UNISYS

TPC Benchmark™ H Full Disclosure Report

Unisys ES7000 Orion 130 Enterprise Server

using

Microsoft SQL Server 2000 Enterprise Edition 64-bit

on

Microsoft Windows .NET Datacenter Server 2003 64-bit

October 2002

Unisys Part Number 6889 6901-0000, Rev B

First Printing - October 28, 2002

Unisys 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. Unisys Corporation assumes no responsibility for any errors that may appear in this document.

The pricing information in this document is believed to reflect accurately the current prices as of the publication date. However, Unisys Corporation and Microsoft Corporation provide no warranty on the pricing information in this document.

Benchmark results are highly dependent upon workload, specific application requirements, and systems' design and implementation. Relative system performance will vary as a result of these and other factors. Therefore, 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, and therefore results obtained in other operating environments may vary significantly. Unisys Corporation and Microsoft Corporation do not warrant or represent that a user can or will achieve similar performance expressed in composite query-per-hour ratings. No warranty of system performance or price/performance is expressed or implied with this document.

Unisys assumes no responsibility for any errors that may appear in this document. Unisys reserves the right to make changes in specifications and other information contained in this document without prior notice, and the reader should in all cases consult Unisys to determine whether any such changes have been made.

Copyright © 2002 Unisys Corporation All rights reserved.

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 on the title page of each item reproduced.

Printed in USA, October 2002

The following terms used in this publication are trademarks of their respective companies:

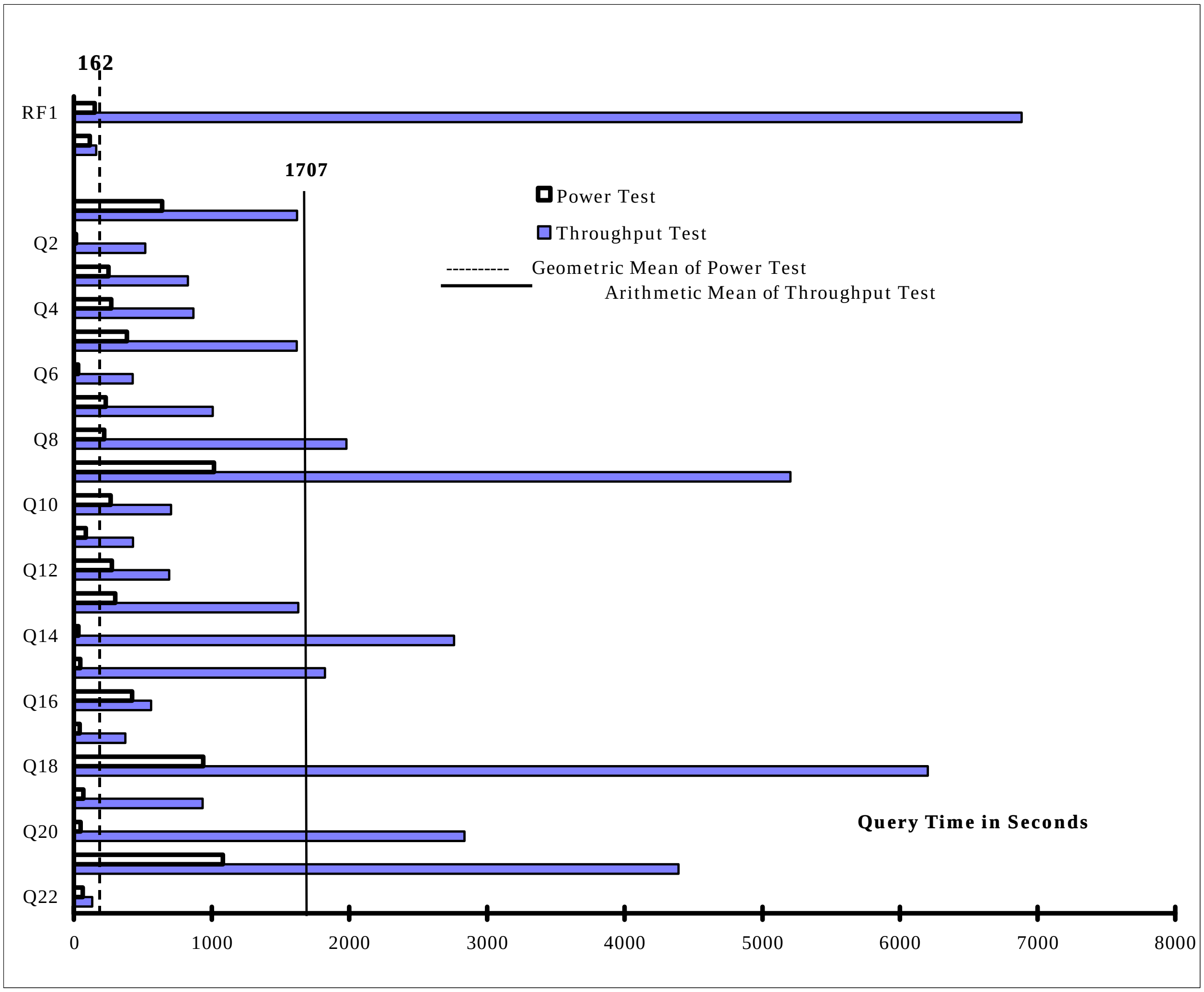
TPC Benchmark™	Trademark of the Transaction Processing Performance Council
TPC-H, QppH, QthH, and QphH	Trademark of the Transaction Processing Performance Council
Microsoft	Trademark of the Microsoft Corporation
SQL Server 2000	Trademark of the Microsoft Corporation
Windows 2000	Trademark of the Microsoft Corporation
Unisys	Trademark of the Unisys Corporation

Other product names used in this document may be trademarks and/or registered trademarks of their respective companies.

UNISYS	ES7000 Orion 130 Enterprise Server	TPC-H Rev. 2.0 Report Date 15 Oct 2002 Upgraded 28 Oct 2002

Total System Cost	Composite Query per Hour Rating	Price Performance
$988,328	4,774.3 / QphH @ 300GB	$207.02 / QphH @ 300GB

Database size	Database Manager	Operating System	Other Software	Availability Date
300 GB	Microsoft SQL Server 2000 Enterprise Edition 64-bit	Microsoft Windows .NET Datacenter Server 2003 64-bit	Windows 2000 Server w/ IIS 5.0 and COM+ Microsoft Visual C++	31 Mar 2003



Database Load Time = 8:30:00	Load included backup: Y	Total Data Storage / Database Size = 20.16
RAID (Base tables): N	RAID (Base tables and Auxilieary Data Structures): N	RAID (All): N

System Configuration		
Processors	16 x 1.0GHz Intel® Itanium2™ with 3MB Level 3 Cache	
Memory	64 GB Main Memory	
Disk Controllers	18	PCI Fibre Channel
	1	PCI SCSI
Disk Drives	358	18GB FC (16.4 GB useable)
	2	72GB FC (71.8 GB useable)
	1	36GB SCSI (34.8 GB useable)
Total Disk Storage	6049.6 GB	

UNISYS

ES7000 Orion 130 Enterprise Server

TPC-H Rev. 2.0

Report Date
15 Oct 2002
Upgraded 28 Oct 2002

Description	Part No.	Third Party Brand	Third Party Price	Unit Price	Qty.	Extended Price	3 yr.Maint. Price
Server Hardware:							
SYS: ES7000-130 Orion, 16x1GHz Procs, 32GB Mem	ES7004163-GS		1	$272,000	1	$272,000	$23,412
1x RAID Controller, 2x 36GB Disk (boot media)	Included						
Sentinel System Management S/W and Media	Included						
MEM: 4GB, 1 GB DIMMs	MEM41-4GB		1	$5,485	8	$43,880	
IO: Module, PCI-Adapter Enclosure	MOD3000-PCI		1	$1,723	2	$3,446	
CTRL: Fibre Channel HBA, 2-Port, 64-bit PCI	FCH720111-P64		1	$1,813	18	$32,634	
CTRL: 10/100Mbs, 1 Ch., PCI	ETH32112-PCI		1	$104	1	$104	
I/F: Monitor, 17-inch Color, Kybrd, Mse & Cable	ES70003-UIF		1	$544	1	$544	
				Server Subtotal		**$352,608**	**$23,412**
Storage Hardware:							
DISK: 18GB Drive, 15K FC, SCA	ESM18304-F44		1	$864	358	$309,312	Spared
DISK: 18GB Drive, 15K FC, SCA 10% spares*	ESM18304-F44		1	$864	36		$31,104
DISK: 73GB Drive, 10K FC, SCA	ESM73203-F94		1	$1,718	2	$3,436	Spared
DISK: 73GB Drive, 10K FC, SCA 10% spares*	ESM73203-F94		1	$1,718	2		$3,436
DAE: Enclsr, FC JBOD 2 LCC w/ 0 Disk	ESM702-JBD		1	$1,990	36	$71,640	$14,256
CBL: FC, 10 meter, DB9 Conn's, non-eql.	CBL135-10		1	$193	18	$3,474	
PWR: Distribution Strip, 9-Plug, 220V	SFR9-PWR		1	$295	16	$4,720	
CBL: Power, U.S. (Domestic), C20 - L6-20P	USE1936-LC6		1	$126	16	$2,016	
CAB: 36U x 19" x 34" Open Front Cabinet	HRT361934-OFT		1	$1,473	4	$5,892	
DOOR: 36U x 19", Rear	HRT3619-RDR		1	$368	4	$1,472	
PNL: 36U x 34" Side Skins, L&R HRT	HRT3634-SDS		1	$589	4	$2,356	
INSTL: Stablizer Foot	RM1936-FOT		1	$126	4	$504	
				Storage Subtotal		**$404,822**	**$48,796**
Server Software:							
O/S: Microsoft Windows .NET Datacenter & SQL Server	WNQ641616-LIT	2	1	$234,827	1	$234,827	$23,760
ACC: Microsoft Visual Studio Professional 6.0 Win32	659-00390	2	2	$1,079	1	$1,079	Inc. below
ACC: Microsoft Windows 2000 Server Resource Kit		2	2	$300	1	$300	Inc. below
SRVC: Microsoft 3-year Maintenance for SQL Server	PRO-PRORS-16U-01	2	2	$1,950	3		$5,850
				Software Subtotal		**$236,206**	**$29,610**
				Configuration Total		**$993,636**	**$101,818**
Comark Large Volume Discount						($102,680)	
Unisys Service Pre-Pay Discount							($4,446)

Notes:

1. 3rd Party Brand and Pricing: 1 = Comark supplied product and Unisys supplied Maintenance price, 2 = Microsoft supplied pricing
2. HW & SW maintenance figured at 24 x 7 w/ 4 hr. max. response time for spares.
3. * = 10% spare disks added in place of onsite service.
4. This reflects SQL per processor pricing, which negates the need to price SQL CALs.

Three Year Cost of Ownership:	**$988,328**
QphH @ 300GB:	**4,774.3**
$ / QphH@300GB:	**$207.02**

Benchmark results and test methodology audited by Lorna Livingtree of Performance Metrics, Inc.

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 assumption 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 benchmarks specifications. If you find that the stated prices are not available according to these terms, please inform the TPC at pricing@tpc.org. Thank You.

Numerical Quantities Summary

Measurement Results

Scale Factor	300	
Total Data Storage / Database Size	20.16	
Start of Database Load	10/1/2002	22:01:17
End of Database Load	10/2/2002	5:13:38
Start of Database Backup	10/2/2002	5:13:47
End of Database Backup	10/2/2002	6:31:26
Database Load Time	8:30:00	
Query Streams for Throughput Test	6	
TPC-H Power	6768.6	
TPC-H Throughput	3367.6	
Composite Query per Hour Rating(QphH@100GB)	4774.3	
Total System Price Over 5 Years	$988,328	
TPC-H Price Performance Metric	$207.02	

Measurement Intervals

Measurement Interval in Throughput Test (Ts)	42333.2	seconds

Duration of Stream Execution:

		Query Start Date/Time	RF1 Start Date/Time	RF2 Start Date/Time	
	Seed	Query End Date/Time	RF1 End Date/Time	RF2 End Date/Time	Duration
Stream 0	1002051338	10/2/02 21:14:06 10/2/02 23:06:27	10/2/02 21:11:26 10/2/02 21:13:56	10/2/02 23:06:28 10/2/02 23:08:23	1:52:20
Stream 1	1002051339	10/2/02 23:08:27 10/3/02 10:17:03	10/2/02 23:08:26 10/3/02 10:19:58	10/3/02 10:20:03 10/3/02 10:22:19	11:08:36
Stream 2	1002051340	10/2/02 23:08:28 10/3/02 9:20:17	10/3/02 10:22:23 10/3/02 10:25:46	10/3/02 10:25:52 10/3/02 10:28:32	10:11:49
Stream 3	1002051341	10/2/02 23:08:29 10/3/02 8:50:43	10/3/02 10:28:36 10/3/02 10:31:57	10/3/02 10:32:04 10/3/02 10:34:48	9:42:14
Stream 4	1002051342	10/2/02 23:08:30 10/3/02 9:41:04	10/3/02 10:34:52 10/3/02 10:38:12	10/3/02 10:38:19 10/3/02 10:41:05	10:32:34
Stream 5	1002051343	10/2/02 23:08:32 10/3/02 9:24:19	10/3/02 10:41:10 10/3/02 10:44:44	10/3/02 10:44:48 10/3/02 10:47:36	10:15:47
Stream 6	1002051344	10/2/02 23:08:38 10/3/02 9:52:38	10/3/02 10:47:41 10/3/02 10:51:03	10/3/02 10:51:10 10/3/02 10:53:59	10:43:59

TPC-H Timing Intervals (in seconds):

Query	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8
Stream 0	642.5	15.0	251.5	272.6	383.6	30.6	231.8	219.6
Stream 1	640.7	907.6	1021.2	255.7	1870.1	476.7	1346.6	2422.9
Stream 2	2117.6	403.8	739.4	994.1	1740.0	108.5	865.7	1527.8
Stream 3	1624.2	311.8	762.7	1002.1	1439.4	497.0	1302.5	2218.2
Stream 4	1837.6	812.7	610.7	1186.3	1775.2	484.8	553.0	1843.6
Stream 5	1662.7	418.4	876.2	822.3	1674.6	496.3	866.8	1176.9
Stream 6	1847.2	256.9	960.5	947.8	1212.8	497.7	1104.9	2688.2
Min Qi	640.7	256.9	610.7	255.7	1212.8	108.5	553.0	1176.9
Max Qi	2117.6	907.6	1021.2	1186.3	1870.1	497.7	1346.6	2688.2
Avg Qi	1621.7	518.5	828.5	868.1	1618.7	426.8	1006.6	1979.6
Query	**Q9**	**Q10**	**Q11**	**Q12**	**Q13**	**Q14**	**Q15**	**Q16**
Stream 0	1018.2	266.3	84.8	276.8	301.0	32.7	45.8	423.0
Stream 1	1961.5	848.4	329.9	857.5	1811.9	2120.2	1248.5	576.0
Stream 2	6126.0	666.5	647.6	605.7	1188.6	3639.7	1660.7	793.2
Stream 3	5190.0	589.2	465.5	670.8	1345.8	2466.7	2572.5	403.0
Stream 4	5955.4	600.1	367.3	705.9	1567.5	2213.2	2139.2	493.4
Stream 5	5302.6	857.9	443.3	721.7	1459.3	2565.4	2200.0	483.1
Stream 6	6693.2	667.9	316.7	582.7	2402.4	3562.7	1124.4	617.9
Min Qi	1961.5	589.2	316.7	582.7	1188.6	2120.2	1124.4	403.0
Max Qi	6693.2	857.9	647.6	857.5	2402.4	3639.7	2572.5	793.2
Avg Qi	5204.8	705.0	428.4	690.7	1629.3	2761.3	1824.2	561.1
Query	**Q17**	**Q18**	**Q19**	**Q20**	**Q21**	**Q22**	**RF1**	**RF2**
Stream 0	41.0	939.0	69.9	47.5	1082.1	64.9	150.6	114.5
Stream 1	135.2	6131.6	119.0	9268.8	5700.4	65.7	40291.4	135.8
Stream 2	608.4	8189.6	182.1	306.9	3450.3	146.7	202.8	160.1
Stream 3	386.5	5752.6	923.3	233.0	4587.6	190.1	200.7	163.8
Stream 4	392.5	6201.8	1282.6	1522.7	5284.4	123.8	199.6	166.4
Stream 5	674.0	6844.4	1332.2	205.3	5737.1	126.8	214.0	168.2
Stream 6	46.5	4100.2	1764.3	5496.1	1595.3	153.0	201.4	169.5
Min Qi	46.5	4100.2	119.0	205.3	1595.3	65.7	199.6	135.8
Max Qi	674.0	8189.6	1764.3	9268.8	5737.1	190.1	40291.4	169.5
Avg Qi	373.9	6203.4	933.9	2838.8	4392.5	134.4	6885.0	160.6

PERFORMANCE METRICS INC.
TPC Certified Auditors

October 15, 2002

Jerrold Buggert
Director of Modeling and Measurement
Unisys Corporation
25725 Jeronimo Road
Mission Viejo, CA 92691

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

Platform:	Unisys ES7000 Orion 130 Enterprise Server
Database Manager:	Microsoft SQL Server 2000 Enterprise Edition 64-bit
Operating System:	Microsoft Windows .NET Datacenter Server 2003 64-bit

CPU's	Memory	Total Disks	QppH@300GB	QthH@300GB	QphH@300GB
16 Itanium2 @ 1.0 Ghz	64 GB	358 @18 GB 2 @ 72 GB 1 @ 36 GB	**6,768.6**	**3,367.6**	**4,774.3**

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 300GB using DBGEN. The version of DBGEN was 1.3.0.
- The qualification database layout was identical to the tested database except for the 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 Microsoft's standard OSQL interactive interface. The version of QGEN was 1.3.0.
- 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.
- The performance metrics were correctly computed.
- The repeatability of the measurement was verified.
- The ACID properties were tested and verified.
- Sufficient mirrored log space was present on the tested system.
- 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

Microsoft Corporation
One Microsoft Way
Redmond, WA 98052-6399

Tel 425 882 8080
Fax 425 936 7329
http://www.microsoft.com/

Microsoft

October 11, 2002

Unisys Corporation
Bob Murphy
M/S 4683
PO Box 64942
St. Paul, MN 55164-0942

Mr. Murphy:

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
659-00390	**Visual Studio Professional 6.0 Win32** *No discounts applied*	$1,079	1	$1,079
N/A	**Windows 2000 Server Resource Kit** *No discounts applied*	$300	1	$300
PRO-PRORS-16U-01	**Database Server Support Package** *1 Year Term*	$1,950	5	$9,750

Some products may not be currently orderable but will be available through Microsoft's normal distribution channels by December 31, 2002.

This quote is valid for the next 90 days.

If we can be of any further assistance, please contact Jamie Reding at
(425) 703-0510 or jamiere@microsoft.com.

Reference ID: PHbomu0211103275
Please include this Reference ID in any correspondence regarding this price quote.

COMARK
TECHNOLOGY SERVICES
Advanced Systems Division

Quote # DM021008-UN009
Oct 8, 2002

To: Unisys Corporation TPC-H

Qty	Prod #	Description	Unit Price	Net Price
ES7000 Server				
1	ES7004163-GS	SYS: ES7000-130 Orion, 16x1GHz Procs, 32GB Mem	$272,000.00	**$272,000.00**
8	MEM41-4GB	MEM: 4GB, 1 GB DIMMs	$5,485.00	**$43,880.00**
2	MOD3000-PCI	IO: Module, PCI-Adapter Enclosure	$1,723.00	**$3,446.00**
18	FCH720111-P64	CTRL: Fibre Channel HBA, 2-Port, 64-bit PCI	$1,813.00	**$32,634.00**
1	ETH32112-PCI	CTRL: 10/100Mbs, 1 Ch., PCI	$104.00	**$104.00**
1	ES70003-UIF	I/F: Monitor, 17-inch Color, Kybrd, Mse & Cable	$544.00	**$544.00**
1	WNQ641616-LIT	O/S: Microsoft Windows .NET Datacenter & SQL Server	$234,827.00	**$234,827.00**
ESM700 Storage				
394	ESM18304-F44	DISK: 18GB Drive, 15K FC, SCA + 10% spares	$864.00	**$340,416.00**
4	ESM73203-F94	DISK: 73GB Drive, 10K FC, SCA + 10% spares	$1,718.00	**$6,872.00**
36	ESM702-JBD	DAE: Enclsr, FC JBOD 2 LCC w/ 0 Disk	$1,990.00	**$71,640.00**
18	CBL135-10	CBL: FC, 10 meter, DB9 Conn's, non-eql.	$193.00	**$3,474.00**
16	SFR9-PWR	PWR: Distribution Strip, 9-Plug, 220V	$295.00	**$4,720.00**
16	USE1936-LC6	CBL: Power, U.S. (Domestic), C20 - L6-20P	$126.00	**$2,016.00**
4	HRT361934-OFT	CAB: 36U x 19" x 34" Open Front Cabinet	$1,473.00	**$5,892.00**
4	HRT3619-RDR	DOOR: 36U x 19", Rear	$368.00	**$1,472.00**
4	HRT3634-SDS	PNL: 36U x 34" Side Skins, L&R HRT	$589.00	**$2,356.00**
4	RM1936-FOT	INSTL: Stablizer Foot	$126.00	**$504.00**
1		Large Volume Cash Discount	-$102,680.00	**-$102,680.00**

Prices may vary when items are purchased separately.

Disks come with a 5 year return-to-factory warranty,
7 day replenishment. Quote valid for 90 days.

TOTAL **$924,117.00**

PREFACE

Document Overview

This report documents the methodology and results of the TPC Benchmark™ H (TPC-H) test conducted on the Unisys ES7000 Orion 130 using Microsoft SQL Server 2000 Enterprise Edition 64-bit, in conformance with the requirements of the TPC Benchmark™H Standard Specification Revision 1.5.0. The tests documented in this report were sponsored by Unisys Corporation. The operating system used for the benchmark was Microsoft Windows .NET Datacenter Server 2003 64-bit.

The Transaction Processing Performance Council (TPC) developed the TPC-H Benchmark. The TPC Benchmark™ H Standard represents an effort by Unisys Corporation and other members of the Transaction Processing Performance Council (TPC) to create an industry-wide benchmark for evaluating the performance and price/performance of decision support systems, and to disseminate objective, verifiable performance data to the data processing industry.

A certified audit of these measurements and the reported results was performed by Lorna Livingtree of Performance Metrics Inc. (Folsom, CA). She has verified compliance with the relevant TPC Benchmark™ H specifications; audited the benchmark configuration, environment, and methodology used to produce and validate the test results; and audited the pricing model used to calculate the price/ performance. The auditor’s letter of attestation is attached to the Executive Summary and precedes this section.

TPC Benchmark™H Overview

The TPC Benchmark™H (TPC-H) is a decision support benchmark. It consists of a suite of business oriented ad-hoc queries and concurrent updates. The queries and the data populating the database have been chosen to have broad industry-wide relevance while maintaining a sufficient degree of ease of implementation. This benchmark illustrates decision support systems that:

- Examine large volumes of data;
- Execute queries with a high degree of complexity;
- Give answers to critical business questions.

TPC-H evaluates the performance of various decision support systems by the execution of sets of queries against a standard database under controlled conditions. The TPC-H queries:

- Give answers to real-world business questions;
- Simulate generated ad-hoc queries (e.g., via a point and click GUI interface);
- Are far more complex than most OLTP transactions;
- Include a rich breadth of operators and selectivity constraints;
- Generate intensive activity on the part of the database server component of the system under test;
- Are executed against a database complying to specific population and scaling requirements;
- Are implemented with constraints derived from staying closely synchronized with an on-line production database.

The TPC-H operations are modeled as follows:

- The database is continuously available 24 hours a day, 7 days a week, for ad-hoc queries from multiple end users and updates against all tables, except possibly during infrequent (e.g., once a month) maintenance sessions;
- The TPC-H database tracks, possibly with some delay, the state of the OLTP database through on-going updates which batch together a number of modifications impacting some part of the decision support database;
- Due to the world-wide nature of the business data stored in the TPC-H database, the queries and the updates may be executed against the database at any time, especially in relation to each other. In addition, this mix of queries and updates is subject to specific ACIDity requirements, since queries and updates may execute concurrently;
- To achieve the optimal compromise between performance and operational requirements the database administrator can set, once and for all, the locking levels and the concurrent scheduling rules for queries and updates.

The minimum database required to run the benchmark holds business data from 10,000 suppliers. It contains almost ten million rows representing a raw storage capacity of about 1 gigabyte. Compliant benchmark implementations may also use one of the larger permissible database populations (e.g., 300 gigabytes), as defined in Clause 4.1.3.

The performance metric reported by TPC-H is called the TPC-H Composite Query-per-Hour Performance Metric (QphH@Size), and reflects multiple aspects of the capability of the system to process queries. These aspects include the selected database size against which the queries are executed, the query processing power when queries are submitted by a single stream, and the query throughput when queries are submitted by multiple concurrent users. The TPC-H Price/Performance metric is expressed as $/QphH@Size. To be compliant with the TPC-H standard, all references to TPC-H results for a given configuration must include all required reporting components. *The TPC believes that comparisons of TPC-H results measured against different database sizes are misleading and discourages such comparisons.*

The TPC-H database must be implemented using a commercially available database management system (DBMS) and the queries executed via an interface using dynamic SQL. The specification provides for variants of SQL, as implementers are not required to have implemented a specific SQL standard in full. TPC-H uses terminology and metrics that are similar to other benchmarks, originated by the TPC and others. Such similarity in terminology does not in any way imply that TPC-H results are comparable to other benchmarks. The only benchmark results comparable to TPC-H are other TPC-H results compliant with the same revision.

Despite the fact that this benchmark offers a rich environment representative of many decision support systems, this benchmark does not reflect the entire range of decision support requirements. In addition, the extent to which a customer can achieve the results reported by a vendor is highly dependent on how closely TPC-H approximates the customer application. The relative performance of systems derived from this benchmark does not necessarily hold for other workloads or environments. Extrapolations to any other environment are not recommended.

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

Benchmark sponsors are permitted several possible system designs, provided that they adhere to the model described in Clause 6. A full disclosure report (FDR) of the implementation details, as specified in Clause 8, must be made available along with the reported results.

General Implementation Guidelines

The purpose of TPC benchmarks is to provide relevant, objective performance data to industry users. To achieve that purpose, TPC benchmark specifications require that benchmark tests be implemented with systems, products, technologies and pricing that:

- Are generally available to users;
- Are relevant to the market segment that the individual TPC benchmark models or represents (e.g. TPC-H models and represents complex, high data volume, decision support environments);
- Would plausibly be implemented by a significant number of users in the market segment the benchmark models or represents.

A Table of Contents follows after this page.

Related Product Information

The TPC Benchmark™ H Standard requires that test sponsors provide a Full Disclosure Report in addition to published results. You can obtain copies of the test results as well as additional copies of this full disclosure report by sending a request to the following address:

Unisys Corporation
TPC Benchmark Administrator, MS 4683
Systems Analysis Modeling & Measurement
PO Box 64942
Saint Paul, MN 55164-0942

End Table of Contents

[To omit Appendixes E and F, print only the first 78 pages.]

1. GENERAL ITEMS

1.1 Benchmark Sponsor

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

This TPC benchmark H was sponsored by Unisys Corporation. The benchmark test was developed by Microsoft and Unisys. The benchmark was conducted at Unisys, Roseville, Minnesota.

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 but not limited to:

- *Data Base 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.*

Comment 1: In the event that some parameters and options are set multiple times, it must be easily discernible by an interested reader when the parameter or option was modified and what new value it received each time.
Comment 2: This requirement can be satisfied by providing a full list of all parameters and options, as long as all those that have been modified from their default values have been clearly identified and these parameters and options are only set once.

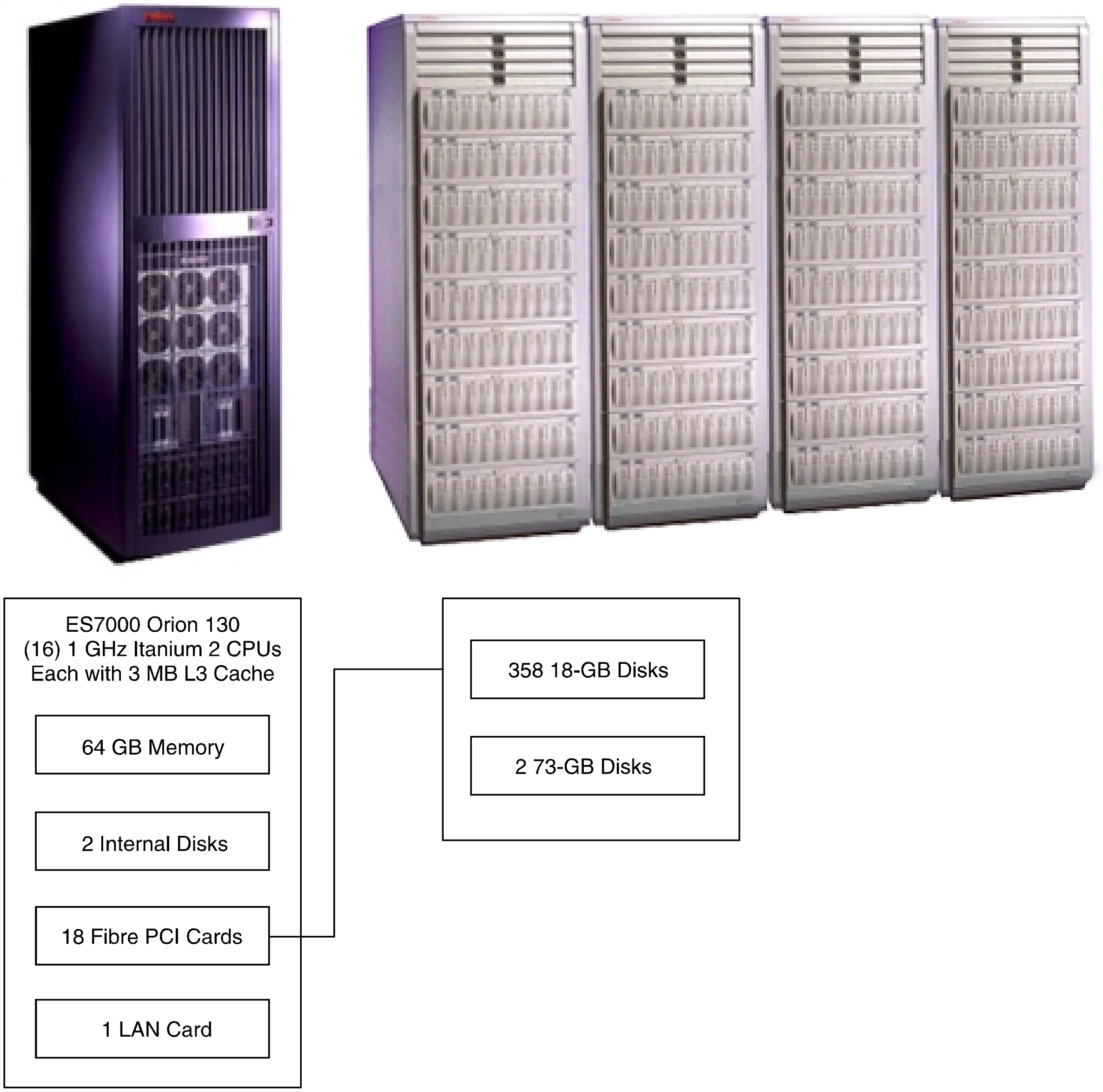
Details of system and database configurations and parameters are provided in Appendixes A and B.

1.3 Configuration Diagrams

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, work stations, terminals, etc., that were physically used in the test or are incorporated into the pricing structure;*
- *Type and run-time execution location of software components (e.g., DBMS, query processing tools/languages, middle-ware components, software drivers, etc.).*

The SUT and priced system are identical. The priced configuration is shown in the diagram that follows.



*The priced system does not include the rack mounts for the system console and keyboard.

See Section 5.2 for a more detailed summary of the configuration of the controllers and disks.

Figure 1.1 Benchmark and Priced Configuration for ES7000 Orion 130

2. CLAUSE 1: LOGICAL DATA BASE DESIGN

2.1 Table Definitions

Listings must be provided for all table definition statements and all other statements used to setup the test and qualification databases.

Appendix B contains the scripts that define, create, and analyze the tables and indexes for the TPC-H database.

2.2 Database Organization

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.

Clustered indexes were used. See Appendix B, which contains the database and table creation statements.

2.3. Horizontal Partitioning

Horizontal partitioning of base tables or auxiliary structures created by database directives is allowed. Groups of rows from a table or auxiliary structure may be assigned to different files, disks, or areas. If this assignment is a function of data in the table or auxiliary structure, the assignment must be based on the value of a partitioning field. A partitioning field must be one and only one of the following:

- *A primary*
- *A foreign*
- *A single date column*

Some partitioning schemes require the use of directives that specify explicit values for the partitioning field. If such directives are used they must satisfy the following conditions:

- *They may not rely on any knowledge of the data stored in the table except the minimum and maximum values of columns used for the partitioning field.*
- *Within the limitations of integer division, they must define each partition to accept an equal portion of the range between the minimum and maximum values of the partitioning column(s).*
- *The directives must allow the insertion of values of the partitioning column(s) outside the range covered by the minimum and maximum values.*

Multiple-level partitioning of base tables or auxiliary structures is allowed only if each level of partitioning satisfies the conditions stated above and each level references only one partitioning field as defined above. If implemented, the details of such partitioning must be disclosed.

Horizontal partitioning was not used. See Appendix B, which contains the database and table creation statements.

2.4 Vertical Partitioning

Vertical partitioning of tables is not allowed. For example, groups of columns of one row shall not be assigned to files, disks, or areas different from those storing the other columns of that row. The row must be processed as an atomic series of contiguous columns.

Comment*: The effect of vertical partitioning is to reduce the effective row size accessed by the system. Given the synthetic nature of this benchmark, the effect of vertical partitioning is achieved by the choice of row sizes. No further vertical partitioning of the data set is allowed. Specifically, the above Clause prohibits assigning one or more of the columns not accessed by the TPC-H query set to a vertical partition.*

Vertical partitioning was not used. See Appendix B, which contains the database and table creation statements.

2.5 Replication

Any replication of physical objects must be disclosed and must conform to the requirements of Clause1.5.6.

No replication was used. See Appendix B, which contains the database and table creation statements.

3. CLAUSE 2: QUERIES AND UPDATE FUNCTIONS

3.1 Query Language

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

SQL was the query language used to implement all queries.

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 1.3.0 and QGEN version 1.3.0 were used to generate random numbers for these runs.

3.3 Substitution Parameters

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 supplied QGEN version 1.3.0 was used.

3.4 Query Text and Output Data from Qualification 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 minor query modifications used in this implementation The following allowed minor query modifications were used in this implementation:

- 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

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.

3.6 Query 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 the isolation level “Level 1.”

3.7 Source Code of Refresh Functions

The details of how the refresh functions were implemented must be disclosed (including source code of any non-commercial program used).

The refresh function is part of the implementation-specific driver code included in Appendix E.

3.8 Database Maintenance Option

The details of the database maintenance option selected (i.e., reset or evolve) must be disclosed (including source code of any non-commercial program used).

This implementation of the TPC-H benchmark uses the reset option.

4. CLAUSE 3: DATABASE SYSTEM PROPERTIES

4.1 Atomicity

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.

4.1.1 Completed Transaction

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.

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 was committed.
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 Step 1. It was verified that the appropriate rows had been inserted.

4.1.2 Aborted Transaction

Perform the Acid transaction for a randomly selected set of input data, substituting 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.

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 Step 1. It was verified that the appropriate rows had not been changed.

4.2 Consistency

Consistency is the property of the application that requires any execution of transactions to take the database from one consistent state to another.

4.2.1 Consistency Test

Verify that ORDER and LINEITEM tables are initially consistent, submit the required number of Acid transactions with randomly selected input parameters, and re-verify the consistency of the ORDER and LINEITEM tables.

The consistency of the ORDER and LINEITEM tables was verified based on randomly selected values of the column O_ORDERKEY.

More than 100 Acid transactions were submitted from each of two execution streams.

1. The consistency of the ORDER and LINEITEM tables was re-verified.

4.3 Isolation

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 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.

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 completed and did not see the uncommitted changes made by the Acid transaction.

3. The Acid transaction was COMMITTED.

4.3.2 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.

1. An ACID transaction was started for a randomly selected O_KEY, L_KEY, and DELA. 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 the uncommitted changes made by the ACID transaction.

3. The ACID transaction was ROLLED BACK.

4. The ACID query completed.

4.3.3 Write-Write Conflict with Commit

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

1. An ACID transaction, T1, was started for a randomly selected O_KEY, L_KEY, and DELTA. The ACID transaction 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. T1 was allowed to COMMIT and T2 completed.

5. It was verified that T2.L_EXTENDEDPRICE was calculated correctly.
   T2.L_EXTENDEDPRICE = T1.L_EXTENDEDPRICE +(DELTA1*(T1.L_EXTENDEDPRICE/T1.L_QUANTITY))

4.3.4 Write-Write Conflict with Rollback

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

1. An Acid transaction, T1, was started for a randomly selected O_KEY, L_KEY, and DELTA. The Acid transaction was suspended prior to ROLLBACK.

2. Another Acid transaction, T2, was started using the same O_KEY and L_KEY and a different randomly selected DELA.

3. T2 waited

4. T1 was allowed to ROLLBACK and T2 completed

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

4.3.5 Concurrent Progress of Read and Write on Different Tables

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

1. An ACID Transaction, T1, was started for a randomly selected O_KEY, L_KEY, and DELTA. T1 was suspended prior to COMMIT.

2. Another ACID transaction, T2 was started using random values for PS_PARTKEY and PS_SUPPKEY.

3. ACID Transaction T2 completed.

4. ACID transaction T1 completed and the appropriate rows in the ORDER, LINEITEM, and HISTORY tables were changed.

4.3.6 Updates not Indefinitely Delayed by Reads on Same Table

Demonstrate that 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.

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 length 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. Verified that the appropriate rows in ORDER, LINEITEM and HISTORY tables have been changed.

4.4 Durability

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 Failure of a Durable Medium and System Crash

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 database logs were placed on software mirrored volumes.

The tables for the database were stored on raw partitions, on two drives of the same characteristics as the drives used for the test database, and the two drives were on one controller.

1. The datafiles were backed up to an alternate disk media.

2. Seven streams of ACID transactions were started.

3. After at least 100 transactions had occurred on each stream and the streams were still running when one side of the software mirrored set of logs was removed.

4. After it was determined that the test would still run with the loss of a log disk, and after running at least another 100 transactions on each stream, a data disk was removed.

5. The seven streams of ACID transactions failed and recorded their numbers of committed transactions in success files.

6. The database was brought down.

7. Two new drives were used to replace the removed log and data disks.

8. The datafiles were restored to their state prior to the ACID transaction streams.

9. The database ran through its recovery mode.

10. The counts in the success files and the HISTORY table count were compared and the counts matched.

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.

The system crash and memory failure tests were combined.

1. Seven streams of ACID transactions were started.

2. After at least 100 transactions had occurred on each stream, and the streams of ACID transactions were still running, the system was powered off.

3. When power was restored 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 they matched.

4.4.3 Memory Failure

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

The system crash and memory failure tests were combined. See the previous section.

5. CLAUSE 4: SCALING AND DATABASE POPULATION

5.1 Cardinality of Tables

The cardinality (e.g., 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	# of ROWS
Orders	450,000,000
Lineitem	1,799,989,091
Customer	45,000,000
Parts	60,000,000
Supplier	3,000,000
Partsupp	240,000,000
Nation	25
Region	5

5.2 Distribution of Tables and Logs Across Media

The distribution of tables and logs across all media must be explicitly described using a format similar to that shown in the following example for both the tested and priced systems.

The SUT had 360 external drives and 1 internal drive. The priced systems has 360 external drives and 2 internal drives.

Utilization of the drives. Test database components:

- 350 drives for the 300GB database. See Appendix F for exact disk configuration.
- Lineitem, General and Tempdb file groups, consisting of 350 logical single volumes each, mounted as junction points.
- 12 logical drives used for the backup devices of the 300GB database. Each logical drive consists of 5 disk partitions, formatted as software Raid-5. The database backup files were stored on the same physical drives as the database.
- The Tpch300g log was placed on 2 mirrored drives, 72GB each drive, using 45GB on each one of the drives.
- The operating system, Microsoft Windows .NET Datacenter Server 2003 64-bit, and Microsoft SQL Server 2000 Enterprise Edition 64-bit, as well as the operating system page file, were installed on one internal 18GB drive and 20 external drives.

		Disk Partition Description (See App. F)			
Cntrlr	**# Drives**	**Lineitem_FG**	**General_FG**	**Tempdb**	**Other allocation (*)**
1	20	2.34GB/drive	0.6GB/drive	1.46GB/drive	12.43GB non-db space/drive
2	20	2.34GB/drive	0.6GB/drive	1.46GB/drive	12.43GB non-db space/drive
3	20	2.34GB/drive	0.6GB/drive	1.46GB/drive	12.43GB non-db space/drive
4	20	2.34GB/drive	0.6GB/drive	1.46GB/drive	12.43GB non-db space/drive
5	20	2.34GB/drive	0.6GB/drive	1.46GB/drive	12.43GB non-db space/drive
6	20	2.34GB/drive	0.6GB/drive	1.46GB/drive	12.43GB non-db space/drive
7	20	2.34GB/drive	0.6GB/drive	1.46GB/drive	12.43GB non-db space/drive
8	20	2.34GB/drive	0.6GB/drive	1.46GB/drive	12.43GB non-db space/drive
9	20	2.34GB/drive	0.6GB/drive	1.46GB/drive	12.43GB non-db space/drive
10	20	2.34GB/drive	0.6GB/drive	1.46GB/drive	12.43GB non-db space/drive
11	20	2.34GB/drive	0.6GB/drive	1.46GB/drive	12.43GB non-db space/drive
12	20	2.34GB/drive	0.6GB/drive	1.46GB/drive	12.43GB non-db space/drive
13	20	2.34GB/drive	0.6GB/drive	1.46GB/drive	12.43GB non-db space/drive
14	20	2.34GB/drive	0.6GB/drive	1.46GB/drive	12.43GB non-db space/drive
15	20	2.34GB/drive	0.6GB/drive	1.46GB/drive	12.43GB non-db space/drive
16	20	2.34GB/drive	0.6GB/drive	1.46GB/drive	12.43GB non-db space/drive
17	18	2.34GB/drive	0.6GB/drive	1.46GB/drive	12.43GB non-db space/drive
	1	45GB db Log - Mirror			28GB non-db space/drive
	1				17GB non-db space/drive
18	12	2.34GB/drive	0.6GB/drive	1.46GB/drive	12.43GB non-db space/drive
	1	10GB Tempdb Log			7GB non-db space/drive
	1	45GB db Log - Mirror			28GB non-db space/drive
	6				17GB non-db space/drive
Internal	1	Operating System, Database Manager, Page File			

(*) 148 Dynamic disks and 212 Basic disks
12 Backup devices, using 5 drives per device, formatted as RAID-5
43 drives used for Flat Files
20 drives used for additional page file
2 drives, mirrored, used for mount/junction points
4 full drives used for ACID tests & 2 drives used for Acid db backup

5.3 Partitions/Replications Mapping

The mapping of data base partitions/replications must be explicitly described.

***Comment**: The intent is to provide sufficient detail about partitioning and replication to allow independent reconstruction of the test database.*

Database partitioning and replication were not used.

5.4 Use of RAID

Implementations may use some form of RAID . The RAID level used must be disclosed for each device.

No hardware RAID was used in the implementation. The log file for the qualification and test database was stored on a drive, which was software mirrored.

5.5 DBGEN Modifications

The version number, release number, modification number, and patch level of DBGEN must be disclosed. Any modifications to the DBGEN 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.

The supplied DBGEN 1.3.0 was used for populating the database.

5.6 Database Load Time

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

The Numerical Quantities summary (pp. v) contains the database load time, which was 8:30:09.

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. The ratio must be reported to the nearest 1/100th, rounded up. For example, a system configured with 96 disks of 2.1 GB capacity for a 100GB test database has a data storage ratio of 2.02.

***Comment**: For the reporting of configured disk capacity, gigabyte (GB) is defined to be 2^30 bytes. Since disk manufacturers typically report disk size using base ten (i.e., GB = 10^9), it may be necessary to convert the advertised size from base ten to base two.*

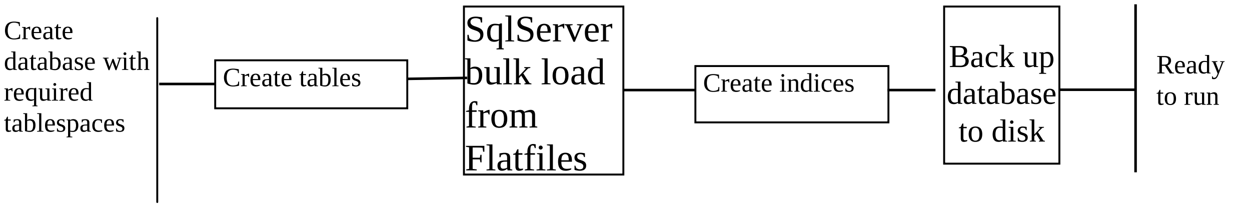
The Numerical Quantities summary (pp. v) contains the data storage ratio (20.64) for the system used.

5.8 Database Loading

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 following steps were used to load the database:

1) DBGEN version 1.3.0 was used to create flat files.
2) SQL Server 2000 was used to define the database, to define tables, and to load the tables via a “bulk insert” command.
3) Clustered indexes were created using SQL Server 2000.
4) Non clustered indexes were created using SQL Server 2000.
5) A database backup was performed to 12 logical devices
6) Rows were inserted into the database by running 16 concurrent threads, each of which performed a “bulk insert” operation that loaded one sixteenth of each of the LINEITEM, ORDERS, PART, PARTSUPP, SUPPLIER and CUSTOMER tables. The NATION and REGION tables were loaded sequentially, each by a single thread.


Create database with required tablespaces
Create tables
SqlServer bulk load from Flatfiles
Create indices
Back up database to disk
Ready to run

5.9 Qualification Database Configuration

Any differences between the configuration of the qualification database and the test database must be disclosed. .

The qualification database was created using scripts identical to those of the test database, except for variances due to the sizes of the two databases.

6. Clause 5: Performance Metrics and Execution Rules

6.1 System Activity Between Load and Performance Tests

Any system activity on the SUT which takes place between the conclusion of the load test and the beginning of the performance test must be fully disclosed including listings of scripts or command logs.

Auditor requested queries were run against the database to verify the completeness and correctness of the database load.

6.2 Power Test Implementation

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 followed to run the power test.

1. SQL Server 2000 was started.
2. RF1 refresh transactions were run
3. Stream 00 execution was run
4. RF2 refresh transactions were run.

6.3 Timing Intervals and Reporting

The timing intervals for each query and for both refresh functions must be reported for the power test.

This information is contained in the Numerical Quantities Summary page in the Executive Summary at the beginning of this report. For convenience, it is repeated in Section 6.10.

6.4 Number of Streams in the Throughput Test

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

Six streams were run for the throughput test

6.5 Start and End Date/Time for Each Query Stream

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

This information is contained in the Numerical Quantities Summary page in the Executive Summary at the beginning of this report. For convenience, it is repeated in Section 6.10.

6.6 Total Elapsed Time for the Measurement Interval

The total elapsed time of the measurement interval must be reported for the throughput test.

This information is contained in the Numerical Quantities Summary page in the Executive Summary at the beginning of this report. For convenience, it is repeated in Section 6.10.

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

The start time and finish time for each refresh function in the refresh stream must be reported for the throughput test.

This information is contained in the Numerical Quantities Summary page in the Executive Summary at the beginning of this report. For convenience, it is repeated in Section 6.10.

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

The timing intervals for each query of each stream and for each refresh function must be reported for the throughput test.

This information is contained in the Numerical Quantities Summary page in the Executive Summary at the beginning of this report. For convenience, it is repeated in Section 6.10.

6.9 Performance Metrics

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

This information is contained in the Numerical Quantities Summary section of the Executive Summary (p. v in front). For convenience, it is repeated in Section 6.10.

6.10 The Performance Metric and Numerical Quantities from Both Runs

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

	QppH@300GB	QthH@300GB	QphH@300GB
Run 1	6852.7	3422.3	4842.7
Run 2	6768.6	3367.6	4774.3
% Difference	-1.2%	-1.6%	-1.4%

(Run 2 was reported.)

Tables from Numerical Quantities pages in the front of this report:

Numerical Quantities Summary

Ieasurement Results

Scale Factor	300	
Total Data Storage / Database Size	20.16	
Start of Database Load	10/1/2002	22:01:17
End of Database Load	10/2/2002	5:13:38
Start of Database Backup	10/2/2002	5:13:47
End of Database Backup	10/2/2002	6:31:26
Database Load Time	8:30:00	
Query Streams for Throughput Test	6	
TPC-H Power	6768.6	
TPC-H Throughput	3367.6	
Composite Query per Hour Rating(QphH@100GB)	4774.3	
Total System Price Over 5 Years	$988,328	
TPC-H Price Performance Metric	$207.02	

Ieasurement Intervals

Measurement Interval in Throughput Test (Ts)	42333.2	seconds

uration of Stream Execution:

	Seed	Query Start Date/Time Query End Date/Time	RF1 Start Date/Time RF1 End Date/Time	RF2 Start Date/Time RF2 End Date/Time	Duration
tream 0	1002051338	10/2/02 21:14:06 10/2/02 23:06:27	10/2/02 21:11:26 10/2/02 21:13:56	10/2/02 23:06:28 10/2/02 23:08:23	1:52:20
tream 1	1002051339	10/2/02 23:08:27 10/3/02 10:17:03	10/2/02 23:08:26 10/3/02 10:19:58	10/3/02 10:20:03 10/3/02 10:22:19	11:08:36
tream 2	1002051340	10/2/02 23:08:28 10/3/02 9:20:17	10/3/02 10:22:23 10/3/02 10:25:46	10/3/02 10:25:52 10/3/02 10:28:32	10:11:49
tream 3	1002051341	10/2/02 23:08:29 10/3/02 8:50:43	10/3/02 10:28:36 10/3/02 10:31:57	10/3/02 10:32:04 10/3/02 10:34:48	9:42:14
tream 4	1002051342	10/2/02 23:08:30 10/3/02 9:41:04	10/3/02 10:34:52 10/3/02 10:38:12	10/3/02 10:38:19 10/3/02 10:41:05	10:32:34
tream 5	1002051343	10/2/02 23:08:32 10/3/02 9:24:19	10/3/02 10:41:10 10/3/02 10:44:44	10/3/02 10:44:48 10/3/02 10:47:36	10:15:47
tream 6	1002051344	10/2/02 23:08:38 10/3/02 9:52:38	10/3/02 10:47:41 10/3/02 10:51:03	10/3/02 10:51:10 10/3/02 10:53:59	10:43:59

TPC-H Timing Intervals (in seconds):

Query	**Q1**	**Q2**	**Q3**	**Q4**	**Q5**	**Q6**	**Q7**	**Q8**
Stream 0	642.5	15.0	251.5	272.6	383.6	30.6	231.8	219.6
Stream 1	640.7	907.6	1021.2	255.7	1870.1	476.7	1346.6	2422.9
Stream 2	2117.6	403.8	739.4	994.1	1740.0	108.5	865.7	1527.8
Stream 3	1624.2	311.8	762.7	1002.1	1439.4	497.0	1302.5	2218.2
Stream 4	1837.6	812.7	610.7	1186.3	1775.2	484.8	553.0	1843.6
Stream 5	1662.7	418.4	876.2	822.3	1674.6	496.3	866.8	1176.9
Stream 6	1847.2	256.9	960.5	947.8	1212.8	497.7	1104.9	2688.2
Min Qi	640.7	256.9	610.7	255.7	1212.8	108.5	553.0	1176.9
Max Qi	2117.6	907.6	1021.2	1186.3	1870.1	497.7	1346.6	2688.2
Avg Qi	1621.7	518.5	828.5	868.1	1618.7	426.8	1006.6	1979.6
Query	**Q9**	**Q10**	**Q11**	**Q12**	**Q13**	**Q14**	**Q15**	**Q16**
Stream 0	1018.2	266.3	84.8	276.8	301.0	32.7	45.8	423.0
Stream 1	1961.5	848.4	329.9	857.5	1811.9	2120.2	1248.5	576.0
Stream 2	6126.0	666.5	647.6	605.7	1188.6	3639.7	1660.7	793.2
Stream 3	5190.0	589.2	465.5	670.8	1345.8	2466.7	2572.5	403.0
Stream 4	5955.4	600.1	367.3	705.9	1567.5	2213.2	2139.2	493.4
Stream 5	5302.6	857.9	443.3	721.7	1459.3	2565.4	2200.0	483.1
Stream 6	6693.2	667.9	316.7	582.7	2402.4	3562.7	1124.4	617.9
Min Qi	1961.5	589.2	316.7	582.7	1188.6	2120.2	1124.4	403.0
Max Qi	6693.2	857.9	647.6	857.5	2402.4	3639.7	2572.5	793.2
Avg Qi	5204.8	705.0	428.4	690.7	1629.3	2761.3	1824.2	561.1
Query	**Q17**	**Q18**	**Q19**	**Q20**	**Q21**	**Q22**	**RF1**	**RF2**
Stream 0	41.0	939.0	69.9	47.5	1082.1	64.9	150.6	114.5
Stream 1	135.2	6131.6	119.0	9268.8	5700.4	65.7	40291.4	135.8
Stream 2	608.4	8189.6	182.1	306.9	3450.3	146.7	202.8	160.1
Stream 3	386.5	5752.6	923.3	233.0	4587.6	190.1	200.7	163.8
Stream 4	392.5	6201.8	1282.6	1522.7	5284.4	123.8	199.6	166.4
Stream 5	674.0	6844.4	1332.2	205.3	5737.1	126.8	214.0	168.2
Stream 6	46.5	4100.2	1764.3	5496.1	1595.3	153.0	201.4	169.5
Min Qi	46.5	4100.2	119.0	205.3	1595.3	65.7	199.6	135.8
Max Qi	674.0	8189.6	1764.3	9268.8	5737.1	190.1	40291.4	169.5
Avg Qi	373.9	6203.4	933.9	2838.8	4392.5	134.4	6885.0	160.6

6.11 System Activity Between Tests

Any activity on the SUT that takes place between the conclusion of Run1 and the beginning of Run2 must be fully disclosed including listings of scripts or command logs along with any system reboots or database restarts.

The following activities took place between the conclusion of Run 1 and the beginning of Run 2:

1) Shutdown Sql Server
2) Restarted Sql Server

7. Clause 6: SUT and Driver Implementation Related Items

7.1 Driver

A detailed textual description of how the driver performs its functions, how its various components interact and any product functionalities or environmental setting on which it relies must be provided. All related source code, scripts and configuration files must be disclosed. The information provided should be sufficient for an independent reconstruction of the driver.

The TPC-H benchmark was implemented using a Microsoft internal tool called StepMaster. StepMaster is a general purpose test harness 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 for post-processing.

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 for post-processing.

Two types of ODBC connections are supported: static and dynamic. 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.

SQL Server operations executed from StepMaster do not gain any performance advantage compared to osql, the command prompt utility for ad hoc, interactive execution of Transact-SQL statements and scripts. Rather, StepMaster simplifies the task of benchmark execution, event timing, and reporting. This was confirmed during the audit.

7.2 Implementation-Specific Layer (ISL)

If an implementation specific layer is used, then a detailed description of how it performs its functions, how its various components interact and any product functionalities or environmental setting on which it relies must be provided. All related source code, scripts and configuration files must be disclosed. The information provided should be sufficient for an independent reconstruction of the implementation specific layer.

StepMaster program is used to control and track the execution of queries, via commands stored in an external Microsoft Access database. The source of this program is contained in Appendix E. The following steps are performed, to accomplish the Power and Throughput Runs:

1. Power Run
Execute 32 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.

- Execute the Stream 0 queries, in the prescribed order.
- Execute 32 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.

2. Throughput Run

- Execute six concurrent query streams. Each stream executes queries in the prescribed order for the appropriate Stream Id (01-06). Upon completion of each stream, a semaphore is set to indication completion.

- Execute six 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 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.

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 *iv* in the Executive Summary at the front of this report. The source for all prices is indicated. The hardware is available October 11, 2002. See page x for the quote from Comark for the hardware used. The pricing and availability of the Microsoft software used is given in a quote from Microsoft, which is included in this report on page ix of this report.

8.2 Five-Year Cost of System Configuration

The total 5-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 iv 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.

Summary by category from the measured and priced configuration:

Category	**Available**
Server Hardware	Now
Storage	Now
Server Software	03/31/03

9. Clause 8: Audit Related Items

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.

Lorna Livingtree of Performance, a certified TPC-H auditor, audited this benchmark

Lorna Livingtree
Performance Metrics Inc.
137 Yankton St., Suite 101
Folsom, CA 95630
(916) 985-1131 Fax: 916-985-1185

See pages vii-viii in the front of this paper for a copy of the auditor's attestation letter.

Further information regarding the audit process may be obtained from Ms. Livingtree.

APPENDIX A: System and Database Tunable Parameters

Software levels:

```
Microsoft Windows .NET Datacenter Server 2003 64-bit build 3663
Microsoft SQL Server 2000 Enterprise Edition 64-bit build 724
```

System Information:

```
OS Name     Microsoft® Windows® .NET Datacenter Server
Version     5.2.3663  Build 3663
OS Manufacturer   Microsoft Corporation
System Name SAMC06
System Manufacturer     Intel
System Model      870_SMP
System Type Itanium (TM) -based System
Processor   ia64 Family 31 Model 0 Stepping 6 GenuineIntel ~999 Mhz
Processor   ia64 Family 31 Model 0 Stepping 6 GenuineIntel ~999 Mhz
Processor   ia64 Family 31 Model 0 Stepping 6 GenuineIntel ~999 Mhz
Processor   ia64 Family 31 Model 0 Stepping 6 GenuineIntel ~999 Mhz
Processor   ia64 Family 31 Model 0 Stepping 6 GenuineIntel ~999 Mhz
Processor   ia64 Family 31 Model 0 Stepping 6 GenuineIntel ~999 Mhz
Processor   ia64 Family 31 Model 0 Stepping 6 GenuineIntel ~999 Mhz
Processor   ia64 Family 31 Model 0 Stepping 6 GenuineIntel ~999 Mhz
Processor   ia64 Family 31 Model 0 Stepping 6 GenuineIntel ~999 Mhz
Processor   ia64 Family 31 Model 0 Stepping 6 GenuineIntel ~999 Mhz
Processor   ia64 Family 31 Model 0 Stepping 6 GenuineIntel ~999 Mhz
Processor   ia64 Family 31 Model 0 Stepping 6 GenuineIntel ~999 Mhz
Processor   ia64 Family 31 Model 0 Stepping 6 GenuineIntel ~999 Mhz
Processor   ia64 Family 31 Model 0 Stepping 6 GenuineIntel ~999 Mhz
Processor   ia64 Family 31 Model 0 Stepping 6 GenuineIntel ~999 Mhz
Processor   ia64 Family 31 Model 0 Stepping 6 GenuineIntel ~999 Mhz
BIOS Version/Date Phoenix Technologies LTD BIOS Release 1.2.257, 9/10/2002
SMBIOS Version    2.3
Windows Directory C:\WINDOWS
System Directory  C:\WINDOWS\system32
Boot Device \Device\HarddiskVolume1
Locale      United States
Hardware Abstraction Layer    Version = "5.2.3663.0 (main.020715-1506)"
User Name   SAMC06\samm
Time Zone   Central Daylight Time
Total Physical Memory   65,536.00 MB
Available Physical Memory     161.18 MB
Total Virtual Memory    187.43 GB
Available Virtual Memory      60.68 GB
Page File Space   123.42 GB
Page File   C:\pagefile.sys
```

SQL Server 2000 Enterprise Edition 64-bit Installation

Microsoft SQL Server 2000 Enterprise Edition 64-bit was installed on the SUT. All default options were selected during the install except:

- “Custom installation” was selected. The SQL Server Development tools were not installed.

- Latin1_General binary sort order was used. (Collation Settings > Collation Designator > Latin1_General Binary)

- Services Accounts > Customized > SQL Server > Use Local System Account
Authentication Mode > Mixed > Blank Password allowed

SQL Server 2000 Enterprise Edition 64-bit Startup Parameters

SQLSERVR –c –x –g100 -E
Where:

- -c Start SQL Server independently of the Windows Service Control Manager
- -x Disable the keeping of CPU time and cache-hit ratio statistics
- -g Reserve 100MB for non-buffer pool allocation
- -E increase the number of consecutive extents allocated per file to 4

SQL Server 2000 Enterprise Edition 64-bit Parameter Settings:

```
name                                minimum     maximum     config_value run_value
----------------------------------- ----------- ----------- ------------ -----------
affinity mask                       -2147483648 2147483647  65535        65535
affinity64 mask                     -2147483648 2147483647  0            0
allow updates                       0           1           1            1
awe enabled                         0           1           0            0
c2 audit mode                       0           1           0            0
cost threshold for parallelism      0           32767       0            0
cursor threshold                    -1          2147483647  -1           -1
default full-text language          0           2147483647  1033         1033
default language                    0           9999        0            0
fill factor (%)                     0           100         0            0
index create memory (KB)            704         2147483647  0            0
lightweight pooling                 0           1           1            1
locks                               5000        2147483647  0            0
max degree of parallelism           0           32          16           16
max server memory (MB)              4           2147483647  62000        62000
max text repl size (B)              0           2147483647  65536        65536
max worker threads                  32          32767       355          355
media retention                     0           365         0            0
min memory per query (KB)           512         2147483647  512          512
min server memory (MB)              0           2147483647  58000        58000
nested triggers                     0           1           1            1
network packet size (B)             512         65536       32767        32767
open objects                        0           2147483647  0            0
priority boost                      0           1           0            0
query governor cost limit           0           2147483647  0            0
query wait (s)                      -1          2147483647  2147483647   2147483647
recovery interval (min)             0           32767       32767        32767
remote access                       0           1           1            1
remote login timeout (s)            0           2147483647  20           20
remote proc trans                   0           1           0            0
remote query timeout (s)            0           2147483647  600          600
scan for startup procs              0           1           0            0
set working set size                0           1           0            0
show advanced options               0           1           1            1
```

```
two digit year cutoff               1753        9999        2049         2049
user connections                    0           32767       0            0
user options                        0           32767       0            0
```

APPENDIX B: Database, Tables, and Indexes Creation

Create database

```
--  CreateDatabase
--  for use with StepMaster
--  Uses FileGroups

use master


--          Create temporary table for timing in the Master Database

if exists ( select name from sysobjects where name = 'tpch_temp_timer' )
            drop table tpch_temp_timer


create table tpch_temp_timer
(
            load_start_time                                         datetime
)

--          store the starting time in the temporary table

insert      into tpch_temp_timer values (getdate())


--
--          Drop the existing database
--

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

CREATE DATABASE tpch300g
ON PRIMARY
(           NAME                = tpch300g_root,
            FILENAME            = "C:\tpch300g_root.mdf",
            SIZE                = 7MB,
            FILEGROWTH          = 0),

FILEGROUP           LINEITEM_FG

  (NAME=lineitem_0,FILENAME='M:\2_J\LI\0\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_20,FILENAME='M:\2_J\LI\20\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_40,FILENAME='M:\2_J\LI\40\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_60,FILENAME='M:\2_J\LI\60\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_80,FILENAME='M:\2_J\LI\80\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_100,FILENAME='M:\2_J\LI\100\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_120,FILENAME='M:\2_J\LI\120\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_140,FILENAME='M:\2_J\LI\140\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_160,FILENAME='M:\2_J\LI\160\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_180,FILENAME='M:\2_J\LI\180\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_203,FILENAME='M:\2_J\LI\203\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_229,FILENAME='M:\2_J\LI\229\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_250,FILENAME='M:\2_J\LI\250\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_271,FILENAME='M:\2_J\LI\271\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_292,FILENAME='M:\2_J\LI\292\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_313,FILENAME='M:\2_J\LI\313\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_334,FILENAME='M:\2_J\LI\334\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_355,FILENAME='M:\2_J\LI\355\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_1,FILENAME='M:\2_J\LI\1\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_21,FILENAME='M:\2_J\LI\21\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_41,FILENAME='M:\2_J\LI\41\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_61,FILENAME='M:\2_J\LI\61\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_81,FILENAME='M:\2_J\LI\81\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_101,FILENAME='M:\2_J\LI\101\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_121,FILENAME='M:\2_J\LI\121\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_141,FILENAME='M:\2_J\LI\141\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_161,FILENAME='M:\2_J\LI\161\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_181,FILENAME='M:\2_J\LI\181\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_204,FILENAME='M:\2_J\LI\204\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_230,FILENAME='M:\2_J\LI\230\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_251,FILENAME='M:\2_J\LI\251\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_272,FILENAME='M:\2_J\LI\272\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_293,FILENAME='M:\2_J\LI\293\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_314,FILENAME='M:\2_J\LI\314\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_335,FILENAME='M:\2_J\LI\335\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_356,FILENAME='M:\2_J\LI\356\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_2,FILENAME='M:\2_J\LI\2\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_22,FILENAME='M:\2_J\LI\22\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_42,FILENAME='M:\2_J\LI\42\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_62,FILENAME='M:\2_J\LI\62\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_82,FILENAME='M:\2_J\LI\82\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_102,FILENAME='M:\2_J\LI\102\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_122,FILENAME='M:\2_J\LI\122\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_142,FILENAME='M:\2_J\LI\142\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_162,FILENAME='M:\2_J\LI\162\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_182,FILENAME='M:\2_J\LI\182\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_205,FILENAME='M:\2_J\LI\205\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_231,FILENAME='M:\2_J\LI\231\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_252,FILENAME='M:\2_J\LI\252\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_273,FILENAME='M:\2_J\LI\273\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_294,FILENAME='M:\2_J\LI\294\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_315,FILENAME='M:\2_J\LI\315\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_336,FILENAME='M:\2_J\LI\336\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_357,FILENAME='M:\2_J\LI\357\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_3,FILENAME='M:\2_J\LI\3\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_23,FILENAME='M:\2_J\LI\23\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_43,FILENAME='M:\2_J\LI\43\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_63,FILENAME='M:\2_J\LI\63\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_83,FILENAME='M:\2_J\LI\83\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_103,FILENAME='M:\2_J\LI\103\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_123,FILENAME='M:\2_J\LI\123\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_143,FILENAME='M:\2_J\LI\143\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_163,FILENAME='M:\2_J\LI\163\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_183,FILENAME='M:\2_J\LI\183\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_206,FILENAME='M:\2_J\LI\206\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_232,FILENAME='M:\2_J\LI\232\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_253,FILENAME='M:\2_J\LI\253\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_274,FILENAME='M:\2_J\LI\274\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_295,FILENAME='M:\2_J\LI\295\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_316,FILENAME='M:\2_J\LI\316\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_337,FILENAME='M:\2_J\LI\337\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_358,FILENAME='M:\2_J\LI\358\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_4,FILENAME='M:\2_J\LI\4\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_24,FILENAME='M:\2_J\LI\24\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_44,FILENAME='M:\2_J\LI\44\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_64,FILENAME='M:\2_J\LI\64\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_84,FILENAME='M:\2_J\LI\84\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_104,FILENAME='M:\2_J\LI\104\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_124,FILENAME='M:\2_J\LI\124\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_144,FILENAME='M:\2_J\LI\144\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_164,FILENAME='M:\2_J\LI\164\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_184,FILENAME='M:\2_J\LI\184\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_207,FILENAME='M:\2_J\LI\207\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_233,FILENAME='M:\2_J\LI\233\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_254,FILENAME='M:\2_J\LI\254\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_275,FILENAME='M:\2_J\LI\275\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_296,FILENAME='M:\2_J\LI\296\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_317,FILENAME='M:\2_J\LI\317\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_338,FILENAME='M:\2_J\LI\338\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_359,FILENAME='M:\2_J\LI\359\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_5,FILENAME='M:\2_J\LI\5\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_25,FILENAME='M:\2_J\LI\25\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_45,FILENAME='M:\2_J\LI\45\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_65,FILENAME='M:\2_J\LI\65\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_85,FILENAME='M:\2_J\LI\85\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_105,FILENAME='M:\2_J\LI\105\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_125,FILENAME='M:\2_J\LI\125\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_145,FILENAME='M:\2_J\LI\145\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_165,FILENAME='M:\2_J\LI\165\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_185,FILENAME='M:\2_J\LI\185\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_208,FILENAME='M:\2_J\LI\208\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_234,FILENAME='M:\2_J\LI\234\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_255,FILENAME='M:\2_J\LI\255\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_276,FILENAME='M:\2_J\LI\276\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_297,FILENAME='M:\2_J\LI\297\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_318,FILENAME='M:\2_J\LI\318\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_339,FILENAME='M:\2_J\LI\339\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_360,FILENAME='M:\2_J\LI\360\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_6,FILENAME='M:\2_J\LI\6\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_26,FILENAME='M:\2_J\LI\26\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_46,FILENAME='M:\2_J\LI\46\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_66,FILENAME='M:\2_J\LI\66\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_86,FILENAME='M:\2_J\LI\86\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_106,FILENAME='M:\2_J\LI\106\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_126,FILENAME='M:\2_J\LI\126\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_146,FILENAME='M:\2_J\LI\146\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_166,FILENAME='M:\2_J\LI\166\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_186,FILENAME='M:\2_J\LI\186\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_209,FILENAME='M:\2_J\LI\209\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_235,FILENAME='M:\2_J\LI\235\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_256,FILENAME='M:\2_J\LI\256\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_277,FILENAME='M:\2_J\LI\277\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_298,FILENAME='M:\2_J\LI\298\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_319,FILENAME='M:\2_J\LI\319\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_340,FILENAME='M:\2_J\LI\340\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_361,FILENAME='M:\2_J\LI\361\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_7,FILENAME='M:\2_J\LI\7\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_27,FILENAME='M:\2_J\LI\27\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_47,FILENAME='M:\2_J\LI\47\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_67,FILENAME='M:\2_J\LI\67\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_87,FILENAME='M:\2_J\LI\87\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_107,FILENAME='M:\2_J\LI\107\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_127,FILENAME='M:\2_J\LI\127\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_147,FILENAME='M:\2_J\LI\147\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_167,FILENAME='M:\2_J\LI\167\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_187,FILENAME='M:\2_J\LI\187\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_210,FILENAME='M:\2_J\LI\210\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_236,FILENAME='M:\2_J\LI\236\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_257,FILENAME='M:\2_J\LI\257\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_278,FILENAME='M:\2_J\LI\278\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_299,FILENAME='M:\2_J\LI\299\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_320,FILENAME='M:\2_J\LI\320\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_341,FILENAME='M:\2_J\LI\341\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_362,FILENAME='M:\2_J\LI\362\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_8,FILENAME='M:\2_J\LI\8\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_28,FILENAME='M:\2_J\LI\28\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_48,FILENAME='M:\2_J\LI\48\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_68,FILENAME='M:\2_J\LI\68\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_88,FILENAME='M:\2_J\LI\88\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_108,FILENAME='M:\2_J\LI\108\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_128,FILENAME='M:\2_J\LI\128\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_148,FILENAME='M:\2_J\LI\148\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_168,FILENAME='M:\2_J\LI\168\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_188,FILENAME='M:\2_J\LI\188\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_211,FILENAME='M:\2_J\LI\211\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_237,FILENAME='M:\2_J\LI\237\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_258,FILENAME='M:\2_J\LI\258\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_279,FILENAME='M:\2_J\LI\279\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_300,FILENAME='M:\2_J\LI\300\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_321,FILENAME='M:\2_J\LI\321\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_342,FILENAME='M:\2_J\LI\342\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_363,FILENAME='M:\2_J\LI\363\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_9,FILENAME='M:\2_J\LI\9\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_29,FILENAME='M:\2_J\LI\29\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_49,FILENAME='M:\2_J\LI\49\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_69,FILENAME='M:\2_J\LI\69\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_89,FILENAME='M:\2_J\LI\89\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_109,FILENAME='M:\2_J\LI\109\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_129,FILENAME='M:\2_J\LI\129\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_149,FILENAME='M:\2_J\LI\149\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_169,FILENAME='M:\2_J\LI\169\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_189,FILENAME='M:\2_J\LI\189\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_212,FILENAME='M:\2_J\LI\212\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_238,FILENAME='M:\2_J\LI\238\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_259,FILENAME='M:\2_J\LI\259\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_280,FILENAME='M:\2_J\LI\280\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_301,FILENAME='M:\2_J\LI\301\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_322,FILENAME='M:\2_J\LI\322\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_343,FILENAME='M:\2_J\LI\343\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_364,FILENAME='M:\2_J\LI\364\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_10,FILENAME='M:\2_J\LI\10\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_30,FILENAME='M:\2_J\LI\30\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_50,FILENAME='M:\2_J\LI\50\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_70,FILENAME='M:\2_J\LI\70\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_90,FILENAME='M:\2_J\LI\90\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_110,FILENAME='M:\2_J\LI\110\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_130,FILENAME='M:\2_J\LI\130\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_150,FILENAME='M:\2_J\LI\150\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_170,FILENAME='M:\2_J\LI\170\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_190,FILENAME='M:\2_J\LI\190\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_219,FILENAME='M:\2_J\LI\219\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_240,FILENAME='M:\2_J\LI\240\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_261,FILENAME='M:\2_J\LI\261\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_282,FILENAME='M:\2_J\LI\282\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_303,FILENAME='M:\2_J\LI\303\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_324,FILENAME='M:\2_J\LI\324\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_345,FILENAME='M:\2_J\LI\345\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_11,FILENAME='M:\2_J\LI\11\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_31,FILENAME='M:\2_J\LI\31\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_51,FILENAME='M:\2_J\LI\51\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_71,FILENAME='M:\2_J\LI\71\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_91,FILENAME='M:\2_J\LI\91\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_111,FILENAME='M:\2_J\LI\111\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_131,FILENAME='M:\2_J\LI\131\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_151,FILENAME='M:\2_J\LI\151\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_171,FILENAME='M:\2_J\LI\171\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_191,FILENAME='M:\2_J\LI\191\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_220,FILENAME='M:\2_J\LI\220\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_241,FILENAME='M:\2_J\LI\241\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_262,FILENAME='M:\2_J\LI\262\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_283,FILENAME='M:\2_J\LI\283\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_304,FILENAME='M:\2_J\LI\304\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_325,FILENAME='M:\2_J\LI\325\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_346,FILENAME='M:\2_J\LI\346\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_12,FILENAME='M:\2_J\LI\12\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_32,FILENAME='M:\2_J\LI\32\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_52,FILENAME='M:\2_J\LI\52\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_72,FILENAME='M:\2_J\LI\72\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_92,FILENAME='M:\2_J\LI\92\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_112,FILENAME='M:\2_J\LI\112\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_132,FILENAME='M:\2_J\LI\132\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_152,FILENAME='M:\2_J\LI\152\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_172,FILENAME='M:\2_J\LI\172\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_192,FILENAME='M:\2_J\LI\192\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_221,FILENAME='M:\2_J\LI\221\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_242,FILENAME='M:\2_J\LI\242\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_263,FILENAME='M:\2_J\LI\263\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_284,FILENAME='M:\2_J\LI\284\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_305,FILENAME='M:\2_J\LI\305\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_326,FILENAME='M:\2_J\LI\326\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_347,FILENAME='M:\2_J\LI\347\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_13,FILENAME='M:\2_J\LI\13\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_33,FILENAME='M:\2_J\LI\33\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_53,FILENAME='M:\2_J\LI\53\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_73,FILENAME='M:\2_J\LI\73\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_93,FILENAME='M:\2_J\LI\93\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_113,FILENAME='M:\2_J\LI\113\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_133,FILENAME='M:\2_J\LI\133\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_153,FILENAME='M:\2_J\LI\153\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_173,FILENAME='M:\2_J\LI\173\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_196,FILENAME='M:\2_J\LI\196\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_222,FILENAME='M:\2_J\LI\222\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_243,FILENAME='M:\2_J\LI\243\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_264,FILENAME='M:\2_J\LI\264\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_285,FILENAME='M:\2_J\LI\285\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_306,FILENAME='M:\2_J\LI\306\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_327,FILENAME='M:\2_J\LI\327\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_348,FILENAME='M:\2_J\LI\348\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_14,FILENAME='M:\2_J\LI\14\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_34,FILENAME='M:\2_J\LI\34\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_54,FILENAME='M:\2_J\LI\54\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_74,FILENAME='M:\2_J\LI\74\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_94,FILENAME='M:\2_J\LI\94\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_114,FILENAME='M:\2_J\LI\114\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_134,FILENAME='M:\2_J\LI\134\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_154,FILENAME='M:\2_J\LI\154\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_174,FILENAME='M:\2_J\LI\174\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_197,FILENAME='M:\2_J\LI\197\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_223,FILENAME='M:\2_J\LI\223\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_244,FILENAME='M:\2_J\LI\244\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_265,FILENAME='M:\2_J\LI\265\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_286,FILENAME='M:\2_J\LI\286\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_307,FILENAME='M:\2_J\LI\307\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_328,FILENAME='M:\2_J\LI\328\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_349,FILENAME='M:\2_J\LI\349\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_15,FILENAME='M:\2_J\LI\15\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_35,FILENAME='M:\2_J\LI\35\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_55,FILENAME='M:\2_J\LI\55\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_75,FILENAME='M:\2_J\LI\75\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_95,FILENAME='M:\2_J\LI\95\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_115,FILENAME='M:\2_J\LI\115\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_135,FILENAME='M:\2_J\LI\135\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_155,FILENAME='M:\2_J\LI\155\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_175,FILENAME='M:\2_J\LI\175\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_198,FILENAME='M:\2_J\LI\198\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_224,FILENAME='M:\2_J\LI\224\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_245,FILENAME='M:\2_J\LI\245\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_266,FILENAME='M:\2_J\LI\266\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_287,FILENAME='M:\2_J\LI\287\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_308,FILENAME='M:\2_J\LI\308\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_329,FILENAME='M:\2_J\LI\329\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_350,FILENAME='M:\2_J\LI\350\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_16,FILENAME='M:\2_J\LI\16\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_36,FILENAME='M:\2_J\LI\36\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_56,FILENAME='M:\2_J\LI\56\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_76,FILENAME='M:\2_J\LI\76\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_96,FILENAME='M:\2_J\LI\96\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_116,FILENAME='M:\2_J\LI\116\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_136,FILENAME='M:\2_J\LI\136\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_156,FILENAME='M:\2_J\LI\156\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_176,FILENAME='M:\2_J\LI\176\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_199,FILENAME='M:\2_J\LI\199\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_225,FILENAME='M:\2_J\LI\225\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_246,FILENAME='M:\2_J\LI\246\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_267,FILENAME='M:\2_J\LI\267\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_288,FILENAME='M:\2_J\LI\288\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_309,FILENAME='M:\2_J\LI\309\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_330,FILENAME='M:\2_J\LI\330\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_351,FILENAME='M:\2_J\LI\351\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_17,FILENAME='M:\2_J\LI\17\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_37,FILENAME='M:\2_J\LI\37\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_57,FILENAME='M:\2_J\LI\57\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_77,FILENAME='M:\2_J\LI\77\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_97,FILENAME='M:\2_J\LI\97\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_117,FILENAME='M:\2_J\LI\117\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_137,FILENAME='M:\2_J\LI\137\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_157,FILENAME='M:\2_J\LI\157\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_177,FILENAME='M:\2_J\LI\177\',SIZE=2300mb, FILEGROWTH=0),
  (NAME=lineitem_200,FILENAME='M:\2_J\LI\200\',SIZE=2300mb, FILEGROWTH=0),
   (NAME=lineitem_226,FILENAME='M:\2_J\LI\226\',SIZE=2300mb, FILEGROWTH=0),
   (NAME=lineitem_247,FILENAME='M:\2_J\LI\247\',SIZE=2300mb, FILEGROWTH=0),
   (NAME=lineitem_268,FILENAME='M:\2_J\LI\268\',SIZE=2300mb, FILEGROWTH=0),
   (NAME=lineitem_289,FILENAME='M:\2_J\LI\289\',SIZE=2300mb, FILEGROWTH=0),
   (NAME=lineitem_310,FILENAME='M:\2_J\LI\310\',SIZE=2300mb, FILEGROWTH=0),
   (NAME=lineitem_331,FILENAME='M:\2_J\LI\331\',SIZE=2300mb, FILEGROWTH=0),
   (NAME=lineitem_352,FILENAME='M:\2_J\LI\352\',SIZE=2300mb, FILEGROWTH=0),
   (NAME=lineitem_18,FILENAME='M:\2_J\LI\18\',SIZE=2300mb, FILEGROWTH=0),
   (NAME=lineitem_38,FILENAME='M:\2_J\LI\38\',SIZE=2300mb, FILEGROWTH=0),
   (NAME=lineitem_58,FILENAME='M:\2_J\LI\58\',SIZE=2300mb, FILEGROWTH=0),
   (NAME=lineitem_78,FILENAME='M:\2_J\LI\78\',SIZE=2300mb, FILEGROWTH=0),
   (NAME=lineitem_98,FILENAME='M:\2_J\LI\98\',SIZE=2300mb, FILEGROWTH=0),
   (NAME=lineitem_118,FILENAME='M:\2_J\LI\118\',SIZE=2300mb, FILEGROWTH=0),
   (NAME=lineitem_138,FILENAME='M:\2_J\LI\138\',SIZE=2300mb, FILEGROWTH=0),
   (NAME=lineitem_158,FILENAME='M:\2_J\LI\158\',SIZE=2300mb, FILEGROWTH=0),
   (NAME=lineitem_178,FILENAME='M:\2_J\LI\178\',SIZE=2300mb, FILEGROWTH=0),
   (NAME=lineitem_201,FILENAME='M:\2_J\LI\201\',SIZE=2300mb, FILEGROWTH=0),
   (NAME=lineitem_227,FILENAME='M:\2_J\LI\227\',SIZE=2300mb, FILEGROWTH=0),
   (NAME=lineitem_248,FILENAME='M:\2_J\LI\248\',SIZE=2300mb, FILEGROWTH=0),
   (NAME=lineitem_269,FILENAME='M:\2_J\LI\269\',SIZE=2300mb, FILEGROWTH=0),
   (NAME=lineitem_290,FILENAME='M:\2_J\LI\290\',SIZE=2300mb, FILEGROWTH=0),
   (NAME=lineitem_311,FILENAME='M:\2_J\LI\311\',SIZE=2300mb, FILEGROWTH=0),
   (NAME=lineitem_332,FILENAME='M:\2_J\LI\332\',SIZE=2300mb, FILEGROWTH=0),
   (NAME=lineitem_353,FILENAME='M:\2_J\LI\353\',SIZE=2300mb, FILEGROWTH=0),
   (NAME=lineitem_19,FILENAME='M:\2_J\LI\19\',SIZE=2300mb, FILEGROWTH=0),
   (NAME=lineitem_39,FILENAME='M:\2_J\LI\39\',SIZE=2300mb, FILEGROWTH=0),
   (NAME=lineitem_59,FILENAME='M:\2_J\LI\59\',SIZE=2300mb, FILEGROWTH=0),
   (NAME=lineitem_79,FILENAME='M:\2_J\LI\79\',SIZE=2300mb, FILEGROWTH=0),
   (NAME=lineitem_99,FILENAME='M:\2_J\LI\99\',SIZE=2300mb, FILEGROWTH=0),
   (NAME=lineitem_119,FILENAME='M:\2_J\LI\119\',SIZE=2300mb, FILEGROWTH=0),
   (NAME=lineitem_139,FILENAME='M:\2_J\LI\139\',SIZE=2300mb, FILEGROWTH=0),
   (NAME=lineitem_159,FILENAME='M:\2_J\LI\159\',SIZE=2300mb, FILEGROWTH=0),
   (NAME=lineitem_179,FILENAME='M:\2_J\LI\179\',SIZE=2300mb, FILEGROWTH=0),
   (NAME=lineitem_202,FILENAME='M:\2_J\LI\202\',SIZE=2300mb, FILEGROWTH=0),
   (NAME=lineitem_228,FILENAME='M:\2_J\LI\228\',SIZE=2300mb, FILEGROWTH=0),
   (NAME=lineitem_249,FILENAME='M:\2_J\LI\249\',SIZE=2300mb, FILEGROWTH=0),
   (NAME=lineitem_270,FILENAME='M:\2_J\LI\270\',SIZE=2300mb, FILEGROWTH=0),
   (NAME=lineitem_291,FILENAME='M:\2_J\LI\291\',SIZE=2300mb, FILEGROWTH=0),
   (NAME=lineitem_312,FILENAME='M:\2_J\LI\312\',SIZE=2300mb, FILEGROWTH=0),
   (NAME=lineitem_333,FILENAME='M:\2_J\LI\333\',SIZE=2300mb, FILEGROWTH=0),
   (NAME=lineitem_354,FILENAME='M:\2_J\LI\354\',SIZE=2300mb, FILEGROWTH=0),


FILEGROUP          GENERAL_FG

   (NAME=general_0,FILENAME='M:\2_J\GEN\0\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_20,FILENAME='M:\2_J\GEN\20\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_40,FILENAME='M:\2_J\GEN\40\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_60,FILENAME='M:\2_J\GEN\60\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_80,FILENAME='M:\2_J\GEN\80\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_100,FILENAME='M:\2_J\GEN\100\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_120,FILENAME='M:\2_J\GEN\120\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_140,FILENAME='M:\2_J\GEN\140\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_160,FILENAME='M:\2_J\GEN\160\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_180,FILENAME='M:\2_J\GEN\180\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_203,FILENAME='M:\2_J\GEN\203\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_229,FILENAME='M:\2_J\GEN\229\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_250,FILENAME='M:\2_J\GEN\250\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_271,FILENAME='M:\2_J\GEN\271\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_292,FILENAME='M:\2_J\GEN\292\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_313,FILENAME='M:\2_J\GEN\313\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_334,FILENAME='M:\2_J\GEN\334\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_355,FILENAME='M:\2_J\GEN\355\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_1,FILENAME='M:\2_J\GEN\1\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_21,FILENAME='M:\2_J\GEN\21\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_41,FILENAME='M:\2_J\GEN\41\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_61,FILENAME='M:\2_J\GEN\61\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_81,FILENAME='M:\2_J\GEN\81\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_101,FILENAME='M:\2_J\GEN\101\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_121,FILENAME='M:\2_J\GEN\121\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_141,FILENAME='M:\2_J\GEN\141\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_161,FILENAME='M:\2_J\GEN\161\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_181,FILENAME='M:\2_J\GEN\181\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_204,FILENAME='M:\2_J\GEN\204\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_230,FILENAME='M:\2_J\GEN\230\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_251,FILENAME='M:\2_J\GEN\251\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_272,FILENAME='M:\2_J\GEN\272\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_293,FILENAME='M:\2_J\GEN\293\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_314,FILENAME='M:\2_J\GEN\314\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_335,FILENAME='M:\2_J\GEN\335\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_356,FILENAME='M:\2_J\GEN\356\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_2,FILENAME='M:\2_J\GEN\2\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_22,FILENAME='M:\2_J\GEN\22\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_42,FILENAME='M:\2_J\GEN\42\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_62,FILENAME='M:\2_J\GEN\62\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_82,FILENAME='M:\2_J\GEN\82\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_102,FILENAME='M:\2_J\GEN\102\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_122,FILENAME='M:\2_J\GEN\122\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_142,FILENAME='M:\2_J\GEN\142\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_162,FILENAME='M:\2_J\GEN\162\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_182,FILENAME='M:\2_J\GEN\182\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_205,FILENAME='M:\2_J\GEN\205\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_231,FILENAME='M:\2_J\GEN\231\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_252,FILENAME='M:\2_J\GEN\252\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_273,FILENAME='M:\2_J\GEN\273\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_294,FILENAME='M:\2_J\GEN\294\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_315,FILENAME='M:\2_J\GEN\315\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_336,FILENAME='M:\2_J\GEN\336\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_357,FILENAME='M:\2_J\GEN\357\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_3,FILENAME='M:\2_J\GEN\3\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_23,FILENAME='M:\2_J\GEN\23\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_43,FILENAME='M:\2_J\GEN\43\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_63,FILENAME='M:\2_J\GEN\63\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_83,FILENAME='M:\2_J\GEN\83\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_103,FILENAME='M:\2_J\GEN\103\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_123,FILENAME='M:\2_J\GEN\123\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_143,FILENAME='M:\2_J\GEN\143\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_163,FILENAME='M:\2_J\GEN\163\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_183,FILENAME='M:\2_J\GEN\183\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_206,FILENAME='M:\2_J\GEN\206\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_232,FILENAME='M:\2_J\GEN\232\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_253,FILENAME='M:\2_J\GEN\253\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_274,FILENAME='M:\2_J\GEN\274\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_295,FILENAME='M:\2_J\GEN\295\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_316,FILENAME='M:\2_J\GEN\316\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_337,FILENAME='M:\2_J\GEN\337\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_358,FILENAME='M:\2_J\GEN\358\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_4,FILENAME='M:\2_J\GEN\4\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_24,FILENAME='M:\2_J\GEN\24\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_44,FILENAME='M:\2_J\GEN\44\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_64,FILENAME='M:\2_J\GEN\64\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_84,FILENAME='M:\2_J\GEN\84\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_104,FILENAME='M:\2_J\GEN\104\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_124,FILENAME='M:\2_J\GEN\124\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_144,FILENAME='M:\2_J\GEN\144\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_164,FILENAME='M:\2_J\GEN\164\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_184,FILENAME='M:\2_J\GEN\184\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_207,FILENAME='M:\2_J\GEN\207\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_233,FILENAME='M:\2_J\GEN\233\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_254,FILENAME='M:\2_J\GEN\254\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_275,FILENAME='M:\2_J\GEN\275\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_296,FILENAME='M:\2_J\GEN\296\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_317,FILENAME='M:\2_J\GEN\317\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_338,FILENAME='M:\2_J\GEN\338\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_359,FILENAME='M:\2_J\GEN\359\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_5,FILENAME='M:\2_J\GEN\5\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_25,FILENAME='M:\2_J\GEN\25\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_45,FILENAME='M:\2_J\GEN\45\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_65,FILENAME='M:\2_J\GEN\65\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_85,FILENAME='M:\2_J\GEN\85\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_105,FILENAME='M:\2_J\GEN\105\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_125,FILENAME='M:\2_J\GEN\125\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_145,FILENAME='M:\2_J\GEN\145\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_165,FILENAME='M:\2_J\GEN\165\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_185,FILENAME='M:\2_J\GEN\185\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_208,FILENAME='M:\2_J\GEN\208\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_234,FILENAME='M:\2_J\GEN\234\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_255,FILENAME='M:\2_J\GEN\255\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_276,FILENAME='M:\2_J\GEN\276\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_297,FILENAME='M:\2_J\GEN\297\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_318,FILENAME='M:\2_J\GEN\318\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_339,FILENAME='M:\2_J\GEN\339\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_360,FILENAME='M:\2_J\GEN\360\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_6,FILENAME='M:\2_J\GEN\6\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_26,FILENAME='M:\2_J\GEN\26\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_46,FILENAME='M:\2_J\GEN\46\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_66,FILENAME='M:\2_J\GEN\66\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_86,FILENAME='M:\2_J\GEN\86\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_106,FILENAME='M:\2_J\GEN\106\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_126,FILENAME='M:\2_J\GEN\126\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_146,FILENAME='M:\2_J\GEN\146\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_166,FILENAME='M:\2_J\GEN\166\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_186,FILENAME='M:\2_J\GEN\186\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_209,FILENAME='M:\2_J\GEN\209\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_235,FILENAME='M:\2_J\GEN\235\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_256,FILENAME='M:\2_J\GEN\256\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_277,FILENAME='M:\2_J\GEN\277\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_298,FILENAME='M:\2_J\GEN\298\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_319,FILENAME='M:\2_J\GEN\319\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_340,FILENAME='M:\2_J\GEN\340\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_361,FILENAME='M:\2_J\GEN\361\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_7,FILENAME='M:\2_J\GEN\7\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_27,FILENAME='M:\2_J\GEN\27\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_47,FILENAME='M:\2_J\GEN\47\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_67,FILENAME='M:\2_J\GEN\67\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_87,FILENAME='M:\2_J\GEN\87\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_107,FILENAME='M:\2_J\GEN\107\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_127,FILENAME='M:\2_J\GEN\127\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_147,FILENAME='M:\2_J\GEN\147\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_167,FILENAME='M:\2_J\GEN\167\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_187,FILENAME='M:\2_J\GEN\187\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_210,FILENAME='M:\2_J\GEN\210\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_236,FILENAME='M:\2_J\GEN\236\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_257,FILENAME='M:\2_J\GEN\257\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_278,FILENAME='M:\2_J\GEN\278\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_299,FILENAME='M:\2_J\GEN\299\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_320,FILENAME='M:\2_J\GEN\320\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_341,FILENAME='M:\2_J\GEN\341\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_362,FILENAME='M:\2_J\GEN\362\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_8,FILENAME='M:\2_J\GEN\8\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_28,FILENAME='M:\2_J\GEN\28\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_48,FILENAME='M:\2_J\GEN\48\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_68,FILENAME='M:\2_J\GEN\68\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_88,FILENAME='M:\2_J\GEN\88\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_108,FILENAME='M:\2_J\GEN\108\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_128,FILENAME='M:\2_J\GEN\128\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_148,FILENAME='M:\2_J\GEN\148\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_168,FILENAME='M:\2_J\GEN\168\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_188,FILENAME='M:\2_J\GEN\188\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_211,FILENAME='M:\2_J\GEN\211\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_237,FILENAME='M:\2_J\GEN\237\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_258,FILENAME='M:\2_J\GEN\258\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_279,FILENAME='M:\2_J\GEN\279\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_300,FILENAME='M:\2_J\GEN\300\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_321,FILENAME='M:\2_J\GEN\321\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_342,FILENAME='M:\2_J\GEN\342\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_363,FILENAME='M:\2_J\GEN\363\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_9,FILENAME='M:\2_J\GEN\9\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_29,FILENAME='M:\2_J\GEN\29\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_49,FILENAME='M:\2_J\GEN\49\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_69,FILENAME='M:\2_J\GEN\69\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_89,FILENAME='M:\2_J\GEN\89\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_109,FILENAME='M:\2_J\GEN\109\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_129,FILENAME='M:\2_J\GEN\129\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_149,FILENAME='M:\2_J\GEN\149\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_169,FILENAME='M:\2_J\GEN\169\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_189,FILENAME='M:\2_J\GEN\189\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_212,FILENAME='M:\2_J\GEN\212\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_238,FILENAME='M:\2_J\GEN\238\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_259,FILENAME='M:\2_J\GEN\259\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_280,FILENAME='M:\2_J\GEN\280\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_301,FILENAME='M:\2_J\GEN\301\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_322,FILENAME='M:\2_J\GEN\322\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_343,FILENAME='M:\2_J\GEN\343\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_364,FILENAME='M:\2_J\GEN\364\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_10,FILENAME='M:\2_J\GEN\10\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_30,FILENAME='M:\2_J\GEN\30\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_50,FILENAME='M:\2_J\GEN\50\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_70,FILENAME='M:\2_J\GEN\70\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_90,FILENAME='M:\2_J\GEN\90\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_110,FILENAME='M:\2_J\GEN\110\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_130,FILENAME='M:\2_J\GEN\130\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_150,FILENAME='M:\2_J\GEN\150\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_170,FILENAME='M:\2_J\GEN\170\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_190,FILENAME='M:\2_J\GEN\190\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_219,FILENAME='M:\2_J\GEN\219\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_240,FILENAME='M:\2_J\GEN\240\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_261,FILENAME='M:\2_J\GEN\261\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_282,FILENAME='M:\2_J\GEN\282\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_303,FILENAME='M:\2_J\GEN\303\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_324,FILENAME='M:\2_J\GEN\324\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_345,FILENAME='M:\2_J\GEN\345\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_11,FILENAME='M:\2_J\GEN\11\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_31,FILENAME='M:\2_J\GEN\31\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_51,FILENAME='M:\2_J\GEN\51\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_71,FILENAME='M:\2_J\GEN\71\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_91,FILENAME='M:\2_J\GEN\91\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_111,FILENAME='M:\2_J\GEN\111\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_131,FILENAME='M:\2_J\GEN\131\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_151,FILENAME='M:\2_J\GEN\151\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_171,FILENAME='M:\2_J\GEN\171\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_191,FILENAME='M:\2_J\GEN\191\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_220,FILENAME='M:\2_J\GEN\220\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_241,FILENAME='M:\2_J\GEN\241\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_262,FILENAME='M:\2_J\GEN\262\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_283,FILENAME='M:\2_J\GEN\283\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_304,FILENAME='M:\2_J\GEN\304\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_325,FILENAME='M:\2_J\GEN\325\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_346,FILENAME='M:\2_J\GEN\346\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_12,FILENAME='M:\2_J\GEN\12\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_32,FILENAME='M:\2_J\GEN\32\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_52,FILENAME='M:\2_J\GEN\52\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_72,FILENAME='M:\2_J\GEN\72\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_92,FILENAME='M:\2_J\GEN\92\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_112,FILENAME='M:\2_J\GEN\112\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_132,FILENAME='M:\2_J\GEN\132\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_152,FILENAME='M:\2_J\GEN\152\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_172,FILENAME='M:\2_J\GEN\172\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_192,FILENAME='M:\2_J\GEN\192\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_221,FILENAME='M:\2_J\GEN\221\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_242,FILENAME='M:\2_J\GEN\242\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_263,FILENAME='M:\2_J\GEN\263\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_284,FILENAME='M:\2_J\GEN\284\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_305,FILENAME='M:\2_J\GEN\305\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_326,FILENAME='M:\2_J\GEN\326\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_347,FILENAME='M:\2_J\GEN\347\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_13,FILENAME='M:\2_J\GEN\13\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_33,FILENAME='M:\2_J\GEN\33\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_53,FILENAME='M:\2_J\GEN\53\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_73,FILENAME='M:\2_J\GEN\73\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_93,FILENAME='M:\2_J\GEN\93\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_113,FILENAME='M:\2_J\GEN\113\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_133,FILENAME='M:\2_J\GEN\133\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_153,FILENAME='M:\2_J\GEN\153\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_173,FILENAME='M:\2_J\GEN\173\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_196,FILENAME='M:\2_J\GEN\196\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_222,FILENAME='M:\2_J\GEN\222\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_243,FILENAME='M:\2_J\GEN\243\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_264,FILENAME='M:\2_J\GEN\264\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_285,FILENAME='M:\2_J\GEN\285\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_306,FILENAME='M:\2_J\GEN\306\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_327,FILENAME='M:\2_J\GEN\327\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_348,FILENAME='M:\2_J\GEN\348\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_14,FILENAME='M:\2_J\GEN\14\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_34,FILENAME='M:\2_J\GEN\34\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_54,FILENAME='M:\2_J\GEN\54\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_74,FILENAME='M:\2_J\GEN\74\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_94,FILENAME='M:\2_J\GEN\94\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_114,FILENAME='M:\2_J\GEN\114\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_134,FILENAME='M:\2_J\GEN\134\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_154,FILENAME='M:\2_J\GEN\154\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_174,FILENAME='M:\2_J\GEN\174\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_197,FILENAME='M:\2_J\GEN\197\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_223,FILENAME='M:\2_J\GEN\223\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_244,FILENAME='M:\2_J\GEN\244\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_265,FILENAME='M:\2_J\GEN\265\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_286,FILENAME='M:\2_J\GEN\286\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_307,FILENAME='M:\2_J\GEN\307\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_328,FILENAME='M:\2_J\GEN\328\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_349,FILENAME='M:\2_J\GEN\349\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_15,FILENAME='M:\2_J\GEN\15\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_35,FILENAME='M:\2_J\GEN\35\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_55,FILENAME='M:\2_J\GEN\55\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_75,FILENAME='M:\2_J\GEN\75\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_95,FILENAME='M:\2_J\GEN\95\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_115,FILENAME='M:\2_J\GEN\115\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_135,FILENAME='M:\2_J\GEN\135\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_155,FILENAME='M:\2_J\GEN\155\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_175,FILENAME='M:\2_J\GEN\175\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_198,FILENAME='M:\2_J\GEN\198\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_224,FILENAME='M:\2_J\GEN\224\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_245,FILENAME='M:\2_J\GEN\245\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_266,FILENAME='M:\2_J\GEN\266\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_287,FILENAME='M:\2_J\GEN\287\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_308,FILENAME='M:\2_J\GEN\308\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_329,FILENAME='M:\2_J\GEN\329\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_350,FILENAME='M:\2_J\GEN\350\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_16,FILENAME='M:\2_J\GEN\16\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_36,FILENAME='M:\2_J\GEN\36\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_56,FILENAME='M:\2_J\GEN\56\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_76,FILENAME='M:\2_J\GEN\76\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_96,FILENAME='M:\2_J\GEN\96\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_116,FILENAME='M:\2_J\GEN\116\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_136,FILENAME='M:\2_J\GEN\136\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_156,FILENAME='M:\2_J\GEN\156\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_176,FILENAME='M:\2_J\GEN\176\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_199,FILENAME='M:\2_J\GEN\199\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_225,FILENAME='M:\2_J\GEN\225\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_246,FILENAME='M:\2_J\GEN\246\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_267,FILENAME='M:\2_J\GEN\267\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_288,FILENAME='M:\2_J\GEN\288\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_309,FILENAME='M:\2_J\GEN\309\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_330,FILENAME='M:\2_J\GEN\330\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_351,FILENAME='M:\2_J\GEN\351\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_17,FILENAME='M:\2_J\GEN\17\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_37,FILENAME='M:\2_J\GEN\37\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_57,FILENAME='M:\2_J\GEN\57\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_77,FILENAME='M:\2_J\GEN\77\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_97,FILENAME='M:\2_J\GEN\97\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_117,FILENAME='M:\2_J\GEN\117\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_137,FILENAME='M:\2_J\GEN\137\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_157,FILENAME='M:\2_J\GEN\157\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_177,FILENAME='M:\2_J\GEN\177\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_200,FILENAME='M:\2_J\GEN\200\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_226,FILENAME='M:\2_J\GEN\226\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_247,FILENAME='M:\2_J\GEN\247\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_268,FILENAME='M:\2_J\GEN\268\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_289,FILENAME='M:\2_J\GEN\289\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_310,FILENAME='M:\2_J\GEN\310\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_331,FILENAME='M:\2_J\GEN\331\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_352,FILENAME='M:\2_J\GEN\352\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_18,FILENAME='M:\2_J\GEN\18\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_38,FILENAME='M:\2_J\GEN\38\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_58,FILENAME='M:\2_J\GEN\58\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_78,FILENAME='M:\2_J\GEN\78\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_98,FILENAME='M:\2_J\GEN\98\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_118,FILENAME='M:\2_J\GEN\118\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_138,FILENAME='M:\2_J\GEN\138\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_158,FILENAME='M:\2_J\GEN\158\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_178,FILENAME='M:\2_J\GEN\178\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_201,FILENAME='M:\2_J\GEN\201\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_227,FILENAME='M:\2_J\GEN\227\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_248,FILENAME='M:\2_J\GEN\248\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_269,FILENAME='M:\2_J\GEN\269\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_290,FILENAME='M:\2_J\GEN\290\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_311,FILENAME='M:\2_J\GEN\311\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_332,FILENAME='M:\2_J\GEN\332\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_353,FILENAME='M:\2_J\GEN\353\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_19,FILENAME='M:\2_J\GEN\19\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_39,FILENAME='M:\2_J\GEN\39\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_59,FILENAME='M:\2_J\GEN\59\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_79,FILENAME='M:\2_J\GEN\79\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_99,FILENAME='M:\2_J\GEN\99\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_119,FILENAME='M:\2_J\GEN\119\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_139,FILENAME='M:\2_J\GEN\139\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_159,FILENAME='M:\2_J\GEN\159\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_179,FILENAME='M:\2_J\GEN\179\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_202,FILENAME='M:\2_J\GEN\202\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_228,FILENAME='M:\2_J\GEN\228\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_249,FILENAME='M:\2_J\GEN\249\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_270,FILENAME='M:\2_J\GEN\270\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_291,FILENAME='M:\2_J\GEN\291\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_312,FILENAME='M:\2_J\GEN\312\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_333,FILENAME='M:\2_J\GEN\333\',SIZE=590mb, FILEGROWTH=0),
   (NAME=general_354,FILENAME='M:\2_J\GEN\354\',SIZE=590mb, FILEGROWTH=0)

LOG ON
  (NAME= tpch300g_log,FILENAME="H:",SIZE=45800MB, FILEGROWTH=0)
```

Create Tables

```
--  File:     CREATETABLES.SQL
--            Microsoft TPC-H Benchmark Kit Ver. 1.00
--            Copyright Microsoft, 1999
--

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      money                     not null,
               P_COMMENT          varchar(23)               not null)
on GENERAL_FG

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          money                     not null,
               S_COMMENT          varchar(101)              not null)
on GENERAL_FG

create table PARTSUPP
              (PS_PARTKEY         int                       not null,
               PS_SUPPKEY      int                   not null,
               PS_AVAILQTY     int                   not null,
               PS_SUPPLYCOST   money                 not null,
               PS_COMMENT      varchar(199)          not null)
on GENERAL_FG

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       money                 not null,
               C_MKTSEGMENT    char(10)   not null,
               C_COMMENT       varchar(117)          not null)
on GENERAL_FG

create table ORDERS
              (O_ORDERKEY      int                   not null,
               O_CUSTKEY       int                   not null,
               O_ORDERSTATUS   char(1)               not null,
               O_TOTALPRICE    money                 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

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      money                 not null,
               L_EXTENDEDPRICE money                 not null,
               L_DISCOUNT      money                 not null,
               L_TAX           money                 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 LINEITEM_FG

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 GENERAL_FG

create table REGION
              (R_REGIONKEY     int                   not null,
               R_NAME          char(25)   not null,
               R_COMMENT       varchar(152)          not null)
on GENERAL_FG
```

Create Indexes

```
--  File:     CREATECLUSTEREDINDEXES.SQL
--          Microsoft TPC-H Benchmark Kit Ver. 1.00
--          Copyright Microsoft, 1999
--

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

```
                on LINEITEM_FG

 create unique clustered index N_KEY_CLUIDX
                on NATION(N_NATIONKEY) with SORT_IN_TEMPDB
                on GENERAL_FG

 create unique clustered index R_KEY_CLUIDX
                on REGION(R_REGIONKEY) with SORT_IN_TEMPDB
                on GENERAL_FG

 create unique clustered index P_KEY_CLUIDX
                on PART(P_PARTKEY) with SORT_IN_TEMPDB
                on GENERAL_FG

 create unique clustered index S_KEY_CLUIDX
                on SUPPLIER(S_SUPPKEY) with SORT_IN_TEMPDB
                on GENERAL_FG

 create unique clustered index C_KEY_CLUIDX
                on CUSTOMER(C_CUSTKEY) with SORT_IN_TEMPDB
                on GENERAL_FG

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

 create unique clustered index PS_KEY_CLUIDX
                on PARTSUPP(PS_PARTKEY,PS_SUPPKEY) with SORT_IN_TEMPDB
                on GENERAL_FG


--  File:    CREATEINDEXESSTREAM1.SQL
--           Microsoft TPC-H Benchmark Kit Ver. 1.00
--           Copyright Microsoft, 1999
--

 create unique index O_OKEY_IDX
                on ORDERS(O_ORDERKEY)
                with fillfactor=95, SORT_IN_TEMPDB
                on GENERAL_FG


--  File:    CREATEINDEXESSTREAM2.SQL
--           Microsoft TPC-H Benchmark Kit Ver. 1.00
--           Copyright Microsoft, 1999
--

 create index O_CUSTKEY_IDX
                on ORDERS(O_CUSTKEY)
                with fillfactor=95, SORT_IN_TEMPDB
                on GENERAL_FG

create index L_SUPPKEY_IDX
                on LINEITEM(L_SUPPKEY)
                with FILLFACTOR=95, SORT_IN_TEMPDB
                on LINEITEM_FG


--  File:    CREATEINDEXESSTREAM3.SQL
--           Microsoft TPC-H Benchmark Kit Ver. 1.00
--           Copyright Microsoft, 1999
--

 create index L_ORDERKEY_IDX
        on LINEITEM(L_ORDERKEY)
        with fillfactor=95, SORT_IN_TEMPDB
        on LINEITEM_FG

 create index PS_SUPPKEY_IDX
        on PARTSUPP(PS_SUPPKEY)
        with fillfactor=100, SORT_IN_TEMPDB
```

```
       on GENERAL_FG

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

 create index S_NATIONKEY_IDX
                on SUPPLIER(S_NATIONKEY)
                with fillfactor=100, SORT_IN_TEMPDB
                on GENERAL_FG

 create index C_NATIONKEY_IDX
                on CUSTOMER(C_NATIONKEY)
                WITH FILLFACTOR=100, SORT_IN_TEMPDB
                on GENERAL_FG


--  File:    CREATEINDEXESSTREAM4.SQL
--          Microsoft TPC-H Benchmark Kit Ver. 1.00
--          Copyright Microsoft, 1999
--

 create index L_PARTKEY_SUPPKEY_IDX
                on LINEITEM(L_PARTKEY,L_SUPPKEY)
                with FILLFACTOR=95, SORT_IN_TEMPDB
                on LINEITEM_FG
```

APPENDIX C: Query Text & Output

/* TPC_H Query 1 - Pricing Summary Report */

-- using 1002051338 as a seed to the RNG

```
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, -96, '1998-12-01')
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          11337885223.0000     17000944891372.2700  16150885216231.8744  16796915053455.0898  25.4997            38236.4334          .0500
444626848
N          F          295858495.0000       443679933892.9400    421493456470.5201    438352334968.6499    25.5005            38241.5185          .0500
11602048
N          O          22239292107.0000     33347936352411.6700  31680533439763.8780  32947757131986.8482  25.5000            38237.4431          .0499
872127779
R          F          11338005221.0000     17000997445503.4200  16150930534687.0214  16796969115343.5277  25.5001            38236.7762          .0500
444624236

(4 row(s) affected)
+++++++++++++++++++++++++
```

/* TPC_H Query 2 - Minimum Cost Supplier */

-- using 1002051338 as a seed to the RNG

```
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            = 6 AND
         P_TYPE            LIKE '%TIN' AND
```

```
        S_NATIONKEY      = N_NATIONKEY AND
        N_REGIONKEY      = R_REGIONKEY AND
        R_NAME           = 'AFRICA' 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             = 'AFRICA'
                           )
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
-------------------- ------------------------ ------------------------ ----------- ------------------------ ---------------------------------------- --------------- ----------------------------------------
------------------------------------------------------------------
9999.9800         Supplier#000523705      MOZAMBIQUE            8023700    Manufacturer#3          0XL78nMq4KiHniMOcIbfDYSab0CmJkWkoOofZqUs 26-
473-916-5258 deposits x-ray slyly. quickly pending packages cajole slyly. fluffily final as
9999.8300         Supplier#001191072      KENYA                51441054    Manufacturer#1          5SnOaChYZT57pdtoCeYC0r3900qAx9X2bB       24-144-966-
9370 even accounts boost across the packages: ironically final pinto beans must sleep.
9999.8200         Supplier#000910128      MOZAMBIQUE            39160114    Manufacturer#4          xAaEvKV0rOyKYd                           26-738-711-8974
final, ironic dependencies are. slyly
9999.7900         Supplier#000722871      MOROCCO               11972867    Manufacturer#4          LbE wuB9WRGYkmClMcroCLHReXHf ZDYxnw      25-
519-939-9174 ironic ideas after the fluffily brave accounts integrate according to the final pa
9999.7400         Supplier#001151171      ETHIOPIA             37151170    Manufacturer#5          Bo6Ts7rzkgaB                             15-905-945-3677 regular,
ironic dolphins integrate furiously among the quickly slow instructions. carefully i
9999.6700         Supplier#002599109      MOROCCO               12349096    Manufacturer#4          DffaoE,eED5TEw mI2lWQyhViO               25-782-607-1479
furiously unusual platelets use fluffily pearls. furiously even accounts sleep
9999.6700         Supplier#002599109      MOROCCO               19099096    Manufacturer#2          DffaoE,eED5TEw mI2lWQyhViO               25-782-607-1479
furiously unusual platelets use fluffily pearls. furiously even accounts sleep
9999.6300         Supplier#002389031      MOZAMBIQUE            9889024    Manufacturer#1          tWJ,HpiXXkD7ZWfuO0Rdp adzSynimX          26-841-256-
7439 slyly regular dependencies alongside of the ironic, final dolphins cajole fluffi
9999.6300         Supplier#002389031      MOZAMBIQUE            24139006    Manufacturer#4          tWJ,HpiXXkD7ZWfuO0Rdp adzSynimX          26-841-256-
7439 slyly regular dependencies alongside of the ironic, final dolphins cajole fluffi
9999.6300         Supplier#002389031      MOZAMBIQUE            33889008    Manufacturer#5          tWJ,HpiXXkD7ZWfuO0Rdp adzSynimX          26-841-256-
7439 slyly regular dependencies alongside of the ironic, final dolphins cajole fluffi
9999.5100         Supplier#000932819      ETHIOPIA             49682770    Manufacturer#5          lAfbY1DjEjk0Jw49aihH1p S                 15-834-399-8134 ironic
ideas wake furiously. unusual deposits use against the carefully even packages. quick
9999.4500         Supplier#002424698      MOZAMBIQUE            43674683    Manufacturer#5          ZTbyGwPqwuvSdzjIi e5RJEsPefcj            26-210-618-9262
accounts snooze idly-- final excuses above the furiously enticing platelets nag care
.
.
.
9994.0500         Supplier#002913822      ETHIOPIA             56163803    Manufacturer#4          deO4,2RmC5sf5wyz5o0nfmnQz40d6j1W702r4pw  15-158-957-
8846 carefully special notornis haggle carefully according to
9994.0100         Supplier#001532080      KENYA                45032049    Manufacturer#5          x8QQa3AdkZoOWUIxZK79eo68irB03dWjDi       24-551-937-
4976 blithely unusual deposits cajole! blithely regular ideas promise
9993.4900         Supplier#002781203      MOROCCO               28281184    Manufacturer#3          T6kY5LxlGD7hfycooinOIHZrD6IcJb7kiahTmr   25-115-214-
7903 ironic platelets wake slyly fluffily regular requests. deposits doubt care
9993.2200         Supplier#002827026      MOZAMBIQUE            13327017    Manufacturer#1          Gx,lJ,ZythWN8eBMGMCanzqDjHeTkqySnVwEZPh1 26-
395-240-4855 carefully brave escapades wake quickly even instructions. slyly regular
9993.1900         Supplier#002049139      ALGERIA              19299132    Manufacturer#2          Ys52lJWNAm5ovyvt6Gt                      10-462-286-4378
regular, bold instructions sleep after the carefully ironic pinto beans. carefully f
9993.0100         Supplier#000662038      KENYA                30662037    Manufacturer#3          ZRVSl4VQ1gJavOjnrR                       24-539-319-2323 blithely
final ideas cajole after the quickly even requests. always bold packages are acco
9992.8200         Supplier#000699214      MOROCCO               24699213    Manufacturer#1          16NJa206A3ZdJa2DiiKdhvnHXivhcCNUF4       25-594-897-
6727 boldly regular deposits after the carefully regular pinto beans inte
9992.6200         Supplier#000392949      ALGERIA              13142936    Manufacturer#2          vLCdwooF3pKRAJgFf7nW5lwKNSTZipPw2        10-853-937-
3619 special, even accounts nag blithely against the quickly regular theodolites. un
9992.4300         Supplier#002629023      ALGERIA              24378998    Manufacturer#2          zbMxEfBeVyaJnfE                          10-421-142-9014 even
decoys cajole carefully about the attainments.
9992.4300         Supplier#002310865      ETHIOPIA             10560861    Manufacturer#1          lZhp9Nc5,d PrJLjS5c                      15-830-633-2526 carefully
unusual ideas boost slyly among the regular asymptotes. requests wake slyly
```

```
9992.4100       Supplier#002501466      ETHIOPIA        21251444    Manufacturer#2      rkc4ssM3,gkcDSvZiRr9RCIfLiN         15-299-490-3868
express, special dependencies use slyly across the final accounts. slyly final ideas wa
9992.3000       Supplier#000192816      MOZAMBIQUE          38442803    Manufacturer#2      SYyEb8bxtbNNHNPvhbhJeXpCwRXhdcFu, OPa    26-
839-160-1100 fluffily even pinto beans kindle slyly according to the unusual, pendi
9992.1900       Supplier#000345798      MOZAMBIQUE          27345797    Manufacturer#3      RBygVQv4LdeL0IaUWNwNM                   26-570-401-
8840 slyly express deposits are furiously against the quickly final requests.
9991.9900       Supplier#002609830      KENYA           39359790    Manufacturer#2      Grp1,dSMY7Z3mMLfHEoJ1OuHb0tzbeO4jbX      24-820-680-
9106 ironic, ironic patterns sleep furiously acr

(100 row(s) affected)

+++++++++++++++++++++++++
```

/* TPC_H Query 3 - Shipping Priority */

```
-- using 1002051338 as a seed to the RNG

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     < '1995-03-03' AND
        L_SHIPDATE      > '1995-03-03'
GROUP   BY      L_ORDERKEY,
                O_ORDERDATE,
                O_SHIPPRIORITY
ORDER   BY      REVENUE DESC,
                O_ORDERDATE
-----------
-----------
L_ORDERKEY  REVENUE             O_ORDERDATE             O_SHIPPRIORITY
----------- --------------------- ----------------------- --------------
265539204   502466.7526         1995-02-14 00:00:00.000 0
1149491235  480457.8901         1995-02-26 00:00:00.000 0
268564416   478059.3023         1995-02-24 00:00:00.000 0
841426821   477884.9506         1995-02-16 00:00:00.000 0
257589767   476019.8902         1995-02-20 00:00:00.000 0
781065317   475267.9386         1995-02-05 00:00:00.000 0
1437490755  474959.5562         1995-02-01 00:00:00.000 0
1780704326  471172.4472         1995-02-23 00:00:00.000 0
248634404   469527.6355         1995-02-04 00:00:00.000 0
173325283   468533.4626         1995-02-18 00:00:00.000 0

(10 row(s) affected)

+++++++++++++++++++++++++
```

/* TPC_H Query 4 - Order Priority Checking */

```
-- using 1002051338 as a seed to the RNG

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

```
O_ORDERPRIORITY ORDER_COUNT
--------------- -----------
1-URGENT        3159453
2-HIGH          3160970
3-MEDIUM        3160650
4-NOT SPECIFIED 3157297
5-LOW           3161787

(5 row(s) affected)

++++++++++++++++++++++++
```

/* TPC_H Query 5 - Local Supplier Volume */

```
-- using 1002051338 as a seed to the RNG

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             = 'EUROPE' AND
         O_ORDERDATE        >= '1996-01-01' AND
         O_ORDERDATE        < DATEADD(YY, 1, '1996-01-01')
GROUP    BY        N_NAME
ORDER    BY        REVENUE DESC
-----------
-----------
N_NAME                    REVENUE
------------------------- ---------------------
UNITED KINGDOM            16037516572.3503
ROMANIA                   15978935734.8804
FRANCE                    15977604861.4343
GERMANY                   15924213726.7344
RUSSIA                    15903601080.2593

(5 row(s) affected)

++++++++++++++++++++++++
```

/* TPC_H Query 6 - Forecasting Revenue Change */

```
-- using 1002051338 as a seed to the RNG

SELECT   SUM(L_EXTENDEDPRICE*L_DISCOUNT)         AS REVENUE
FROM     LINEITEM
WHERE    L_SHIPDATE         >= '1996-01-01' AND
         L_SHIPDATE         < dateadd (yy, 1, '1996-01-01') AND
         L_DISCOUNT         BETWEEN 0.08 - 0.01 AND 0.08 + 0.01 AND
         L_QUANTITY         < 24
-----------
-----------
REVENUE
---------------------
49491039792.0792

(1 row(s) affected)

++++++++++++++++++++++++
```

/* TPC_H Query 7 - Volume Shipping */

```
-- using 1002051338 as a seed to the RNG
```

```
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          = 'VIETNAM'         AND N2.N_NAME    = 'INDONESIA')
                                        OR
                                        (N1.N_NAME          = 'INDONESIA'       AND N2.N_NAME    = 'VIETNAM')
                              ) 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
------------------------ ------------------------ ----------- --------------------
INDONESIA                VIETNAM                  1995        15882980280.3769
INDONESIA                VIETNAM                  1996        15903033710.2730
VIETNAM                  INDONESIA                1995        15881381005.5999
VIETNAM                  INDONESIA                1996        15940940514.5451

(4 row(s) affected)

++++++++++++++++++++++++
```

/* TPC_H Query 8 - National Market Share */

```
-- using 1002051338 as a seed to the RNG

SELECT    O_YEAR,
          SUM(CASE            WHEN      NATION    = 'INDONESIA'
                              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              = 'ASIA' AND
```

```
                        S_NATIONKEY     = N2.N_NATIONKEY AND
                        O_ORDERDATE     BETWEEN '1995-01-01' AND '1996-12-31' AND
                        P_TYPE          = 'PROMO BRUSHED COPPER'
        )       AS      ALL_NATIONS
GROUP   BY      O_YEAR
ORDER   BY      O_YEAR
-----------
-----------
O_YEAR   MKT_SHARE
----------- ---------------------
1995     .0406
1996     .0396

(2 row(s) affected)

++++++++++++++++++++++++
```

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

```
-- using 1002051338 as a seed to the RNG

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 '%yellow%'
        )       AS PROFIT
GROUP   BY      NATION,
                O_YEAR
ORDER   BY      NATION,
                O_YEAR  DESC
-----------
-----------
NATION                  O_YEAR   SUM_PROFIT
------------------------- ----------- ---------------------
ALGERIA                 1998     6127841795.1057
ALGERIA                 1997     10500653502.3601
ALGERIA                 1996     10511435188.4813
ALGERIA                 1995     10446776130.5763
ALGERIA                 1994     10474089926.5481
ALGERIA                 1993     10472618621.7348
ALGERIA                 1992     10459400277.7922
ARGENTINA               1998     6158148937.0759
ARGENTINA               1997     10477462402.9577
ARGENTINA               1996     10509175963.3635
ARGENTINA               1995     10467195554.8310
.
.
.
UNITED STATES           1993     10385210228.2457
UNITED STATES           1992     10431901680.5262
VIETNAM                 1998     6120423435.5616
VIETNAM                 1997     10498852246.7952
VIETNAM                 1996     10511406895.5207
VIETNAM                 1995     10494930883.4235
VIETNAM                 1994     10495561598.8599
VIETNAM                 1993     10492459846.0566
VIETNAM                 1992     10512562093.2499
```

```
(175 row(s) affected)
+++++++++++++++++++++++++
```

/* TPC_H Query 10 - Returned Item Reporting */

```
-- using 1002051338 as a seed to the RNG

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       >= '1994-10-01'                        AND
         O_ORDERDATE       < dateadd(mm, 3, '1994-10-01') 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
----------- ------------------------ -------------------- -------------------- ------------------------ ---------------------------------------- -------------- ----------------------------------------
---------------------------------------------------------------------------
30211402    Customer#030211402       804898.3141          6068.0700            PERU                     rveP3HHp98WwyWs1uN no2A1ZiAgAgpQFADV4DjX 27-442-
548-6757 blithely final requests haggle boldly according to the even, s
1709105     Customer#001709105       799923.9523          4206.2400            CHINA                    LgnHXuBKgB                               28-315-764-9456 carefully
pending packages print across the dependencies. quickly express packages to th
35519824    Customer#035519824       786811.8517          7918.9000            VIETNAM                  uR5dU3KwWsAMvh                           31-474-708-9177 packages
after the pinto beans are furiously slyly unusual theodolite
21911671    Customer#021911671       758082.7080          7422.4600            KENYA                    vIVXoNnYtA3lKvuOZX                       24-123-335-3127 bold,
special requests affix. ironic asymptotes wak
11537764    Customer#011537764       753027.8173          6859.9600            KENYA                    ob8Fxody9ZQlBRUH4JeMxwgAgQo9iDElmvQrHcz  24-478-491-
6323 ironic platelets cajole blithely regular instructions. fluffily unusual platelets according to
9731938     Customer#009731938       746256.5649          904.9800             GERMANY                  BVuj9EtErrZJAAUlbfoZbMAUeZqiI            17-485-777-1792
packages cajole carefully silent accounts. fluffily even accounts ha
6606760     Customer#006606760       743786.0310          2472.5100            INDIA                    mjmuaB8LNCw1wZ0dUCbtwYtpnV               18-445-278-8789
express attainments hinder accounts. slyly regular excuses cajole along the carefully ironic excuses. blithely
25309906    Customer#025309906       743688.1759          9439.7400            PERU                     uf5anwWvBooXZaBmauneiJFMQIwta6ufObAZkwY  27-707-669-
5166 carefully final excuses use furiously. bold
27942976    Customer#027942976       743228.9679          570.3200             ARGENTINA                MOwPxE7SjAfERK2v 4S Vd6hVBJISvXSz        11-318-633-
4126 furiously ironic pinto beans along the bold platelets cajole quickl
27886042    Customer#027886042       733379.9219          2609.7200            UNITED STATES            7CmALmXQkxbO                             34-623-770-2176 fluffily
regular packages alongside of the care
13203130    Customer#013203130       733376.4554          6749.0400            GERMANY                  2zQO1lvZNj7EjTKdcgJG1YMt,xgeEA3kivHwPb2c 17-930-
162-1071 furiously bold foxes use carefully. quickly regular requests haggle carefully final requests. bold d
21986801    Customer#021986801       727336.5144          8161.3000            JAPAN                    vmt4nWiVSjrEYI,TL8o,S0rArxtWo0k,         22-932-942-8986
blithely sly requests wake carefully against the carefully express requests. ironic packages kindle furiously about
2745946     Customer#002745946       725256.8598          9734.9400            UNITED STATES            GbpiLZD7jJudnWusYgcSHJOUQ7gE,vkp2F22     34-168-
509-8733 carefully ironic dolphins sleep carefully. slyly ironic accounts nag carefully am
9605737     Customer#009605737       724421.1911          5538.0300            ROMANIA                  aCHIgNerugZa51Bv                         29-237-608-1771 even
requests boost fluffily across the pending, even accounts. slyly special
11296414    Customer#011296414       720824.5063          34.2100              ALGERIA                  IZNno8aKchSqaPCVKisOibw,a                10-211-603-7553
pending braids along the realms sleep carefully furiously express requests. ironic e
```

```
29807674    Customer#029807674        719183.1478           5937.2500             INDIA                  JJJjnzuIkZUQLNLzHnTyjsLm1LYib,          18-160-190-9763
regular instructions affix carefully furiously final deposits. pending dependencies serve blithely at the unusual re
35958946    Customer#035958946        715549.4571           892.8700              BRAZIL                 oGyRul Ec5Sa,SXPCVIivDEY               12-922-537-3083 slyly
express requests across the quickly even ideas affix carefully slyly pending packages. regular ideas ca
29227222    Customer#029227222        710041.9472           4655.4600             UNITED KINGDOM           CXPli68n3y                            33-789-424-1141 quickly
express pinto beans wake silently above the blithely thin platelets. express ideas
8716873     Customer#008716873        708187.8876           7164.9200             UNITED STATES            ZAIGNaubiQ7Zr2U,hj,JwknwMuZ5cFf8lQdePz,  34-625-886-
1269 carefully special excuses play fluffily through the special, express theodolites. fluffily unusua
13214047    Customer#013214047        706498.8957           7516.1600             MOZAMBIQUE             46l42o5E,Z                            26-644-553-7913 ironic
requests are blithely ironic foxes. even, bold pinto beans impress. carefully even packages are slyly? f

(20 row(s) affected)

++++++++++++++++++++++++
```

/* TPC_H Query 11 - Important Stock Indentification */

```
-- using 1002051338 as a seed to the RNG

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            = 'MOROCCO'
GROUP    BY       PS_PARTKEY
HAVING   SUM(PS_SUPPLYCOST*PS_AVAILQTY) >
                  (        SELECT   SUM(PS_SUPPLYCOST*PS_AVAILQTY) * 0.0000003333
                           FROM     PARTSUPP,
                                    SUPPLIER,
                                    NATION
                           WHERE    PS_SUPPKEY        = S_SUPPKEY       AND
                                    S_NATIONKEY       = N_NATIONKEY     AND
                                    N_NAME            = 'MOROCCO'
                  )
ORDER    BY       VALUE    DESC
-----------
-----------
PS_PARTKEY  VALUE
----------- ---------------------
28796439    23121943.9100
16550920    23016606.4600
3875836     22980858.1700
11336224    22775563.8200
19522022    22502753.7100
50717567    22284438.1400
14506636    22222237.3000
43497427    22010751.8000
34724498    21761375.8100
.
.
.
18769224    8017350.0300
4633003     8017346.8200
13608394    8017330.6500
51953763    8017327.0300
54970110    8017320.0000
32557871    8017316.5000
10920853    8017308.9700
50821399    8017307.2800
6783363     8017305.0000
6864415     8017303.7600

(279808 row(s) affected)

++++++++++++++++++++++++
```

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

```
-- using 1002051338 as a seed to the RNG
```

```
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 ('SHIP','MAIL')          AND
        L_COMMITDATE    < L_RECEIPTDATE             AND
        L_SHIPDATE      < L_COMMITDATE              AND
        L_RECEIPTDATE   >= '1997-01-01'                     AND
        L_RECEIPTDATE   < dateadd(yy, 1, '1997-01-01')
GROUP   BY      L_SHIPMODE
ORDER   BY      L_SHIPMODE
-----------
-----------
L_SHIPMODE HIGH_LINE_COUNT LOW_LINE_COUNT
---------- --------------- --------------
MAIL       1870754         2804809
SHIP       1868057         2808608

(2 row(s) affected)

++++++++++++++++++++++++
```

/* TPC_H Query 13 - Customer Distribution */

```
-- using 1002051338 as a seed to the RNG

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 '%unusual%deposits%'
                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
----------- -----------
C_COUNT     CUSTDIST
----------- -----------
0           15000238
10          2165168
9           2111312
11          1999386
8           1818896
12          1717235
19          1518452
20          1501624
.
.
.
37          379
38          134
39          63
40          24
41          5
44          1
```

```
43         1
42         1

(45 row(s) affected)

+++++++++++++++++++++++++
```

/* TPC_H Query 14 - Promotion Effect */

```
-- using 1002051338 as a seed to the RNG

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          >= '1997-09-01'                AND
         L_SHIPDATE          < dateadd(mm, 1, '1997-09-01')
-----------
-----------
PROMO_REVENUE
----------------------------------------
16.658531188964844

(1 row(s) affected)

+++++++++++++++++++++++++
```

/* TPC_H Query 15 - Create View for Top Supplier Query */

```
--using 1002051338 as a seed to the RNG

CREATE   VIEW REVENUE0 (SUPPLIER_NO, TOTAL_REVENUE)
AS
SELECT   L_SUPPKEY,
         SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT))
FROM     LINEITEM
WHERE    L_SHIPDATE          >= '1995-05-01' AND
         L_SHIPDATE          < dateadd(mm, 3, '1995-05-01')
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
----------- ------------------------- ---------------------------------------- --------------- ---------------------
1993996     Supplier#001993996        rc6OakpBvL5LZrVmy1X9cduSMlIotuHt 3       15-204-202-6798 2463249.4483

(1 row(s) affected)

+++++++++++++++++++++++++
```

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

```
-- using 1002051338 as a seed to the RNG

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#24'                                                 AND
          P_TYPE              NOT LIKE 'PROMO ANODIZED%'                                              AND
          P_SIZE              IN (39, 4, 17, 16, 30, 9, 2, 35)      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   PROMO PLATED TIN          35          1639
Brand#41   STANDARD PLATED TIN       17          1556
Brand#14   SMALL BRUSHED NICKEL      9           1553
Brand#53   STANDARD PLATED TIN       9           1542
Brand#41   PROMO BRUSHED COPPER      9           1532
Brand#33   STANDARD ANODIZED BRASS   30          1531
Brand#54   STANDARD BURNISHED NICKEL 30          1528
Brand#42   STANDARD BRUSHED NICKEL   4           1523
.
.
.
Brand#34   LARGE PLATED COPPER       39          1051
Brand#41   STANDARD ANODIZED BRASS   17          1047
Brand#35   MEDIUM POLISHED STEEL     4           1044
Brand#53   SMALL POLISHED TIN        35          1036
Brand#42   SMALL PLATED TIN          9           1034
Brand#23   LARGE PLATED STEEL        16          1028
Brand#42   SMALL BURNISHED BRASS     35          1024
Brand#53   ECONOMY PLATED NICKEL     39          1023
Brand#45   MEDIUM BRUSHED NICKEL     2           1000
Brand#22   PROMO BURNISHED STEEL     4           987

(27840 row(s) affected)

++++++++++++++++++++++++
```

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

```
-- using 1002051338 as a seed to the RNG

SELECT    SUM(L_EXTENDEDPRICE)/7.0                  AS AVG_YEARLY
FROM      LINEITEM,
          PART
WHERE     P_PARTKEY           = L_PARTKEY         AND
          P_BRAND                       = 'Brand#45'                  AND
          P_CONTAINER         = 'MED PKG'                   AND
          L_QUANTITY          <         (         SELECT    0.2 * AVG(L_QUANTITY)
                                                  FROM       LINEITEM
                                                  WHERE     L_PARTKEY           = P_PARTKEY
                                        )
-----------
-----------
AVG_YEARLY
------------------------
```

```
96593576.0000000

(1 row(s) affected)

++++++++++++++++++++++++
```

/* TPC_H Query 18 - Large Volume Customer */

```
-- using 1002051338 as a seed to the RNG

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) > 315
                                    )         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#005825251       5825251     1211370950  1995-10-01 00:00:00.000 594669.5300          330.0000
Customer#011659001       11659001    1607224230  1994-03-24 00:00:00.000 588524.1500          334.0000
Customer#027622552       27622552    1009170407  1992-03-12 00:00:00.000 586192.2500          325.0000
Customer#028281745       28281745    48881602    1993-09-19 00:00:00.000 580638.2500          327.0000
Customer#032203250       32203250    1741987878  1995-03-20 00:00:00.000 575999.5000          318.0000
Customer#040181347       40181347    1503592230  1996-07-15 00:00:00.000 574034.3200          318.0000
.
.
.
Customer#025929247       25929247    84927619    1997-06-29 00:00:00.000 534802.0200          316.0000
Customer#019184890       19184890    410876964   1994-11-15 00:00:00.000 534739.2900          324.0000
Customer#039899956       39899956    1622003009  1997-02-06 00:00:00.000 534437.2800          321.0000
Customer#023274065       23274065    1009666115  1998-03-06 00:00:00.000 533873.9500          322.0000
Customer#005669002       5669002     979742369   1996-12-07 00:00:00.000 533814.3500          317.0000
Customer#029443667       29443667    20971013    1997-08-12 00:00:00.000 533204.4200          322.0000
Customer#010479067       10479067    182898470   1996-01-09 00:00:00.000 532934.9700          318.0000
Customer#032871172       32871172    1312277635  1995-12-18 00:00:00.000 532734.9600          316.0000

(100 row(s) affected)

++++++++++++++++++++++++
```

/* TPC_H Query 19 - Discounted Revenue */

```
-- using 1002051338 as a seed to the RNG

SELECT   SUM(L_EXTENDEDPRICE* (1 - L_DISCOUNT))      AS REVENUE
FROM     LINEITEM,
         PART
WHERE    (        P_PARTKEY         = L_PARTKEY                                           AND
                  P_BRAND                     = 'Brand#32'                                          AND
                  P_CONTAINER       IN ( 'SM CASE', 'SM BOX', 'SM PACK', 'SM PKG')        AND
                  L_QUANTITY        >= 7                                                  AND
                  L_QUANTITY        <= 7 + 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#21'                                            AND
                P_CONTAINER     IN ( 'MED BAG', 'MED BOX', 'MED PKG', 'MED PACK')       AND
                L_QUANTITY      >= 19                                                   AND
                L_QUANTITY      <= 19 + 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#35'                                            AND
                P_CONTAINER     IN ( 'LG CASE', 'LG BOX', 'LG PACK', 'LG PKG')          AND
                L_QUANTITY      >= 21                                                   AND
                L_QUANTITY      <= 21 + 10                                      AND
                P_SIZE          BETWEEN 1 AND 15                                        AND
                L_SHIPMODE      IN ('AIR', 'AIR REG')                           AND
                L_SHIPINSTRUCT  = 'DELIVER IN PERSON'
        )
-----------
-----------
REVENUE
--------------------
1122246135.9660

(1 row(s) affected)

+++++++++++++++++++++++++
```

/* TPC_H Query 20 - Potential Part Promotion */

```
-- using 1002051338 as a seed to the RNG

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 'floral%'
                                                                )       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          = 'ALGERIA'
ORDER   BY      S_NAME
-----------
-----------
S_NAME                  S_ADDRESS
------------------------- ----------------------------------------
Supplier#000000024      C4nPvLrVmKPPabFCj
Supplier#000000028      GBhvoRh,7YIN V
Supplier#000000118      BYtvNtFpQAHHoBFWF
Supplier#000000281      A2sesSQAAj6wvPPKL X4caRp,O
Supplier#000000327      MoC7Jc7oThpZ34HmJPKuUbOZwOyPOb1ksGlvT8o
Supplier#000000370      yyNSJAG9UXcWit4SeMkEIrNcdVq5
Supplier#000000425      a KnEGf,bqEnGd2Wd9Tl
Supplier#000000454      K8p1uXD3L,L
Supplier#000000474      USHBMdX8iFodU
```

```
Supplier#000000476   ZvT qI2gMbh
.
.
.
Supplier#002999555   snn8WNH9Lhisb3XTNKMB5pmZhDhiFc3RgAPqRw
Supplier#002999567   24MtId3p,zaelbsCqSMz VEfX
Supplier#002999572   bYzd90wht83JDu,Mk
Supplier#002999689   1T3dRxVFPv7JGM0gGlgSP gHz
Supplier#002999711   9RWA6J,47kHGQLeOayxWVd1kzcX
Supplier#002999754   s4i2KhNan0Pbkh4R6BbvAbe4uXr1
Supplier#002999775   atR2fn5vFQx2R8DSObCFm1teowz
Supplier#002999781   H1EXipITUN4ylr,h7 Ozx7KgVmvu51m9S95oP
Supplier#002999943   B4hq  ucUIwnGW68s UxQSigl

(55843 row(s) affected)

++++++++++++++++++++++++
```

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

```
-- using 1002051338 as a seed to the RNG

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          = 'INDIA'
GROUP   BY      S_NAME
ORDER   BY      NUMWAIT         DESC,
                S_NAME
-----------
-----------
S_NAME          NUMWAIT
------------------------- -----------
Supplier#000299923   29
Supplier#001155800   26
Supplier#001968096   26
Supplier#002099414   26
Supplier#000091017   25
Supplier#000386404   25
Supplier#001456592   25
Supplier#001886774   25
Supplier#002100032   25
Supplier#002800117   25
.
.
.
Supplier#001367862   22
Supplier#001386524   22
Supplier#001510500   22
Supplier#001605077   22
Supplier#001645237   22
Supplier#001655781   22
```

```
Supplier#001676017      22
Supplier#001748511      22
Supplier#001753886      22
Supplier#001760347      22

(100 row(s) affected)

++++++++++++++++++++++++
```

/* TPC_H Query 22 - Global Sales Opportunity */

```
-- using 1002051338 as a seed to the RNG

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        ('10', '21', '15', '16', '26', '17', '29') AND
                           C_ACCTBAL                 >         (         SELECT   AVG(C_ACCTBAL)
                                                                         FROM     CUSTOMER
                                                                         WHERE    C_ACCTBAL          > 0.00     AND
                                                                                  SUBSTRING(C_PHONE,1,2)    IN         ('10', '21', '15', '16',
'26', '17', '29')
                                                               )         AND
                           NOT EXISTS         (        SELECT   *
                                                       FROM     ORDERS
                                                       WHERE    O_CUSTKEY          = C_CUSTKEY
                                              )
         )        AS CUSTSALE
GROUP    BY       CNTRYCODE
ORDER    BY       CNTRYCODE
-----------
-----------
CNTRYCODE NUMCUST     TOTACCTBAL
--------- ----------- ---------------------
10        271811      2038639859.6900
15        272918      2047301445.9900
16        272216      2040336250.1600
17        273120      2048483539.9400
21        271429      2036250318.9600
26        272169      2041028685.8600
29        273075      2048480352.7200

(7 row(s) affected)
```

APPENDIX D: Seed & Query Substitution

Parameters+++

Substitution Parameters for Stream 00

++

-- using 1002051338 as a seed to the RNG

```
1         96
2         6         TIN       AFRICA
3         BUILDING            1995-03-03
4         1993-07-01
5         EUROPE    1996-01-01
6         1996-01-01          0.08      24
7         VIETNAM             INDONESIA
8         INDONESIA           ASIA      PROMO BRUSHED COPPER
9         yellow
10        1994-10-01
11        MOROCCO             0.0000003333
12        SHIP      MAIL      1997-01-01
13        unusual   deposits
14        1997-09-01
15        1995-05-01
16        Brand#24  PROMO ANODIZED39        4         17        16        30        9         2         35
17        Brand#45  MED PKG
18        315
19        Brand#32  Brand#21  Brand#35  7         19        21
20        floral    1994-01-01          ALGERIA
21        INDIA
22        10        21        15        16        26        17        29
```

++

Substitution Parameters for Stream 01

++

-- using 1002051339 as a seed to the RNG

```
21        ALGERIA
3         MACHINERY           1995-03-20
18        312
5         MIDDLE EAST         1996-01-01
11        CANADA 0.0000003333
7         JORDAN    ARGENTINA
6         1996-01-01          0.06      24
20        powder    1993-01-01          MOROCCO
17        Brand#52  JUMBO CASE
```

```
12        FOB       MAIL      1997-01-01
16        Brand#55  SMALL BURNISHED     3         10        18        16        38        22        14        47
15        1993-02-01
13        unusual   deposits
10        1993-08-01
2         44        STEEL     EUROPE
8         ARGENTINA           AMERICA             PROMO PLATED COPPER
14        1997-12-01
19        Brand#44  Brand#14  Brand#24  2         20        28
9         thistle
22        34        24        20        13        31        32        11
1         105
4         1996-02-01
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Substitution Parameters for Stream 02

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

-- using 1002051340 as a seed to the RNG

```
6         1996-01-01          0.03      25
17        Brand#54  JUMBO BAG
14        1993-04-01
16        Brand#45  LARGE POLISHED      13        19        35        11        6         15        5         34
19        Brand#41  Brand#42  Brand#23  7         10        24
10        1994-05-01
9         slate
2         31        BRASS     AFRICA
15        1995-08-01
8         CHINA     ASIA      PROMO ANODIZED COPPER
5         AFRICA    1996-01-01
22        25        18        22        23        13        20        34
12        TRUCK     MAIL      1993-01-01
7         ETHIOPIA            CHINA
13        unusual   packages
18        314
1         113
4         1993-11-01
20        burnished 1996-01-01          ETHIOPIA
3         BUILDING            1995-03-05
11        MOZAMBIQUE          0.0000003333
21        PERU
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Substitution Parameters for Stream 03

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

-- using 1002051341 as a seed to the RNG

```
8         IRAN      MIDDLE EAST         ECONOMY BRUSHED TIN
```

```
5         AMERICA             1996-01-01
4         1996-06-01
6         1996-01-01          0.09      24
17        Brand#55  JUMBO PKG
7         RUSSIA    IRAN
1         60
18        315
22        29        15        13        26        18        16        20
14        1993-07-01
9         saddle
10        1993-02-01
15        1993-05-01
11        EGYPT     0.0000003333
20        metallic  1994-01-01          ROMANIA
2         19        NICKEL    EUROPE
21        INDONESIA
19        Brand#43  Brand#24  Brand#22  2         11        20
13        unusual   packages
16        Brand#25  PROMO BRUSHED       38        28        32        1         13        8         34        7
12        RAIL      FOB       1993-01-01
3         HOUSEHOLD           1995-03-22
```

++

Substitution Parameters for Stream 04

++

-- using 1002051342 as a seed to the RNG

```
5         ASIA      1997-01-01
21        ARGENTINA
14        1993-10-01
19        Brand#55  Brand#12  Brand#11  8         12        28
15        1995-12-01
17        Brand#52  WRAP CASE
12        AIR       FOB       1993-01-01
6         1997-01-01          0.06      24
4         1994-02-01
9         puff
8         BRAZIL    AMERICA             ECONOMY PLATED TIN
16        Brand#55  MEDIUM BURNISHED              41        44        4         2         10        13        36        3
11        PERU      0.0000003333
2         7         TIN       AMERICA
10        1993-11-01
18        313
1         68
13        unusual   packages
7         KENYA     BRAZIL
```

```
22        19        14        11        21        10        28        20
3         BUILDING            1995-03-07
20        wheat     1993-01-01          INDONESIA
```

++

Substitution Parameters for Stream 05

++

-- using 1002051343 as a seed to the RNG

```
21        CHINA
15        1993-08-01
4         1996-09-01
6         1997-01-01          0.04      25
7         FRANCE    ROMANIA
16        Brand#45  ECONOMY PLATED                21        5         12        49        22        17        26        45
19        Brand#52  Brand#45  Brand#11  3         13        24
18        315
14        1994-01-01
22        22        12        18        30        17        20        19
11        ETHIOPIA            0.0000003333
13        express   packages
3         HOUSEHOLD           1995-03-24
1         76
2         45        COPPER    EUROPE
5         EUROPE    1997-01-01
8         ROMANIA             EUROPE    ECONOMY ANODIZED TIN
20        honeydew  1996-01-01          UNITED STATES
12        REG AIR   FOB       1993-01-01
17        Brand#54  WRAP BAG
10        1994-09-01
9         papaya
```

++

Substitution Parameters for Stream 06

++

-- using 1002051344 as a seed to the RNG

```
10        1993-06-01
3         AUTOMOBILE          1995-03-09
15        1996-03-01
13        express   packages
6         1997-01-01          0.09      24
8         IRAQ      MIDDLE EAST         LARGE POLISHED TIN
9         navajo
7         UNITED KINGDOM IRAQ
4         1994-06-01
11        CHINA     0.0000003333
```

22	27	16	32	18	34	17	25					
18	312											
12	SHIP	FOB	1994-01-01									
1	84											
5	MIDDLE EAST		1997-01-01									
16	Brand#25	STANDARD BRUSHED			21	12	5	44	47	45	8	20
2	32	BRASS	AMERICA									
14	1994-04-01											
19	Brand#55	Brand#33	Brand#15	8	14	20						
20	salmon	1995-01-01		JORDAN								
17	Brand#51	WRAP PKG										
21	IRAQ											

APPENDIX E: StepMaster Code

This section lists VB code for StepMaster.

cArrConstraints.cls

```
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(errCreateNewConstraintVersionFailed)

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
      ' 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(gstrEmptyString, 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

   lngDeleteElement = QueryConstraintIndex(cOldConstraint.ConstraintId)

   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
    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
```

cArrParameters.cls

```
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 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)
        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
```

cArrSteps.cls

```
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)

    Validate cModifiedStep

    Call mcarrSteps.Modify(cModifiedStep)

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
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 Then
            ShowError errStepLabelUnique
            On Error GoTo 0
            Err.Raise vbObjectError + errValidateFailed, _
                mstrModuleName & "Validate", _
                LoadResString(errValidateFailed)
        End If
    Next lngIndex

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)

    Validate cStepToAdd

    Set cStepToAdd.NodeDB = mcarrSteps.NodeDB

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

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

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
    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
'    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
```

cConnDtl.cls

```
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
   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_NAME & " = [c_name]" & _
      " and " & FLD_ID_CONN_NAME & " <> [c_id]"

   Set qy = dbsAttTool.CreateQueryDef(gstrEmptyString, 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)

    ' 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
```

cConnDtls.cls

```
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

    ' 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
```

cCONNECTION.CLS

```
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

            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
    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
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
```

cConnections.cls

```
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

    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
```

cCONSTRAINT.CLS

```
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
    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(gstrEmptyString, 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()
    ' 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(gstrEmptyString, 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(gstrEmptyString, 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
        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
```

cFailedStep.cls

```
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
```

cFailedSteps.cls

```
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

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
```

cFileInfo.cls

```
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
```

cFileSM.cls

```
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

            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
```

cGlobalStep.cls

```
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_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


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

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

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
```

cInstance.cls

```
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

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)
    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
   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
```

cInstances.cls

```
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)

    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
```

CITERATOR.CLS

```
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

            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
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
```

cManager.cls

```
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 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)

   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

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
```

```
End Property


    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

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
```

cNode.cls

```
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
```

cNodeCollections.cls

```
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)

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
```

COMMON.BAS

```
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
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
    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
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

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(lngWorkspaceId, _
          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(TranslateStepLabel, "/",
gstrUnderscore)
  TranslateStepLabel = cTempStr.ReplaceSubString(TranslateStepLabel, ":",
gstrUnderscore)
  TranslateStepLabel = cTempStr.ReplaceSubString(TranslateStepLabel, "*",
gstrUnderscore)
  TranslateStepLabel = cTempStr.ReplaceSubString(TranslateStepLabel, "?",
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)

  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
```

CONNDTLCOMMON.BAS

```
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_NAME))
    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
```

CONNNECTIONCOMMON.BAS

```
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
```

cParameter.cls

```
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
    ' 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

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(mstrParameterName) & _
'          ", " & mFieldValue.MakeStringFieldValid(mstrParameterValue) & " ) "
'    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

    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 = "Parameter_id"
    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
```

cRunColIt.cls

```
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
```

cRunColIt.cls

```
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
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 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 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
    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
        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

    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
        Case 0
            Set cExecStep1 = Nothing
        Case 1
            Set cExecStep2 = Nothing
        Case 2
            Set cExecStep3 = Nothing
        Case 3
            Set cExecStep4 = Nothing
        Case 4
            Set cExecStep5 = Nothing
        Case 5
            Set cExecStep6 = Nothing
        Case 6
            Set cExecStep7 = Nothing
        Case 7
            Set cExecStep8 = Nothing
        Case 8
            Set cExecStep9 = Nothing
        Case 9
            Set cExecStep10 = Nothing
        Case 10
            Set cExecStep11 = Nothing
        Case 11
            Set cExecStep12 = Nothing
        Case 12
            Set cExecStep13 = Nothing
        Case 13
            Set cExecStep14 = Nothing
        Case 14
            Set cExecStep15 = Nothing
        Case 15
            Set cExecStep16 = 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.InstanceId)
                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).ParentInstanceId)
            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

    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)
    cNewInstance.DegreeParallelism =
SubstituteParameters(cExecStep.DegreeParallelism, _
            cExecStep.WorkspaceId, WspParameters:=mcParameters)

    If mcNavSteps.HasChild(StepKey:=cNewInstance.Key) Then
        Set nodChild = mcNavSteps.ChildStep(StepKey:=cNewInstance.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.StepId)
        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.ParentInstanceId)

        ' 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.ParentInstanceId)
        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, _
            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(cParentNode.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.ParentStepId) 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,
    ' 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.ParentInstanceId)

    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.ParentInstanceId)

                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.ParentInstanceId)
    ' Decrement the number of running processes on the
    ' parent by 1
    Call cParentInstance.ChildTerminated(cTermInstance.Step.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.Step.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.ParentInstanceId)
                If cParentInstance.IsRunning Then
                    cNextInst.AllStarted = True
                Else
                    ' No more sub-steps to execute
                    Call cNextInst.ChildCompleted(cParentInstance.Step.StepId)
                    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.StepId) 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
         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.ParentInstanceId)
    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
        Case 0
            Set cExecStep1 = New cRunStep
            Set InitExecStep = cExecStep1
        Case 1
            Set cExecStep2 = New cRunStep
            Set InitExecStep = cExecStep2
        Case 2
            Set cExecStep3 = New cRunStep
            Set InitExecStep = cExecStep3
        Case 3
            Set cExecStep4 = New cRunStep
            Set InitExecStep = cExecStep4
        Case 4
            Set cExecStep5 = New cRunStep
            Set InitExecStep = cExecStep5
        Case 5
            Set cExecStep6 = New cRunStep
            Set InitExecStep = cExecStep6
        Case 6
            Set cExecStep7 = New cRunStep
            Set InitExecStep = cExecStep7
        Case 7
            Set cExecStep8 = New cRunStep
            Set InitExecStep = cExecStep8
        Case 8
            Set cExecStep9 = New cRunStep
            Set InitExecStep = cExecStep9
        Case 9
            Set cExecStep10 = New cRunStep
            Set InitExecStep = cExecStep10
        Case 10
            Set cExecStep11 = New cRunStep
            Set InitExecStep = cExecStep11
        Case 11
            Set cExecStep12 = New cRunStep
            Set InitExecStep = cExecStep12
        Case 12
            Set cExecStep13 = New cRunStep
            Set InitExecStep = cExecStep13
        Case 13
            Set cExecStep14 = New cRunStep
            Set InitExecStep = cExecStep14
        Case 14
            Set cExecStep15 = New cRunStep
            Set InitExecStep = cExecStep15
        Case 15
            Set cExecStep16 = New cRunStep
            Set InitExecStep = cExecStep16
        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:=mstrRootKey) Then
            Set cTempStep = mcNavSteps.ChildStep(StepKey:=mstrRootKey)
            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:=mstrCurBranchRoot)
         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
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.StepId, 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.ParentInstanceId)
   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.ParentInstanceId)
   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 & 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)

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

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, _
      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(cStepRecord 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 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

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)

End Sub

Private Sub mcTermSteps_TermStepExists(cStepDetails As cTermStep)

   Call RunNextStep(cStepDetails.TimeComplete, cStepDetails.Index, _
          cStepDetails.InstanceId, cStepDetails.ExecutionStatus)

End Sub
```

cRunItDetails.cls

```
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
```

```
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.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)
   WspLog = strWspLog
End Sub
```

cRunStep.cls

```
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
    ' 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
      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)
    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(BuildCommandString(), 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(BuildCommandString(), 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.StepTextFile, _
                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:
   ' 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)

   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
```

cRunWorkspace.cls

```
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)
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.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"

    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

   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 identifier using the sequence class
    Set cRunSeq = New cSequence
    Set cRunSeq.IdDatabase = dbsAttTool
    cRunSeq.IdentifierColumn = "run_id"
    mlngRunId = cRunSeq.Identifier
    Set cRunSeq = Nothing

    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
    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)
    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
```

cSequence.cls

```
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 ) values ( " & _
            "[w_id], [p_id], [s_id], [c_id], [r_id], [conn_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
   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
```

cStack.cls

```
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
```

cStep.cls

```
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
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)
    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)

    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(mstrStepTextFile)
    strInsert = strInsert & ", " & mFieldValue.MakeStringFieldValid(mstrStepText)
    strInsert = strInsert & ", " & mFieldValue.MakeStringFieldValid(mstrStartDir)

    strInsert = strInsert & ", " & Str(mlngParentStepId) & _
        ", " & mFieldValue.MakeStringFieldValid(mstrParentVersionNo) & _
        ", " & Str(mintSequenceNo) & _
        ", " & Str(mblnEnabledFlag) & ", " & Str(mintStepLevel)

    strInsert = strInsert & ", " &
mFieldValue.MakeStringFieldValid(mstrDegreeParallelism)
    strInsert = strInsert & ", " & Str(mintExecutionMechanism)

    strInsert = strInsert & ", " & mFieldValue.MakeStringFieldValid(mstrFailureDetails) &
_
        ", " & Str(mintContinuationCriteria) & _
        ", " & Str(mblnGlobalFlag) & _
        ", " & Str(mblnArchivedFlag)

    strInsert = strInsert & ", " & mFieldValue.MakeStringFieldValid(mstrOutputFile)
'    strInsert = strInsert & ", " & mFieldValue.MakeStringFieldValid(mstrLogFile)
    strInsert = strInsert & ", " & mFieldValue.MakeStringFieldValid(mstrErrorFile)
    strInsert = strInsert & ", " & mFieldValue.MakeStringFieldValid(mstrIteratorName)

    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(gstrEmptyString, 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(mstrVersionNo)

    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 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


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(mstrStepTextFile)
    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(mstrParentVersionNo) & _
        " , sequence_no = " & Str(mintSequenceNo) & _
        " , step_level = " & Str(mintStepLevel) & _
        " , enabled_flag = " & Str(mblnEnabledFlag) & _
        " , degree_parallelism = " &
mFieldValue.MakeStringFieldValid(mstrDegreeParallelism) & _
        " , execution_mechanism = " & Str(mintExecutionMechanism) & _
        " , failure_details = " & mFieldValue.MakeStringFieldValid(mstrFailureDetails) & _
        " , continuation_criteria = " & Str(mintContinuationCriteria) & _
        " , global_flag = " & Str(mblnGlobalFlag) & _
        " , archived_flag = " & Str(mblnArchivedFlag) & _
        " , output_file_name = " & mFieldValue.MakeStringFieldValid(mstrOutputFile) & _
        " , error_file_name = " & mFieldValue.MakeStringFieldValid(mstrErrorFile) & _
        " , iterator_name = " & mFieldValue.MakeStringFieldValid(mstrIteratorName)

'        " , log_file_name = " & mFieldValue.MakeStringFieldValid(mstrLogFile) & _

    strUpdate = strUpdate & " where step_id = " & Str(mlngStepId) & _
        " and version_no = " & mFieldValue.MakeStringFieldValid(mstrVersionNo)

    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(gstrEmptyString, 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
   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, _
      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
```

cStepTree.cls

```
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
```

cStringSM.cls

```
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
        ' 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
```

cSubStep.cls

```
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
    ' 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
'       ' This procedure depends on the fact that the iterator type
'       ' hasn't been initialized - it may well have been, though
'       ' Try to modify the check later.
'       If mcLastIterator.IteratorType = 0 Then
'           ' The iterator details have not been initialized on the
'           ' run node for the step - call a procedure to carry out
'           ' the initialization
'           BugMessage "Initialize later happens!!!"
'           Call InitializeIt(cStepRec, RunParams, vntIterators)
'       End If
'
'       ' mcLastIterator will be set to Nothing if no iterators
'       ' have been defined for the step
'       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

'       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

      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
```

cSubSteps.cls

```
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
```

cTermProcess.cls

```
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

    If bTermProcessExists Then
        RaiseEvent TermProcessExists
    End If

    moTimer.Enabled = False
    bTermProcessExists = False

    Exit Sub

moTimer_TimerErr:
    LogErrors Errors

End Sub
```

cTermStep.cls

```
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
```

cTermSteps.cls

```
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
```

cTimer.cls

```
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
End Sub

Friend Sub Tick()
   RaiseEvent Timer
End Sub
```

cVBErrors.cls

```
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
```

cVector.cls

```
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)

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
```

cVectorLng.cls

```
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)
   ' 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
```

cVectorStr.cls

```
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
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()

    Call Clear

End Sub
```

cWorker.cls

```
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 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

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

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

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
```

cWorkspace.cls

```
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)
    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
    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(gstrEmptyString, 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(mstrWorkspaceName) & _
'            ", " & 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

'    strUpdate = "update att_workspaces " & _
'        " set workspace_name = " & _
'        mFieldValue.MakeStringFieldValid(mstrWorkspaceName) & _
'        ", 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
```

DatabaseSM.bas

```
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_DBFILE 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 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_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)"

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_DBFILE, 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
   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

               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) & _
        " 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(gstrEmptyString) & _
         " 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 cTempStr As New cStringSM
   Dim lId As Long
   Dim rTemp As DAO.Recordset
   Dim rParam As DAO.Recordset
   Dim cTempSeq As cSequence

   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_SERVER_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
   Set cTempSeq = New cSequence
   Set cTempSeq.IdDatabase = dbFile
   cTempSeq.IdentifierColumn = "parameter_id"

   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(PARAM_DEFAULT_DIR)
         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) & _
                  " 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(PARAM_DEFAULT_DIR) & ", " & _
                  cTempStr.MakeStringFieldValid(gstrEmptyString) & ", " & _
                  cTempStr.MakeStringFieldValid(PARAM_DEFAULT_DIR_DESC) & ", "
& _
                  CStr(gintParameterBuiltIn) & _
                  " ) "
         End If
         dbFile.Execute sBuf, dbFailOnError
         rParam.Close

         rTemp.MoveNext
      Wend
   End If
   rTemp.Close

   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 & _
         "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_SUFFIX
   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_NAME & _
         ", " & FLD_CONN_DTL_CONNECTION_STRING & _
         ", " & 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_CONNECTION_NAME) &
CONNECTION_STRINGS_TO_NAME_SUFFIX
         qy.Parameters("c_str").Value =
rTemp.Fields(FLD_CONN_STR_CONNECTION_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_MECHANISM).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_SUFFIX

               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 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)

  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 gsVersion243
         Call Upgrade243to25(UpgradeWsp, dbFile, gsVersion25)

      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)

      Case gsVersion23
         Call Upgrade23to24(UpgradeWsp, dbFile, gsVersion24)
         Call Upgrade242to243(UpgradeWsp, dbFile, gsVersion242)
         Call Upgrade243to25(UpgradeWsp, dbFile, gsVersion25)

      Case gsVersion21
         Call Upgrade21to23(UpgradeWsp, dbFile, gsVersion23)
         Call Upgrade23to24(UpgradeWsp, dbFile, gsVersion24)
         Call Upgrade242to243(UpgradeWsp, dbFile, gsVersion242)
         Call Upgrade243to25(UpgradeWsp, dbFile, gsVersion25)

      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)

      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)

   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 = 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
```

DEBUGSM.BAS

```
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_SYSTEM Or _
            FORMAT_MESSAGE_IGNORE_INSERTS, _
            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 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
```

FILECOMMON.BAS

```
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"
    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_FLAGS =
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_FILTER = "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
```

ITERATORCOMMON.BAS

```
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 ) ) ) " & _
        " 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(gstrEmptyString, 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
```

MsgConfirm.bas

```
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
```

OpenFiles.bas

```
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
    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

        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
```

ParameterCommon.bas

```
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(rstWorkSpaceParameters 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
```

PUBLIC.BAS

```
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
   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
```

RunCommon.bas

```
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

    If Not StringEmpty(cInstanceRec.Step.IteratorName) Then
      BugAssert cStepIt.Count > 0, "Iterator Count is greater " & _
          "than zero for a step that has an iterator defined."
      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

  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
```

cRunInst.cls

```
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
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 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
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

    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
            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
    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
    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
  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.InstanceId)
                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).ParentInstanceId)
            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

    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)
    cNewInstance.DegreeParallelism =
SubstituteParameters(cExecStep.DegreeParallelism, _
            cExecStep.WorkspaceId, WspParameters:=mcParameters)

    If mcNavSteps.HasChild(StepKey:=cNewInstance.Key) Then
        Set nodChild = mcNavSteps.ChildStep(StepKey:=cNewInstance.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.StepId)
        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.ParentInstanceId)

        ' 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.ParentInstanceId)
        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, _
            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(cParentNode.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.ParentStepId) 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,
    ' 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.ParentInstanceId)

    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.ParentInstanceId)

            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.ParentInstanceId)

   ' Decrement the number of running processes on the
   ' parent by 1
   Call cParentInstance.ChildTerminated(cTermInstance.Step.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.Step.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.ParentInstanceId)
            If cParentInstance.IsRunning Then
               cNextInst.AllStarted = True
            Else
               ' No more sub-steps to execute
               Call cNextInst.ChildCompleted(cParentInstance.Step.StepId)
               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.StepId) 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
            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.ParentInstanceId)
    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
            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
    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:=mstrRootKey) Then
            Set cTempStep = mcNavSteps.ChildStep(StepKey:=mstrRootKey)
```

```
            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:=mstrCurBranchRoot)
            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
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.StepId, 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.ParentInstanceId)
    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.ParentInstanceId)
   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 & 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)

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

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, _
        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(cStepRecord 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)

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


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)

    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(cStepRecord 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)

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)

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(cStepRecord 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)

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)

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(cStepRecord 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

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

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, _
        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, _
        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(cStepRecord 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, _
        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, _
        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(cStepRecord 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, _
        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)

    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(cStepRecord 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)

   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)

    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(cStepRecord 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)

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)

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(cStepRecord 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)

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)

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(cStepRecord 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

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

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, _
       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, _
       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(cStepRecord 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, _
        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, _
        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(cStepRecord 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, _
        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)

    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

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)

End Sub

Private Sub mcTermSteps_TermStepExists(cStepDetails As cTermStep)

   Call RunNextStep(cStepDetails.TimeComplete, cStepDetails.Index, _
         cStepDetails.InstanceId, cStepDetails.ExecutionStatus)

End Sub
```

RunInstHelper.bas

```
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
```

ShellSM.bas

```
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
```

SMErr.bas

```
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
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
    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
```

SortSM.bas

```
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
```

STARTUP.BAS

```
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

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(gintOnFailureCompleteSiblings) = "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.tblConstraintCommands)

#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
```

STEPCOMMON.BAS

```
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

' 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
   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"))

        ' 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
```

TIMERSM.BAS

```
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
```

TOOLSCOMMON.BAS

```
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 " & _
");"

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, " & _
       "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
```

WindowsApiCommon.bas

```
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
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
  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
```

WorkspaceCommon.bas

```
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
    Err.Raise vbObjectError + errGetWorkspaceDetailsFailed, _
            gstrSource, _
            LoadResString(errGetWorkspaceDetailsFailed)

End Function

Public Sub ReadStepsInWorkspace(rstStepsInWorkSpace 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
            ' 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.ItemData(frmWorkspaceOpen.lstWorkspaces.New
Index) = _
                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(gstrEmptyString, strSql)
    qyWspParams.Parameters("w_id").Value = lngWorkspaceId

    Set rstWorkSpaceParameters = qyWspParams.OpenRecordset(dbOpenSnapshot)

    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.

    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
```

The listings in this section implement the SMTime module.

SMTime.cpp

```
// SMTime.cpp : Implementation of DLL Exports.
//
//          Microsoft TPC-H Kit Ver. 1.00
//          Copyright Microsoft, 1999
//          All Rights Reserved
//
//  Contact:   Reshma Tharamal (reshmat@microsoft.com)
//

// 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);
}
```

SMTime.def

```
; SMTime.def : Declares the module parameters.

LIBRARY      "SMTime.DLL"

EXPORTS
                DllCanUnloadNow     @1 PRIVATE
                DllGetClassObject   @2 PRIVATE
                DllRegisterServer   @3 PRIVATE
                DllUnregisterServer        @4 PRIVATE
                Get64BitTime               @5
                SMTime_JulianToTime  @6
```

SMTime.idl

```
// SMTime.idl : IDL source for SMTime.dll
//
//            Microsoft TPC-H Kit Ver. 1.00
//            Copyright Microsoft, 1999
//            All Rights Reserved
//
//  Contact:   Reshma Tharamal (reshmat@microsoft.com)
//

// This file will be processed by the MIDL tool to
// produce the type library (SMTime.tlb) and marshalling code.

import "oaidl.idl";
import "ocidl.idl";
                [
                                object,
                                uuid(1A6D0AE4-8528-453B-B8E3-8DAD1F0561B7),
                                dual,
                                helpstring("ISMTimer Interface"),
                                pointer_default(unique)
                ]
                interface ISMTimer : IDispatch
                {
                                [id(1), helpstring("method Start")] HRESULT Start();
                                [id(2), helpstring("method Stop")] HRESULT Stop(CURRENCY *pElapsedTime);
                                [propget, id(3), helpstring("property Running")] HRESULT Running([out, retval] BOOL *pVal);
                };

[
                uuid(1B31AB30-D7C1-41DB-B654-C9FA1A7D267F),
                version(1.0),
                helpstring("SMTime 1.0 Type Library")
]
library SMTIMELib
{
                importlib("stdole32.tlb");
                importlib("stdole2.tlb");

                // Now define the module that will "declare" your C functions.
                [
                helpstring("Functions exported by SMTime.dll"),
                version(1.0),
                dllname("SMTime.dll")
                ]
                module StepMasterTimeFunctions
                {

                                [
			// Add a description for your function that the developer can
			// read in the VB Object Browser.
			helpstring("Returns the time in 64 bits."),
			// Specify the actual DLL entry point for the function. Notice
			// the entry field is like the Alias keyword in a VB Declare
			// statement -- it allows you to specify a more friendly name
			// for your exported functions.
			entry("SMTime_Get64BitTime")
		]
		// The [in], [out], and [in, out] keywords tell the Automation
		// client which direction parameters need to be passed. Some
		// calls can be optimized if a function only needs a parameter
		// to be passed one-way.
		CURRENCY __stdcall Get64BitTime([in] LPSYSTEMTIME lpInitTime);
		[
			helpstring("Converts the Julian time into it's components."),
			entry("SMTime_JulianToTime")
		]
		void __stdcall JulianToTime([in] CURRENCY julianTS, [in, out] int *yr, [in, out] int* mm, [in, out] int* dd, [in, out] int *hh, [in, out] int *mi, [in, out] int *ss, [in, out] int *ms);

	} // End of Module

	[
		uuid(27BAB71B-89E1-4A78-8854-FDFFBDC8037E),
		helpstring("SMTimer Class")
	]
	coclass SMTimer
	{
		[default] interface ISMTimer;
	};
};
```

SMTimer.cpp

```
// SMTimer.cpp : Implementation of CSMTimer
//
//	Microsoft TPC-H Kit Ver. 1.00
//	Copyright Microsoft, 1999
//	All Rights Reserved
//
//  Contact:   Reshma Tharamal (reshmat@microsoft.com)
//
#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;
}

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 };

        // 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;
}
```

SMTimer.h

```
// SMTimer.h : Declaration of the CSMTimer
//
//        Microsoft TPC-H Kit Ver. 1.00
//        Copyright Microsoft, 1999
//        All Rights Reserved
//
//  Contact:   Reshma Tharamal (reshmat@microsoft.com)
//

#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_LINK CURRENCY                     __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);
};

#endif //__SMTIMER_H_
```

The listings in this section implement the executedll.dll module.

EXECUTE.CPP

```
// Execute.cpp : Implementation of CExecute
//
//		Microsoft TPC-H Kit Ver. 1.00
//		Copyright Microsoft, 1999
//		All Rights Reserved
//
//  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 * CExecute::m_szOdbcOps[] = {
		"SQLAllocHandle",
		"SQLDriverConnect",
		"SQLExecDirect",
		"SQLSetStmtAttr",
		"SQLCancel",
		"SQLNumResultCols",
		"SQLDescribeCol",
		"SQLColAttribute",
		"SQLFetch",
		"SQLGetData",
		"SQLRowCount",
		"SQLMoreResults"
	};

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;
  }

          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(MyExecute);
	else
		MyExecute->m_tElapsedTime = MyExecute->ExecuteODBC(MyExecute);

	// 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;
	char		szBuf[MAXLOGCMDBUF];
	char		szCmd[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(CExecute *p)
{
	STARTUPINFOA					Start;
	PROCESS_INFORMATION				proc;
	DWORD								exitCode;
	TC_TIME								tElapsed = 0;
	_bstr_t								szCommand("cmd /c ");
	LPSTR								szStartDir;
	CURRENCY							Elapsed;

	szCommand += p->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;

	p->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(p->m_pErrorFile);

		p->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);
		p->m_ExecTime->Stop(&Elapsed);

		if (!GetExitCodeProcess(m_hHandle, &exitCode))
		{
			m_StepStatus = gintFailed;
			LogSystemError(p->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(CExecute *p)
{
		TC_TIME					tElapsed = 0;
		HDBC					m_hdbc;
		SQLRETURN				rc;
		LPSTR					szCmd;
		CURRENCY				Elapsed;
		BOOL					bDoConnect = FALSE;

		// ODBC specific initialization
		m_hdbc = SQL_NULL_HDBC;

		// Allocate a new connection if we are creating a dynamic connection or if
		// the named connection doesn't exist
		if (!InitializeConnection(&m_hdbc, &bDoConnect))
			return(tElapsed);

		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)p->m_szExecDtls, SQL_NTS,
				NULL, 0, NULL, SQL_DRIVER_NOPROMPT);
			if (rc != SQL_SUCCESS)
			{
				if (!HandleODBCError(rc, SQL_HANDLE_DBC, m_hdbc, &m_hdbc, SMSQLDriverConnect))
					return(tElapsed);
			}
		}

#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)
		{
			if (!HandleODBCError(rc, SQL_HANDLE_DBC, m_hdbc, &m_hdbc, SMSQLAllocHandle))
```

```
				return(tElapsed);
		}

		// 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);
			if (!SetConnectionOption(szConnOptions, &m_hdbc))
				return(tElapsed);
		}

		if (m_bQuotedIds)
		{
			if (!SetConnectionOption("SET QUOTED_IDENTIFIER ON ", &m_hdbc))
				return(tElapsed);
		}

		if (!m_bAnsiNulls)
		{
			if (!SetConnectionOption("SET ANSI_NULL_DFLT_OFF ON ", &m_hdbc))
				return(tElapsed);
		}
		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);

			if (!HandleODBCError(rc, SQL_HANDLE_STMT, m_hHandle, &m_hdbc, SMSQLSetStmtAttr))
				return(tElapsed);
		}

		if (m_bNoExecute)
		{
			if (!SetConnectionOption("SET NOEXEC ON ", &m_hdbc))
				return(tElapsed);
		}
		else if (m_bParseOnly)
		{
			if (!SetConnectionOption("SET PARSEONLY ON ", &m_hdbc))
				return(tElapsed);
		}
		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
			if (!SetConnectionOption("SET SHOWPLAN_TEXT ON ", &m_hdbc))
				return(tElapsed);
		}
		else
		{
			if (m_bNoCount)
			{
				if (!SetConnectionOption("SET NOCOUNT ON ", &m_hdbc))
					return(tElapsed);
			}

			if (m_bStatsIO)
			{
				if (!SetConnectionOption("SET STATISTICS IO ON ", &m_hdbc))
					return(tElapsed);
			}

			// 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)
		{
			if (!SetConnectionOption("SET STATISTICS TIME ON ", &m_hdbc))
				return(tElapsed);
		}
	}

	m_szCmd = (LPSTR)p->m_szCommand;
	p->m_ExecTime->Start();

	while ((szCmd = NextCmdInBatch((LPSTR)p->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);
		if (!HandleODBCError(rc, SQL_HANDLE_STMT, m_hHandle, &m_hdbc, SMSQLExecDirect))
			return(tElapsed);

		free(szCmd);

		// Call a procedure to log the results to the output file
		ProcessResultsets();
	}
	p->m_ExecTime->Stop(&Elapsed);

	ResetConnectionProperties(&m_hdbc);

	ODBCCleanup(&m_hdbc, &m_hHandle);

	if (m_StepStatus != gintFailed)
		m_StepStatus = gintComplete;

	return((DWORD)Elapsed.int64);
}

BOOL 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

		if (!m_bAbort && (rc = SQLAllocHandle(SQL_HANDLE_DBC, henv, phdbc)) != SQL_SUCCESS)
		{
			if (!HandleODBCError(rc, SQL_HANDLE_ENV, henv, phdbc, SMSQLAllocHandle))
				return FALSE;
		}
		return TRUE;
	}

	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);

                m_StepStatus = gintFailed;
                _bstr_t temp(SM_ERR_CONN_IN_USE);
                if (m_pErrorFile)
                    m_pErrorFile->WriteLine((BSTR)temp);

                return FALSE;
            }
        }
    }

    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

        if (!m_bAbort && (rc = SQLAllocHandle(SQL_HANDLE_DBC, henv, phdbc)) != SQL_SUCCESS)
        {
            if (!HandleODBCError(rc, SQL_HANDLE_ENV, henv, phdbc, SMSQLAllocHandle))
                return FALSE;
        }

        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 TRUE;
}

void CExecute::ResetConnectionUsage()
{
    if(m_iConnectionIndex >= 0 && m_iConnectionIndex < iConnectionCount)
    {
        EnterCriticalSection(&hConnections);
        (p_Connections + m_iConnectionIndex)->bInUse = FALSE;
        LeaveCriticalSection(&hConnections);
    }

    return;
}

BOOL CExecute::ResetConnectionProperties(HDBC *p_hdbc)
{
    SQLRETURN           rc;

    // Reset connection attributes if any have been modified from the default values

    if (m_bNoExecute)
    {
        if (!SetConnectionOption("SET NOEXEC OFF ", p_hdbc))
            return FALSE;
    }
    else if (m_bParseOnly)
    {
        if (!SetConnectionOption("SET PARSEONLY OFF ", p_hdbc))
            return FALSE;
    }
    else if (m_bShowQP)
    {
        // Reset connection attributes in reverse order
```

```
			if (!SetConnectionOption("SET SHOWPLAN_TEXT OFF ", p_hdbc))
				return FALSE;
		}
		else
		{
			// Reset connection attributes in reverse order
			if (m_bStatsTime)
			{
				if (!SetConnectionOption("SET STATISTICS TIME OFF ", p_hdbc))
					return FALSE;
			}

			if (m_bNoCount)
			{
				if (!SetConnectionOption("SET NOCOUNT OFF ", p_hdbc))
					return FALSE;
			}

			if (m_bStatsIO)
			{
				if (!SetConnectionOption("SET STATISTICS IO OFF ", p_hdbc))
					return FALSE;
			}
		}

		if (m_lRowCount > 0)
		{
			char				szConnOptions[512];

			sprintf(szConnOptions, "SET ROWCOUNT 0 ");
			if (!SetConnectionOption(szConnOptions, p_hdbc))
				return FALSE;
		}

		if (m_bQuotedIds)
		{
			if (!SetConnectionOption("SET QUOTED_IDENTIFIER OFF ", p_hdbc))
				return FALSE;
		}

		if (!m_bAnsiNulls)
		{
			if (!SetConnectionOption("SET ANSI_NULL_DFLT_OFF OFF ", p_hdbc))
				return FALSE;
		}

		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);

			if (!HandleODBCError(rc, SQL_HANDLE_STMT, m_hHandle, p_hdbc, SMSQLSetStmtAttr))
				return FALSE;
		}

		return TRUE;
}

LPSTR CExecute::NextCmdInBatch(LPSTR szBatch)
{
		LPSTR	szCmd, szSeparator, szStart;
		char	szNext;

		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);
}

BOOL 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 FALSE;

        rc = SQLExecDirect(m_hHandle, (unsigned char *)szConnOptions, SQL_NTS);
        if (rc != SQL_SUCCESS)
                LogODBCErrors(rc, SQL_HANDLE_STMT, m_hHandle, SMSQLExecDirect);

        if (!SQL_SUCCEEDED(rc))
        {
                ODBCCleanup(pHdbc, &m_hHandle);
                return FALSE;
        }

        return TRUE;
}

BOOL CExecute::HandleODBCError(SQLRETURN rc, SWORD fHandleType, SQLHANDLE handle, HDBC *pHdbc, OdbcOperations OdbcOp)
{
        if (rc != SQL_SUCCESS)
  {
                LogODBCErrors(rc, fHandleType, handle, OdbcOp);
                if (!SQL_SUCCEEDED(rc))
                {
                        ODBCCleanup(pHdbc, &m_hHandle);
                        return FALSE;
                }
  }
        return TRUE;
}

STDMETHODIMP CExecute::AbortODBC()
{
        m_bAbort = TRUE;

        if (m_hHandle != SQL_NULL_HSTMT)
        {
#ifdef _DEBUG
                _CrtDbgReport(_CRT_WARN, NULL, 0, NULL, "Executing SQLCancel.\n");
#endif

                SQLRETURN rc = SQLCancel(m_hHandle);
                if (rc != SQL_SUCCESS)
                        LogODBCErrors(rc, SQL_HANDLE_STMT, m_hHandle, SMSQLCancel);
        }

        return(S_OK);
}

void CExecute::ProcessResultsets()
{
  SQLSMALLINT *CTypeArray, *CScaleArray;
  SQLINTEGER  *ColLenArray, *DispLenArray;
        SQLSMALLINT iColNameLen, SQLType, iColNull, i, NumCols = 0;
        SQLINTEGER              iDispLen, iRowCount, LenOrInd;
  SQLRETURN     rc;
        char                    szColName[MAX_DATA_LEN + 1];
  void          *DataPtr;


        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);
                if (rc != SQL_SUCCESS)
                {
                        LogODBCErrors(rc, SQL_HANDLE_STMT, m_hHandle, SMSQLNumResultCols);
                        if (!SQL_SUCCEEDED(rc))
                                break;
                }

                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));

                        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
                                rc = SQLDescribeCol(m_hHandle, ((SQLUSMALLINT) i)+1,
                                        (unsigned char *)szColName, sizeof(szColName), &iColNameLen,
                    &SQLType, (unsigned long *)&ColLenArray[i], &CScaleArray[i], &iColNull);
                if (rc != SQL_SUCCESS)
                {
                    LogODBCErrors(rc, SQL_HANDLE_STMT, m_hHandle, SMSQLDescribeCol);
                    if (!SQL_SUCCEEDED(rc))
                        return;
                }

#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);
                if (rc != SQL_SUCCESS)
                {
                    LogODBCErrors(rc, SQL_HANDLE_STMT, m_hHandle, SMSQLColAttribute);
                    if (!SQL_SUCCEEDED(rc))
                        return;
                }

                // 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]);
            }
            m_pOutputFile->WriteLine(NULL);

            // Underline each column name
            for (i = 0; i < NumCols; i++)
            {
                memset(szColName, '-', DispLenArray[i]);
                *(szColName + DispLenArray[i]) = '\0';
                PrintData(szColName, SQL_C_CHAR, DispLenArray[i], 0, m_pOutputFile);
            }
            m_pOutputFile->WriteLine(NULL);

            // Retrieve and print each row. PrintData accepts a pointer to the data, its C type,
            // and its byte length/indicator.
#ifdef _DEBUG
            _CrtDbgReport(_CRT_WARN, NULL, 0, NULL, "Executing SQLFetch.\n");
#endif


            while (!m_bAbort && (rc = SQLFetch(m_hHandle)) != SQL_NO_DATA)
            {
                if (!SQL_SUCCEEDED(rc))
                {
                    LogODBCErrors(rc, SQL_HANDLE_STMT, m_hHandle, SMSQLFetch);
                    break;
                }

                for (i = 0; i < NumCols; i++)
                {
                    // Allocate the data buffer.
                    DataPtr = malloc(ColLenArray[i]);
```

```
#ifdef _DEBUG
                    _CrtDbgReport(_CRT_WARN, NULL, 0, NULL, "Executing SQLGetData.\n");
#endif
                    while (!m_bAbort && (rc=SQLGetData(m_hHandle, i + 1, CTypeArray[i],
                        DataPtr, ColLenArray[i], &LenOrInd)) != SQL_NO_DATA)
                    {
                        if (!SQL_SUCCEEDED(rc))
                        {
                            LogODBCErrors(rc, SQL_HANDLE_STMT, m_hHandle, SMSQLGetData);
                            if (!SQL_SUCCEEDED(rc))
                                return;
                        }

                        if (LenOrInd == SQL_NULL_DATA)
                            PrintData(S_NULL, SQL_C_CHAR, DispLenArray[i], 0, m_pOutputFile);
                        else
                        {
                            PrintData((SQLCHAR *)DataPtr, CTypeArray[i], DispLenArray[i],
                                CScaleArray[i], m_pOutputFile);
                            // Currently  printing a maximum of MAX_DATA_LEN chars.
                            break;
                        }
                    }

                    free(DataPtr);
                }
                m_pOutputFile->WriteLine(NULL);
            }
            m_pOutputFile->WriteLine(NULL);

            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);
        if (rc != SQL_SUCCESS)
        {
            LogODBCErrors(rc, SQL_HANDLE_STMT, m_hHandle, SMSQLRowCount);
            if (!SQL_SUCCEEDED(rc))
                break;
        }

        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_SUCCESS)
                {
                        LogODBCErrors(rc, SQL_HANDLE_STMT, m_hHandle, SMSQLMoreResults);

                        if (!SQL_SUCCEEDED(rc))
                                break;
                }

   } 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:
                                sprintf(fmt, "%%.0%df", iScale);
                                j = sprintf(s, fmt, *(float *)vData);
                                break;

                        case SQL_C_DOUBLE:
                        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);

   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",
                                        m_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 != 5701 && dwErrCode != 5703)
                            {
                                        // 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: 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;
}
```

EXECUTE.H

```
// Execute.h : Declaration of the CExecute
//
//	Microsoft TPC-H Kit Ver. 1.00
//	Copyright Microsoft, 1999
//	All Rights Reserved
//
//  Contact:   Reshma Tharamal (reshmat@microsoft.com)
//

#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_ERR_CONN_IN_USE              "StepMaster Error: Connection is already in use."

#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 ALIGNSIZE 4
#define S_NULL          "NULL"
#define ALIGNBUF(Length) Length % ALIGNSIZE ? \
   Length + ALIGNSIZE - (Length % ALIGNSIZE) : Length

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()
        {
        }

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;
}

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(IConnectionPointContainer)
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(CExecute *p);
        TC_TIME                 ExecuteODBC(CExecute *p);
        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:
        typedef enum OdbcOperations
        {
                SMSQLAllocHandle,
                SMSQLDriverConnect,
                SMSQLExecDirect,
                SMSQLSetStmtAttr,
                SMSQLCancel,
                SMSQLNumResultCols,
                SMSQLDescribeCol,
                SMSQLColAttribute,
                SMSQLFetch,
					SMSQLGetData,
					SMSQLRowCount,
					SMSQLMoreResults
		};

		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);
		BOOL				SetConnectionOption(LPSTR szConn, HDBC *pHdbc);
		BOOL				ResetConnectionProperties(HDBC *p_hdbc);
		BOOL				HandleODBCError(SQLRETURN rc, SWORD fHandleType, SQLHANDLE handle, HDBC *pHdbc, OdbcOperations OdbcOp);

		BOOL				InitializeConnection(HDBC *phdbc, BOOL *pbDoConnect);
		void				ResetConnectionUsage();

		int					m_iConnectionIndex;

		static char			*m_szOdbcOps[];

		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;

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( (iRetCode != SQL_SUCCESS) && (iRetCode != SQL_SUCCESS_WITH_INFO) && (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, ""));}
};

void				ExecutionThread(LPVOID lpParameter);

#ifdef _TPCH_AUDIT
		void				WriteFileToTpchLog(LPSTR szFile, LPSTR szFmt);
		void				WriteToTpchLog(char *szMsg);
#endif

#endif //__EXECUTE_H_
```

ExecuteDll.cpp

```
// ExecuteDll.cpp : Implementation of DLL Exports.
//
//		Microsoft TPC-H Kit Ver. 1.00
//		Copyright Microsoft, 1999
//		All Rights Reserved
//
//  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();

/////////////////////////////////////////////////////////////////////////////
// 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)))
                        return FALSE;
                /*
                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);
                        */

                if (!SQL_SUCCEEDED(SQLSetEnvAttr(henv, SQL_ATTR_ODBC_VERSION, (LPVOID)SQL_OV_ODBC3, 0)))
                        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);
        }
}

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);
}
```

EXECUTEDLL.DEF

```
; ExecuteDll.def : Declares the module parameters.

LIBRARY      "ExecuteDll.DLL"
```

```
EXPORTS
        DllCanUnloadNow     @1 PRIVATE
        DllGetClassObject   @2 PRIVATE
        DllRegisterServer   @3 PRIVATE
        DllUnregisterServer        @4 PRIVATE
```

EXECUTEDLL.IDL

```
// ExecuteDll.idl : IDL source for ExecuteDll.dll
//
//      Microsoft TPC-H Kit Ver. 1.00
//      Copyright Microsoft, 1999
//      All Rights Reserved
//
//  Contact:   Reshma Tharamal (reshmat@microsoft.com)
//

// This file will be processed by the MIDL tool to
// produce the type library (ExecuteDll.tlb) and marshalling code.

import "oaidl.idl";
import "ocidl.idl";
        typedef
        [
                uuid(0AC32070-B0DB-11d2-BC0D-00A0C90D2CA5),
                helpstring("Execution Types"),
        ]

        enum ExecutionType
        {
                [helpstring("Shell")]       execODBC  = 0x0001,
                [helpstring("ODBC")]        execShell    = 0x0002
        } ExecutionType;

        typedef
        [
                uuid(D4A4B9B0-BAE3-11d2-BC0F-00A0C90D2CA5),
                helpstring("Run Status Values"),
        ]

        enum InstanceStatus
        {
                [helpstring("Disabled")]   gintDisabled            = 0x0001,
                [helpstring("Pending")]            gintPending             = 0x0002,
                [helpstring("Running")]            gintRunning             = 0x0003,
                [helpstring("Complete")]  gintComplete            = 0x0004,
                [helpstring("Failed")]             gintFailed              = 0x0005,
                [helpstring("Aborted")]            gintAborted             = 0x0006
        } InstanceStatus;


[
        uuid(551AC525-AB1C-11D2-BC0C-00A0C90D2CA5),
        version(1.0),
        helpstring("ExecuteDll 1.0 Type Library")
]
library EXECUTEDLLLib
{
        importlib("stdole32.tlb");
        importlib("stdole2.tlb");

        [
                uuid(551AC532-AB1C-11D2-BC0C-00A0C90D2CA5),
                helpstring("_IExecuteEvents Interface")
        ]
        dispinterface _IExecuteEvents
        {
                properties:
                methods:
                [id(1), helpstring("method Start")] void Start([in] CURRENCY StartTime);
                [id(2), helpstring("method Complete")] void Complete([in] CURRENCY EndTime, [in] long Elapsed);
        };
```

```
	[
		object,
		uuid(551AC531-AB1C-11D2-BC0C-00A0C90D2CA5),
		dual,
		helpstring("IExecute Interface"),
		pointer_default(unique)
	]
	interface IExecute : IDispatch
	{
		[id(1), helpstring("method DoExecute")] HRESULT 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);
		[propget, id(2), helpstring("property StepStatus")] HRESULT StepStatus([out, retval] InstanceStatus *pVal);
		[id(3), helpstring("method Abort")] HRESULT Abort();
		[id(4), helpstring("method WriteError")] HRESULT WriteError(BSTR szMsg);
		[propput, id(5), helpstring("property OutputFile")] HRESULT OutputFile([in] BSTR newVal);
		[propput, id(6), helpstring("property ErrorFile")] HRESULT ErrorFile([in] BSTR newVal);
	};
	[
		uuid(2EFC198E-AA8D-11D2-BC0C-00A0C90D2CA5),
		helpstring("Execute Class")
	]
	coclass Execute
	{
		[default] interface IExecute;
		[default, source] dispinterface _IExecuteEvents;
	};
};
```

ExecuteDllCP.h

```
//
//	Microsoft TPC-H Kit Ver. 1.00
//	Copyright Microsoft, 1999
//	All Rights Reserved
//
//  Contact:   Reshma Tharamal (reshmat@microsoft.com)
//
#ifndef _EXECUTEDLLCP_H_
#define _EXECUTEDLLCP_H_

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
```

SMExecute.h

```
//
//		Microsoft TPC-H Kit Ver. 1.00
//		Copyright Microsoft, 1999
//		All Rights Reserved
//
//  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
{
	char		szConnectionName[CONNECTION_NAME_LEN];
	HDBC		hdbc;
	BOOL		bInUse;
} SM_Connection_Info;
```

The listings in this section implement the Log Writer module.

LogWriter.cpp

```
// LogWriter.cpp : Implementation of DLL Exports.
//
//		Microsoft TPC-H Kit Ver. 1.00
//		Copyright Microsoft, 1999
//		All Rights Reserved
//
//  Contact:   Reshma Tharamal (reshmat@microsoft.com)
//

// 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);
}
```

LogWriter.def

```
; LogWriter.def : Declares the module parameters.

LIBRARY      "LogWriter.DLL"

EXPORTS
                DllCanUnloadNow     @1 PRIVATE
                DllGetClassObject   @2 PRIVATE
                DllRegisterServer   @3 PRIVATE
                DllUnregisterServer        @4 PRIVATE
```

LogWriter.idl

```
// LogWriter.idl : IDL source for LogWriter.dll
//
//            Microsoft TPC-H Kit Ver. 1.00
//            Copyright Microsoft, 1999
//            All Rights Reserved
//
//  Contact:   Reshma Tharamal (reshmat@microsoft.com)
//

// This file will be processed by the MIDL tool to
// produce the type library (LogWriter.tlb) and marshalling code.

import "oaidl.idl";
import "ocidl.idl";
                [
                                object,
                                uuid(5AC75DAD-1936-11D3-BC2D-00A0C90D2CA5),
                                dual,
                                helpstring("ISMLog Interface"),
                                pointer_default(unique)
                ]
                interface ISMLog : IDispatch
                {
                                [propput, id(1), helpstring("property FileHeader")] HRESULT FileHeader([in] BSTR newVal);
                                [id(2), helpstring("method WriteLine")] HRESULT WriteLine(BSTR szMsg);
                                [id(3), helpstring("method WriteField")] HRESULT WriteField(/*[in]*/ BSTR szMsg);
                                [propput, id(4), helpstring("property FileName")] HRESULT FileName([in] BSTR newVal);
                                [propput, id(5), helpstring("property Append")] HRESULT Append([in] BOOL newVal);
                };

[
                uuid(5AC75DA1-1936-11D3-BC2D-00A0C90D2CA5),
                version(1.0),
                helpstring("LogWriter 1.0 Type Library")
]
library LOGWRITERLib
{
                importlib("stdole32.tlb");
                importlib("stdole2.tlb");

                [
                                uuid(5AC75DB1-1936-11D3-BC2D-00A0C90D2CA5),
                                helpstring("_ISMLogEvents Interface")
                ]
                dispinterface _ISMLogEvents
                {
                                properties:
                                methods:
                };

                [
                                uuid(5AC75DB0-1936-11D3-BC2D-00A0C90D2CA5),
        helpstring("SMLog Class")
    ]
    coclass SMLog
    {
        [default] interface ISMLog;
        [default, source] dispinterface _ISMLogEvents;
    };
};
```

LogWriteCP.h

```
//
//      Microsoft TPC-H Kit Ver. 1.00
//      Copyright Microsoft, 1999
//      All Rights Reserved
//
//  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
```

SMLog.cpp

```
// SMLog.cpp : Implementation of CSMLog
//
//      Microsoft TPC-H Kit Ver. 1.00
//      Copyright Microsoft, 1999
//      All Rights Reserved
//
//  Contact:   Reshma Tharamal (reshmat@microsoft.com)
//
#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()
{
    char    szDrive[256];
    char    szDir[256];
    char    szLogDir[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();
        }

        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;
}

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 }

STDMETHODIMP CSMLog::put_Append(BOOL newVal)
{
		m_bAppend = newVal;
		return S_OK;
}
```

WAIT4SQL.CPP

```
//
//      Microsoft TPC-H Kit Ver. 1.00
//      Copyright Microsoft, 1999
//      All Rights Reserved
//
////////////////////////////////////////////////////////////////////////////
// wait4sql
//
// created 5/15/96 by Jack Richins
//
// waits for sqlserverRecComplete (recovery complete event) for command line
// specified time(in milliseconds). Returns 0 if signaled (meaning server is
// already up and running), 1 if time out or other error.
//
////////////////////////////////////////////////////////////////////////////
#include <windows.h>
#include <stdlib.h>
#include <iostream.h>
#include <stdio.h>

int main (int argc, char *argv[])
{
        char                    eventString[MAX_PATH];
        char                    *instanceName = "";
        const char   *szName = "sqlserverRecComplete";

        // Check valid time argument
        if(argv[1] == NULL || *(argv[1]) == '-' || *(argv[1]) == '/')
        {
                cout << "Correct usage:  wait4sql <time-in-ms> [-s<instanceName>]" << endl;

                return EXIT_FAILURE;
        }

        // Set Time
        DWORD time = atol (argv[1]);

        // Setting the time argument equal to zero causes an infinite wait
        //
        if(time == 0)
                time = INFINITE;

        // Check whether the optional instance name argument was specified
        if(argv[2])
        {
                if(*(argv[2]) == '-' || *(argv[2]) == '/')
                {
                        if(*(argv[2]+1) == 's')
                        {
                                instanceName = _strupr(argv[2]+2);
                        }
                }
        }

        // Create the event name. For a named instance, the instancename is appended
        // to the end.
        //
        if(strcmp(instanceName, "") && stricmp(instanceName, "MSSQLServer"))
        {
                sprintf(eventString, "%s$%s", szName, instanceName);
        }
        else
        {
                strcpy(eventString, szName);
        }

        // Try and open the SQL server event
        //
        HANDLE hRecovered = CreateEvent (NULL, TRUE, FALSE, eventString);
```

```
	// Do we have a valid handle?
	//
	if (NULL == hRecovered)
		{
		cout << "wait4sql: Error - cannot open event '" << eventString << "'" << endl;
		return EXIT_FAILURE;
		}

	// Wait for recovery or timeout. If result equals object signaled, success, else failure.
	//
	BOOL fSrvUp = (WaitForSingleObject (hRecovered, time) == WAIT_OBJECT_0);

	// Close the event handle
	CloseHandle (hRecovered);

	if (fSrvUp)
		return EXIT_SUCCESS;
	else
		return EXIT_FAILURE;
}
```

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 );
        }
        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)
                        {
                                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"
                );

        exit(EXIT_FAILURE);
}
```

The text that follows is the contents of the Access database used to drive Stepmaster:

Att_workspaces

workspace_id	workspace_name	archived_flag
2	Setup for TPC-H 300gb	0

Workspace_parameters

workspace_id =	2	***parameter_id***	16	***parameter_type =***	0
parameter_name		DBGEN_PARALLELISM			
parameter_valu		16			
workspace_id =	2	***parameter_id***	3	***parameter_type =***	0
parameter_name		DBNAME			
parameter_valu		tpch300g			
workspace_id =	2	***parameter_id***	4	***parameter_type =***	0
parameter_name		FLATFILE_DIR			
parameter_valu		f:\Flat_Files			
workspace_id =	2	***parameter_id***	5	***parameter_type =***	0
parameter_name		SETUP_DIR			
parameter_valu		%KIT_DIR%\Setup			
workspace_id =	2	***parameter_id***	6	***parameter_type =***	3
parameter_name		OUTPUT_DIR			
parameter_valu		c:\OUTPUT\200210~1\781			
workspace_id =	2	***parameter_id***	7	***parameter_type =***	0
parameter_name		SCALEFACTOR			
parameter_valu		300			
workspace_id =	2	***parameter_id***	8	***parameter_type =***	0
parameter_name		UPDATE_SETS			
parameter_valu		24			
workspace_id =	2	***parameter_id***	10	***parameter_type =***	0
parameter_name		DELETE_SEGMENTS_PER_UPDATE_SET			
parameter_valu		32			

workspace_id =	2	***parameter_id***	11	***parameter_type =***	0
parameter_name		INSERT_SEGMENTS_PER_UPDATE_SET			
parameter_valu		32			
workspace_id =	2	***parameter_id***	13	***parameter_type =***	0
parameter_name		VALIDATION_DIR			
parameter_valu		%SETUP_DIR%\Validation			
workspace_id =	2	***parameter_id***	2	***parameter_type =***	0
parameter_name		TRACEFLAGS			
parameter_valu					
workspace_id =	2	***parameter_id***	15	***parameter_type =***	0
parameter_name		TABLE_LOAD_PARALLELISM			
parameter_valu		16			
workspace_id =	2	***parameter_id***	150	***parameter_type =***	3
parameter_name		RUN_ID			
parameter_valu		781			
workspace_id =	2	***parameter_id***	36	***parameter_type =***	0
parameter_name		RF_FLATFILE_DIR			
parameter_valu		f:\RF_Flat_Files-32			
workspace_id =	2	***parameter_id***	40	***parameter_type =***	3
parameter_name		DEFAULT_DIR			
parameter_valu		c:\OUTPUT\20021001_Audit_Setup			
workspace_id =	2	***parameter_id***	55	***parameter_type =***	0
parameter_name		TOOLS_DIR			
parameter_valu		%KIT_DIR%\tools			
workspace_id =	2	***parameter_id***	59	***parameter_type =***	0
parameter_name		MAX_STREAMS			
parameter_valu		6			
workspace_id =	2	***parameter_id***	61	***parameter_type =***	0
parameter_name		TEMPLATE_DIR			
parameter_valu		%RUN_DIR%\templates			

workspace_id =	2	*parameter_id*	62	*parameter_type =*	0
parameter_name	RUN_DIR				
parameter_valu	%KIT_DIR%\run				
workspace_id =	2	*parameter_id*	63	*parameter_type =*	0
parameter_name	QUERY_DIR				
parameter_valu	%RUN_DIR%\Queries-6				
workspace_id =	2	*parameter_id*	66	*parameter_type =*	0
parameter_name	KIT_DIR				
parameter_valu	C:\mstpch_Feb2002				
workspace_id =	2	*parameter_id*	90	*parameter_type =*	0
parameter_name	DBGEN_THREAD_PARALLELISM				
parameter_valu	16				
workspace_id =	2	*parameter_id*	149	*parameter_type =*	0
parameter_name	TEST_QGEN_SEED				
parameter_valu	1213214432				
workspace_id =	2	*parameter_id*	14	*parameter_type =*	0
parameter_name	INDEX_CREATE_PARALLELISM				
parameter_valu	32				

Connection_dtls

workspace_id	connection_name_id	connection_name	connection_string_name	connection_type
2	26	Stream5	DBCONNECTION	1
2	25	Stream4	DBCONNECTION	1
2	24	Stream3	DBCONNECTION	1
2	23	Stream2	DBCONNECTION	1
2	22	Stream1	DBCONNECTION	1
2	21	Stream0	DBCONNECTION	1
2	7	VALIDATION_STREAM_CONN	DBCONNECTION	1
2	2	DYNAMIC_MASTER_DB_CONNECTION	MASTERDBCONNECTION	2
2	1	DYNAMIC_DB_CONNECTION	DBCONNECTION	2

Workspace_connections

workspace_i	2
connection_i	2
connection_name	MASTERDBCONNECTION
connection_valu	DRIVER=SQL Server;SERVER=;UID=sa;PWD=;
descriptio	
no_count_displa	0
no_execute	0
parse_query_onl	0
ANSI_quoted_identifier	0
ANSI_nulls	-1
show_query_pla	0
show_stats_tim	0
show_stats_i	0
parse_odbc_msg_prefix	-1
row_count	0
tsql_batch_separat	GO
query_time_out	0
server_languag	(Default)
character_translatio	-1
regional_setting	0

workspace_i	2
connection_i	1
connection_name	DBCONNECTION
connection_valu	DRIVER=SQL Server;SERVER=;UID=sa;PWD=;DATABASE=%DBNAME%;
descriptio	
no_count_displa	0
no_execute	0
parse_query_onl	0
ANSI_quoted_identifier	0
ANSI_nulls	-1
show_query_pla	0
show_stats_tim	0
show_stats_i	0
parse_odbc_msg_prefix	-1
row_count	0
tsql_batch_separat	GO
query_time_out	0
server_languag	(Default)

dcharacter_translatio -1
dregional_setting 0

enAtt_steps

yworkspace_id = 2

step_label = *y* SqlServer Startup*step_id =* 2 *global_flag =* -1
s
sequence_no = 1 *step_level =* 0 *parent_step_id =* 0 *enabled_flag =* 0
iterator_name = *n* *degree_parallelism* 0
execution_mechanism = *e* 2 *continuation_criteria =* 0 *failure_details =*
step_file_name = *o* *version_no =* 13.0
start_directory = *es* *parent_version_no =*
0.0

step_text = start "SQLSERVR" sqlservr -E -c -x -g100 %TRACEFLAGS%
or

e
step_label = SqlServer Shutdown *step_id =* 3 *global_flag =* -1
sequence_no = 2 *step_level =* 0 *parent_step_id =* 0 *enabled_flag =* 0
iterator_name = *degree_parallelism* 0
execution_mechanism = 2 *continuation_criteria =* 0 *failure_details =*
step_file_name = *version_no =* 6.0
start_directory = *parent_version_no =* 0.0
step_text = isql -Usa -P -t60 -Q"shutdown"

step_label = Syntax Check Parameters *step_id =* 4 *global_flag =* 0
sequence_no = 1 *step_level =* 0 *parent_step_id =* 0 *enabled_flag =* 0
iterator_name = *degree_parallelism* 1
execution_mechanism = 0 *continuation_criteria =* 0 *failure_details =*
step_file_name = *version_no =* 41.0
start_directory = *parent_version_no =* 0.0
step_text =

step_label = Generate FlatFiles ***step_id =*** 5 ***global_flag =*** 0
sequence_no = 2 ***step_level =*** 0 ***parent_step_id =*** 0 ***enabled_flag =*** 0
iterator_name = ***degree_parallelism*** 1
execution_mechanism = 0 ***continuation_criteria =*** 0 ***failure_details =***
step_file_name = ***version_no =*** 60.0
start_directory = ***parent_version_no =*** 0.0
step_text =

step_label = Start SqlServer ***step_id =*** 6 ***global_flag =*** 0
sequence_no = 3 ***step_level =*** 0 ***parent_step_id =*** 0 ***enabled_flag =*** -1
iterator_name = ***degree_parallelism*** 1
execution_mechanism = 0 ***continuation_criteria =*** 0 ***failure_details =***
step_file_name = ***version_no =*** 44.0
start_directory = ***parent_version_no =*** 0.0
step_text =

step_label = Configure SqlServer ***step_id =*** 7 ***global_flag =*** 0
sequence_no = 5 ***step_level =*** 0 ***parent_step_id =*** 0 ***enabled_flag =*** -1
iterator_name = ***degree_parallelism*** 1
execution_mechanism = 0 ***continuation_criteria =*** 0 ***failure_details =***
step_file_name = ***version_no =*** 93.1
start_directory = ***parent_version_no =*** 0.0
step_text =

step_label = Create database ***step_id =*** 8 ***global_flag =*** 0
sequence_no = 4 ***step_level =*** 0 ***parent_step_id =*** 0 ***enabled_flag =*** -1
iterator_name = ***degree_parallelism*** 1
execution_mechanism = 0 ***continuation_criteria =*** 0 ***failure_details =***
step_file_name = ***version_no =*** 86.0
start_directory = ***parent_version_no =*** 0.0
step_text =

step_label = Create and Load Tables ***step_id =*** 9 ***global_flag =*** 0
sequence_no = 6 ***step_level =*** 0 ***parent_step_id =*** 0 ***enabled_flag =*** -1
iterator_name = ***degree_parallelism*** 1
execution_mechanism = 0 ***continuation_criteria =*** 0 ***failure_details =***
step_file_name = ***version_no =*** 76.0
start_directory = ***parent_version_no =*** 0.0
step_text =

step_label = Create Indexes ***step_id =*** 10 ***global_flag =*** 0
sequence_no = 7 ***step_level =*** 0 ***parent_step_id =*** 0 ***enabled_flag =*** -1
iterator_name = ***degree_parallelism*** 1
execution_mechanism = 0 ***continuation_criteria =*** 0 ***failure_details =***
step_file_name = ***version_no =*** 73.0
start_directory = ***parent_version_no =*** 0.0
step_text =

step_label = Misc Cleanup ***step_id =*** 11 ***global_flag =*** 0
sequence_no = 8 ***step_level =*** 0 ***parent_step_id =*** 0 ***enabled_flag =*** -1
iterator_name = ***degree_parallelism*** 1
execution_mechanism = 0 ***continuation_criteria =*** 0 ***failure_details =***
step_file_name = ***version_no =*** 71.1
start_directory = ***parent_version_no =*** 0.0
step_text =

step_label = Validation (for SCALEFACTOR=1 Only) ***step_id =*** 12 ***global_flag =*** 0
sequence_no = 12 ***step_level =*** 0 ***parent_step_id =*** 0 ***enabled_flag =*** 0
iterator_name = ***degree_parallelism*** 1
execution_mechanism = 0 ***continuation_criteria =*** 0 ***failure_details =***
step_file_name = ***version_no =*** 8.1
start_directory = ***parent_version_no =*** 0.0
step_text =

step_label = Generate FlatFiles in Parallel ***step_id =*** 14 ***global_flag =*** 0

sequence_no = 1 ***step_level =*** 1 ***parent_step_id =*** 5 ***enabled_flag =*** 0

iterator_name = ***degree_parallelism*** 16

execution_mechanism = 0 ***continuation_criteria =*** 0 ***failure_details =***

step_file_name = ***version_no =*** 11.0

start_directory = ***parent_version_no =*** 60.0

step_text =

step_label = Generate Update Files ***step_id =*** 15 ***global_flag =*** 0

sequence_no = 2 ***step_level =*** 1 ***parent_step_id =*** 5 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** 1

execution_mechanism = 2 ***continuation_criteria =*** 1 ***failure_details =***

step_file_name = ***version_no =*** 16.0

start_directory = %RF_FLATFILE_DIR% ***parent_version_no =*** 60.0

step_text =

```
%TOOLS_DIR%\DBGEN\dbgen -q -b %TOOLS_DIR%\dists.dss -U %UPDATE_SETS% -s
%SCALEFACTOR% -f -C %UPDATE_SETS% -i %INSERT_SEGMENTS_PER_UPDATE_SET% -d
%DELETE_SEGMENTS_PER_UPDATE_SET%
```

step_label = Create TPC-H Database ***step_id =*** 17 ***global_flag =*** 0

sequence_no = 1 ***step_level =*** 1 ***parent_step_id =*** 8 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** 1

execution_mechanism = 1 ***continuation_criteria =*** 1 ***failure_details =***

step_file_name = %SETUP_DIR%\%DBNAME%\CreateDatabase.sql ***version_no =*** 33.0

start_directory = DYNAMIC_MASTER_DB_CONNECTION ***parent_version_no =*** 86.0

step_text =

step_label = Set DB_Option "Select Into" ***step_id =*** 18 ***global_flag =*** 0

sequence_no = 2 ***step_level =*** 1 ***parent_step_id =*** 8 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** 1

execution_mechanism = 1 ***continuation_criteria =*** 1 ***failure_details =***

step_file_name = ***version_no =*** 10.0

start_directory = DYNAMIC_MASTER_DB_CONNECTION ***parent_version_no =*** 86.0

step_text = sp_dboption %DBNAME%, 'select ',true

step_label = Set DB_Option "Trunc" ***step_id =*** 19 ***global_flag =*** 0

sequence_no = 3 ***step_level =*** 1 ***parent_step_id =*** 8 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** 1

execution_mechanism = 1 ***continuation_criteria =*** 1 ***failure_details =***

step_file_name = ***version_no =*** 10.0

start_directory = DYNAMIC_MASTER_DB_CONNECTION ***parent_version_no =*** 86.0

step_text = sp_dboption %DBNAME%, 'trunc. ',true

step_label = (Drop/)Create/Pin Tables ***step_id =*** 20 ***global_flag =*** 0

sequence_no = 1 ***step_level =*** 1 ***parent_step_id =*** 9 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** 1

execution_mechanism = 0 ***continuation_criteria =*** 0 ***failure_details =***

step_file_name = ***version_no =*** 8.1

start_directory = ***parent_version_no =*** 76.0

step_text =

step_label = Drop Existing Indexes ***step_id =*** 23 ***global_flag =*** 0

sequence_no = 1 ***step_level =*** 1 ***parent_step_id =*** 10 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** 1

execution_mechanism = 1 ***continuation_criteria =*** 1 ***failure_details =***

step_file_name = ***version_no =*** 14.0

start_directory = DYNAMIC_DB_CONNECTION ***parent_version_no =*** 73.0

step_text =

```
declare @index      sysname
declare @table      sysname

-- Drop the non-clustered indexes first

declare nc_index cursor for
select sysindexes.name,sysobjects.name
  from sysindexes,sysobjects
  where
    sysobjects.id=sysindexes.id and
    sysobjects.type='U'     and
    sysindexes.indid>1      and
    sysindexes.status<>96
  order by sysindexes.name

open nc_index
fetch nc_index into @index,@table
while @@fetch_status = 0
  begin
    print 'dropping NC index ' + @table + '.' + @index
    exec('drop index '+@table+'.'+@index)
    fetch nc_index into @index,@table
  end

-- Drop the Clustered Indexes last

declare cl_index cursor for
select sysindexes.name,sysobjects.name
  from sysindexes,sysobjects
  where
    sysobjects.id=sysindexes.id and
    sysobjects.type='U'     and
    sysindexes.indid=1      and
    sysindexes.status<>96
  order by sysindexes.name

open cl_index
fetch cl_index into @index,@table
while @@fetch_status = 0
  begin
    print 'dropping Clustered index ' + @table + '.' + @index
    exec('drop index '+@table+'.'+@index)
    fetch cl_index into @index,@table
  end
```

step_label = Verify TPC-H Load ***step_id =*** 25 ***global_flag =*** 0
sequence_no = 9 ***step_level =*** 1 ***parent_step_id =*** 11 ***enabled_flag =*** -1
iterator_name = ***degree_parallelism*** 1
execution_mechanism = 1 ***continuation_criteria =*** 2 ***failure_details =***
step_file_name = %SETUP_DIR%\Utility\VerifyTpchLoad.sql ***version_no =*** 20.0
start_directory = DYNAMIC_DB_CONNECTION ***parent_version_no =*** 71.1
step_text =

step_label = Create Statistics and Disable AutoUpdate ***step_id =*** 26 ***global_flag =*** 0
sequence_no = 1 ***step_level =*** 1 ***parent_step_id =*** 11 ***enabled_flag =*** -1
iterator_name = ***degree_parallelism*** 1
execution_mechanism = 0 ***continuation_criteria =*** 0 ***failure_details =***
step_file_name = ***version_no =*** 16.1
start_directory = ***parent_version_no =*** 71.1
step_text =

step_label = Generate QGEN Seed ***step_id =*** 27 ***global_flag =*** 0

sequence_no = 10 ***step_level =*** 1 ***parent_step_id =*** 11 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** 1

execution_mechanism = 1 ***continuation_criteria =*** 2 ***failure_details =***

step_file_name = ***version_no =*** 8.0

start_directory = DYNAMIC_DB_CONNECTION ***parent_version_no =*** 71.1

step_text =

```
DECLARE @Finish datetime
DECLARE @seed0 integer

--
-- Get ending time of database load
--
SELECT @Finish=LoadFinish FROM TPCH_AUX_TABLE

--
-- Calculate seed per clause 2.1.3.3
--
SET @seed0 =
CONVERT(integer,100000000*DATEPART(MM,@Finish)+1000000*DATEPART(DD,@Finish)+10000*DAT
EPART(HH,@Finish)+100*DATEPART(MI,@Finish)+DATEPART(SS,@Finish))

--
-- Update the benchmark auxillary table
--
UPDATE TPCH_AUX_TABLE SET QgenSeed=@seed0

SELECT * from TPCH_AUX_TABLE
```

step_label = Move TempDB ***step_id =*** 29 ***global_flag =*** 0

sequence_no = 2 ***step_level =*** 1 ***parent_step_id =*** 7 ***enabled_flag =*** 0

iterator_name = ***degree_parallelism*** 1

execution_mechanism = 1 ***continuation_criteria =*** 1 ***failure_details =***

step_file_name = %SETUP_DIR%\%DBNAME%\tempdb\MoveTempDB.sql ***version_no =*** 32.0

start_directory = DYNAMIC_MASTER_DB_CONNECTION ***parent_version_no =*** 93.1

step_text =

step_label = ReSize TempDB ***step_id =*** 30 ***global_flag =*** 0

sequence_no = 3 ***step_level =*** 1 ***parent_step_id =*** 7 ***enabled_flag =*** 0

iterator_name = ***degree_parallelism*** 1

execution_mechanism = 1 ***continuation_criteria =*** 1 ***failure_details =***

step_file_name = %SETUP_DIR%\%DBNAME%\tempdb\ReSizeTempDB.sql ***version_no =*** 29.0

start_directory = DYNAMIC_MASTER_DB_CONNECTION ***parent_version_no =*** 93.1

step_text =

step_label = Set sp_configure Options ***step_id =*** 31 ***global_flag =*** 0

sequence_no = 4 ***step_level =*** 1 ***parent_step_id =*** 7 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** 1

execution_mechanism = 1 ***continuation_criteria =*** 2 ***failure_details =***

step_file_name = %SETUP_DIR%\Utility\conf_tpch.sql ***version_no =*** 26.0

start_directory = DYNAMIC_MASTER_DB_CONNECTION ***parent_version_no =*** 93.1

step_text =

step_label = Start SqlServer ***step_id =*** 32 ***global_flag =*** 0

sequence_no = 1 ***step_level =*** 1 ***parent_step_id =*** 6 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** 1

execution_mechanism = 2 ***continuation_criteria =*** 1 ***failure_details =***

step_file_name = ***version_no =*** 6.0

start_directory = ***parent_version_no =*** 44.0

step_text =

```
c:
start sqlservr -E -c -x -g100 %TRACEFLAGS%
```

step_label = Check Directories ***step_id =*** 56 ***global_flag =*** 0

sequence_no = 1 ***step_level =*** 1 ***parent_step_id =*** 4 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** 1

execution_mechanism = 2 ***continuation_criteria =*** 1 ***failure_details =*** %SETUP_

step_file_name = ***version_no =*** 16.0

start_directory = C:\ ***parent_version_no =*** 41.0

step_text =

```
echo off
if exist %SETUP_DIR% echo SETUP_DIR found
if exist %SETUP_DIR%\%DBNAME% echo SETUP_DIR\DBNAME found
if exist %FLATFILE_DIR% echo FLATFILE_DIR found
if exist %OUTPUT_DIR% echo OUTPUT_DIR found
if exist %VALIDATION_DIR% echo VALIDATE_DIR found
if exist %TOOLS_DIR% echo TOOLS_DIR found
```

step_label = Check Numerical Quantities ***step_id =*** 57 ***global_flag =*** 0

sequence_no = 2 ***step_level =*** 1 ***parent_step_id =*** 4 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** 1

execution_mechanism = 2 ***continuation_criteria =*** 1 ***failure_details =*** %SETUP_

step_file_name = ***version_no =*** 4.0

start_directory = C:\ ***parent_version_no =*** 41.0

step_text =

```
echo off

if "%DBGEN_PARALLELISM%" EQU "" goto :eof
if "%UPDATE_SETS%" EQU "" goto :eof
if "%INSERT_SEGMENTS_PER_UPDATE_SET%" EQU "" goto :eof
if "%DELETE_SEGMENTS_PER_UPDATE_SET%" EQU "" goto :eof
if "%SCALEFACTOR%" EQU "" goto :eof

if %DBGEN_PARALLELISM% GEQ 1 echo DBGEN_PARALLELISM is large enough
if %DBGEN_PARALLELISM% LEQ 16 echo DBGEN_PARALLELISM is small enough
if %UPDATE_SETS% GEQ 1 echo UPDATE_SETS is large enough
if %UPDATE_SETS% LEQ 32 echo UPDATE_SETS is small enough
if %INSERT_SEGMENTS_PER_UPDATE_SET% GEQ 1 echo
INSERT_SEGMENTS_PER_UPDATE_SET is large enough
if %INSERT_SEGMENTS_PER_UPDATE_SET% LEQ 32 echo
INSERT_SEGMENTS_PER_UPDATE_SET is small enough
if %DELETE_SEGMENTS_PER_UPDATE_SET% GEQ 1 echo
DELETE_SEGMENTS_PER_UPDATE_SET is large enough
if %DELETE_SEGMENTS_PER_UPDATE_SET% LEQ 32 echo
DELETE_SEGMENTS_PER_UPDATE_SET is small enough
if %SCALEFACTOR% EQU .1 echo SCALEFACTOR is okay
if %SCALEFACTOR% EQU .3 echo SCALEFACTOR is okay
if %SCALEFACTOR% EQU 1 echo SCALEFACTOR is okay
if %SCALEFACTOR% EQU 10 echo SCALEFACTOR is okay
if %SCALEFACTOR% EQU 30 echo SCALEFACTOR is okay
if %SCALEFACTOR% EQU 100 echo SCALEFACTOR is okay
if %SCALEFACTOR% EQU 300 echo SCALEFACTOR is okay
if %SCALEFACTOR% EQU 1000 echo SCALEFACTOR is okay
if %SCALEFACTOR% EQU 3000 echo SCALEFACTOR is okay
```

step_label = Drop Tables ***step_id =*** 58 ***global_flag =*** 0

sequence_no = 1 ***step_level =*** 2 ***parent_step_id =*** 20 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** 1

execution_mechanism = 1 ***continuation_criteria =*** 1 ***failure_details =***

step_file_name = ***version_no =*** 15.0

start_directory = DYNAMIC_DB_CONNECTION ***parent_version_no =*** 8.1

step_text =

```
declare @table     sysname

declare tables cursor for
select name from sysobjects
  where sysobjects.type='U'

open tables
fetch tables into @table
while @@fetch_status = 0
  begin
    exec('drop table '+@table)
    fetch tables into @table
  end
```

step_label = Create Base Tables ***step_id =*** 59 ***global_flag =*** 0

sequence_no = 3 ***step_level =*** 2 ***parent_step_id =*** 20 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** 1

execution_mechanism = 1 ***continuation_criteria =*** 1 ***failure_details =***

step_file_name = %SETUP_DIR%\%DBNAME%\CreateTables.sql ***version_no =*** 15.0

start_directory = DYNAMIC_DB_CONNECTION ***parent_version_no =*** 8.1

step_text =

step_label = Pin Base and Other Tables ***step_id =*** 60 ***global_flag =*** 0

sequence_no = 4 ***step_level =*** 2 ***parent_step_id =*** 20 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** 1

execution_mechanism = 1 ***continuation_criteria =*** 2 ***failure_details =***

step_file_name = ***version_no =*** 12.0

start_directory = DYNAMIC_DB_CONNECTION ***parent_version_no =*** 8.1

step_text =

```
exec sp_tableoption 'NATION','pintable',1
exec sp_tableoption 'REGION','pintable',1
exec sp_tableoption 'PART', 'pintable', 1
exec sp_tableoption 'SUPPLIER', 'pintable', 1
exec sp_tableoption 'CUSTOMER', 'pintable', 1
```

step_label = CreateIndexStream1 ***step_id =*** 64 ***global_flag =*** 0
sequence_no = 1 ***step_level =*** 2 ***parent_step_id =*** 167 ***enabled_flag =*** -1
iterator_name = ***degree_parallelism*** 4
execution_mechanism = 1 ***continuation_criteria =*** 1 ***failure_details =***
step_file_name = %SETUP_DIR%\%DBNAME%\CreateIndexesStream1.sql ***version_no =*** 6.0
start_directory = DYNAMIC_DB_CONNECTION ***parent_version_no =*** 2.0
step_text =

step_label = CreateIndexStream2 ***step_id =*** 65 ***global_flag =*** 0
sequence_no = 2 ***step_level =*** 2 ***parent_step_id =*** 167 ***enabled_flag =*** -1
iterator_name = ***degree_parallelism*** 4
execution_mechanism = 1 ***continuation_criteria =*** 1 ***failure_details =***
step_file_name = %SETUP_DIR%\%DBNAME%\CreateIndexesStream2.sql ***version_no =*** 7.0
start_directory = DYNAMIC_DB_CONNECTION ***parent_version_no =*** 2.0
step_text =

step_label = CreateIndexStream3 ***step_id =*** 66 ***global_flag =*** 0
sequence_no = 3 ***step_level =*** 2 ***parent_step_id =*** 167 ***enabled_flag =*** -1
iterator_name = ***degree_parallelism*** 1
execution_mechanism = 1 ***continuation_criteria =*** 1 ***failure_details =***
step_file_name = %SETUP_DIR%\%DBNAME%\CreateIndexesStream3.sql ***version_no =*** 7.0
start_directory = DYNAMIC_DB_CONNECTION ***parent_version_no =*** 2.0
step_text =

step_label = CreateIndexStream4 ***step_id =*** 67 ***global_flag =*** 0
sequence_no = 4 ***step_level =*** 2 ***parent_step_id =*** 167 ***enabled_flag =*** -1
iterator_name = ***degree_parallelism*** 1
execution_mechanism = 1 ***continuation_criteria =*** 1 ***failure_details =***
step_file_name = %SETUP_DIR%\%DBNAME%\CreateIndexesStream4.sql ***version_no =*** 7.0
start_directory = DYNAMIC_DB_CONNECTION ***parent_version_no =*** 2.0
step_text =

step_label = Generate FlatFile-Original ***step_id =*** 69 ***global_flag =*** 0

sequence_no = 1 ***step_level =*** 2 ***parent_step_id =*** 14 ***enabled_flag =*** -1

iterator_name = PARALLEL_PROCESS ***degree_parallelism*** 8

execution_mechanism = 2 ***continuation_criteria =*** 1 ***failure_details =***

step_file_name = ***version_no =*** 10.0

start_directory = %FLATFILE_DIR% ***parent_version_no =*** 11.0

step_text =

```
%TOOLS_DIR%\DBGen\dbgen -fF -q -b %TOOLS_DIR%\dists.dss -s%SCALEFACTOR%
-C%DBGEN_PARALLELISM% -S%PARALLEL_PROCESS%
if %PARALLEL_PROCESS% NEQ 1 goto :EOF
if %DBGEN_PARALLELISM% NEQ 1 goto :EOF
if exist lineitem.tbl.1 del lineitem.tbl.1
rename lineitem.tbl lineitem.tbl.1
if exist orders.tbl.1 del orders.tbl.1
rename orders.tbl orders.tbl.1
if exist customer.tbl.1 del customer.tbl.1
rename customer.tbl customer.tbl.1
if exist part.tbl.1 del part.tbl.1
rename part.tbl part.tbl.1
if exist supplier.tbl.1 del supplier.tbl.1
rename supplier.tbl supplier.tbl.1
if exist partsupp.tbl.1 del partsupp.tbl.1
rename partsupp.tbl partsupp.tbl.1
```

step_label = Create Statistics ***step_id =*** 70 ***global_flag =*** 0

sequence_no = 1 ***step_level =*** 2 ***parent_step_id =*** 26 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** 1

execution_mechanism = 1 ***continuation_criteria =*** 2 ***failure_details =***

step_file_name = ***version_no =*** 1.0

start_directory = DYNAMIC_DB_CONNECTION ***parent_version_no =*** 16.1

step_text =

```
sp_createstats
```

step_label = Turn Off Auto Create Statistics ***step_id =*** 71 ***global_flag =*** 0

sequence_no = 2 ***step_level =*** 2 ***parent_step_id =*** 26 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** 1

execution_mechanism = 1 ***continuation_criteria =*** 2 ***failure_details =***

step_file_name = ***version_no =*** 1.0

start_directory = DYNAMIC_DB_CONNECTION ***parent_version_no =*** 16.1

step_text =

```
sp_dboption '%DBNAME%','auto create statistics','OFF'
```

step_label = Turn Off Update Statistics ***step_id =*** 72 ***global_flag =*** 0
sequence_no = 3 ***step_level =*** 2 ***parent_step_id =*** 26 ***enabled_flag =*** -1
iterator_name = ***degree_parallelism*** 1
execution_mechanism = 1 ***continuation_criteria =*** 2 ***failure_details =***
step_file_name = ***version_no =*** 1.0
start_directory = DYNAMIC_DB_CONNECTION ***parent_version_no =*** 16.1
step_text = sp_dboption '%DBNAME%','auto update statistics','OFF'

step_label = Create Clustered Indexes ***step_id =*** 143 ***global_flag =*** 0
sequence_no = 2 ***step_level =*** 1 ***parent_step_id =*** 10 ***enabled_flag =*** -1
iterator_name = ***degree_parallelism*** 4
execution_mechanism = 1 ***continuation_criteria =*** 1 ***failure_details =***
step_file_name = %SETUP_DIR%\%DBNAME%\CreateClusteredIndexes.sql ***version_no =*** 16.0
start_directory = DYNAMIC_DB_CONNECTION ***parent_version_no =*** 73.0
step_text =

step_label = Build TPC-H Auxilliary Timing Table ***step_id =*** 155 ***global_flag =*** 0
sequence_no = 2 ***step_level =*** 2 ***parent_step_id =*** 20 ***enabled_flag =*** -1
iterator_name = ***degree_parallelism*** 1
execution_mechanism = 1 ***continuation_criteria =*** 1 ***failure_details =***
step_file_name = %SETUP_DIR%\%DBNAME%\CreateBuildTimer.sql ***version_no =*** 24.0
start_directory = DYNAMIC_DB_CONNECTION ***parent_version_no =*** 8.1
step_text =

step_label = Create Insert Tables ***step_id =*** 156 ***global_flag =*** 0
sequence_no = 6 ***step_level =*** 1 ***parent_step_id =*** 11 ***enabled_flag =*** -1
iterator_name = INSERT_SEGMENT ***degree_parallelism*** 1
execution_mechanism = 1 ***continuation_criteria =*** 1 ***failure_details =***
step_file_name = %SETUP_DIR%\%DBNAME%\RefreshTables\CreateInsertTables.sql ***version_no =*** 29.0
start_directory = DYNAMIC_DB_CONNECTION ***parent_version_no =*** 71.1
step_text =

step_label = Create Delete Tables ***step_id =*** 157 ***global_flag =*** 0
sequence_no = 7 ***step_level =*** 1 ***parent_step_id =*** 11 ***enabled_flag =*** -1
iterator_name = DELETE_SEGMENT ***degree_parallelism*** 1
execution_mechanism = 1 ***continuation_criteria =*** 1 ***failure_details =***
step_file_name = %SETUP_DIR%\%DBNAME%\RefreshTables\CreateDeleteTables.sql ***version_no =*** 27.0
start_directory = DYNAMIC_DB_CONNECTION ***parent_version_no =*** 71.1
step_text =

step_label = Parallel Partitioned Table Load ***step_id =*** 158 ***global_flag =*** 0
sequence_no = 2 ***step_level =*** 1 ***parent_step_id =*** 9 ***enabled_flag =*** -1
iterator_name = TABLE ***degree_parallelism*** %TABLE_LOAD_PARALLELISM%
execution_mechanism = 0 ***continuation_criteria =*** 0 ***failure_details =***
step_file_name = ***version_no =*** 16.0
start_directory = ***parent_version_no =*** 76.0
step_text =

step_label = Original-Load Partitioned Tables ***step_id =*** 159 ***global_flag =*** 0
sequence_no = 1 ***step_level =*** 2 ***parent_step_id =*** 158 ***enabled_flag =*** -1
iterator_name = PARALLEL_PROCESS ***degree_parallelism*** 1
execution_mechanism = 1 ***continuation_criteria =*** 1 ***failure_details =***
step_file_name = ***version_no =*** 5.0
start_directory = DYNAMIC_DB_CONNECTION ***parent_version_no =*** 16.0
step_text =

```
bulk insert %DBNAME%..%TABLE%
from '%FLATFILE_DIR%\%TABLE%.tbl.%PARALLEL_PROCESS%'
with (FieldTerminator = '|', RowTerminator ='|\n',tablock)
```

step_label = Parallel Load Simple Tables ***step_id =*** 160 ***global_flag =*** 0
sequence_no = 3 ***step_level =*** 1 ***parent_step_id =*** 9 ***enabled_flag =*** -1
iterator_name = ***degree_parallelism*** %TABLE_LOAD_PARALLELISM%
execution_mechanism = 0 ***continuation_criteria =*** 0 ***failure_details =***
step_file_name = ***version_no =*** 13.0
start_directory = ***parent_version_no =*** 76.0
step_text =

step_label = Load Nation Table ***step_id =*** 161 ***global_flag =*** 0

sequence_no = 1 ***step_level =*** 2 ***parent_step_id =*** 160 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** 1

execution_mechanism = 1 ***continuation_criteria =*** 1 ***failure_details =***

step_file_name = ***version_no =*** 2.0

start_directory = DYNAMIC_DB_CONNECTION ***parent_version_no =*** 13.0

step_text =

```
bulk insert %DBNAME%..NATION
from  '%FLATFILE_DIR%\nation.tbl'
with (FieldTerminator = '|', RowTerminator ='|\n',tablock)
```

step_label = Load Region Table ***step_id =*** 162 ***global_flag =*** 0

sequence_no = 2 ***step_level =*** 2 ***parent_step_id =*** 160 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** 1

execution_mechanism = 1 ***continuation_criteria =*** 1 ***failure_details =***

step_file_name = ***version_no =*** 2.0

start_directory = DYNAMIC_DB_CONNECTION ***parent_version_no =*** 13.0

step_text =

```
bulk insert %DBNAME%..REGION
from '%FLATFILE_DIR%\region.tbl'
with (FieldTerminator = '|', RowTerminator ='|\n',tablock)
```

step_label = End of Load ***step_id =*** 163 ***global_flag =*** 0

sequence_no = 8 ***step_level =*** 1 ***parent_step_id =*** 11 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** 1

execution_mechanism = 1 ***continuation_criteria =*** 1 ***failure_details =***

step_file_name = ***version_no =*** 19.0

start_directory = DYNAMIC_DB_CONNECTION ***parent_version_no =*** 71.1

step_text =

```
UPDATE TPCH_AUX_TABLE SET LoadFinish = getdate()
```

step_label = Create NC Indexes in Parallel ***step_id =*** 167 ***global_flag =*** 0

sequence_no = 3 ***step_level =*** 1 ***parent_step_id =*** 10 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** %INDEX_CREATE_PARALLELISM%

execution_mechanism = 0 ***continuation_criteria =*** 0 ***failure_details =***

step_file_name = ***version_no =*** 2.0

start_directory = ***parent_version_no =*** 73.0

step_text =

step_label = Wait For SQL Server ***step_id =*** 341 ***global_flag =*** -1

sequence_no = 3 ***step_level =*** 0 ***parent_step_id =*** 0 ***enabled_flag =*** 0

iterator_name = ***degree_parallelism*** 0

execution_mechanism = 2 ***continuation_criteria =*** 0 ***failure_details =***

step_file_name = ***version_no =*** 12.0

start_directory = %TOOLS_DIR% ***parent_version_no =*** 0.0

step_text = %TOOLS_DIR%\Utility\sleep 180

step_label = Backup TPC-H Database (If not used for ACID) ***step_id =*** 359 ***global_flag =*** 0

sequence_no = 9 ***step_level =*** 0 ***parent_step_id =*** 0 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** 1

execution_mechanism = 0 ***continuation_criteria =*** 0 ***failure_details =***

step_file_name = ***version_no =*** 45.0

start_directory = ***parent_version_no =*** 0.0

step_text =

step_label = Create Backup Device(s) ***step_id =*** 360 ***global_flag =*** 0

sequence_no = 1 ***step_level =*** 1 ***parent_step_id =*** 359 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** 1

execution_mechanism = 1 ***continuation_criteria =*** 2 ***failure_details =***

step_file_name = %SETUP_DIR%\%DBNAME%\CreateBackupDevices.sql ***version_no =*** 8.0

start_directory = DYNAMIC_MASTER_DB_CONNECTION ***parent_version_no =*** 45.0

step_text =

step_label = Execute the backup ***step_id =*** 361 ***global_flag =*** 0
sequence_no = 2 ***step_level =*** 1 ***parent_step_id =*** 359 ***enabled_flag =*** -1
iterator_name = ***degree_parallelism*** 1
execution_mechanism = 1 ***continuation_criteria =*** 2 ***failure_details =***
step_file_name = %SETUP_DIR%\%DBNAME%\BackupDatabase.sql ***version_no =*** 5.0
start_directory = DYNAMIC_DB_CONNECTION ***parent_version_no =*** 45.0
step_text =

step_label = Generate Test QGen Seed ***step_id =*** 424 ***global_flag =*** 0
sequence_no = 11 ***step_level =*** 1 ***parent_step_id =*** 11 ***enabled_flag =*** 0
iterator_name = ***degree_parallelism*** 1
execution_mechanism = 1 ***continuation_criteria =*** 2 ***failure_details =***
step_file_name = ***version_no =*** 8.0
start_directory = DYNAMIC_DB_CONNECTION ***parent_version_no =*** 71.1
step_text =

```
--
-- Update the benchmark auxillary table
--
UPDATE TPCH_AUX_TABLE SET QgenSeed=%TEST_QGEN_SEED%

SELECT * from TPCH_AUX_TABLE
```

step_label = Capture Database and Table Space Used ***step_id =*** 431 ***global_flag =*** 0
sequence_no = 10 ***step_level =*** 0 ***parent_step_id =*** 0 ***enabled_flag =*** -1
iterator_name = ***degree_parallelism*** 1
execution_mechanism = 0 ***continuation_criteria =*** 0 ***failure_details =***
step_file_name = ***version_no =*** 30.0
start_directory = ***parent_version_no =*** 0.0
step_text =

step_label = Execute DBCC UPDATEUSAGE ***step_id =*** 432 ***global_flag =*** 0

sequence_no = 1 ***step_level =*** 1 ***parent_step_id =*** 431 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** 1

execution_mechanism = 1 ***continuation_criteria =*** 2 ***failure_details =***

step_file_name = ***version_no =*** 1.0

start_directory = DYNAMIC_DB_CONNECTION ***parent_version_no =*** 30.0

step_text =

```
--
-- Correct any potential inaccuracies in the system tables before running sp_spaceused
--

dbcc updateusage (%DBNAME%)
GO

dbcc updateusage (tempdb)
GO
```

step_label = Execute SpaceUsed Procedure ***step_id =*** 433 ***global_flag =*** 0

sequence_no = 2 ***step_level =*** 1 ***parent_step_id =*** 431 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** 1

execution_mechanism = 1 ***continuation_criteria =*** 2 ***failure_details =***

step_file_name = ***version_no =*** 0.0

start_directory = DYNAMIC_DB_CONNECTION ***parent_version_no =*** 30.0

step_text =

```
sp_spaceused
GO
sp_spaceused 'REGION'
GO
sp_spaceused 'NATION'
GO
sp_spaceused 'PART'
GO
sp_spaceused 'SUPPLIER'
GO
sp_spaceused 'PARTSUPP'
GO
sp_spaceused 'CUSTOMER'
GO
sp_spaceused 'ORDERS'
GO
sp_spaceused 'LINEITEM'
GO
```

step_label = Execute Table Row Counts ***step_id =*** 434 ***global_flag =*** 0

sequence_no = 3 ***step_level =*** 1 ***parent_step_id =*** 431 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** 1

execution_mechanism = 1 ***continuation_criteria =*** 2 ***failure_details =***

step_file_name = ***version_no =*** 1.0

start_directory = DYNAMIC_DB_CONNECTION ***parent_version_no =*** 30.0

step_text =

```
print 'Count of REGION Table'
GO
select count(*) from REGION
GO
print 'Count of NATION Table'
GO
select count(*) from NATION
GO
print 'Count of PART Table'
GO
select count(*) from PART
GO
print 'Count of SUPPLIER Table'
GO
select count(*) from SUPPLIER
GO
print 'Count of PARTSUPP Table'
GO
select count(*) from PARTSUPP
GO
print 'Count of CUSTOMER Table'
GO
select count(*) from CUSTOMER
GO
print 'Count of ORDERS Table'
GO
select count(*) from ORDERS
GO
print 'Count of LINEITEM Table'
GO
select count(*) from LINEITEM
GO
```

step_label = Execute Log File Spaceused ***step_id =*** 435 ***global_flag =*** 0

sequence_no = 4 ***step_level =*** 1 ***parent_step_id =*** 431 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** 1

execution_mechanism = 1 ***continuation_criteria =*** 2 ***failure_details =***

step_file_name = ***version_no =*** 0.0

start_directory = DYNAMIC_DB_CONNECTION ***parent_version_no =*** 30.0

step_text =

```
dbcc sqlperf(logspace)
```

step_label = Execute TempDB spaceused ***step_id =*** 436 ***global_flag =*** 0

sequence_no = 5 ***step_level =*** 1 ***parent_step_id =*** 431 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** 1

execution_mechanism = 1 ***continuation_criteria =*** 2 ***failure_details =***

step_file_name = ***version_no =*** 2.0

start_directory = DYNAMIC_DB_CONNECTION ***parent_version_no =*** 30.0

step_text =

```
use tempdb
GO

sp_spaceused
GO

use %DBNAME%
GO
```

step_label = Store Load Starting Timestamp ***step_id =*** 445 ***global_flag =*** 0

sequence_no = 1 ***step_level =*** 1 ***parent_step_id =*** 7 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** 1

execution_mechanism = 1 ***continuation_criteria =*** 2 ***failure_details =***

step_file_name = ***version_no =*** 19.0

start_directory = DYNAMIC_MASTER_DB_CONNECTION ***parent_version_no =*** 93.1

step_text =

```
--    Create temporary table for timing in the Master Database
--
--            This is just temporary until we build the TPCH_AUX_TABLE later
--


--
--  Delete any existing tpch_temp_timer table
--
if exists ( select name from sysobjects where name = 'tpch_temp_timer' )
      drop table tpch_temp_timer

--
--  Create the temporary table
--
create table tpch_temp_timer
(
      load_start_time                                    datetime
)

--
--  Store the starting time in the temporary table
--
insert          into tpch_temp_timer values (getdate())
```

step_label = Reset DB_Option "Select Into" ***step_id =*** 446 ***global_flag =*** 0
sequence_no = 2 ***step_level =*** 1 ***parent_step_id =*** 11 ***enabled_flag =*** -1
iterator_name = ***degree_parallelism*** 1
execution_mechanism = 1 ***continuation_criteria =*** 1 ***failure_details =***
step_file_name = ***version_no =*** 17.0
start_directory = DYNAMIC_DB_CONNECTION ***parent_version_no =*** 71.1
step_text = sp_dboption %DBNAME%, 'select ',false

step_label = Reset DB_Option "Trunc" ***step_id =*** 447 ***global_flag =*** 0
sequence_no = 3 ***step_level =*** 1 ***parent_step_id =*** 11 ***enabled_flag =*** -1
iterator_name = ***degree_parallelism*** 1
execution_mechanism = 1 ***continuation_criteria =*** 1 ***failure_details =***
step_file_name = ***version_no =*** 17.0
start_directory = DYNAMIC_DB_CONNECTION ***parent_version_no =*** 71.1
step_text = sp_dboption %DBNAME%, 'trunc. ',false

step_label = Backup TPC-H Database (Only if using backup to satisfy ***step_id =*** 448 ***global_flag =*** 0
sequence_no = 4 ***step_level =*** 1 ***parent_step_id =*** 11 ***enabled_flag =*** 0
iterator_name = ***degree_parallelism*** 1
execution_mechanism = 0 ***continuation_criteria =*** 0 ***failure_details =***
step_file_name = ***version_no =*** 2.0
start_directory = ***parent_version_no =*** 71.1
step_text =

step_label = Create Backup Device(s) (For ACID) ***step_id =*** 449 ***global_flag =*** 0
sequence_no = 1 ***step_level =*** 2 ***parent_step_id =*** 448 ***enabled_flag =*** -1
iterator_name = ***degree_parallelism*** 1
execution_mechanism = 1 ***continuation_criteria =*** 2 ***failure_details =***
step_file_name = %SETUP_DIR%\%DBNAME%\CreateBackupDevices.sql ***version_no =*** 4.0
start_directory = DYNAMIC_MASTER_DB_CONNECTION ***parent_version_no =*** 2.0
step_text =

step_label = Execute the backup (For ACID) ***step_id =*** 450 ***global_flag =*** 0

sequence_no = 2 ***step_level =*** 2 ***parent_step_id =*** 448 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** 1

execution_mechanism = 1 ***continuation_criteria =*** 2 ***failure_details =***

step_file_name = %SETUP_DIR%\%DBNAME%\BackupDatabase.sql ***version_no =*** 0.0

start_directory = DYNAMIC_DB_CONNECTION ***parent_version_no =*** 2.0

step_text =

step_label = Generate Validation Queries via QGEN ***step_id =*** 451 ***global_flag =*** 0

sequence_no = 1 ***step_level =*** 1 ***parent_step_id =*** 12 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** %MAX_STREAMS%

execution_mechanism = 0 ***continuation_criteria =*** 0 ***failure_details =***

step_file_name = ***version_no =*** 1.0

start_directory = ***parent_version_no =*** 8.1

step_text =

step_label = Generate Validation Queries ***step_id =*** 452 ***global_flag =*** 0

sequence_no = 1 ***step_level =*** 2 ***parent_step_id =*** 451 ***enabled_flag =*** -1

iterator_name = QUERY ***degree_parallelism*** 1

execution_mechanism = 2 ***continuation_criteria =*** 2 ***failure_details =***

step_file_name = ***version_no =*** 0.0

start_directory = %TEMPLATE_DIR% ***parent_version_no =*** 1.0

step_text =

```
%TOOLS_DIR%\QGen\qgen -d -b %TOOLS_DIR%\dists.dss  %QUERY% >
%VALIDATION_DIR%\Queries\v%QUERY%.sql
```

step_label = Execute Validation Queries ***step_id =*** 453 ***global_flag =*** 0

sequence_no = 2 ***step_level =*** 1 ***parent_step_id =*** 12 ***enabled_flag =*** 0

iterator_name = ***degree_parallelism*** 1

execution_mechanism = 0 ***continuation_criteria =*** 0 ***failure_details =***

step_file_name = ***version_no =*** 1.0

start_directory = ***parent_version_no =*** 8.1

step_text =

step_label = Validation Query 1 ***step_id =*** 454 ***global_flag =*** 0

sequence_no = 1 ***step_level =*** 2 ***parent_step_id =*** 453 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** 1

execution_mechanism = 1 ***continuation_criteria =*** 2 ***failure_details =***

step_file_name = %VALIDATION_DIR%\Queries\v1.sql ***version_no =*** 1.0

start_directory = VALIDATION_STREAM_CONN ***parent_version_no =*** 1.0

step_text =

step_label = Validation Query 2 ***step_id =*** 455 ***global_flag =*** 0

sequence_no = 2 ***step_level =*** 2 ***parent_step_id =*** 453 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** 1

execution_mechanism = 1 ***continuation_criteria =*** 2 ***failure_details =***

step_file_name = %VALIDATION_DIR%\Queries\v2.sql ***version_no =*** 1.0

start_directory = VALIDATION_STREAM_CONN ***parent_version_no =*** 1.0

step_text =

step_label = Validation Query 3 ***step_id =*** 456 ***global_flag =*** 0

sequence_no = 3 ***step_level =*** 2 ***parent_step_id =*** 453 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** 1

execution_mechanism = 1 ***continuation_criteria =*** 2 ***failure_details =***

step_file_name = %VALIDATION_DIR%\Queries\v3.sql ***version_no =*** 1.0

start_directory = VALIDATION_STREAM_CONN ***parent_version_no =*** 1.0

step_text =

step_label = Validation Query 4 ***step_id =*** 457 ***global_flag =*** 0

sequence_no = 4 ***step_level =*** 2 ***parent_step_id =*** 453 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** 1

execution_mechanism = 1 ***continuation_criteria =*** 2 ***failure_details =***

step_file_name = %VALIDATION_DIR%\Queries\v4.sql ***version_no =*** 1.0

start_directory = VALIDATION_STREAM_CONN ***parent_version_no =*** 1.0

step_text =

step_label = Validation Query 5 ***step_id =*** 458 ***global_flag =*** 0
sequence_no = 5 ***step_level =*** 2 ***parent_step_id =*** 453 ***enabled_flag =*** -1
iterator_name = ***degree_parallelism*** 1
execution_mechanism = 1 ***continuation_criteria =*** 2 ***failure_details =***
step_file_name = %VALIDATION_DIR%\Queries\v5.sql ***version_no =*** 1.0
start_directory = VALIDATION_STREAM_CONN ***parent_version_no =*** 1.0
step_text =

step_label = Validation Query 6 ***step_id =*** 459 ***global_flag =*** 0
sequence_no = 6 ***step_level =*** 2 ***parent_step_id =*** 453 ***enabled_flag =*** -1
iterator_name = ***degree_parallelism*** 1
execution_mechanism = 1 ***continuation_criteria =*** 2 ***failure_details =***
step_file_name = %VALIDATION_DIR%\Queries\v6.sql ***version_no =*** 1.0
start_directory = VALIDATION_STREAM_CONN ***parent_version_no =*** 1.0
step_text =

step_label = Validation Query 7 ***step_id =*** 460 ***global_flag =*** 0
sequence_no = 7 ***step_level =*** 2 ***parent_step_id =*** 453 ***enabled_flag =*** -1
iterator_name = ***degree_parallelism*** 1
execution_mechanism = 1 ***continuation_criteria =*** 2 ***failure_details =***
step_file_name = %VALIDATION_DIR%\Queries\v7.sql ***version_no =*** 1.0
start_directory = VALIDATION_STREAM_CONN ***parent_version_no =*** 1.0
step_text =

step_label = Validation Query 8 ***step_id =*** 461 ***global_flag =*** 0
sequence_no = 8 ***step_level =*** 2 ***parent_step_id =*** 453 ***enabled_flag =*** -1
iterator_name = ***degree_parallelism*** 1
execution_mechanism = 1 ***continuation_criteria =*** 2 ***failure_details =***
step_file_name = %VALIDATION_DIR%\Queries\v8.sql ***version_no =*** 1.0
start_directory = VALIDATION_STREAM_CONN ***parent_version_no =*** 1.0
step_text =

step_label = Validation Query 9 ***step_id =*** 462 ***global_flag =*** 0

sequence_no = 9 ***step_level =*** 2 ***parent_step_id =*** 453 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** 1

execution_mechanism = 1 ***continuation_criteria =*** 2 ***failure_details =***

step_file_name = %VALIDATION_DIR%\Queries\v9.sql ***version_no =*** 1.0

start_directory = VALIDATION_STREAM_CONN ***parent_version_no =*** 1.0

step_text =

step_label = Validation Query 10 ***step_id =*** 463 ***global_flag =*** 0

sequence_no = 10 ***step_level =*** 2 ***parent_step_id =*** 453 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** 1

execution_mechanism = 1 ***continuation_criteria =*** 2 ***failure_details =***

step_file_name = %VALIDATION_DIR%\Queries\v10.sql ***version_no =*** 1.0

start_directory = VALIDATION_STREAM_CONN ***parent_version_no =*** 1.0

step_text =

step_label = Validation Query 11 ***step_id =*** 464 ***global_flag =*** 0

sequence_no = 11 ***step_level =*** 2 ***parent_step_id =*** 453 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** 1

execution_mechanism = 1 ***continuation_criteria =*** 2 ***failure_details =***

step_file_name = %VALIDATION_DIR%\Queries\v11.sql ***version_no =*** 1.0

start_directory = VALIDATION_STREAM_CONN ***parent_version_no =*** 1.0

step_text =

step_label = Validation Query 12 ***step_id =*** 465 ***global_flag =*** 0

sequence_no = 12 ***step_level =*** 2 ***parent_step_id =*** 453 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** 1

execution_mechanism = 1 ***continuation_criteria =*** 2 ***failure_details =***

step_file_name = %VALIDATION_DIR%\Queries\v12.sql ***version_no =*** 1.0

start_directory = VALIDATION_STREAM_CONN ***parent_version_no =*** 1.0

step_text =

step_label = Validation Query 13 ***step_id =*** 466 ***global_flag =*** 0
sequence_no = 13 ***step_level =*** 2 ***parent_step_id =*** 453 ***enabled_flag =*** -1
iterator_name = ***degree_parallelism*** 1
execution_mechanism = 1 ***continuation_criteria =*** 2 ***failure_details =***
step_file_name = %VALIDATION_DIR%\Queries\v13.sql ***version_no =*** 1.0
start_directory = VALIDATION_STREAM_CONN ***parent_version_no =*** 1.0
step_text =

step_label = Validation Query 14 ***step_id =*** 467 ***global_flag =*** 0
sequence_no = 14 ***step_level =*** 2 ***parent_step_id =*** 453 ***enabled_flag =*** -1
iterator_name = ***degree_parallelism*** 1
execution_mechanism = 1 ***continuation_criteria =*** 2 ***failure_details =***
step_file_name = %VALIDATION_DIR%\Queries\v14.sql ***version_no =*** 1.0
start_directory = VALIDATION_STREAM_CONN ***parent_version_no =*** 1.0
step_text =

step_label = Validation Query 15 ***step_id =*** 468 ***global_flag =*** 0
sequence_no = 15 ***step_level =*** 2 ***parent_step_id =*** 453 ***enabled_flag =*** -1
iterator_name = ***degree_parallelism*** 1
execution_mechanism = 1 ***continuation_criteria =*** 2 ***failure_details =***
step_file_name = %VALIDATION_DIR%\Queries\v15.sql ***version_no =*** 1.0
start_directory = VALIDATION_STREAM_CONN ***parent_version_no =*** 1.0
step_text =

step_label = Validation Query 16 ***step_id =*** 469 ***global_flag =*** 0
sequence_no = 16 ***step_level =*** 2 ***parent_step_id =*** 453 ***enabled_flag =*** -1
iterator_name = ***degree_parallelism*** 1
execution_mechanism = 1 ***continuation_criteria =*** 2 ***failure_details =***
step_file_name = %VALIDATION_DIR%\Queries\v16.sql ***version_no =*** 1.0
start_directory = VALIDATION_STREAM_CONN ***parent_version_no =*** 1.0
step_text =

step_label = Validation Query 17 ***step_id =*** 470 ***global_flag =*** 0
sequence_no = 17 ***step_level =*** 2 ***parent_step_id =*** 453 ***enabled_flag =*** -1
iterator_name = ***degree_parallelism*** 1
execution_mechanism = 1 ***continuation_criteria =*** 2 ***failure_details =***
step_file_name = %VALIDATION_DIR%\Queries\v17.sql ***version_no =*** 0.0
start_directory = VALIDATION_STREAM_CONN ***parent_version_no =*** 1.0
step_text =

step_label = Validation Query 18 ***step_id =*** 471 ***global_flag =*** 0
sequence_no = 18 ***step_level =*** 2 ***parent_step_id =*** 453 ***enabled_flag =*** -1
iterator_name = ***degree_parallelism*** 1
execution_mechanism = 1 ***continuation_criteria =*** 2 ***failure_details =***
step_file_name = %VALIDATION_DIR%\Queries\v18.sql ***version_no =*** 0.0
start_directory = VALIDATION_STREAM_CONN ***parent_version_no =*** 1.0
step_text =

step_label = Validation Query 19 ***step_id =*** 472 ***global_flag =*** 0
sequence_no = 19 ***step_level =*** 2 ***parent_step_id =*** 453 ***enabled_flag =*** -1
iterator_name = ***degree_parallelism*** 1
execution_mechanism = 1 ***continuation_criteria =*** 2 ***failure_details =***
step_file_name = %VALIDATION_DIR%\Queries\v19.sql ***version_no =*** 0.0
start_directory = VALIDATION_STREAM_CONN ***parent_version_no =*** 1.0
step_text =

step_label = Validation Query 20 ***step_id =*** 473 ***global_flag =*** 0
sequence_no = 20 ***step_level =*** 2 ***parent_step_id =*** 453 ***enabled_flag =*** -1
iterator_name = ***degree_parallelism*** 1
execution_mechanism = 1 ***continuation_criteria =*** 2 ***failure_details =***
step_file_name = %VALIDATION_DIR%\Queries\v20.sql ***version_no =*** 0.0
start_directory = VALIDATION_STREAM_CONN ***parent_version_no =*** 1.0
step_text =

step_label = Validation Query 21 ***step_id =*** 474 ***global_flag =*** 0
sequence_no = 21 ***step_level =*** 2 ***parent_step_id =*** 453 ***enabled_flag =*** -1
iterator_name = ***degree_parallelism*** 1
execution_mechanism = 1 ***continuation_criteria =*** 2 ***failure_details =***
step_file_name = %VALIDATION_DIR%\Queries\v21.sql ***version_no =*** 0.0
start_directory = VALIDATION_STREAM_CONN ***parent_version_no =*** 1.0
step_text =

step_label = Validation Query 22 ***step_id =*** 475 ***global_flag =*** 0
sequence_no = 22 ***step_level =*** 2 ***parent_step_id =*** 453 ***enabled_flag =*** -1
iterator_name = ***degree_parallelism*** 1
execution_mechanism = 1 ***continuation_criteria =*** 2 ***failure_details =***
step_file_name = %VALIDATION_DIR%\Queries\v22.sql ***version_no =*** 0.0
start_directory = VALIDATION_STREAM_CONN ***parent_version_no =*** 1.0
step_text =

step_label = Generate Queries via QGen ***step_id =*** 476 ***global_flag =*** 0
sequence_no = 11 ***step_level =*** 0 ***parent_step_id =*** 0 ***enabled_flag =*** -1
iterator_name = ***degree_parallelism*** 1
execution_mechanism = 0 ***continuation_criteria =*** 0 ***failure_details =***
step_file_name = ***version_no =*** 19.0
start_directory = ***parent_version_no =*** 0.0
step_text =

step_label = Generate Power Queries ***step_id =*** 477 ***global_flag =*** 0
sequence_no = 1 ***step_level =*** 1 ***parent_step_id =*** 476 ***enabled_flag =*** -1
iterator_name = ***degree_parallelism*** 1
execution_mechanism = 0 ***continuation_criteria =*** 0 ***failure_details =***
step_file_name = ***version_no =*** 2.0
start_directory = ***parent_version_no =*** 19.0
step_text =

step_label = Generate Power Query Directory ***step_id =*** 478 ***global_flag =*** 0

sequence_no = 1 ***step_level =*** 2 ***parent_step_id =*** 477 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** 1

execution_mechanism = 2 ***continuation_criteria =*** 1 ***failure_details =***

step_file_name = ***version_no =*** 3.0

start_directory = %RUN_DIR% ***parent_version_no =*** 2.0

step_text =

```
if not exist %QUERY_DIR%\Power mkdir %QUERY_DIR%\Power
```

step_label = Generate QGen Command File ***step_id =*** 479 ***global_flag =*** 0

sequence_no = 2 ***step_level =*** 2 ***parent_step_id =*** 477 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** 1

execution_mechanism = 2 ***continuation_criteria =*** 1 ***failure_details =***

step_file_name = ***version_no =*** 7.0

start_directory = %TEMPLATE_DIR% ***parent_version_no =*** 2.0

step_text =

```
::
:: First use OSQL to grab the QgenSeed value from TPCH_AUX_TABLE
::
osql -Usa -P -n -d%DBNAME% -w 255 -i%SETUP_DIR%\Generate\GenQGENcmd.sql >
%SETUP_DIR%\Generate\QgenCmd.cmd
```

step_label = Parallel Power Query Generation ***step_id =*** 480 ***global_flag =*** 0

sequence_no = 3 ***step_level =*** 2 ***parent_step_id =*** 477 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** %MAX_STREAMS%

execution_mechanism = 0 ***continuation_criteria =*** 0 ***failure_details =***

step_file_name = ***version_no =*** 0.4

start_directory = ***parent_version_no =*** 2.0

step_text =

step_label = Power - Execute Generated QGen Command File ***step_id =*** 481 ***global_flag =*** 0

sequence_no = 1 ***step_level =*** 3 ***parent_step_id =*** 480 ***enabled_flag =*** -1

iterator_name = QUERY ***degree_parallelism*** 1

execution_mechanism = 2 ***continuation_criteria =*** 1 ***failure_details =***

step_file_name = %SETUP_DIR%\Generate\QgenCmd.cmd ***version_no =*** 4.0

start_directory = %TEMPLATE_DIR% ***parent_version_no =*** 0.4

step_text =

step_label = Generate Stream Queries ***step_id =*** 482 ***global_flag =*** 0

sequence_no = 2 ***step_level =*** 1 ***parent_step_id =*** 476 ***enabled_flag =*** -1

iterator_name = STREAM_NUM ***degree_parallelism*** 1

execution_mechanism = 0 ***continuation_criteria =*** 0 ***failure_details =***

step_file_name = ***version_no =*** 5.0

start_directory = ***parent_version_no =*** 19.0

step_text =

step_label = Set Seed Value for Stream ***step_id =*** 483 ***global_flag =*** 0

sequence_no = 1 ***step_level =*** 2 ***parent_step_id =*** 482 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** 1

execution_mechanism = 1 ***continuation_criteria =*** 1 ***failure_details =***

step_file_name = ***version_no =*** 3.0

start_directory = DYNAMIC_DB_CONNECTION ***parent_version_no =*** 5.0

step_text =

```
--
--    Increment QGen Seed in TPCH_AUX_TABLE
--
UPDATE   TPCH_AUX_TABLE
SET          QgenSeed = QgenSeed + %STREAM_NUM%
```

step_label = Create Query Stream Directories ***step_id =*** 484 ***global_flag =*** 0

sequence_no = 2 ***step_level =*** 2 ***parent_step_id =*** 482 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** 1

execution_mechanism = 2 ***continuation_criteria =*** 2 ***failure_details =***

step_file_name = ***version_no =*** 1.0

start_directory = %QUERY_DIR% ***parent_version_no =*** 5.0

step_text =

```
if not exist %QUERY_DIR%\STREAM%STREAM_NUM% mkdir
%QUERY_DIR%\STREAM%STREAM_NUM%
```

step_label = Generate QGen Stream Command File ***step_id =*** 485 ***global_flag =*** 0

sequence_no = 3 ***step_level =*** 2 ***parent_step_id =*** 482 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** 1

execution_mechanism = 2 ***continuation_criteria =*** 2 ***failure_details =***

step_file_name = ***version_no =*** 3.0

start_directory = %TEMPLATE_DIR% ***parent_version_no =*** 5.0

step_text =

```
::
:: First use OSQL to grab the QgenSeed value from TPCH_AUX_TABLE
::
osql -Usa -P -n -d%DBNAME% -w 255 -i%SETUP_DIR%\Generate\GenQgenStreamCmd.sql >
%SETUP_DIR%\Generate\QgenStreamCmd.cmd
```

step_label = Parallel Stream Query Generation ***step_id =*** 486 ***global_flag =*** 0

sequence_no = 4 ***step_level =*** 2 ***parent_step_id =*** 482 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** %MAX_STREAMS%

execution_mechanism = 0 ***continuation_criteria =*** 0 ***failure_details =***

step_file_name = ***version_no =*** 0.3

start_directory = ***parent_version_no =*** 5.0

step_text =

step_label = Throughput - Execute Generated QGen Command File ***step_id =*** 487 ***global_flag =*** 0

sequence_no = 1 ***step_level =*** 3 ***parent_step_id =*** 486 ***enabled_flag =*** -1

iterator_name = QUERY ***degree_parallelism*** 1

execution_mechanism = 2 ***continuation_criteria =*** 2 ***failure_details =***

step_file_name = %SETUP_DIR%\Generate\QgenStreamCmd.cmd ***version_no =*** 3.0

start_directory = %TEMPLATE_DIR% ***parent_version_no =*** 0.3

step_text =

step_label = Re-set QGen Seed Value ***step_id =*** 488 ***global_flag =*** 0

sequence_no = 5 ***step_level =*** 2 ***parent_step_id =*** 482 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** 1

execution_mechanism = 1 ***continuation_criteria =*** 2 ***failure_details =***

step_file_name = ***version_no =*** 0.0

start_directory = DYNAMIC_DB_CONNECTION ***parent_version_no =*** 5.0

step_text =

```
--
--   Resets the QGen seed kept in the temporary table
--
UPDATE    TPCH_AUX_TABLE
SET          QgenSeed  =  QgenSeed - %STREAM_NUM%
```

step_label = Install Refresh Function Stored Procedures ***step_id =*** 492 ***global_flag =*** 0

sequence_no = 5 ***step_level =*** 1 ***parent_step_id =*** 11 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** %MAX_STREAMS%

execution_mechanism = 0 ***continuation_criteria =*** 0 ***failure_details =***

step_file_name = ***version_no =*** 15.0

start_directory = ***parent_version_no =*** 71.1

step_text =

step_label = Install RF1 Stored Procedure(s) ***step_id =*** 493 ***global_flag =*** 0
sequence_no = 1 ***step_level =*** 2 ***parent_step_id =*** 492 ***enabled_flag =*** -1
iterator_name = INSERT_SEGMENT ***degree_parallelism*** 1
execution_mechanism = 1 ***continuation_criteria =*** 2 ***failure_details =***
step_file_name = %SETUP_DIR%\StoredProcs\CreateRF1Proc.sql ***version_no =*** 3.0
start_directory = DYNAMIC_DB_CONNECTION ***parent_version_no =*** 15.0
step_text =

step_label = Install RF2 Stored Procedure(s) ***step_id =*** 494 ***global_flag =*** 0
sequence_no = 2 ***step_level =*** 2 ***parent_step_id =*** 492 ***enabled_flag =*** -1
iterator_name = DELETE_SEGMENT ***degree_parallelism*** 1
execution_mechanism = 1 ***continuation_criteria =*** 2 ***failure_details =***
step_file_name = %SETUP_DIR%\StoredProcs\CreateRF2Proc.sql ***version_no =*** 1.0
start_directory = DYNAMIC_DB_CONNECTION ***parent_version_no =*** 15.0
step_text =

step_label = Execute the second backup ***step_id =*** 915 ***global_flag =*** 0
sequence_no = 3 ***step_level =*** 1 ***parent_step_id =*** 359 ***enabled_flag =*** 0
iterator_name = ***degree_parallelism*** 1
execution_mechanism = 1 ***continuation_criteria =*** 2 ***failure_details =***
step_file_name = %SETUP_DIR%\%DBNAME%\Backup-2-Database.sql ***version_no =*** 5.0
start_directory = DYNAMIC_DB_CONNECTION ***parent_version_no =*** 37.1
step_text =

step_label = Execute the second backup ***step_id =*** 919 ***global_flag =*** 0
sequence_no = 3 ***step_level =*** 2 ***parent_step_id =*** 448 ***enabled_flag =*** -1
iterator_name = ***degree_parallelism*** 1
execution_mechanism = 1 ***continuation_criteria =*** 2 ***failure_details =***
step_file_name = %SETUP_DIR%\%DBNAME%\Backup-2-Database.sql ***version_no =*** 4.0
start_directory = DYNAMIC_DB_CONNECTION ***parent_version_no =*** 0.0
step_text =

step_label = Test Wait for Sqlservr start up ***step_id =*** 927 ***global_flag =*** 0

sequence_no = 5 ***step_level =*** 1 ***parent_step_id =*** 7 ***enabled_flag =*** 0

iterator_name = ***degree_parallelism*** 1

execution_mechanism = 1 ***continuation_criteria =*** 2 ***failure_details =***

step_file_name = %SETUP_DIR%\Utility\do_nothing.sql ***version_no =*** 31.0

start_directory = DYNAMIC_MASTER_DB_CONNECTION ***parent_version_no =*** 93.1

step_text =

step_label = Test Wait for Sqlservr start up1 ***step_id =*** 928 ***global_flag =*** 0

sequence_no = 6 ***step_level =*** 1 ***parent_step_id =*** 7 ***enabled_flag =*** 0

iterator_name = ***degree_parallelism*** 1

execution_mechanism = 1 ***continuation_criteria =*** 2 ***failure_details =***

step_file_name = %SETUP_DIR%\Utility\do_nothing_1.sql ***version_no =*** 27.0

start_directory = DYNAMIC_MASTER_DB_CONNECTION ***parent_version_no =*** 93.1

step_text =

step_label = Wait to close Log file ***step_id =*** 931 ***global_flag =*** -1

sequence_no = 4 ***step_level =*** 0 ***parent_step_id =*** 0 ***enabled_flag =*** 0

iterator_name = ***degree_parallelism*** 0

execution_mechanism = 2 ***continuation_criteria =*** 0 ***failure_details =***

step_file_name = ***version_no =*** 1.0

start_directory = %TOOLS_DIR% ***parent_version_no =*** 0.0

step_text = %TOOLS_DIR%\Utility\sleep 10

Step_constraints

workspace_id	constraint_id	step_id	version_no	constraint_type	global_step_id	global_version_no	sequence_no
2	2	31	26.0	2	2	7.0	2
2	41	29	32.0	2	931	1.0	1
2	30	29	32.0	2	341	12.0	3
2	5	29	32.0	2	3	6.0	0
2	3	29	32.0	2	2	12.0	2

2	2	31	26.0	2	2	13.0	2
2	12	31	26.0	2	341	12.0	3
2	12	31	26.0	2	341	7.0	3
2	2	31	26.0	2	2	11.0	2
2	42	31	26.0	2	931	1.0	1
2	2	31	26.0	2	2	6.0	2
2	4	31	26.0	2	3	5.0	0
2	4	31	26.0	2	3	6.0	0
2	3	29	32.0	2	2	13.0	2
2	12	31	26.0	2	341	8.0	3
2	32	927	31.0	2	2	13.0	2
2	2	31	26.0	2	2	12.0	2
2	2	31	26.0	2	2	9.0	2
2	12	31	26.0	2	341	10.0	3
2	2	31	26.0	2	2	10.0	2
2	2	31	26.0	2	2	8.0	2
2	12	31	26.0	2	341	11.0	3
2	2	31	26.0	2	2	5.0	2
2	4	31	26.0	2	3	4.0	0
2	11	32	6.0	2	341	12.0	0
2	40	927	31.0	2	931	1.0	1
2	38	927	31.0	2	341	12.0	3
2	33	927	31.0	2	3	6.0	0
2	32	927	31.0	2	2	12.0	2
2	12	31	26.0	2	341	9.0	3

Iterator_values

workspace_id	step_id	version_no	type	iterator_value	sequence_no
2	157	27.0	1	1	0
2	69	10.0	2	%DBGEN_PARALLELISM%	0

2	159	5.0	3	1	0
2	158	16.0	4	LINEITEM	0
2	158	16.0	4	CUSTOMER	5
2	158	16.0	4	SUPPLIER	3
2	158	16.0	4	PARTSUPP	4
2	159	5.0	1	1	0
2	158	16.0	4	ORDERS	1
2	452	0.0	3	1	0
2	157	27.0	3	1	0
2	157	27.0	2	%DELETE_SEGMENTS_PER_UPDATE_SET%	0
2	156	29.0	1	1	0
2	156	29.0	2	%INSERT_SEGMENTS_PER_UPDATE_SET%	0
2	156	29.0	3	1	0
2	69	10.0	1	1	0
2	69	10.0	3	1	0
2	158	16.0	4	PART	2
2	482	5.0	1	1	0
2	494	1.0	2	%DELETE_SEGMENTS_PER_UPDATE_SET%	0
2	494	1.0	3	1	0
2	493	3.0	1	1	0
2	493	3.0	2	%INSERT_SEGMENTS_PER_UPDATE_SET%	0
2	493	3.0	3	1	0
2	487	3.0	2	22	0
2	159	5.0	2	%DBGEN_PARALLELISM%	0
2	487	3.0	1	1	0
2	494	1.0	1	1	0
2	482	5.0	2	%MAX_STREAMS%	0
2	482	5.0	3	1	0
2	481	4.0	3	1	0
2	481	4.0	2	22	0
2	481	4.0	1	1	0

2	452	0.0	1	1	0
2	452	0.0	2	22	0
2	487	3.0	3	1	0

3 TPC-H Power/Throughput Runs 0

Workspace_parameters

workspace_id = 3 ***parameter_id*** 39 ***parameter_type =*** 0
parameter_name SCALEFACTOR
parameter_valu 300

workspace_id = 3 ***parameter_id*** 19 ***parameter_type =*** 3
parameter_name OUTPUT_DIR
parameter_valu c:\OUTPUT\200210~2\783

workspace_id = 3 ***parameter_id*** 21 ***parameter_type =*** 0
parameter_name QUERY_DIR
parameter_valu %RUN_DIR%\Queries-6

workspace_id = 3 ***parameter_id*** 23 ***parameter_type =*** 0
parameter_name DBNAME
parameter_valu tpch300g

workspace_id = 3 ***parameter_id*** 25 ***parameter_type =*** 0
parameter_name DELETE_PARALLELISM
parameter_valu 32

workspace_id = 3 ***parameter_id*** 26 ***parameter_type =*** 0
parameter_name INSERT_PARALLELISM
parameter_valu 32

workspace_id = 3 ***parameter_id*** 28 ***parameter_type =*** 0
parameter_name DELETE_SEGMENTS_PER_UPDATE_SET
parameter_valu 32

workspace_id = 3 ***parameter_id*** 18 ***parameter_type =*** 0
parameter_name RUN_DIR
parameter_valu %KIT_DIR%\Run

workspace_id = 3 ***parameter_id*** 38 ***parameter_type =*** 0
parameter_name MAX_STREAMS
parameter_valu 6

workspace_id =	3	***parameter_id***	151	***parameter_type =***	3
parameter_name		RUN_ID			
parameter_valu		783			

workspace_id =	3	***parameter_id***	41	***parameter_type =***	3
parameter_name		DEFAULT_DIR			
parameter_valu		c:\OUTPUT\20021002_Audit_SMRunOnly			

workspace_id =	3	***parameter_id***	53	***parameter_type =***	0
parameter_name		RF_FLATFILE_DIR			
parameter_valu		f:\RF_Flat_Files-32			

workspace_id =	3	***parameter_id***	54	***parameter_type =***	0
parameter_name		TOOLS_DIR			
parameter_valu		%KIT_DIR%\Tools			

workspace_id =	3	***parameter_id***	64	***parameter_type =***	0
parameter_name		BATCH_SIZE			
parameter_valu		200			

workspace_id =	3	***parameter_id***	67	***parameter_type =***	0
parameter_name		KIT_DIR			
parameter_valu		c:\mstpch_FEB2002			

workspace_id =	3	***parameter_id***	125	***parameter_type =***	0
parameter_name		FLATFILE_DIR			
parameter_valu		F:\Flat_Files			

workspace_id =	3	***parameter_id***	147	***parameter_type =***	0
parameter_name		SETUP_DIR			
parameter_valu		%KIT_DIR%\Setup			

workspace_id =	3	***parameter_id***	29	***parameter_type =***	0
parameter_name		INSERT_SEGMENTS_PER_UPDATE_SET			
parameter_valu		32			

Connection_dtls

workspace_id	*connection_name_id*	*connection_name*	*connection_string_name*	*connection_type*
3	20	DYNAMIC_RF_DBCONNECTION	DBCONNECTION	2

3	18	THROUGHPUT_STREAM10_DB_CONNECTION	DBCONNECTION	1
3	17	THROUGHPUT_STREAM9_DB_CONNECTION	DBCONNECTION	1
3	16	THROUGHPUT_STREAM8_DB_CONNECTION	DBCONNECTION	1
3	15	THROUGHPUT_STREAM7_DB_CONNECTION	DBCONNECTION	1
3	14	THROUGHPUT_STREAM6_DB_CONNECTION	DBCONNECTION	1
3	13	THROUGHPUT_STREAM5_DB_CONNECTION	DBCONNECTION	1
3	12	THROUGHPUT_STREAM4_DB_CONNECTION	DBCONNECTION	1
3	11	THROUGHPUT_STREAM3_DB_CONNECTION	DBCONNECTION	1
3	10	THROUGHPUT_STREAM2_DB_CONNECTION	DBCONNECTION	1
3	9	THROUGHPUT_STREAM1_DB_CONNECTION	DBCONNECTION	1
3	8	POWER_DB_CONNECTION	DBCONNECTION	1
3	5	DYNAMIC_MASTERCONNECTION	MASTERCONNECTION	2
3	3	DYNAMIC_DBCONNECTION	DBCONNECTION	2

Workspace_connections

workspace_i	3
connection_i	7
connection_name	MASTERCONNECTION
connection_valu	DRIVER=SQL Server;SERVER=;UID=sa;PWD=;
descriptio	
no_count_displa	0
no_execute	0
parse_query_onl	0
ANSI_quoted_identifier	0
ANSI_nulls	-1
show_query_pla	0
show_stats_tim	0
show_stats_i	0
parse_odbc_msg_prefix	-1
row_count	0
tsql_batch_separat	GO
query_time_out	0
server_languag	(Default)
character_translatio	-1
regional_setting	0

workspace_i	3
connection_i	3
connection_name	DBCONNECTION
connection_valu	DRIVER=SQL Server;SERVER=;UID=sa;PWD=;DATABASE=%DBNAME%;
descriptio	
no_count_displa	0
no_execute	0
parse_query_onl	0
ANSI_quoted_identifier	0
ANSI_nulls	-1
show_query_pla	0
show_stats_tim	0
show_stats_i	0
parse_odbc_msg_prefix	-1
row_count	0
tsql_batch_separat	GO
query_time_out	0
server_languag	(Default)
character_translatio	-1
regional_setting	0

Att_steps

workspace_id = 3

step_label = Syntax Check Parameters ***step_id =*** 73 ***global_flag =*** 0

sequence_no = 2 ***step_level =*** 0 ***parent_step_id =*** 0 ***enabled_flag =*** 0

iterator_name = ***degree_parallelism*** 1

execution_mechanism = 0 ***continuation_criteria =*** 0 ***failure_details =***

step_file_name = ***version_no =*** 24.0

start_directory = ***parent_version_no =*** 0.0

step_text =

step_label = Execute Power Run ***step_id =*** 76 ***global_flag =*** 0

sequence_no = 3 ***step_level =*** 0 ***parent_step_id =*** 0 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** 1

execution_mechanism = 0 ***continuation_criteria =*** 0 ***failure_details =***

step_file_name = ***version_no =*** 56.1

start_directory = ***parent_version_no =*** 0.0

step_text =

step_label = Execute Throughput Run ***step_id =*** 77 ***global_flag =*** 0

sequence_no = 4 ***step_level =*** 0 ***parent_step_id =*** 0 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** 2

execution_mechanism = 0 ***continuation_criteria =*** 0 ***failure_details =***

step_file_name = ***version_no =*** 79.0

start_directory = ***parent_version_no =*** 0.0

step_text =

step_label = Power - Execute Query 14 ***step_id =*** 80 ***global_flag =*** 0

sequence_no = 1 ***step_level =*** 2 ***parent_step_id =*** 180 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** 1

execution_mechanism = 1 ***continuation_criteria =*** 1 ***failure_details =***

step_file_name = %QUERY_DIR%\Power\14.sql ***version_no =*** 60.0

start_directory = POWER_DB_CONNECTION ***parent_version_no =*** 28.2

step_text =

step_label = Power - Execute Query 02 ***step_id =*** 81 ***global_flag =*** 0

sequence_no = 2 ***step_level =*** 2 ***parent_step_id =*** 180 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** 1

execution_mechanism = 1 ***continuation_criteria =*** 1 ***failure_details =***

step_file_name = %QUERY_DIR%\Power\2.sql ***version_no =*** 55.0

start_directory = POWER_DB_CONNECTION ***parent_version_no =*** 28.2

step_text =

step_label = Power - Execute Query 09 ***step_id =*** 82 ***global_flag =*** 0
sequence_no = 3 ***step_level =*** 2 ***parent_step_id =*** 180 ***enabled_flag =*** -1
iterator_name = ***degree_parallelism*** 1
execution_mechanism = 1 ***continuation_criteria =*** 1 ***failure_details =***
step_file_name = %QUERY_DIR%\Power\9.sql ***version_no =*** 52.0
start_directory = POWER_DB_CONNECTION ***parent_version_no =*** 28.2
step_text =

step_label = Power - Execute Query 20 ***step_id =*** 83 ***global_flag =*** 0
sequence_no = 4 ***step_level =*** 2 ***parent_step_id =*** 180 ***enabled_flag =*** -1
iterator_name = ***degree_parallelism*** 1
execution_mechanism = 1 ***continuation_criteria =*** 1 ***failure_details =***
step_file_name = %QUERY_DIR%\Power\20.sql ***version_no =*** 53.0
start_directory = POWER_DB_CONNECTION ***parent_version_no =*** 28.2
step_text =

step_label = Power - Execute Query 06 ***step_id =*** 84 ***global_flag =*** 0
sequence_no = 5 ***step_level =*** 2 ***parent_step_id =*** 180 ***enabled_flag =*** -1
iterator_name = ***degree_parallelism*** 1
execution_mechanism = 1 ***continuation_criteria =*** 1 ***failure_details =***
step_file_name = %QUERY_DIR%\Power\6.sql ***version_no =*** 48.0
start_directory = POWER_DB_CONNECTION ***parent_version_no =*** 28.2
step_text =

step_label = Power - Execute Query 17 ***step_id =*** 85 ***global_flag =*** 0
sequence_no = 6 ***step_level =*** 2 ***parent_step_id =*** 180 ***enabled_flag =*** -1
iterator_name = ***degree_parallelism*** 1
execution_mechanism = 1 ***continuation_criteria =*** 1 ***failure_details =***
step_file_name = %QUERY_DIR%\Power\17.sql ***version_no =*** 51.0
start_directory = POWER_DB_CONNECTION ***parent_version_no =*** 28.2
step_text =

step_label = Power - Execute Query 18 ***step_id =*** 86 ***global_flag =*** 0
sequence_no = 7 ***step_level =*** 2 ***parent_step_id =*** 180 ***enabled_flag =*** -1
iterator_name = ***degree_parallelism*** 1
execution_mechanism = 1 ***continuation_criteria =*** 1 ***failure_details =***
step_file_name = %QUERY_DIR%\Power\18.sql ***version_no =*** 48.0
start_directory = POWER_DB_CONNECTION ***parent_version_no =*** 28.2
step_text =

step_label = Power - Execute Query 08 ***step_id =*** 87 ***global_flag =*** 0
sequence_no = 8 ***step_level =*** 2 ***parent_step_id =*** 180 ***enabled_flag =*** -1
iterator_name = ***degree_parallelism*** 1
execution_mechanism = 1 ***continuation_criteria =*** 1 ***failure_details =***
step_file_name = %QUERY_DIR%\Power\8.sql ***version_no =*** 48.0
start_directory = POWER_DB_CONNECTION ***parent_version_no =*** 28.2
step_text =

step_label = Power - Execute Query 21 ***step_id =*** 88 ***global_flag =*** 0
sequence_no = 9 ***step_level =*** 2 ***parent_step_id =*** 180 ***enabled_flag =*** -1
iterator_name = ***degree_parallelism*** 1
execution_mechanism = 1 ***continuation_criteria =*** 1 ***failure_details =***
step_file_name = %QUERY_DIR%\Power\21.sql ***version_no =*** 48.0
start_directory = POWER_DB_CONNECTION ***parent_version_no =*** 28.2
step_text =

step_label = Power - Execute Query 13 ***step_id =*** 89 ***global_flag =*** 0
sequence_no = 10 ***step_level =*** 2 ***parent_step_id =*** 180 ***enabled_flag =*** -1
iterator_name = ***degree_parallelism*** 1
execution_mechanism = 1 ***continuation_criteria =*** 1 ***failure_details =***
step_file_name = %QUERY_DIR%\Power\13.sql ***version_no =*** 44.0
start_directory = POWER_DB_CONNECTION ***parent_version_no =*** 28.2
step_text =

step_label = Power - Execute Query 03 ***step_id =*** 90 ***global_flag =*** 0
sequence_no = 11 ***step_level =*** 2 ***parent_step_id =*** 180 ***enabled_flag =*** -1
iterator_name = ***degree_parallelism*** 1
execution_mechanism = 1 ***continuation_criteria =*** 1 ***failure_details =***
step_file_name = %QUERY_DIR%\Power\3.sql ***version_no =*** 45.0
start_directory = POWER_DB_CONNECTION ***parent_version_no =*** 28.2
step_text =

step_label = Power - Execute Query 22 ***step_id =*** 91 ***global_flag =*** 0
sequence_no = 12 ***step_level =*** 2 ***parent_step_id =*** 180 ***enabled_flag =*** -1
iterator_name = ***degree_parallelism*** 1
execution_mechanism = 1 ***continuation_criteria =*** 1 ***failure_details =***
step_file_name = %QUERY_DIR%\Power\22.sql ***version_no =*** 42.0
start_directory = POWER_DB_CONNECTION ***parent_version_no =*** 28.2
step_text =

step_label = Power - Execute Query 16 ***step_id =*** 92 ***global_flag =*** 0
sequence_no = 13 ***step_level =*** 2 ***parent_step_id =*** 180 ***enabled_flag =*** -1
iterator_name = ***degree_parallelism*** 1
execution_mechanism = 1 ***continuation_criteria =*** 1 ***failure_details =***
step_file_name = %QUERY_DIR%\Power\16.sql ***version_no =*** 41.0
start_directory = POWER_DB_CONNECTION ***parent_version_no =*** 28.2
step_text =

step_label = Power - Execute Query 04 ***step_id =*** 93 ***global_flag =*** 0
sequence_no = 14 ***step_level =*** 2 ***parent_step_id =*** 180 ***enabled_flag =*** -1
iterator_name = ***degree_parallelism*** 1
execution_mechanism = 1 ***continuation_criteria =*** 1 ***failure_details =***
step_file_name = %QUERY_DIR%\Power\4.sql ***version_no =*** 41.0
start_directory = POWER_DB_CONNECTION ***parent_version_no =*** 28.2
step_text =

step_label = Power - Execute Query 11 ***step_id =*** 94 ***global_flag =*** 0

sequence_no = 15 ***step_level =*** 2 ***parent_step_id =*** 180 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** 1

execution_mechanism = 1 ***continuation_criteria =*** 1 ***failure_details =***

step_file_name = %QUERY_DIR%\Power\11.sql ***version_no =*** 42.0

start_directory = POWER_DB_CONNECTION ***parent_version_no =*** 28.2

step_text =

step_label = Power - Execute Query 01 ***step_id =*** 96 ***global_flag =*** 0

sequence_no = 17 ***step_level =*** 2 ***parent_step_id =*** 180 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** 1

execution_mechanism = 1 ***continuation_criteria =*** 1 ***failure_details =***

step_file_name = %QUERY_DIR%\Power\1.sql ***version_no =*** 34.0

start_directory = POWER_DB_CONNECTION ***parent_version_no =*** 28.2

step_text =

step_label = Power - Execute Query 10 ***step_id =*** 97 ***global_flag =*** 0

sequence_no = 18 ***step_level =*** 2 ***parent_step_id =*** 180 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** 1

execution_mechanism = 1 ***continuation_criteria =*** 1 ***failure_details =***

step_file_name = %QUERY_DIR%\Power\10.sql ***version_no =*** 35.0

start_directory = POWER_DB_CONNECTION ***parent_version_no =*** 28.2

step_text =

step_label = Power - Execute Query 19 ***step_id =*** 98 ***global_flag =*** 0

sequence_no = 19 ***step_level =*** 2 ***parent_step_id =*** 180 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** 1

execution_mechanism = 1 ***continuation_criteria =*** 1 ***failure_details =***

step_file_name = %QUERY_DIR%\Power\19.sql ***version_no =*** 32.0

start_directory = POWER_DB_CONNECTION ***parent_version_no =*** 28.2

step_text =

step_label = Power - Execute Query 05 ***step_id =*** 99 ***global_flag =*** 0
sequence_no = 20 ***step_level =*** 2 ***parent_step_id =*** 180 ***enabled_flag =*** -1
iterator_name = ***degree_parallelism*** 1
execution_mechanism = 1 ***continuation_criteria =*** 1 ***failure_details =***
step_file_name = %QUERY_DIR%\Power\5.sql ***version_no =*** 34.0
start_directory = POWER_DB_CONNECTION ***parent_version_no =*** 28.2
step_text =

step_label = Power - Execute Query 07 ***step_id =*** 100 ***global_flag =*** 0
sequence_no = 21 ***step_level =*** 2 ***parent_step_id =*** 180 ***enabled_flag =*** -1
iterator_name = ***degree_parallelism*** 1
execution_mechanism = 1 ***continuation_criteria =*** 1 ***failure_details =***
step_file_name = %QUERY_DIR%\Power\7.sql ***version_no =*** 31.0
start_directory = POWER_DB_CONNECTION ***parent_version_no =*** 28.2
step_text =

step_label = Power - Execute Query 12 ***step_id =*** 101 ***global_flag =*** 0
sequence_no = 22 ***step_level =*** 2 ***parent_step_id =*** 180 ***enabled_flag =*** -1
iterator_name = ***degree_parallelism*** 1
execution_mechanism = 1 ***continuation_criteria =*** 1 ***failure_details =***
step_file_name = %QUERY_DIR%\Power\12.sql ***version_no =*** 24.0
start_directory = POWER_DB_CONNECTION ***parent_version_no =*** 28.2
step_text =

step_label = Power - Parallel RF2 Execution ***step_id =*** 102 ***global_flag =*** 0
sequence_no = 3 ***step_level =*** 1 ***parent_step_id =*** 76 ***enabled_flag =*** -1
iterator_name = ***degree_parallelism*** %DELETE_PARALLELISM%
execution_mechanism = 0 ***continuation_criteria =*** 0 ***failure_details =***
step_file_name = ***version_no =*** 41.0
start_directory = ***parent_version_no =*** 56.1
step_text =

step_label = Syntax Check Directories ***step_id =*** 119 ***global_flag =*** 0

sequence_no = 1 ***step_level =*** 1 ***parent_step_id =*** 73 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** 1

execution_mechanism = 2 ***continuation_criteria =*** 1 ***failure_details =*** %RUN_DI

step_file_name = ***version_no =*** 6.0

start_directory = C:\ ***parent_version_no =*** 24.0

step_text =

```
echo off

if "%RUN_DIR%"== "" goto :eof
if "%RF_FLATFILE_DIR%" == "" goto :eof
if "%OUTPUT_DIR%"=="" goto :eof
if "%QUERY_DIR%"=="" goto :eof
if "%TOOLS_DIR%"=="" goto :eof

if exist %RUN_DIR% echo RUN_DIR found
if exist %RUN_DIR%\%DBNAME% echo RUN_DIR\DBNAME found
if exist %TOOLS_DIR% echo TOOLS_DIR found
if exist %TOOLS_DIR%\QGen echo TOOLS_DIR\QGen found
if exist %TOOLS_DIR%\Utility echo TOOLS_DIR\Utility found
if exist %RF_FLATFILE_DIR% echo RF_FLATFILE_DIR found
if exist %OUTPUT_DIR% echo OUTPUT_DIR found
if exist %QUERY_DIR% echo QUERY_DIR found
```

step_label = Power - Parallel RF1 Execution ***step_id =*** 178 ***global_flag =*** 0

sequence_no = 1 ***step_level =*** 1 ***parent_step_id =*** 76 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** %INSERT_PARALLELISM%

execution_mechanism = 0 ***continuation_criteria =*** 0 ***failure_details =***

step_file_name = ***version_no =*** 55.0

start_directory = ***parent_version_no =*** 56.1

step_text =

step_label = Power - Execute RF1 ***step_id =*** 179 ***global_flag =*** 0

sequence_no = 1 ***step_level =*** 2 ***parent_step_id =*** 178 ***enabled_flag =*** -1

iterator_name = INSERT_SEGMENT ***degree_parallelism*** 1

execution_mechanism = 1 ***continuation_criteria =*** 1 ***failure_details =***

step_file_name = ***version_no =*** 21.0

start_directory = DYNAMIC_RF_DBCONNECTION ***parent_version_no =*** 55.0

step_text =

```
DECLARE @SQLstring NVARCHAR(255)
DECLARE @updateset INTEGER

--
-- Get the current update set value
--
SELECT @updateset=updateset from TPCH_AUX_TABLE

--
--  Delete any previous columns from the insert table
--
TRUNCATE TABLE NEWORDERS_%INSERT_SEGMENT%
TRUNCATE TABLE NEWLINEITEM_%INSERT_SEGMENT%

-- DECLARE @timefrom datetime
-- SELECT    @timefrom=getdate()

--
--  Generate an SQL statement inserting the current updateset value into
--   the command.  Next execute the statement to bulk load the new lineitem
--  insert values.
--
SET @SQLstring='bulk insert %DBNAME%..NEWLINEITEM_%INSERT_SEGMENT% from
''%RF_FLATFILE_DIR%\Lineitem.tbl.u' + RTRIM(Convert(char,@updateset)) +
'.%INSERT_SEGMENT%'' with (FieldTerminator = ''|'', RowTerminator =''|\n'',tablock)'
PRINT @SQLstring
EXEC sp_executesql @SQLstring

--
--  Generate an SQL statement inserting the current updateset value into
--  the command.  Next execute the statement to bulk load the new order
--  insert values.
--
SET @SQLstring='bulk insert %DBNAME%..NEWORDERS_%INSERT_SEGMENT% from
''%RF_FLATFILE_DIR%\Orders.tbl.u' + RTRIM(Convert(char,@updateset)) +
'.%INSERT_SEGMENT%'' with (FieldTerminator = ''|'', RowTerminator =''|\n'',tablock)'
PRINT @SQLstring
EXEC sp_executesql @SQLstring

exec  RF1_%INSERT_SEGMENT% %BATCH_SIZE%
```

step_label = Power - Sequential Query Execution ***step_id =*** 180 ***global_flag =*** 0

sequence_no = 2 ***step_level =*** 1 ***parent_step_id =*** 76 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** 1

execution_mechanism = 0 ***continuation_criteria =*** 0 ***failure_details =***

step_file_name = ***version_no =*** 28.2

start_directory = ***parent_version_no =*** 56.1

step_text =

step_label =	Power - Execute RF2			***step_id =***	181	***global_flag =***	0
sequence_no =	1	***step_level =***	2	***parent_step_id =***	102	***enabled_flag =***	-1
iterator_name =	DELETE_SEGMENT			***degree_parallelism***	1		
execution_mechanism =	1			***continuation_criteria =***	1	***failure_details =***	
step_file_name =						***version_no =***	15.0
start_directory =	DYNAMIC_RF_DBCONNECTION					***parent_version_no =***	41.0

step_text =

```
DECLARE @SQLstring NVARCHAR(255)
DECLARE @updateset INTEGER

--
-- Get the current update set value
--
SELECT @updateset=updateset from TPCH_AUX_TABLE

--
--  Delete any existing index(s) on the temporary table(s)
--
if exists (select name from sysindexes where name = 'OLDORDERS_%DELETE_SEGMENT%_idx')
        drop index OLDORDERS_%DELETE_SEGMENT%.OLDORDERS_%DELETE_SEGMENT%_idx

--
--  Delete any previous columns from the delete table
--
TRUNCATE TABLE OLDORDERS_%DELETE_SEGMENT%

--
--  Generate an SQL statement inserting the current updateset value into
--  the command.  Next execute the statement to bulk load the old order
--  delete values
--
SET @SQLstring='bulk insert %DBNAME%..OLDORDERS_%DELETE_SEGMENT% from
''%RF_FLATFILE_DIR%\Delete.u'  + RTRIM(Convert(char,@updateset)) +  '.%DELETE_SEGMENT%''
with (FieldTerminator = ''|'', RowTerminator =''\n'',tablock)'
EXEC sp_executesql @SQLstring

--
--  Create index on OLDORDERS
--
SET @SQLstring='create unique index OLDORDERS_%DELETE_SEGMENT%_idx on
OLDORDERS_%DELETE_SEGMENT% (O_ORDERKEY)'
EXEC sp_executesql @SQLstring

exec RF2_%DELETE_SEGMENT% %BATCH_SIZE%
```

step_label = Power - Increment Update Set ***step_id =*** 182 ***global_flag =*** 0
sequence_no = 4 ***step_level =*** 1 ***parent_step_id =*** 76 ***enabled_flag =*** -1
iterator_name = ***degree_parallelism*** 1
execution_mechanism = 1 ***continuation_criteria =*** 2 ***failure_details =***
step_file_name = ***version_no =*** 42.0
start_directory = DYNAMIC_DBCONNECTION ***parent_version_no =*** 56.1
step_text = UPDATE TPCH_AUX_TABLE SET updateset=updateset+1

step_label = Parallel Stream Execution ***step_id =*** 185 ***global_flag =*** 0
sequence_no = 2 ***step_level =*** 1 ***parent_step_id =*** 77 ***enabled_flag =*** -1
iterator_name = ***degree_parallelism*** %MAX_STREAMS%
execution_mechanism = 0 ***continuation_criteria =*** 0 ***failure_details =***
step_file_name = ***version_no =*** 22.0
start_directory = ***parent_version_no =*** 79.0
step_text =

step_label = Sequential Refresh Stream Execution ***step_id =*** 188 ***global_flag =*** 0
sequence_no = 1 ***step_level =*** 1 ***parent_step_id =*** 77 ***enabled_flag =*** -1
iterator_name = ***degree_parallelism*** 1
execution_mechanism = 0 ***continuation_criteria =*** 0 ***failure_details =***
step_file_name = ***version_no =*** 29.0
start_directory = ***parent_version_no =*** 79.0
step_text =

step_label = Throughput - Refresh Stream 1 ***step_id =*** 189 ***global_flag =*** 0
sequence_no = 2 ***step_level =*** 2 ***parent_step_id =*** 188 ***enabled_flag =*** -1
iterator_name = STREAM_NUM ***degree_parallelism*** 1
execution_mechanism = 0 ***continuation_criteria =*** 0 ***failure_details =***
step_file_name = ***version_no =*** 21.1
start_directory = ***parent_version_no =*** 29.0
step_text =

step_label = Throughput - Refresh Stream 2 - n ***step_id =*** 191 ***global_flag =*** 0

sequence_no = 3 ***step_level =*** 2 ***parent_step_id =*** 188 ***enabled_flag =*** -1

iterator_name = STREAM_NUM ***degree_parallelism*** 1

execution_mechanism = 0 ***continuation_criteria =*** 0 ***failure_details =***

step_file_name = ***version_no =*** 10.3

start_directory = ***parent_version_no =*** 29.0

step_text =

step_label = Throughput - Stream n - Increment Update Set ***step_id =*** 193 ***global_flag =*** 0

sequence_no = 3 ***step_level =*** 3 ***parent_step_id =*** 191 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** 1

execution_mechanism = 1 ***continuation_criteria =*** 2 ***failure_details =***

step_file_name = ***version_no =*** 12.0

start_directory = DYNAMIC_RF_DBCONNECTION ***parent_version_no =*** 10.3

step_text =

```
select updateset from TPCH_AUX_TABLE
GO

UPDATE TPCH_AUX_TABLE SET updateset=updateset+1

select updateset from TPCH_AUX_TABLE
GO
```

step_label = Throughput - Query Stream 1 ***step_id =*** 225 ***global_flag =*** 0

sequence_no = 1 ***step_level =*** 3 ***parent_step_id =*** 238 ***enabled_flag =*** -1

iterator_name = QUERY ***degree_parallelism*** 1

execution_mechanism = 1 ***continuation_criteria =*** 2 ***failure_details =***

step_file_name = %QUERY_DIR%\STREAM1\Stream1Q%QUERY%.sql ***version_no =*** 24.0

start_directory = THROUGHPUT_STREAM1_DB_CONNECTION ***parent_version_no =*** 2.7

step_text =

step_label = Throughput - Query Stream 2 ***step_id =*** 226 ***global_flag =*** 0
sequence_no = 1 ***step_level =*** 3 ***parent_step_id =*** 239 ***enabled_flag =*** -1
iterator_name = QUERY ***degree_parallelism*** 1
execution_mechanism = 1 ***continuation_criteria =*** 2 ***failure_details =***
step_file_name = %QUERY_DIR%\STREAM2\Stream2Q%QUERY%.sql ***version_no =*** 22.0
start_directory = THROUGHPUT_STREAM2_DB_CONNECTION ***parent_version_no =*** 1.7
step_text =

step_label = Throughput - Query Stream 3 ***step_id =*** 227 ***global_flag =*** 0
sequence_no = 1 ***step_level =*** 3 ***parent_step_id =*** 240 ***enabled_flag =*** -1
iterator_name = QUERY ***degree_parallelism*** 1
execution_mechanism = 1 ***continuation_criteria =*** 2 ***failure_details =***
step_file_name = %QUERY_DIR%\STREAM3\Stream3Q%QUERY%.sql ***version_no =*** 8.0
start_directory = THROUGHPUT_STREAM3_DB_CONNECTION ***parent_version_no =*** 2.7
step_text =

step_label = Throughput - Query Stream 4 ***step_id =*** 228 ***global_flag =*** 0
sequence_no = 1 ***step_level =*** 3 ***parent_step_id =*** 241 ***enabled_flag =*** -1
iterator_name = QUERY ***degree_parallelism*** 1
execution_mechanism = 1 ***continuation_criteria =*** 2 ***failure_details =***
step_file_name = %QUERY_DIR%\STREAM4\Stream4Q%QUERY%.sql ***version_no =*** 8.0
start_directory = THROUGHPUT_STREAM4_DB_CONNECTION ***parent_version_no =*** 2.7
step_text =

step_label = Stream 1 Manager ***step_id =*** 238 ***global_flag =*** 0
sequence_no = 1 ***step_level =*** 2 ***parent_step_id =*** 185 ***enabled_flag =*** -1
iterator_name = ***degree_parallelism*** 1
execution_mechanism = 0 ***continuation_criteria =*** 0 ***failure_details =***
step_file_name = ***version_no =*** 2.7
start_directory = ***parent_version_no =*** 22.0
step_text =

step_label = Stream 2 Manager ***step_id =*** 239 ***global_flag =*** 0
sequence_no = 2 ***step_level =*** 2 ***parent_step_id =*** 185 ***enabled_flag =*** -1
iterator_name = ***degree_parallelism*** 1
execution_mechanism = 0 ***continuation_criteria =*** 0 ***failure_details =***
step_file_name = ***version_no =*** 1.7
start_directory = ***parent_version_no =*** 22.0
step_text =

step_label = Stream 3 Manager ***step_id =*** 240 ***global_flag =*** 0
sequence_no = 3 ***step_level =*** 2 ***parent_step_id =*** 185 ***enabled_flag =*** -1
iterator_name = ***degree_parallelism*** 1
execution_mechanism = 0 ***continuation_criteria =*** 0 ***failure_details =***
step_file_name = ***version_no =*** 2.7
start_directory = ***parent_version_no =*** 22.0
step_text =

step_label = Stream 4 Manager ***step_id =*** 241 ***global_flag =*** 0
sequence_no = 4 ***step_level =*** 2 ***parent_step_id =*** 185 ***enabled_flag =*** -1
iterator_name = ***degree_parallelism*** 1
execution_mechanism = 0 ***continuation_criteria =*** 0 ***failure_details =***
step_file_name = ***version_no =*** 2.7
start_directory = ***parent_version_no =*** 22.0
step_text =

step_label = Stream 5 Manager ***step_id =*** 242 ***global_flag =*** 0
sequence_no = 5 ***step_level =*** 2 ***parent_step_id =*** 185 ***enabled_flag =*** -1
iterator_name = ***degree_parallelism*** 1
execution_mechanism = 0 ***continuation_criteria =*** 0 ***failure_details =***
step_file_name = ***version_no =*** 3.7
start_directory = ***parent_version_no =*** 22.0
step_text =

```
step_label =   Throughput - Query Stream 5                         step_id =        243      global_flag =      0
sequence_no =            1  step_level =          3        parent_step_id =         242  enabled_flag =        -1
iterator_name =       QUERY                                 degree_parallelism    1
execution_mechanism =    1                           continuation_criteria =    2            failure_details =
step_file_name =      %QUERY_DIR%\STREAM5\Stream5Q%QUERY%.sql                                     version_no =   8.0
start_directory =     THROUGHPUT_STREAM5_DB_CONNECTION                                 parent_version_no =       3.7
step_text =

step_label =   Stream 6 Manager                                    step_id =        244      global_flag =      0
sequence_no =            6  step_level =          2        parent_step_id =         185  enabled_flag =        -1
iterator_name =                                             degree_parallelism    1
execution_mechanism =    0                           continuation_criteria =    0            failure_details =
step_file_name =                                                                                  version_no =   2.0
start_directory =                                                                      parent_version_no =       22.0
step_text =

step_label =   Throughput - Query Stream 6                         step_id =        245      global_flag =      0
sequence_no =            1  step_level =          3        parent_step_id =         244  enabled_flag =        -1
iterator_name =       QUERY                                 degree_parallelism    1
execution_mechanism =    1                           continuation_criteria =    2            failure_details =
step_file_name =      %QUERY_DIR%\STREAM6\Stream6Q%QUERY%.sql                                     version_no =   1.0
start_directory =     THROUGHPUT_STREAM6_DB_CONNECTION                                 parent_version_no =       2.0
step_text =

step_label =   Stream 7 Manager                                    step_id =        246      global_flag =      0
sequence_no =            7  step_level =          2        parent_step_id =         185  enabled_flag =         0
iterator_name =                                             degree_parallelism    1
execution_mechanism =    0                           continuation_criteria =    0            failure_details =
step_file_name =                                                                                  version_no =   2.0
start_directory =                                                                      parent_version_no =       22.0
step_text =
```

step_label = Throughput - Query Stream 7 ***step_id =*** 247 ***global_flag =*** 0
sequence_no = 1 ***step_level =*** 3 ***parent_step_id =*** 246 ***enabled_flag =*** -1
iterator_name = QUERY ***degree_parallelism*** 1
execution_mechanism = 1 ***continuation_criteria =*** 2 ***failure_details =***
step_file_name = %QUERY_DIR%\STREAM7\Stream7Q%QUERY%.sql ***version_no =*** 0.0
start_directory = THROUGHPUT_STREAM7_DB_CONNECTION ***parent_version_no =*** 2.0
step_text =

step_label = Stream 8 Manager ***step_id =*** 248 ***global_flag =*** 0
sequence_no = 8 ***step_level =*** 2 ***parent_step_id =*** 185 ***enabled_flag =*** 0
iterator_name = ***degree_parallelism*** 1
execution_mechanism = 0 ***continuation_criteria =*** 0 ***failure_details =***
step_file_name = ***version_no =*** 0.0
start_directory = ***parent_version_no =*** 22.0
step_text =

step_label = Throughput - Query Stream 8 ***step_id =*** 249 ***global_flag =*** 0
sequence_no = 1 ***step_level =*** 3 ***parent_step_id =*** 248 ***enabled_flag =*** -1
iterator_name = QUERY ***degree_parallelism*** 1
execution_mechanism = 1 ***continuation_criteria =*** 2 ***failure_details =***
step_file_name = %QUERY_DIR%\STREAM8\Stream8Q%QUERY%.sql ***version_no =*** 0.0
start_directory = THROUGHPUT_STREAM8_DB_CONNECTION ***parent_version_no =*** 0.0
step_text =

step_label = Stream 9 Manager ***step_id =*** 250 ***global_flag =*** 0
sequence_no = 9 ***step_level =*** 2 ***parent_step_id =*** 185 ***enabled_flag =*** 0
iterator_name = ***degree_parallelism*** 1
execution_mechanism = 0 ***continuation_criteria =*** 0 ***failure_details =***
step_file_name = ***version_no =*** 0.0
start_directory = ***parent_version_no =*** 22.0
step_text =

step_label = Throughput - Query Stream 9 ***step_id =*** 251 ***global_flag =*** 0
sequence_no = 1 ***step_level =*** 3 ***parent_step_id =*** 250 ***enabled_flag =*** -1
iterator_name = QUERY ***degree_parallelism*** 1
execution_mechanism = 1 ***continuation_criteria =*** 2 ***failure_details =***
step_file_name = %QUERY_DIR%\STREAM9\Stream9Q%QUERY%.sql ***version_no =*** 0.0
start_directory = THROUGHPUT_STREAM9_DB_CONNECTION ***parent_version_no =*** 0.0
step_text =

step_label = Stream 10 Manager ***step_id =*** 252 ***global_flag =*** 0
sequence_no = 10 ***step_level =*** 2 ***parent_step_id =*** 185 ***enabled_flag =*** 0
iterator_name = ***degree_parallelism*** 1
execution_mechanism = 0 ***continuation_criteria =*** 0 ***failure_details =***
step_file_name = ***version_no =*** 0.0
start_directory = ***parent_version_no =*** 22.0
step_text =

step_label = Throughput - Query Stream 10 ***step_id =*** 253 ***global_flag =*** 0
sequence_no = 1 ***step_level =*** 3 ***parent_step_id =*** 252 ***enabled_flag =*** -1
iterator_name = QUERY ***degree_parallelism*** 1
execution_mechanism = 1 ***continuation_criteria =*** 2 ***failure_details =***
step_file_name = %QUERY_DIR%\STREAM10\Stream10Q%QUERY%.sql ***version_no =*** 0.0
start_directory = THROUGHPUT_STREAM10_DB_CONNECTION ***parent_version_no =*** 0.0
step_text =

step_label = Power - Execute Query 15 ***step_id =*** 329 ***global_flag =*** 0
sequence_no = 16 ***step_level =*** 2 ***parent_step_id =*** 180 ***enabled_flag =*** -1
iterator_name = ***degree_parallelism*** 1
execution_mechanism = 1 ***continuation_criteria =*** 1 ***failure_details =***
step_file_name = %QUERY_DIR%\Power\15.sql ***version_no =*** 40.0
start_directory = POWER_DB_CONNECTION ***parent_version_no =*** 28.2
step_text =

step_label = Throughput - Post to Semaphore (S1) ***step_id =*** 334 ***global_flag =*** 0

sequence_no = 2 ***step_level =*** 3 ***parent_step_id =*** 238 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** 1

execution_mechanism = 2 ***continuation_criteria =*** 2 ***failure_details =***

step_file_name = ***version_no =*** 7.0

start_directory = %TOOLS_DIR% ***parent_version_no =*** 2.7

step_text = %TOOLS_DIR%\Utility\semaphore -signal S.1

step_label = Throughput - Post to Semaphore (S2) ***step_id =*** 335 ***global_flag =*** 0

sequence_no = 2 ***step_level =*** 3 ***parent_step_id =*** 239 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** 1

execution_mechanism = 2 ***continuation_criteria =*** 2 ***failure_details =***

step_file_name = ***version_no =*** 4.0

start_directory = %TOOLS_DIR% ***parent_version_no =*** 1.7

step_text = %TOOLS_DIR%\Utility\semaphore -signal S.2

step_label = Throughput - Post to Semaphore (S3) ***step_id =*** 338 ***global_flag =*** 0

sequence_no = 2 ***step_level =*** 3 ***parent_step_id =*** 240 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** 1

execution_mechanism = 2 ***continuation_criteria =*** 2 ***failure_details =***

step_file_name = ***version_no =*** 3.0

start_directory = %TOOLS_DIR% ***parent_version_no =*** 2.7

step_text = %TOOLS_DIR%\Utility\semaphore -signal S.3

step_label = Throughput - Post to Semaphore (S4) ***step_id =*** 339 ***global_flag =*** 0

sequence_no = 2 ***step_level =*** 3 ***parent_step_id =*** 241 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** 1

execution_mechanism = 2 ***continuation_criteria =*** 2 ***failure_details =***

step_file_name = ***version_no =*** 3.0

start_directory = %TOOLS_DIR% ***parent_version_no =*** 2.7

step_text =

```
%TOOLS_DIR%\Utility\semaphore -signal S.4
```

step_label = Clear any Outstanding Semaphores ***step_id =*** 340 ***global_flag =*** 0

sequence_no = 1 ***step_level =*** 0 ***parent_step_id =*** 0 ***enabled_flag =*** 0

iterator_name = ***degree_parallelism*** 1

execution_mechanism = 2 ***continuation_criteria =*** 2 ***failure_details =***

step_file_name = ***version_no =*** 33.0

start_directory = %TOOLS_DIR% ***parent_version_no =*** 0.0

step_text =

```
::
::  This step must always be run to insure that a semaphore.exe was not left open by a
::  previous run
::
::  If there are no open semaphore.exe's then the 'KILL' will do nothing.
::
%TOOLS_DIR%\Utility\KILL.EXE SEMAPHORE.EXE
```

step_label = Throughput - Post to Semaphore (S5) ***step_id =*** 351 ***global_flag =*** 0

sequence_no = 2 ***step_level =*** 3 ***parent_step_id =*** 242 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** 1

execution_mechanism = 2 ***continuation_criteria =*** 2 ***failure_details =***

step_file_name = ***version_no =*** 3.0

start_directory = %TOOLS_DIR% ***parent_version_no =*** 3.7

step_text =

```
%TOOLS_DIR%\Utility\semaphore -signal S.5
```

step_label = Throughput - Post to Semaphore (S6) ***step_id =*** 352 ***global_flag =*** 0

sequence_no = 2 ***step_level =*** 3 ***parent_step_id =*** 244 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** 1

execution_mechanism = 2 ***continuation_criteria =*** 2 ***failure_details =***

step_file_name = ***version_no =*** 3.0

start_directory = %TOOLS_DIR% ***parent_version_no =*** 2.0

step_text = %TOOLS_DIR%\Utility\semaphore -signal S.6

step_label = Throughput - Post to Semaphore (S7) ***step_id =*** 353 ***global_flag =*** 0

sequence_no = 2 ***step_level =*** 3 ***parent_step_id =*** 246 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** 1

execution_mechanism = 2 ***continuation_criteria =*** 2 ***failure_details =***

step_file_name = ***version_no =*** 3.0

start_directory = %TOOLS_DIR% ***parent_version_no =*** 2.0

step_text = %TOOLS_DIR%\Utility\semaphore -signal S.7

step_label = Throughput - Post to Semaphore (S8) ***step_id =*** 354 ***global_flag =*** 0

sequence_no = 2 ***step_level =*** 3 ***parent_step_id =*** 248 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** 1

execution_mechanism = 2 ***continuation_criteria =*** 2 ***failure_details =***

step_file_name = ***version_no =*** 3.0

start_directory = %TOOLS_DIR% ***parent_version_no =*** 0.0

step_text = %TOOLS_DIR%\Utility\semaphore -signal S.8

step_label = Throughput - Post to Semaphore (S9) ***step_id =*** 355 ***global_flag =*** 0

sequence_no = 2 ***step_level =*** 3 ***parent_step_id =*** 250 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** 1

execution_mechanism = 2 ***continuation_criteria =*** 2 ***failure_details =***

step_file_name = ***version_no =*** 3.0

start_directory = %TOOLS_DIR% ***parent_version_no =*** 0.0

step_text = %TOOLS_DIR%\Utility\semaphore -signal S.9

step_label = Throughput - Post to Semaphore (S10) ***step_id =*** 356 ***global_flag =*** 0

sequence_no = 2 ***step_level =*** 3 ***parent_step_id =*** 252 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** 1

execution_mechanism = 2 ***continuation_criteria =*** 2 ***failure_details =***

step_file_name = ***version_no =*** 3.0

start_directory = %TOOLS_DIR% ***parent_version_no =*** 0.0

step_text = %TOOLS_DIR%\Utility\semaphore -signal S.10

step_label = Throughput - Semaphore Loop for RF Delay ***step_id =*** 430 ***global_flag =*** 0

sequence_no = 1 ***step_level =*** 2 ***parent_step_id =*** 188 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** 1

execution_mechanism = 2 ***continuation_criteria =*** 2 ***failure_details =***

step_file_name = ***version_no =*** 11.0

start_directory = %TOOLS_DIR% ***parent_version_no =*** 29.0

step_text = %TOOLS_DIR%\Utility\semaphore -waitgroup S -count %MAX_STREAMS%

step_label = Throughput - Stream1 - RF1 ***step_id =*** 437 ***global_flag =*** 0

sequence_no = 1 ***step_level =*** 3 ***parent_step_id =*** 189 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** %INSERT_PARALLELISM%

execution_mechanism = 0 ***continuation_criteria =*** 0 ***failure_details =***

step_file_name = ***version_no =*** 4.0

start_directory = ***parent_version_no =*** 21.1

step_text =

step_label = Throughput - Execute Stream1 RF1 ***step_id =*** 438 ***global_flag =*** 0

sequence_no = 1 ***step_level =*** 4 ***parent_step_id =*** 437 ***enabled_flag =*** -1

iterator_name = INSERT_SEGMENT ***degree_parallelism*** 1

execution_mechanism = 1 ***continuation_criteria =*** 1 ***failure_details =***

step_file_name = ***version_no =*** 5.0

start_directory = DYNAMIC_RF_DBCONNECTION ***parent_version_no =*** 4.0

step_text =

```
DECLARE @updateset INTEGER
DECLARE @SQLstring NVARCHAR(255)

--
--  Delete any previous columns from the insert table
--
DELETE from NEWORDERS_%INSERT_SEGMENT%
DELETE from NEWLINEITEM_%INSERT_SEGMENT%

--
-- Get the current update set value
--
SELECT @updateset=updateset from TPCH_AUX_TABLE

--
--  Generate an SQL statement inserting the current updateset value into the command.
--  Next execute the statement to bulk load the new order insert values
--
SET @SQLstring='bulk insert %DBNAME%..NEWORDERS_%INSERT_SEGMENT% from
''%RF_FLATFILE_DIR%\Orders.tbl.u' + RTRIM(Convert(char,@updateset)) +
'.%INSERT_SEGMENT%'' with (FieldTerminator = ''|'', RowTerminator =''|\n'',tablock)'
EXEC sp_executesql @SQLstring

--
--  Generate an SQL statement inserting the current updateset value into the command.
--  Next execute the statement to bulk load the new lineitem insert values
--
SET @SQLstring='bulk insert %DBNAME%..NEWLINEITEM_%INSERT_SEGMENT% from
''%RF_FLATFILE_DIR%\Lineitem.tbl.u' + RTRIM(Convert(char,@updateset)) +
'.%INSERT_SEGMENT%'' with (FieldTerminator = ''|'', RowTerminator =''|\n'',tablock)'-- PRINT
@SQLstring
EXEC sp_executesql @SQLstring

--
--  Execute the Refresh RF1 inserts
--
exec  RF1_%INSERT_SEGMENT% %BATCH_SIZE%
```

step_label = Throughput - Stream1 - RF2 ***step_id =*** 439 ***global_flag =*** 0

sequence_no = 2 ***step_level =*** 3 ***parent_step_id =*** 189 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** %DELETE_PARALLELISM%

execution_mechanism = 0 ***continuation_criteria =*** 0 ***failure_details =***

step_file_name = ***version_no =*** 2.0

start_directory = ***parent_version_no =*** 21.1

step_text =

step_label = Throughput - Execute Stream1 RF2 ***step_id =*** 440 ***global_flag =*** 0

sequence_no = 1 ***step_level =*** 4 ***parent_step_id =*** 439 ***enabled_flag =*** -1

iterator_name = DELETE_SEGMENT ***degree_parallelism*** 1

execution_mechanism = 1 ***continuation_criteria =*** 1 ***failure_details =***

step_file_name = ***version_no =*** 5.0

start_directory = DYNAMIC_RF_DBCONNECTION ***parent_version_no =*** 2.0

step_text =

```
DECLARE @SQLstring NVARCHAR(255)
DECLARE @updateset INTEGER

--
-- Get the current update set value
--
SELECT @updateset=updateset from TPCH_AUX_TABLE

--
--  Delete any existing index(s) on the temporary table(s)
--
if exists (select name from sysindexes where name = 'OLDORDERS_%DELETE_SEGMENT%_idx')
        drop index OLDORDERS_%DELETE_SEGMENT%.OLDORDERS_%DELETE_SEGMENT%_idx

--
--  Delete any previous columns from the delete table
--
TRUNCATE TABLE OLDORDERS_%DELETE_SEGMENT%

--
--  Generate an SQL statement inserting the current updateset value into
--  the command.  Next execute the statement to bulk load the old order
--  delete values
--
SET @SQLstring='bulk insert %DBNAME%..OLDORDERS_%DELETE_SEGMENT% from
''%RF_FLATFILE_DIR%\Delete.u'  + RTRIM(Convert(char,@updateset)) +  '.%DELETE_SEGMENT%''
with (FieldTerminator = ''|'', RowTerminator =''\n'',tablock)'
EXEC sp_executesql @SQLstring

--
--  Create index on OLDORDERS
--
SET @SQLstring='create unique index OLDORDERS_%DELETE_SEGMENT%_idx on
OLDORDERS_%DELETE_SEGMENT% (O_ORDERKEY)'
EXEC sp_executesql @SQLstring

exec RF2_%DELETE_SEGMENT% %BATCH_SIZE%
```

step_label = Throughput - StreamN - RF1 ***step_id =*** 441 ***global_flag =*** 0

sequence_no = 1 ***step_level =*** 3 ***parent_step_id =*** 191 ***enabled_flag =*** -1

iterator_name = ***degree_parallelism*** %INSERT_PARALLELISM%

execution_mechanism = 0 ***continuation_criteria =*** 0 ***failure_details =***

step_file_name = ***version_no =*** 1.0

start_directory = ***parent_version_no =*** 10.3

step_text =

step_label = Throughput - Execute StreamN RF1 ***step_id =*** 442 ***global_flag =*** 0

sequence_no = 1 ***step_level =*** 4 ***parent_step_id =*** 441 ***enabled_flag =*** -1

iterator_name = INSERT_SEGMENT ***degree_parallelism*** 1

execution_mechanism = 1 ***continuation_criteria =*** 1 ***failure_details =***

step_file_name = ***version_no =*** 4.0

start_directory = DYNAMIC_RF_DBCONNECTION ***parent_version_no =*** 1.0

step_text =

```
DECLARE @updateset INTEGER
DECLARE @SQLstring NVARCHAR(255)

--
--  Delete any previous columns from the insert table
--
DELETE from NEWORDERS_%INSERT_SEGMENT%
DELETE from NEWLINEITEM_%INSERT_SEGMENT%

--
-- Get the current update set value
--
SELECT @updateset=updateset from TPCH_AUX_TABLE

--
--  Generate an SQL statement inserting the current updateset value into the command.
--  Next execute the statement to bulk load the new order insert values
--
SET @SQLstring='bulk insert %DBNAME%..NEWORDERS_%INSERT_SEGMENT% from
''%RF_FLATFILE_DIR%\Orders.tbl.u' + RTRIM(Convert(char,@updateset)) +
'.%INSERT_SEGMENT%'' with (FieldTerminator = ''|'', RowTerminator =''|\n'',tablock)'
EXEC sp_executesql @SQLstring

--
--  Generate an SQL statement inserting the current updateset value into the command.
--  Next execute the statement to bulk load the new lineitem insert values
--
SET @SQLstring='bulk insert %DBNAME%..NEWLINEITEM_%INSERT_SEGMENT% from
''%RF_FLATFILE_DIR%\Lineitem.tbl.u' + RTRIM(Convert(char,@updateset)) +
'.%INSERT_SEGMENT%'' with (FieldTerminator = ''|'', RowTerminator =''|\n'',tablock)'-- PRINT
@SQLstring
EXEC sp_executesql @SQLstring

--
--  Execute the Refresh RF1 inserts
--
exec  RF1_%INSERT_SEGMENT% %BATCH_SIZE%
```

step_label =	Throughput - StreamN - RF2			***step_id =***	443	***global_flag =***	0
sequence_no =	2	***step_level =***	3	***parent_step_id =***	191	***enabled_flag =***	-1
iterator_name =				***degree_parallelism***	%DELETE_PARALLELISM%		
execution_mechanism =	0			***continuation_criteria =***	0	***failure_details =***	
step_file_name =						***version_no =***	1.0
start_directory =						***parent_version_no =***	10.3
step_text =							

step_label = Throughput - Execute StreamN RF2 ***step_id =*** 444 ***global_flag =*** 0

sequence_no = 1 ***step_level =*** 4 ***parent_step_id =*** 443 ***enabled_flag =*** -1

iterator_name = DELETE_SEGMENT ***degree_parallelism*** 1

execution_mechanism = 1 ***continuation_criteria =*** 1 ***failure_details =***

step_file_name = ***version_no =*** 6.0

start_directory = DYNAMIC_RF_DBCONNECTION ***parent_version_no =*** 1.0

step_text =

```
DECLARE @SQLstring NVARCHAR(255)
DECLARE @updateset INTEGER

--
-- Get the current update set value
--
SELECT @updateset=updateset from TPCH_AUX_TABLE

--
--  Delete any existing index(s) on the temporary table(s)
--
if exists (select name from sysindexes where name = 'OLDORDERS_%DELETE_SEGMENT%_idx')
        drop index OLDORDERS_%DELETE_SEGMENT%.OLDORDERS_%DELETE_SEGMENT%_idx

--
--  Delete any previous columns from the delete table
--
TRUNCATE TABLE OLDORDERS_%DELETE_SEGMENT%

--
--  Generate an SQL statement inserting the current updateset value into
--  the command.  Next execute the statement to bulk load the old order
--  delete values
--
SET @SQLstring='bulk insert %DBNAME%..OLDORDERS_%DELETE_SEGMENT% from
''%RF_FLATFILE_DIR%\Delete.u'  + RTRIM(Convert(char,@updateset)) +  '.%DELETE_SEGMENT%''
with (FieldTerminator = ''|'', RowTerminator =''\n'',tablock)'
EXEC sp_executesql @SQLstring

--
--  Create index on OLDORDERS
--
SET @SQLstring='create unique index OLDORDERS_%DELETE_SEGMENT%_idx on
OLDORDERS_%DELETE_SEGMENT% (O_ORDERKEY)'
EXEC sp_executesql @SQLstring

exec RF2_%DELETE_SEGMENT% %BATCH_SIZE%
```

step_label = Throughput - Stream 1 - Increment Update Set ***step_id =*** 497 ***global_flag =*** 0
sequence_no = 3 ***step_level =*** 3 ***parent_step_id =*** 189 ***enabled_flag =*** -1
iterator_name = ***degree_parallelism*** 1
execution_mechanism = 1 ***continuation_criteria =*** 2 ***failure_details =***
step_file_name = ***version_no =*** 1.0
start_directory = DYNAMIC_RF_DBCONNECTION ***parent_version_no =*** 21.1
step_text = UPDATE TPCH_AUX_TABLE SET updateset=updateset+1

step_label = Restore-1-Database ***step_id =*** 916 ***global_flag =*** 0
sequence_no = 6 ***step_level =*** 0 ***parent_step_id =*** 0 ***enabled_flag =*** 0
iterator_name = ***degree_parallelism*** 1
execution_mechanism = 1 ***continuation_criteria =*** 1 ***failure_details =***
step_file_name = %SETUP_DIR%\%DBNAME%\Restore-1-Database.sql ***version_no =*** 5.0
start_directory = DYNAMIC_DBCONNECTION ***parent_version_no =*** 0.0
step_text =

step_label = Restore-2-Database ***step_id =*** 917 ***global_flag =*** 0
sequence_no = 7 ***step_level =*** 0 ***parent_step_id =*** 0 ***enabled_flag =*** 0
iterator_name = ***degree_parallelism*** 1
execution_mechanism = 1 ***continuation_criteria =*** 1 ***failure_details =***
step_file_name = %SETUP_DIR%\%DBNAME%\Restore-2-Database.sql ***version_no =*** 2.0
start_directory = DYNAMIC_DBCONNECTION ***parent_version_no =*** 0.0
step_text =

step_label = SqlServer Startup ***step_id =*** 920 ***global_flag =*** -1
sequence_no = 1 ***step_level =*** 0 ***parent_step_id =*** 0 ***enabled_flag =*** 0
iterator_name = ***degree_parallelism*** 0
execution_mechanism = 2 ***continuation_criteria =*** 0 ***failure_details =***
step_file_name = ***version_no =*** 4.0
start_directory = ***parent_version_no =*** 0.0
step_text =

```
start "SqlServer" sqlservr -c %TRACEFLAGS%
%TOOLS_DIR%\Utility\sleep 25
```

step_label = SqlServer Shutdown ***step_id =*** 921 ***global_flag =*** -1
sequence_no = 2 ***step_level =*** 0 ***parent_step_id =*** 0 ***enabled_flag =*** 0
iterator_name = ***degree_parallelism*** 0
execution_mechanism = 2 ***continuation_criteria =*** 0 ***failure_details =***
step_file_name = ***version_no =*** 4.0
start_directory = ***parent_version_no =*** 0.0
step_text =

```
isql -Usa -P -t10 -Q"shutdown"
%TOOLS_DIR%\Utility\sleep 10
```

step_label = Wait For SQL Server ***step_id =*** 922 ***global_flag =*** -1
sequence_no = 3 ***step_level =*** 0 ***parent_step_id =*** 0 ***enabled_flag =*** 0
iterator_name = ***degree_parallelism*** 0
execution_mechanism = 2 ***continuation_criteria =*** 0 ***failure_details =***
step_file_name = ***version_no =*** 7.0
start_directory = %TOOLS_DIR% ***parent_version_no =*** 0.0
step_text =

```
%TOOLS_DIR%\Utility\sleep 25
```

step_label = Restart SqlServer ***step_id =*** 924 ***global_flag =*** 0
sequence_no = 5 ***step_level =*** 0 ***parent_step_id =*** 0 ***enabled_flag =*** 0
iterator_name = ***degree_parallelism*** 1
execution_mechanism = 0 ***continuation_criteria =*** 0 ***failure_details =***
step_file_name = ***version_no =*** 0.0
start_directory = ***parent_version_no =*** 0.0
step_text =

step_label =	Start SqlServer			***step_id =***	925	***global_flag =***	0
sequence_no =	1	***step_level =***	1	***parent_step_id =***	924	***enabled_flag =***	-1
iterator_name =				***degree_parallelism***	1		
execution_mechanism =	2			***continuation_criteria =***	1	***failure_details =***	
step_file_name =						***version_no =***	3.0
start_directory =						***parent_version_no =***	0.0
step_text =	start sqlservr -c -x -g100						

Step_constraints

workspace_id	constraint_id	step_id	version_no	constraint_type	global_step_id	global_version_no	sequence_no
3	31	925	3.0	1	921	4.0	0

Iterator_values

workspace_id	step_id	version_no	type	iterator_value	sequence_no
3	225	24.0	2	22	0
3	179	21.0	3	1	0
3	243	8.0	3	1	0
3	228	8.0	2	22	0
3	228	8.0	3	1	0
3	228	8.0	1	1	0
3	227	8.0	2	22	0
3	227	8.0	3	1	0
3	227	8.0	1	1	0
3	226	22.0	1	1	0
3	226	22.0	3	1	0
3	226	22.0	2	22	0
3	243	8.0	1	1	0
3	225	24.0	3	1	0

3	245	1.0	3	1	0
3	191	10.3	1	2	0
3	191	10.3	2	%MAX_STREAMS%	0
3	191	10.3	3	1	0
3	189	21.1	3	1	0
3	189	21.1	2	1	0
3	189	21.1	1	1	0
3	181	15.0	1	1	0
3	181	15.0	2	%DELETE_SEGMENTS_PER_UPDATE_SET%	0
3	181	15.0	3	1	0
3	179	21.0	1	1	0
3	179	21.0	2	%INSERT_SEGMENTS_PER_UPDATE_SET%	0
3	225	24.0	1	1	0
3	253	0.0	2	22	0
3	444	6.0	1	1	0
3	444	6.0	3	1	0
3	442	4.0	3	1	0
3	442	4.0	2	%INSERT_SEGMENTS_PER_UPDATE_SET%	0
3	442	4.0	1	1	0
3	440	5.0	3	1	0
3	440	5.0	1	1	0
3	440	5.0	2	%DELETE_SEGMENTS_PER_UPDATE_SET%	0
3	438	5.0	2	%INSERT_SEGMENTS_PER_UPDATE_SET%	0
3	438	5.0	3	1	0
3	438	5.0	1	1	0
3	243	8.0	2	22	0
3	253	0.0	3	1	0
3	444	6.0	2	%DELETE_SEGMENTS_PER_UPDATE_SET%	0
3	251	0.0	3	1	0
3	251	0.0	2	22	0
3	251	0.0	1	1	0
3	249	0.0	2	22	0
3	249	0.0	3	1	0
3	249	0.0	1	1	0
3	247	0.0	1	1	0
3	247	0.0	3	1	0
3	247	0.0	2	22	0
3	245	1.0	1	1	0

3	245	1.0	2	22	0
3	253	0.0	1	1	

Appendix F: Disk Configuration

Disk #	Type	Lineitem	General	Tempdb			Notes:		
0	D	2.34GB	600 MB	1.46 GB	F: FlatFiles 12.43 GB		JunctP. 240		
1	D	2.34GB	600 MB	1.46 GB	F: FlatFiles 12.43 GB		JunctP. 241		
2	D	2.34GB	600 MB	1.46 GB	F: FlatFiles 12.43 GB		JunctP. 242		
3	D	2.34GB	600 MB	1.46 GB	F: FlatFiles 12.43 GB		JunctP. 243		
4	D	2.34GB	600 MB	1.46 GB	F: FlatFiles 12.43 GB		JunctP. 244		
5	D	2.34GB	600 MB	1.46 GB	F: FlatFiles 12.43 GB		JunctP. 245		
6	D	2.34GB	600 MB	1.46 GB	F: FlatFiles 12.43 GB		JunctP. 246		
7	D	2.34GB	600 MB	1.46 GB	F: FlatFiles 12.43 GB		JunctP. 247		
8	D	2.34GB	600 MB	1.46 GB	F: FlatFiles 12.43 GB		JunctP. 248		
9	D	2.34GB	600 MB	1.46 GB	F: FlatFiles 12.43 GB		JunctP. 249		
10	D	2.34GB	600 MB	1.46 GB	F: FlatFiles 12.43 GB		JunctP. 250		
11	D	2.34GB	600 MB	1.46 GB	F: FlatFiles 12.43 GB		JunctP. 251		
12	D	2.34GB	600 MB	1.46 GB	F: FlatFiles 12.43 GB		JunctP. 252		
13	D	2.34GB	600 MB	1.46 GB	F: FlatFiles 12.43 GB		JunctP. 253		
14	D	2.34GB	600 MB	1.46 GB	F: FlatFiles 12.43 GB		JunctP. 254		
15	D	2.34GB	600 MB	1.46 GB	F: FlatFiles 12.43 GB		JunctP. 255		
16	D	2.34GB	600 MB	1.46 GB	F: FlatFiles 12.43 GB		JunctP. 256		
17	D	2.34GB	600 MB	1.46 GB	F: FlatFiles 12.43 GB		JunctP. 257		
18	D	2.34GB	600 MB	1.46 GB	F: FlatFiles 12.43 GB		JunctP. 258		
19	D	2.34GB	600 MB	1.46 GB	F: FlatFiles 12.43 GB		JunctP. 259		
20	D	2.34GB	600 MB	1.46 GB			JunctP. 238	PagFile 3.22GB	3.35GB U
21	D	2.34GB	600 MB	1.46 GB	B: Acid Bkup	M: MountP.	JunctP. 220	PagFile 3.22GB	3.1GB Ur
22	D	2.34GB	600 MB	1.46 GB			JunctP. 237	PagFile 3.22GB	3.35GB U
23	D	2.34GB	600 MB	1.46 GB			JunctP. 222	PagFile 3.22GB	3.35GB U
24	D	2.34GB	600 MB	1.46 GB			JunctP. 236	PagFile 3.22GB	3.35GB U
25	D	2.34GB	600 MB	1.46 GB			JunctP. 224	PagFile 3.22GB	3.35GB U
26	D	2.34GB	600 MB	1.46 GB			JunctP. 235	PagFile 3.22GB	3.35GB U
27	D	2.34GB	600 MB	1.46 GB			JunctP. 226	PagFile 3.22GB	3.35GB U
28	D	2.34GB	600 MB	1.46 GB			JunctP. 234	PagFile 3.22GB	3.35GB U
29	D	2.34GB	600 MB	1.46 GB			JunctP. 228	PagFile 3.22GB	3.35GB U
30	D	2.34GB	600 MB	1.46 GB			JunctP. 229	PagFile 3.22GB	3.35GB U
31	D	2.34GB	600 MB	1.46 GB			JunctP. 230	PagFile 3.22GB	3.35GB U
32	D	2.34GB	600 MB	1.46 GB			JunctP. 231	PagFile 3.22GB	3.35GB U
33	D	2.34GB	600 MB	1.46 GB			JunctP. 232	PagFile 3.22GB	3.35GB U
34	D	2.34GB	600 MB	1.46 GB			JunctP. 233	PagFile 3.22GB	3.35GB U
35	D	2.34GB	600 MB	1.46 GB			JunctP. 227	PagFile 3.22GB	3.35GB U
36	D	2.34GB	600 MB	1.46 GB			JunctP. 225	PagFile 3.22GB	3.35GB U
37	D	2.34GB	600 MB	1.46 GB			JunctP. 223	PagFile 3.22GB	3.35GB U
38	D	2.34GB	600 MB	1.46 GB	B: Acid Bkup		JunctP. 221	PagFile 3.22GB	3.35GB U
39	D	2.34GB	600 MB	1.46 GB	B: Acid Bkup	M: MountP.	JunctP. 219	PagFile 3.22GB	3.1GB Ur
40	D	2.34GB	600 MB	1.46 GB	F: FF300g - 12.43 GB		JunctP. 261		

41	D	2.34GB	600 MB	1.46 GB	F: FF300g - 12.43 GB	JunctP. 262
42	D	2.34GB	600 MB	1.46 GB	F: FF300g - 12.43 GB	JunctP. 263
43	D	2.34GB	600 MB	1.46 GB	F: FF300g - 12.43 GB	JunctP. 264
44	D	2.34GB	600 MB	1.46 GB	F: FF300g - 12.43 GB	JunctP. 265
45	D	2.34GB	600 MB	1.46 GB	F: FF300g - 12.43 GB	JunctP. 266
46	D	2.34GB	600 MB	1.46 GB	F: FF300g - 12.43 GB	JunctP. 267
47	D	2.34GB	600 MB	1.46 GB	F: FF300g - 12.43 GB	JunctP. 268
48	D	2.34GB	600 MB	1.46 GB	F: FF300g - 12.43 GB	JunctP. 269
49	D	2.34GB	600 MB	1.46 GB	F: FF300g - 12.43 GB	JunctP. 270
50	D	2.34GB	600 MB	1.46 GB	F: FF300g - 12.43 GB	JunctP. 271
51	D	2.34GB	600 MB	1.46 GB	F: FF300g - 12.43 GB	JunctP. 272
52	D	2.34GB	600 MB	1.46 GB	300GB Bkup G:	JunctP. 273
53	D	2.34GB	600 MB	1.46 GB	300GB Bkup G:	JunctP. 274
54	D	2.34GB	600 MB	1.46 GB	300GB Bkup G:	JunctP. 275
55	D	2.34GB	600 MB	1.46 GB	300GB Bkup G:	JunctP. 276
56	D	2.34GB	600 MB	1.46 GB	300GB Bkup G:	JunctP. 277
57	D	2.34GB	600 MB	1.46 GB	300GB Bkup I:	JunctP. 278
58	D	2.34GB	600 MB	1.46 GB	300GB Bkup I:	JunctP. 279
59	D	2.34GB	600 MB	1.46 GB	300GB Bkup I:	JunctP. 280
60	D	2.34GB	600 MB	1.46 GB	300GB Bkup I:	JunctP. 282
61	D	2.34GB	600 MB	1.46 GB	300GB Bkup I:	JunctP. 283
62	D	2.34GB	600 MB	1.46 GB	300GB Bkup J:	JunctP. 284
63	D	2.34GB	600 MB	1.46 GB	300GB Bkup J:	JunctP. 285
64	D	2.34GB	600 MB	1.46 GB	300GB Bkup J:	JunctP. 286
65	D	2.34GB	600 MB	1.46 GB	300GB Bkup J:	JunctP. 287
66	D	2.34GB	600 MB	1.46 GB	300GB Bkup J:	JunctP. 288
67	D	2.34GB	600 MB	1.46 GB	300GB Bkup N:	JunctP. 289
68	D	2.34GB	600 MB	1.46 GB	300GB Bkup N:	JunctP. 290
69	D	2.34GB	600 MB	1.46 GB	300GB Bkup N:	JunctP. 291
70	D	2.34GB	600 MB	1.46 GB	300GB Bkup N:	JunctP. 292
71	D	2.34GB	600 MB	1.46 GB	300GB Bkup N:	JunctP. 293
72	D	2.34GB	600 MB	1.46 GB	300GB Bkup O:	JunctP. 294
73	D	2.34GB	600 MB	1.46 GB	300GB Bkup O:	JunctP. 295
74	D	2.34GB	600 MB	1.46 GB	300GB Bkup O:	JunctP. 296
75	D	2.34GB	600 MB	1.46 GB	300GB Bkup O:	JunctP. 297
76	D	2.34GB	600 MB	1.46 GB	300GB Bkup O:	JunctP. 298
77	D	2.34GB	600 MB	1.46 GB	300GB Bkup R:	JunctP. 299
78	D	2.34GB	600 MB	1.46 GB	300GB Bkup R:	JunctP. 300
79	D	2.34GB	600 MB	1.46 GB	300GB Bkup R:	JunctP. 301
80	D	2.34GB	600 MB	1.46 GB	300GB Bkup R:	JunctP. 303
81	D	2.34GB	600 MB	1.46 GB	300GB Bkup R:	JunctP. 304
82	D	2.34GB	600 MB	1.46 GB	300GB Bkup S:	JunctP. 305
83	D	2.34GB	600 MB	1.46 GB	300GB Bkup S:	JunctP. 306
84	D	2.34GB	600 MB	1.46 GB	300GB Bkup S:	JunctP. 307
85	D	2.34GB	600 MB	1.46 GB	300GB Bkup S:	JunctP. 308
86	D	2.34GB	600 MB	1.46 GB	300GB Bkup S:	JunctP. 309
87	D	2.34GB	600 MB	1.46 GB	300GB Bkup T:	JunctP. 310

88	D	2.34GB	600 MB	1.46 GB	300GB Bkup T:	JunctP. 311
89	D	2.34GB	600 MB	1.46 GB	300GB Bkup T:	JunctP. 312
90	D	2.34GB	600 MB	1.46 GB	300GB Bkup T:	JunctP. 313
91	D	2.34GB	600 MB	1.46 GB	300GB Bkup T:	JunctP. 314
92	D	2.34GB	600 MB	1.46 GB	300GB Bkup U:	JunctP. 315
93	D	2.34GB	600 MB	1.46 GB	300GB Bkup U:	JunctP. 316
94	D	2.34GB	600 MB	1.46 GB	300GB Bkup U:	JunctP. 317
95	D	2.34GB	600 MB	1.46 GB	300GB Bkup U:	JunctP. 318
96	D	2.34GB	600 MB	1.46 GB	300GB Bkup U:	JunctP. 319
97	D	2.34GB	600 MB	1.46 GB	300GB Bkup V:	JunctP. 320
98	D	2.34GB	600 MB	1.46 GB	300GB Bkup V:	JunctP. 321
99	D	2.34GB	600 MB	1.46 GB	300GB Bkup V:	JunctP. 322
100	D	2.34GB	600 MB	1.46 GB	300GB Bkup V:	JunctP. 324
101	D	2.34GB	600 MB	1.46 GB	300GB Bkup V:	JunctP. 325
102	D	2.34GB	600 MB	1.46 GB	300GB Bkup W:	JunctP. 326
103	D	2.34GB	600 MB	1.46 GB	300GB Bkup W:	JunctP. 327
104	D	2.34GB	600 MB	1.46 GB	300GB Bkup W:	JunctP. 328
105	D	2.34GB	600 MB	1.46 GB	300GB Bkup W:	JunctP. 329
106	D	2.34GB	600 MB	1.46 GB	300GB Bkup W:	JunctP. 330
107	D	2.34GB	600 MB	1.46 GB	300GB Bkup X:	JunctP. 331
108	D	2.34GB	600 MB	1.46 GB	300GB Bkup X:	JunctP. 332
109	D	2.34GB	600 MB	1.46 GB	300GB Bkup X:	JunctP. 333
110	D	2.34GB	600 MB	1.46 GB	300GB Bkup X:	JunctP. 334
111	D	2.34GB	600 MB	1.46 GB	300GB Bkup X:	JunctP. 335
112	D	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 336
113	D	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 337
114	D	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 338
115	D	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 339
116	D	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 340
117	D	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 341
118	D	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 342
119	D	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 343
120	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 345
121	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 346
122	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 347
123	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 348
124	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 349
125	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 350
126	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 351
127	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 352
128	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 353
129	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 354
130	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 355
131	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 356
132	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 357
133	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 358
134	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 359

135	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 360
136	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 361
137	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 362
138	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 363
139	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 364
140	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 0
141	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 1
142	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 2
143	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 3
144	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 4
145	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 5
146	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 6
147	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 7
148	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 8
149	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 9
150	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 10
151	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 11
152	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 12
153	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 13
154	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 14
155	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 15
156	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 16
157	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 17
158	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 18
159	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 19
160	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 20
161	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 21
162	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 22
163	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 23
164	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 24
165	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 25
166	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 26
167	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 27
168	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 28
169	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 29
170	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 30
171	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 31
172	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 32
173	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 33
174	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 34
175	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 35
176	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 36
177	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 37
178	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 38
179	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 39
180	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 40
181	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 41

182	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 42
183	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 43
184	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 44
185	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 45
186	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 46
187	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 47
188	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 48
189	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 49
190	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 50
191	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 51
192	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 52
193	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 53
194	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 54
195	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 55
196	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 56
197	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 57
198	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 58
199	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 59
200	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 60
201	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 61
202	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 62
203	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 63
204	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 64
205	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 65
206	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 66
207	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 67
208	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 68
209	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 69
210	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 70
211	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 71
212	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 72
213	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 73
214	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 74
215	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 75
216	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 76
217	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 77
218	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 78
219	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 79
220	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 80
221	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 81
222	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 82
223	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 83
224	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 84
225	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 85
226	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 86
227	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 87
228	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 88

229	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 89
230	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 90
231	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 91
232	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 92
233	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 93
234	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 94
235	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 95
236	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 96
237	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 97
238	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 98
239	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 99
240	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 100
241	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 101
242	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 102
243	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 103
244	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 104
245	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 105
246	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 106
247	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 107
248	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 108
249	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 109
250	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 110
251	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 111
252	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 112
253	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 113
254	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 114
255	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 115
256	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 116
257	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 117
258	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 118
259	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 119
260	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 120
261	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 121
262	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 122
263	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 123
264	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 124
265	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 125
266	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 126
267	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 127
268	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 128
269	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 129
270	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 130
271	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 131
272	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 132
273	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 133
274	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 134
275	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 135

276	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 136
277	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 137
278	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 138
279	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 139
280	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 140
281	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 141
282	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 142
283	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 143
284	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 144
285	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 145
286	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 146
287	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 147
288	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 148
289	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 149
290	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 150
291	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 151
292	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 152
293	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 153
294	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 154
295	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 155
296	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 156
297	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 157
298	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 158
299	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 159
300	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 160
301	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 161
302	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 162
303	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 163
304	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 164
305	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 165
306	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 166
307	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 167
308	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 168
309	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 169
310	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 170
311	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 171
312	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 172
313	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 173
314	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 174
315	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 175
316	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 176
317	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 177
318	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 178
319	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 179
320	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 180
321	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 181
322	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 182

323	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 183
324	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 184
325	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 185
326	B	2.34GB	600 MB	1.46 GB	L: data from LE041	JunctP. 186
327	D	Z: ACID Log			Unallocated	
328	B	2.34GB	600 MB	1.46 GB		JunctP. 188
329	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 189
330	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 190
331	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 191
332	B	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 192
333	D		Unallocated			
334	D		Unallocated			
335	D	H: 300GB Log Mirror - 47.4GB			21 GB Unallocated	
336	D	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 196
337	D	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 197
338	D	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 198
339	D	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 199
340	D	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 200
341	D	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 201
342	D	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 202
343	D	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 203
344	D	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 204
345	D	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 205
346	D	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 206
347	D	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 207
348	D	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 208
349	D	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 209
350	D	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 210
351	D	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 211
352	D	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 212
353	D		Unallocated			
354	D	2.34GB	600 MB	1.46 GB	12.43 GB Unallocated	JunctP. 187
355	D	K: TempDB Log			1.2 GB Unallocated	
356	D		Unallocated			Z: ACID Log
357	D	ACID db			10.9 GB Unallocated	
358	D	ACID db			10.9 GB Unallocated	
359	D	H: 300GB Log Mirror - 47.4GB			21 GB Unallocated	
360	B	C:				