will be the pre-compiled header
//  stdafx.obj will contain the pre-compiled type information

#include "stdafx.h"
#ifdef _ATL_STATIC_REGISTRY
#include <statreg.h>
#include <statreg.cpp>
#endif
#include <atlimpl.cpp>
//  FILE:	stdafx.h
//		Microsoft TPC-H Kit Ver. 1.00
//		Copyright Microsoft, 1999
//		All Rights Reserved
//  PURPOSE:	include file for standard system include files,
//	or project specific include files that are used frequently,
//	but are changed infrequently
//  Contact:	Reshma Tharamal (reshmat@microsoft.com)
//
#if
!defined(AFX_STDAFX_H__5AC75DA4_1936_11D3_BC2D_
00A0C90D2CA5__INCLUDED_)
#define
AFX_STDAFX_H__5AC75DA4_1936_11D3_BC2D_00A0C90
D2CA5__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#define STRICT
#ifndef _WIN32_WINNT
#define _WIN32_WINNT 0x0400
#endif
#define _ATL_APARTMENT_THREADED

#include <atlbase.h>
//You may derive a class from CComModule and use it if you
//want to override something, but do not change the name of
//_Module extern CComModule _Module;
#include <atlcom.h>
//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations
immediately before the previous line.
#endif //
!defined(AFX_STDAFX_H__5AC75DA4_1936_11D3_BC2D_
00A0C90D2CA5__INCLUDED)
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
//  FILE:      Execute.cpp
//             Microsoft TPC-H Kit Ver. 1.00
//             Copyright Microsoft, 1999
//             All Rights Reserved
//
//  PURPOSE:    Implementation of CExecute.
//  Contact:    Reshma Tharamal (reshmat@microsoft.com)
//
#include "stdafx.h"

#include "ExecuteDll.h"
#include "SMExecute.h"
#include "Execute.h"

extern SQLHENV henv;

extern SM_Connection_Info     *p_Connections;        // Pointer to
open connections
extern int                    iConnectionCount;     // Number of open
connections
extern CRITICAL_SECTION       hConnections;         //
Critical section to serialize
                                        // access to available connections

#ifdef _TPCH_AUDIT
      extern FILE *pfLogFile;          // Log file containing
timestamps
      extern CRITICAL_SECTION hLogFileWrite;    // Handle to
critical section
#endif

//////////////////////////////////////////////////////////////////////////////
// CExecute

char * g_szOdbcOps[] = {
      "SQLAllocHandle",
      "SQLDriverConnect",
      "SQLExecDirect",
      "SQLSetStmtAttr",
      "SQLCancel",
      "SQLNumResultCols",
      "SQLDescribeCol",
      "SQLColAttribute",
      "SQLFetch",
      "SQLGetData",
      "SQLRowCount",
      "SQLMoreResults",
      "SQLBindCol"
      };

char * CExecError::m_szExecErrorDesc[] = {
            "Connection is already in use."
      };
STDMETHODIMP
CExecute::InterfaceSupportsErrorInfo(REFIID riid)
{
      static const IID* arr[] =
      {
            &IID_IExecute
      };
      for (int i=0; i < sizeof(arr) / sizeof(arr[0]); i++)
      {
            if (InlineIsEqualGUID(*arr[i],riid))
                  return S_OK;
      }
      return S_FALSE;
}
STDMETHODIMP CExecute::put_OutputFile(BSTR newVal)
{
      assert(m_pOutputFile);
      m_OutputFile = newVal;
      HRESULT    hr = m_pOutputFile->put_FileName(newVal);
      if FAILED(hr)
      {
            m_pOutputFile->Release();
            m_pOutputFile = NULL;
      }
      return hr;
}
//DEL STDMETHODIMP CExecute::put_LogFile(BSTR
newVal)
//DEL {
//DEL    assert(m_pLogFile);
//DEL
//DEL    m_pLogFile->put_FileName(newVal);
//DEL    return S_OK;
//DEL }

STDMETHODIMP CExecute::put_ErrorFile(BSTR newVal)
{
      assert(m_pErrorFile);
      m_ErrorFile = newVal;

      HRESULT    hr = m_pErrorFile->put_FileName(newVal);
      if FAILED(hr)
      {
            m_pErrorFile->Release();
            m_pErrorFile = NULL;
      }
      return hr;
}
STDMETHODIMP CExecute::DoExecute(BSTR szCommand,
BSTR szExecutionDtls, ExecutionType ExecMethod, \
 BOOL bNoCount, BOOL bNoExecute, BOOL bParseOnly,
BOOL bQuotedIds, \
BOOL bAnsiNulls, BOOL bShowQP, BOOL bStatsTime, BOOL
bStatsIO, \
long lRowCount, long lQueryTmout, BSTR szConnection)
{
HANDLE          hThrd;
DWORD           tid;

  _CrtSetReportFile(_CRT_WARN,
_CRTDBG_FILE_STDOUT);
      m_szCommand = szCommand;
      m_szExecDtls = szExecutionDtls;

      m_ExecMthd = ExecMethod;
      if (m_ExecMthd == execODBC)
   {
       m_bNoCount = bNoCount;
      m_bNoExecute = bNoExecute;
      m_bParseOnly = bParseOnly;
      m_bQuotedIds = bQuotedIds;
      m_bAnsiNulls = bAnsiNulls;
      m_bShowQP = bShowQP;
      m_bStatsTime = bStatsTime;
      m_bStatsIO = bStatsIO;
      m_lRowCount = lRowCount;
      m_lQueryTmout = lQueryTmout;
      m_szConnection = szConnection;
   }
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
	if((hThrd = CreateThread( 0, 0,
(LPTHREAD_START_ROUTINE)ExecutionThread,
	this, 0, &tid)) == NULL)
	return(RaiseSystemError());
	CloseHandle(hThrd);
	return S_OK;
}

STDMETHODIMP CExecute::Abort()
{
	if (m_ExecMthd == execShell)
		return(AbortShell());
	else
		return(AbortODBC());
}
void ExecutionThread(LPVOID lpParameter)
{
	CExecute	*MyExecute = (CExecute*)lpParameter;
	MyExecute->m_tElapsedTime = 0;
	GetLocalTime(&MyExecute->m_tStartTime);
	MyExecute->PostMessage(WM_TASK_START, 0, 0);

#ifdef _TPCH_AUDIT
	char	szBuffer[MAXLOGCMDBUF];
	char	szFmt[MAXBUFLEN];

	sprintf(szFmt, "Start Step: '%%.%ds' at '%d/%d/%d
%d:%d:%d:%d'\n",
MAXLOGCMDLEN,
MyExecute->m_tStartTime.wMonth, MyExecute-
>m_tStartTime.wDay,
MyExecute->m_tStartTime.wYear, MyExecute-
>m_tStartTime.wHour,
		MyExecute->m_tStartTime.wMinute, MyExecute-
>m_tStartTime.wSecond,
		MyExecute->m_tStartTime.wMilliseconds);
if (MyExecute->m_ExecMthd == execShell)
	WriteFileToTpchLog((LPSTR)MyExecute-
>m_szCommand, szFmt);
else
{
sprintf(szBuffer, szFmt, (LPSTR)MyExecute->m_szCommand);
WriteToTpchLog(szBuffer);
}
#endif

	// Initialize the run status for the step to running. The
completion status for
	// the step will be initialized by the Shell and ODBC
execution functions.
   MyExecute->m_StepStatus = gintRunning;
if (MyExecute->m_ExecMthd == execShell)
MyExecute->m_tElapsedTime = MyExecute->ExecuteShell();
else
MyExecute->m_tElapsedTime = MyExecute->ExecuteODBC();
// Close the output, log and error files
if (MyExecute->m_pOutputFile)
	MyExecute->m_pOutputFile->Release();
	MyExecute->m_pOutputFile = NULL;

	MyExecute->m_ExecTime = NULL;
	GetLocalTime(&MyExecute->m_tEndTime);
#ifdef _TPCH_AUDIT
	sprintf(szFmt, "Complete Step: '%%.%ds' at '%d/%d/%d
%d:%d:%d:%d'\n",
	MAXLOGCMDLEN,
	MyExecute->m_tEndTime.wMonth, MyExecute-
>m_tEndTime.wDay,
	MyExecute->m_tEndTime.wYear, MyExecute-
>m_tEndTime.wHour,
	MyExecute->m_tEndTime.wMinute, MyExecute-
>m_tEndTime.wSecond,
	MyExecute->m_tEndTime.wMilliseconds);
	if (MyExecute->m_ExecMthd == execShell)
	WriteFileToTpchLog((LPSTR)MyExecute-
>m_szCommand, szFmt);
	else
	{
	sprintf(szBuffer, szFmt, (LPSTR)MyExecute-
>m_szCommand);
	WriteToTpchLog(szBuffer);
	}
#endif

MyExecute->PostMessage(WM_TASK_FINISH, 0, 0);

return;
}
#ifdef _TPCH_AUDIT
void WriteFileToTpchLog(LPSTR szFile, LPSTR szFmt)
{
	// Reads a maximum of MAXLOGCMDBUF characters
from the command file and writes it to the log
	FILE	*fpCmd;
	int	iRead;
	charszBuf[MAXLOGCMDBUF];
	charszCmd[MAXLOGCMDLEN];
	if ( pfLogFile != NULL )
	{
		if ( (fpCmd = fopen(szFile, FILE_ACCESS_READ)) !=
NULL)
{
iRead = fread(szCmd, sizeof(char), sizeof(szCmd) / sizeof(char),
fpCmd);
	if (iRead < MAXLOGCMDLEN)
	szCmd[iRead] = '\0';
	else
	szCmd[MAXLOGCMDLEN - 1] = '\0';
	sprintf(szBuf, szFmt, szCmd);
	WriteToTpchLog(szBuf);
	fclose(fpCmd);
	}
}
}
void WriteToTpchLog(char *szMsg)
{
	if (pfLogFile != NULL)
	{
		EnterCriticalSection(&hLogFileWrite);
		fprintf(pfLogFile, szMsg);
		LeaveCriticalSection(&hLogFileWrite);
	}

	return;
}
#endif
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
TC_TIME CExecute::ExecuteShell()
{
	STARTUPINFOA		Start;
	PROCESS_INFORMATION		proc;
	DWORD			exitCode;
	TC_TIME			tElapsed = 0;
	_bstr_t		szCommand("cmd /c ");
	LPSTR		szStartDir;
	CURRENCY		Elapsed;

	szCommand += m_szCommand;

	// Redirect output and error information
	szCommand += " > " + m_OutputFile + " 2> " +
m_ErrorFile;

	// Initialize the STARTUPINFO structure:
	memset(&Start, 0, sizeof(STARTUPINFOA)) ;
	Start.cb		= sizeof(Start);
	Start.dwFlags		= STARTF_USESHOWWINDOW;
	Start.wShowWindow	= SW_SHOWMINNOACTIVE;

	memset(&proc, 0, sizeof(PROCESS_INFORMATION)) ;

	szStartDir = strcmp((LPCTSTR)m_szExecDtls, "") == 0 ?
NULL : (LPSTR)m_szExecDtls;

	m_ExecTime->Start();

	// Start the shelled application:
	if (!CreateProcessA( NULL, (LPSTR)szCommand, NULL,
NULL, FALSE,
		NORMAL_PRIORITY_CLASS, NULL, szStartDir,
&Start, &proc ))
	{
		m_StepStatus = gintFailed;
		LogSystemError(m_pErrorFile);

		m_ExecTime->Stop(&Elapsed);
		return((TC_TIME)Elapsed.int64);
	}

	m_hHandle = proc.hProcess;
	// Give the process time to execute and finish
	WaitForSingleObject(m_hHandle, INFINITE);
	m_ExecTime->Stop(&Elapsed);

	if (!GetExitCodeProcess(m_hHandle, &exitCode))
	{
		m_StepStatus = gintFailed;
		LogSystemError(m_pErrorFile);
	}
	else
		m_StepStatus = gintComplete;

	// Close all open handles to the shelled process
	CloseHandle(m_hHandle);

	return((TC_TIME)Elapsed.int64);
}
```

```
STDMETHODIMP CExecute::AbortShell()
{
	if (m_hHandle != SQL_NULL_HSTMT)
		if (!TerminateProcess(m_hHandle, 0))
			return(RaiseSystemError());

	return(S_OK);
}

TC_TIME CExecute::ExecuteODBC()
{
	TC_TIME		tElapsed = 0;
	HDBC		m_hdbc;
	SQLRETURN		rc;
	LPSTR		szCmd;
	CURRENCY		Elapsed;
	BOOL		bDoConnect = FALSE;

	// ODBC specific initialization
	m_hdbc = SQL_NULL_HDBC;

	try
	{
		// Allocate a new connection if we are creating a
dynamic connection or if
		// the named connection doesn't exist
		InitializeConnection(&m_hdbc, &bDoConnect);

		// Ensure that the connection is valid.
#ifdef _DEBUG
		_CrtDbgReport(_CRT_WARN, NULL, 0, NULL,
"Executing SQLDriverConnect.\n");
#endif

		if (bDoConnect)
		{
			// Allocate connection handle, open a connection
and set connection attributes.
#ifdef _DEBUG
			_CrtDbgReport(_CRT_WARN, NULL, 0, NULL,
"Executing SQLDriverConnect.\n");
#endif

			if (m_bAbort)
				return(tElapsed);

			// Connect to the server using the passed in
connection string
			rc = SQLDriverConnect(m_hdbc, NULL,
				(unsigned char *)(LPSTR)m_szExecDtls,
SQL_NTS,
				NULL, 0, NULL,
SQL_DRIVER_NOPROMPT);
			HandleODBCError(rc, SQL_HANDLE_DBC,
m_hdbc, SMSQLDriverConnect);
		}

		ReConnectDeadConnection(&m_hdbc);

#ifdef _DEBUG
		_CrtDbgReport(_CRT_WARN, NULL, 0, NULL,
"Executing SQLAllocHandle for hdbc.\n");
#endif
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
if (!m_bAbort && (rc =
SQLAllocHandle(SQL_HANDLE_STMT, m_hdbc,
&m_hHandle)) != SQL_SUCCESS)
HandleODBCError(rc, SQL_HANDLE_DBC, m_hdbc,
SMSQLAllocHandle);

// Set connection attributes if any have been modified from the
default values
if (m_lRowCount > 0)
{
char      szConnOptions[512];
      sprintf(szConnOptions, "SET ROWCOUNT %d ",
m_lRowCount);
      SetConnectionOption(szConnOptions, &m_hdbc);
      }

      if (m_bQuotedIds)
      SetConnectionOption("SET QUOTED_IDENTIFIER ON ",
&m_hdbc);

      if (!m_bAnsiNulls)
      SetConnectionOption("SET ANSI_NULL_DFLT_OFF ON
", &m_hdbc);

      if (!m_bAbort && m_lQueryTmout > 0)
      {
#ifdef _DEBUG
      _CrtDbgReport(_CRT_WARN, NULL, 0, NULL,
"Executing SQLSetStmtAttr.\n");
#endif
// Set the query timeout on the statement handle
rc = SQLSetStmtAttr(m_hHandle,
SQL_ATTR_QUERY_TIMEOUT, &m_lQueryTmout,
SQL_IS_UINTEGER);
HandleODBCError(rc, SQL_HANDLE_STMT, m_hHandle,
SMSQLSetStmtAttr);
}

            if (m_bNoExecute)
                  SetConnectionOption("SET NOEXEC ON ",
&m_hdbc);
else if (m_bParseOnly)
SetConnectionOption("SET PARSEONLY ON ", &m_hdbc);
else if (m_bShowQP)
// Important to ensure that this is the last connection attributes
being set - otherwise showplans are generated for all remaining
SET statements  SetConnectionOption("SET
SHOWPLAN_TEXT ON ", &m_hdbc);
else
{
if (m_bNoCount)
SetConnectionOption("SET NOCOUNT ON ", &m_hdbc);

if (m_bStatsIO)
SetConnectionOption("SET STATISTICS IO ON ", &m_hdbc);
// Important to ensure that this is the last connection attributes
being set - // otherwise timing statistics are generated for all
remaining SET statements if (m_bStatsTime)
SetConnectionOption("SET STATISTICS TIME ON ",
&m_hdbc);
}

m_szCmd = (LPSTR)m_szCommand;
m_ExecTime->Start();

while ((szCmd = NextCmdInBatch((LPSTR)m_szCommand)) !=
NULL && !m_bAbort)
{
#ifdef _DEBUG
            _CrtDbgReport(_CRT_WARN, NULL, 0, NULL,
"Executing SQLExecDirect.\n");
#endif

      // Execute the ODBC command
      rc = SQLExecDirect(m_hHandle, (unsigned char *)szCmd,
SQL_NTS);
      HandleODBCError(rc, SQL_HANDLE_STMT, m_hHandle,
SMSQLExecDirect);

      free(szCmd);

      // Call a procedure to log the results to the output file
      ProcessResultsets();

            }
            m_ExecTime->Stop(&Elapsed);

            ResetConnectionProperties(&m_hdbc);
      }
      catch(CODBCError *pErr)
      {
            m_StepStatus = gintFailed;
            delete pErr;
      }
      catch(CExecError *pErr)
      {
            m_StepStatus = gintFailed;
            pErr->LogErrors(this);
            delete pErr;
      }

      ODBCCleanup(&m_hdbc, &m_hHandle);

      if (m_StepStatus != gintFailed)
            m_StepStatus = gintComplete;

      return((DWORD)Elapsed.int64);
}

void CExecute::InitializeConnection(HDBC *phdbc, BOOL
*pbDoConnect)
{
      SQLRETURN                  rc;

      *pbDoConnect = TRUE;

      if (IsDynamicConnection())
      {
#ifdef _DEBUG
            _CrtDbgReport(_CRT_WARN, NULL, 0, NULL,
"Executing SQLAllocHandle for m_hdbc.\n");
#endif

            rc = SQLAllocHandle(SQL_HANDLE_DBC, henv,
phdbc);
            HandleODBCError(rc, SQL_HANDLE_ENV, henv,
SMSQLAllocHandle);

            return;
      }
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
	EnterCriticalSection(&hConnections);
	// Returns the connection handle if the connection,
m_szConnection, exists
	for (m_iConnectionIndex = iConnectionCount - 1;
m_iConnectionIndex >= 0; m_iConnectionIndex--)
	{
		if (!strcmp( (p_Connections + m_iConnectionIndex)-
>szConnectionName,
			(LPSTR)m_szConnection))
		{
			if (!(p_Connections + m_iConnectionIndex)-
>bInUse)
			{
				*phdbc = (p_Connections +
m_iConnectionIndex)->hdbc;
				(p_Connections + m_iConnectionIndex)-
>bInUse = TRUE;

				*pbDoConnect = FALSE;
				break;
			}
			else
			{
				LeaveCriticalSection(&hConnections);

				throw new
CExecError(CExecError::SM_ERR_CONN_IN_USE);
			}
		}
	}

	if (m_iConnectionIndex < 0)
	{
		// Connection was not found. Allocate connection
handle and add it to list of
		// available connections.
#ifdef _DEBUG
		_CrtDbgReport(_CRT_WARN, NULL, 0, NULL,
"Executing SQLAllocHandle for m_hdbc.\n");
#endif

	rc = SQLAllocHandle(SQL_HANDLE_DBC, henv, phdbc);
	HandleODBCError(rc, SQL_HANDLE_ENV, henv,
SMSQLAllocHandle);

	m_iConnectionIndex = iConnectionCount++;

	p_Connections = (SM_Connection_Info
*)realloc(p_Connections,
	iConnectionCount * sizeof(SM_Connection_Info));
	strcpy((p_Connections + m_iConnectionIndex)-
>szConnectionName, (LPSTR)m_szConnection);
	(p_Connections + m_iConnectionIndex)->hdbc = *phdbc;
	(p_Connections + m_iConnectionIndex)->bInUse = TRUE;
	}
	LeaveCriticalSection(&hConnections);

	return;
}
```

```
void CExecute::ReConnectDeadConnection(HDBC *phdbc)
{
	SQLRETURN		rc;
	SQLUINTEGER		uConnDead;

	// Connect to the server using the passed in connection string
	rc = SQLGetConnectAttr(*phdbc,
SQL_ATTR_CONNECTION_DEAD,
		&uConnDead, SQL_IS_UINTEGER, NULL);
	HandleODBCError(rc, SQL_HANDLE_DBC, *phdbc,
SMSQLDriverConnect);

	if (uConnDead == SQL_CD_TRUE)
	{
		// Cleanup the old connection and re-connect.
#ifdef _DEBUG
		_CrtDbgReport(_CRT_WARN, NULL, 0, NULL,
"Executing SQLDisconnect.\n");
#endif
		rc = SQLDisconnect(*phdbc);
#ifdef _DEBUG
		_CrtDbgReport(_CRT_WARN, NULL, 0, NULL,
"Executing SQLFreeHandle for hdbc.\n");
#endif
		SQLFreeHandle(SQL_HANDLE_DBC, *phdbc);
		*phdbc = SQL_NULL_HDBC;

#ifdef _DEBUG
		_CrtDbgReport(_CRT_WARN, NULL, 0, NULL,
"Executing SQLDriverConnect.\n");
#endif

		rc = SQLAllocHandle(SQL_HANDLE_DBC, henv,
phdbc);
		HandleODBCError(rc, SQL_HANDLE_ENV, henv,
SMSQLAllocHandle);

		// Connect to the server using the passed in connection
string
		rc = SQLDriverConnect(*phdbc, NULL,
			(unsigned char *)(LPSTR)m_szExecDtls,
SQL_NTS,
			NULL, 0, NULL, SQL_DRIVER_NOPROMPT);
		HandleODBCError(rc, SQL_HANDLE_DBC, *phdbc,
SMSQLDriverConnect);
	}

	return;
}
void CExecute::ResetConnectionUsage()
{
	if(m_iConnectionIndex >= 0 && m_iConnectionIndex <
iConnectionCount)
	{
		EnterCriticalSection(&hConnections);
		(p_Connections + m_iConnectionIndex)->bInUse =
FALSE;
		LeaveCriticalSection(&hConnections);
	}

	return;
}
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
void CExecute::ResetConnectionProperties(HDBC *p_hdbc)
{
	SQLRETURN		rc;

// Reset connection attributes if any have been modified from the
default values
if (m_bNoExecute)
SetConnectionOption("SET NOEXEC OFF ", p_hdbc);
else if (m_bParseOnly)
SetConnectionOption("SET PARSEONLY OFF ", p_hdbc);
else if (m_bShowQP)
// Reset connection attributes in reverse order
SetConnectionOption("SET SHOWPLAN_TEXT OFF ",
p_hdbc);
else
{
// Reset connection attributes in reverse order
if (m_bStatsTime)
SetConnectionOption("SET STATISTICS TIME OFF ",
p_hdbc);

		if (m_bNoCount)
			SetConnectionOption("SET NOCOUNT OFF ",
p_hdbc);

		if (m_bStatsIO)
			SetConnectionOption("SET STATISTICS IO OFF
", p_hdbc);
	}

	if (m_lRowCount > 0)
	{
		char		szConnOptions[512];

		sprintf(szConnOptions, "SET ROWCOUNT 0 ");
		SetConnectionOption(szConnOptions, p_hdbc);
	}

	if (m_bQuotedIds)
		SetConnectionOption("SET QUOTED_IDENTIFIER
OFF ", p_hdbc);

	if (!m_bAnsiNulls)
		SetConnectionOption("SET ANSI_NULL_DFLT_OFF
OFF ", p_hdbc);

	if (m_lQueryTmout > 0)
	{
		SQLUINTEGER	lQueryTmout = 0;

#ifdef _DEBUG
		_CrtDbgReport(_CRT_WARN, NULL, 0, NULL,
"Executing SQLSetStmtAttr.\n");
#endif
// Set the query timeout on the statement handle
		rc = SQLSetStmtAttr(m_hHandle,
SQL_ATTR_QUERY_TIMEOUT, &lQueryTmout,
			SQL_IS_UINTEGER);
		HandleODBCError(rc, SQL_HANDLE_STMT,
m_hHandle, SMSQLSetStmtAttr);
	}

	return;
}

LPSTR CExecute::NextCmdInBatch(LPSTR szBatch)
{
	LPSTR	szCmd, szSeparator, szStart;
	charszNext;

	szStart = m_szCmd;

	while ( (szSeparator = strstr(szStart, CMD_SEPARATOR))
!= NULL)
	{
		szNext = *(szSeparator + strlen(CMD_SEPARATOR));
		if ( szNext == '\n' || szNext == '\r' || szNext == '\0')
			break;
		else
			szStart = szSeparator +
strlen(CMD_SEPARATOR);
	}

	if (!szSeparator)
	{
		// No more GO's
		if (strlen(m_szCmd) > 0)
		{
			szCmd = (LPSTR)malloc(strlen(m_szCmd) + 1);
			strcpy(szCmd, m_szCmd);
			m_szCmd += strlen(m_szCmd);
		}
		else
			szCmd = NULL;
	}
	else if (szSeparator - m_szCmd > 0)
	{
// Strip the succeeding newline
szCmd = (LPSTR)malloc(szSeparator - m_szCmd);
strncpy(szCmd, m_szCmd, szSeparator - m_szCmd - 1);
*(szCmd + (szSeparator - m_szCmd - 1)) = '\0';
m_szCmd += szSeparator - m_szCmd +
strlen(CMD_SEPARATOR);
		if ( szNext == '\n' || szNext == '\r')
			m_szCmd += 1;
	}
	else
		szCmd = NULL;

	return(szCmd);
}

void CExecute::SetConnectionOption(LPSTR szConn, HDBC
*pHdbc)
{
	// Executes the passed in connection options 'set' statement.
Returns True if it succeeded
	char		szConnOptions[512];
	SQLRETURN		rc;

	sprintf(szConnOptions, szConn);

#ifdef _DEBUG
	_CrtDbgReport(_CRT_WARN, NULL, 0, NULL,
"Executing SQLExecDirect for connection option.\n");
#endif
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
	if (m_bAbort)
		return;

	rc = SQLExecDirect(m_hHandle, (unsigned char
*)szConnOptions, SQL_NTS);
	HandleODBCError(rc, SQL_HANDLE_STMT, m_hHandle,
SMSQLExecDirect);

	return;
}

STDMETHODIMP CExecute::AbortODBC()
{
	m_bAbort = TRUE;

	try
	{
		if (m_hHandle != SQL_NULL_HSTMT)
		{
#ifdef _DEBUG
			_CrtDbgReport(_CRT_WARN, NULL, 0, NULL,
"Executing SQLCancel.\n");
#endif

			SQLRETURN rc = SQLCancel(m_hHandle);
			HandleODBCError(rc, SQL_HANDLE_STMT,
m_hHandle, SMSQLCancel);
		}
	}
	catch(CODBCError *pErr)
	{
		delete pErr;
	}

	return(S_OK);
}

void CExecute::ProcessResultsets()
{
   SQLSMALLINT		*CTypeArray, *CScaleArray;
   SQLINTEGER	*ColLenArray, *DispLenArray,
*OffsetArray;
	SQLSMALLINT	iColNameLen, SQLType, iColNull, i,
NumCols = 0;
	SQLINTEGER		iDispLen, iRowCount;
   SQLRETURN	rc;
	char		szColName[MAX_DATA_LEN + 1];
   void		*DataPtr;
	SQLINTEGER		iLenOrInd =
ALIGNBUF(sizeof(SQLINTEGER));
	SQLUINTEGER	iRowArraySize, iArrayElementSize;
//	SQLUINTEGER	NumRowsFetched;
//	SQLUSMALLINT	*RowStatusArray;


	if (!m_pOutputFile || m_bAbort)
		return;
	do
	{
#ifdef _DEBUG
		_CrtDbgReport(_CRT_WARN, NULL, 0, NULL,
"Executing SQLNumResultCols.\n");
#endif
		// Determine the number of result set columns.
		rc = SQLNumResultCols(m_hHandle, &NumCols);
		HandleODBCError(rc, SQL_HANDLE_STMT,
m_hHandle, SMSQLNumResultCols);

		if (NumCols > 0)
		{

			// Allocate arrays to hold the C type, scale, column
and display length of the data
			CTypeArray = (SQLSMALLINT *)
malloc(NumCols * sizeof(SQLSMALLINT));
			CScaleArray = (SQLSMALLINT *)
malloc(NumCols * sizeof(SQLSMALLINT));
			ColLenArray = (SQLINTEGER *)
malloc(NumCols * sizeof(SQLINTEGER));
			DispLenArray = (SQLINTEGER *)
malloc(NumCols * sizeof(SQLINTEGER));

			OffsetArray = (SQLINTEGER *) malloc(NumCols
* sizeof(SQLINTEGER));
			OffsetArray[0] = 0;

			for (i = 0; i < NumCols && !m_bAbort; i++)
			{
#ifdef _DEBUG
				_CrtDbgReport(_CRT_WARN, NULL, 0,
NULL, "Executing SQLDescribeCol.\n");
#endif
// Get the column description, include the SQL type
// Determine the column's byte length. Calculate the offset in the
//buffer to the data as the offset to the previous column, plus the
//byte length of the previous column, plus the byte length of the
//previous column's length/indicator buffer.
// Note that the byte length of the column and the length/indicator
//buffer are increasedso that, assuming they start on an alignment
//boundary, they will end on the byte before the next alignment
//boundary. Although this might leave some holes in the buffer, it
//is a relatively inexpensive way to guarantee alignment.
rc = SQLDescribeCol(m_hHandle, ((SQLUSMALLINT) i)+1,
(unsigned char *)szColName, sizeof(szColName),
&iColNameLen,
&SQLType, (unsigned long *)&ColLenArray[i],
&CScaleArray[i], &iColNull);
HandleODBCError(rc, SQL_HANDLE_STMT, m_hHandle,
SMSQLDescribeCol);
#ifdef _DEBUG
_CrtDbgReport(_CRT_WARN, NULL, 0, NULL, "Executing
SQLColAttribute.\n");
#endif
if (m_bAbort)
return;
rc = SQLColAttribute(m_hHandle, ((SQLUSMALLINT) i)+1,
SQL_DESC_DISPLAY_SIZE, NULL, 0, NULL, &iDispLen);
	HandleODBCError(rc, SQL_HANDLE_STMT, m_hHandle,
SMSQLColAttribute);
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
// GetDefaultCType contains a switch statement that returns the
//default C type for each SQL type.
CTypeArray[i] = GetDefaultCType(SQLType);
if ( (CTypeArray[i] == SQL_C_CHAR || CTypeArray[i] ==
SQL_C_BINARY) && ColLenArray[i] > MAX_DATA_LEN)
{
ColLenArray[i] = MAX_DATA_LEN;
iDispLen = MAX_DATA_LEN;
}

DispLenArray[i] = max(iColNameLen, iDispLen);
DispLenArray[i] = max(DispLenArray[i], sizeof(S_NULL));
// Print the column names in the header
PrintData(szColName, SQL_C_CHAR, DispLenArray[i], 0,
m_pOutputFile);
// Add a byte for the null-termination character
ColLenArray[i] += 1;
ColLenArray[i] = ALIGNBUF(ColLenArray[i]);
// Calculate the offset in the buffer to the data as the offset to the
previous column,
// plus the byte length of the previous column, plus the byte
//length of the previous column's length/indicator buffer.
if (i) OffsetArray[i] = OffsetArray[i-1] + ColLenArray[i-1] +
iLenOrInd;
}
m_pOutputFile->WriteLine(NULL);
iArrayElementSize = OffsetArray[NumCols-1] +
ColLenArray[NumCols-1] + iLenOrInd;
iRowArraySize = 1;
// Allocate the data buffer. The size of the buffer is equal to the
//offset to the data buffer for the final column, plus the byte
//length o f the data buffer and length/indicator
// buffer for the last column.
DataPtr = malloc(iRowArraySize * iArrayElementSize);
// Specify the size of the structure with the
SQL_ATTR_ROW_BIND_TYPE
      // statement attribute. This also declares that row-wise
binding will
            // be used. Declare the rowset size with the
SQL_ATTR_ROW_ARRAY_SIZE
            // statement attribute. Set the
SQL_ATTR_ROW_STATUS_PTR statement
            // attribute to point to the row status array. Set the
            // SQL_ATTR_ROWS_FETCHED_PTR statement
attribute to point to
            // NumRowsFetched.
            /*
            RowStatusArray = (SQLUSMALLINT
*)malloc(iRowArraySize * sizeof(SQLUSMALLINT));

            SQLSetStmtAttr(m_hHandle,
SQL_ATTR_ROW_BIND_TYPE, &iArrayElementSize,
SQL_IS_UINTEGER);
            SQLSetStmtAttr(m_hHandle,
SQL_ATTR_ROW_ARRAY_SIZE, &iRowArraySize,
SQL_IS_UINTEGER);
            SQLSetStmtAttr(m_hHandle,
SQL_ATTR_ROW_STATUS_PTR, RowStatusArray,
SQL_IS_POINTER);
            SQLSetStmtAttr(m_hHandle,
SQL_ATTR_ROWS_FETCHED_PTR, &NumRowsFetched,
SQL_IS_POINTER);
            */
```

```
// For each column, bind the address in the buffer at the start of
the memory allocated
// for that column's data and the address at the start of the
memory allocated for that
// column's length/indicator buffer.
for (i = 0; i < NumCols; i++)
{
SQLBindCol(m_hHandle, i + 1, CTypeArray[i],
(SQLPOINTER)((SQLCHAR *)DataPtr + OffsetArray[i]),
            ColLenArray[i], (SQLINTEGER *)((SQLCHAR
*)DataPtr + OffsetArray[i] + ColLenArray[i]));
            HandleODBCError(rc, SQL_HANDLE_STMT,
m_hHandle, SMSQLBindCol);

            // Underline each column name
            memset(szColName, '-', DispLenArray[i]);
            *(szColName + DispLenArray[i]) = '\0';
            PrintData(szColName, SQL_C_CHAR,
DispLenArray[i], 0, m_pOutputFile);
            }
            m_pOutputFile->WriteLine(NULL);

#ifdef _DEBUG
            _CrtDbgReport(_CRT_WARN, NULL, 0, NULL,
"Executing SQLFetch.\n");
#endif
while (!m_bAbort && (rc = SQLFetch(m_hHandle)) !=
SQL_NO_DATA)
{
HandleODBCError(rc, SQL_HANDLE_STMT, m_hHandle,
SMSQLFetch);
/*
for (i = 0; i < NumRowsFetched; i++)
{
if (RowStatusArray[i] == SQL_ROW_SUCCESS||
RowStatusArray[i] == SQL_ROW_SUCCESS_WITH_INFO)
{
*/
for (i = 0; i < NumCols; i++)
{
// Retrieve and print each row. PrintData accepts a pointer to the
data, its C type, and its byte length/indicator.
if ( *((SQLINTEGER *)((SQLCHAR *)DataPtr + OffsetArray[i]
+ ColLenArray[i])) == SQL_NULL_DATA)
PrintData(S_NULL, SQL_C_CHAR, DispLenArray[i], 0,
m_pOutputFile);
else
PrintData((LPVOID)((SQLCHAR *)DataPtr + OffsetArray[i]),
CTypeArray[i],
DispLenArray[i], CScaleArray[i], m_pOutputFile);
}
      m_pOutputFile->WriteLine(NULL);
      }
      m_pOutputFile->WriteLine(NULL);
      free(DataPtr);
      free(CTypeArray);
      free(CScaleArray);
      free(ColLenArray);
      free(DispLenArray);
      }
      // Write io statistics, if applicable
      LogODBCErrors(rc, SQL_HANDLE_STMT, m_hHandle,
SMSQLFetch);
#ifdef _DEBUG
_CrtDbgReport(_CRT_WARN, NULL, 0, NULL, "Executing
SQLRowCount.\n");
#endif
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
            if (m_bAbort)
                break;

            // action (insert, update, delete) query
            rc = SQLRowCount(m_hHandle, &iRowCount);
            HandleODBCError(rc, SQL_HANDLE_STMT,
m_hHandle, SMSQLRowCount);

            if (!m_bNoCount && iRowCount != -1)
            {
sprintf(szColName, "(%d row(s) affected)", iRowCount);
_bstr_t temp(szColName);
m_pOutputFile->WriteLine((BSTR)temp);
m_pOutputFile->WriteLine(NULL);
}
#ifdef _DEBUG
            _CrtDbgReport(_CRT_WARN, NULL, 0, NULL,
"Executing SQLFreeStmt.\n");
#endif

            if (m_bAbort)
                break;

            SQLFreeStmt(m_hHandle, SQL_UNBIND);

#ifdef _DEBUG
            _CrtDbgReport(_CRT_WARN, NULL, 0, NULL,
"Executing SQLMoreResults.\n");
#endif

            if (m_bAbort)
                break;

            // Process the next resultset. This function returns
'success with info' even
            // if there is no other resultset and there are statistics
messages to be printed.
            // Hence the check for -1 rows before printing.
            rc=SQLMoreResults(m_hHandle);
            if (rc != SQL_NO_DATA)
                HandleODBCError(rc, SQL_HANDLE_STMT,
m_hHandle, SMSQLMoreResults);

    } while (rc != SQL_NO_DATA);

        return;
}

void CExecute::PrintData(void *vData, SQLSMALLINT CType,
SQLINTEGER IndPtr, SQLSMALLINT iScale, ISMLog
*pOutput)
{
        // PrintData accepts a pointer to the data, its C type,
        // and its byte length/indicator. It contains a switch statement
that casts and prints
        // the data according to its type.

        char        *s;
        char        fmt[MAXBUFLEN];
        int                 j = 0;
        SQLINTEGER    iColLen = IndPtr + 1;
        assert(iColLen);
        s = (LPSTR)malloc(iColLen + 1);

        if (s)
{
if (vData)
{
        switch(CType)
        {
        case SQL_C_CHAR:
        case SQL_C_WCHAR:
        case SQL_C_TYPE_DATE:
        case SQL_C_TYPE_TIME:
        case SQL_C_TYPE_TIMESTAMP:
        case SQL_C_INTERVAL_YEAR:
        case SQL_C_INTERVAL_MONTH:
        case SQL_C_INTERVAL_YEAR_TO_MONTH:
        case SQL_C_INTERVAL_DAY:
        case SQL_C_INTERVAL_HOUR:
        case SQL_C_INTERVAL_MINUTE:
        case SQL_C_INTERVAL_SECOND:
        case SQL_C_INTERVAL_DAY_TO_HOUR:
        case SQL_C_INTERVAL_DAY_TO_MINUTE:
        case SQL_C_INTERVAL_DAY_TO_SECOND:
        case SQL_C_INTERVAL_HOUR_TO_MINUTE:
        case SQL_C_INTERVAL_HOUR_TO_SECOND:
        case SQL_C_INTERVAL_MINUTE_TO_SECOND:
        case SQL_C_BINARY:
            sprintf(fmt, "%%.%ds", iColLen);
            j = sprintf(s, fmt, (char *)vData);
            break;

        case SQL_C_SHORT:
            j = sprintf(s, "%d", *(short *)vData);
            break;

        case SQL_C_LONG:
            j = sprintf(s, "%ld", *(long *)vData);
            break;

        case SQL_C_UBIGINT:
            j = sprintf(s, "%I64d", *(__int64 *)vData);
            break;

        case SQL_C_FLOAT:
        j = sprintf(s, "%f", *(float *)vData);
            break;

        case SQL_C_DOUBLE:
            j = sprintf(s, "%f", *(double *)vData);
                break;
            case SQL_C_NUMERIC:
        sprintf(fmt, "%%.0%df", iScale);
        j = sprintf(s, fmt, *(double *)vData);
        break;
        default:
        j = sprintf(s, "%s", vData);
        break;
        }
        }
        if (iColLen - j > 0)
            memset(s + j, ' ', iColLen - j);

        *(s + iColLen) = '\0';

        // Write the field to the output file
        _bstr_t temp(s);
        pOutput->WriteField((BSTR)temp);
        free(s);
}
return;
}
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
SQLSMALLINT CExecute::GetDefaultCType(SQLINTEGER
SQLType)
{
      // GetDefaultCType returns the C type for the passed in SQL
datatype.

   switch(SQLType)
   {
   case SQL_CHAR:
      case SQL_VARCHAR:
      case SQL_LONGVARCHAR:
      case SQL_WCHAR:
      case SQL_WVARCHAR:
      case SQL_WLONGVARCHAR:
         return(SQL_C_CHAR);

   case SQL_TINYINT:
                return(SQL_C_CHAR);

   case SQL_SMALLINT:
                return(SQL_C_SHORT);

   case SQL_INTEGER:
                return(SQL_C_LONG);

   case SQL_BIGINT:
                return(SQL_C_UBIGINT);

   case SQL_REAL:
                return(SQL_C_FLOAT);

   case SQL_FLOAT:
   case SQL_DOUBLE:
//    case SQL_DECIMAL:
                return(SQL_C_DOUBLE);

      case SQL_DECIMAL:
                return(SQL_C_CHAR);

      case SQL_BIT:
                return(SQL_C_CHAR);

   case SQL_BINARY:
      case SQL_VARBINARY:
      case SQL_LONGVARBINARY:
                return(SQL_C_CHAR);
//              return(SQL_C_BINARY);

   case SQL_TYPE_DATE:
                return(SQL_C_CHAR);
//              return(SQL_C_TYPE_DATE);

   case SQL_TYPE_TIME:
                return(SQL_C_CHAR);
//              return(SQL_C_TYPE_TIME);

   case SQL_TYPE_TIMESTAMP:
                return(SQL_C_CHAR);
//              return(SQL_C_TYPE_TIMESTAMP);

      case SQL_NUMERIC:
         return(SQL_C_FLOAT);
   case SQL_INTERVAL_YEAR:
                return(SQL_C_CHAR);
//              return(SQL_C_INTERVAL_YEAR);

   case SQL_INTERVAL_MONTH:
                return(SQL_C_CHAR);
//              return(SQL_C_INTERVAL_MONTH);

   case SQL_INTERVAL_YEAR_TO_MONTH:
                return(SQL_C_CHAR);
// return(SQL_C_INTERVAL_YEAR_TO_MONTH);

   case SQL_INTERVAL_DAY:
                return(SQL_C_CHAR);
//              return(SQL_C_INTERVAL_DAY);

   case SQL_INTERVAL_HOUR:
                return(SQL_C_CHAR);
// return(SQL_C_INTERVAL_HOUR);

   case SQL_INTERVAL_MINUTE:
                return(SQL_C_CHAR);
// return(SQL_C_INTERVAL_MINUTE);

   case SQL_INTERVAL_SECOND:
                return(SQL_C_CHAR);
//  return(SQL_C_INTERVAL_SECOND);

   case SQL_INTERVAL_DAY_TO_HOUR:
                return(SQL_C_CHAR);
// return(SQL_C_INTERVAL_DAY_TO_HOUR);

   case SQL_INTERVAL_DAY_TO_MINUTE:
                return(SQL_C_CHAR);
// return(SQL_C_INTERVAL_DAY_TO_MINUTE);

   case SQL_INTERVAL_DAY_TO_SECOND:
                return(SQL_C_CHAR);
// return(SQL_C_INTERVAL_DAY_TO_SECOND);

   case SQL_INTERVAL_HOUR_TO_MINUTE:
                return(SQL_C_CHAR);
// return(SQL_C_INTERVAL_HOUR_TO_MINUTE);

   case SQL_INTERVAL_HOUR_TO_SECOND:
                return(SQL_C_CHAR);
//
      return(SQL_C_INTERVAL_HOUR_TO_SECOND);

   case SQL_INTERVAL_MINUTE_TO_SECOND:
                return(SQL_C_CHAR);
// r eturn(SQL_C_INTERVAL_MINUTE_TO_SECOND);

      default:
         assert(TRUE);
         return(SQL_C_CHAR);
         break;
   }
}
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
/*
   FUNCTION: LogODBCErrors(SQLRETURN rc, SWORD
fHandleType, SQLHANDLE handle)
   COMMENTS: Formats ODBC errors or warnings and logs
them. Also initializes the
                    completion status for the step to failure, if an
ODBC error has occurred.
*/

void CExecute::LogODBCErrors(SQLRETURN nResult,
SWORD fHandleType, SQLHANDLE handle, OdbcOperations
FailedOp)
{
      // Messages returned by the server (e.g. Print statements) will
be logged to the output file
      // ODBC warnings will be logged to the log file
      // All other ODBC errors will be logged to the error file.

      UCHAR       szErrState[SQL_SQLSTATE_SIZE+1];       //
SQL Error State string
      UCHAR
      szErrText[SQL_MAX_MESSAGE_LENGTH+1];    // SQL
Error Text string
      char
      szBuffer[SQL_SQLSTATE_SIZE+SQL_MAX_MESSAGE
_LENGTH+MAXBUFLEN+1] = "";
                  // formatted Error text Buffer
      SWORD             wErrMsgLen;                          // Error
message length
      SQLINTEGER    dwErrCode;                        // Native
Error code
      SQLRETURN     nErrResult;                       // Return
Code from SQLGetDiagRec
      SWORD             sMsgNum = 1;                         // Error
sequence number
      _bstr_t         temp;

      if (IsErrorReturn(nResult))
      {
            sprintf(szBuffer, "ODBC Operation: '%s' returned error
code: %d",
                  g_szOdbcOps[FailedOp], nResult);
            temp = szBuffer;
            m_pErrorFile->WriteLine((BSTR) temp);
            m_StepStatus = gintFailed;
      }

      if (handle == SQL_NULL_HSTMT)
            return;

#ifdef _DEBUG
      _CrtDbgReport(_CRT_WARN, NULL, 0, NULL,
"Executing SQLGetDiagRec.\n");
#endif

      // call SQLGetDiagRec function with proper ODBC handles,
repeatedly until
      // function returns SQL_NO_DATA.
      while (!m_bAbort && (nErrResult =
SQLGetDiagRec(fHandleType, handle, sMsgNum++,
            szErrState, &dwErrCode, szErrText,
SQL_MAX_MESSAGE_LENGTH-1, &wErrMsgLen))
            != SQL_NO_DATA)
      {
            if (!SQL_SUCCEEDED(nErrResult))
                  break;

            if (m_pOutputFile && IsServerMessage(dwErrCode,
szErrText))
            {
                  wsprintf(szBuffer, SM_SQLMSG_FORMAT,
(LPSTR)szErrText);
                  temp = szBuffer;
                  m_pOutputFile->WriteLine((BSTR) temp);
            }
            else if (IsODBCWarning(szErrState) && dwErrCode !=
SM_SQL_ERR_CHANGED_DB &&
            dwErrCode != SM_SQL_ERR_CHANGED_LANG)
            {
                  // Suppress warnings - 'Changed database context
to...' and 'Changed language setting to...'
                  wsprintf(szBuffer, SM_SQLMSG_FORMAT,
ParseOdbcMsgPrefixes((LPCSTR)szErrText));
                  temp = szBuffer;
                  m_pOutputFile->WriteLine((BSTR) temp);
            }
            else if (m_pErrorFile &&
!IsODBCWarning(szErrState))
            {
                  wsprintf(szBuffer, SM_SQLERR_FORMAT,
(LPSTR)szErrState, dwErrCode, (LPSTR)szErrText);
                  temp = szBuffer;
                  m_pErrorFile->WriteLine((BSTR) temp);
            }
      }
}

/*
   FUNCTION: LogErrors(SQLRETURN rc, SWORD
fHandleType, SQLHANDLE handle)
   COMMENTS: Writes the error message to the error log
*/

void CExecError::LogErrors(CExecute *p)
{
      _bstr_t temp(m_szExecErrorDesc[m_iErrCode]);

      if (p->m_pErrorFile)
            p->m_pErrorFile->WriteLine((BSTR)temp);

      return;
}
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
/*
   FUNCTION: ODBCCleanup(HDBC *hdbc, HSTMT *hstmt)
   COMMENTS: Cleanup of all ODBC structures
*/

void CExecute::ODBCCleanup(HDBC *hdbc, HSTMT *hstmt)
{
      SQLRETURN     lReturn;

#ifdef _DEBUG
      _CrtDbgReport(_CRT_WARN, NULL, 0, NULL,
"Executing ODBCCleanup.\n");
#endif

      if (*hstmt != SQL_NULL_HSTMT)
      {
#ifdef _DEBUG
            _CrtDbgReport(_CRT_WARN, NULL, 0, NULL,
"Executing SQLCloseCursor.\n");
#endif
            SQLCloseCursor(hstmt);
#ifdef _DEBUG
            _CrtDbgReport(_CRT_WARN, NULL, 0, NULL,
"Executing SQLFreeHandle for hstmt.\n");
#endif
            SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
            *hstmt = SQL_NULL_HSTMT;
      }

      // Cleanup connection if it is a dynamic connection
      if (IsDynamicConnection())
      {
            if (*hdbc != SQL_NULL_HDBC)
            {
#ifdef _DEBUG
                  _CrtDbgReport(_CRT_WARN, NULL, 0, NULL,
"Executing SQLDisconnect.\n");
#endif
                  lReturn = SQLDisconnect(*hdbc);
#ifdef _DEBUG
                  _CrtDbgReport(_CRT_WARN, NULL, 0, NULL,
"Executing SQLFreeHandle for hdbc.\n");
#endif
                  SQLFreeHandle(SQL_HANDLE_DBC, hdbc);
                  *hdbc = SQL_NULL_HDBC;
            }
      }
      else
            ResetConnectionUsage();

      return;
}

// Wrapper function that raises an error if a Windows Api fails
STDMETHODIMP CExecute::RaiseSystemError(void)
{
      char s[MAXBUFLEN];

      GetSystemError(s);
      return Error(s, 0, NULL, GUID_NULL);
}

// Wrapper function that logs the error raised by an Api function
to the passed in file
void CExecute::LogSystemError(ISMLog *pFile)
{
      if (pFile)
      {
            char s[MAXBUFLEN];
            GetSystemError(s);

            _bstr_t temp(s);
            pFile->WriteLine((BSTR)temp);
      }
}

// Populates the passed in string with the last Windows Api error
that occurred
void CExecute::GetSystemError(LPSTR s)
{
      long c;
      DWORD e;

      e = GetLastError();

      c = sprintf(s, "Error code: %ld. ", e);
      c =
FormatMessage(FORMAT_MESSAGE_FROM_SYSTEM |
FORMAT_MESSAGE_IGNORE_INSERTS,
            NULL, e, 0, s + c, MAXBUFLEN - c, NULL);

      return;
}

STDMETHODIMP CExecute::get_StepStatus(InstanceStatus
*pVal)
{
      *pVal = m_StepStatus;
      return S_OK;
}

STDMETHODIMP CExecute::WriteError(BSTR szMsg)
{
      if (m_pErrorFile)
            return(m_pErrorFile->WriteLine(szMsg));

      return S_OK;
}
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
//  FILE:      Execute.h
//             Microsoft TPC-H Kit Ver. 1.00
//             Copyright Microsoft, 1999
//             All Rights Reserved
//
//
//  PURPOSE:    Declaration of the CExecute
//  Contact:    Reshma Tharamal (reshmat@microsoft.com)
//
// Execute.h : Declaration of the CExecute

#ifndef __EXECUTE_H_
#define __EXECUTE_H_

#include <atlwin.h>
#include <comdef.h>
#include <stdio.h>
#include "resource.h"       // main symbols
#include "ExecuteDllCP.h"
#include "..\LogWriter\LogWriter.h"
#include "..\LogWriter\SMLog.h"
#include "..\common\SMTime\SMTime.h"
#include "..\common\SMTime\SMTimer.h"

// ODBC-specific includes
#define DBNTWIN32
#include <sqltypes.h>
#include <sql.h>
#include <sqlext.h>

/////////////////////////////////////////////////////////////////////////////
// CExecute

#define WM_TASK_START  (WM_USER + 101)
#define WM_TASK_FINISH  (WM_USER + 102)

#define SM_SQLERR_FORMAT          "SQL Error State:%s,
Native Error Code: %ld\r\nODBC Error: %s"
// format for ODBC error messages
#define SM_SQLWARN_FORMAT SM_SQLERR_FORMAT
// format for ODBC warnings
#define SM_SQLMSG_FORMAT "%s" // format for messages
from the server

#define SM_SQL_STATE_WARNING "01000"
#define SM_MSG_SERVER "[Microsoft][ODBC SQL Server
Driver][SQL Server]"

#define SM_SQL_ERR_CHANGED_DB     5701
#define SM_SQL_ERR_CHANGED_LANG 5703

#define SM_STEPMASTER_ERROR          "StepMaster Error: "

#define CMD_SEPARATOR                "\nGO"
#define INV_ARRAY_INDEX     -1                  // invalid index
into an array
#define MAXBUFLEN           256        // display buffer size
#define MAXLOGCMDLEN        256        // maximum
characters in command that will be // printed to log
#define MAXLOGCMDBUF        512        // maximum
characters in command that will be // printed to log
#define MAX_DATA_LEN        4000       // maximum buffer
size for variable-length data types // viz. character and binary
fields
#define FILE_ACCESS_READ    "r" // Open file for read access
#define S_NULL "NULL"
// Define a macro to increase the size of a buffer so it is a
multiple of teh alignment size.
// Thus, if a buffer starts on an alignment boundary, it will end
just before the next
// alignment boundary. Here, an alignment size of 4 is used
because this is the size of the
// largest data type used in the application's buffer - the size of an
SDWORD and of the largest
// default C data type are both 4. If a larger data type (such as
__int64) is used, it will be
// necessary to align for that size.
#define ALIGNSIZE 4
#define ALIGNBUF(Length) ((Length) % ALIGNSIZE) ? \
   ((Length) + ALIGNSIZE - ((Length) % ALIGNSIZE)) :
(Length)

#define MAX_BUFFER_SIZE       64000

typedef enum OdbcOperations
{
     SMSQLAllocHandle,
     SMSQLDriverConnect,
     SMSQLExecDirect,
     SMSQLSetStmtAttr,
     SMSQLCancel,
     SMSQLNumResultCols,
     SMSQLDescribeCol,
     SMSQLColAttribute,
     SMSQLFetch,
     SMSQLGetData,
     SMSQLRowCount,
     SMSQLMoreResults,
     SMSQLBindCol,
};

class CODBCError
{
public:
     CODBCError(SQLRETURN nResult, SWORD
fHandleType, SQLHANDLE handle, OdbcOperations FailedOp)
     {
          m_fHandleType   = fHandleType;
          m_handle        = handle;
          m_FailedOp      = FailedOp;
          m_nResult       = nResult;
     };

private:
     SWORD               m_fHandleType;
     SQLHANDLE           m_handle;
     OdbcOperations  m_FailedOp;
     SQLRETURN           m_nResult;

private:
inline BOOL IsServerMessage(SQLINTEGER lNativeError,
UCHAR *szErr){
     return( (strstr((LPCTSTR)szErr, SM_MSG_SERVER) !=
NULL) ? (lNativeError == 0) : FALSE); }
inline BOOL IsODBCWarning(UCHAR *szSqlState){
     return(strcmp((LPCSTR)szSqlState,
SM_SQL_STATE_WARNING) == 0);}
inline LPCSTR ParseOdbcMsgPrefixes(LPCSTR szMsg){ char
*pDest;
     return( (pDest = strstr(szMsg, SM_MSG_SERVER)) ==
NULL ? szMsg : pDest + strlen(SM_MSG_SERVER));}
};
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
class ATL_NO_VTABLE CExecute :
	public CWindowImpl<CExecute>,
	public CComObjectRootEx<CComSingleThreadModel>,
	public CComCoClass<CExecute, &CLSID_Execute>,
	public IConnectionPointContainerImpl<CExecute>,
	public ISupportErrorInfo,
	public IDispatchImpl<IExecute, &IID_IExecute,
&LIBID_EXECUTEDLLLib>,
	public CProxy_IExecuteEvents< CExecute >
{
public:
	CExecute()
	{
		m_pErrorFile = NULL;
		//m_pLogFile = NULL;
		m_pOutputFile = NULL;

		// Initialize the elapsed time for the step
		m_tElapsedTime = 0;

		// Initialize the run status for the step
		m_StepStatus = gintPending;

		m_hHandle = SQL_NULL_HSTMT;
		m_bAbort = FALSE;

		m_iConnectionIndex = INV_ARRAY_INDEX;
	}

	~CExecute()
	{
	}

	friend class CExecError;

public:
	DECLARE_WND_CLASS("Execute")

		BEGIN_MSG_MAP(CExecute)
		MESSAGE_HANDLER(WM_TASK_FINISH,
OnTaskFinished)
		MESSAGE_HANDLER(WM_TASK_START,
OnTaskStarted)
		END_MSG_MAP()
public:

	LRESULT OnTaskStarted(UINT uMsg, WPARAM
wParam,
		LPARAM lParam, BOOL& bHandled)
	{
		CURRENCY	CStartTime =
Get64BitTime(&m_tStartTime);

		Fire_Start(CStartTime);
		return 0;
	}

	LRESULT OnTaskFinished(UINT uMsg, WPARAM
wParam,
		LPARAM lParam, BOOL& bHandled)
	{
		CURRENCY	CEndTime =
Get64BitTime(&m_tEndTime);

		Fire_Complete(CEndTime, (long)m_tElapsedTime);
		return 0;
	}

	HRESULT FinalConstruct()
	{
		HRESULT		hr;
		RECT		rect;

		rect.left=0;
		rect.right=100;
		rect.top=0;
		rect.bottom=100;

		HWND hwnd = Create( NULL, rect,
"ExecuteWindow", WS_POPUP);

		if (!hwnd)
			return HRESULT_FROM_WIN32(GetLastError());

		hr = CoCreateInstance(CLSID_SMLog, NULL,
CLSCTX_INPROC,
			IID_ISMLog, (void **)&m_pErrorFile);
		if FAILED(hr)
			return(hr);
		m_pErrorFile->put_Append(TRUE);

		//hr = CoCreateInstance(CLSID_SMLog, NULL,
CLSCTX_INPROC,
		//	IID_ISMLog, (void **)&m_pLogFile);
		//if FAILED(hr)
		//	return(hr);

		hr = CoCreateInstance(CLSID_SMLog, NULL,
CLSCTX_INPROC,
			IID_ISMLog, (void **)&m_pOutputFile);
		if FAILED(hr)
			return(hr);
		m_pOutputFile->put_Append(TRUE);

		hr = CoCreateInstance(CLSID_SMTimer, NULL,
CLSCTX_INPROC,
			IID_ISMTimer, (void **)&m_ExecTime);
		if FAILED(hr)
			return(hr);

		return S_OK;
	}
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
    void FinalRelease()
    {
        if (m_hWnd != NULL)
        DestroyWindow();
        // Close the log and error files
        if (m_pErrorFile)
        m_pErrorFile->Release();
        m_pErrorFile = NULL;
        if (m_ExecTime)
        m_ExecTime->Release();
        m_ExecTime = NULL;
    }

DECLARE_REGISTRY_RESOURCEID(IDR_EXECUTE)
DECLARE_PROTECT_FINAL_CONSTRUCT()
BEGIN_COM_MAP(CExecute)
COM_INTERFACE_ENTRY(IExecute)
COM_INTERFACE_ENTRY(ISupportErrorInfo)
COM_INTERFACE_ENTRY(IDispatch)
COM_INTERFACE_ENTRY(IConnectionPointContainer)
COM_INTERFACE_ENTRY_IMPL(IConnectionPointContaine
r)
END_COM_MAP()
BEGIN_CONNECTION_POINT_MAP(CExecute)
CONNECTION_POINT_ENTRY(DIID__IExecuteEvents)
END_CONNECTION_POINT_MAP()
// ISupportsErrorInfo
STDMETHOD(InterfaceSupportsErrorInfo)(REFIID riid);
// IExecute
public:
STDMETHOD(put_ErrorFile)(/*[in]*/ BSTR newVal);
STDMETHOD(put_OutputFile)(/*[in]*/ BSTR newVal);
STDMETHOD(WriteError)(BSTR szMsg);
STDMETHOD(Abort)();
STDMETHOD(get_StepStatus)(/*[out, retval]*/ InstanceStatus
*pVal);
STDMETHOD(DoExecute)(/*[in]*/ BSTR szCommand, /*[in]*/
BSTR szExecutionDtls,
/*[in]*/ ExecutionType ExecMethod,
/*[in]*/ BOOL bNoCount, /*[in]*/ BOOL bNoExecute, /*[in]*/
BOOL bParseOnly,
/*[in]*/ BOOL bQuotedIds, /*[in]*/ BOOL bAnsiNulls, /*[in]*/
BOOL bShowQP,
/*[in]*/ BOOL bStatsTime, /*[in]*/ BOOL bStatsIO, /*[in]*/
long lRowCount,
/*[in]*/ long lQueryTmout, /*[in]*/ BSTR szConnection);
TC_TIME         ExecuteShell();
TC_TIME         ExecuteODBC();
STDMETHODIMP    AbortShell();
STDMETHODIMP    AbortODBC();
_bstr_t         m_szCommand;
_bstr_t         m_szExecDtls;
_bstr_t         m_szConnection;
DWORD           m_lMode;
SYSTEMTIME      m_tStartTime;
SYSTEMTIME      m_tEndTime;
TC_TIME         m_tElapsedTime;
ISMLog          *m_pErrorFile;
//ILog          *m_pLogFile;
ISMLog          *m_pOutputFile;
ISMTimer        *m_ExecTime;
ExecutionType   m_ExecMthd;
InstanceStatus  m_StepStatus;
HANDLE          m_hHandle; // Process handle for shell
commands and
//Statement handle for ODBC commands LPSTRm_szCmd;
private:
    LPSTR           NextCmdInBatch(LPSTR szBatch);
    void            ProcessResultsets();
    SQLSMALLINT     GetDefaultCType(SQLINTEGER
SQLType);
    void            PrintData(void *vData, SQLSMALLINT
CType, SQLINTEGER IndPtr,
                    SQLSMALLINT iScale, ISMLog *pOutput);
    void            LogODBCErrors(SQLRETURN nResult,
SWORD fHandleType, SQLHANDLE handle,
                    OdbcOperations FailedOp);
    void            ODBCCleanup(HDBC *hdbc, HSTMT
*hstmt);
    STDMETHODIMP        RaiseSystemError(void);
    void            LogSystemError(ISMLog *pFile);
    void            GetSystemError(LPSTR s);
    void            SetConnectionOption(LPSTR szConn, HDBC
*pHdbc);
    void            ResetConnectionProperties(HDBC *p_hdbc);

    void            InitializeConnection(HDBC *phdbc, BOOL
*pbDoConnect);
    void            ReConnectDeadConnection(HDBC *phdbc);

    void            ResetConnectionUsage();

    int             m_iConnectionIndex;

    BOOL            m_bNoCount, m_bNoExecute,
m_bParseOnly, m_bQuotedIds, m_bAnsiNulls, \
                    m_bShowQP, m_bStatsTime, m_bStatsIO;
    long            m_lRowCount;
    SQLUINTEGER     m_lQueryTmout;
    _bstr_t         m_ErrorFile, m_OutputFile;
    BOOL            m_bAbort;
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
private:
inline BOOL IsServerMessage(SQLINTEGER lNativeError,
UCHAR *szErr){
	return( (strstr((LPCTSTR)szErr, SM_MSG_SERVER) !=
NULL) ? (lNativeError == 0) : FALSE); }
inline BOOL IsODBCWarning(UCHAR *szSqlState){
	return(strcmp((LPCSTR)szSqlState,
SM_SQL_STATE_WARNING) == 0);}
inline BOOL IsErrorReturn(SQLRETURN iRetCode){
	return( (!SQL_SUCCEEDED(iRetCode)) && (iRetCode !=
SQL_NO_DATA) );}
inline LPCSTR ParseOdbcMsgPrefixes(LPCSTR szMsg){ char
*pDest;
	return( (pDest = strstr(szMsg, SM_MSG_SERVER)) ==
NULL ? szMsg : pDest + strlen(SM_MSG_SERVER));}
inline BOOL IsDynamicConnection(){
return(!strcmp((LPSTR)m_szConnection, ""));}
inline void HandleODBCError(SQLRETURN rc, SWORD
fHandleType, SQLHANDLE handle, OdbcOperations OdbcOp)
{
	if (rc != SQL_SUCCESS)
	{
		LogODBCErrors(rc, fHandleType, handle, OdbcOp);
		if (IsErrorReturn(rc))
			throw new CODBCError(rc, fHandleType, handle,
OdbcOp);
	}
	return;
}

};

class CExecError
{
public:
	typedef enum ExecErrorCodes
	{
		SM_ERR_CONN_IN_USE,
	};

	CExecError(int iError)
	{
		m_iErrCode  = iError;
	};

	void		LogErrors(CExecute *p);

private:
	int		m_iErrCode;
	static char	*m_szExecErrorDesc[];
};

void		ExecutionThread(LPVOID lpParameter);

#ifdef _TPCH_AUDIT
	void		WriteFileToTpchLog(LPSTR szFile, LPSTR
szFmt);
	void		WriteToTpchLog(char *szMsg);
#endif
```

```
#endif //__EXECUTE_H_
//  FILE:	ExecuteDll.cpp
//		Microsoft TPC-H Kit Ver. 1.00
//		Copyright Microsoft, 1999
//		All Rights Reserved
//
//
//  PURPOSE:	Implementation of DLL Exports.
//  Contact:	Reshma Tharamal (reshmat@microsoft.com)
//


// Note: Proxy/Stub Information
//	To build a separate proxy/stub DLL,
//	run nmake -f ExecuteDllps.mk in the project directory.

#include "stdafx.h"
#include "resource.h"
#include <initguid.h>

#include "..\LogWriter\LogWriter.h"
#include "..\LogWriter\LogWriter_i.c"

#include "..\common\SMTime\SMTime.h"
#include "..\common\SMTime\SMTime_i.c"

#include "ExecuteDll.h"
#include "SMExecute.h"

#include "ExecuteDll_i.c"
#include "Execute.h"

CComModule _Module;

BEGIN_OBJECT_MAP(ObjectMap)
OBJECT_ENTRY(CLSID_Execute, CExecute)
END_OBJECT_MAP()
SQLHENV henv = NULL; // ODBC environment handle

static char szCaption[] = "StepMaster";  // Message box caption

CRITICAL_SECTION hConnections;		// Critical section to
serialize access to available connections
SM_Connection_Info *p_Connections = NULL;	// Pointer to
open connections
int		iConnectionCount = 0;	// Number of
open connections

#ifdef _TPCH_AUDIT
	FILE *pfLogFile = NULL;		// Log file
containing timestamps
	CRITICAL_SECTION hLogFileWrite;	// Critical
section to serialize writes to log
	static char szFileOpenModeAppend[] = "a+";	// Log file
open mode
	static char szEnvVarLogFile[] = "TPCH_LOG_FILE";//
Environment variable - initialized to
		// log file name if timing information
		// is to be logged
#endif

void ShowODBCErrors(SWORD fHandleType, SQLHANDLE
handle);
void CloseOpenConnections();
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
/////////////////////////////////////////////////////////////////////////////
// DLL Entry Point
extern "C"
BOOL WINAPI DllMain(HINSTANCE hInstance, DWORD
dwReason, LPVOID /*lpReserved*/)
{
   if (dwReason == DLL_PROCESS_ATTACH)
   {
      _Module.Init(ObjectMap, hInstance,
&LIBID_EXECUTEDLLLib);
      DisableThreadLibraryCalls(hInstance);
#ifdef _TPCH_AUDIT
      char szMsg[MAXBUFLEN];
      LPSTR szLogFileName = getenv(szEnvVarLogFile);
      if (szLogFileName == NULL)
      {
      sprintf(szMsg, "The environment variable '%s' does not
exist. "
      "Step timing information will not be written to a log.",
szEnvVarLogFile);
 MessageBox(NULL, szMsg, szCaption, MB_OK);
      }
      else
      {
      if ( (pfLogFile = fopen(szLogFileName,
szFileOpenModeAppend)) == NULL )
      {
      sprintf(szMsg, "The file '%s' does not exist. "
      "Step timing information will not be written to log.",
szLogFileName);
      MessageBox(NULL, szMsg, szCaption, MB_OK);
      }
      else
      InitializeCriticalSection(&hLogFileWrite);
      }
#endif
      InitializeCriticalSection(&hConnections);
      p_Connections = NULL;
      iConnectionCount = 0;
      if (!SQL_SUCCEEDED(SQLSetEnvAttr(NULL,
      SQL_ATTR_CONNECTION_POOLING,
      (SQLPOINTER)SQL_CP_ONE_PER_HENV , 0)))
      ShowODBCErrors(SQL_HANDLE_ENV, henv);
      if
(!SQL_SUCCEEDED(SQLAllocHandle(SQL_HANDLE_ENV,
SQL_NULL_HANDLE, &henv)))
      {
      ShowODBCErrors(SQL_HANDLE_ENV, henv);
      return FALSE;
      }
      if (!SQL_SUCCEEDED(SQLSetEnvAttr(henv,
SQL_ATTR_ODBC_VERSION, (LPVOID)SQL_OV_ODBC3,
0)))
      ShowODBCErrors(SQL_HANDLE_ENV, henv);
      SQLUINTEGER CpMatch;
      if (!SQL_SUCCEEDED(SQLGetEnvAttr(henv,
SQL_ATTR_CP_MATCH, &CpMatch, 0, NULL)))
      ShowODBCErrors(SQL_HANDLE_ENV, henv);

      if (!SQL_SUCCEEDED(SQLSetEnvAttr(henv,
SQL_ATTR_CP_MATCH,
(SQLPOINTER)SQL_CP_STRICT_MATCH,
SQL_IS_INTEGER)))
ShowODBCErrors(SQL_HANDLE_ENV, henv);
   }
   else if (dwReason == DLL_PROCESS_DETACH)
      {

#ifdef _TPCH_AUDIT
            if (pfLogFile != NULL)
            {
                  fclose(pfLogFile);

                  DeleteCriticalSection(&hLogFileWrite);
            }
#endif

            CloseOpenConnections();

            if (henv != NULL)
                  SQLFreeEnv(henv);

            DeleteCriticalSection(&hConnections);

      _Module.Term();
      }
   return TRUE;    // ok
}

void ShowODBCErrors(SWORD fHandleType, SQLHANDLE
handle)
{
      UCHAR       szErrState[SQL_SQLSTATE_SIZE+1];// SQL
Error State string
      UCHAR
      szErrText[SQL_MAX_MESSAGE_LENGTH+1];    // SQL
Error Text string
      char
      szBuffer[SQL_SQLSTATE_SIZE+SQL_MAX_MESSAGE
_LENGTH+MAXBUFLEN+1] = "";
                                    // formatted Error text Buffer
      SWORD          wErrMsgLen;                  // Error
message length
      SQLINTEGER     dwErrCode;                   // Native Error
code
      SQLRETURN      nErrResult;                  // Return Code
from SQLGetDiagRec
      SWORD          sMsgNum = 1;                 // Error
sequence number

      // call SQLGetDiagRec function with proper ODBC handles,
repeatedly until
      // function returns SQL_NO_DATA.
      while ((nErrResult = SQLGetDiagRec(fHandleType, handle,
sMsgNum++,
            szErrState, &dwErrCode, szErrText,
SQL_MAX_MESSAGE_LENGTH-1, &wErrMsgLen))
            != SQL_NO_DATA)
      {
      if (!SQL_SUCCEEDED(nErrResult))
            break;

      wsprintf(szBuffer, SM_SQLERR_FORMAT,
(LPSTR)szErrState, dwErrCode, (LPSTR)szErrText);
      MessageBox(NULL, szBuffer, szCaption, MB_OK);
      }
}
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
void CloseOpenConnections()
{
	// Closes all open connections

	if (p_Connections)
	{
		for (int iConnIndex = iConnectionCount - 1;
iConnIndex >= 0; iConnIndex--)
		{
			if ((p_Connections + iConnIndex)->hdbc !=
SQL_NULL_HDBC)
			{
#ifdef _DEBUG
			_CrtDbgReport(_CRT_WARN, NULL, 0, NULL,
"Executing SQLDisconnect.\n");
#endif
			SQLDisconnect((p_Connections + iConnIndex)-
>hdbc);
#ifdef _DEBUG
			_CrtDbgReport(_CRT_WARN, NULL, 0, NULL,
"Executing SQLFreeHandle for hdbc.\n");
#endif
			SQLFreeHandle(SQL_HANDLE_DBC,
(p_Connections + iConnIndex)->hdbc );
			(p_Connections + iConnIndex)->hdbc =
SQL_NULL_HDBC;
			}

		}

		free(p_Connections);
	}
	p_Connections = NULL;

	return;
}

///////////////////////////////////////////////////////////////////////////////
// Used to determine whether the DLL can be unloaded by OLE

STDAPI DllCanUnloadNow(void)
{
   return (_Module.GetLockCount()==0) ? S_OK : S_FALSE;
}

///////////////////////////////////////////////////////////////////////////////
// Returns a class factory to create an object of the requested type

STDAPI DllGetClassObject(REFCLSID rclsid, REFIID riid,
LPVOID* ppv)
{
   return _Module.GetClassObject(rclsid, riid, ppv);
}

///////////////////////////////////////////////////////////////////////////////
// DllRegisterServer - Adds entries to the system registry

STDAPI DllRegisterServer(void)
{
   // registers object, typelib and all interfaces in typelib
   return _Module.RegisterServer(TRUE);
}

///////////////////////////////////////////////////////////////////////////////
// DllUnregisterServer - Removes entries from the system
registry

STDAPI DllUnregisterServer(void)
{
   return _Module.UnregisterServer(TRUE);
}
/* this ALWAYS GENERATED file contains the definitions for
the interfaces */


/* File created by MIDL compiler version 5.01.0164 */
/* at Mon Jun 09 19:33:03 2003
 */
/* Compiler settings for
C:\charles\Stepmaster\ExecuteDll\ExecuteDll.idl:
   Oicf (OptLev=i2), W1, Zp8, env=Win32, ms_ext, c_ext
   error checks: allocation ref bounds_check enum stub_data
*/
//@@MIDL_FILE_HEADING(  )


/* verify that the <rpcndr.h> version is high enough to compile
this file*/
#ifndef __REQUIRED_RPCNDR_H_VERSION__
#define __REQUIRED_RPCNDR_H_VERSION__ 440
#endif

#include "rpc.h"
#include "rpcndr.h"

#ifndef __ExecuteDll_h__
#define __ExecuteDll_h__

#ifdef __cplusplus
extern "C"{
#endif

/* Forward Declarations */

#ifndef ___IExecuteEvents_FWD_DEFINED__
#define ___IExecuteEvents_FWD_DEFINED__
typedef interface _IExecuteEvents _IExecuteEvents;
#endif	/* ___IExecuteEvents_FWD_DEFINED__ */


#ifndef __IExecute_FWD_DEFINED__
#define __IExecute_FWD_DEFINED__
typedef interface IExecute IExecute;
#endif	/* __IExecute_FWD_DEFINED__ */


#ifndef __Execute_FWD_DEFINED__
#define __Execute_FWD_DEFINED__

#ifdef __cplusplus
typedef class Execute Execute;
#else
typedef struct Execute Execute;
#endif /* __cplusplus */

#endif	/* __Execute_FWD_DEFINED__ */
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
/* header files for imported files */
#include "oaidl.h"
#include "ocidl.h"

void __RPC_FAR * __RPC_USER MIDL_user_allocate(size_t);
void __RPC_USER MIDL_user_free( void __RPC_FAR * );

/* interface __MIDL_itf_ExecuteDll_0000 */
/* [local] */

typedef /* [helpstring][uuid] */
enum ExecutionType
   { execODBC   = 0x1,
     execShell      = 0x2
   } ExecutionType;

typedef /* [helpstring][uuid] */
enum InstanceStatus
   { gintDisabled = 0x1,
     gintPending  = 0x2,
     gintRunning  = 0x3,
     gintComplete       = 0x4,
     gintFailed     = 0x5,
     gintAborted  = 0x6
   } InstanceStatus;

extern RPC_IF_HANDLE
__MIDL_itf_ExecuteDll_0000_v0_0_c_ifspec;
extern RPC_IF_HANDLE
__MIDL_itf_ExecuteDll_0000_v0_0_s_ifspec;

#ifndef __EXECUTEDLLLib_LIBRARY_DEFINED__
#define __EXECUTEDLLLib_LIBRARY_DEFINED__

/* library EXECUTEDLLLib */
/* [helpstring][version][uuid] */

EXTERN_C const IID LIBID_EXECUTEDLLLib;

#ifndef ___IExecuteEvents_DISPINTERFACE_DEFINED__
#define ___IExecuteEvents_DISPINTERFACE_DEFINED__

/* dispinterface _IExecuteEvents */
/* [helpstring][uuid] */

EXTERN_C const IID DIID__IExecuteEvents;

#if defined(__cplusplus) && !defined(CINTERFACE)

   MIDL_INTERFACE("551AC532-AB1C-11D2-BC0C-
00A0C90D2CA5")
   _IExecuteEvents : public IDispatch
   {
   };
#else      /* C style interface */
   typedef struct _IExecuteEventsVtbl
   {
      BEGIN_INTERFACE
      HRESULT ( STDMETHODCALLTYPE __RPC_FAR
*QueryInterface )(
         _IExecuteEvents __RPC_FAR * This,
         /* [in] */ REFIID riid,
         /* [iid_is][out] */ void __RPC_FAR *__RPC_FAR
*ppvObject);

      ULONG ( STDMETHODCALLTYPE __RPC_FAR
*AddRef )(
         _IExecuteEvents __RPC_FAR * This);

      ULONG ( STDMETHODCALLTYPE __RPC_FAR
*Release )(
         _IExecuteEvents __RPC_FAR * This);

      HRESULT ( STDMETHODCALLTYPE __RPC_FAR
*GetTypeInfoCount )(
         _IExecuteEvents __RPC_FAR * This,
         /* [out] */ UINT __RPC_FAR *pctinfo);

      HRESULT ( STDMETHODCALLTYPE __RPC_FAR
*GetTypeInfo )(
         _IExecuteEvents __RPC_FAR * This,
         /* [in] */ UINT iTInfo,
         /* [in] */ LCID lcid,
         /* [out] */ ITypeInfo __RPC_FAR *__RPC_FAR
*ppTInfo);

      HRESULT ( STDMETHODCALLTYPE __RPC_FAR
*GetIDsOfNames )(
         _IExecuteEvents __RPC_FAR * This,
         /* [in] */ REFIID riid,
         /* [size_is][in] */ LPOLESTR __RPC_FAR *rgszNames,
         /* [in] */ UINT cNames,
         /* [in] */ LCID lcid,
         /* [size_is][out] */ DISPID __RPC_FAR *rgDispId);

      /* [local] */ HRESULT ( STDMETHODCALLTYPE
__RPC_FAR *Invoke )(
         _IExecuteEvents __RPC_FAR * This,
         /* [in] */ DISPID dispIdMember,
         /* [in] */ REFIID riid,
         /* [in] */ LCID lcid,
         /* [in] */ WORD wFlags,
         /* [out][in] */ DISPPARAMS __RPC_FAR
*pDispParams,
         /* [out] */ VARIANT __RPC_FAR *pVarResult,
         /* [out] */ EXCEPINFO __RPC_FAR *pExcepInfo,
         /* [out] */ UINT __RPC_FAR *puArgErr);

      END_INTERFACE
   } _IExecuteEventsVtbl;

   interface _IExecuteEvents
   {
      CONST_VTBL struct _IExecuteEventsVtbl __RPC_FAR
*lpVtbl;
   };



#ifdef COBJMACROS
#define _IExecuteEvents_QueryInterface(This,riid,ppvObject)\
   (This)->lpVtbl -> QueryInterface(This,riid,ppvObject)

#define _IExecuteEvents_AddRef(This)\
   (This)->lpVtbl -> AddRef(This)
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
#define _IExecuteEvents_Release(This)\
   (This)->lpVtbl -> Release(This)

#define _IExecuteEvents_GetTypeInfoCount(This,pctinfo)      \
   (This)->lpVtbl -> GetTypeInfoCount(This,pctinfo)

#define _IExecuteEvents_GetTypeInfo(This,iTInfo,lcid,ppTInfo)
      \
   (This)->lpVtbl -> GetTypeInfo(This,iTInfo,lcid,ppTInfo)

#define
_IExecuteEvents_GetIDsOfNames(This,riid,rgszNames,cNames,
lcid,rgDispId)      \
   (This)->lpVtbl ->
GetIDsOfNames(This,riid,rgszNames,cNames,lcid,rgDispId)

#define
_IExecuteEvents_Invoke(This,dispIdMember,riid,lcid,wFlags,pD
ispParams,pVarResult,pExcepInfo,puArgErr)      \
   (This)->lpVtbl ->
Invoke(This,dispIdMember,riid,lcid,wFlags,pDispParams,pVarR
esult,pExcepInfo,puArgErr)

#endif /* COBJMACROS */

#endif    /* C style interface */

#endif    /*
___IExecuteEvents_DISPINTERFACE_DEFINED__ */

#ifndef __IExecute_INTERFACE_DEFINED__
#define __IExecute_INTERFACE_DEFINED__

/* interface IExecute */
/* [unique][helpstring][dual][uuid][object] */

EXTERN_C const IID IID_IExecute;

#if defined(__cplusplus) && !defined(CINTERFACE)

   MIDL_INTERFACE("551AC531-AB1C-11D2-BC0C-
00A0C90D2CA5")
   IExecute : public IDispatch
   {
   public:
      virtual /* [helpstring][id] */ HRESULT
STDMETHODCALLTYPE DoExecute(
         /* [in] */ BSTR szCommand,
         /* [in] */ BSTR szExecutionDtls,
         /* [in] */ ExecutionType ExecMethod,
         /* [in] */ BOOL bNoCount,
         /* [in] */ BOOL bNoExecute,
         /* [in] */ BOOL bParseOnly,
         /* [in] */ BOOL bQuotedIds,
         /* [in] */ BOOL bAnsiNulls,
         /* [in] */ BOOL bShowQP,
         /* [in] */ BOOL bStatsTime,
         /* [in] */ BOOL bStatsIO,
         /* [in] */ long lRowCount,
         /* [in] */ long lQueryTmout,
         /* [in] */ BSTR szConnection) = 0;

      virtual /* [helpstring][id][propget] */ HRESULT
STDMETHODCALLTYPE get_StepStatus(
         /* [retval][out] */ InstanceStatus __RPC_FAR *pVal) =
0;

      virtual /* [helpstring][id] */ HRESULT
STDMETHODCALLTYPE Abort( void) = 0;

      virtual /* [helpstring][id] */ HRESULT
STDMETHODCALLTYPE WriteError(
         BSTR szMsg) = 0;

      virtual /* [helpstring][id][propput] */ HRESULT
STDMETHODCALLTYPE put_OutputFile(
         /* [in] */ BSTR newVal) = 0;

      virtual /* [helpstring][id][propput] */ HRESULT
STDMETHODCALLTYPE put_ErrorFile(
         /* [in] */ BSTR newVal) = 0;
   };
#else     /* C style interface */

   typedef struct IExecuteVtbl
   {
      BEGIN_INTERFACE

      HRESULT ( STDMETHODCALLTYPE __RPC_FAR
*QueryInterface )(
         IExecute __RPC_FAR * This,
         /* [in] */ REFIID riid,
         /* [iid_is][out] */ void __RPC_FAR *__RPC_FAR
*ppvObject);

      ULONG ( STDMETHODCALLTYPE __RPC_FAR
*AddRef )(
         IExecute __RPC_FAR * This);
      ULONG ( STDMETHODCALLTYPE __RPC_FAR
*Release )(
         IExecute __RPC_FAR * This);
      HRESULT ( STDMETHODCALLTYPE __RPC_FAR
*GetTypeInfoCount )(
         IExecute __RPC_FAR * This,
         /* [out] */ UINT __RPC_FAR *pctinfo);

      HRESULT ( STDMETHODCALLTYPE __RPC_FAR
*GetTypeInfo )(
         IExecute __RPC_FAR * This,
         /* [in] */ UINT iTInfo,
         /* [in] */ LCID lcid,
         /* [out] */ ITypeInfo __RPC_FAR *__RPC_FAR
*ppTInfo);
      HRESULT ( STDMETHODCALLTYPE __RPC_FAR
*GetIDsOfNames )(
         IExecute __RPC_FAR * This,
         /* [in] */ REFIID riid,
         /* [size_is][in] */ LPOLESTR __RPC_FAR *rgszNames,
         /* [in] */ UINT cNames,
         /* [in] */ LCID lcid,
         /* [size_is][out] */ DISPID __RPC_FAR *rgDispId);
      /* [local] */ HRESULT ( STDMETHODCALLTYPE
__RPC_FAR *Invoke )(
         IExecute __RPC_FAR * This,
         /* [in] */ DISPID dispIdMember,
         /* [in] */ REFIID riid,
         /* [in] */ LCID lcid,
         /* [in] */ WORD wFlags,
         /* [out][in] */ DISPPARAMS __RPC_FAR
*pDispParams,
         /* [out] */ VARIANT __RPC_FAR *pVarResult,
         /* [out] */ EXCEPINFO __RPC_FAR *pExcepInfo,
         /* [out] */ UINT __RPC_FAR *puArgErr);
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
    /* [helpstring][id] */ HRESULT (
STDMETHODCALLTYPE __RPC_FAR *DoExecute )(
        IExecute __RPC_FAR * This,
        /* [in] */ BSTR szCommand,
         /* [in] */ BSTR szExecutionDtls,
         /* [in] */ ExecutionType ExecMethod,
         /* [in] */ BOOL bNoCount,
         /* [in] */ BOOL bNoExecute,
         /* [in] */ BOOL bParseOnly,
         /* [in] */ BOOL bQuotedIds,
         /* [in] */ BOOL bAnsiNulls,
         /* [in] */ BOOL bShowQP,
         /* [in] */ BOOL bStatsTime,
         /* [in] */ BOOL bStatsIO,
         /* [in] */ long lRowCount,
         /* [in] */ long lQueryTmout,
         /* [in] */ BSTR szConnection);

    /* [helpstring][id][propget] */ HRESULT (
STDMETHODCALLTYPE __RPC_FAR *get_StepStatus )(
        IExecute __RPC_FAR * This,
        /* [retval][out] */ InstanceStatus __RPC_FAR *pVal);

    /* [helpstring][id] */ HRESULT (
STDMETHODCALLTYPE __RPC_FAR *Abort )(
        IExecute __RPC_FAR * This);

    /* [helpstring][id] */ HRESULT (
STDMETHODCALLTYPE __RPC_FAR *WriteError )(
        IExecute __RPC_FAR * This,
        BSTR szMsg);

    /* [helpstring][id][propput] */ HRESULT (
STDMETHODCALLTYPE __RPC_FAR *put_OutputFile )(
        IExecute __RPC_FAR * This,
        /* [in] */ BSTR newVal);

    /* [helpstring][id][propput] */ HRESULT (
STDMETHODCALLTYPE __RPC_FAR *put_ErrorFile )(
        IExecute __RPC_FAR * This,
        /* [in] */ BSTR newVal);

    END_INTERFACE
  } IExecuteVtbl;

  interface IExecute
  {
    CONST_VTBL struct IExecuteVtbl __RPC_FAR *lpVtbl;
  };



#ifdef COBJMACROS
#define IExecute_QueryInterface(This,riid,ppvObject)     \
  (This)->lpVtbl -> QueryInterface(This,riid,ppvObject)

#define IExecute_AddRef(This)     \
  (This)->lpVtbl -> AddRef(This)

#define IExecute_Release(This)     \
  (This)->lpVtbl -> Release(This)

#define IExecute_GetTypeInfoCount(This,pctinfo)    \
  (This)->lpVtbl -> GetTypeInfoCount(This,pctinfo)

#define IExecute_GetTypeInfo(This,iTInfo,lcid,ppTInfo) \
  (This)->lpVtbl -> GetTypeInfo(This,iTInfo,lcid,ppTInfo)
#define
IExecute_GetIDsOfNames(This,riid,rgszNames,cNames,lcid,rgD
ispId)    \
  (This)->lpVtbl ->
GetIDsOfNames(This,riid,rgszNames,cNames,lcid,rgDispId)

#define
IExecute_Invoke(This,dispIdMember,riid,lcid,wFlags,pDispPara
ms,pVarResult,pExcepInfo,puArgErr)    \
  (This)->lpVtbl ->
Invoke(This,dispIdMember,riid,lcid,wFlags,pDispParams,pVarR
esult,pExcepInfo,puArgErr)

#define
IExecute_DoExecute(This,szCommand,szExecutionDtls,ExecMe
thod,bNoCount,
     bNoExecute,bParseOnly,bQuotedIds,bAnsiNulls,bShowQP,
bStatsTime,bStatsIO,lRowCount,lQueryTmout,szConnection)\
  (This)->lpVtbl ->
DoExecute(This,szCommand,szExecutionDtls,ExecMethod,bNo
Count,bNoExecute,bParseOnly,bQuotedIds,
     bAnsiNulls,bShowQP,bStatsTime,bStatsIO,lRowCount,lQue
ryTmout,szConnection)

#define IExecute_get_StepStatus(This,pVal) \
  (This)->lpVtbl -> get_StepStatus(This,pVal)

#define IExecute_Abort(This)  \
  (This)->lpVtbl -> Abort(This)

#define IExecute_WriteError(This,szMsg)    \
  (This)->lpVtbl -> WriteError(This,szMsg)

#define IExecute_put_OutputFile(This,newVal)  \
  (This)->lpVtbl -> put_OutputFile(This,newVal)

#define IExecute_put_ErrorFile(This,newVal)     \
  (This)->lpVtbl -> put_ErrorFile(This,newVal)

#endif /* COBJMACROS */

#endif    /* C style interface */

/* [helpstring][id] */ HRESULT STDMETHODCALLTYPE
IExecute_DoExecute_Proxy(
  IExecute __RPC_FAR * This,
  /* [in] */ BSTR szCommand,
  /* [in] */ BSTR szExecutionDtls,
  /* [in] */ ExecutionType ExecMethod,
  /* [in] */ BOOL bNoCount,
  /* [in] */ BOOL bNoExecute,
  /* [in] */ BOOL bParseOnly,
  /* [in] */ BOOL bQuotedIds,
  /* [in] */ BOOL bAnsiNulls,
  /* [in] */ BOOL bShowQP,
  /* [in] */ BOOL bStatsTime,
  /* [in] */ BOOL bStatsIO,
  /* [in] */ long lRowCount,
  /* [in] */ long lQueryTmout,
  /* [in] */ BSTR szConnection);

void __RPC_STUB IExecute_DoExecute_Stub(
  IRpcStubBuffer *This,
  IRpcChannelBuffer *_pRpcChannelBuffer,
  PRPC_MESSAGE _pRpcMessage,
  DWORD *_pdwStubPhase);
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
/* [helpstring][id][propget] */ HRESULT
STDMETHODCALLTYPE IExecute_get_StepStatus_Proxy(
    IExecute __RPC_FAR * This,
    /* [retval][out] */ InstanceStatus __RPC_FAR *pVal);


void __RPC_STUB IExecute_get_StepStatus_Stub(
    IRpcStubBuffer *This,
    IRpcChannelBuffer *_pRpcChannelBuffer,
    PRPC_MESSAGE _pRpcMessage,
    DWORD *_pdwStubPhase);

/* [helpstring][id] */ HRESULT STDMETHODCALLTYPE
IExecute_Abort_Proxy(
    IExecute __RPC_FAR * This);

void __RPC_STUB IExecute_Abort_Stub(
    IRpcStubBuffer *This,
    IRpcChannelBuffer *_pRpcChannelBuffer,
    PRPC_MESSAGE _pRpcMessage,
    DWORD *_pdwStubPhase);

/* [helpstring][id] */ HRESULT STDMETHODCALLTYPE
IExecute_WriteError_Proxy(
    IExecute __RPC_FAR * This,
    BSTR szMsg);

void __RPC_STUB IExecute_WriteError_Stub(
    IRpcStubBuffer *This,
    IRpcChannelBuffer *_pRpcChannelBuffer,
    PRPC_MESSAGE _pRpcMessage,
    DWORD *_pdwStubPhase);

/* [helpstring][id][propput] */ HRESULT
STDMETHODCALLTYPE IExecute_put_OutputFile_Proxy(
    IExecute __RPC_FAR * This,
    /* [in] */ BSTR newVal);

void __RPC_STUB IExecute_put_OutputFile_Stub(
    IRpcStubBuffer *This,
    IRpcChannelBuffer *_pRpcChannelBuffer,
    PRPC_MESSAGE _pRpcMessage,
    DWORD *_pdwStubPhase);


/* [helpstring][id][propput] */ HRESULT
STDMETHODCALLTYPE IExecute_put_ErrorFile_Proxy(
    IExecute __RPC_FAR * This,
    /* [in] */ BSTR newVal);

void __RPC_STUB IExecute_put_ErrorFile_Stub(
    IRpcStubBuffer *This,
    IRpcChannelBuffer *_pRpcChannelBuffer,
    PRPC_MESSAGE _pRpcMessage,
    DWORD *_pdwStubPhase);
#endif    /* __IExecute_INTERFACE_DEFINED__ */
EXTERN_C const CLSID CLSID_Execute;
ifdef __cplusplus
class DECLSPEC_UUID("2EFC198E-AA8D-11D2-BC0C-
00A0C90D2CA5")
Execute;
#endif
#endif /* __EXECUTEDLLLib_LIBRARY_DEFINED__ */
/* Additional Prototypes for ALL interfaces */
/* end of Additional Prototypes */
#ifdef __cplusplus
}
#endif
```

```
//  FILE:      Execute.cpp
//             Microsoft TPC-H Kit Ver. 1.00
//             Copyright Microsoft, 1999
//             All Rights Reserved
//
//
//  PURPOSE:    Implementation of CExecute.
//  Contact:    Reshma Tharamal (reshmat@microsoft.com)
//
#include "stdafx.h"

#include "ExecuteDll.h"
#include "SMExecute.h"
#include "Execute.h"

extern SQLHENV henv;
extern SM_Connection_Info     *p_Connections;  // Pointer to
open connections
extern int      iConnectionCount;// Number of open connections
extern CRITICAL_SECTION         hConnections;     // Critical
section to serialize
      // access to available connections

#ifdef _TPCH_AUDIT
      extern FILE *pfLogFile;                  // Log file containing
timestamps
      extern CRITICAL_SECTION hLogFileWrite;     // Handle to
critical section
#endif

/////////////////////////////////////////////////////////////////////////////
// CExecute

char * g_szOdbcOps[] = {
            "SQLAllocHandle",
            "SQLDriverConnect",
            "SQLExecDirect",
            "SQLSetStmtAttr",
            "SQLCancel",
            "SQLNumResultCols",
            "SQLDescribeCol",
            "SQLColAttribute",
            "SQLFetch",
            "SQLGetData",
            "SQLRowCount",
            "SQLMoreResults",
            "SQLBindCol"
      };

char * CExecError::m_szExecErrorDesc[] = {
            "Connection is already in use."
      };
STDMETHODIMP
CExecute::InterfaceSupportsErrorInfo(REFIID riid)
{
      static const IID* arr[] =
      {
      &IID_IExecute
      };
      for (int i=0; i < sizeof(arr) / sizeof(arr[0]); i++)
      {
            if (InlineIsEqualGUID(*arr[i],riid))
                  return S_OK;
      }
      return S_FALSE;
}
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
STDMETHODIMP CExecute::put_OutputFile(BSTR newVal)
{
	assert(m_pOutputFile);
	m_OutputFile = newVal;

	HRESULT	hr = m_pOutputFile->put_FileName(newVal);
	if FAILED(hr)
	{
		m_pOutputFile->Release();
		m_pOutputFile = NULL;
	}
	return hr;
}

//DEL STDMETHODIMP CExecute::put_LogFile(BSTR
newVal)
//DEL {
//DEL	assert(m_pLogFile);
//DEL
//DEL	m_pLogFile->put_FileName(newVal);
//DEL	return S_OK;
//DEL }

STDMETHODIMP CExecute::put_ErrorFile(BSTR newVal)
{
	assert(m_pErrorFile);
	m_ErrorFile = newVal;

	HRESULT	hr = m_pErrorFile->put_FileName(newVal);
	if FAILED(hr)
	{
		m_pErrorFile->Release();
		m_pErrorFile = NULL;
	}
	return hr;
}

STDMETHODIMP CExecute::DoExecute(BSTR szCommand,
BSTR szExecutionDtls, ExecutionType ExecMethod, \
 BOOL bNoCount, BOOL bNoExecute, BOOL bParseOnly,
BOOL bQuotedIds, \
 BOOL bAnsiNulls, BOOL bShowQP, BOOL bStatsTime,
BOOL bStatsIO, \
 long lRowCount, long lQueryTmout, BSTR szConnection)
{
   HANDLE	hThrd;
   DWORD	tid;
  _CrtSetReportFile(_CRT_WARN,
_CRTDBG_FILE_STDOUT);
	m_szCommand = szCommand;
	m_szExecDtls = szExecutionDtls;
	m_ExecMthd = ExecMethod;
	if (m_ExecMthd == execODBC)
   {
	 m_bNoCount = bNoCount;
		m_bNoExecute = bNoExecute;
		m_bParseOnly = bParseOnly;
		m_bQuotedIds = bQuotedIds;
		m_bAnsiNulls = bAnsiNulls;
		m_bShowQP = bShowQP;
		m_bStatsTime = bStatsTime;
		m_bStatsIO = bStatsIO;
		m_lRowCount = lRowCount;
		m_lQueryTmout = lQueryTmout;
		m_szConnection = szConnection;
   }
	if((hThrd = CreateThread( 0, 0,
(LPTHREAD_START_ROUTINE)ExecutionThread,
		this, 0, &tid)) == NULL)
		return(RaiseSystemError());

	CloseHandle(hThrd);

	return S_OK;
}

STDMETHODIMP CExecute::Abort()
{
	if (m_ExecMthd == execShell)
		return(AbortShell());
	else
		return(AbortODBC());
}

void ExecutionThread(LPVOID lpParameter)
{

	CExecute	*MyExecute = (CExecute*)lpParameter;

	MyExecute->m_tElapsedTime = 0;

	GetLocalTime(&MyExecute->m_tStartTime);
	MyExecute->PostMessage(WM_TASK_START, 0, 0);

#ifdef _TPCH_AUDIT
	char	szBuffer[MAXLOGCMDBUF];
	char	szFmt[MAXBUFLEN];

	sprintf(szFmt, "Start Step: '%%.%ds' at '%d/%d/%d
%d:%d:%d:%d'\n",
		MAXLOGCMDLEN,
		MyExecute->m_tStartTime.wMonth, MyExecute-
>m_tStartTime.wDay,
		MyExecute->m_tStartTime.wYear, MyExecute-
>m_tStartTime.wHour,
		MyExecute->m_tStartTime.wMinute, MyExecute-
>m_tStartTime.wSecond,
		MyExecute->m_tStartTime.wMilliseconds);
	if (MyExecute->m_ExecMthd == execShell)
		WriteFileToTpchLog((LPSTR)MyExecute-
>m_szCommand, szFmt);
	else
	{
		sprintf(szBuffer, szFmt, (LPSTR)MyExecute-
>m_szCommand);
		WriteToTpchLog(szBuffer);
	}
#endif
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
// Initialize the run status for the step to running. The completion
status for the step will be initialized by the Shell and ODBC
//execution functions.
   MyExecute->m_StepStatus = gintRunning;

      if (MyExecute->m_ExecMthd == execShell)
            MyExecute->m_tElapsedTime = MyExecute-
>ExecuteShell();
      else
            MyExecute->m_tElapsedTime = MyExecute-
>ExecuteODBC();

      // Close the output, log and error files
      if (MyExecute->m_pOutputFile)
            MyExecute->m_pOutputFile->Release();
      MyExecute->m_pOutputFile = NULL;

      MyExecute->m_ExecTime = NULL;

      GetLocalTime(&MyExecute->m_tEndTime);

#ifdef _TPCH_AUDIT
      sprintf(szFmt, "Complete Step: '%%.%ds' at '%d/%d/%d
%d:%d:%d:%d'\n",
            MAXLOGCMDLEN,
            MyExecute->m_tEndTime.wMonth, MyExecute-
>m_tEndTime.wDay,
            MyExecute->m_tEndTime.wYear, MyExecute-
>m_tEndTime.wHour,
            MyExecute->m_tEndTime.wMinute, MyExecute-
>m_tEndTime.wSecond,
            MyExecute->m_tEndTime.wMilliseconds);
      if (MyExecute->m_ExecMthd == execShell)
            WriteFileToTpchLog((LPSTR)MyExecute-
>m_szCommand, szFmt);
      else
      {
            sprintf(szBuffer, szFmt, (LPSTR)MyExecute-
>m_szCommand);
            WriteToTpchLog(szBuffer);
      }
#endif

      MyExecute->PostMessage(WM_TASK_FINISH, 0, 0);

      return;
}

#ifdef _TPCH_AUDIT

void WriteFileToTpchLog(LPSTR szFile, LPSTR szFmt)
{
      // Reads a maximum of MAXLOGCMDBUF characters
from the command file and writes it to the log
      FILE      *fpCmd;
      int       iRead;
      charszBuf[MAXLOGCMDBUF];
      charszCmd[MAXLOGCMDLEN];
      if ( pfLogFile != NULL )
      {
            if ( (fpCmd = fopen(szFile,
FILE_ACCESS_READ)) != NULL)
            {
                  iRead = fread(szCmd, sizeof(char), sizeof(szCmd) /
sizeof(char), fpCmd);
                  if (iRead < MAXLOGCMDLEN)
      szCmd[iRead] = '\0';
      else
      szCmd[MAXLOGCMDLEN - 1] = '\0';
      sprintf(szBuf, szFmt, szCmd);
      WriteToTpchLog(szBuf);
      fclose(fpCmd);
      }
      }
}

void WriteToTpchLog(char *szMsg)
{
      if (pfLogFile != NULL)
      {
      EnterCriticalSection(&hLogFileWrite);
      fprintf(pfLogFile, szMsg);
      LeaveCriticalSection(&hLogFileWrite);
      }

      return;
}
#endif

TC_TIME CExecute::ExecuteShell()
{
      STARTUPINFOA            Start;
      PROCESS_INFORMATION   proc;
      DWORD             exitCode;
      TC_TIME           tElapsed = 0;
      _bstr_t         szCommand("cmd /c ");
      LPSTR         szStartDir;
      CURRENC   Elapsed;
      szCommand += m_szCommand;
// Redirect output and error information
szCommand += " > " + m_OutputFile + " 2> " + m_ErrorFile;
// Initialize the STARTUPINFO structure:
memset(&Start, 0, sizeof(STARTUPINFOA)) ;
Start.cb            = sizeof(Start);
Start.dwFlags         = STARTF_USESHOWWINDOW;
Start.wShowWindow     = SW_SHOWMINNOACTIVE;
memset(&proc, 0, sizeof(PROCESS_INFORMATION)) ;
szStartDir = strcmp((LPCTSTR)m_szExecDtls, "") == 0 ? NULL
: (LPSTR)m_szExecDtls;
m_ExecTime->Start();
// Start the shelled application:
if (!CreateProcessA( NULL, (LPSTR)szCommand, NULL,
NULL, FALSE,
NORMAL_PRIORITY_CLASS, NULL, szStartDir, &Start,
&proc ))
{
            m_StepStatus = gintFailed;
            LogSystemError(m_pErrorFile);

            m_ExecTime->Stop(&Elapsed);
            return((TC_TIME)Elapsed.int64);
      }
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
	m_hHandle = proc.hProcess;
	// Give the process time to execute and finish
	WaitForSingleObject(m_hHandle, INFINITE);
	m_ExecTime->Stop(&Elapsed);

	if (!GetExitCodeProcess(m_hHandle, &exitCode))
	{
		m_StepStatus = gintFailed;
		LogSystemError(m_pErrorFile);
	}
	else
		m_StepStatus = gintComplete;

	// Close all open handles to the shelled process
	CloseHandle(m_hHandle);

	return((TC_TIME)Elapsed.int64);
}

STDMETHODIMP CExecute::AbortShell()
{
	if (m_hHandle != SQL_NULL_HSTMT)
		if (!TerminateProcess(m_hHandle, 0))
			return(RaiseSystemError());

	return(S_OK);
}

TC_TIME CExecute::ExecuteODBC()
{
	TC_TIME		tElapsed = 0;
	HDBC		m_hdbc;
	SQLRETURN		rc;
	LPSTR		szCmd;
	CURRENCY		Elapsed;
	BOOL		bDoConnect = FALSE;

	// ODBC specific initialization
	m_hdbc = SQL_NULL_HDBC;

	try
	{
		// Allocate a new connection if we are creating a
dynamic connection or if
		// the named connection doesn't exist
		InitializeConnection(&m_hdbc, &bDoConnect);

		// Ensure that the connection is valid.

#ifdef _DEBUG
		_CrtDbgReport(_CRT_WARN, NULL, 0, NULL,
"Executing SQLDriverConnect.\n");
#endif

		if (bDoConnect)
		{
			// Allocate connection handle, open a connection
and set connection attributes.
#ifdef _DEBUG
			_CrtDbgReport(_CRT_WARN, NULL, 0, NULL,
"Executing SQLDriverConnect.\n");
#endif
if (m_bAbort)
return(tElapsed);
// Connect to the server using the passed in connection string
rc = SQLDriverConnect(m_hdbc, NULL,
(unsigned char *)(LPSTR)m_szExecDtls, SQL_NTS,
NULL, 0, NULL, SQL_DRIVER_NOPROMPT);
HandleODBCError(rc, SQL_HANDLE_DBC, m_hdbc,
SMSQLDriverConnect);
		}

		ReConnectDeadConnection(&m_hdbc);

#ifdef _DEBUG
_CrtDbgReport(_CRT_WARN, NULL, 0, NULL, "Executing
SQLAllocHandle for hdbc.\n");
#endif
if (!m_bAbort && (rc =
SQLAllocHandle(SQL_HANDLE_STMT, m_hdbc,
&m_hHandle)) != SQL_SUCCESS)
HandleODBCError(rc, SQL_HANDLE_DBC, m_hdbc,
SMSQLAllocHandle);
// Set connection attributes if any have been modified from the
default values
		if (m_lRowCount > 0)
		{
		char	szConnOptions[512];
			sprintf(szConnOptions, "SET ROWCOUNT %d ",
m_lRowCount);
		SetConnectionOption(szConnOptions, &m_hdbc);
		}

		if (m_bQuotedIds)
		SetConnectionOption("SET QUOTED_IDENTIFIER ON ",
&m_hdbc);
		if (!m_bAnsiNulls)
		SetConnectionOption("SET ANSI_NULL_DFLT_OFF ON
", &m_hdbc);

		if (!m_bAbort && m_lQueryTmout > 0)
		{
#ifdef _DEBUG
		_CrtDbgReport(_CRT_WARN, NULL, 0, NULL,
"Executing SQLSetStmtAttr.\n");
#endif
		// Set the query timeout on the statement handle
		rc = SQLSetStmtAttr(m_hHandle,
SQL_ATTR_QUERY_TIMEOUT, &m_lQueryTmout,
		SQL_IS_UINTEGER);
		HandleODBCError(rc, SQL_HANDLE_STMT, m_hHandle,
SMSQLSetStmtAttr);
		}

		if (m_bNoExecute)
		SetConnectionOption("SET NOEXEC ON ", &m_hdbc);
		else if (m_bParseOnly)
		SetConnectionOption("SET PARSEONLY ON ",
&m_hdbc);
		else if (m_bShowQP)
		// Important to ensure that this is the last connection
attributes being set -
		// otherwise showplans are generated for all remaining SET
statements
		SetConnectionOption("SET SHOWPLAN_TEXT ON ",
&m_hdbc);
		else
		{
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
                if (m_bNoCount)
                        SetConnectionOption("SET NOCOUNT ON ",
&m_hdbc);

                if (m_bStatsIO)
                        SetConnectionOption("SET STATISTICS IO
ON ", &m_hdbc);

                // Important to ensure that this is the last connection
attributes being set -
                // otherwise timing statistics are generated for all
remaining SET statements
                if (m_bStatsTime)
                        SetConnectionOption("SET STATISTICS
TIME ON ", &m_hdbc);
            }

            m_szCmd = (LPSTR)m_szCommand;
            m_ExecTime->Start();

            while ((szCmd =
NextCmdInBatch((LPSTR)m_szCommand)) != NULL &&
!m_bAbort)
            {
#ifdef _DEBUG
                _CrtDbgReport(_CRT_WARN, NULL, 0, NULL,
"Executing SQLExecDirect.\n");
#endif

                // Execute the ODBC command
                rc = SQLExecDirect(m_hHandle, (unsigned char
*)szCmd, SQL_NTS);
                HandleODBCError(rc, SQL_HANDLE_STMT,
m_hHandle, SMSQLExecDirect);

                free(szCmd);

                // Call a procedure to log the results to the output
file
                ProcessResultsets();
            }
            m_ExecTime->Stop(&Elapsed);

            ResetConnectionProperties(&m_hdbc);
        }
        catch(CODBCError *pErr)
        {
            m_StepStatus = gintFailed;
            delete pErr;
        }
        catch(CExecError *pErr)
        {
            m_StepStatus = gintFailed;
            pErr->LogErrors(this);
            delete pErr;
        }

        ODBCCleanup(&m_hdbc, &m_hHandle);

        if (m_StepStatus != gintFailed)
            m_StepStatus = gintComplete;

        return((DWORD)Elapsed.int64);
}
```

```
void CExecute::InitializeConnection(HDBC *phdbc, BOOL
*pbDoConnect)
{
    SQLRETURN               rc;

    *pbDoConnect = TRUE;

    if (IsDynamicConnection())
    {
#ifdef _DEBUG
        _CrtDbgReport(_CRT_WARN, NULL, 0, NULL,
"Executing SQLAllocHandle for m_hdbc.\n");
#endif

        rc = SQLAllocHandle(SQL_HANDLE_DBC, henv,
phdbc);
        HandleODBCError(rc, SQL_HANDLE_ENV, henv,
SMSQLAllocHandle);

        return;
    }

    EnterCriticalSection(&hConnections);
    // Returns the connection handle if the connection,
m_szConnection, exists
    for (m_iConnectionIndex = iConnectionCount - 1;
m_iConnectionIndex >= 0; m_iConnectionIndex--)
    {
        if (!strcmp( (p_Connections + m_iConnectionIndex)-
>szConnectionName, (LPSTR)m_szConnection))
        {
            if (!(p_Connections + m_iConnectionIndex)-
>bInUse)
            {
                *phdbc = (p_Connections +
m_iConnectionIndex)->hdbc;
                (p_Connections + m_iConnectionIndex)-
>bInUse = TRUE;

                *pbDoConnect = FALSE;
                break;
            }
            else
            {
                LeaveCriticalSection(&hConnections);

                throw new
CExecError(CExecError::SM_ERR_CONN_IN_USE);
            }
        }
    }

    if (m_iConnectionIndex < 0)
    {
        // Connection was not found. Allocate connection
handle and add it to list of
        // available connections.
#ifdef _DEBUG
        _CrtDbgReport(_CRT_WARN, NULL, 0, NULL,
"Executing SQLAllocHandle for m_hdbc.\n");
#endif
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
		rc = SQLAllocHandle(SQL_HANDLE_DBC, henv,
phdbc);
		HandleODBCError(rc, SQL_HANDLE_ENV, henv,
SMSQLAllocHandle);

		m_iConnectionIndex = iConnectionCount++;

		p_Connections = (SM_Connection_Info
*)realloc(p_Connections,
			iConnectionCount *
sizeof(SM_Connection_Info));

		strcpy((p_Connections + m_iConnectionIndex)-
>szConnectionName, (LPSTR)m_szConnection);
		(p_Connections + m_iConnectionIndex)->hdbc =
*phdbc;
		(p_Connections + m_iConnectionIndex)->bInUse =
TRUE;
	}

	LeaveCriticalSection(&hConnections);

	return;
}

void CExecute::ReConnectDeadConnection(HDBC *phdbc)
{
	SQLRETURN		rc;
	SQLUINTEGER		uConnDead;

	// Connect to the server using the passed in connection string
	rc = SQLGetConnectAttr(*phdbc,
SQL_ATTR_CONNECTION_DEAD,
		&uConnDead, SQL_IS_UINTEGER, NULL);
	HandleODBCError(rc, SQL_HANDLE_DBC, *phdbc,
SMSQLDriverConnect);

	if (uConnDead == SQL_CD_TRUE)
	{
		// Cleanup the old connection and re-connect.
#ifdef _DEBUG
		_CrtDbgReport(_CRT_WARN, NULL, 0, NULL,
"Executing SQLDisconnect.\n");
#endif
		rc = SQLDisconnect(*phdbc);
#ifdef _DEBUG
		_CrtDbgReport(_CRT_WARN, NULL, 0, NULL,
"Executing SQLFreeHandle for hdbc.\n");
#endif
		SQLFreeHandle(SQL_HANDLE_DBC, *phdbc);
		*phdbc = SQL_NULL_HDBC;

#ifdef _DEBUG
		_CrtDbgReport(_CRT_WARN, NULL, 0, NULL,
"Executing SQLDriverConnect.\n");
#endif
		rc = SQLAllocHandle(SQL_HANDLE_DBC, henv,
phdbc);
		HandleODBCError(rc, SQL_HANDLE_ENV, henv,
SMSQLAllocHandle);

		// Connect to the server using the passed in connection
string
		rc = SQLDriverConnect(*phdbc, NULL,
			(unsigned char *)(LPSTR)m_szExecDtls,
SQL_NTS,
			NULL, 0, NULL, SQL_DRIVER_NOPROMPT);
		HandleODBCError(rc, SQL_HANDLE_DBC, *phdbc,
SMSQLDriverConnect);
	}

	return;
}

void CExecute::ResetConnectionUsage()
{
	if(m_iConnectionIndex >= 0 && m_iConnectionIndex <
iConnectionCount)
	{
		EnterCriticalSection(&hConnections);
		(p_Connections + m_iConnectionIndex)->bInUse =
FALSE;
		LeaveCriticalSection(&hConnections);
	}

	return;
}

void CExecute::ResetConnectionProperties(HDBC *p_hdbc)
{
	SQLRETURN		rc;

	// Reset connection attributes if any have been modified from
the default values

	if (m_bNoExecute)
		SetConnectionOption("SET NOEXEC OFF ", p_hdbc);
	else if (m_bParseOnly)
		SetConnectionOption("SET PARSEONLY OFF ",
p_hdbc);
	else if (m_bShowQP)
		// Reset connection attributes in reverse order
		SetConnectionOption("SET SHOWPLAN_TEXT OFF
", p_hdbc);
	else
	{
		// Reset connection attributes in reverse order
		if (m_bStatsTime)
			SetConnectionOption("SET STATISTICS TIME
OFF ", p_hdbc);

		if (m_bNoCount)
			SetConnectionOption("SET NOCOUNT OFF ",
p_hdbc);

		if (m_bStatsIO)
			SetConnectionOption("SET STATISTICS IO OFF
", p_hdbc);
	}
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
	if (m_lRowCount > 0)
	{
		char		szConnOptions[512];

		sprintf(szConnOptions, "SET ROWCOUNT 0 ");
		SetConnectionOption(szConnOptions, p_hdbc);
	}

	if (m_bQuotedIds)
		SetConnectionOption("SET QUOTED_IDENTIFIER
OFF ", p_hdbc);

	if (!m_bAnsiNulls)
		SetConnectionOption("SET ANSI_NULL_DFLT_OFF
OFF ", p_hdbc);

	if (m_lQueryTmout > 0)
	{
		SQLUINTEGER	lQueryTmout = 0;

#ifdef _DEBUG
		_CrtDbgReport(_CRT_WARN, NULL, 0, NULL,
"Executing SQLSetStmtAttr.\n");
#endif

		// Set the query timeout on the statement handle
		rc = SQLSetStmtAttr(m_hHandle,
SQL_ATTR_QUERY_TIMEOUT, &lQueryTmout,
			SQL_IS_UINTEGER);
		HandleODBCError(rc, SQL_HANDLE_STMT,
m_hHandle, SMSQLSetStmtAttr);
	}

	return;
}

LPSTR CExecute::NextCmdInBatch(LPSTR szBatch)
{
	LPSTR	szCmd, szSeparator, szStart;
	charszNext;

	szStart = m_szCmd;

	while ( (szSeparator = strstr(szStart, CMD_SEPARATOR))
!= NULL)
	{
		szNext = *(szSeparator + strlen(CMD_SEPARATOR));
		if ( szNext == '\n' || szNext == '\r' || szNext == '\0')
			break;
		else
			szStart = szSeparator +
strlen(CMD_SEPARATOR);
	}
	if (!szSeparator)
	{
		// No more GO's
		if (strlen(m_szCmd) > 0)
		{
			szCmd = (LPSTR)malloc(strlen(m_szCmd) + 1);
			strcpy(szCmd, m_szCmd);
			m_szCmd += strlen(m_szCmd);
		}
		else
			szCmd = NULL;
	}
	else if (szSeparator - m_szCmd > 0)
	{
		// Strip the succeeding newline
		szCmd = (LPSTR)malloc(szSeparator - m_szCmd);
		strncpy(szCmd, m_szCmd, szSeparator - m_szCmd - 1);
		*(szCmd + (szSeparator - m_szCmd - 1)) = '\0';
		m_szCmd += szSeparator - m_szCmd +
strlen(CMD_SEPARATOR);
		if ( szNext == '\n' || szNext == '\r')
			m_szCmd += 1;
	}
	else
		szCmd = NULL;

	return(szCmd);
}

void CExecute::SetConnectionOption(LPSTR szConn, HDBC
*pHdbc)
{
	// Executes the passed in connection options 'set' statement.
Returns True if it succeeded
	char		szConnOptions[512];
	SQLRETURN		rc;

	sprintf(szConnOptions, szConn);

#ifdef _DEBUG
	_CrtDbgReport(_CRT_WARN, NULL, 0, NULL,
"Executing SQLExecDirect for connection option.\n");
#endif

	if (m_bAbort)
		return;

	rc = SQLExecDirect(m_hHandle, (unsigned char
*)szConnOptions, SQL_NTS);
	HandleODBCError(rc, SQL_HANDLE_STMT, m_hHandle,
SMSQLExecDirect);

	return;
}
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
STDMETHODIMP CExecute::AbortODBC()
{
	m_bAbort = TRUE;

	try
	{
		if (m_hHandle != SQL_NULL_HSTMT)
		{
#ifdef _DEBUG
			_CrtDbgReport(_CRT_WARN, NULL, 0, NULL,
"Executing SQLCancel.\n");
#endif

			SQLRETURN rc = SQLCancel(m_hHandle);
			HandleODBCError(rc, SQL_HANDLE_STMT,
m_hHandle, SMSQLCancel);
		}
	}
	catch(CODBCError *pErr)
	{
		delete pErr;
	}

	return(S_OK);
}

void CExecute::ProcessResultsets()
{
	SQLSMALLINT		*CTypeArray, *CScaleArray;
	SQLINTEGER		*ColLenArray, *DispLenArray,
*OffsetArray;
	SQLSMALLINT		iColNameLen, SQLType, iColNull, i,
NumCols = 0;
	SQLINTEGER		iDispLen, iRowCount;
	SQLRETURN		rc;
	char			szColName[MAX_DATA_LEN + 1];
	void			*DataPtr;
	SQLINTEGER		iLenOrInd =
ALIGNBUF(sizeof(SQLINTEGER));
	SQLUINTEGER		iRowArraySize, iArrayElementSize;
//	SQLUINTEGER		NumRowsFetched;
//	SQLUSMALLINT		*RowStatusArray;


	if (!m_pOutputFile || m_bAbort)
		return;

	do
	{
#ifdef _DEBUG
		_CrtDbgReport(_CRT_WARN, NULL, 0, NULL,
"Executing SQLNumResultCols.\n");
#endif
		// Determine the number of result set columns.
		rc = SQLNumResultCols(m_hHandle, &NumCols);
		HandleODBCError(rc, SQL_HANDLE_STMT,
m_hHandle, SMSQLNumResultCols);
	if (NumCols > 0)
	{
	// Allocate arrays to hold the C type, scale, column and
display length of the data
	CTypeArray = (SQLSMALLINT *) malloc(NumCols *
sizeof(SQLSMALLINT));
	CScaleArray = (SQLSMALLINT *) malloc(NumCols *
sizeof(SQLSMALLINT));
	ColLenArray = (SQLINTEGER *) malloc(NumCols *
sizeof(SQLINTEGER));
	DispLenArray = (SQLINTEGER *) malloc(NumCols *
sizeof(SQLINTEGER));
	OffsetArray = (SQLINTEGER *) malloc(NumCols *
sizeof(SQLINTEGER));
	OffsetArray[0] = 0;

	for (i = 0; i < NumCols && !m_bAbort; i++)
	{
#ifdef _DEBUG
	_CrtDbgReport(_CRT_WARN, NULL, 0, NULL,
"Executing SQLDescribeCol.\n");
#endif

	// Get the column description, include the SQL type
	// Determine the column's byte length. Calculate the offset in
the buffer to the
	// data as the offset to the previous column, plus the byte
length of the previous
	// column, plus the byte length of the previous column's
length/indicator buffer.
	// Note that the byte length of the column and the
length/indicator buffer are increased
	// so that, assuming they start on an alignment boundary,
they will end on the byte
	// before the next alignment boundary. Although this might
leave some holes in the
	// buffer, it is a relatively inexpensive way to guarantee
alignment.
	rc = SQLDescribeCol(m_hHandle, ((SQLUSMALLINT)
i)+1,
		(unsigned char *)szColName, sizeof(szColName),
&iColNameLen,
		&SQLType, (unsigned long *)&ColLenArray[i],
&CScaleArray[i], &iColNull);
	HandleODBCError(rc, SQL_HANDLE_STMT, m_hHandle,
SMSQLDescribeCol);

#ifdef _DEBUG
	_CrtDbgReport(_CRT_WARN, NULL, 0, NULL,
"Executing SQLColAttribute.\n");
#endif

	if (m_bAbort)
		return;

		rc = SQLColAttribute(m_hHandle,
((SQLUSMALLINT) i)+1, SQL_DESC_DISPLAY_SIZE,
NULL, 0,
			NULL, &iDispLen);
		HandleODBCError(rc, SQL_HANDLE_STMT,
m_hHandle, SMSQLColAttribute);
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
		// GetDefaultCType contains a switch statement that
returns the default C type
		// for each SQL type.
		CTypeArray[i] = GetDefaultCType(SQLType);
		if ( (CTypeArray[i] == SQL_C_CHAR || CTypeArray[i]
== SQL_C_BINARY) &&
				ColLenArray[i] > MAX_DATA_LEN)
		{
			ColLenArray[i] = MAX_DATA_LEN;
			iDispLen = MAX_DATA_LEN;
		}

		DispLenArray[i] = max(iColNameLen, iDispLen);
		DispLenArray[i] = max(DispLenArray[i],
sizeof(S_NULL));

		// Print the column names in the header
		PrintData(szColName, SQL_C_CHAR,
DispLenArray[i], 0, m_pOutputFile);

		// Add a byte for the null-termination character
		ColLenArray[i] += 1;
		ColLenArray[i] = ALIGNBUF(ColLenArray[i]);

		// Calculate the offset in the buffer to the data as the
offset to the previous column,
		// plus the byte length of the previous column, plus the
byte length of the previous
		// column's length/indicator buffer.
		if (i)	OffsetArray[i] = OffsetArray[i-1] +
ColLenArray[i-1] + iLenOrInd;
		}
		m_pOutputFile->WriteLine(NULL);
		iArrayElementSize = OffsetArray[NumCols-1] +
ColLenArray[NumCols-1] + iLenOrInd;
		iRowArraySize = 1;
		// Allocate the data buffer. The size of the buffer is equal
to the offset to the data
		// buffer for the final column, plus the byte length of the
data buffer and length/indicator
		// buffer for the last column.
		DataPtr = malloc(iRowArraySize * iArrayElementSize);
		// Specify the size of the structure with the
SQL_ATTR_ROW_BIND_TYPE
		// statement attribute. This also declares that row-wise
binding will
		// be used. Declare the rowset size with the
SQL_ATTR_ROW_ARRAY_SIZE
		// statement attribute. Set the
SQL_ATTR_ROW_STATUS_PTR statement
		// attribute to point to the row status array. Set the
		// SQL_ATTR_ROWS_FETCHED_PTR statement
attribute to point to
		// NumRowsFetched.
		/*
		RowStatusArray = (SQLUSMALLINT
*)malloc(iRowArraySize * sizeof(SQLUSMALLINT));
	SQLSetStmtAttr(m_hHandle,
SQL_ATTR_ROW_BIND_TYPE, &iArrayElementSize,
SQL_IS_UINTEGER);
	SQLSetStmtAttr(m_hHandle,
SQL_ATTR_ROW_ARRAY_SIZE, &iRowArraySize,
SQL_IS_UINTEGER);
	SQLSetStmtAttr(m_hHandle,
SQL_ATTR_ROW_STATUS_PTR, RowStatusArray,
SQL_IS_POINTER);
	SQLSetStmtAttr(m_hHandle,
SQL_ATTR_ROWS_FETCHED_PTR, &NumRowsFetched,
SQL_IS_POINTER);
	*/

	// For each column, bind the address in the buffer at the start
of the memory allocated
	// for that column's data and the address at the start of the
memory allocated for that
	// column's length/indicator buffer.
	for (i = 0; i < NumCols; i++)
	{
	SQLBindCol(m_hHandle, i + 1, CTypeArray[i],
(SQLPOINTER)((SQLCHAR *)DataPtr + OffsetArray[i]),
	ColLenArray[i], (SQLINTEGER *)((SQLCHAR *)DataPtr
+ OffsetArray[i] + ColLenArray[i]));
	HandleODBCError(rc, SQL_HANDLE_STMT, m_hHandle,
SMSQLBindCol);
	// Underline each column name
	memset(szColName, '-', DispLenArray[i]);
	*(szColName + DispLenArray[i]) = '\0';
	PrintData(szColName, SQL_C_CHAR, DispLenArray[i], 0,
m_pOutputFile);
	}
	m_pOutputFile->WriteLine(NULL);

#ifdef _DEBUG
	_CrtDbgReport(_CRT_WARN, NULL, 0, NULL,
"Executing SQLFetch.\n");
#endif

	while (!m_bAbort && (rc = SQLFetch(m_hHandle)) !=
SQL_NO_DATA)
	{
	HandleODBCError(rc, SQL_HANDLE_STMT, m_hHandle,
SMSQLFetch);
			/*
	for (i = 0; i < NumRowsFetched; i++)
	{
	if (RowStatusArray[i] == SQL_ROW_SUCCESS||
RowStatusArray[i] == SQL_ROW_SUCCESS_WITH_INFO)
	{
	*/
	for (i = 0; i < NumCols; i++)
			{
	// Retrieve and print each row. PrintData accepts a pointer to
the data, its C type,
	// and its byte length/indicator.
	if ( *((SQLINTEGER *)((SQLCHAR *)DataPtr +
OffsetArray[i] + ColLenArray[i])) == SQL_NULL_DATA)
		PrintData(S_NULL, SQL_C_CHAR, DispLenArray[i],
0, m_pOutputFile);
	else
		PrintData((LPVOID)((SQLCHAR *)DataPtr +
OffsetArray[i]), CTypeArray[i],
				DispLenArray[i], CScaleArray[i],
m_pOutputFile);
	}
	m_pOutputFile->WriteLine(NULL);
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
    }
    m_pOutputFile->WriteLine(NULL);
    /*
    free(RowStatusArray);
    */
        free(DataPtr);

    free(CTypeArray);
    free(CScaleArray);
    free(ColLenArray);
    free(DispLenArray);
    }

    // Write io statistics, if applicable
    LogODBCErrors(rc, SQL_HANDLE_STMT, m_hHandle,
SMSQLFetch);

#ifdef _DEBUG
        _CrtDbgReport(_CRT_WARN, NULL, 0, NULL,
"Executing SQLRowCount.\n");
#endif

        if (m_bAbort)
            break;

        // action (insert, update, delete) query
        rc = SQLRowCount(m_hHandle, &iRowCount);
        HandleODBCError(rc, SQL_HANDLE_STMT,
m_hHandle, SMSQLRowCount);

        if (!m_bNoCount && iRowCount != -1)
        {
            sprintf(szColName, "(%d row(s) affected)",
iRowCount);
            _bstr_t temp(szColName);
            m_pOutputFile->WriteLine((BSTR)temp);
            m_pOutputFile->WriteLine(NULL);
        }

#ifdef _DEBUG
        _CrtDbgReport(_CRT_WARN, NULL, 0, NULL,
"Executing SQLFreeStmt.\n");
#endif

    if (m_bAbort)
    break;
    SQLFreeStmt(m_hHandle, SQL_UNBIND);

#ifdef _DEBUG
        _CrtDbgReport(_CRT_WARN, NULL, 0, NULL,
"Executing SQLMoreResults.\n");
#endif
if (m_bAbort)
break;
// Process the next resultset. This function returns 'success with
//info' even if there is no other resultset and there are statistics
//messages to be printed. Hence the check for -1 rows before
//printing.
    rc=SQLMoreResults(m_hHandle);
    if (rc != SQL_NO_DATA)
    HandleODBCError(rc, SQL_HANDLE_STMT, m_hHandle,
SMSQLMoreResults);
  } while (rc != SQL_NO_DATA);
    return;
}
```

```
void CExecute::PrintData(void *vData, SQLSMALLINT CType,
SQLINTEGER IndPtr, SQLSMALLINT iScale, ISMLog
*pOutput)
{
// PrintData accepts a pointer to the data, its C type,
// and its byte length/indicator. It contains a switch statement that
casts and prints
// the data according to its type.

char    *s;
char    fmt[MAXBUFLEN];
int         j = 0;
SQLINTEGER  iColLen = IndPtr + 1;
assert(iColLen);
s = (LPSTR)malloc(iColLen + 1);
if (s)
{
    if (vData)
    {
        switch(CType)
        {
        case SQL_C_CHAR:
        case SQL_C_WCHAR:
        case SQL_C_TYPE_DATE:
        case SQL_C_TYPE_TIME:
        case SQL_C_TYPE_TIMESTAMP:
        case SQL_C_INTERVAL_YEAR:
        case SQL_C_INTERVAL_MONTH:
        case SQL_C_INTERVAL_YEAR_TO_MONTH:
        case SQL_C_INTERVAL_DAY:
        case SQL_C_INTERVAL_HOUR:
        case SQL_C_INTERVAL_MINUTE:
        case SQL_C_INTERVAL_SECOND:
        case SQL_C_INTERVAL_DAY_TO_HOUR:
        case SQL_C_INTERVAL_DAY_TO_MINUTE:
        case SQL_C_INTERVAL_DAY_TO_SECOND:
        case SQL_C_INTERVAL_HOUR_TO_MINUTE:
        case SQL_C_INTERVAL_HOUR_TO_SECOND:
        case SQL_C_INTERVAL_MINUTE_TO_SECOND:
        case SQL_C_BINARY:
            sprintf(fmt, "%%.%ds", iColLen);
            j = sprintf(s, fmt, (char *)vData);
            break;

        case SQL_C_SHORT:
            j = sprintf(s, "%d", *(short *)vData);
            break;

        case SQL_C_LONG:
            j = sprintf(s, "%ld", *(long *)vData);
            break;

        case SQL_C_UBIGINT:
            j = sprintf(s, "%I64d", *(__int64 *)vData);
            break;
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
            case SQL_C_FLOAT:
                j = sprintf(s, "%f", *(float *)vData);
                break;

            case SQL_C_DOUBLE:
                j = sprintf(s, "%f", *(double *)vData);
                break;

            case SQL_C_NUMERIC:
                sprintf(fmt, "%%.0%df", iScale);
                j = sprintf(s, fmt, *(double *)vData);
                break;

            default:
                j = sprintf(s, "%s", vData);
                break;
            }
        }

        // Strip off terminating null character and pad the string with
blanks
            if (iColLen - j > 0)
                memset(s + j, ' ', iColLen - j);

            *(s + iColLen) = '\0';

            // Write the field to the output file
            _bstr_t temp(s);
            pOutput->WriteField((BSTR)temp);
            free(s);
        }

        return;
}

SQLSMALLINT CExecute::GetDefaultCType(SQLINTEGER
SQLType)
{
        // GetDefaultCType returns the C type for the passed in SQL
datatype.

    switch(SQLType)
    {
    case SQL_CHAR:
        case SQL_VARCHAR:
        case SQL_LONGVARCHAR:
        case SQL_WCHAR:
        case SQL_WVARCHAR:
        case SQL_WLONGVARCHAR:
            return(SQL_C_CHAR);

    case SQL_TINYINT:
                return(SQL_C_CHAR);
    case SQL_SMALLINT:
                return(SQL_C_SHORT);
    case SQL_INTEGER:
                return(SQL_C_LONG);
    case SQL_BIGINT:
        return(SQL_C_UBIGINT);

    case SQL_REAL:
                return(SQL_C_FLOAT);
    case SQL_FLOAT:
    case SQL_DOUBLE:
//      case SQL_DECIMAL:
                return(SQL_C_DOUBLE);

        case SQL_DECIMAL:
                return(SQL_C_CHAR);

        case SQL_BIT:
                return(SQL_C_CHAR);

    case SQL_BINARY:
        case SQL_VARBINARY:
        case SQL_LONGVARBINARY:
                return(SQL_C_CHAR);
//              return(SQL_C_BINARY);

    case SQL_TYPE_DATE:
                return(SQL_C_CHAR);
//              return(SQL_C_TYPE_DATE);

    case SQL_TYPE_TIME:
                return(SQL_C_CHAR);
//              return(SQL_C_TYPE_TIME);

    case SQL_TYPE_TIMESTAMP:
                return(SQL_C_CHAR);
//              return(SQL_C_TYPE_TIMESTAMP);

        case SQL_NUMERIC:
            return(SQL_C_FLOAT);

    case SQL_INTERVAL_YEAR:
                return(SQL_C_CHAR);
//              return(SQL_C_INTERVAL_YEAR);

    case SQL_INTERVAL_MONTH:
                return(SQL_C_CHAR);
//              return(SQL_C_INTERVAL_MONTH);

    case SQL_INTERVAL_YEAR_TO_MONTH:
                return(SQL_C_CHAR);
//
        return(SQL_C_INTERVAL_YEAR_TO_MONTH);

    case SQL_INTERVAL_DAY:
                return(SQL_C_CHAR);
//              return(SQL_C_INTERVAL_DAY);
    case SQL_INTERVAL_HOUR:
                return(SQL_C_CHAR);
//              return(SQL_C_INTERVAL_HOUR);

    case SQL_INTERVAL_MINUTE:
                return(SQL_C_CHAR);
//              return(SQL_C_INTERVAL_MINUTE);

    case SQL_INTERVAL_SECOND:
                return(SQL_C_CHAR);
//              return(SQL_C_INTERVAL_SECOND);
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
   case SQL_INTERVAL_DAY_TO_HOUR:
                return(SQL_C_CHAR);
//              return(SQL_C_INTERVAL_DAY_TO_HOUR);

   case SQL_INTERVAL_DAY_TO_MINUTE:
                return(SQL_C_CHAR);
//              return(SQL_C_INTERVAL_DAY_TO_MINUTE);

   case SQL_INTERVAL_DAY_TO_SECOND:
                return(SQL_C_CHAR);
//              return(SQL_C_INTERVAL_DAY_TO_SECOND);

   case SQL_INTERVAL_HOUR_TO_MINUTE:
                return(SQL_C_CHAR);
//
      return(SQL_C_INTERVAL_HOUR_TO_MINUTE);

   case SQL_INTERVAL_HOUR_TO_SECOND:
                return(SQL_C_CHAR);
//
      return(SQL_C_INTERVAL_HOUR_TO_SECOND);

   case SQL_INTERVAL_MINUTE_TO_SECOND:
                return(SQL_C_CHAR);
//
      return(SQL_C_INTERVAL_MINUTE_TO_SECOND);

      default:
            assert(TRUE);
            return(SQL_C_CHAR);
            break;
   }
}

/*
   FUNCTION: LogODBCErrors(SQLRETURN rc, SWORD
fHandleType, SQLHANDLE handle)
   COMMENTS: Formats ODBC errors or warnings and logs
them. Also initializes the
                  completion status for the step to failure, if an
ODBC error has occurred.
*/

void CExecute::LogODBCErrors(SQLRETURN nResult,
SWORD fHandleType, SQLHANDLE handle, OdbcOperations
FailedOp)
{
      // Messages returned by the server (e.g. Print statements) will
be logged to the output file
      // ODBC warnings will be logged to the log file
      // All other ODBC errors will be logged to the error file.

UCHAR       szErrState[SQL_SQLSTATE_SIZE+1];// SQL
Error State string
UCHAR       szErrText[SQL_MAX_MESSAGE_LENGTH+1];
// SQL Error Text string
char
      szBuffer[SQL_SQLSTATE_SIZE+SQL_MAX_MESSAGE
_LENGTH+MAXBUFLEN+1] = "";
                                    // formatted Error text Buffer
SWORD           wErrMsgLen; // Error message length
SQLINTEGER      dwErrCode;  // Native Error code
SQLRETURN       nErrResult;         // Return Code from
SQLGetDiagRec
SWORD           sMsgNum = 1; // Error sequence number
_bstr_t         temp;
if (IsErrorReturn(nResult))
{
sprintf(szBuffer, "ODBC Operation: '%s' returned error code:
%d",
            g_szOdbcOps[FailedOp], nResult);
      temp = szBuffer;
      m_pErrorFile->WriteLine((BSTR) temp);
      m_StepStatus = gintFailed;
}

      if (handle == SQL_NULL_HSTMT)
            return;

#ifdef _DEBUG
      _CrtDbgReport(_CRT_WARN, NULL, 0, NULL,
"Executing SQLGetDiagRec.\n");
#endif
      // call SQLGetDiagRec function with proper ODBC handles,
repeatedly until
      // function returns SQL_NO_DATA.
      while (!m_bAbort && (nErrResult =
SQLGetDiagRec(fHandleType, handle, sMsgNum++,
            szErrState, &dwErrCode, szErrText,
SQL_MAX_MESSAGE_LENGTH-1, &wErrMsgLen))
            != SQL_NO_DATA)
      {
            if (!SQL_SUCCEEDED(nErrResult))
                  break;

            if (m_pOutputFile && IsServerMessage(dwErrCode,
szErrText))
      {
      wsprintf(szBuffer, SM_SQLMSG_FORMAT,
(LPSTR)szErrText);
      temp = szBuffer;
      m_pOutputFile->WriteLine((BSTR) temp);
            }
      else if (IsODBCWarning(szErrState) && dwErrCode !=
SM_SQL_ERR_CHANGED_DB &&
                  dwErrCode !=
SM_SQL_ERR_CHANGED_LANG)
            {
            // Suppress warnings - 'Changed database context to...'
and 'Changed language setting to...'
            wsprintf(szBuffer, SM_SQLMSG_FORMAT,
ParseOdbcMsgPrefixes((LPCSTR)szErrText));
            temp = szBuffer;
            m_pOutputFile->WriteLine((BSTR) temp);
            }
            else if (m_pErrorFile &&
!IsODBCWarning(szErrState))
            {
            wsprintf(szBuffer, SM_SQLERR_FORMAT,
(LPSTR)szErrState, dwErrCode, (LPSTR)szErrText);
            temp = szBuffer;
            m_pErrorFile->WriteLine((BSTR) temp);
            }
      }
}
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
/*
   FUNCTION: LogErrors(SQLRETURN rc, SWORD
fHandleType, SQLHANDLE handle)
   COMMENTS: Writes the error message to the error log
*/
void CExecError::LogErrors(CExecute *p)
{
      _bstr_t temp(m_szExecErrorDesc[m_iErrCode]);
      if (p->m_pErrorFile)
            p->m_pErrorFile->WriteLine((BSTR)temp);
return;
}
/*
   FUNCTION: ODBCCleanup(HDBC *hdbc, HSTMT *hstmt)
   COMMENTS: Cleanup of all ODBC structures
*/

void CExecute::ODBCCleanup(HDBC *hdbc, HSTMT *hstmt)
{
SQLRETURN     lReturn;

#ifdef _DEBUG
      _CrtDbgReport(_CRT_WARN, NULL, 0, NULL,
"Executing ODBCCleanup.\n");
#endif

      if (*hstmt != SQL_NULL_HSTMT)
      {
#ifdef _DEBUG
            _CrtDbgReport(_CRT_WARN, NULL, 0, NULL,
"Executing SQLCloseCursor.\n");
#endif
            SQLCloseCursor(hstmt);
#ifdef _DEBUG
            _CrtDbgReport(_CRT_WARN, NULL, 0, NULL,
"Executing SQLFreeHandle for hstmt.\n");
#endif
            SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
            *hstmt = SQL_NULL_HSTMT;
      }
      // Cleanup connection if it is a dynamic connection
      if (IsDynamicConnection())
      {
            if (*hdbc != SQL_NULL_HDBC)
            {
#ifdef _DEBUG
      _CrtDbgReport(_CRT_WARN, NULL, 0, NULL,
"Executing SQLDisconnect.\n");
#endif
      lReturn = SQLDisconnect(*hdbc);
#ifdef _DEBUG
                  _CrtDbgReport(_CRT_WARN, NULL, 0, NULL,
"Executing SQLFreeHandle for hdbc.\n");
#endif
                  SQLFreeHandle(SQL_HANDLE_DBC, hdbc);
                  *hdbc = SQL_NULL_HDBC;
            }
      }
      else
            ResetConnectionUsage();

      return;
}
```

```
// Wrapper function that raises an error if a Windows Api fails
STDMETHODIMP CExecute::RaiseSystemError(void)
{
      char s[MAXBUFLEN];

      GetSystemError(s);
      return Error(s, 0, NULL, GUID_NULL);
}
// Wrapper function that logs the error raised by an Api function
to the passed in file
void CExecute::LogSystemError(ISMLog *pFile)
{
      if (pFile)
      {
            char s[MAXBUFLEN];
            GetSystemError(s);

            _bstr_t temp(s);
            pFile->WriteLine((BSTR)temp);
      }
}

// Populates the passed in string with the last Windows Api error
that occurred
void CExecute::GetSystemError(LPSTR s)
{
      long c;
      DWORD e;

      e = GetLastError();

      c = sprintf(s, "Error code: %ld. ", e);
      c =
FormatMessage(FORMAT_MESSAGE_FROM_SYSTEM |
FORMAT_MESSAGE_IGNORE_INSERTS,
            NULL, e, 0, s + c, MAXBUFLEN - c, NULL);

      return;
}

STDMETHODIMP CExecute::get_StepStatus(InstanceStatus
*pVal)
{
      *pVal = m_StepStatus;
      return S_OK;
}

STDMETHODIMP CExecute::WriteError(BSTR szMsg)
{
      if (m_pErrorFile)
            return(m_pErrorFile->WriteLine(szMsg));

      return S_OK;
}
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
//  FILE:      Execute.h
//             Microsoft TPC-H Kit Ver. 1.00
//             Copyright Microsoft, 1999
//             All Rights Reserved
//
//
//  PURPOSE:    Declaration of the CExecute
//  Contact:    Reshma Tharamal (reshmat@microsoft.com)
//
// Execute.h : Declaration of the CExecute

#ifndef __EXECUTE_H_
#define __EXECUTE_H_

#include <atlwin.h>
#include <comdef.h>
#include <stdio.h>
#include "resource.h"       // main symbols
#include "ExecuteDllCP.h"
#include "..\LogWriter\LogWriter.h"
#include "..\LogWriter\SMLog.h"
#include "..\common\SMTime\SMTime.h"
#include "..\common\SMTime\SMTimer.h"

// ODBC-specific includes
#define DBNTWIN32
#include <sqltypes.h>
#include <sql.h>
#include <sqlext.h>

/////////////////////////////////////////////////////////////////////////////
// CExecute

#define WM_TASK_START  (WM_USER + 101)
#define WM_TASK_FINISH  (WM_USER + 102)

#define SM_SQLERR_FORMAT   "SQL Error State:%s, Native
Error Code: %ld\r\nODBC Error: %s"
                                    // format for ODBC error
messages
#define SM_SQLWARN_FORMA SM_SQLERR_FORMAT //
format for ODBC warnings
#define SM_SQLMSG_FORMAT  "%s"                  // format
for messages from the server

#define SM_SQL_STATE_WARNING "01000"
#define SM_MSG_SERVER                "[Microsoft][ODBC
SQL Server Driver][SQL Server]"
#define SM_SQL_ERR_CHANGED_DB     5701
#define SM_SQL_ERR_CHANGED_LANG       5703
#define SM_STEPMASTER_ERROR "StepMaster Error: "
#define CMD_SEPARATOR   "\nGO"
#define INV_ARRAY_INDEX     -1 // invalid index into an
array
#define MAXBUFLEN           256        // display buffer size
#define MAXLOGCMDLEN        256        // maximum
//characters in command that will be  printed to log
#define MAXLOGCMDBUF        512        // maximum
//characters in command that will be printed to log
#define MAX_DATA_LEN        4000        // maximum buffer
//size for variable-length data types // viz. character and binary
//fields
#define FILE_ACCESS_READ    "r" // Open file for read access
#define S_NULL "NULL"
// Define a macro to increase the size of a buffer so it is a
multiple of teh alignment size.
// Thus, if a buffer starts on an alignment boundary, it will end
just before the next
// alignment boundary. Here, an alignment size of 4 is used
because this is the size of the
// largest data type used in the application's buffer - the size of an
SDWORD and of the largest
// default C data type are both 4. If a larger data type (such as
__int64) is used, it will be
// necessary to align for that size.
#define ALIGNSIZE 4
#define ALIGNBUF(Length) ((Length) % ALIGNSIZE) ? \
   ((Length) + ALIGNSIZE - ((Length) % ALIGNSIZE)) :
(Length)

#define MAX_BUFFER_SIZE       64000

typedef enum OdbcOperations
{
      SMSQLAllocHandle,
      SMSQLDriverConnect,
      SMSQLExecDirect,
      SMSQLSetStmtAttr,
      SMSQLCancel,
      SMSQLNumResultCols,
      SMSQLDescribeCol,
      SMSQLColAttribute,
      SMSQLFetch,
      SMSQLGetData,
      SMSQLRowCount,
      SMSQLMoreResults,
      SMSQLBindCol,
};

class CODBCError
{
public:
      CODBCError(SQLRETURN nResult, SWORD
fHandleType, SQLHANDLE    handle, OdbcOperations
FailedOp)
      {
      m_fHandleType   = fHandleType;
      m_handle        = handle;
      m_FailedOp      = FailedOp;
      m_nResult       = nResult;
      };

private:
      SWORD               m_fHandleType;
      SQLHANDLE           m_handle;
      OdbcOperations  m_FailedOp;
      SQLRETURN           m_nResult;

private:
inline BOOL IsServerMessage(SQLINTEGER lNativeError,
UCHAR *szErr){
      return( (strstr((LPCTSTR)szErr, SM_MSG_SERVER) !=
NULL) ? (lNativeError == 0) : FALSE); }
inline BOOL IsODBCWarning(UCHAR *szSqlState){
      return(strcmp((LPCSTR)szSqlState,
SM_SQL_STATE_WARNING) == 0);}
inline LPCSTR ParseOdbcMsgPrefixes(LPCSTR szMsg){ char
*pDest;
      return( (pDest = strstr(szMsg, SM_MSG_SERVER)) ==
NULL ? szMsg : pDest + strlen(SM_MSG_SERVER));}
};
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
class ATL_NO_VTABLE CExecute :
	public CWindowImpl<CExecute>,
	public CComObjectRootEx<CComSingleThreadModel>,
	public CComCoClass<CExecute, &CLSID_Execute>,
	public IConnectionPointContainerImpl<CExecute>,
	public ISupportErrorInfo,
	public IDispatchImpl<IExecute, &IID_IExecute,
&LIBID_EXECUTEDLLLib>,
	public CProxy_IExecuteEvents< CExecute >
{
public:
	CExecute()
	{
		m_pErrorFile = NULL;
		//m_pLogFile = NULL;
		m_pOutputFile = NULL;

		// Initialize the elapsed time for the step
		m_tElapsedTime = 0;

		// Initialize the run status for the step
		m_StepStatus = gintPending;

		m_hHandle = SQL_NULL_HSTMT;
		m_bAbort = FALSE;

		m_iConnectionIndex = INV_ARRAY_INDEX;
	}

	~CExecute()
	{
	}

	friend class CExecError;

public:
	DECLARE_WND_CLASS("Execute")

		BEGIN_MSG_MAP(CExecute)
		MESSAGE_HANDLER(WM_TASK_FINISH,
OnTaskFinished)
		MESSAGE_HANDLER(WM_TASK_START,
OnTaskStarted)
		END_MSG_MAP()
public:

	LRESULT OnTaskStarted(UINT uMsg, WPARAM
wParam,
		LPARAM lParam, BOOL& bHandled)
	{
		CURRENCY	CStartTime =
Get64BitTime(&m_tStartTime);

		Fire_Start(CStartTime);
		return 0;
	}
```

```
LRESULT OnTaskFinished(UINT uMsg, WPARAM wParam,
	LPARAM lParam, BOOL& bHandled)
{
	CURRENCY	CEndTime =
Get64BitTime(&m_tEndTime);
		Fire_Complete(CEndTime, (long)m_tElapsedTime);
	return 0;
}

HRESULT FinalConstruct()
{
	HRESULT		hr;
	RECT		rect;
		rect.left=0;
	rect.right=100;
	rect.top=0;
	rect.bottom=100;
		HWND hwnd = Create( NULL, rect,
"ExecuteWindow", WS_POPUP);

	if (!hwnd)
		return HRESULT_FROM_WIN32(GetLastError());
		hr = CoCreateInstance(CLSID_SMLog, NULL,
CLSCTX_INPROC,
		IID_ISMLog, (void **)&m_pErrorFile);
	if FAILED(hr)
		return(hr);
	m_pErrorFile->put_Append(TRUE);
		//hr = CoCreateInstance(CLSID_SMLog, NULL,
CLSCTX_INPROC,
	//	IID_ISMLog, (void **)&m_pLogFile);
	//if FAILED(hr)
	//	return(hr);
		hr = CoCreateInstance(CLSID_SMLog, NULL,
CLSCTX_INPROC,
		IID_ISMLog, (void **)&m_pOutputFile);
	if FAILED(hr)
		return(hr);
	m_pOutputFile->put_Append(TRUE);
		hr = CoCreateInstance(CLSID_SMTimer, NULL,
CLSCTX_INPROC,
		IID_ISMTimer, (void **)&m_ExecTime);
	if FAILED(hr)
		return(hr);

	return S_OK;
}
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
void FinalRelease()
{
	if (m_hWnd != NULL)
		DestroyWindow();
		// Close the log and error files
	if (m_pErrorFile)
		m_pErrorFile->Release();
	m_pErrorFile = NULL;

		//if (m_pLogFile)
		//	m_pLogFile->Release();
		//m_pLogFile = NULL;

		if (m_ExecTime)
			m_ExecTime->Release();
		m_ExecTime = NULL;
	}

DECLARE_REGISTRY_RESOURCEID(IDR_EXECUTE)

DECLARE_PROTECT_FINAL_CONSTRUCT()

BEGIN_COM_MAP(CExecute)
	COM_INTERFACE_ENTRY(IExecute)
	COM_INTERFACE_ENTRY(ISupportErrorInfo)
	COM_INTERFACE_ENTRY(IDispatch)
	COM_INTERFACE_ENTRY(IConnectionPointContainer)
	COM_INTERFACE_ENTRY_IMPL(IConnectionPointCont
ainer)
END_COM_MAP()
BEGIN_CONNECTION_POINT_MAP(CExecute)
CONNECTION_POINT_ENTRY(DIID__IExecuteEvents)
END_CONNECTION_POINT_MAP()

// ISupportsErrorInfo
STDMETHOD(InterfaceSupportsErrorInfo)(REFIID riid);

// IExecute
public:
	STDMETHOD(put_ErrorFile)(/*[in]*/ BSTR newVal);
	STDMETHOD(put_OutputFile)(/*[in]*/ BSTR newVal);
	STDMETHOD(WriteError)(BSTR szMsg);
	STDMETHOD(Abort)();
	STDMETHOD(get_StepStatus)(/*[out, retval]*/
InstanceStatus *pVal);
	STDMETHOD(DoExecute)(/*[in]*/ BSTR szCommand,
/*[in]*/ BSTR szExecutionDtls,
		/*[in]*/ ExecutionType ExecMethod,
		/*[in]*/ BOOL bNoCount, /*[in]*/ BOOL bNoExecute,
/*[in]*/ BOOL bParseOnly,
		/*[in]*/ BOOL bQuotedIds, /*[in]*/ BOOL bAnsiNulls,
/*[in]*/ BOOL bShowQP,
		/*[in]*/ BOOL bStatsTime, /*[in]*/ BOOL bStatsIO,
/*[in]*/ long lRowCount,
		/*[in]*/ long lQueryTmout, /*[in]*/ BSTR
szConnection);
	TC_TIME		ExecuteShell();
	TC_TIME		ExecuteODBC();
	STDMETHODIMP	AbortShell();
	STDMETHODIMP	AbortODBC();
	_bstr_t		m_szCommand;
	_bstr_t		m_szExecDtls;
	_bstr_t		m_szConnection;
	DWORD		m_lMode;
	//DATE		m_CurTime;
	SYSTEMTIME		m_tStartTime;
	SYSTEMTIME		m_tEndTime;
	TC_TIME		m_tElapsedTime;
	ISMLog		*m_pErrorFile;
	//ILog		*m_pLogFile;
	ISMLog		*m_pOutputFile;
	ISMTimer		*m_ExecTime;
	ExecutionType	m_ExecMthd;
	InstanceStatus	m_StepStatus;
	HANDLE		m_hHandle;	// Process handle for
shell commands and
						// Statement handle for
ODBC commands
	LPSTR		m_szCmd;

private:
	LPSTR		NextCmdInBatch(LPSTR szBatch);
	void		ProcessResultsets();
	SQLSMALLINT	GetDefaultCType(SQLINTEGER
SQLType);
	void		PrintData(void *vData, SQLSMALLINT
CType, SQLINTEGER IndPtr,
		SQLSMALLINT iScale, ISMLog *pOutput);
	void		LogODBCErrors(SQLRETURN nResult,
SWORD fHandleType, SQLHANDLE handle,
		OdbcOperations FailedOp);
	void		ODBCCleanup(HDBC *hdbc, HSTMT
*hstmt);
	STDMETHODIMP	RaiseSystemError(void);
	void		LogSystemError(ISMLog *pFile);
	void		GetSystemError(LPSTR s);
	void		SetConnectionOption(LPSTR szConn, HDBC
*pHdbc);
	void		ResetConnectionProperties(HDBC *p_hdbc);

	void		InitializeConnection(HDBC *phdbc, BOOL
*pbDoConnect);
	void		ReConnectDeadConnection(HDBC *phdbc);

	void		ResetConnectionUsage();

	int	m_iConnectionIndex;
	BOOL	m_bNoCount, m_bNoExecute, m_bParseOnly,
m_bQuotedIds, m_bAnsiNulls, \
		m_bShowQP, m_bStatsTime, m_bStatsIO;
	long		m_lRowCount;
	SQLUINTEGER	m_lQueryTmout;
	_bstr_t		m_ErrorFile, m_OutputFile;
	BOOL		m_bAbort;
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
private:
inline BOOL IsServerMessage(SQLINTEGER lNativeError,
UCHAR *szErr){
	return( (strstr((LPCTSTR)szErr, SM_MSG_SERVER) !=
NULL) ? (lNativeError == 0) : FALSE); }
inline BOOL IsODBCWarning(UCHAR *szSqlState){
	return(strcmp((LPCSTR)szSqlState,
SM_SQL_STATE_WARNING) == 0);}
inline BOOL IsErrorReturn(SQLRETURN iRetCode){
	return( (!SQL_SUCCEEDED(iRetCode)) && (iRetCode !=
SQL_NO_DATA) );}
inline LPCSTR ParseOdbcMsgPrefixes(LPCSTR szMsg){ char
*pDest;
	return( (pDest = strstr(szMsg, SM_MSG_SERVER)) ==
NULL ? szMsg : pDest + strlen(SM_MSG_SERVER));}
inline BOOL IsDynamicConnection(){
return(!strcmp((LPSTR)m_szConnection, ""));}
inline void HandleODBCError(SQLRETURN rc, SWORD
fHandleType, SQLHANDLE handle, OdbcOperations OdbcOp)
{
	if (rc != SQL_SUCCESS)
	{
		LogODBCErrors(rc, fHandleType, handle, OdbcOp);
		if (IsErrorReturn(rc))
			throw new CODBCError(rc, fHandleType, handle,
OdbcOp);
	}
	return;
}

};

class CExecError
{
public:
	typedef enum ExecErrorCodes
	{
		SM_ERR_CONN_IN_USE,
	};

	CExecError(int iError)
	{
		m_iErrCode  = iError;
	};

	void		LogErrors(CExecute *p);

private:
	int		m_iErrCode;
	static char	*m_szExecErrorDesc[];
};

void		ExecutionThread(LPVOID lpParameter);

#ifdef _TPCH_AUDIT
	void		WriteFileToTpchLog(LPSTR szFile, LPSTR
szFmt);
	void		WriteToTpchLog(char *szMsg);
#endif
```

```
#endif //__EXECUTE_H_
//  FILE:	ExecuteDll.cpp
//		Microsoft TPC-H Kit Ver. 1.00
//		Copyright Microsoft, 1999
//		All Rights Reserved
//
//
//  PURPOSE:	Implementation of DLL Exports.
//  Contact:	Reshma Tharamal (reshmat@microsoft.com)
//


// Note: Proxy/Stub Information
//	To build a separate proxy/stub DLL,
//	run nmake -f ExecuteDllps.mk in the project directory.

#include "stdafx.h"
#include "resource.h"
#include <initguid.h>

#include "..\LogWriter\LogWriter.h"
#include "..\LogWriter\LogWriter_i.c"

#include "..\common\SMTime\SMTime.h"
#include "..\common\SMTime\SMTime_i.c"

#include "ExecuteDll.h"
#include "SMExecute.h"

#include "ExecuteDll_i.c"
#include "Execute.h"

CComModule _Module;

BEGIN_OBJECT_MAP(ObjectMap)
OBJECT_ENTRY(CLSID_Execute, CExecute)
END_OBJECT_MAP()

SQLHENV henv = NULL;				// ODBC
environment handle
static char szCaption[] = "StepMaster";		// Message box
caption

CRITICAL_SECTION hConnections;			// Critical
section to serialize access
						// to available connections
SM_Connection_Info *p_Connections = NULL;	// Pointer to
open connections
int		iConnectionCount = 0;// Number of open
connections

#ifdef _TPCH_AUDIT
	FILE *pfLogFile = NULL;			// Log file
containing timestamps
	CRITICAL_SECTION hLogFileWrite;		// Critical
section to serialize writes to log
	static char szFileOpenModeAppend[] = "a+";	// Log file
open mode

	static char szEnvVarLogFile[] = "TPCH_LOG_FILE";//
Environment variable - initialized to
						// log file name if timing
information
						// is to be logged
#endif
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
void ShowODBCErrors(SWORD fHandleType, SQLHANDLE
handle);
void CloseOpenConnections();

////////////////////////////////////////////////////////////////////////////
// DLL Entry Point

extern "C"
BOOL WINAPI DllMain(HINSTANCE hInstance, DWORD
dwReason, LPVOID /*lpReserved*/)
{

   if (dwReason == DLL_PROCESS_ATTACH)
   {
      _Module.Init(ObjectMap, hInstance,
&LIBID_EXECUTEDLLLib);
      DisableThreadLibraryCalls(hInstance);

#ifdef _TPCH_AUDIT
            char szMsg[MAXBUFLEN];
            LPSTR szLogFileName = getenv(szEnvVarLogFile);

            if (szLogFileName == NULL)
            {
                  sprintf(szMsg, "The environment variable '%s' does
not exist. "
                        "Step timing information will not be written to
a log.", szEnvVarLogFile);
                  MessageBox(NULL, szMsg, szCaption, MB_OK);
            }
            else
            {
if ( (pfLogFile = fopen(szLogFileName,
szFileOpenModeAppend)) == NULL )
{
      sprintf(szMsg, "The file '%s' does not exist. "
"Step timing information will not be written to log.",
szLogFileName);
      MessageBox(NULL, szMsg, szCaption, MB_OK);
      }
      else
      InitializeCriticalSection(&hLogFileWrite);
      }
#endif
      InitializeCriticalSection(&hConnections);
      p_Connections = NULL;
      iConnectionCount = 0;
      if (!SQL_SUCCEEDED(SQLSetEnvAttr(NULL,
SQL_ATTR_CONNECTION_POOLING,
                  (SQLPOINTER)SQL_CP_ONE_PER_HENV , 0)))
                  ShowODBCErrors(SQL_HANDLE_ENV, henv);


            if
(!SQL_SUCCEEDED(SQLAllocHandle(SQL_HANDLE_ENV,
SQL_NULL_HANDLE, &henv)))
            {
                  ShowODBCErrors(SQL_HANDLE_ENV, henv);
                  return FALSE;
            }
      if (!SQL_SUCCEEDED(SQLSetEnvAttr(henv,
SQL_ATTR_ODBC_VERSION, (LPVOID)SQL_OV_ODBC3,
0)))
      ShowODBCErrors(SQL_HANDLE_ENV, henv);
SQLUINTEGER CpMatch;
if (!SQL_SUCCEEDED(SQLGetEnvAttr(henv,
SQL_ATTR_CP_MATCH, &CpMatch, 0, NULL)))
      ShowODBCErrors(SQL_HANDLE_ENV, henv);
      if (!SQL_SUCCEEDED(SQLSetEnvAttr(henv,
SQL_ATTR_CP_MATCH,
            (SQLPOINTER)SQL_CP_STRICT_MATCH,
SQL_IS_INTEGER)))
      ShowODBCErrors(SQL_HANDLE_ENV, henv);

   }
   else if (dwReason == DLL_PROCESS_DETACH)
      {
#ifdef _TPCH_AUDIT
            if (pfLogFile != NULL)
            {
            fclose(pfLogFile);

            DeleteCriticalSection(&hLogFileWrite);
            }
#endif
      CloseOpenConnections();
            if (henv != NULL)
            SQLFreeEnv(henv);
            DeleteCriticalSection(&hConnections);
      _Module.Term();
      }
   return TRUE;    // ok
}
void ShowODBCErrors(SWORD fHandleType, SQLHANDLE
handle)
{
      UCHAR       szErrState[SQL_SQLSTATE_SIZE+1];// SQL
Error State string
      UCHAR
      szErrText[SQL_MAX_MESSAGE_LENGTH+1];    // SQL
Error Text string
      char
      szBuffer[SQL_SQLSTATE_SIZE+SQL_MAX_MESSAGE
_LENGTH+MAXBUFLEN+1] = "";
                  // formatted Error text Buffer
      SWORD           wErrMsgLen;  // Error message length
      SQLINTEGER      dwErrCode;                  // Native Error
code
      SQLRETURN       nErrResult;                 // Return Code
from SQLGetDiagRec
SWORD       sMsgNum = 1; // Error sequence number
      // call SQLGetDiagRec function with proper ODBC handles,
repeatedly until
      // function returns SQL_NO_DATA.
      while ((nErrResult = SQLGetDiagRec(fHandleType, handle,
sMsgNum++,
            szErrState, &dwErrCode, szErrText,
SQL_MAX_MESSAGE_LENGTH-1, &wErrMsgLen))
            != SQL_NO_DATA)
      {
            if (!SQL_SUCCEEDED(nErrResult))
            break;
            wsprintf(szBuffer, SM_SQLERR_FORMAT,
(LPSTR)szErrState, dwErrCode, (LPSTR)szErrText);
      MessageBox(NULL, szBuffer, szCaption, MB_OK);
      }
}
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
void CloseOpenConnections()
{
	// Closes all open connections

	if (p_Connections)
	{
		for (int iConnIndex = iConnectionCount - 1;
iConnIndex >= 0; iConnIndex--)
		{
			if ((p_Connections + iConnIndex)->hdbc !=
SQL_NULL_HDBC)
			{
#ifdef _DEBUG
			_CrtDbgReport(_CRT_WARN, NULL, 0, NULL,
"Executing SQLDisconnect.\n");
#endif
			SQLDisconnect((p_Connections + iConnIndex)-
>hdbc);
#ifdef _DEBUG
			_CrtDbgReport(_CRT_WARN, NULL, 0, NULL,
"Executing SQLFreeHandle for hdbc.\n");
#endif
			SQLFreeHandle(SQL_HANDLE_DBC,
(p_Connections + iConnIndex)->hdbc );
			(p_Connections + iConnIndex)->hdbc =
SQL_NULL_HDBC;
			}

		}

		free(p_Connections);
	}
	p_Connections = NULL;

	return;
}

///////////////////////////////////////////////////////////////////////////////
// Used to determine whether the DLL can be unloaded by OLE

STDAPI DllCanUnloadNow(void)
{
   return (_Module.GetLockCount()==0) ? S_OK : S_FALSE;
}

///////////////////////////////////////////////////////////////////////////////
// Returns a class factory to create an object of the requested type

STDAPI DllGetClassObject(REFCLSID rclsid, REFIID riid,
LPVOID* ppv)
{
   return _Module.GetClassObject(rclsid, riid, ppv);
}

///////////////////////////////////////////////////////////////////////////////
// DllRegisterServer - Adds entries to the system registry

STDAPI DllRegisterServer(void)
{
   // registers object, typelib and all interfaces in typelib
   return _Module.RegisterServer(TRUE);
}
```

```
///////////////////////////////////////////////////////////////////////////////
// DllUnregisterServer - Removes entries from the system
registry

STDAPI DllUnregisterServer(void)
{
   return _Module.UnregisterServer(TRUE);
}
/* this ALWAYS GENERATED file contains the definitions for
the interfaces */


/* File created by MIDL compiler version 5.01.0164 */
/* at Mon Jun 09 19:33:03 2003
 */
/* Compiler settings for
C:\charles\Stepmaster\ExecuteDll\ExecuteDll.idl:
   Oicf (OptLev=i2), W1, Zp8, env=Win32, ms_ext, c_ext
   error checks: allocation ref bounds_check enum stub_data
*/
//@@MIDL_FILE_HEADING(  )


/* verify that the <rpcndr.h> version is high enough to compile
this file*/
#ifndef __REQUIRED_RPCNDR_H_VERSION__
#define __REQUIRED_RPCNDR_H_VERSION__ 440
#endif

#include "rpc.h"
#include "rpcndr.h"

#ifndef __ExecuteDll_h__
#define __ExecuteDll_h__

#ifdef __cplusplus
extern "C"{
#endif

/* Forward Declarations */

#ifndef ___IExecuteEvents_FWD_DEFINED__
#define ___IExecuteEvents_FWD_DEFINED__
typedef interface _IExecuteEvents _IExecuteEvents;
#endif	/* ___IExecuteEvents_FWD_DEFINED__ */


#ifndef __IExecute_FWD_DEFINED__
#define __IExecute_FWD_DEFINED__
typedef interface IExecute IExecute;
#endif	/* __IExecute_FWD_DEFINED__ */


#ifndef __Execute_FWD_DEFINED__
#define __Execute_FWD_DEFINED__

#ifdef __cplusplus
typedef class Execute Execute;
#else
typedef struct Execute Execute;
#endif /* __cplusplus */

#endif	/* __Execute_FWD_DEFINED__ */
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
/* header files for imported files */
#include "oaidl.h"
#include "ocidl.h"

void __RPC_FAR * __RPC_USER MIDL_user_allocate(size_t);
void __RPC_USER MIDL_user_free( void __RPC_FAR * );

/* interface __MIDL_itf_ExecuteDll_0000 */
/* [local] */

typedef /* [helpstring][uuid] */
enum ExecutionType
   { execODBC   = 0x1,
     execShell      = 0x2
   } ExecutionType;

typedef /* [helpstring][uuid] */
enum InstanceStatus
   { gintDisabled = 0x1,
     gintPending  = 0x2,
     gintRunning  = 0x3,
     gintComplete       = 0x4,
     gintFailed     = 0x5,
     gintAborted  = 0x6
   } InstanceStatus;



extern RPC_IF_HANDLE
__MIDL_itf_ExecuteDll_0000_v0_0_c_ifspec;
extern RPC_IF_HANDLE
__MIDL_itf_ExecuteDll_0000_v0_0_s_ifspec;


#ifndef __EXECUTEDLLLib_LIBRARY_DEFINED__
#define __EXECUTEDLLLib_LIBRARY_DEFINED__

/* library EXECUTEDLLLib */
/* [helpstring][version][uuid] */


EXTERN_C const IID LIBID_EXECUTEDLLLib;

#ifndef ___IExecuteEvents_DISPINTERFACE_DEFINED__
#define ___IExecuteEvents_DISPINTERFACE_DEFINED__

/* dispinterface _IExecuteEvents */
/* [helpstring][uuid] */


EXTERN_C const IID DIID__IExecuteEvents;


#if defined(__cplusplus) && !defined(CINTERFACE)

   MIDL_INTERFACE("551AC532-AB1C-11D2-BC0C-
00A0C90D2CA5")
   _IExecuteEvents : public IDispatch
   {
   };
```

```
#else      /* C style interface */

   typedef struct _IExecuteEventsVtbl
   {
      BEGIN_INTERFACE

      HRESULT ( STDMETHODCALLTYPE __RPC_FAR
*QueryInterface )(
         _IExecuteEvents __RPC_FAR * This,
         /* [in] */ REFIID riid,
         /* [iid_is][out] */ void __RPC_FAR *__RPC_FAR
*ppvObject);

      ULONG ( STDMETHODCALLTYPE __RPC_FAR
*AddRef )(
         _IExecuteEvents __RPC_FAR * This);

      ULONG ( STDMETHODCALLTYPE __RPC_FAR
*Release )(
         _IExecuteEvents __RPC_FAR * This);

      HRESULT ( STDMETHODCALLTYPE __RPC_FAR
*GetTypeInfoCount )(
         _IExecuteEvents __RPC_FAR * This,
         /* [out] */ UINT __RPC_FAR *pctinfo);

      HRESULT ( STDMETHODCALLTYPE __RPC_FAR
*GetTypeInfo )(
         _IExecuteEvents __RPC_FAR * This,
         /* [in] */ UINT iTInfo,
         /* [in] */ LCID lcid,
         /* [out] */ ITypeInfo __RPC_FAR *__RPC_FAR
*ppTInfo);

      HRESULT ( STDMETHODCALLTYPE __RPC_FAR
*GetIDsOfNames )(
         _IExecuteEvents __RPC_FAR * This,
         /* [in] */ REFIID riid,
         /* [size_is][in] */ LPOLESTR __RPC_FAR *rgszNames,
         /* [in] */ UINT cNames,
         /* [in] */ LCID lcid,
         /* [size_is][out] */ DISPID __RPC_FAR *rgDispId);

      /* [local] */ HRESULT ( STDMETHODCALLTYPE
__RPC_FAR *Invoke )(
         _IExecuteEvents __RPC_FAR * This,
         /* [in] */ DISPID dispIdMember,
         /* [in] */ REFIID riid,
         /* [in] */ LCID lcid,
         /* [in] */ WORD wFlags,
         /* [out][in] */ DISPPARAMS __RPC_FAR
*pDispParams,
         /* [out] */ VARIANT __RPC_FAR *pVarResult,
         /* [out] */ EXCEPINFO __RPC_FAR *pExcepInfo,
         /* [out] */ UINT __RPC_FAR *puArgErr);

      END_INTERFACE
   } _IExecuteEventsVtbl;

   interface _IExecuteEvents
   {
      CONST_VTBL struct _IExecuteEventsVtbl __RPC_FAR
*lpVtbl;
   };
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
#ifdef COBJMACROS

#define _IExecuteEvents_QueryInterface(This,riid,ppvObject)\
   (This)->lpVtbl -> QueryInterface(This,riid,ppvObject)

#define _IExecuteEvents_AddRef(This)\
   (This)->lpVtbl -> AddRef(This)

#define _IExecuteEvents_Release(This)\
   (This)->lpVtbl -> Release(This)


#define _IExecuteEvents_GetTypeInfoCount(This,pctinfo)      \
   (This)->lpVtbl -> GetTypeInfoCount(This,pctinfo)

#define _IExecuteEvents_GetTypeInfo(This,iTInfo,lcid,ppTInfo)
      \
   (This)->lpVtbl -> GetTypeInfo(This,iTInfo,lcid,ppTInfo)

#define
_IExecuteEvents_GetIDsOfNames(This,riid,rgszNames,cNames,
lcid,rgDispId)      \
   (This)->lpVtbl ->
GetIDsOfNames(This,riid,rgszNames,cNames,lcid,rgDispId)

#define
_IExecuteEvents_Invoke(This,dispIdMember,riid,lcid,wFlags,pD
ispParams,pVarResult,pExcepInfo,puArgErr)      \
   (This)->lpVtbl ->
Invoke(This,dispIdMember,riid,lcid,wFlags,pDispParams,pVarR
esult,pExcepInfo,puArgErr)

#endif /* COBJMACROS */
#endif    /* C style interface */
#endif    /*
___IExecuteEvents_DISPINTERFACE_DEFINED__ */

#ifndef __IExecute_INTERFACE_DEFINED__
#define __IExecute_INTERFACE_DEFINED__

/* interface IExecute */
/* [unique][helpstring][dual][uuid][object] */
EXTERN_C const IID IID_IExecute;
#if defined(__cplusplus) && !defined(CINTERFACE)

   MIDL_INTERFACE("551AC531-AB1C-11D2-BC0C-
00A0C90D2CA5")
   IExecute : public IDispatch
   {
   public:
      virtual /* [helpstring][id] */ HRESULT
STDMETHODCALLTYPE DoExecute(
         /* [in] */ BSTR szCommand,
         /* [in] */ BSTR szExecutionDtls,
         /* [in] */ ExecutionType ExecMethod,
         /* [in] */ BOOL bNoCount,
         /* [in] */ BOOL bNoExecute,
         /* [in] */ BOOL bParseOnly,
         /* [in] */ BOOL bQuotedIds,
         /* [in] */ BOOL bAnsiNulls,
         /* [in] */ BOOL bShowQP,
         /* [in] */ BOOL bStatsTime,
         /* [in] */ BOOL bStatsIO,
         /* [in] */ long lRowCount,
         /* [in] */ long lQueryTmout,
         /* [in] */ BSTR szConnection) = 0;

      virtual /* [helpstring][id][propget] */ HRESULT
STDMETHODCALLTYPE get_StepStatus(
         /* [retval][out] */ InstanceStatus __RPC_FAR *pVal) =
0;

      virtual /* [helpstring][id] */ HRESULT
STDMETHODCALLTYPE Abort( void) = 0;

      virtual /* [helpstring][id] */ HRESULT
STDMETHODCALLTYPE WriteError(
         BSTR szMsg) = 0;

      virtual /* [helpstring][id][propput] */ HRESULT
STDMETHODCALLTYPE put_OutputFile(
         /* [in] */ BSTR newVal) = 0;

      virtual /* [helpstring][id][propput] */ HRESULT
STDMETHODCALLTYPE put_ErrorFile(
         /* [in] */ BSTR newVal) = 0;

   };

#else      /* C style interface */

   typedef struct IExecuteVtbl
   {
      BEGIN_INTERFACE

      HRESULT ( STDMETHODCALLTYPE __RPC_FAR
*QueryInterface )(
         IExecute __RPC_FAR * This,
         /* [in] */ REFIID riid,
         /* [iid_is][out] */ void __RPC_FAR *__RPC_FAR
*ppvObject);

      ULONG ( STDMETHODCALLTYPE __RPC_FAR
*AddRef )(
         IExecute __RPC_FAR * This);

      ULONG ( STDMETHODCALLTYPE __RPC_FAR
*Release )(
         IExecute __RPC_FAR * This);

      HRESULT ( STDMETHODCALLTYPE __RPC_FAR
*GetTypeInfoCount )(
         IExecute __RPC_FAR * This,
         /* [out] */ UINT __RPC_FAR *pctinfo);

      HRESULT ( STDMETHODCALLTYPE __RPC_FAR
*GetTypeInfo )(
         IExecute __RPC_FAR * This,
         /* [in] */ UINT iTInfo,
         /* [in] */ LCID lcid,
         /* [out] */ ITypeInfo __RPC_FAR *__RPC_FAR
*ppTInfo);

      HRESULT ( STDMETHODCALLTYPE __RPC_FAR
*GetIDsOfNames )(
         IExecute __RPC_FAR * This,
         /* [in] */ REFIID riid,
         /* [size_is][in] */ LPOLESTR __RPC_FAR *rgszNames,
         /* [in] */ UINT cNames,
         /* [in] */ LCID lcid,
         /* [size_is][out] */ DISPID __RPC_FAR *rgDispId);
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
    /* [local] */ HRESULT ( STDMETHODCALLTYPE
__RPC_FAR *Invoke )(
        IExecute __RPC_FAR * This,
        /* [in] */ DISPID dispIdMember,
        /* [in] */ REFIID riid,
        /* [in] */ LCID lcid,
        /* [in] */ WORD wFlags,
        /* [out][in] */ DISPPARAMS __RPC_FAR
*pDispParams,
        /* [out] */ VARIANT __RPC_FAR *pVarResult,
        /* [out] */ EXCEPINFO __RPC_FAR *pExcepInfo,
        /* [out] */ UINT __RPC_FAR *puArgErr);

    /* [helpstring][id] */ HRESULT (
STDMETHODCALLTYPE __RPC_FAR *DoExecute )(
        IExecute __RPC_FAR * This,
        /* [in] */ BSTR szCommand,
        /* [in] */ BSTR szExecutionDtls,
        /* [in] */ ExecutionType ExecMethod,
        /* [in] */ BOOL bNoCount,
        /* [in] */ BOOL bNoExecute,
        /* [in] */ BOOL bParseOnly,
        /* [in] */ BOOL bQuotedIds,
        /* [in] */ BOOL bAnsiNulls,
        /* [in] */ BOOL bShowQP,
        /* [in] */ BOOL bStatsTime,
        /* [in] */ BOOL bStatsIO,
        /* [in] */ long lRowCount,
        /* [in] */ long lQueryTmout,
        /* [in] */ BSTR szConnection);

    /* [helpstring][id][propget] */ HRESULT (
STDMETHODCALLTYPE __RPC_FAR *get_StepStatus )(
        IExecute __RPC_FAR * This,
        /* [retval][out] */ InstanceStatus __RPC_FAR *pVal);

    /* [helpstring][id] */ HRESULT (
STDMETHODCALLTYPE __RPC_FAR *Abort )(
        IExecute __RPC_FAR * This);

    /* [helpstring][id] */ HRESULT (
STDMETHODCALLTYPE __RPC_FAR *WriteError )(
        IExecute __RPC_FAR * This,
        BSTR szMsg);

    /* [helpstring][id][propput] */ HRESULT (
STDMETHODCALLTYPE __RPC_FAR *put_OutputFile )(
        IExecute __RPC_FAR * This,
        /* [in] */ BSTR newVal);

    /* [helpstring][id][propput] */ HRESULT (
STDMETHODCALLTYPE __RPC_FAR *put_ErrorFile )(
        IExecute __RPC_FAR * This,
        /* [in] */ BSTR newVal);

    END_INTERFACE
  } IExecuteVtbl;

  interface IExecute
  {
    CONST_VTBL struct IExecuteVtbl __RPC_FAR *lpVtbl;
  };
```

```
#ifdef COBJMACROS

#define IExecute_QueryInterface(This,riid,ppvObject)     \
  (This)->lpVtbl -> QueryInterface(This,riid,ppvObject)

#define IExecute_AddRef(This)     \
  (This)->lpVtbl -> AddRef(This)

#define IExecute_Release(This)     \
  (This)->lpVtbl -> Release(This)

#define IExecute_GetTypeInfoCount(This,pctinfo)    \
  (This)->lpVtbl -> GetTypeInfoCount(This,pctinfo)

#define IExecute_GetTypeInfo(This,iTInfo,lcid,ppTInfo) \
  (This)->lpVtbl -> GetTypeInfo(This,iTInfo,lcid,ppTInfo)

#define
IExecute_GetIDsOfNames(This,riid,rgszNames,cNames,lcid,rgD
ispId)    \
  (This)->lpVtbl ->
GetIDsOfNames(This,riid,rgszNames,cNames,lcid,rgDispId)

#define
IExecute_Invoke(This,dispIdMember,riid,lcid,wFlags,pDispPara
ms,pVarResult,pExcepInfo,puArgErr)   \
  (This)->lpVtbl ->
Invoke(This,dispIdMember,riid,lcid,wFlags,pDispParams,pVarR
esult,pExcepInfo,puArgErr)


#define
IExecute_DoExecute(This,szCommand,szExecutionDtls,ExecMe
thod,bNoCount,bNoExecute,
    bParseOnly,bQuotedIds,bAnsiNulls,bShowQP,bStatsTime,b
StatsIO,lRowCount,lQueryTmout,szConnection)\
  (This)->lpVtbl ->
DoExecute(This,szCommand,szExecutionDtls,ExecMethod,bNo
Count,bNoExecute,bParseOnly,
    bQuotedIds,bAnsiNulls,bShowQP,bStatsTime,bStatsIO,lRo
wCount,lQueryTmout,szConnection)

#define IExecute_get_StepStatus(This,pVal) \
  (This)->lpVtbl -> get_StepStatus(This,pVal)

#define IExecute_Abort(This)  \
  (This)->lpVtbl -> Abort(This)

#define IExecute_WriteError(This,szMsg)   \
  (This)->lpVtbl -> WriteError(This,szMsg)

#define IExecute_put_OutputFile(This,newVal)  \
  (This)->lpVtbl -> put_OutputFile(This,newVal)

#define IExecute_put_ErrorFile(This,newVal)    \
  (This)->lpVtbl -> put_ErrorFile(This,newVal)

#endif /* COBJMACROS */


#endif    /* C style interface */
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
/* [helpstring][id] */ HRESULT STDMETHODCALLTYPE
IExecute_DoExecute_Proxy(
    IExecute __RPC_FAR * This,
    /* [in] */ BSTR szCommand,
    /* [in] */ BSTR szExecutionDtls,
    /* [in] */ ExecutionType ExecMethod,
    /* [in] */ BOOL bNoCount,
    /* [in] */ BOOL bNoExecute,
    /* [in] */ BOOL bParseOnly,
    /* [in] */ BOOL bQuotedIds,
    /* [in] */ BOOL bAnsiNulls,
    /* [in] */ BOOL bShowQP,
    /* [in] */ BOOL bStatsTime,
    /* [in] */ BOOL bStatsIO,
    /* [in] */ long lRowCount,
    /* [in] */ long lQueryTmout,
    /* [in] */ BSTR szConnection);


void __RPC_STUB IExecute_DoExecute_Stub(
    IRpcStubBuffer *This,
    IRpcChannelBuffer *_pRpcChannelBuffer,
    PRPC_MESSAGE _pRpcMessage,
    DWORD *_pdwStubPhase);


/* [helpstring][id][propget] */ HRESULT
STDMETHODCALLTYPE IExecute_get_StepStatus_Proxy(
    IExecute __RPC_FAR * This,
    /* [retval][out] */ InstanceStatus __RPC_FAR *pVal);


void __RPC_STUB IExecute_get_StepStatus_Stub(
    IRpcStubBuffer *This,
    IRpcChannelBuffer *_pRpcChannelBuffer,
    PRPC_MESSAGE _pRpcMessage,
    DWORD *_pdwStubPhase);


/* [helpstring][id] */ HRESULT STDMETHODCALLTYPE
IExecute_Abort_Proxy(
    IExecute __RPC_FAR * This);


void __RPC_STUB IExecute_Abort_Stub(
    IRpcStubBuffer *This,
    IRpcChannelBuffer *_pRpcChannelBuffer,
    PRPC_MESSAGE _pRpcMessage,
    DWORD *_pdwStubPhase);

/* [helpstring][id] */ HRESULT STDMETHODCALLTYPE
IExecute_WriteError_Proxy(
    IExecute __RPC_FAR * This,
    BSTR szMsg);

void __RPC_STUB IExecute_WriteError_Stub(
    IRpcStubBuffer *This,
    IRpcChannelBuffer *_pRpcChannelBuffer,
    PRPC_MESSAGE _pRpcMessage,
    DWORD *_pdwStubPhase);

/* [helpstring][id][propput] */ HRESULT
STDMETHODCALLTYPE IExecute_put_OutputFile_Proxy(
    IExecute __RPC_FAR * This,
    /* [in] */ BSTR newVal);


void __RPC_STUB IExecute_put_OutputFile_Stub(
    IRpcStubBuffer *This,
    IRpcChannelBuffer *_pRpcChannelBuffer,
    PRPC_MESSAGE _pRpcMessage,
    DWORD *_pdwStubPhase);


/* [helpstring][id][propput] */ HRESULT
STDMETHODCALLTYPE IExecute_put_ErrorFile_Proxy(
    IExecute __RPC_FAR * This,
    /* [in] */ BSTR newVal);


void __RPC_STUB IExecute_put_ErrorFile_Stub(
    IRpcStubBuffer *This,
    IRpcChannelBuffer *_pRpcChannelBuffer,
    PRPC_MESSAGE _pRpcMessage,
    DWORD *_pdwStubPhase);



#endif    /* __IExecute_INTERFACE_DEFINED__ */


EXTERN_C const CLSID CLSID_Execute;

#ifdef __cplusplus

class DECLSPEC_UUID("2EFC198E-AA8D-11D2-BC0C-
00A0C90D2CA5")
Execute;
#endif
#endif /* __EXECUTEDLLLib_LIBRARY_DEFINED__ */

/* Additional Prototypes for ALL interfaces */

/* end of Additional Prototypes */

#ifdef __cplusplus
}
#endif

#endif
//  FILE:      Execute.cpp
//    //  FILE:      ExecuteDllCP.h
//            Microsoft TPC-H Kit Ver. 1.00
//            Copyright Microsoft, 1999
//            All Rights Reserved
//
//
//  PURPOSE:    Connection point implementation
//  Contact:    Reshma Tharamal (reshmat@microsoft.com)
//
#ifndef _EXECUTEDLLCP_H_
#define _EXECUTEDLLCP_H_
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
template <class T>
class CProxy_IExecuteEvents : public IConnectionPointImpl<T,
&DIID__IExecuteEvents, CComDynamicUnkArray>
{
	//Warning this class may be recreated by the wizard.
public:
	VOID Fire_Start(CY StartTime)
	{
		T* pT = static_cast<T*>(this);
		int nConnectionIndex;
		CComVariant* pvars = new CComVariant[1];
		int nConnections = m_vec.GetSize();

		for (nConnectionIndex = 0; nConnectionIndex <
nConnections; nConnectionIndex++)
		{
			pT->Lock();
			CComPtr<IUnknown> sp =
m_vec.GetAt(nConnectionIndex);
			pT->Unlock();
			IDispatch* pDispatch =
reinterpret_cast<IDispatch*>(sp.p);
			if (pDispatch != NULL)
			{
			pvars[0] = StartTime;
			DISPPARAMS disp = { pvars, NULL, 1, 0 };
			pDispatch->Invoke(0x1, IID_NULL,
LOCALE_USER_DEFAULT, DISPATCH_METHOD,
			&disp, NULL, NULL, NULL);
			}
		}
		delete[] pvars;

	}
	VOID Fire_Complete(CY EndTime, LONG Elapsed)
	{
		T* pT = static_cast<T*>(this);
		int nConnectionIndex;
		CComVariant* pvars = new CComVariant[2];
		int nConnections = m_vec.GetSize();
		for (nConnectionIndex = 0; nConnectionIndex <
nConnections; nConnectionIndex++)
		{
			pT->Lock();
			CComPtr<IUnknown> sp =
m_vec.GetAt(nConnectionIndex);
			pT->Unlock();
			IDispatch* pDispatch =
reinterpret_cast<IDispatch*>(sp.p);
	if (pDispatch != NULL)
	{
	pvars[1] = EndTime;
	pvars[0] = Elapsed;
	DISPPARAMS disp = { pvars, NULL, 2, 0 };
			pDispatch->Invoke(0x2, IID_NULL,
LOCALE_USER_DEFAULT, DISPATCH_METHOD,
&disp, NULL, NULL, NULL);
}
}
delete[] pvars;
}
};
```

```
#endif//  FILE:	resource.h
//		Microsoft TPC-H Kit Ver. 1.00
//		Copyright Microsoft, 1999
//		All Rights Reserved
//
//
//  PURPOSE:	Resource file
//  Contact:	Reshma Tharamal (reshmat@microsoft.com)
//
//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.
// Used by ExecuteDll.rc
//
#define IDS_PROJNAME		100
#define IDR_EXECUTE		101
#define IDR_LOG		102
#define IDR_EXECUTESHELL		103

#define SM_SYSTEM_ERROR		0x0201

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE	201
#define _APS_NEXT_COMMAND_VALUE	32768
#define _APS_NEXT_CONTROL_VALUE	201
#define _APS_NEXT_SYMED_VALUE	104
#endif
#endif
//  FILE:	SMExecute.h
//		Microsoft TPC-H Kit Ver. 1.00
//		Copyright Microsoft, 1999
//		All Rights Reserved
//
//
//  PURPOSE:	Common include file for Execute.cpp and
ExecuteDll.cpp
//  Contact:	Reshma Tharamal (reshmat@microsoft.com)
//

#pragma once

// ODBC-specific includes
#define DBNTWIN32
#include <sqltypes.h>
#include <sql.h>
#include <sqlext.h>

#define CONNECTION_NAME_LEN 256	// connection
name length

typedef struct _SM_Connection_Info
{
	charszConnectionName[CONNECTION_NAME_LEN];
	HDBC	hdbc;
	BOOL	bInUse;
} SM_Connection_Info;
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
// FILE:      stdafx.cpp
//            Microsoft TPC-H Kit Ver. 1.00
//            Copyright Microsoft, 1999
//            All Rights Reserved
//
//
// PURPOSE:   source file that includes just the standard
includes
// Contact:   Reshma Tharamal (reshmat@microsoft.com)
//
// stdafx.cpp : source file that includes just the standard includes
// stdafx.pch will be the pre-compiled header
// stdafx.obj will contain the pre-compiled type information

#include "stdafx.h"

#ifdef _ATL_STATIC_REGISTRY
#include <statreg.h>
#include <statreg.cpp>
#endif

#include <atlimpl.cpp>
// FILE:      stdafx.h
//            Microsoft TPC-H Kit Ver. 1.00
//            Copyright Microsoft, 1999
//            All Rights Reserved
//
//
// PURPOSE:   include file for standard system include files,
//     or project specific include files that are used frequently,
//     but are changed infrequently
// Contact:   Reshma Tharamal (reshmat@microsoft.com)
//

#if
!defined(AFX_STDAFX_H__551AC528_AB1C_11D2_BC0C_
00A0C90D2CA5__INCLUDED_)
#define
AFX_STDAFX_H__551AC528_AB1C_11D2_BC0C_00A0C90
D2CA5__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#define STRICT
#ifndef _WIN32_WINNT
#define _WIN32_WINNT 0x0400
#endif
#define _ATL_APARTMENT_THREADED

#include <atlbase.h>
//You may derive a class from CComModule and use it if you
want to override
//something, but do not change the name of _Module
extern CComModule _Module;
#include <atlcom.h>

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations
immediately before the previous line.

#endif //
!defined(AFX_STDAFX_H__551AC528_AB1C_11D2_BC0C_
00A0C90D2CA5__INCLUDED)
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
//  FILE:      resource.h
//            Microsoft TPC-H Kit Ver. 1.00
//            Copyright Microsoft, 1999
//            All Rights Reserved
//
//
//  PURPOSE:    Resource file
//  Contact:    Reshma Tharamal (reshmat@microsoft.com)
//
//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.
// Used by SMTime.rc
//
#define IDS_PROJNAME                100
#define IDR_SMTIMER                 102

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE        201
#define _APS_NEXT_COMMAND_VALUE         32768
#define _APS_NEXT_CONTROL_VALUE         201
#define _APS_NEXT_SYMED_VALUE           103
#endif
#endif
//  FILE:      SMTime.cpp
//            Microsoft TPC-H Kit Ver. 1.00
//            Copyright Microsoft, 1999
//            All Rights Reserved
//
//
//  PURPOSE:    Implementation of DLL Exports.
//  Contact:    Reshma Tharamal (reshmat@microsoft.com)
//
// SMTime.cpp : Implementation of DLL Exports.


// Note: Proxy/Stub Information
//      To build a separate proxy/stub DLL,
//      run nmake -f SMTimeps.mk in the project directory.

#include "stdafx.h"
#include "resource.h"
#include <initguid.h>
#include "SMTime.h"

#include "SMTime_i.c"
#include "SMTimer.h"


CComModule _Module;

BEGIN_OBJECT_MAP(ObjectMap)
OBJECT_ENTRY(CLSID_SMTimer, CSMTimer)
END_OBJECT_MAP()
```

```
/////////////////////////////////////////////////////////////////////////////
// DLL Entry Point

extern "C"
BOOL WINAPI DllMain(HINSTANCE hInstance, DWORD
dwReason, LPVOID /*lpReserved*/)
{
   if (dwReason == DLL_PROCESS_ATTACH)
   {
      _Module.Init(ObjectMap, hInstance,
&LIBID_SMTIMELib);
      DisableThreadLibraryCalls(hInstance);
   }
   else if (dwReason == DLL_PROCESS_DETACH)
      _Module.Term();
   return TRUE;    // ok
}

/////////////////////////////////////////////////////////////////////////////
// Used to determine whether the DLL can be unloaded by OLE

STDAPI DllCanUnloadNow(void)
{
   return (_Module.GetLockCount()==0) ? S_OK : S_FALSE;
}

/////////////////////////////////////////////////////////////////////////////
// Returns a class factory to create an object of the requested type

STDAPI DllGetClassObject(REFCLSID rclsid, REFIID riid,
LPVOID* ppv)
{
   return _Module.GetClassObject(rclsid, riid, ppv);
}

/////////////////////////////////////////////////////////////////////////////
// DllRegisterServer - Adds entries to the system registry

STDAPI DllRegisterServer(void)
{
   // registers object, typelib and all interfaces in typelib
   return _Module.RegisterServer(TRUE);
}

/////////////////////////////////////////////////////////////////////////////
// DllUnregisterServer - Removes entries from the system
registry

STDAPI DllUnregisterServer(void)
{
   return _Module.UnregisterServer(TRUE);
}


/* this ALWAYS GENERATED file contains the definitions for
the interfaces */


/* File created by MIDL compiler version 5.01.0164 */
/* at Thu Sep 19 17:49:55 2002
 */
/* Compiler settings for
C:\charles\Stepmaster\COMMON\SMTime\SMTime.idl:
   Oicf (OptLev=i2), W1, Zp8, env=Win32, ms_ext, c_ext
   error checks: allocation ref bounds_check enum stub_data
*/
//@@MIDL_FILE_HEADING(  )
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
/* verify that the <rpcndr.h> version is high enough to compile
this file*/
#ifndef __REQUIRED_RPCNDR_H_VERSION__
#define __REQUIRED_RPCNDR_H_VERSION__ 440
#endif

#include "rpc.h"
#include "rpcndr.h"

#ifndef __RPCNDR_H_VERSION__
#error this stub requires an updated version of <rpcndr.h>
#endif // __RPCNDR_H_VERSION__

#ifndef COM_NO_WINDOWS_H
#include "windows.h"
#include "ole2.h"
#endif /*COM_NO_WINDOWS_H*/

#ifndef __SMTime_h__
#define __SMTime_h__

#ifdef __cplusplus
extern "C"{
#endif

/* Forward Declarations */

#ifndef __ISMTimer_FWD_DEFINED__
#define __ISMTimer_FWD_DEFINED__
typedef interface ISMTimer ISMTimer;
#endif    /* __ISMTimer_FWD_DEFINED__ */


#ifndef __SMTimer_FWD_DEFINED__
#define __SMTimer_FWD_DEFINED__

#ifdef __cplusplus
typedef class SMTimer SMTimer;
#else
typedef struct SMTimer SMTimer;
#endif /* __cplusplus */

#endif    /* __SMTimer_FWD_DEFINED__ */


/* header files for imported files */
#include "oaidl.h"
#include "ocidl.h"

void __RPC_FAR * __RPC_USER MIDL_user_allocate(size_t);
void __RPC_USER MIDL_user_free( void __RPC_FAR * );

#ifndef __ISMTimer_INTERFACE_DEFINED__
#define __ISMTimer_INTERFACE_DEFINED__

/* interface ISMTimer */
/* [unique][helpstring][dual][uuid][object] */

EXTERN_C const IID IID_ISMTimer;

#if defined(__cplusplus) && !defined(CINTERFACE)

   MIDL_INTERFACE("1A6D0AE4-8528-453B-B8E3-
8DAD1F0561B7")
   ISMTimer : public IDispatch
   {
   public:
      virtual /* [helpstring][id] */ HRESULT
STDMETHODCALLTYPE Start( void) = 0;

      virtual /* [helpstring][id] */ HRESULT
STDMETHODCALLTYPE Stop(
         CURRENCY __RPC_FAR *pElapsedTime) = 0;

      virtual /* [helpstring][id][propget] */ HRESULT
STDMETHODCALLTYPE get_Running(
         /* [retval][out] */ BOOL __RPC_FAR *pVal) = 0;

   };

#else     /* C style interface */

   typedef struct ISMTimerVtbl
   {
      BEGIN_INTERFACE

      HRESULT ( STDMETHODCALLTYPE __RPC_FAR
*QueryInterface )(
         ISMTimer __RPC_FAR * This,
         /* [in] */ REFIID riid,
         /* [iid_is][out] */ void __RPC_FAR *__RPC_FAR
*ppvObject);

      ULONG ( STDMETHODCALLTYPE __RPC_FAR
*AddRef )(
         ISMTimer __RPC_FAR * This);

      ULONG ( STDMETHODCALLTYPE __RPC_FAR
*Release )(
         ISMTimer __RPC_FAR * This);

      HRESULT ( STDMETHODCALLTYPE __RPC_FAR
*GetTypeInfoCount )(
         ISMTimer __RPC_FAR * This,
         /* [out] */ UINT __RPC_FAR *pctinfo);

      HRESULT ( STDMETHODCALLTYPE __RPC_FAR
*GetTypeInfo )(
         ISMTimer __RPC_FAR * This,
         /* [in] */ UINT iTInfo,
         /* [in] */ LCID lcid,
         /* [out] */ ITypeInfo __RPC_FAR *__RPC_FAR
*ppTInfo);

      HRESULT ( STDMETHODCALLTYPE __RPC_FAR
*GetIDsOfNames )(
         ISMTimer __RPC_FAR * This,
         /* [in] */ REFIID riid,
         /* [size_is][in] */ LPOLESTR __RPC_FAR *rgszNames,
         /* [in] */ UINT cNames,
         /* [in] */ LCID lcid,
         /* [size_is][out] */ DISPID __RPC_FAR *rgDispId);

      /* [local] */ HRESULT ( STDMETHODCALLTYPE
__RPC_FAR *Invoke )(
         ISMTimer __RPC_FAR * This,
         /* [in] */ DISPID dispIdMember,
         /* [in] */ REFIID riid,
         /* [in] */ LCID lcid,
         /* [in] */ WORD wFlags,
         /* [out][in] */ DISPPARAMS __RPC_FAR
*pDispParams,
         /* [out] */ VARIANT __RPC_FAR *pVarResult,
         /* [out] */ EXCEPINFO __RPC_FAR *pExcepInfo,
         /* [out] */ UINT __RPC_FAR *puArgErr);
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
    /* [helpstring][id] */ HRESULT (
STDMETHODCALLTYPE __RPC_FAR *Start )(
        ISMTimer __RPC_FAR * This);

    /* [helpstring][id] */ HRESULT (
STDMETHODCALLTYPE __RPC_FAR *Stop )(
        ISMTimer __RPC_FAR * This,
        CURRENCY __RPC_FAR *pElapsedTime);

    /* [helpstring][id][propget] */ HRESULT (
STDMETHODCALLTYPE __RPC_FAR *get_Running )(
        ISMTimer __RPC_FAR * This,
        /* [retval][out] */ BOOL __RPC_FAR *pVal);

    END_INTERFACE
  } ISMTimerVtbl;

  interface ISMTimer
  {
    CONST_VTBL struct ISMTimerVtbl __RPC_FAR *lpVtbl;
  };

#ifdef COBJMACROS

#define ISMTimer_QueryInterface(This,riid,ppvObject)   \
  (This)->lpVtbl -> QueryInterface(This,riid,ppvObject)

#define ISMTimer_AddRef(This)   \
  (This)->lpVtbl -> AddRef(This)

#define ISMTimer_Release(This)   \
  (This)->lpVtbl -> Release(This)

#define ISMTimer_GetTypeInfoCount(This,pctinfo)  \
  (This)->lpVtbl -> GetTypeInfoCount(This,pctinfo)

#define ISMTimer_GetTypeInfo(This,iTInfo,lcid,ppTInfo)      \
  (This)->lpVtbl -> GetTypeInfo(This,iTInfo,lcid,ppTInfo)

#define
ISMTimer_GetIDsOfNames(This,riid,rgszNames,cNames,lcid,rg
DispId)  \
  (This)->lpVtbl ->
GetIDsOfNames(This,riid,rgszNames,cNames,lcid,rgDispId)

#define
ISMTimer_Invoke(This,dispIdMember,riid,lcid,wFlags,pDispPar
ams,pVarResult,pExcepInfo,puArgErr)  \
  (This)->lpVtbl ->
Invoke(This,dispIdMember,riid,lcid,wFlags,pDispParams,pVarR
esult,pExcepInfo,puArgErr)

#define ISMTimer_Start(This) \
  (This)->lpVtbl -> Start(This)

#define ISMTimer_Stop(This,pElapsedTime)      \
  (This)->lpVtbl -> Stop(This,pElapsedTime)

#define ISMTimer_get_Running(This,pVal) \
  (This)->lpVtbl -> get_Running(This,pVal)

#endif /* COBJMACROS */

#endif    /* C style interface */
```

```
/* [helpstring][id] */ HRESULT STDMETHODCALLTYPE
ISMTimer_Start_Proxy(
  ISMTimer __RPC_FAR * This);

void __RPC_STUB ISMTimer_Start_Stub(
  IRpcStubBuffer *This,
  IRpcChannelBuffer *_pRpcChannelBuffer,
  PRPC_MESSAGE _pRpcMessage,
  DWORD *_pdwStubPhase);

/* [helpstring][id] */ HRESULT STDMETHODCALLTYPE
ISMTimer_Stop_Proxy(
  ISMTimer __RPC_FAR * This,
  CURRENCY __RPC_FAR *pElapsedTime);

void __RPC_STUB ISMTimer_Stop_Stub(
  IRpcStubBuffer *This,
  IRpcChannelBuffer *_pRpcChannelBuffer,
  PRPC_MESSAGE _pRpcMessage,
  DWORD *_pdwStubPhase);

/* [helpstring][id][propget] */ HRESULT
STDMETHODCALLTYPE ISMTimer_get_Running_Proxy(
  ISMTimer __RPC_FAR * This,
  /* [retval][out] */ BOOL __RPC_FAR *pVal);

void __RPC_STUB ISMTimer_get_Running_Stub(
  IRpcStubBuffer *This,
  IRpcChannelBuffer *_pRpcChannelBuffer,
  PRPC_MESSAGE _pRpcMessage,
  DWORD *_pdwStubPhase);

#endif    /* __ISMTimer_INTERFACE_DEFINED__ */
#ifndef __SMTIMELib_LIBRARY_DEFINED__
#define __SMTIMELib_LIBRARY_DEFINED__

/* library SMTIMELib */
/* [helpstring][version][uuid] */


EXTERN_C const IID LIBID_SMTIMELib;


#ifndef __StepMasterTimeFunctions_MODULE_DEFINED__
#define __StepMasterTimeFunctions_MODULE_DEFINED__


/* module StepMasterTimeFunctions */
/* [dllname][version][helpstring] */

/* [entry][helpstring] */ CURRENCY __stdcall Get64BitTime(
  /* [in] */ LPSYSTEMTIME lpInitTime);

/* [entry][helpstring] */ void __stdcall JulianToTime(
  /* [in] */ CURRENCY julianTS,
  /* [out][in] */ int __RPC_FAR *yr,
  /* [out][in] */ int __RPC_FAR *mm,
  /* [out][in] */ int __RPC_FAR *dd,
  /* [out][in] */ int __RPC_FAR *hh,
  /* [out][in] */ int __RPC_FAR *mi,
  /* [out][in] */ int __RPC_FAR *ss,
  /* [out][in] */ int __RPC_FAR *ms);
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
#endif /* __StepMasterTimeFunctions_MODULE_DEFINED__
*/

EXTERN_C const CLSID CLSID_SMTimer;

#ifdef __cplusplus

class DECLSPEC_UUID("27BAB71B-89E1-4A78-8854-
FDFFBDC8037E")
SMTimer;
#endif
#endif /* __SMTIMELib_LIBRARY_DEFINED__ */

/* Additional Prototypes for ALL interfaces */

/* end of Additional Prototypes */

#ifdef __cplusplus
}
#endif

#endif
//  FILE:      resource.h
//            Microsoft TPC-H Kit Ver. 1.00
//            Copyright Microsoft, 1999
//            All Rights Reserved
//
//
//  PURPOSE:    Resource file
//  Contact:    Reshma Tharamal (reshmat@microsoft.com)
//
//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.
// Used by SMTime.rc
//
#define IDS_PROJNAME                    100
#define IDR_SMTIMER                     102

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE        201
#define _APS_NEXT_COMMAND_VALUE         32768
#define _APS_NEXT_CONTROL_VALUE         201
#define _APS_NEXT_SYMED_VALUE           103
#endif
#endif
```

```
//  FILE:      SMTime.cpp
//            Microsoft TPC-H Kit Ver. 1.00
//            Copyright Microsoft, 1999
//            All Rights Reserved
//
//
//  PURPOSE:    Implementation of DLL Exports.
//  Contact:    Reshma Tharamal (reshmat@microsoft.com)
//
// SMTime.cpp : Implementation of DLL Exports.


// Note: Proxy/Stub Information
//     To build a separate proxy/stub DLL,
//     run nmake -f SMTimeps.mk in the project directory.

#include "stdafx.h"
#include "resource.h"
#include <initguid.h>
#include "SMTime.h"

#include "SMTime_i.c"
#include "SMTimer.h"


CComModule _Module;

BEGIN_OBJECT_MAP(ObjectMap)
OBJECT_ENTRY(CLSID_SMTimer, CSMTimer)
END_OBJECT_MAP()

/////////////////////////////////////////////////////////////////////////////
// DLL Entry Point

extern "C"
BOOL WINAPI DllMain(HINSTANCE hInstance, DWORD
dwReason, LPVOID /*lpReserved*/)
{
   if (dwReason == DLL_PROCESS_ATTACH)
   {
      _Module.Init(ObjectMap, hInstance,
&LIBID_SMTIMELib);
      DisableThreadLibraryCalls(hInstance);
   }
   else if (dwReason == DLL_PROCESS_DETACH)
      _Module.Term();
   return TRUE;    // ok
}

/////////////////////////////////////////////////////////////////////////////
// Used to determine whether the DLL can be unloaded by OLE

STDAPI DllCanUnloadNow(void)
{
   return (_Module.GetLockCount()==0) ? S_OK : S_FALSE;
}

/////////////////////////////////////////////////////////////////////////////
// Returns a class factory to create an object of the requested type

STDAPI DllGetClassObject(REFCLSID rclsid, REFIID riid,
LPVOID* ppv)
{
   return _Module.GetClassObject(rclsid, riid, ppv);
}
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
///////////////////////////////////////////////////////////////////////////
// DllRegisterServer - Adds entries to the system registry

STDAPI DllRegisterServer(void)
{
   // registers object, typelib and all interfaces in typelib
   return _Module.RegisterServer(TRUE);
}

///////////////////////////////////////////////////////////////////////////
// DllUnregisterServer - Removes entries from the system
registry

STDAPI DllUnregisterServer(void)
{
   return _Module.UnregisterServer(TRUE);
}


/* this ALWAYS GENERATED file contains the definitions for
the interfaces */


/* File created by MIDL compiler version 5.01.0164 */
/* at Thu Sep 19 17:49:55 2002
 */
/* Compiler settings for
C:\charles\Stepmaster\COMMON\SMTime\SMTime.idl:
   Oicf (OptLev=i2), W1, Zp8, env=Win32, ms_ext, c_ext
   error checks: allocation ref bounds_check enum stub_data
*/
//@@MIDL_FILE_HEADING(  )


/* verify that the <rpcndr.h> version is high enough to compile
this file*/
#ifndef __REQUIRED_RPCNDR_H_VERSION__
#define __REQUIRED_RPCNDR_H_VERSION__ 440
#endif

#include "rpc.h"
#include "rpcndr.h"

#ifndef __RPCNDR_H_VERSION__
#error this stub requires an updated version of <rpcndr.h>
#endif // __RPCNDR_H_VERSION__

#ifndef COM_NO_WINDOWS_H
#include "windows.h"
#include "ole2.h"
#endif /*COM_NO_WINDOWS_H*/

#ifndef __SMTime_h__
#define __SMTime_h__

#ifdef __cplusplus
extern "C"{
#endif
```

```
/* Forward Declarations */

#ifndef __ISMTimer_FWD_DEFINED__
#define __ISMTimer_FWD_DEFINED__
typedef interface ISMTimer ISMTimer;
#endif    /* __ISMTimer_FWD_DEFINED__ */


#ifndef __SMTimer_FWD_DEFINED__
#define __SMTimer_FWD_DEFINED__

#ifdef __cplusplus
typedef class SMTimer SMTimer;
#else
typedef struct SMTimer SMTimer;
#endif /* __cplusplus */

#endif    /* __SMTimer_FWD_DEFINED__ */


/* header files for imported files */
#include "oaidl.h"
#include "ocidl.h"

void __RPC_FAR * __RPC_USER MIDL_user_allocate(size_t);
void __RPC_USER MIDL_user_free( void __RPC_FAR * );

#ifndef __ISMTimer_INTERFACE_DEFINED__
#define __ISMTimer_INTERFACE_DEFINED__

/* interface ISMTimer */
/* [unique][helpstring][dual][uuid][object] */


EXTERN_C const IID IID_ISMTimer;

#if defined(__cplusplus) && !defined(CINTERFACE)

   MIDL_INTERFACE("1A6D0AE4-8528-453B-B8E3-
8DAD1F0561B7")
   ISMTimer : public IDispatch
   {
   public:
      virtual /* [helpstring][id] */ HRESULT
STDMETHODCALLTYPE Start( void) = 0;

      virtual /* [helpstring][id] */ HRESULT
STDMETHODCALLTYPE Stop(
         CURRENCY __RPC_FAR *pElapsedTime) = 0;

      virtual /* [helpstring][id][propget] */ HRESULT
STDMETHODCALLTYPE get_Running(
         /* [retval][out] */ BOOL __RPC_FAR *pVal) = 0;

   };
#else      /* C style interface */
   typedef struct ISMTimerVtbl
   {
      BEGIN_INTERFACE

      HRESULT ( STDMETHODCALLTYPE __RPC_FAR
*QueryInterface )(
         ISMTimer __RPC_FAR * This,
         /* [in] */ REFIID riid,
         /* [iid_is][out] */ void __RPC_FAR *__RPC_FAR
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
*ppvObject);
    ULONG ( STDMETHODCALLTYPE __RPC_FAR
*AddRef )(
        ISMTimer __RPC_FAR * This);
    ULONG ( STDMETHODCALLTYPE __RPC_FAR
*Release )(
        ISMTimer __RPC_FAR * This);
    HRESULT ( STDMETHODCALLTYPE __RPC_FAR
*GetTypeInfoCount )(
        ISMTimer __RPC_FAR * This,
        /* [out] */ UINT __RPC_FAR *pctinfo);
    HRESULT ( STDMETHODCALLTYPE __RPC_FAR
*GetTypeInfo )(
        ISMTimer __RPC_FAR * This,
        /* [in] */ UINT iTInfo,
        /* [in] */ LCID lcid,
        /* [out] */ ITypeInfo __RPC_FAR *__RPC_FAR
*ppTInfo);
    HRESULT ( STDMETHODCALLTYPE __RPC_FAR
*GetIDsOfNames )(
        ISMTimer __RPC_FAR * This,
        /* [in] */ REFIID riid,
        /* [size_is][in] */ LPOLESTR __RPC_FAR *rgszNames,
        /* [in] */ UINT cNames,
        /* [in] */ LCID lcid,
        /* [size_is][out] */ DISPID __RPC_FAR *rgDispId);
    /* [local] */ HRESULT ( STDMETHODCALLTYPE
__RPC_FAR *Invoke )(
        ISMTimer __RPC_FAR * This,
        /* [in] */ DISPID dispIdMember,
        /* [in] */ REFIID riid,
        /* [in] */ LCID lcid,
        /* [in] */ WORD wFlags,
        /* [out][in] */ DISPPARAMS __RPC_FAR
*pDispParams,
        /* [out] */ VARIANT __RPC_FAR *pVarResult,
        /* [out] */ EXCEPINFO __RPC_FAR *pExcepInfo,
        /* [out] */ UINT __RPC_FAR *puArgErr);
    END_INTERFACE
  } ISMTimerVtbl;
  interface ISMTimer
  {
    CONST_VTBL struct ISMTimerVtbl __RPC_FAR *lpVtbl;
  };
    /* [helpstring][id] */ HRESULT (
STDMETHODCALLTYPE __RPC_FAR *Start )(
        ISMTimer __RPC_FAR * This);
    /* [helpstring][id] */ HRESULT (
STDMETHODCALLTYPE __RPC_FAR *Stop )(
        ISMTimer __RPC_FAR * This,
        CURRENCY __RPC_FAR *pElapsedTime);
    /* [helpstring][id][propget] */ HRESULT (
STDMETHODCALLTYPE __RPC_FAR *get_Running )(
        ISMTimer __RPC_FAR * This,
        /* [retval][out] */ BOOL __RPC_FAR *pVal);

#ifdef COBJMACROS

#define ISMTimer_QueryInterface(This,riid,ppvObject)   \
  (This)->lpVtbl -> QueryInterface(This,riid,ppvObject)

#define ISMTimer_AddRef(This)   \
  (This)->lpVtbl -> AddRef(This)
```

```
#define ISMTimer_Release(This)   \
  (This)->lpVtbl -> Release(This)

#define ISMTimer_GetTypeInfoCount(This,pctinfo)  \
  (This)->lpVtbl -> GetTypeInfoCount(This,pctinfo)

#define ISMTimer_GetTypeInfo(This,iTInfo,lcid,ppTInfo)      \
  (This)->lpVtbl -> GetTypeInfo(This,iTInfo,lcid,ppTInfo)

#define
ISMTimer_GetIDsOfNames(This,riid,rgszNames,cNames,lcid,rg
DispId)  \
  (This)->lpVtbl ->
GetIDsOfNames(This,riid,rgszNames,cNames,lcid,rgDispId)

#define
ISMTimer_Invoke(This,dispIdMember,riid,lcid,wFlags,pDispPar
ams,pVarResult,pExcepInfo,puArgErr)  \
  (This)->lpVtbl ->
Invoke(This,dispIdMember,riid,lcid,wFlags,pDispParams,pVarR
esult,pExcepInfo,puArgErr)


#define ISMTimer_Start(This) \
  (This)->lpVtbl -> Start(This)

#define ISMTimer_Stop(This,pElapsedTime)      \
  (This)->lpVtbl -> Stop(This,pElapsedTime)

#define ISMTimer_get_Running(This,pVal) \
  (This)->lpVtbl -> get_Running(This,pVal)

#endif /* COBJMACROS */


#endif    /* C style interface */



/* [helpstring][id] */ HRESULT STDMETHODCALLTYPE
ISMTimer_Start_Proxy(
  ISMTimer __RPC_FAR * This);


void __RPC_STUB ISMTimer_Start_Stub(
  IRpcStubBuffer *This,
  IRpcChannelBuffer *_pRpcChannelBuffer,
  PRPC_MESSAGE _pRpcMessage,
  DWORD *_pdwStubPhase);


/* [helpstring][id] */ HRESULT STDMETHODCALLTYPE
ISMTimer_Stop_Proxy(
  ISMTimer __RPC_FAR * This,
  CURRENCY __RPC_FAR *pElapsedTime);

void __RPC_STUB ISMTimer_Stop_Stub(
  IRpcStubBuffer *This,
  IRpcChannelBuffer *_pRpcChannelBuffer,
  PRPC_MESSAGE _pRpcMessage,
  DWORD *_pdwStubPhase);
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
/* [helpstring][id][propget] */ HRESULT
STDMETHODCALLTYPE ISMTimer_get_Running_Proxy(
    ISMTimer __RPC_FAR * This,
    /* [retval][out] */ BOOL __RPC_FAR *pVal);

void __RPC_STUB ISMTimer_get_Running_Stub(
    IRpcStubBuffer *This,
    IRpcChannelBuffer *_pRpcChannelBuffer,
    PRPC_MESSAGE _pRpcMessage,
    DWORD *_pdwStubPhase);

#endif    /* __ISMTimer_INTERFACE_DEFINED__ */

#ifndef __SMTIMELib_LIBRARY_DEFINED__
#define __SMTIMELib_LIBRARY_DEFINED__

/* library SMTIMELib */
/* [helpstring][version][uuid] */

EXTERN_C const IID LIBID_SMTIMELib;

#ifndef __StepMasterTimeFunctions_MODULE_DEFINED__
#define __StepMasterTimeFunctions_MODULE_DEFINED__


/* module StepMasterTimeFunctions */
/* [dllname][version][helpstring] */

/* [entry][helpstring] */ CURRENCY __stdcall Get64BitTime(
    /* [in] */ LPSYSTEMTIME lpInitTime);

/* [entry][helpstring] */ void __stdcall JulianToTime(
    /* [in] */ CURRENCY julianTS,
    /* [out][in] */ int __RPC_FAR *yr,
    /* [out][in] */ int __RPC_FAR *mm,
    /* [out][in] */ int __RPC_FAR *dd,
    /* [out][in] */ int __RPC_FAR *hh,
    /* [out][in] */ int __RPC_FAR *mi,
    /* [out][in] */ int __RPC_FAR *ss,
    /* [out][in] */ int __RPC_FAR *ms);

#endif /* __StepMasterTimeFunctions_MODULE_DEFINED__
*/

EXTERN_C const CLSID CLSID_SMTimer;

#ifdef __cplusplus

class DECLSPEC_UUID("27BAB71B-89E1-4A78-8854-
FDFFBDC8037E")
SMTimer;
#endif
#endif /* __SMTIMELib_LIBRARY_DEFINED__ */

/* Additional Prototypes for ALL interfaces */

/* end of Additional Prototypes */

#ifdef __cplusplus
}
#endif
```

```
//  FILE:       resource.h
//        //  FILE:       SMTimer.cpp
//            Microsoft TPC-H Kit Ver. 1.00
//            Copyright Microsoft, 1999
//            All Rights Reserved
//
//
//  PURPOSE:    Implementation of CSMTimer
//  Contact:    Reshma Tharamal (reshmat@microsoft.com)
//
// SMTimer.cpp : Implementation of CSMTimer
#include "stdafx.h"
#include "SMTime.h"
#include "SMTimer.h"

/////////////////////////////////////////////////////////////////////////////
// CSMTimer

/////////////////////////////////////////////////////////////////////////////
// Construction/Destruction
/////////////////////////////////////////////////////////////////////////////

CSMTimer::~CSMTimer()
{

}

STDMETHODIMP CSMTimer::Start()
{
    // Starts the timer
    assert(!m_bInProcess);
    m_bInProcess = TRUE;

    m_lStartTime = MyTickCount();

    return S_OK;
}

STDMETHODIMP CSMTimer::Stop(CURRENCY
*pElapsedTime)
{
    TC_TIME         lEndTime = MyTickCount();

    // Stops the timer and returns the elapsed time
    assert(m_bInProcess);
    m_bInProcess = FALSE;

    pElapsedTime->int64 = lEndTime - m_lStartTime;

    return S_OK;
}
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
TC_TIME CSMTimer::MyTickCount(void)
{
	TC_TIME	currentTC;
	LARGE_INTEGER	l;
	__int64 count;

	//The purpose of this function is to prevent the 49 day
wrapping effect of the
	//system API GetTickCount(). This function essentially
provides a monotonically
	//increasing timer value which is milliseconds from class
instantiation.

	if ( m_bCountUnavailable )
	{
		count = (__int64)GetTickCount();
		currentTC = (TC_TIME)(count-m_baseTC);
	}
	else
	{
		QueryPerformanceCounter(&l);
		count = (__int64)l.HighPart << 32 | (__int64)l.LowPart;
		currentTC = (TC_TIME)(((count-m_baseTC) * 1000) /
m_Timerfreq);
	}

	return currentTC;
}

STDMETHODIMP CSMTimer::get_Running(BOOL *pVal)
{
	*pVal = m_bInProcess;

	return S_OK;
}

CURRENCY __stdcall Get64BitTime(LPSYSTEMTIME
lpInitTime)
{
	__int64  ms_day, ms_hour, ms_minute, ms_seconds,
ms_milliseconds, ms_total;
	int	day;
	SYSTEMTIME	tim;
	CURRENCY	tmReturn;
	if ( lpInitTime )
		memcpy(&tim, lpInitTime, sizeof(SYSTEMTIME));
	else
	GetLocalTime(&tim);
	day = JulianDay((int)tim.wYear, (int)tim.wMonth,
(int)tim.wDay);
ms_day		= (__int64)day * (__int64)(24 * 1000 * 60 *
60);
	ms_hour		= (__int64)tim.wHour* (__int64)(1000 *
3600);
	ms_minute	= (__int64)tim.wMinute * (1000 * 60);
	ms_seconds	= (__int64)(tim.wSecond * 1000);
	ms_milliseconds	= (__int64)tim.wMilliseconds;
	ms_total = ms_day + ms_hour + ms_minute + ms_seconds +
ms_milliseconds;
	tmReturn.int64 = ms_total;
	return tmReturn;
}
```

```
// JulianDay computes the number of days since Jan 1, 1900.
// This function is valid for dates from 1-Jan-1900 to 1-Jan-2100.
// 1-Jan-1900 = 0
int JulianDay( int yr, int mm, int dd )
{
	// MonthArray contains cumulative days for months in a non
leap-year
	int MonthArray[12] = { 0, 31, 59, 90, 120, 151, 181, 212,
243, 273, 304, 334};
	int j1, j2;

	// compute day of year (j1)
	j1 = MonthArray[mm-1] + dd - 1;
	// adjust day of year if this is a leap year and it is after
February
	if ((yr % 4)==0 && (yr != 1900) && (mm > 2))
		j1++;
	// compute number of days from 1/1/1900 to beginning of
present year
	j2 = (yr-1900)*365 + (yr-1901)/4;
	return j1+j2;

}

// Breaks up the Julian Time into it's sub-components
void __stdcall SMTime_JulianToTime( CURRENCY CurJulian,
int* yr, int* mm, int* dd,
				int *hh, int *mi, int *ss, int *ms )
{
	int julianDay, msLeft;
	JULIAN_TIME		julianTS = CurJulian.int64;

	*ms = julianTS % 1000;

	julianTS /= 1000;

	julianDay = (int)(julianTS / ( 60 * 60 * 24 ));

	JulianToCalendar(julianDay, yr, mm, dd );

	msLeft = (int)(julianTS - (julianDay * (__int64)( 60 * 60 *
24 )));

	*hh = msLeft / (60 * 60);
	msLeft = msLeft - *hh * 3600;
	*mi = msLeft / (60);
	*ss = msLeft % 60;
}
// JulianToCalendar converts a day index (from the JulianDay
function) to
// its corresponding calendar value (mm/dd/yr).  The valid range
for days
// is { 0 .. 73049 } for dates from 1-Jan-1900 to 1-Jan-2100.
void JulianToCalendar( int day, int* yr, int* mm, int* dd )
{
	int y, m, d;
	// month array contains days of months for months in a non
leap-year
	int month[12] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30,
31 };
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
	// compute year from days
	if (day < 365)
	y = 1900;
	else
	y = 1901 + ((day-365)/1461)*4 + (4*((day-
365)%1461)+3)/1461;
	// adjust February if this year is a leap year
	if ((y % 4)==0 && (y != 1900))
		month[1] = 29;
	else
		month[1] = 28;

	d = day - JulianDay( y, 1, 1 ) + 1;
	m = 1;

	while (d > month[m-1])
	{
		d = d - month[m-1];
		m++;
	}

	*yr = y;
	*mm = m;
	*dd = d;
}//  FILE:	SMTimer.h
//		Microsoft TPC-H Kit Ver. 1.00
//		Copyright Microsoft, 1999
//		All Rights Reserved
//
//
//  PURPOSE:	Declaration of the CSMTimer
//  Contact:	Reshma Tharamal (reshmat@microsoft.com)
//
// SMTimer.h : Declaration of the CSMTimer

#ifndef __SMTIMER_H_
#define __SMTIMER_H_

#include "resource.h"	// main symbols

#include "assert.h"

#define  MAX_JULIAN_TIME  0x7FFFFFFFFFFFFFFF
#define JULIAN_TIME	__int64
#define  TC_TIME		DWORD

#ifdef SMTIMER
#define DLL_LINK	__declspec( dllexport )
#else
#define DLL_LINK	__declspec( dllimport )
#endif

#ifdef __cplusplus
extern "C"
{
#endif
//DLL_LINKCURRENCY	__stdcall
SMTime_Get64BitTime(LPSYSTEMTIME lpInitTime);
int		JulianDay( int yr, int mm, int dd );
void	JulianToCalendar( int day, int* yr, int* mm, int* dd );
#ifdef __cplusplus
}
#endif
```

```
/////////////////////////////////////////////////////////////////////////////
// CSMTimer
class ATL_NO_VTABLE CSMTimer :
	public CComObjectRootEx<CComSingleThreadModel>,
	public CComCoClass<CSMTimer, &CLSID_SMTimer>,
	public IDispatchImpl<ISMTimer, &IID_ISMTimer,
&LIBID_SMTIMELib>
{
public:
	CSMTimer()
	{
		LARGE_INTEGER	l;

		if ( !QueryPerformanceFrequency(&l) )
		{
			m_baseTC = (__int64)GetTickCount();
			m_bCountUnavailable = TRUE;
		}
		else
		{
			m_bCountUnavailable = FALSE;

			m_Timerfreq = (__int64)l.HighPart << 32 |
(__int64)l.LowPart;
			QueryPerformanceCounter(&l);
			m_baseTC = (__int64)l.HighPart << 32 |
(__int64)l.LowPart;
		}
		m_bInProcess = FALSE;
	}

DECLARE_REGISTRY_RESOURCEID(IDR_SMTIMER)

DECLARE_PROTECT_FINAL_CONSTRUCT()

BEGIN_COM_MAP(CSMTimer)
	COM_INTERFACE_ENTRY(ISMTimer)
	COM_INTERFACE_ENTRY(IDispatch)
//	COM_INTERFACE_ENTRY2(IDispatch, ISMTimer)
END_COM_MAP()

// ISMTimer
public:
	STDMETHOD(get_Running)(/*[out, retval]*/ BOOL
*pVal);
	STDMETHOD(Stop)(CURRENCY *pElapsedTime);
	STDMETHOD(Start)();
	virtual ~CSMTimer();

private:
	__int64		m_baseTC;
	__int64		m_Timerfreq;
	BOOL		m_bCountUnavailable;
	TC_TIME		m_lStartTime;
	BOOL		m_bInProcess;

	TC_TIME		MyTickCount(void);
};
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
#endif //__SMTIMER_H_
// stdafx.cpp : source file that includes just the standard includes
//  stdafx.pch will be the pre-compiled header
//  stdafx.obj will contain the pre-compiled type information

#include "stdafx.h"

#ifdef _ATL_STATIC_REGISTRY
#include <statreg.h>
#include <statreg.cpp>
#endif

#include <atlimpl.cpp>
//  FILE:      stdafx.h
//             Microsoft TPC-H Kit Ver. 1.00
//             Copyright Microsoft, 1999
//             All Rights Reserved
//
//
//  PURPOSE:    include file for standard system include files,
//      or project specific include files that are used frequently,
//      but are changed infrequently
//  Contact:    Reshma Tharamal (reshmat@microsoft.com)
//

#if
!defined(AFX_STDAFX_H__4CDF88F4_EE9C_4F29_8212_61
557F251BDD__INCLUDED_)
#define
AFX_STDAFX_H__4CDF88F4_EE9C_4F29_8212_61557F251
BDD__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#define STRICT
#ifndef _WIN32_WINNT
#define _WIN32_WINNT 0x0400
#endif
#define _ATL_APARTMENT_THREADED

#include <atlbase.h>
//You may derive a class from CComModule and use it if you
want to override
//something, but do not change the name of _Module
extern CComModule _Module;
#include <atlcom.h>

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations
immediately before the previous line.

#endif //
!defined(AFX_STDAFX_H__4CDF88F4_EE9C_4F29_8212_61
557F251BDD__INCLUDED)
```


© 2009 Hewlett-Packard Company. All rights reserved.

Semaphore.cpp

```
#define _WIN32_WINNT 0x0400
#include <windows.h>
#include <string.h>
#include <iostream.h>
#include <stdlib.h>
#include <stdio.h>
#include <assert.h>
void main(int argc, char **argv)
{
typedef enum { eUnknown, eWait, eSignal, eRelease, eWaitList, eWaitGroup } OPERATION;
OPERATION eOP = eUnknown;
int iCount;
int i;
HANDLE hSemaphore;
HANDLE *pHandles;
SYSTEMTIME Time;
if (argc < 3)
goto usage;
if (_stricmp(argv[1], "-wait") == 0)
eOP = eWait;
else if (_stricmp(argv[1], "-signal") == 0)
eOP = eSignal;
else if (_stricmp(argv[1], "-release") == 0)
eOP = eRelease;
else if (_stricmp(argv[1], "-waitlist") == 0)
eOP = eWaitList;
else if (_stricmp(argv[1], "-waitgroup") == 0)
eOP = eWaitGroup;
else goto usage;
if ((eOP == eWait) || (eOP == eRelease))
{
// argv[2] is the semaphore name
// if -count option specified, then there must be exactly 5 args
if ((argc == 5) && (_stricmp(argv[3], "-count") == 0))
{
iCount = atoi(argv[4]);
if (iCount < 1)
goto usage;
}
// check that
else if (argc != 3)
goto usage;
else
iCount = 1;
}
else if (eOP == eWaitGroup)
{
if ((argc != 5) || (_stricmp(argv[3], "-count") != 0))
goto usage;
iCount = atoi(argv[4]);
if (iCount < 1)
goto usage;
}
else
// eWaitList or eSignal
iCount = argc - 2;
if (eOP == eWait)
{
printf( "semaphore name = %s\n", argv[2] );
printf( "semaphore count = %d\n", iCount );
hSemaphore = CreateSemaphore( NULL, 0, 2000000000, argv[2] );
if (hSemaphore == NULL)
{
DWORD dwError = GetLastError();
cout << "*ERROR* CreateSemaphore returned " << dwError << endl;
exit(EXIT_FAILURE);
}
for (i=0; i<iCount; i++)
{
WaitForSingleObject( hSemaphore, INFINITE );
GetLocalTime( &Time );
printf( "%4.4d-%2.2d-%2.2d %2.2d:%2.2d:%2.2d - released \n",
Time.wYear, Time.wMonth, Time.wDay, Time.wHour, Time.wMinute, Time.wSecond );
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
}
CloseHandle( hSemaphore );
}
else if ((eOP == eWaitGroup) || (eOP == eWaitList))
{
char **szEventNames;
szEventNames = new char*[iCount];
char szTmp[128];
printf( "event-list =" );
for (i=0; i<iCount; i++)
{
if (eOP == eWaitGroup)
{
wsprintf( szTmp, "%s.%d", argv[2], i+1 );
szEventNames[i] = new char[strlen(szTmp)+1];
strcpy( szEventNames[i], szTmp );
}
else
{
szEventNames[i] = new char[strlen(argv[i+2])+1];
strcpy( szEventNames[i], argv[i+2] );
}
printf( " %s", szEventNames[i] );
}
printf( "\n" );
pHandles = new HANDLE[iCount-1];
for (i=0; i<iCount; i++)
{
pHandles[i] = CreateEvent( NULL, TRUE /* manual reset */, FALSE /* initially non-signaled */, szEventNames[i] );
if (pHandles[i] == NULL)
{
DWORD dwError = GetLastError();
cout << "*ERROR* CreateEvent returned " << dwError << endl;
exit(EXIT_FAILURE);
}
}
for (i=iCount; i>0;i--)
{
int idx = WaitForMultipleObjects( i, pHandles, FALSE /* wait for all */, INFINITE ) - WAIT_OBJECT_0;
GetLocalTime( &Time );
printf( "%4.4d-%2.2d-%2.2d %2.2d:%2.2d:%2.2d - signaled: %s \n",
Time.wYear, Time.wMonth, Time.wDay, Time.wHour, Time.wMinute, Time.wSecond, szEventNames[idx] );
HANDLE hTmp = pHandles[idx];
pHandles[idx] = pHandles[i-1];
pHandles[i-1] = hTmp;
char* szTmp = szEventNames[idx];
szEventNames[idx] = szEventNames[i-1];
szEventNames[i-1] = szTmp;
}
for (i=0; i<iCount; i++)
CloseHandle( pHandles[i] );
}
else if (eOP == eRelease)
{
hSemaphore = OpenSemaphore( SEMAPHORE_MODIFY_STATE, FALSE, argv[2] );
if (hSemaphore == NULL)
{
DWORD dwError = GetLastError();
cout << "*ERROR* OpenSemaphore returned " << dwError << endl;
exit(EXIT_FAILURE);
}
if (!ReleaseSemaphore( hSemaphore, iCount, NULL ))
{
DWORD dwError = GetLastError();
cout << "*ERROR* ReleaseSemaphore returned " << dwError << endl;
exit(EXIT_FAILURE);
}
CloseHandle( hSemaphore );
}
else if (eOP == eSignal)
{
for (i=0; i<iCount; i++)
{
HANDLE hHandle = OpenEvent( EVENT_MODIFY_STATE, FALSE, argv[i+2] );
if (hHandle == NULL)
```


© 2009 Hewlett-Packard Company. All rights reserved.

```
{
DWORD dwError = GetLastError();
cout << "*ERROR* OpenEvent returned " << dwError << endl;
exit(EXIT_FAILURE);
}
SetEvent( hHandle );
CloseHandle( hHandle );
}
}
exit(EXIT_SUCCESS);
// syntax was bad; show usage and quit
usage:
printf(
"Semaphore Utility - Ver. 1.2 - 26-Jul-99 \n"
"Copyright (C) Microsoft Corp 1999. All rights reserved.\n\n"
"usage: \n"
" semaphore { -wait | -release } <semaphore-name> [ -count <count> ] \n"
" semaphore { -waitlist | -signal } <event-list> \n"
" semaphore -waitgroup <event-prefix> -count <count>\n"
"\n"
" <semaphore-name> == alpha-numeric identifier \n"
" <count> == integer > 0; default value = 1 \n"
" <event-list> == { <event-name> ... } \n"
" <event-name> == alpha-numeric identifier \n"
" <event-prefix> == alpha-numeric identifier \n"
"\n"
"There are two modes to choose from: a semaphore or a list of events. \n"
"\n"
"Semaphore mode: \n"
"A semaphore is a single identifier with an associated count. Each time \n"
"the semaphore is released, the count is decremented by one (or the amount \n"
"specified). When the count reaches zero, the waiter completes. If there \n"
"are multiple waiters on the same semaphore, each release releases only \n"
"the number of waiters specified in count.\n"
"\n"
"List of Events: \n"
"A list of events (alpha-numeric tags) is specified for the waiter. The \n"
"waiter doesn't complete until all of the events have been signaled. A \n"
"given event may be signaled more than once. There are two ways to define \n"
"the list of events, either explicitly (-waitlist) by naming all of them or \n"
"implicitly (-waitgroup) with a prefix and a count. Using the -waitgroup \n"
"option, you provide an alpha-numeric tag which is used as the prefix for a \n"
"group of events. The event names are generated by concatenating the prefix \n"
"with \".<n>\", where <n> is 1 to the specified count. \n"
);
```


© 2009 Hewlett-Packard Company. All rights reserved.

Appendix G: Price Quotations

Microsoft Corporation
One Microsoft Way
Redmond, WA 98052-6399

Tel 425 882 8080
Fax 425 936 7329
http://www.microsoft.com/

Microsoft

July 14, 2009

HP
Mike Fitzner
One Microsoft Way
Redmond, WA 98052

Here is the information you requested regarding pricing for several Microsoft products to be used in conjunction with your TPC-H benchmark testing.

All pricing shown is in US Dollars ($).

Part Number	Description	Unit Price	Quantity	Price
810-07578	SQL Server 2008 Enterprise x64 Edition Server License with 25 CALs Discount Schedule: Open Program - Level C Unit Price reflects a 40% discount from the retail unit price of $13,969.	$8,318	1	$8,318
359-01912	SQL Server 2008 Client License Client Access License Discount Schedule: Open Program - No Level Unit Price reflects a 4% discount from the retail unit price of $163.	$156	35	$5,460
P72-03168	Windows Server 2008 Enterprise Edition (x64) Server License with 25 CALs Discount Schedule: Open Program - Level C Unit Price reflects a 42% discount from the retail unit price of $3,999.	$2,310	1	$2,310
N/A	Microsoft Problem Resolution Services Professional Support (1 Incident)	$245	1	$245

A list of Microsoft's resellers can be found at
http://www.microsoft.com/products/info/render.aspx?view=22&type=mnp&content=22/licensing

All products listed above are currently orderable and available.

Defect support is included in the purchase price. Additional support is available from Microsoft PSS on an incident by incident basis at $245 per call.

This quote is valid for the next 90 days.

Reference ID: PHmifi09070140000009853.

© 2009 Hewlett-Packard Company. All rights reserved.