

FUJITSU

TPC Benchmark™ C

Full Disclosure Report

**Fujitsu GRANPOWER 5000
Model 670**

using

Microsoft SQL Server 6.5 Enterprise Edition

and

Microsoft Windows NT Server 4.0 Enterprise Edition

**First Edition
Submitted for review 5th February 1998**

First Printing, February 1998

FUJITSU ICL Computers LTD. believes that the information in this document is accurate as of its publication date. The information in this document is subject to change without notice. FUJITSU ICL Computers LTD. assumes no responsibility for any errors that may appear in this document.

The pricing information in this document is believed to accurately reflect prices in effect on the indicated dates. However, FUJITSU ICL Computers LTD. provides no warranty on the pricing information in this document.

The performance information in this document is for guidance only. System performance is highly dependent on many factors including system hardware, system use and software, and user application characteristics. Customer applications must be carefully evaluated before estimating performance. FUJITSU ICL Computers LTD. does not warrant or represent that a user can or will achieve similar performance expressed in transactions per minute (tpmC) or normalised price/performance (\$/tpmC). No warranty on system performance or price/performance is expressed or implied in this document.

GRANPOWER is a registered trademark of Fujitsu Limited.

Intel, PentiumII, and Pentium Pro are registered trademarks of Intel Corporation,

Microsoft Windows NT and SQL Server are registered trademarks of Microsoft Corporation.

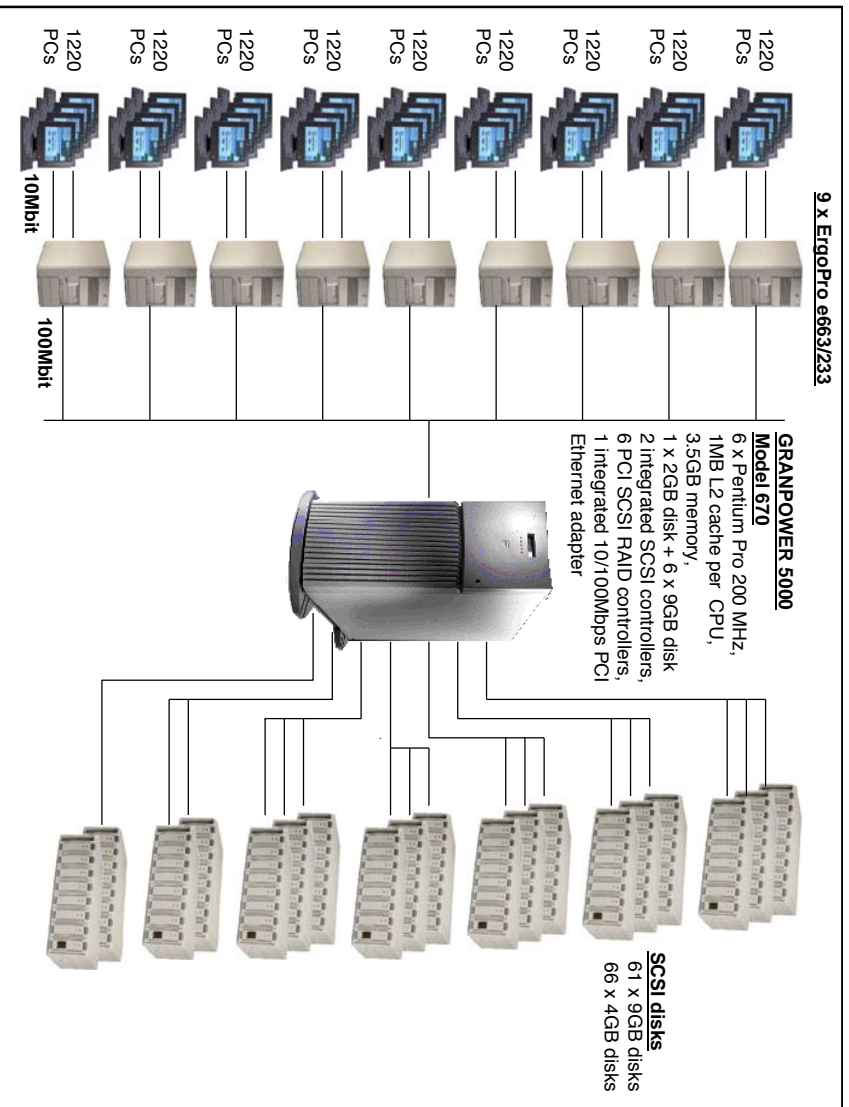
TPC Benchmark, TPC-C and tpmC are registered trademarks of the Transaction Processing Performance Council.



GRANPOWER 5000 Model 670 C/S

TPC-C Rev 3.3
Report Date:
5th February
1998

Total System Cost	TPC-C Throughput	Price/Performance	Availability Date
\$503,720	13,391.13 tpmc	\$37.62/tpmc	31 st March 1998
Processors	Database Manager	Operating System	Other Software
6 x 200MHz Pentium®Pro 1MB L2 cache	Microsoft SQL Server 6.5 Enterprise Edition	Microsoft Windows NT Server 4.0 Enterprise Edition	Microsoft IIS 3.0 BEA Tuxedo 6.3 CFS
			Number of Users 10,980



System Component	Quantity	Server Description	Quantity	Each Client (9 total)
Processors	6	200 MHz Pentium Pro	1	233 Mhz Pentium II
Cache Memory		1MB L2		512KB L2
Memory	1	3584MB	1	128MB
Disk Controller	6	3-channel Mylex DAC960PJ Integrated Adaptec 7880	1	Integrated IDE
Disk Drives	1	2 GB	1	2GB IDE Drive
	66	4 GB		
	67	9 GB		
Total Storage		837.54 GB	1	2GB
CNT/Pane Drive	1	CD-ROM	1	CD-ROM



GRANPOWER 5000 Model 670

C/S

TPC-C Rev 3.3
Report Date:
5th February 1998

Description	Part Number	Supplier	Unit Price	Qty	Extended Price	5 yr. Maint. Price
Server Hardware						
GRANPOWER 5000 Model 670 6xPentium Pro 200MHz/1024KB Memory Board	GP5000	Brand Fujitsu	37495	1	37495	13089
512MB ECC RAM Upgrade	SV642014	Fujitsu	360	1	360	126
Mylex DAC960P/3 channel RAID controller (including 2 spares)	SV642015	Fujitsu	5000	7	35000	10065
Disk Shelf Kit	DACP1-3-8E-MV1	Mylex	1995	8	15960	400
TRI-Link Connector and Cable	SV642052	Fujitsu	1125	19	21375	1018
3m VHD to VHD SCSI Cable	SV642048	Fujitsu	270	1	270	24
2GB 7200 1" SCA DISK	SV642055	Fujitsu	155	19	2945	265
9GB 10,000 1,6" SCA DISK EXP	SV642022	Fujitsu	525	1	525	184
9GB 7200 1,6" SCA DISK	SV565050	Fujitsu	1695	10	16950	2653
9GB 7200 1,6" SCA DISK EXP	SV642024	Fujitsu	1450	6	8700	1362
4GB 7200 1" SCA DISK EXP	SV642032	Fujitsu	1450	51	73950	11575
4GB Internal DDS-2 DAT Drive	SV642031	Fujitsu	850	66	56100	9009
ErgoPro e142 Colour Monitor	SV520009	Fujitsu	1140	1	1140	399
ErgoPro e105 keyboard (US-104)	PD42E193	Fujitsu	265	1	265	93
Mouse Kit	PK040657	Fujitsu	45	1	45	18
Power Lead (USA)	PK080130	Fujitsu	26	1	26	0
	57495004	Fujitsu	30	1	30	30
				1	271136	50310
Server Software						
Microsoft Windows NT Server v.4.0 Enterprise Edition (inc 25 CAL)		Microsoft	3,999	1	3999	0
Microsoft SQL Server 6.5 Enterprise Edition unlimited user licence		Microsoft	28,999	1	28999	0
Microsoft Software Maintenance		Microsoft	10,475	1	0	10475
				1	32998	10475
				Subtotal		
Client Hardware						
ErgoPro e663/233	UID64ASE3001	Fujitsu	1884	9	16956	6782
64MB EDO Memory Module	PL060152	Fujitsu	336	18	6048	1814
Ether Express PRO/100B	PN010051	Fujitsu	104	27	2808	758
ErgoPro e142 Colour Monitor	PD42E193	Fujitsu	265	9	2385	811
ErgoPro e105 keyboard (US-104)	PK040657	Fujitsu	45	9	405	162
Fujitsu Mouse Kit	PK080130	Fujitsu	26	9	234	0
				9	28602	10327
Client Software						
Microsoft Windows NT Server v. 4.0		Microsoft	809	9	7281	0
Microsoft SQL Server Workstation (includes prog toolkit)		Microsoft	499	1	499	0
Microsoft Visual C++ 32 bit edition (subscription)		Microsoft	499	1	499	0
Tuxedo 6.3 Client threads for NT4.0		BEA	3000	9	27000	20250
				9	35279	20250
				Subtotal		
User Connectivity						
Complex MicroHub TP-1008C 8 port 10BaseT Hubs (including 10% spares)		Complex	34	1515	52631	spared
Complex MicroHub MX1-205 100TX 5 port Hubs (including 2 spares)		Complex	187	4	747	spared
				4	53378	0
				Subtotal		
				Other Discounts	0	(\$9,036)
				-/5		
				Total	\$421,393	\$82,326
					Five-Year TCO*	\$503,720
					tpmc:	13,391.13
					\$/tpmc:	37.62

Notes:

- * Pricing: 1=FUJITSU ICL Computers LTD NAD Price book/Lexia, 2=Microsoft, 3=BEA, 4=NetSales, 5=Northern Telecom, 6=Mylex
- * All Microsoft maintenance is covered by maintenance for SQL Server
- * Northern Telecom discount of 15% is based on dollar volume discount and 5-year prepaid maintenance
- * Benchmark results and test methodology were audited by Tom Sawyer 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. Individual negotiated discounts are not permitted. Special Prices based on assumptions about past or future purchases are not permitted. All discounts reflect standard pricing policies for the listed components. For complete details, see the pricing section of the TPC benchmark 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

MQTh, computed Maximum Qualified Throughput		13,391.13 tpmc
% throughput difference, reported & reproducibility runs		
		1.37%
Response Times (in seconds)	90th percentile	Average
New-Order	1.11	0.74
Payment	0.62	0.42
Order-Status	1.81	1.20
Delivery (interactive portion)	0.31	0.29
Delivery (deferred portion)	1.29	0.86
Stock-Level	6.82	3.95
Menu	0.30	0.20
Response time delay added for emulated components	Response Time	Menu
New-Order	0.1	0.1
Payment	0.1	0.1
Order-Status	0.1	0.1
Delivery (interactive portion)	0.1	0.1
Stock-Level	0.1	0.1
Transaction Mix in percentage of total transactions		
New-Order		44.60%
Payment		43.17%
Order-Status		4.11%
Delivery		4.11%
Stock-Level		4.01%
Keying/Think Times (in seconds)	Minimum	Average
New-Order	18.01/0.00	18.01/12.10
Payment	3.00/0.00	3.01/12.04
Order-Status	2.00/0.00	2.01/10.16
Delivery	2.00/0.00	2.01/5.10
Stock-Level	2.00/0.00	2.01/5.12
Test Duration and Checkpointing		
Measurement interval		30 mins
Transactions (all types) completed during measurement interval		900,799
Number of checkpoints		One
Checkpoint interval		30 mins
Ramp up period		59 mins
		Maximum
		18.07/121.00
		3.07/121.01
		2.07/83.84
		2.05/40.47
		2.07/42.37

ABSTRACT

This report documents the results of the TPC Benchmark C test conducted on the Fujitsu GRANPOWER 5000 Model 670 according to the TPC Benchmark C Standard Specification, Revision 3.3, released April 8, 1997. The tests were performed using the Microsoft SQL Server 6.5 Enterprise Edition running under the Microsoft Windows NT Server 4.0 Enterprise Edition operating system.

Two standard metrics, transactions per minute (tpmC) and price per tpmC (\$/tpmC), are reported.

Company	System Name	Database Software	Operating System Software
Fujitsu	GRANPOWER 5000 Model 670	Microsoft SQL Server 6.5 Enterprise Edition	Microsoft Windows NT Server 4.0 Enterprise Edition
Availability Date: 31 st March 1998			

Total System Cost	TPC-C Throughput	Price Performance
Hardware Software 5 Years Maintenance	Sustained maximum throughput of system running TPC Benchmark C expressed in transactions per minute	Total system cost/tpmC throughput (\$503,720/13,391.13 tpmC)
\$503,720	13,391.13 tpmC	\$37.62 per tpmC

CONTENTS

ABSTRACT.....	1
CONTENTS	2
PREFACE.....	5
1 GENERAL ITEMS.....	6
1.1 APPLICATION CODE AND DEFINITION STATEMENTS	6
1.2 BENCHMARK SPONSOR	6
1.3 PARAMETER SETTINGS	6
1.4 CONFIGURATION DIAGRAMS.....	7
<i>Figure 1.4.1 Benchmarked Configuration.....</i>	<i>8</i>
<i>Figure 1.4.2 Priced Configuration.....</i>	<i>8</i>
2 CLAUSE 1 LOGICAL DATABASE DESIGN.....	9
2.1 DATA AND SPACE DEFINITION.....	9
2.2 PHYSICAL ORGANISATION OF DATA.....	9
2.3 INSERT AND/OR DELETE OPERATIONS.....	9
2.4 PARTITIONING.....	9
2.5 TABLE REPLICATION	9
2.6 TABLE ATTRIBUTES.....	9
3 CLAUSE 2 TRANSACTION AND TERMINAL PROFILE RELATED ITEMS.....	10
3.1 RANDOM NUMBER VERIFICATION METHOD	10
3.2 TERMINAL SCREEN LAYOUTS.....	10
3.3 PRICED TERMINAL FEATURE VERIFICATION.....	10
3.4 PRESENTATION MANAGER OR INTELLIGENT TERMINAL.....	10
3.5 TRANSACTION STATISTICS.....	11
<i>Table 3.5.1 Transaction Statistics.....</i>	<i>11</i>
3.6 QUEUING MECHANISM OF DELIVERY.....	11
4 CLAUSE 3 TRANSACTION AND SYSTEM PROPERTIES.....	12
4.1 TRANSACTION SYSTEM PROPERTIES (ACID).....	12
4.2 ATOMICITY REQUIREMENTS.....	12
4.2 CONSISTENCY REQUIREMENTS.....	12
4.3 ISOLATION.....	13
4.4 DURABILITY REQUIREMENTS.....	14
5 CLAUSE 4 SCALING AND DATABASE POPULATION.....	16
5.1 INITIAL CARDINALITY OF TABLES.....	16
<i>Table 5.1.1 Initial Cardinality of Tables.....</i>	<i>16</i>
5.2 DATA LAYOUT.....	16
<i>Table 5.2.1 RAID Disk Configuration for Benchmarked System.....</i>	<i>17</i>
<i>Table 5.3.1 Disk Administrator Configuration.....</i>	<i>18</i>
5.3 DATABASE MODEL.....	19
5.4 DATABASE MAPPING.....	19
5.5 180 DAY SPACE.....	19
6 CLAUSE 5 PERFORMANCE METRICS AND RESPONSE TIMES.....	20
6.1 MEASURED TPMC.....	20
6.2 RESPONSE TIMES.....	20
<i>Table 6.2.1 Response Times.....</i>	<i>20</i>

Table 6.3.1 Keying Times..... 20

Table 6.3.2 Think Times..... 20

6.4 RESPONSE TIME FREQUENCY DISTRIBUTIONS..... 21

Figure 6.4.1 New Order Response Time Distribution..... 21

Figure 6.4.2 Payment Response Time Distribution..... 21

Figure 6.4.3 Order Status Response Time Distribution..... 22

Figure 6.4.4 Delivery Response Time Distribution..... 22

Figure 6.4.5 Stock Level Response Time Distribution..... 23

6.5 THINK TIME FREQUENCY DISTRIBUTIONS..... 23

Figure 6.5.1 New Order Think Time Distribution..... 23

6.6 PERFORMANCE CURVE FOR RESPONSE TIME VS. THROUGHPUT..... 24

Figure 6.6.1 Response Times versus Throughput..... 24

6.7 THROUGHPUT VS. ELAPSED TIME: NEW-ORDER TRANSACTION..... 25

Figure 6.7.1 New Order Throughput vs Elapsed Time..... 25

6.8 STEADY STATE METHOD..... 25

6.9 WORK PERFORMED DURING STEADY STATE..... 26

6.10 REPRODUCIBILITY METHOD..... 26

6.11 DURATION OF MEASUREMENT..... 26

Table 6.13.1 Transaction Statistics..... 27

6.14 CHECKPOINT STATISTICS..... 28

7 CLAUSE 6 SUT, DRIVER AND COMMUNICATION DEFINITION..... 29

7.1 RTE INPUTS..... 29

7.2 FUNCTIONS AND PERFORMANCE OF EMULATED COMPONENTS..... 29

7.3 FUNCTIONAL DIAGRAMS..... 29

7.4 NETWORK BANDWIDTH AND CONFIGURATION..... 29

7.5 OPERATOR INTERVENTION..... 30

8 CLAUSE 7 PRICING..... 31

8.1 SYSTEM PRICING..... 31

8.2 AVAILABILITY, THROUGHPUT AND PRICE/PERFORMANCE..... 32

8.3 COUNTRY SPECIFIC PRICING..... 32

8.4 USAGE PRICING..... 32

9 CLAUSE 9 AUDIT..... 33

APPENDIX A SOURCE CODE..... 35

A.1 DELIRPT.C..... 35

A.2 DELIVERY.C..... 45

A.3 HTTPEXT.H..... 56

A.4 INSTALL.C..... 58

A.5 INSTALL.H..... 65

A.6 INSTALL.RC..... 66

A.7 RESOURCE.H..... 68

A.8 MAKEFILE..... 68

A.9 NEWORDER.C..... 70

A.10 PAYMENT.C..... 81

A.11 ORDERSTATUS.C..... 91

A.12 STOCKLEVEL.C..... 102

A.13 TPCC.C..... 111

A.14 TPCC.DDF..... 163

A.15 TPCC.H..... 163

A.16 TPCC.RC..... 168

A.17 TRANS.H..... 169

APPENDIX R SERVER SOURCE CODE..... 173

B.1	CREATEDB.SQL.....	173
B.2	DBOPT1.SQL.....	173
B.3	DBOPT2.SQL.....	174
B.4	DISKINIT.SQL.....	174
B.5	IDXCUSCL.SQL.....	175
B.6	IDXCUSNC.SQL.....	176
B.7	IDXDISCL.SQL.....	176
B.8	IDXTMCL.SQL.....	176
B.9	IDXNODCL.SQL.....	176
B.10	IDXODLCL.SQL.....	177
B.11	IDXORDCL.SQL.....	177
B.12	IDXSTKCL.SQL.....	177
B.13	IDXWARCL.SQL.....	178
B.14	PINTABLE.SQL.....	178
B.15	SEGMENT.SQL.....	178
B.16	TABLES.SQL.....	179
B.17	TPCCBCP.SQL.....	181
B.18	TPCCIRL.SQL.....	181
B.19	NEWORD.SQL.....	181
B.20	ORDSTAT.SQL.....	184
B.21	PAYMENT.SQL.....	185
B.22	DELIVERY.SQL.....	188
B.23	STOCKLEV.SQL.....	189
B.24	MAKEFILE.....	190
B.25	TPCCLDR.C.....	191
B.26	GETARGS.C.....	210
B.27	RANDOM.C.....	217
B.28	UTTL.C.....	220

APPENDIX C BENCHCRAFT RTE CONFIGURATION..... 228

APPENDIX D TUNABLE PARAMETERS 231

D.1	MICROSOFT WINDOWS NT SERVER 4.0.....	231
D.2	MICROSOFT WINDOWS NT CLIENT 4.0.....	231
D.3	TUXEDO CONFIGURATION.....	234
D.4	MICROSOFT SQL SERVER 6.5 PARAMETERS.....	235
D.5	MICROSOFT SQL SERVER 6.5 STARTUP PARAMETERS.....	235

APPENDIX F THIRD PARTY LETTERS 237

APPENDIX G 180 DAY SPACE CALCULATIONS 242

APPENDIX H ATTESTATION LETTER 244

PREFACE

This is the full disclosure report for the TPC Benchmark C Benchmark test conducted on the Fujitsu GRANPOWER 5000 Model 670 according to the TPC Benchmark C Standard Specification, Revision 3.3, released April 8 1997. The TPC Benchmark C Standard was developed by the Transaction Processing Performance Council (TPC). Fujitsu is an active member of the TPC.

TPC Benchmark C (TPC-C) is an OLTP workload. It is a mixture of read-only and update intensive transactions that simulate the activities found in complex OLTP application environments. It does so by exercising a breadth of system components associated with such environments, which are characterised by:

- The simultaneous execution of multiple transaction types that span a breadth of complexity
- On-line and deferred transaction execution modes
- Multiple on-line terminal sessions
- Moderate system and application execution time
- Significant disk input/output
- Transaction integrity (ACID properties)
- Non-uniform distribution of data access through primary and secondary keys
- Databases consisting of many tables with a wide variety of sizes, attributes and relationships
- Contention on data access and update

The performance metric reported by TPC-C is a "business throughput" measuring the number of new orders processed per minute. Multiple transactions are used to simulate the business activity of processing an order, and each transaction is subject to a response time constraint. The performance metric for this benchmark is expressed in transactions-per-minute-C (tpmC). To be compliant with the TPC-C standard, all references to tpmC results must include the tpmC rate, the associated price-per-tpmC, and the availability date of the priced configuration.

Despite that fact that this benchmark offers a rich environment that emulates many OLTP applications, this benchmark does not reflect the entire range of OLTP requirements. In addition the extent to which a customer can achieve the results reported by a vendor is highly dependent on how closely TPC-C 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 system design and implementation. Relative system performance will vary as a result of these and other factors. Therefore, TPC-C should not be used as a substitute for a specific customer application benchmarking when critical capacity planning and/or product evaluation decisions are contemplated.

Copies of this Full Disclosure Report are available on request from:

TPC Administrator	or	Transaction Processing Performance Council
FUJITSU ICL Computers LTD		c/o Shanley Public Relations
Lovelace Road		777 North First Street
Bracknell		Suite 6000
Berkshire		San Jose
RG12 8SN		CA 95112-6311
ENGLAND		

Tel: (+44) 1344 473076
Fax: (+44) 1344 473000

Tel: 408 295 8894
Fax 408 295 9968

Copies of the Executive Summary can be downloaded from :

<http://www.tpc.org/execsum.TPCC.html>

1 GENERAL ITEMS

1.1 Application Code and Definition Statements

The applicable program (as defined in Clause 2.1.7) must be disclosed. This includes, but is not limited to, the code implementing the five transactions and the terminal input and output functions.

Appendix A contains all source code implemented in this benchmark.

1.2 Benchmark Sponsor

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

This TPC benchmark test was sponsored by FUJITSU ICL Computers LTD and Fujitsu Limited. The NT/SQL TPC-C benchmark kit was developed and supplied by Microsoft Corporation.

1.3 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:

- *Database tuning options*
- *Recovery/Commit options*
- *Consistency/locking options*
- *Operating system and application configuration parameters.*
- *Compilation and linkage options and run-time optimizations used to create/install applications, OS, and/or databases.*

Appendix D contains the tunable parameters for the database and the operating system.

1.4 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 (any controllers, if applicable).*
- *Number of channels or bus connections to disk units, including their protocol type.*
- *Number of LAN (e.g. Ethernet) connections, including routers, workstations, terminals, etc., that were physically used in the test or are incorporating into the pricing structure (see Clause 8.1.8).*
- *Type and the run time execution location of the software components (e.g., DBMS, client processes, transaction monitors, software drivers, etc.).*

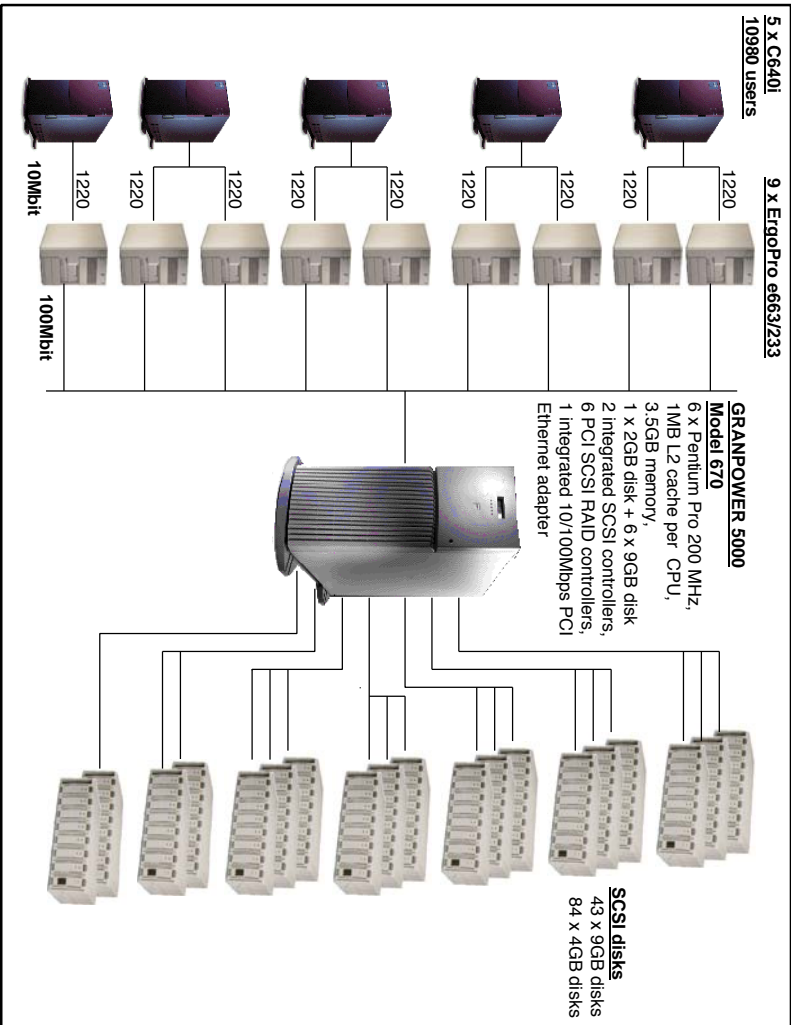


Figure 1.4.1 Benchmarked Configuration

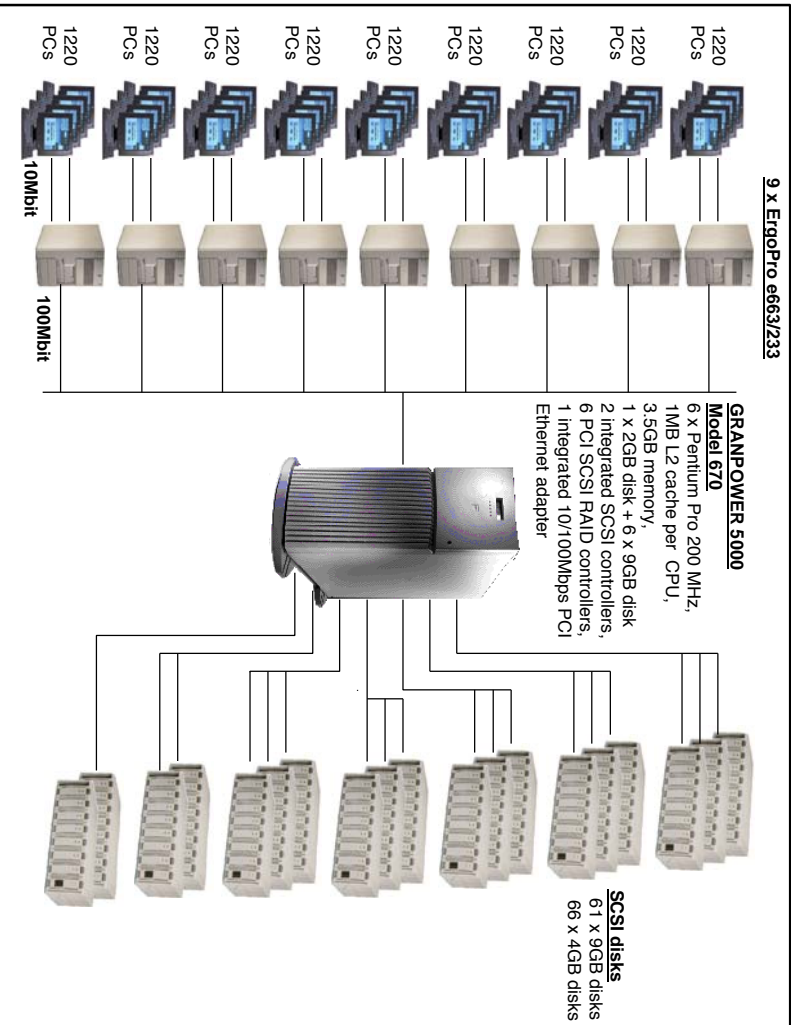


Figure 1.4.2 Priced Configuration

2 CLAUSE 1 Logical Database Design

2.1 Data and Space Definition

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

Appendix B contains the table definition code and the code used to define, create and populate the database. This information contains details of the indexing used on the database tables.

2.2 Physical Organisation of Data

The physical organisation of tables and indices, within the database, must be disclosed.

The disks space was allocated to SQL Server according to the data in Table 5.3. The SQL definitions are in Appendix B

2.3 Insert and/or Delete Operations

It must be ascertained that insert and/or delete operations to any of the tables can occur concurrently with the TPC-C transaction mix. Furthermore, any restriction in the SUT database implementation that precludes inserts beyond the limits defined in Clause 1.4.11 must be disclosed. This includes the maximum number of rows that can be inserted and the maximum key value for these new rows.

The database was set up and accessed using general purpose SQL statements and the TPC-C transaction mix can be run concurrently with other updates/deletes. There are no restrictions on insert and delete operations to any tables except where inserting a row would invalidate the unique indexing conditions of that table.

2.4 Partitioning

While there are a few restrictions placed upon horizontal or vertical partitioning of tables and rows in the TPC-C benchmark (see Clause 1.6), any such partitioning must be disclosed. Using the CUSTOMER table as an example, such partitioning could be denoted as:

This implementation does not use application level partitioning of data.

2.5 Table Replication

Replication of tables, if used, must be disclosed (see Clause 1.4.6).

Not applicable to this configuration.

2.6 Table Attributes

Additional and/or duplicated attributes in any table must be disclosed along with a statement on the impact on performance (see Clause 1.4.7).

Not applicable to this configuration.

3 CLAUSE 2 Transaction and Terminal Profile Related Items

3.1 Random Number Verification Method

The method of verification for the random number generation must be described.

3.2 Terminal Screen Layouts

The actual layouts of the terminal input/output screens must be disclosed.

The screen layouts are based on those in Clauses 2.4.3, 2.5.3, 2.6.3, 2.7.3, and 2.8.3 of the TPC Benchmark C standard specification. There are some minor differences due to the Web client implementation.

3.3 Priced Terminal Feature Verification

The method used to verify that the emulated terminals provide all the features described in Clause 2.2.2.4 must be explained. Although not specifically priced, the type and model of the terminals used for the demonstration in 8.1.3.3 must be disclosed and commercially available (including supporting software and maintenance).

The terminal attributes were verified manually exercising each specification on a representative Fujitsu *ErgoPro* d66fd with Microsoft Windows 3.11, Netscape 2.0 and also with a Fujitsu *ErgoPro* e663/233 with Microsoft Windows NT Server 4.0 Enterprise Edition and Microsoft Internet Explorer 3.0.

3.4 Presentation Manager or Intelligent Terminal

Any usage of presentation managers or intelligent terminals must be explained.

Application code running on the client machines implemented the TPC-C user interface. No presentation manager software or intelligent terminal features were used. The source code for the Web application is listed in Appendix A.

3.5 Transaction Statistics

The percentage of home and remote order-lines in the New-Order transaction must be disclosed.

The percentage of New-Order transaction that were rolled back as a result of an unused item number must be disclosed.

The number of items per order entered by New-Order transaction must be disclosed.

The percentage of home and remote payment transaction must be disclosed.

The percentage of payment and Order-status transaction that used non-primary key (C_LAST) access to the database must be disclosed.

The percentage of Delivery transaction that were skipped as a result of an insufficient number of rows in the NEW-ORDER table must be disclosed.

The mix (i.e percentages) of transaction types seen by the SUT must be disclosed.

The following table lists the data required by Clauses 8.1.3.5 to 8.1.3.11:

Statistic	Value
New Order Home Warehouse	99.03%
New Order Remote Warehouse	0.97%
New Order Rollback Transactions	0.91%
New Order OrderLine Count	9.98
Payment Home Warehouse	85.11%
Payment Remote Warehouse	14.89%
Payment Customer Name Access	59.77%
OrderStatus Customer Name Access	61.66%
Skipped Delivery Transactions	None
Transaction Mix - New Order	44.60%
Transaction Mix - Payment	43.17%
Transaction Mix - Order Status	4.11%
Transaction Mix – Delivery	4.05%
Transaction Mix - Stock Level	4.01%

Table 3.5.1 Transaction Statistics

3.6 Queuing Mechanism of Delivery

The queuing mechanism used to defer execution of the Delivery transaction must be disclosed.

Tuxedo provides the queuing mechanism for delivery servers. The client application process posts delivery transactions to the delivery queue using an asynchronous call with the TPNReply option. The client then posts a delivery queued response to the user. The delivery servers independently take deliveries off the queue and submit transactions against the database and record the results in a file. The source code for the delivery process is included in Appendix A.

4 CLAUSE 3 Transaction and System Properties

4.1 Transaction System Properties (ACID)

The result of the ACID test must be disclosed along with a description of how the ACID requirements were met. This includes disclosing which case was followed for the execution of Isolation Test 7.

4.2 Atomicity Requirements

The system under test must guarantee that database transactions are atomic; the system will either perform all individual operations on the data, or will assure that no partial-completed operations leave any effects on the data.

Completed Transaction

Perform the payment transaction for a randomly selected warehouse, district, and customer (by customer number) and verify that the records in the CUSTOMER, DISTRICT, and WAREHOUSE tables have not been changed appropriately.

A row was selected in a script from the warehouse, district and customer tables and the balances noted. A payment transaction was started with the same warehouse, district and customer identifiers and a known amount. The payment transaction was committed and the rows were verified to contain correctly updated balances.

Aborted Transaction

Perform the payment transaction for a randomly selected warehouse, district, and customer (by customer number) and substitute a ROLLBACK of the transaction for the COMMIT of the transaction. Verify that the records in the CUSTOMER, DISTRICT, and WAREHOUSE tables have NOT been changed.

A row was selected in a script from the warehouse, district and customer tables and the balances noted. A payment transaction was started with the same warehouse, district and customer identifiers and a known amount. The payment transaction was rolled back and the rows were verified to contain the original balances.

4.2 Consistency Requirements

Consistency is the property of the application that requires any execution of a database transaction to take the database from one consistent state to another, assuming that the database is initially in a consistent state.

Consistency conditions one through four were tested using a shell script to issue queries to the database. The results of the queries verified that the database was consistent for all four tests.

A run was executed under full load lasting over 120 minutes and included 4 checkpoints.

The shell script was executed again and the result of the same queries verified that the database remained consistent after the run.

4.3 Isolation

Sufficient conditions must be enabled at either the system or application level to ensure the required isolation defined above (clause 3.4.1) is obtained.

Isolation tests one through seven were executed using shell scripts to issue queries to the database. Each script included time stamps to demonstrate the concurrency of operations. The results of the queries were captured to files. The captured files were verified by the auditor to demonstrate the required isolation had been met.

4.4 Durability Requirements

The tested system must guarantee durability: the ability to preserve the effects of committed transaction and insure database consistency after recovery from any one of the failures listed in Clause 3.5.3.

Durable Media Failure

Loss of data and loss of log tests were combined and performed on a 10 warehouse database.

Loss of Log and Loss of Data

To demonstrate recovery from a permanent failure of durable medium containing TPC-C tables, the following steps were executed:

1. The database was backed up to extra disks.
2. The total number of New Orders was determined by the sum of D_NEXT_O_ID of all rows in the DISTRICT table giving the beginning count.
3. The RTE was started with 100 users.
4. The test was allowed to run for a minimum of 5 minutes.
5. One of the log disks was removed.
6. An NT error message was displayed as the log disks are mirrored by NT. The benchmark run was not affected.
7. After another 5 minutes of running in a steady state one of the data disks was removed.
8. SQL Server recorded errors in NT and Application event log due to the missing disk.
9. The RTE systems were stopped and SQL Server was shut down.
10. SQL Server was restarted and marked the database as "suspect".
11. A dump of the transaction log was taken.
12. The database was dropped, SQL Server and the SUT were shut down.
13. A new data disk was inserted into the drive cabinet, powered back up and the SUT restarted.
14. The Mylex stripe set (of 2 disks) of which this disk was a member was reinitialised using a Mylex utility.
15. The SUT was restarted and the NT logical drives effected was recreated.
16. The database was dropped and recreated as an empty database.
17. The database was restored from the database backup, followed by the restore of the transaction log.
18. Consistency condition #3 was executed and verified.
19. Step 2 was repeated and the different between the first and second counts was noted.
20. An RTE report was generated for the entire run time giving the number of NEW-ORDERS successfully returned to the RTE.
21. The counts in step 19 and 20 were compared and the results verified that all committed transactions had been successfully recovered.
22. Samples were taken from the RTE files and used to query the database to demonstrate successful transactions had corresponding rows in the ORDER table.

Instantaneous Interruption and Loss of Memory

Because loss of power erases the contents of memory, the instantaneous interruption and the loss of memory tests were combined into a single test. This test was executed under a full load of 10980 users. The following steps were executed:

1. The total number of New Orders was determined by the sum of D_NEXT_O_ID of all rows in the DISTRICT table giving the beginning count.
2. The benchmark was executed at full load of 10980 users for a minimum of 10 minutes.

3. System crash and loss of memory were induced switching off the SUT. No battery backup or Uninterruptible Power Supply (UPS) were used to preserve the contents of memory.
4. The RTE and client systems were stopped.
5. Power was restored and the system restarted.
6. SQL Server was restarted and performed an automatic recovery.
7. Consistency condition #3 was executed and verified.
8. The total number of New Orders was determined and the difference between the first and the second counts was noted.
9. An RTE report was generated for the entire run time giving the number of New Orders successfully returned to the RTE.
10. The two New Order counts were compared and the results verified that all committed transactions had been successfully recovered.
11. Samples were taken from the RTE files and used to query the database to demonstrate successful transactions had corresponding rows in the ORDER table.

5 CLAUSE 4 Scaling and Database Population

5.1 Initial Cardinality of Tables

The cardinality (e.g., the number of rows) of each table, as it existed at the start of the benchmark run (see Clause 4.2), must be disclosed. If the database was over-scaled and inactive rows of the WAREHOUSE table were deleted (see Clause 4.2.2), the cardinality of the WAREHOUSE table as initially configured and the rows deleted must be disclosed.

The database was scaled for 1110 warehouses. 12 rows were deleted from the warehouse table prior to measured runs.

Table	Number of rows
Warehouse	1,110
District	11,100
Customer	33,300,000
History	33,300,000
Order	33,300,000
New Order	9,990,000
Order Line	332,999,230
Stock	111,000,000
Item	100,000

Table 5.1.1 Initial Cardinality of Tables

5.2 Data Layout

The distribution of tables and logs must be explicitly depicted for the tested and priced system.

Tables 5.2.1 and 5.3.1 show the distribution of the database over 123 disks and the transaction log over 5 mirrored pairs of disks. In addition there was 1 disk containing Windows NT Server 4.0 Enterprise Edition and SQL Server 6.5 Enterprise Edition. The database backup used an extra 20 disks (not included in the priced configuration). Eighteen (18) 9GB disks were exchanged for Eighteen (18) 4GB disks to meet the 180 day space requirements.

The Allocation of database devices to NT drive letters is contained in Appendix B.

RAID Disk Configuration for TPCC

Adapter	Channel 1			Channel 2			Channel 3		
	#	Capacity	RAID	#	Capacity	RAID	#	Capacity	RAID
3	0	9GB		0	9GB		0	9GB	
	1	9GB		1	9GB		1	9GB	
	2	9GB		2	9GB		2	9GB	
	3	9GB		3	9GB		3	9GB	
	4	9GB		4	9GB		4	9GB	
	5	9GB		5	9GB		5	9GB	
4	0	4GB		0	4GB		0	4GB	
	1	4GB		1	4GB		1	4GB	
	2	4GB		2	4GB		2	4GB	
	3	4GB		3	4GB		3	4GB	
	4	4GB		4	4GB		4	4GB	
	5	4GB		5	4GB		5	4GB	
5	0	4GB		0	4GB		0	4GB	
	1	4GB		1	4GB		1	4GB	
	2	4GB		2	4GB		2	4GB	
	3	4GB		3	4GB		3	4GB	
	4	4GB		4	4GB		4	4GB	
	5	4GB		5	4GB		5	4GB	
6	0	4GB		0	4GB		0	4GB	
	1	4GB		1	4GB		1	4GB	
	2	4GB		2	4GB		2	4GB	
	3	4GB		3	4GB		3	4GB	
	4	4GB		4	4GB		4	4GB	
	5	4GB		5	4GB		5	4GB	
7	0	4GB		0	4GB		0	4GB	
	1	4GB		1	4GB		1	4GB	
	2	4GB		2	4GB		2	4GB	
	3	4GB		3	4GB		3	4GB	
	4	4GB		4	4GB		4	4GB	
	5	4GB		5	4GB		5	4GB	
8	0	9GB		0	9GB		0	9GB	
	1	9GB		1	9GB		1	9GB	
	2	9GB		2	9GB		2	9GB	
	3	9GB		3	9GB		3	9GB	
	4	9GB		4	9GB		4	9GB	
	5	9GB		5	9GB		5	9GB	
#	6	4GB		6	4GB		6	4GB	
	7	4GB		7	4GB		7	4GB	
	8	4GB		8	4GB		8	4GB	
	9	4GB		9	4GB		9	4GB	
	10	4GB		10	4GB		10	4GB	
	11	4GB		11	4GB		11	9GB	
	12	4GB		12	4GB		12	9GB	
	13	4GB		13	4GB		13	9GB	

Tabela 5.2.1 RAID Disk Configuration for Benchmarkload System

Disk Administrator Configuration							
Disk	Partition 1	Partition 2	Partition 3	Unused Capacity	HA#	LD#	Usage
0	C: NTFS 2047MB					Adaptec 1	system
1,4	V: (raw) 8699MB			(none)		Adaptec 2	log
2,5	Q: (raw) 8699MB			(none)		Adaptec 2	log
3,6	X: (raw) 8699MB			(none)		Adaptec 2	log
7,8	Y: (raw) 8699MB			(none)		Adaptec 2	log
9,10	W: (raw) 8699MB			(none)		Adaptec 2	log
11	J: 182322MB	P: 7040MB				3	1
12	E: 87129MB	K: 13500MB				4	1
13	H: 87129MB	L: 1004MB	R: 5005MB			5	1
14	I: 87129MB	M: 1004MB	S: 5005MB			6	1
15	F: 87129MB	N: 1004MB	T: 5005MB			7	1
16	G: 156276MB	O: 1004MB	U: 10503MB			8	1
17	Z: 74682MB					8	2
18	Z: 17364MB					8	3

Table 5.3.1 Disk Administrator Configuration

Refer to Appendix G 180 Day Space Calculations for specific details of the Priced configuration. The data distribution is identical to the Benchmarked system.

5.3 Database Model

A statement must be provided that describes:

1. *The data model implemented by the DBMS used (e.g., relational, Codasy!, flat file etc.).*
2. *The database interface (e.g., embedded, call level) and access language (e.g., SQL, DL/1, COBAL, read/write) used to implement the TPC-C transaction. If more than one interface /access language is used to implement TPC-C, each interface/access language must be described and a list of which interface/access language is used with which transaction type must be disclosed.*

Microsoft SQL Server 6.5 Enterprise Edition is a Relational Database Management System. Only general purpose methods were used to implement the transactions.

5.4 Database Mapping

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

The database was neither partitioned or replicated.

5.5 180 Day Space

Details of the 180 day space computations along with proof that the database is configured to sustain 8 hours of growth for the dynamic tables (Order, Order-Line, and History) must be disclosed (see Clause 4.2.3).

To calculate the space required to sustain the database log for 8 hours of growth of steady state the following steps were followed:

1. The free space on the logfile was queried using **dbcc checktable(syslogs)**
2. Transactions were run against the database with a full load of users
3. The free space was again queried using **dbcc checktable(syslogs)**
4. The space used was calculated as the difference between the two queries
5. The number of new order transactions was verified from the RTE report covering the whole run (including ramp up and ramp down)

The space used was divided by the number of new order transactions was multiplied by the tpmC rate) and then 480 minutes giving space required to sustain performance for eight hours.

Appendix G shows the space requirements for 180 day space as well as the logical log space for eight hours.

6 CLAUSE 5 Performance Metrics and Response Times

6.1 Measured tpmC

The Measured tpmC must be reported.

The measured tpmC was 13,391.13 tpmC. This figure was the result of 401,734 New Order transactions being serviced by the SUT within a 30 minute period.

6.2 Response Times

Ninetieth percentile, maximum and average response times must be reported for all transaction types as well as for the Menu response times.

The table below shows response times for all the transaction types. The approved timing delays of 0.1 seconds for the Web Browser latency have been included in the response and menu times.

Type	Average	Maximum	90th Percentile
New Order	0.74	74.74	1.11
Payment	0.42	16.44	0.62
Order-Status	1.20	7.31	1.81
Interactive Delivery	0.29	1.29	0.31
Deferred Delivery	0.86	6329	1.29
Stock-Level	3.95	77.71	6.82
Menu	0.20	1.22	0.30

Table 6.2.1 Response Times

6.3 Keying and Think Times

The minimum, the average and the maximum keying and think times must be reported for each transaction type.

Type	Minimum	Average	Maximum
New Order	18.01	18.01	18.07
Payment	3.00	3.01	3.07
Order-Status	2.00	2.01	2.07
Interactive Delivery	2.00	2.01	2.05
Stock-Level	2.00	2.01	2.07

Table 6.3.1 Keying Times

	Minimum	Average	Maximum
New-Order	0.00	12.10	121.00
Payment	0.00	12.04	121.01
Order-Status	0.00	10.16	83.84
Delivery	0.00	5.10	40.47
Stock-Level	0.00	5.12	42.37

Table 6.3.2 Think Times

6.4 Response Time Frequency Distributions

Response Time frequency distribution curves (see Clause 5.6.1) must be reported for each transaction type.

Figure 6.4.1 New Order Response Time Distribution

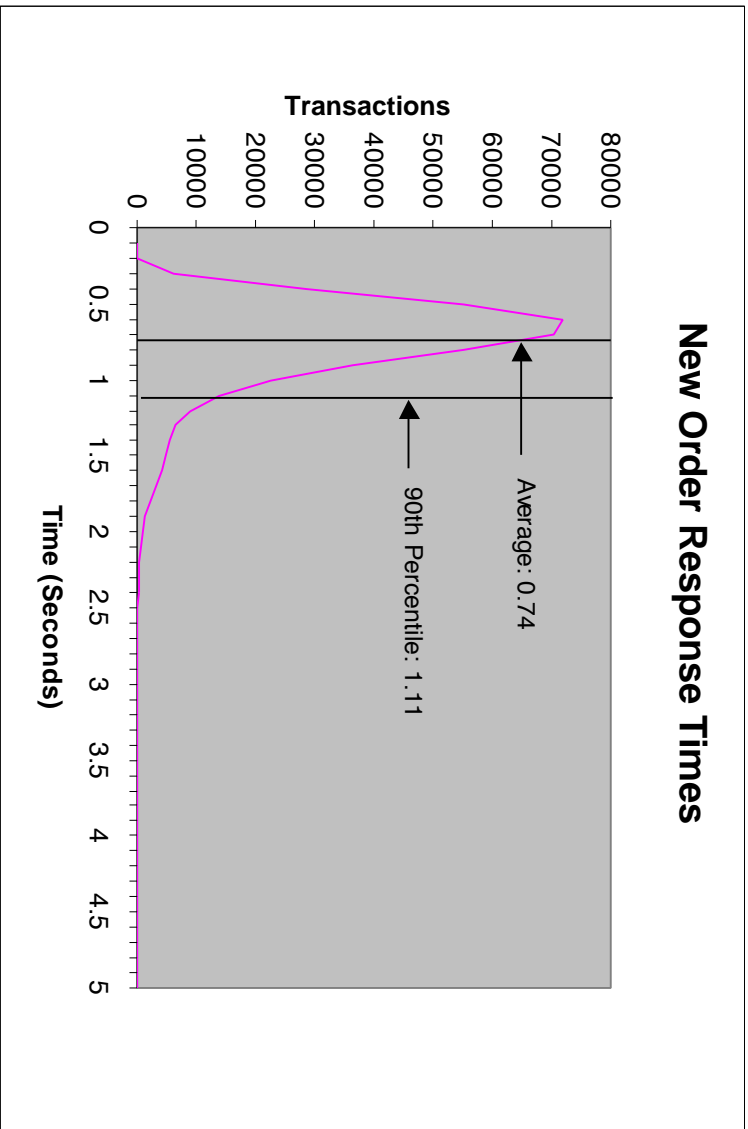


Figure 6.4.2 Payment Response Time Distribution

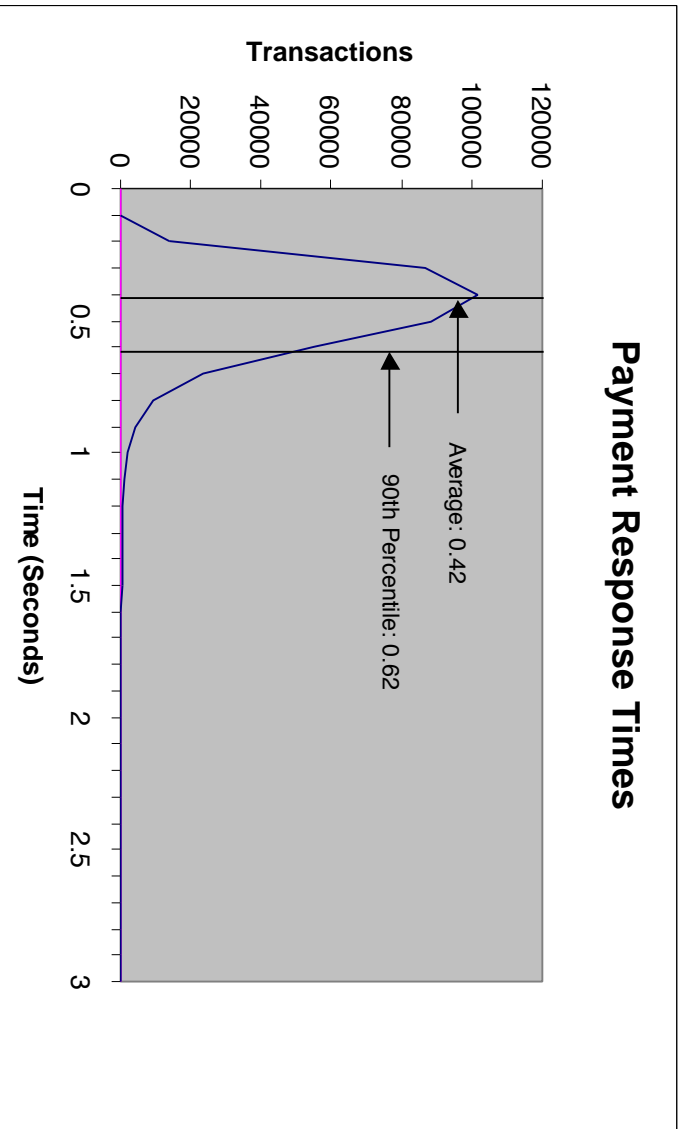


Figure 6.4.3 Order Status Response Time Distribution

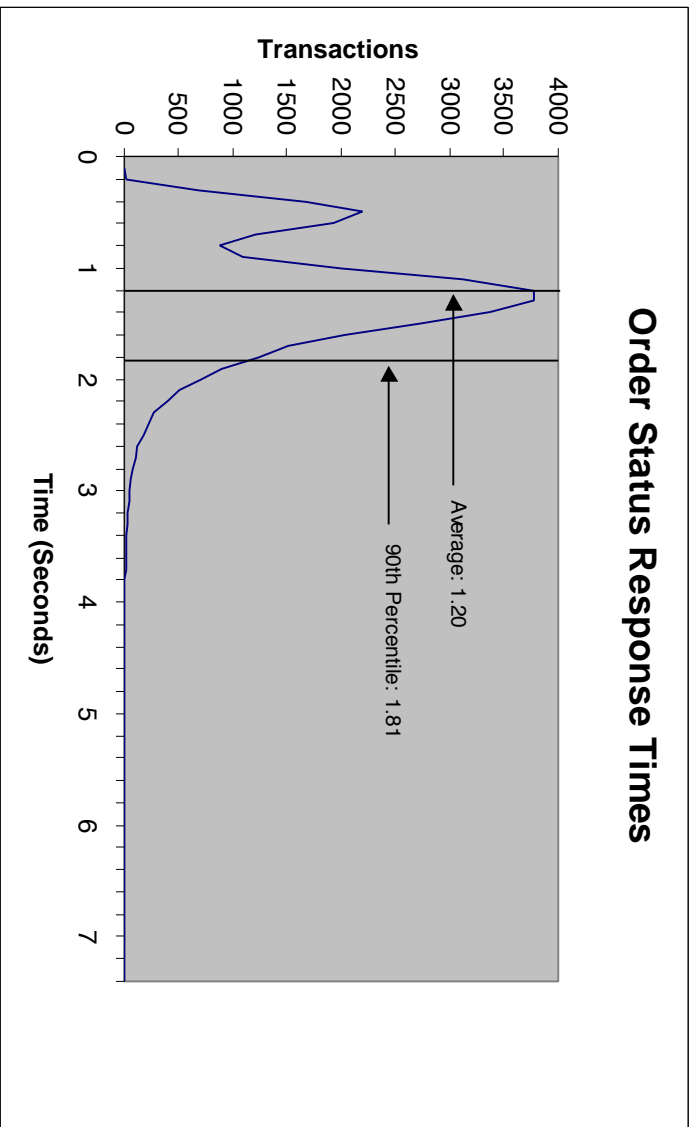


Figure 6.4.4 Delivery Response Time Distribution

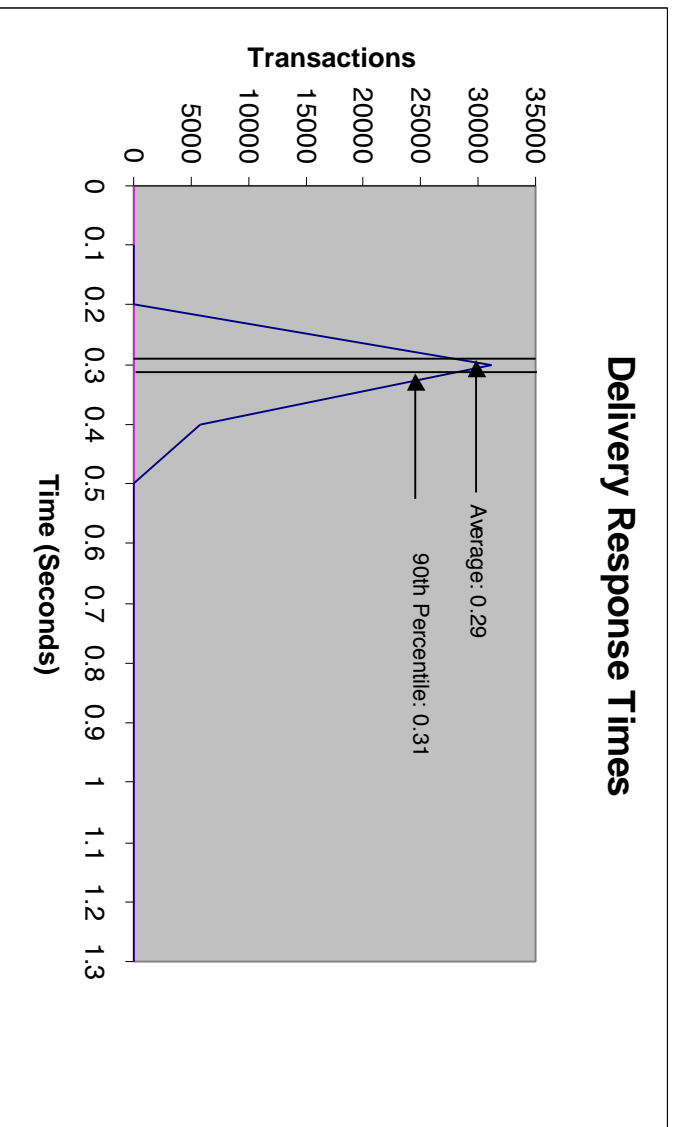
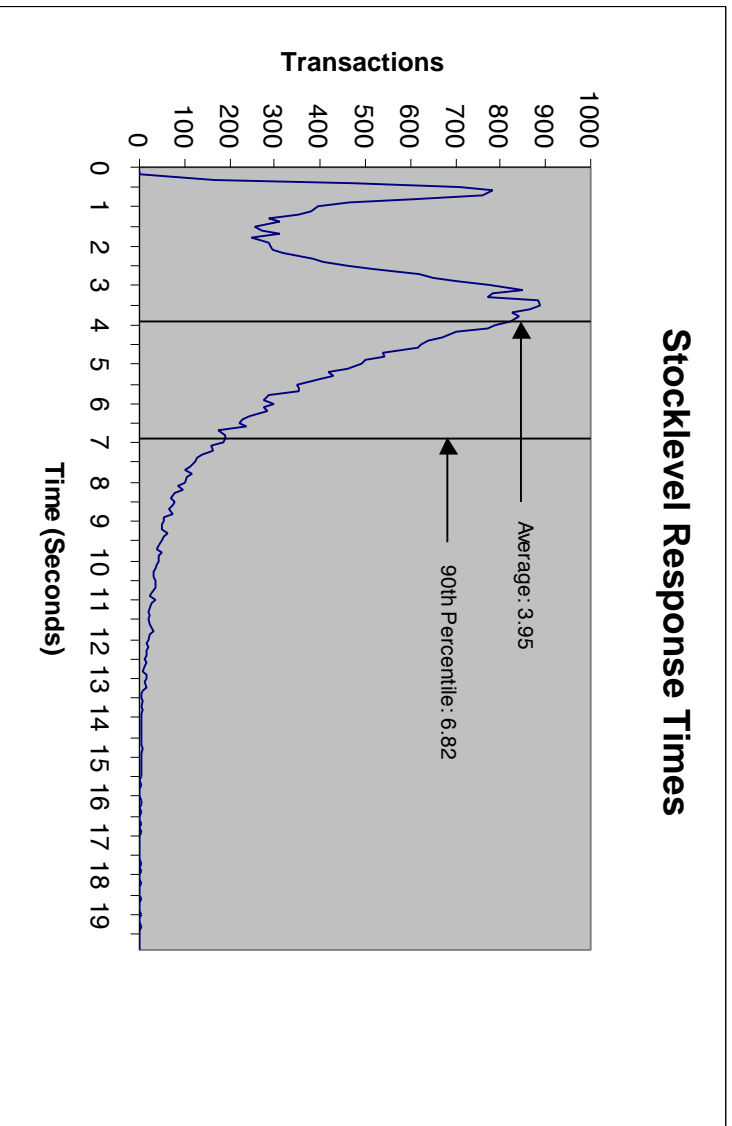


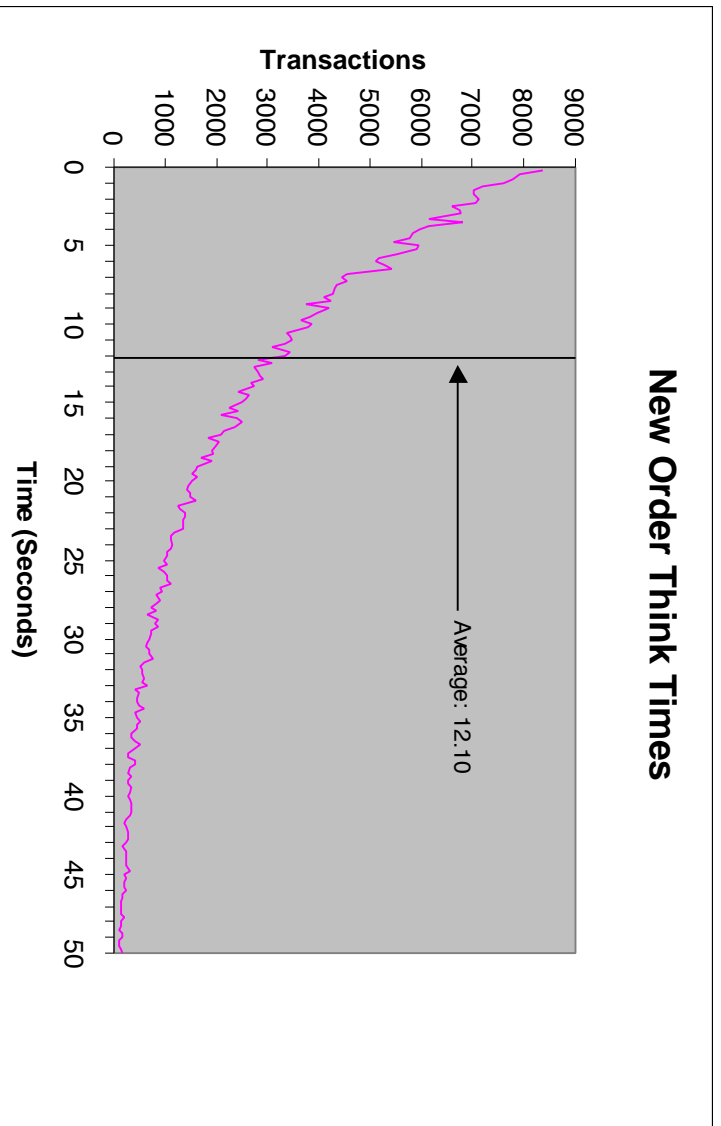
Figure 6.4.5 Stock Level Response Time Distribution



6.5 Think Time Frequency Distributions

ThinkTime frequency distribution curves (see Clause 5.6.3) must be reported for the New-Order transaction.

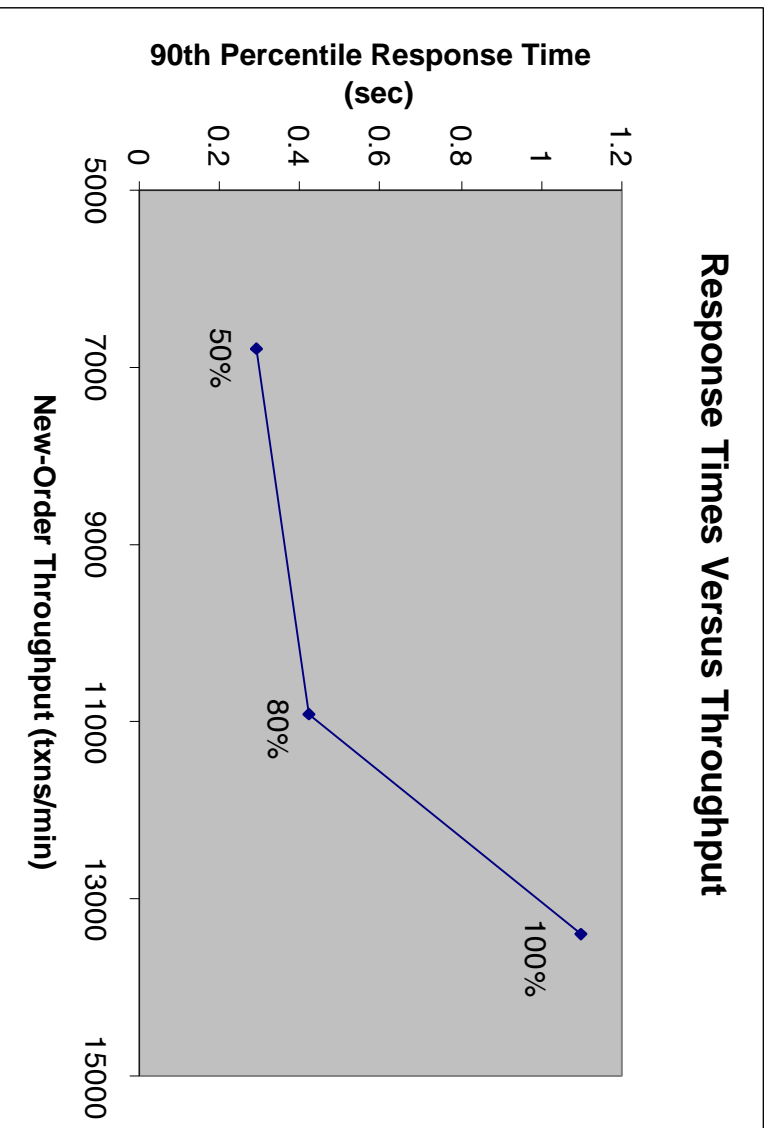
Figure 6.5.1 New Order Think Time Distribution



6.6 Performance Curve for Response Time vs. Throughput

The performance curve for response times versus throughput (see Clause 5.6.2) must be reported for the New-Order transaction.

Figure 6.6.1 Response Times versus Throughput

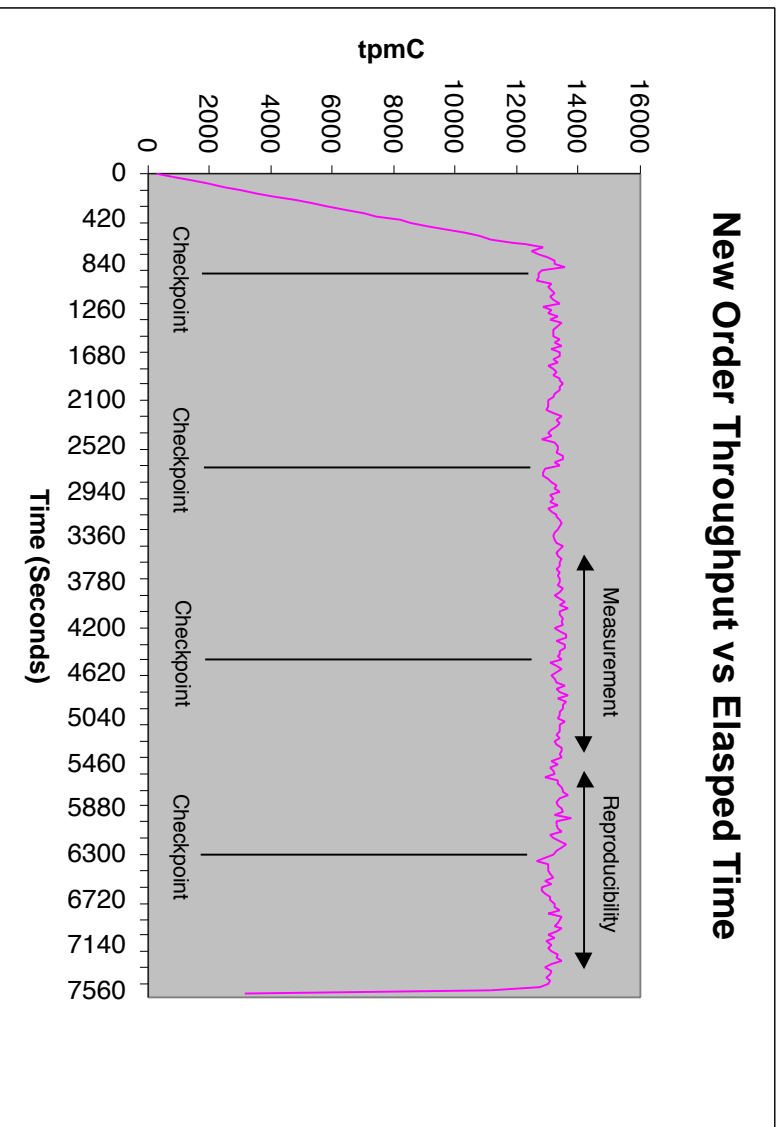


The relevant timing delays for the Web Browser latency have been included in the response times.

6.7 Throughput vs. Elapsed Time: New-Order Transaction

A graph of throughput versus elapsed time (see Clause 5.6.5) must be reported for the New-Order transaction

Figure 6.7.1 New Order Throughput vs Elapsed Time



6.8 Steady State Method

The method used to determine that the SUT had reached a steady state prior to commencing the measurement interval (see Clause 5.5) must be described.

Steady state can be seen on the graph in section 6.8. Both the client machines and the SUT were inspected to ensure that the operating system resource usage was constant.

6.9 Work Performed During Steady State

A description of how the work normally performed during a sustained test (for example checkpointing, writing redo/undo log records, etc.), actually occurred during the measurement interval must be reported.

The RTE selects a transaction type from the menu. A time stamp is logged before this request is sent and when the response comes back. The difference between these timestamps is logged as the menu response time. The RTE then generates the appropriate data to fill in the form and waits for the key time for that transaction before sending the data. A timestamp is taken before the data is sent and after the response is received. The difference between these timestamps is logged as the response time. A delay of 0.1 seconds is included in each of these response times to reflect the delay associated with a web browser. The required transaction data and response times are logged to a file on each RTE driver machine and the RTEs wait for the required think time before submitting the next transaction.

The contents of the RTE log files are consolidated during the reporting phase.

The RTE emulates web browsers. It sends and receives HTTP data via an Ethernet LAN. The RTE systems communicate with an application running under Microsoft IIS in the client machines. This application passes requests to a Tuxedo TPC-C server queue and waits for the request to be processed (except in the case of deliveries) before responding with a formatted HTML page to the RTE. The Tuxedo TPC-C application takes requests from the Tuxedo queues and processes the request by running a stored procedure on the SUT using DBLIB over another Ethernet LAN. SQL Server sends responses back to the client application via Tuxedo.

Delivery transactions are deferred and the client application does not wait for the Tuxedo application to respond. The delivery is processed later and the information required is logged to a file on each of the client machines for processing in the reporting phase.

A script was written to perform checkpoints at specific intervals, the Microsoft SQL Server parameter **recovery interval** was set to its maximum allowable value. By setting the trace flag **3502** SQL Server logged the checkpoint beginning and end times to the Microsoft SQL Server log file **ERRORLOG**. The script included a delay between each script initiated checkpoint of 30 minutes which is equal to the measurement interval. The positioning of the checkpoint was verified to be clear of the guard zones.

At each checkpoint Microsoft SQL Server wrote to disk all memory pages that had been updated but not yet physically written to disk..

6.10 Reproducibility Method

A description of the method used to determine reproducibility of the measurement results must be reported.

The reproducibility measurement was part of the same run that gave the reported result. The two intervals did not overlap. The reproducibility run gave a throughput of 13,208.10 tpmC which is within 2% of the reported tpmC.

6.11 Duration of Measurement

A statement of the duration of the measurement interval for the reported Maximum Qualified Throughput (tpmC) must be included.

The measurement period was 30 minutes. Two checkpoints occurred before this period and another started approximately 16 minutes into it.

6.12 Regulation of Transaction Mix

The method of regulation of the transaction mix (e.g. card decks or weighted random) must be described. If weighted distribution is used and the RTE adjusts the weights associated with each transaction type, the maximum adjustments to the weight from the initial value must be disclosed.

The RTE was given a weighted random distribution which was not altered during the run.

6.13 Transaction Statistics

The percentage of the total mix for each transaction type must be disclosed.

The percentage of the New-Order transaction rolled back as a result of invalid item number must be disclosed.

The average number of order-lines entered per New-Order transaction must be disclosed.

The percentage of remote order-lines entered per New-Order transaction must be disclosed.

The percentage of remote payment transaction must be disclosed.

The percentage of customer selections by customer last name in the Payment and Order-Status transaction must be disclosed.

The percentage of Delivery transaction skipped due to there being fewer than necessary orders in the New-Order table must be disclosed.

Statistic	Value
New Order Home Warehouse	99.03%
New Order Remote Warehouse	0.97%
New Order Rollback Transactions	0.91%
New Order OrderLine Count	9.98
Payment Home Warehouse	85.11%
Payment Remote Warehouse	14.89%
Payment Customer Name Access	59.77%
OrderStatus Customer Name Access	61.66%
Skipped Delivery Transactions	None
Transaction Mix - New Order	44.60%
Transaction Mix - Payment	43.17%
Transaction Mix - Order Status	4.11%
Transaction Mix – Delivery	4.05%
Transaction Mix - Stock Level	4.01%

Table 6.13.1 Transaction Statistics

6.14 Checkpoint Statistics

The number of checkpoints in the measurement interval, the time in seconds from the start of the measurement interval to the first checkpoint, and the Checkpoint Interval must be disclosed.

The was one checkpoint in the measurement interval. This started 954 seconds into the interval.

The checkpoint interval (time from the start of one to the start of the next) is 30 minutes. The duration of a checkpoint is approximately 6 minutes. It has been verified that there is no checkpoint within 7.5 minutes of the start or the end of the measurement interval.

7 CLAUSE 6 SUT, Driver and Communication Definition

7.1 RTE inputs

The RTE input parameters, code fragments, functions, etc. used to generate each transaction input field must be disclosed.

The RTE used in this benchmark was Benchmarkcraft which is proprietary to Microsoft. Appendix C contains all the RTE configuration information.

7.2 Functions and Performance of Emulated Components

It must be demonstrated that the functions and performance of the components being emulated in the Driver System are equivalent to that of the priced system.

The RTE driver systems consisted of five (5) Fujitsu *teasmserver C640b*. These driver machines were connected to the nine (9) Fujitsu *ErgoPro e663J233* through five (5) Ethernet LANs.

The only difference between the benchmarked and price configuration are the addition of one (1) 10Mbit/sec LAN adapter in the client machines to support the number of connections and the lack of a latency delay on the emulated workstations. The Web browser delay was emulated by including a 0.1 second Web browser delay in the RTE.

7.3 Functional Diagrams

A complete functional diagram of both the benchmark configuration and the configuration of the proposed (target) system must be disclosed. A detail list of all software and hardware functionality being performed on the Driver System, and its interface to the SUT must be disclosed (see Clause 6.6.3.6).

Section 1.4 describes and shows functional diagrams for the benchmarked and price systems.

7.4 Network Bandwidth and Configuration

The network configuration of both tested and proposed (target) services which are being represented and a thorough explanation of exactly which parts are being replaced with the Driver system must be disclosed.

The tested system consisted of seven (7) Ethernet LANs, five (5) linking the RTE systems to the client systems, one (1) linking the RTE systems together and one (1) connecting the nine (9) client machines to the server.

The priced configuration consists of nineteen (19) Ethernet LANs. The 10980 physical workstations are connected to the client machines via eighteen (18) LAN segments with 1220 physical workstations per client machine (2 LAN segments per client with 616 and 604 workstations respectively). The client machines are connected to the server via one (1) LAN segment. The 18 workstation-client LAN segments have a bandwidth of 10 Megabits per second and the client-Server LAN has a bandwidth of 100Megabits per second. From the analysis of network utilisation, we believe that the priced configuration would support 10980 physical workstations at the reported transaction rate and still have network capacity available.

7.5 Operator Intervention

If the configuration requires operator intervention (see Clause 6.6.6) mechanism and the frequency of this intervention must be disclosed.

No operator intervention was required.

8 CLAUSE 7 Pricing

8.1 System Pricing

A detailed list of hardware and software used in the priced system must be reported. Each separately orderable item must have vendor part number, description, and release/revision level, and either general availability status or committed delivery date. If package-pricing is used, vendor part number of the package and a description uniquely identifying each of the components of the package must be disclosed. Pricing source(s) and effective date(s) must also be reported.

The total 5-year price of the entire configuration must be reported, including: hardware, software, and maintenance charges. Separate component pricing is recommended. The basis of all discounts used must be disclosed.

System pricing should include subtotals for the following components: server Hardware, server Software, Client Hardware, Client Software, and Network components used for terminal connection (see Clause 7.2.2.3). Clause 6.1 describes the Server and Client components.

System pricing must include line item indication where non-sponsoring companies' brands are used. System pricing must also include line item indication of third party pricing.

The detailed list of hardware and software used in the priced system is included in the Executive Summary at the beginning of this report. The recommended list prices for the Fujitsu branded items are taken from the Fujitsu PC Pricebook for North America valid from 2nd February 1998 and the Servers Pricebook for North America valid from 5th February 1998 and are available in the USA via numerous Fujitsu channel partners including Lexia Inc.

The GRANPOWER 5000 Model 670 (GP5000) contains :-

- 6 x 200MHz Pentium Pro Processor
- 1MB second level cache per processor
- 2 x integrated Adaptec 7880 single-ended Ultra SCSI controllers
- 1 x integrated SVGA controller
- 1 x integrated 10/100Mbps PCI Ethernet adapter
- 3 x 5.25" Exchangeable bays (1 occupied)
- 9 x hot pull disk bays
- 8 x DIMM slots expandable to 16 with extra board
- 6 x dedicated PCI adapter card slots
- 4 x dedicated EISA adapter card slots
- 2 x asynchronous comms ports
- 1 x parallel port
- 1 x 1.44MB 3.5" floppy disk drive
- 1 x Eight speed SCSI-2 CD-ROM drive

The maintenance pricing for the Fujitsu branded items has been supplied by Northern Telecom who are the recommended maintenance supplier for Fujitsu branded items supplied by Lexia Inc. The Northern Telecom maintenance discounts are based on a volume discount and 5 year prepayment discount which, in this case, equates to a total of 15% discount.

The maintenance for the third party items has been priced as follows:

- The Compex hubs have a lifetime warranty and an additional 10% (or 2) spares have also been priced
- All Microsoft maintenance is covered by the 5 year maintenance priced for Microsoft SQL Server.
- Mylex DAC960PJ controllers have a 3 year warranty the additional 2 years costs \$50. An additional 2 spares have also been priced.

8.2 Availability, Throughput and Price/Performance

The committed delivery date for general availability (availability date) of products used in the price calculations must be reported. When the priced system includes products with different availability dates, the reported availability date for the priced system must be the date at which all components are committed to be available.

A statement of the measured tpmC, as well as the respective calculations for 5-year pricing, price/performance (price/tpmC), and the availability date must be included.

All the items used in the priced system are currently available with the exception of:

9GB 10,000RPM 1.6" SCA DISK (SV565050) which will be available from Fujitsu by 31st March 1998.

Mylex DAC960PJ 3 channel RAID controller which will be available from Mylex by 27st February 1998.

The Fujitsu GRANPOWER 5000 Model 670, with Microsoft NT Server 4.0 Enterprise Edition and SQL Server 6.5 Enterprise Edition, achieved 13,391.13 tpmC @ \$37.62 per tpmC.

8.3 Country Specific Pricing

Additional Clause 7 related items may be included in the Full Disclosure Report for each country specific priced configuration. Country specific pricing is subject to Clause 7.1.7.

The system is priced for the USA.

8.4 Usage Pricing

For any usage pricing, the sponsor must disclose:

- *Usage level at which the component was priced.*
- *A statement of the company policy allowing such pricing.*

The component pricing based on usage is shown below:

- 1 Microsoft Windows NT Server 4.0 Enterprise Edition licence
- 1 Microsoft SQL Server 6.5 Enterprise Edition licence
- 9 Microsoft Windows NT Server 4.0 Licences
- 1 Microsoft SQL Server Programmers Toolkit
- 1 Microsoft Visual C++ Subscription
- 9 BEA Tuxedo 6.3 CFS for NT licences.

Microsoft SQL Server and Internet Information Server and BEA Tuxedo were priced for an unlimited number of users.

9 CLAUSE 9 Audit

An independent audit of the benchmark results, by an auditor certified by the TPC, is required.

The auditor's name, address, phone number, and a copy of the audit's attestation letter indicating compliance must be included in the Full Disclosure Report.

A review of the pricing model is required to ensure that all components required are priced (see Clause 9.2.8). The auditor is not required to review the final Full Disclosure Report or the final pricing prior to issuing the attestations letter.

This benchmark was independently audited by:

Tom Sawyer
Performance Metrics Inc.
2229 Benita Dr., Suite 101
Rancho Cordova
CA 95670

Phone: (916) 635-2822
Fax: (916) 858-0109

See appendix H for the auditors attestation letter.

The addresses for obtaining copies of this report is on page 5

Appendix A Source Code

The client source code is listed below.

A.1 delirpt.c

```

/*      FILE:          DELIRPT.C
 *          Microsoft TPC-C Kit Ver. 3.00.000
 *
 *          Copyright Microsoft, 1996
 *
 *      PURPOSE:      Delivery report processing application
 *      Author:       Philip Durr
 *                   philipdu@Microsoft.com
 */

#include <windows.h>
#include <stdio.h>
#include <stdlib.h>

#define LOGFILE_READ_EOF      0
                                //check log file flag return current state
#define LOGFILE_CLEAR_EOF    1
                                //clear end of log file flag
#define LOGFILE_SET_EOF      2
                                //set flag end of log file reached

#define INTERVAL              .01
                                //90th percentile calculation bucket interval

#define ERR_SUCCESS          1000
                                //success no error
#define ERR_READING_LOGFILE  1001
                                //io errors occurred reading delivery log file
#define ERR_INSUFFICIENT_MEMORY 1002
                                //insufficient memory to process 90th percentile report
#define ERR_CANNOT_OPEN_RESULTS_FILE 1005
                                //Cannot open delivery results file delilog.

typedef struct _RPTLINE
{
    SYSTEMTIME      start;
                                //delilog report line start time
    SYSTEMTIME      end;
                                //delilog report line end time

```

```

    int             response;
                                //delilog report line time delivery took in millisecond
    int             w_id;
                                //delilog report line warehouse id for deliv
    int             o_carrier_id;
                                //delilog report line carrier id for delivery
    int             items[10];
                                //delilog report line delivery line items
} RPTLINE, *PRPTLINE;

//error message structure used in ErrorMessage API
typedef struct _SERRORMSG
{
    int             iError;
                                //error id of message
    char            szMsg[80];
                                //message to sent to browser
} SERRORMSG;

int                versionMS = 3;
                                //delirpt version
int                versionMM = 0;
int                versionLS = 2;
int                iReport;
                                //delirpt report to process
int                iStartTime;
                                //begin times to accept for report
int                iEndTime;
                                //end times to accept for report
FILE               *fpLog;
                                //log file stream

//Local function prototypes
void               main(int argc, char *argv[]);
static int         Init(void);
static void        Restore(void);
static int         DoReport(void);
int               AverageResponse(void);
int               SkippedDelivery(void);
int               Percentile90th(void);
BOOL              CheckTimes(PRPTLINE pRptLine);
static int         OpenLogFile(void);
static void        CloseLogFile(void);
static void        ResetLogFile(void);
static BOOL        LogEOF(int iOperation);
static BOOL        ReadReportLine(char *szBuffer, PRPTLINE pRptLine);
static BOOL        ParseReportLine(char *szLine, PRPTLINE pRptLine);
static BOOL        ParseDate(char *szDate, LPSYSTEMTIME pTime);
static BOOL        ParseTime(char *szTime, LPSYSTEMTIME pTime);
static void        ErrorMessage(int iError);
static BOOL        GetParameters(int argc, char *argv[]);
static void        PrintParameters(void);
static void        PrintHeader(void);

```

```

static void cls(void);
static BOOL      IsNumeric(char *ptr);

/* FUNCTION: int main(int argc, char *argv[])
*
* PURPOSE:      This function is the beginning execution point for the delivery executable.
*
* ARGUMENTS:   int          argc      number of command line arguments passed
to delivery
*              char        *argv[]   array of commandline argument
pointers
*
* RETURNS:     None
*
* COMMENTS:   None
*
*/

void main(int argc, char *argv[])
{
    int      iError;

    PrintHeader();

    if ( GetParameters(argc, argv) )
    {
        PrintParameters();
        return;
    }

    if ( (iError=Init()) != ERR_SUCCESS )
    {
        ErrorMessage(iError);
        Restore();
        return;
    }

    if ( (iError = DoReport()) != ERR_SUCCESS )
        ErrorMessage(iError);

    Restore();

    return;
}

/* FUNCTION: static int Init(void)
*

```

```

* PURPOSE:      This function initializes the delirtp application.
*
* ARGUMENTS:   None
*
* RETURNS:     None
*
* COMMENTS:   None
*
*/

static int Init(void)
{
    int iError;

    if ( (iError = OpenLogFile()) )
        return iError;

    return TRUE;
}

/* FUNCTION: static void Restore(void)
*
* PURPOSE:      This function cleans up the delirtp application before termination.
*
* ARGUMENTS:   None
*
* RETURNS:     None
*
* COMMENTS:   None
*
*/

static void Restore(void)
{
    CloseLogFile();
    return;
}

/* FUNCTION: static int DoReport(void)
*
* PURPOSE:      This function dispatches the requested report.
*
* ARGUMENTS:   None
*
* RETURNS:     ERR_SUCCESS if successfull or error code if an error occurs
*
* COMMENTS:   None
*
*/

```

```

static int DoReport(void)
{
    int iRc;

    switch(iReport)
    {
        case 1:
            iRc = AverageResponse();
            break;

        case 2:
            iRc = Percentile90th();
            break;

        case 3:
            iRc = SkippedDelivery();
            break;

        case 4:
            if ( (iRc = AverageResponse()) != ERR_SUCCESS )
                break;
            if ( (iRc = Percentile90th()) != ERR_SUCCESS )
                break;
            if ( (iRc = SkippedDelivery()) != ERR_SUCCESS )
                break;
            break;
    }
    return iRc;
}

/* FUNCTION: int AverageResponse(void)
 *
 * PURPOSE:          This function processes the AverageResponse report.
 *
 * ARGUMENTS:       None
 *
 * RETURNS:         ERR_SUCCESS if successfull or error code if an error occurs.
 *
 * COMMENTS:       None
 */

```

```

int AverageResponse(void)
{
    RPTLINE reportLine;
    int iTotalResponse;
    int iLines;
    double fAverage;
    char szDelivery[128];

```

```

ResetLogFile();

iTotalResponse = 0;
iLines = 0;
printf("\n\n***** Average Response Time Report *****\n");
while ( !LogEOF(LOGFILE_READ_EOF) )
{
    if ( ReadReportLine(szDelivery, &reportLine) )
        return ERR_READING_LOGFILE;
    if ( szDelivery[0] == '*' )
        continue;
    if ( !LogEOF(LOGFILE_READ_EOF) )
    {
        if ( CheckTimes(&reportLine) )
            continue;
        iLines++;
        iTotalResponse += reportLine.response;

        if ( iLines % 10 == 0 )
            printf("Reading Report Line:\t%d\r", iLines);
    }
    printf("                \r");
    if ( iLines == 0 )
    {
        printf("No deliveries found.\n");
    }
    else
    {
        fAverage = ((double)iTotalResponse / (double)iLines)/(double)1000;
        printf("Total Deliveries:  %10.0f\n", (float)iLines);
        printf("Total Response Times:  %10.3f\n",
            ((float)iTotalResponse/(float)1000));
        printf("Average Response Time: %10.3f\n", fAverage);
    }

    return ERR_SUCCESS;
}

/* FUNCTION: int Percentile90th(void)
 *
 * PURPOSE:          This function processes the 90th percentile report.
 *
 * ARGUMENTS:       None
 *
 * RETURNS:         ERR_SUCCESS if successfull or error code if an error occurs.
 *
 * COMMENTS:       This function requires enough space to allocate needed

```

```

*                               buckets which will be 2 * max response time in
*                               deci-seconds.
*
*/

int Percentile90th(void)
{
    RPTLINE reportLine;
    int         iBucketSize;
    int         i;
    int         iResponseSeconds;
    int         iMaxSeconds;
    int         iTotalsBuckets;
    double      iTotal;
    double      i90thPercent;
    short       *psBuckets;
    char        szDelivery[128];

    printf("\n\n***** 90th Percentile *****\n");
    printf("Calculating Max Response Seconds...\n");

    ResetLogFile();

    iMaxSeconds = -1;
    while ( !LogEOF(LOGFILE_READ_EOF) )
    {
        if ( ReadReportLine(szDelivery, &reportLine) )
            return ERR_READING_LOGFILE;
        if ( szDelivery[0] == '*' )
            continue;
        if ( !LogEOF(LOGFILE_READ_EOF) )
        {
            if ( iMaxSeconds < reportLine.response )
                iMaxSeconds = reportLine.response;
        }
    }

    iTotalsBuckets = iMaxSeconds + 1;

    printf("Allocating Buckets...\n");

    iBucketSize = iTotalsBuckets * sizeof(short);

    if ( !(psBuckets = (short *)malloc(iBucketSize)) )
        return ERR_INSUFFICIENT_MEMORY;

    ZeroMemory(psBuckets, iBucketSize);

```

```

iTotal = 0;

ResetLogFile();
printf("Calculating Distribution...\n");

while ( !LogEOF(LOGFILE_READ_EOF) )
{
    if ( ReadReportLine(szDelivery, &reportLine) )
        return ERR_READING_LOGFILE;
    if ( szDelivery[0] == '*' )
        continue;
    if ( !LogEOF(LOGFILE_READ_EOF) )
    {
        if ( CheckTimes(&reportLine) )
            continue;
        psBuckets[reportLine.response]++;
        iTotal++;
    }
}

i90thPercent = iTotal * .9;

for(i=0, iTotal = 0.0; iTotal < i90thPercent; iTotal += (double)psBuckets[i]
    i++);

printf("90th Percentile = %d.%d\n", i/1000, (i % 1000));

free(psBuckets);

return ERR_SUCCESS;
}

/* FUNCTION: int SkippedDelivery(void)
*
* PURPOSE:           This function processes the Skipped Deliveries report.
*
* ARGUMENTS:        None
*
* RETURNS:           ERR_SUCCESS if successfull or error code if an error occurs
*
* COMMENTS:         None
*
*/

int SkippedDelivery(void)
{
    RPTLINE reportLine;
    char        szDelivery[128];

```

```

int          i;
int          items[10];

ResetLogFile();

printf("\n\n***** Skipped Delivery Report *****\n");
memset(items, 0, sizeof(items));
printf("Reading Delivery Log File...");

while ( !LogEOF(LOGFILE_READ_EOF) )
{
    if ( ReadReportLine(szDelivery, &reportLine) )
        return ERR_READING_LOGFILE;
    if ( szDelivery[0] == '*' )
        continue;
    if ( !LogEOF(LOGFILE_READ_EOF) )
    {
        if ( CheckTimes(&reportLine) )
            continue;
        for(i=0; i<10; i++)
        {
            if ( !reportLine.items[i] )
                items[i]++;
        }
    }
}
printf("\n");
printf("Skipped delivery table.\n");
printf(" 1 2 3 4 5 6 7 8 9 10\n");
printf("-----\n");
for(i=0; i<10; i++)
    printf("%4.4d ", items[i]);
printf("\n");

return ERR_SUCCESS;
}

/* FUNCTION: BOOL CheckTimes(PRPTLINE pRptLine)
*
* PURPOSE:      This function checks to see if the delilog record falls within the
*               begin and end time from the command line.
*
* ARGUMENTS:   PRPTLINE      pRptLine delilog processed report line.
*
* RETURNS:     BOOL         FALSE if report line is not within the requested
*               start and end times.

```

```

*
*               TRUE if the report line is
*               within the requested
*               start and end times.
*
* COMMENTS:    If startTime and endTime are both 0 then the user requested
*               the default behavior which is all records in delilog are
*               valid.
*/

BOOL CheckTimes(PRPTLINE pRptLine)
{
    int      iRptEndTime;
    int      iRptStartTime;

    iRptStartTime = (pRptLine->start.wHour * 3600000) + (pRptLine->start.wMinute *
60000) + (pRptLine->start.wSecond * 1000) + pRptLine->start.wMilliseconds;
    iRptEndTime = (pRptLine->end.wHour * 3600000) + (pRptLine->end.wMinute *
60000) + (pRptLine->end.wSecond * 1000) + pRptLine->end.wMilliseconds;

    if ( iStartTime == 0 && iEndTime == 0 )
        return FALSE;

    if ( iStartTime <= iRptStartTime && iEndTime >= iRptEndTime )
        return FALSE;

    return TRUE;
}

/* FUNCTION: int OpenLogFile(void)
*
* PURPOSE:      This function opens the delivery log file for use.
*
* ARGUMENTS:    None
*
* RETURNS:      int      ERR_CANNOT_OPEN_RESULTS_FILE
*               Cannot create results log file.
*               ERR_SUCCESS
*               Log file successfully opened
*
* COMMENTS:     None
*/

static int OpenLogFile(void)
{
    fpLog = fopen("delilog.", "rb");

```

```

        if ( !fpLog )
            return ERR_CANNOT_OPEN_RESULTS_FILE;

        return ERR_SUCCESS;
    }

/* FUNCTION: int CloseLogFile(void)
 *
 * PURPOSE:      This function closes the delivery log file.
 *
 * ARGUMENTS:    None
 *
 * RETURNS:      None
 *
 * COMMENTS:     None
 */

static void CloseLogFile(void)
{
    if ( fpLog )
        fclose(fpLog);

    return;
}

/* FUNCTION: static void ResetLogFile(void)
 *
 * PURPOSE:      This function prepares the delilog. file for reading
 *
 * ARGUMENTS:    None
 *
 * RETURNS:      None
 *
 * COMMENTS:     None
 */

static void ResetLogFile(void)
{
    fseek(fpLog, 0L, SEEK_SET);
    LogEOF(LOGFILE_CLEAR_EOF);

    return;
}

/* FUNCTION: static BOOL LogEOF(int iOperation)
 *
 * PURPOSE:      This function tracks and reports the end of file condition
 *                on the delilog file.
 *
 * ARGUMENTS:    int iOperation      requested operation this can be:
 *
 *                LOGFILE_READ_EOF   check log file flag return current state
 *
 *                LOGFILE_CLEAR_EOF  clear end of log file flag
 *
 *                LOGFILE_SET_EOF    set flag end of log file reached
 *
 * RETURNS:      None
 *
 * COMMENTS:     None
 */

static BOOL LogEOF(int iOperation)
{
    static BOOL bEOF;

    switch(iOperation)
    {
        case LOGFILE_READ_EOF:
            return bEOF;
            break;
        case LOGFILE_CLEAR_EOF:
            bEOF = FALSE;
            break;
        case LOGFILE_SET_EOF:
            bEOF = TRUE;
            break;
    }

    return FALSE;
}

/* FUNCTION: static BOOL ReadReportLine(char *szBuffer, PRPTLINE pRptLine)
 *
 * PURPOSE:      This function reads a text line from the delilog file.
 *                on the delilog file.
 *
 * ARGUMENTS:    char *szBuffer      buffer to placed read delilog file line into.
 *                PRPTLINE pRptLine  returned structure
 *                containing parsed delilog
 *
 *

```

```

report line.
*
* RETURNS:          FALSE  if successfull or TRUE if an error occurs.
*
* COMMENTS:        None
*/
static BOOL ReadReportLine(char *szBuffer, PRPTLINE pRptLine)
{
    int i = 0;
    int ch;
    int iEof;

    while( i < 128 )
    {
        ch = fgetc(fpLog);
        if ( iEof = feof(fpLog) )
            break;
        if ( ch == '\r' )
        {
            if ( i )
                break;
            continue;
        }
        if ( ch == '\n' )
            continue;
        szBuffer[i++] = ch;
    }

    //delivery item format is to long cannot be a valid delivery item
    if ( i >= 128 )
        return TRUE;

    szBuffer[i] = 0;
    if ( iEof )
    {
        LogEOF(LOGFILE_SET_EOF);
        if ( i == 0 )
            return FALSE;
    }
    if ( szBuffer[0] == '*' )
    {
        //error line ignore
        return FALSE;
    }
    return ParseReportLine(szBuffer, pRptLine);
}

```

```

}
/* FUNCTION: static BOOL ParseReportLine(char *szLine, PRPTLINE pRptLine)
*
* PURPOSE:         This function reads a text line from the delilog file.
*                  on the delilog file.
*
* ARGUMENTS:      char          *szLine          buffer containing the delilog file
line to be parsed.
*                  PRPTLINE      pRptLine      returned structure
containing parsed delilog
*                  report line values.
*
* RETURNS:         FALSE  if successfull or TRUE if an error occurs.
*
* COMMENTS:        None
*/
static BOOL ParseReportLine(char *szLine, PRPTLINE pRptLine)
{
    int i;

    if ( ParseDate(szLine, &pRptLine->start) )
        return TRUE;

    pRptLine->end.wYear = pRptLine->start.wYear;
    pRptLine->end.wMonth = pRptLine->start.wMonth;
    pRptLine->end.wDay = pRptLine->start.wDay;

    if ( !(szLine = strchr(szLine, ',')) )
        return TRUE;
    szLine++;

    if ( ParseTime(szLine, &pRptLine->start) )
        return TRUE;

    if ( !(szLine = strchr(szLine, ',')) )
        return TRUE;
    szLine++;

    if ( ParseTime(szLine, &pRptLine->end) )
        return TRUE;

    if ( !(szLine = strchr(szLine, ',')) )
        return TRUE;
    szLine++;
}

```

```

    if ( !IsNumeric(szLine) )
        return TRUE;
    pRptLine->response = atoi(szLine);

    if ( !(szLine = strchr(szLine, ',')) )
        return TRUE;
    szLine++;

    if ( !IsNumeric(szLine) )
        return TRUE;
    pRptLine->w_id = atoi(szLine);

    if ( !(szLine = strchr(szLine, ',')) )
        return TRUE;
    szLine++;

    if ( !IsNumeric(szLine) )
        return TRUE;
    pRptLine->o_carrier_id = atoi(szLine);

    if ( !(szLine = strchr(szLine, ',')) )
        return TRUE;
    szLine++;

    for(i=0; i<10; i++)
    {
        if ( !IsNumeric(szLine) )
            return TRUE;
        pRptLine->items[i] = atoi(szLine);

        if ( i<9 && !(szLine = strchr(szLine, ',')) )
            return TRUE;
        szLine++;
    }

    return FALSE;
}

/* FUNCTION: static BOOL ParseDate(char *szDate, LPSYSTEMTIME pTime)
*
* PURPOSE:      This function validates and extracts a date string in the format
*               yy/mm/dd into an SYSTEMTIME structure.
*
* ARGUMENTS:   char          *szDate          buffer containing the
date to be parsed.
*               LPSYSTEMTIME  pTime          system time
structure where date will be placed.
*
* RETURNS:     FALSE if successfull or TRUE if an error occurs.
*
* COMMENTS:    None
*/

```

```

*
* RETURNS:     FALSE if successfull or TRUE if an error occurs.
*
* COMMENTS:    None
*/

static BOOL ParseDate(char *szDate, LPSYSTEMTIME pTime)
{
    if ( !isdigit(*szDate) || !isdigit(*(szDate+1)) || *(szDate+2) != '/' ||
        !isdigit(*(szDate+3)) || !isdigit(*(szDate+4)) || *(szDate+5) != '/' ||
        !isdigit(*(szDate+6)) || !isdigit(*(szDate+7)) )
        return TRUE;

    pTime->wYear = atoi(szDate);

    pTime->wMonth = atoi(szDate+3);

    pTime->wDay = atoi(szDate+6);

    if ( pTime->wMonth > 12 || pTime->wMonth < 0 || pTime->wDay > 31 || pTime->wDay
< 0 )
        return TRUE;

    return FALSE;
}

/* FUNCTION: static BOOL ParseTime(char *szTime, LPSYSTEMTIME pTime)
*
* PURPOSE:     This function validates and extracts a time string in the format
*               hh:mm:ss:mmm into an SYSTEMTIME structure.
*
* ARGUMENTS:   char          *szTime          buffer containing the
time to be parsed.
*               LPSYSTEMTIME  pTime          system time
structure where date will be placed.
*
* RETURNS:     FALSE if successfull or TRUE if an error occurs.
*
* COMMENTS:    None
*/

static BOOL ParseTime(char *szTime, LPSYSTEMTIME pTime)
{
    if ( !isdigit(*szTime) || !isdigit(*(szTime+1)) || *(szTime+2) != ':' ||
        !isdigit(*(szTime+3)) || !isdigit(*(szTime+4)) || *(szTime+5) != ':' ||
        !isdigit(*(szTime+6)) || !isdigit(*(szTime+7)) || *(szTime+8) != ':' ||

```



```

        !isdigit(*(szTime+9)) || !isdigit(*(szTime+10)) || !isdigit(*(szTime+11)) )
        return TRUE;

    pTime->wHour = atoi(szTime);
    pTime->wMinute = atoi(szTime+3);
    pTime->wSecond = atoi(szTime+6);
    pTime->wMilliseconds = atoi(szTime+9);

    if ( pTime->wHour > 23 || pTime->wHour < 0 ||
        pTime->wMinute > 59 || pTime->wMinute < 0 ||
        pTime->wSecond > 59 || pTime->wSecond < 0 ||
        pTime->wMilliseconds < 0 )
        return TRUE;

    if ( pTime->wMilliseconds > 999 )
    {
        pTime->wSecond += (pTime->wMilliseconds/1000);
        pTime->wMilliseconds = pTime->wMilliseconds % 1000;
    }

    return FALSE;
}

/* FUNCTION: void ErrorMessage(int iError)
 *
 * PURPOSE:      This function displays an error message in the delivery executable's console
 * window.
 *
 * ARGUMENTS:   int          iError    error id to be displayed
 *
 * RETURNS:     None
 *
 * COMMENTS:    None
 */

static void ErrorMessage(int iError)
{
    int i;

    static SERRORMSG errorMsgs[] =
    {
        { ERR_SUCCESS,
          "Success, no error."
        },
        { ERR_CANNOT_OPEN_RESULTS_FILE,
          "Cannot open delivery results file delilog."
        },
    },
};

```

```

        { ERR_READING_LOGFILE,
          "Reading delivery log file, Delivery item format incorrect."
        },
        { ERR_INSUFFICIENT_MEMORY,
          "insufficient memory to process 90th percentile report."
        },
        { 0, ""
        }
    };

    for(i=0; errorMsgs[i].szMsg[0] i++)
    {
        if ( iError == errorMsgs[i].iError )
        {
            printf("\nError(%d): %s", iError, errorMsgs[i].szMsg);
            return;
        }
    }
    printf("Error(%d): %s", errorMsgs[0].szMsg);
    return;
}

/* FUNCTION: BOOL GetParameters(int argc, char *argv[])
 *
 * PURPOSE:      This function parses the command line passed in to the delivery execut
 * initializing
 *                and filling in global variable parameters.
 *
 * ARGUMENTS:   int          argc    number of command line arguments pas
 * to delivery
 *                char      *argv[]  array of command line argumen
 * pointers
 *
 * RETURNS:     BOOL   FALSE   parameter read successfull
 *                TRUE    user has requested
 * parameter information screen be displayed.
 *
 * COMMENTS:    None
 */

static BOOL GetParameters(int argc, char *argv[])
{
    int          i;
    SYSTEMTIME  startTime;
    SYSTEMTIME  endTime;

    iStartTime = 0;

```

```

iEndTime = 0;
iReport = 4;

for(i=0; i<argc; i++)
{
    if ( argv[i][0] == '-' || argv[i][0] == '/' )
    {
        switch(argv[i][1])
        {
            case 'S':
            case 's':
                if ( ParseTime(argv[i]+2, &startTime) )
                    return TRUE;
                iStartTime = (startTime.wHour * 3600000) +
(startTime.wMinute * 60000) + (startTime.wSecond * 1000) + startTime.wMilliseconds;
                break;
            case 'E':
            case 'e':
                if ( ParseTime(argv[i]+2, &endTime) )
                    return TRUE;
                iEndTime = (endTime.wHour * 3600000) +
(endTime.wMinute * 60000) + (endTime.wSecond * 1000) + endTime.wMilliseconds;
                break;
            case 'R':
            case 'r':
                iReport = atoi(argv[i]+2);
                if ( iReport > 4 || iReport < 1 )
                    iReport = 4;
                break;
            case '?':
                return TRUE;
        }
    }
}
return FALSE;
}

/* FUNCTION: void PrintParameters(void)
*
* PURPOSE:      This function displays the supported command line flags.
*
* ARGUMENTS:   None
*
* RETURNS:     None
*
* COMMENTS:    None
*/

```

```

static void PrintParameters(void)
{
    PrintHeader();
    printf("DELIRPT:\n\n");
    printf("Parameter                                     Default\n");
    printf("-----\n");
    printf("-S Start Time HH:MM:SS:MMM                          All  \n");
    printf("-E End Time HH:MM:SS:MMM                            All  \n");
    printf("-R 1)Average Response, 2)90th 3) Skipped 4) All      All  \n");
    printf("-? This help screen\n\n");
    printf("Note: Command line switches are NOT case sensitive.\n");

    return;
}

/* FUNCTION: void PrintHeader(void)
*
* PURPOSE:      This function displays the delivery report applications banner information
*
* ARGUMENTS:   None
*
* RETURNS:     None
*
* COMMENTS:    None
*/

static void PrintHeader(void)
{
    cls();

    printf("*****\n");
    printf("                *\n");
    printf("Microsoft SQL Server 6.5                *\n");
    printf("                *\n");
    printf("HTML TPC-C BENCHMARK KIT: Delivery Report *\n");
    printf("Version %d.%2d.%3d                *\n", versionMS, versionMM,
versionLS);
    printf("                *\n");
    printf("*****\n");

    return;
}

/* FUNCTION: void cls(void)
*
* PURPOSE:      This function clears the console window

```

```

*
* ARGUMENTS:  None
*
* RETURNS:    None
*
* COMMENTS:   None
*/

static void cls(void)
{
    HANDLE hConsole;
    COORD coordScreen = { 0, 0 }; //here's where we'll
home the cursor
    DWORD cCharsWritten;
    CONSOLE_SCREEN_BUFFER_INFO csbi; //to get buffer info
    DWORD dwConSize; //number of character cells in the current buffer

    hConsole = GetStdHandle(STD_OUTPUT_HANDLE);

    //get the number of character cells in the current buffer

    GetConsoleScreenBufferInfo( hConsole, &csbi );
    dwConSize = csbi.dwSize.X * csbi.dwSize.Y;

    //fill the entire screen with blanks
    FillConsoleOutputCharacter( hConsole, (TCHAR) ' ', dwConSize, coordScreen,
&cCharsWritten );
    GetConsoleScreenBufferInfo( hConsole, &csbi );

    //now set the buffer's attributes accordingly
    FillConsoleOutputAttribute( hConsole, csbi.wAttributes,dwConSize, coordScreen,
&cCharsWritten );

    //put the cursor at (0, 0)
    SetConsoleCursorPosition( hConsole, coordScreen );

    return;
}

/* FUNCTION: BOOL IsNumeric(char *ptr)
*
* PURPOSE:    This function determines if a string is numeric. It fails if any characters
other
*             than numeric and null terminator are present.
*
* ARGUMENTS:  char *ptr pointer to string to check.

```

```

*
* RETURNS:    BOOL FALSE if string is not all numeric
*             TRUE  if string contains onl
numeric characters i.e. '0' - '9'
*
* COMMENTS:   A comma is counted as a valid delimiter.
*/

static BOOL IsNumeric(char *ptr)
{
    if ( *ptr == 0 )
        return FALSE;

    while( *ptr && isdigit(*ptr) )
        ptr++;
    if ( !*ptr || *ptr == ',' )
        return TRUE;
    else
        return FALSE;
}

```

A.2 delivery.c

```

/* FILE:      DELIVERY.C
*
* Based on:Microsoft TPC-C Kit Ver. 3.00.000
*
* Copyright Microsoft, 1996
* Copyright Performance Tuning Corporation, 1997
*
* PURPOSE:    New Order Tuxedo Server.
* Author:     Philip Durr
*             philipdu@Microsoft.com
*
* MODIFIED    Changed for modularity and to allow for the Tuxedo TM
*
* Author:     Edward Whalen
*             Performance Tuning Corporation
*             ewhalen@perftuning.com
*/

#include <windows.h>
#include <process.h>
#include <stdio.h>
#include <stdarg.h>
#include <malloc.h>

```

```

#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <sys\timeb.h>
#include <io.h>

#define DBNTWIN32
#include <sqlfront.h>
#include <sqldb.h>

#include "trans.h" //tpckit transaction
header contains definitions of structures specific to TPC-C
#include "httpext.h" //ISAPI DLL information header

#include "tpcc.h" //this dlls specific
structure, value e.t. header.

#include <tmenv.h>
#include <xa.h>
#include <atmi.h>

BOOL bLog = FALSE;
BOOL bFlush; //Flush
delivery log info when written.
BOOL verbose = FALSE;
BOOL bError = FALSE;

int iThreads = 5;
int iMaxWareHouses = 500;
int iDelayMs = 100;
short iMaxConnections = (short)1;
short iDeadlockRetry = (short)3;

DBPROCESS *pdbproc;

char szServer[32];
//SQL server name
char szDatabase[32];
//tpcc database name
char szUser[32];
//user name
char szPassword[32];
//user password
int spld;

#ifdef LOCAL_ALLOC
DELIVERY_DATA DeliveryData;
#else

```

```

TUX_DATA TuxData;
#endif
TERM Term;

static charszTpccLogPath[256]; //path to html log file if logging turned on in registry.
static char szErrorLogPath[256]; //path to error log file.

static CRITICAL_SECTION CriticalSection;
static CRITICAL_SECTION ErrorLogCriticalSection;
static EXTENSION_CONTROL_BLOCK *gpECB = NULL;
static int bTpccExit;
//exit delivery disconnect loop as dll exiting.

extern void TMLog();
extern BOOL SQLInit();
extern void UtilStrCpy();
extern void UtilStrCpy();
extern BOOL SQLOpenConnection();
extern BOOL SQLCloseConnection();

//BOOL bDone;
//delivery executable termination request flag
BOOL bFlush;
//Flush delivery log info when written.

#define ERR_CANNOT_CREATE_THREAD 1000 //Cannot create thre
#define ERR_DBGETDATA_FAILED 1001 //Get data failed.
#define ERR_REGISTRY_NOT_SETUP 1002 //Registry
not setup for tpcc.
#define ERR_CANNOT_ACCESS_DELIVERY_FN 1003 //Cannot access
ReadDelivery cache.
#define ERR_CANNOT_ACCESS_REGISTRY 1004 //Cannot access
registry key TPCC.
#define ERR_CANNOT_CREATE_RESULTS_FILE 1005 //Cannot create resu
file.

FILE *fpLog;

/* FUNCTION: tpsvrint ( int argc, char *argv[])
*
* PURPOSE: Initialize the Server to Database connection.
*
* RETURNS: int 0 Success
-1 Failure
*
*
* COMMENTS: None
*

```

```

*/
int tpsvrinit ( int argc, char *argv[] )
{
    if ( GetParameters(argc, argv) )
    {
        PrintParameters();
        return -1;
    }

    if ( verbose )
        TMLog("TPSVRINIT: Delivery: Server %s, Database %s, User %s,
Password %s, Flush %d.",
            szServer, szDatabase, szUser, szPassword, bFlush);

    if ( ! SQLInit() )
    {
        TMLog( "DELIVERY: SQLInit Failed" );
        return -1;
    }

    if ( SQLOpenConnection ( NULL, 0, 0, &pdbproc, szServer, szDatabase, szUser,
szPassword, szDatabase, &spld) )
    {
        TMLog( "DELIVERY: SQLOpenConnection Failed" );
        dbexit();
        return -1;
    }

    OpenLogFile();

    return 0;
}

/* FUNCTION: tpsvrdone ( void )
*
* PURPOSE:      Initialize the Server to Database connection.
*
* RETURNS:      int      0      Success
*               -1      Failure
*
* COMMENTS:    None
*/
void tpsvrdone ( void )

```

```

{
    SQLCloseConnection( NULL, pdbproc);
    dbexit();
}

/* FUNCTION: DELIVERY ( TPSVCINFO *rqst )
*
* PURPOSE:      Process a New Order request.
*
* RETURNS:      int      0      Success
*               -1      Failure
*
* COMMENTS:    None
*/

void DELIVERY ( TPSVCINFO *rqst )
{
    PECBINFO pECBInfo = dbgetuserdata(pdbproc);
    int size = rqst->len;

    if (verbose)
        TMLog(" DELIVERY: Begin transaction");

#ifdef LOCAL_ALLOC
    memcpy(&DeliveryData, rqst->data, size);

    if (verbose )
    {
        TMLog(" DELIVERY: w_id %d ", DeliveryData.w_id);
        TMLog(" DELIVERY: d_id %d ", DeliveryData.o_carrier_id);
    }

    bError = FALSE;

    DeliveryData.retval = SQLDelivery( pdbproc, &DeliveryData, iDeadlockRetry);

    if (bError == TRUE)
        DeliveryData.retval = -1;

    memcpy( rqst->data, &DeliveryData, size);
#else
    memcpy(&TuxData, rqst->data, size);

    if (verbose )
    {
        TMLog(" DELIVERY: w_id %d ", TuxData.DeliveryData.w_id);

```

```

        TMLog(" DELIVERY: d_id %d ", TuxData.DeliveryData.o_carrier_id);
    }

    bError = FALSE;

    TuxData.DeliveryData.retval = SQLDelivery( pdbproc, &TuxData.DeliveryData,
iDeadlockRetry);

    if (bError == TRUE)
        TuxData.DeliveryData.retval = -1;

    memcpy( rqst->data, &TuxData, size);
#endif
    tpreturn( TPSUCCESS, 0, rqst->data, size, 0);
}

/* FUNCTION: void CalculateElapsedTime(int *pElapsed, LPSYSTEMTIME lpBegin,
LPSYSTEMTIME lpEnd)
*
* PURPOSE:      This function calculates the elapsed time a delivery transaction took.
*
* ARGUMENTS:    int                *pElapsedpointer to int variable to
receive calculated elapsed
*
*               time in milliseconds.
*               LPSYSTEMTIME  lpBegin        Pointer to
system time structure containing
*
*               transaction beginning time.
*               LPSYSTEMTIME  lpEnd         Pointer to
system time structure containing
*
*               transaction ending time.
* RETURNS:      None
*
* COMMENTS:    None
*/

static void CalculateElapsedTime(int *pElapsed, LPSYSTEMTIME lpBegin, LPSYSTEMTIME
lpEnd)
{
    int                beginSeconds;
    int                endSeconds;

```

```

        beginSeconds = (lpBegin->wHour * 3600000) + (lpBegin->wMinute * 60000) +
(lpBegin->wSecond * 1000) + lpBegin->wMilliseconds;
        endSeconds = (lpEnd->wHour * 3600000) + (lpEnd->wMinute * 60000) + (lpEnd-
>wSecond * 1000) + lpEnd->wMilliseconds;
        *pElapsed = endSeconds - beginSeconds;

        //check for day boundry, this will function for 24 houperiod however it will not work
over 48 hours.
        if ( *pElapsed < 0 )
            *pElapsed = *pElapsed + (24 * 60 * 60 * 1000);

        return;
    }

/* FUNCTION: int SQLDelivery(DBPROCESS *dbproc, DELIVERY *pDelivery, short
deadlock_retry )
*
* PURPOSE:      This function processes the delivery transaction.
*
* ARGUMENTS:    DELIVERY          *pDelivery        Pointer to delivery
transaction structure
*
* RETURNS:      int                ERR_DBGETDATA_FAILED    Delivery g
data operation failed.
*
*               ERR_SUCCESS
*               Delivery successfull, no error
*
* COMMENTS:    None
*/

static int SQLDelivery(DBPROCESS *dbproc, DELIVERY_DATA *pDelivery, short
deadlock_retry)
{
    RETCODE          rc;
    int              i;
    int              deadlock_count;
    BYTE             *pData;
    SYSTEMTIME       trans_end;                //delivery transaction finished
time
    int              elapsed;                  //delivery transaction time

    deadlock_count = 0;

    // Start new delivery
    while ( TRUE )

```

```

    {
        if (dbrpcinit(dbproc, "tpcc_delivery", 0) == SUCCEED)
        {
            dbrpcparam(dbproc, NULL, 0, SQLINT2, -1, -1, (BYTE
*)&pDelivery->w_id);
            dbrpcparam(dbproc, NULL, 0, SQLINT1, -1, -1, (BYTE
*)&pDelivery->o_carrier_id);

            if (dbrpcexec(dbproc) == SUCCEED)
            {
                while (((rc = dbresults(dbproc)) !=
NO_MORE_RESULTS) && (rc != FAIL))
                {
                    while (((rc = dbnextrow(dbproc)) !=
NO_MORE_ROWS) && (rc != FAIL))
                    {
                        for (i=0;i<10;i++)
                        {
                            if(pData=dbdata(dbproc, i+1))
                                pDelivery->o_id[i] = *((DBINT *)pData);
                            else
                                pDelivery->o_id[i] = 0;
                        }
                    }
                }

                if ( !SQLDetectDeadlock(dbproc))
                    break;
                deadlock_count++;
                Sleep(10 * deadlock_count);
            }
            GetLocalTime(&trans_end);

            CalculateElapsedTime(&elapsed, &pDelivery->queue_time, &trans_end);

            fprintf(fpLog,
"%2.2d/%2.2d/%2.2d,%2.2d,%2.2d,%2.2d,%3.3d,%2.2d,%2.2d,%2.2d,%3.3d,%d,%d,%d,%d,%
d,%d,%d,%d,%d,%d,%d,%d,%d\r\n",
            trans_end.wYear - 1900, trans_end.wMonth, trans_end.wDay,
            pDelivery->queue_time.wHour, pDelivery->queue_time.wMinute,
            pDelivery->queue_time.wSecond, pDelivery->queue_time.wMilliseconds,
            trans_end.wHour, trans_end.wMinute, trans_end.wSecond,
            trans_end.wMilliseconds,
            elapsed,

```

```

            pDelivery->w_id, pDelivery->o_carrier_id,
            pDelivery->o_id[0], pDelivery->o_id[1], pDelivery->o_id[2], pDelivery->
>o_id[3],
            pDelivery->o_id[4], pDelivery->o_id[5], pDelivery->o_id[6], pDelivery->
>o_id[7],
            pDelivery->o_id[8], pDelivery->o_id[9] );

            if ( bFlush )
                fflush(fpLog);

            return ERR_SUCCESS;
        }

/* FUNCTION: BOOL SQLDetectDeadlock(DBPROCESS *dbproc)
*
* PURPOSE:      This function is used to check for deadlock conditions.
*
* ARGUMENTS:   DBPROCESS          *dbproc          DBPROCESS to
check
*
* RETURNS:     BOOL   FALSE          No lock
condition present
*
*              TRUE
Lock condition detected
*
* COMMENTS:    None
*/

static BOOL SQLDetectDeadlock(DBPROCESS *dbproc)
{
    if (*(BOOL *) dbgetuserdata(dbproc)) == TRUE)
    {
        *(BOOL *) dbgetuserdata(dbproc) = FALSE;
        return TRUE;
    }
    return FALSE;
}

/* FUNCTION: int OpenLogFile(void)
*
* PURPOSE:      This function opens the delivery log file for use.
*
* ARGUMENTS:    None
*
* RETURNS:      int          ERR_REGISTRY_NOT_SETUP
Registry not setup.

```

```

*
*      ERR_CANNOT_CREATE_RESULTS_FILE      Cannot create results log file.
*      ERR_SUCCESS                          ERR_SUCCESS
*      Log file successfully opened
*
*
* COMMENTS:      None
*
*/

static int OpenLogFile(void)
{
    HKEY    hKey;
    BOOL    bRc;
    BYTE    szTmp[256];
    char    szKey[256];
    char    szLogPath[256];
    DWORD   size;
    DWORD   sv;
    int     len;
    char    *ptr;

    szLogPath[0] = 0;
    bRc = TRUE;
    if ( RegOpenKeyEx(HKEY_LOCAL_MACHINE,
"SYSTEM\\CurrentControlSet\\Services\\W3SVC\\Parameters\\Virtual Roots", 0,
KEY_ALL_ACCESS, &hKey) == ERROR_SUCCESS )
    {
        sv = sizeof(szKey);
        size = sizeof(szTmp);

        if ( RegEnumValue(hKey, 0, szKey, &sv, NULL, NULL, szTmp, &size) ==
ERROR_SUCCESS )
        {
            strcpy(szLogPath, szTmp);
            bRc = FALSE;
        }
        RegCloseKey(hKey);
    }

    if ( bRc )
        return ERR_REGISTRY_NOT_SETUP;

    if ( (ptr = strchr(szLogPath, '\\'))
        *ptr = 0;

    len = strlen(szLogPath);
    if ( szLogPath[len-1] != '\\')

```

```

{
    szLogPath[len] = '\\';
    szLogPath[len+1] = 0;
}
strcat(szLogPath, "delilog.");

fpLog = fopen(szLogPath, "ab");

if ( !fpLog )
    return ERR_CANNOT_CREATE_RESULTS_FILE;

return ERR_SUCCESS;
}

/*
*      Common Code for all Servers
*/

/* FUNCTION: BOOL SQLInit()
*
* PURPOSE:      This function initializes SQL Server for later use.
*
* RETURNS:      BOOL    FALSE    if successfull
*              TRUE     if an error occurs an
connection cannot be established.
*
* COMMENTS:      None
*
*/
BOOL SQLInit ()
{
    dbinit();

    if ( dbgetmaxprocs() < iMaxConnections )
    {
        if ( dbsetmaxprocs(iMaxConnections) == FAIL )
        {
            //set for fail error message when HttpExtensionProc() is calle
because
            //at this point we don't have a pECB so no way to show error
message.
            iMaxConnections = -1;
        }
    }

    // install error and message handlers
    dbmsghandle((DBMSGHANDLE_PROC)msg_handler);
}

```



```

        dberrhande((DBERRHANDLE_PROC)err_handler);

        return TRUE;
    }

/* FUNCTION: BOOL SQLOpenConnection(EXTENSION_CONTROL_BLOCK *pECB, int
iTermId, int iSynclD, DBPROCESS **dbproc, char *server, char *database, char *user, char
*password, char *app, int *spid, long *pack_size)
*
* PURPOSE:      This function opens the sql connection for use.
*
* ARGUMENTS:   EXTENSION_CONTROL_BLOCK *pECB passed in structure
pointer from inetsrv.
*
*              int iTermId
*              terminal id of browser
*              int iSynclD
*              sync id of browser
*              DBPROCESS **dbproc pointer to
returned DBPROCESS
*              char *server
*              SQL server name
*              char *databaseSQL server
*              char *user
*              user name
*              char *password
*              user password
*              char *app
*              pointer to returned application array
*              int *spid
*              pointer to returned spid
*              long *pack_size
*              pointer to returned default pack size
*
* RETURNS:     BOOL FALSE if successfull
*              TRUE  if an error occurs
*
* COMMENTS:    None
*
*/

#ifdef USE_ODBC
        static BOOL SQLOpenConnection(EXTENSION_CONTROL_BLOCK *pECB, int
iTermId, int iSynclD, DBPROCESS **dbproc, char *server, char *database, char *user, char
*password, char *app, int *spid, long *pack_size)
        {

```

```

        RETCODE rc;
        char buffer[30];

        *dbproc = (DBPROCESS *)malloc(sizeof(DBPROCESS));
        if (!*dbproc)
            return TRUE;

        //set pECB data into dbproc
        (*dbproc)->bDeadlock = FALSE;
        (*dbproc)->bFailed = FALSE;
        (*dbproc)->pECB = pECB;
        (*dbproc)->iTermId = iTermId;
        (*dbproc)->iSynclD = iSynclD;

        if ( SQLAllocConnect(henv &(*dbproc)->hdbc) == SQL_ERROR )
            return TRUE;

        if ( SQLSetConnectOption((*dbproc)->hdbc, SQL_PACKET_SIZE,
pack_size) == SQL_ERROR )
            return TRUE;

        rc = SQLConnect((*dbproc)->hdbc, server, SQL_NTS, user, SQL_NTS,
password, SQL_NTS);
        if (rc != SQL_SUCCESS && rc != SQL_SUCCESS_WITH_INFO)
            return TRUE;
        rc = SQLAllocStmt((*dbproc)->hdbc, &(*dbproc)->hstmt);
        if (rc == SQL_ERROR)
            return TRUE;

        sprintf(buffer,"use %s", Client->database);

        rc = SQLExecDirect((*dbproc)->hstmt, buffer, SQL_NTS);
        if (rc != SQL_SUCCESS && rc != SQL_SUCCESS_WITH_INFO)
            return TRUE;

        SQLFreeStmt((*dbproc)->hstmt, SQL_CLOSE);
        sprintf(buffer,"set nocount on");
        rc = SQLExecDirect((*dbproc)->hstmt, buffer, SQL_NTS);
        if (rc != SQL_SUCCESS && rc != SQL_SUCCESS_WITH_INFO)
            return TRUE;
        SQLFreeStmt((*dbproc)->hstmt, SQL_CLOSE);

        sprintf(buffer,"select @@spid");

        rc = SQLExecDirect((*dbproc)->hstmt, buffer, SQL_NTS);
        if (rc != SQL_SUCCESS && rc != SQL_SUCCESS_WITH_INFO)

```

```

        return TRUE;

        if ( SQLBindCol((*dbproc)->hstmt, 1, SQL_C_SSHORT, &(*dbproc)->spid,
0, NULL) == SQL_ERROR )
            return TRUE;

        if ( SQLFetch((*dbproc)->hstmt) == SQL_ERROR )
            return TRUE;

        SQLFreeStmt((*dbproc)->hstmt, SQL_CLOSE);

        return FALSE;
    }
#else

    static BOOL SQLOpenConnection(EXTENSION_CONTROL_BLOCK *pECB, int
iTermId, int iSyncId, DBPROCESS **dbproc, char *server, char *database, char *user, char
*password, char *app, int *spid)
    {
        LOGINREC      *login;
        PECBINFO      pEcbInfo;

        //set local msg proc for login record
        //attach pECB record

        //this is necessary as dblib provides no way to pass user data in a login
structure. So until
        //there is an allocated dbproc we need to use a static which means that the
login attempt must
        //be serialized.

        gpECB = pECB;

        login = dblogin();
        if ( !*user )
            DBSETLUSER(login, "sa");
        else
            DBSETLUSER(login, user);

        DBSETLPWD(login, password);
        DBSETLHOST(login, app);

        // Do not set the packet size. Use the size set up in SQL Server.
        // DBSETLPACKET(login, (unsigned short)DEFCLPACKSIZE);

        // This can potentially cut down on data conversion
        DBSETLVERSION(login, DBVER60);

```

```

if ((*dbproc = dbopen(login, server )) == NULL)
    return TRUE;

//set pECB data into dbproc
pEcbInfo = (PECBINFO)malloc(sizeof(PECBINFO));
pEcbInfo->bDeadlock = FALSE;
pEcbInfo->pECB = pECB;
pEcbInfo->iTermId = iTermId;
pEcbInfo->iSyncId = iSyncId;
dbsetuserdata(*dbproc, pEcbInfo);

// Use the the right database
dbuse(*dbproc, database);

dbcmd(*dbproc, "select @@spid");

dbsqlexec(*dbproc);
while (dbresults(*dbproc) != NO_MORE_RESULTS)
{
    dbbind(*dbproc, 1, SMALLBIND, (DBINT) 0, (BYTE *) spid);
    while (dbnextrow(*dbproc) != NO_MORE_ROWS)
        ;
}
dbcmd(*dbproc, "set nocount on");

dbsqlexec(*dbproc);
while (dbresults(*dbproc) != NO_MORE_RESULTS)
{
    while (dbnextrow(*dbproc) != NO_MORE_ROWS)
        ;
}

//rollback transaction on abort
dbcmd(*dbproc, "set XACT_ABORT ON");

dbsqlexec(*dbproc);
while (dbresults(*dbproc) != NO_MORE_RESULTS)
{
    while (dbnextrow(*dbproc) != NO_MORE_ROWS)
        ;
}

return FALSE;
}
#endif

```

```

/* FUNCTION: BOOL SQLCloseConnection(EXTENSION_CONTROL_BLOCK *pECB,
DBPROCESS *dbproc)
*
* PURPOSE:      This function closes the sql connection.
*
* ARGUMENTS:   EXTENSION_CONTROL_BLOCK *pECB  passed in structure
pointer from inetsrv.
*              DBPROCESS *dbproc  pointer to
DBPROCESS
*
* RETURNS:     BOOL  FALSE  if successfull
*              TRUE   if an error occurs
*
* COMMENTS:    None
*/

```

```

#ifdef USE_ODBC
    static BOOL SQLCloseConnection(EXTENSION_CONTROL_BLOCK *pECB,
DBPROCESS *dbproc)
    {
        if ( dbproc )
        {
            SQLFreeStmt(dbproc->hstmt, SQL_DROP);
            SQLDisconnect(dbproc->hdbc);
            SQLFreeConnect(dbproc->hdbc);
            free(dbproc);
            dbproc = NULL;
        }
        return FALSE;
    }
#else
    static BOOL SQLCloseConnection(EXTENSION_CONTROL_BLOCK *pECB,
DBPROCESS *dbproc)
    {
        if (dbcclose(dbproc) == FAIL)
            return TRUE;
        return FALSE;
    }
#endif

```

```

// Lifted from HP FDR since they did such a nice job
void TMLog( char *format, ... )
{
    va_list args;
    char buf[4096];
    int len;
    va_start( args, format );

```

```

    _strtime( buf );
    strcat( buf, " ");
    len = strlen( buf );
    (void)_vsprintf( buf+ len, sizeof( buf) - len - 1, format, args);
    buf[sizeof( buf) - 1]= '\0';
    va_end( args );
    userlog( buf );
}

```

```

/* FUNCTION: void UtilStrCpy(char *pDest, char *pSrc, int n)
*
* PURPOSE:      This function copies n characters from string pSrc to pDst and places a
*              null character at the end of the destination string.
*
* ARGUMENTS:   char *pDest  destination string pointer
*              char *pSrc   source
string pointer
*              int n
*              number of characters to copy
*
* RETURNS:     None
*
* COMMENTS:    Unlike strncpy this function ensures that the result string is
*              always null terminated.
*/

```

```

static void UtilStrCpy(char *pDest, char *pSrc, int n)
{
    strncpy(pDest, pSrc, n);
    pDest[n] = '\0';

    return;
}

```

```

/* FUNCTION: int err_handler(DBPROCESS *dbproc, int severity, int dberr, int oserr, char
*derrmsg, char *oserrmsg)
*
* PURPOSE:      This function handles DB-Libraryerrors
*
* ARGUMENTS:   DBPROCESS *dbproc
DBPROCESS id pointer
*              int severity
*              severity of error
*              int dberr
*              error id
*              int oserr
*              operating system specific error code

```

```

*          char          *dberrstr
printable error description of dberr
*          char          *oserrstr
printable error description of oserr
*
* RETURNS:          int          INT_CONTINUE
continue if error is SQLETIME else INT_CANCEL action
*
* COMMENTS:        None
*/

int err_handler(DBPROCESS *dbproc, int severity, int dberr, int oserr, char *dberrstr, char
*oserrstr)
{
    PECBINFO
    EXTENSION_CONTROL_BLOCK      *pECB;
    FILE
    SYSTEMTIME
    char
    int
    int
    pEcblInfo;
    *fp;
    systemTime;
    szTmp[256];
    iTermId;
    iSynclId;

    pEcblInfo = NULL;

    if ((dbproc == NULL) || (DBDEAD(dbproc)))
    {
        TMLog("DBPROC is invalid");
        return INT_CANCEL;
    }

    if ( !(pEcblInfo = (PECBINFO)dbgetuserdata(dbproc)) )
    {
        pECB = gpECB;
        iTermId = 0;
        iSynclId = 0;
    }
    else
    {
        pECB = pEcblInfo->pECB;
        iTermId = pEcblInfo->iTermId;
        iSynclId = pEcblInfo->iSynclId;
    }

    if ( pEcblInfo && pEcblInfo->bFailed )
    {
        bError == FALSE;

```

```

        return INT_CANCEL;
    }

    if ( oserr != DBNOERR )
    {
        TMLog("DBLIB Error %s", oserrstr);
        if ( pEcblInfo )
        {
            pEcblInfo->bFailed = TRUE;
            bError = TRUE;
        }

        GetLocalTime(&systemTime);
        fp = fopen(szErrorLogPath, "ab");

        sprintf(szTmp, "ErrorHandler: DBLIB(%d): %s", oserr, oserrstr);

        TMLog("%2.2d/%2.2d/%2.2d %2.2d:%2.2d:%2.2d\r\n\r\n%s\r\n\r\n",
            systemTime.wYear, systemTime.wMonth, systemTime.wDay,
            systemTime.wHour, systemTime.wMinute,
            systemTime.wSecond,
            szTmp);

        fclose(fp);
    }

    return INT_CANCEL;
}

/* FUNCTION: int msg_handler(DBPROCESS *dbproc, DBINT msgno, int msgstate, int
severity, char *msgtext)
*
* PURPOSE:          This function handles DB-Library SQL Server error messages
*
* ARGUMENTS:        DBPROCESS          *dbproc
                    DBPROCESS id pointer
*
                    DBINT              msgno
                    message number
*
                    int                msgstate
                    message state
*
                    int                severity
                    message severity
*
                    char                *msgtext
                    printable message description
*
* RETURNS:          int                INT_CONTINUE
                    continue if error is SQLETIME else INT_CANCEL action

```

```

*
*      INT_CANCEL          cancel operation
*
* COMMENTS:   This function also sets the dead lock dbproc variable if necessary.
*/

int msg_handler(DBPROCESS *dbproc, DBINT msgno, int msgstate, int severity, char
*msgtext)
{
    PECBINFO
    EXTENSION_CONTROL_BLOCK    *pECB;
    FILE                        *fp;
    SYSTEMTIME                  systemTime;
    char                        szTmp[256];
    int                         iTermId;
    int                         iSynclId;

    if ( !(pEcbInfo = (PECBINFO)dbgetuserdata(dbproc)) )
    {
        pECB = gpECB;
        iTermId = 0;
        iSynclId = 0;
    }
    else
    {
        pECB = pEcbInfo->pECB;
        iTermId = pEcbInfo->iTermId;
        iSynclId = pEcbInfo->iSynclId;
    }

    if ( (msgno == 5701) || (msgno == 2528) || (msgno == 5703) || (msgno == 6006) )
        return INT_CONTINUE;

    // deadbck message
    if (msgno == 1205)
    {
        // set the deadlock indicator
        if ( pEcbInfo )
            pEcbInfo->bDeadlock = TRUE;
        else
            TMLog("Error, dbgetuserdata returned NULL.");
        return INT_CONTINUE;
    }
    if ( pEcbInfo && pEcbInfo->bFailed )
    {
        TMLog("SQL Error ");

```

```

        return INT_CANCEL;
    }
    if (msgno == 0)
        return INT_CONTINUE;
    else
    {
        TMLog("MsgHandler: SQL Error %s", msgtext);

        if ( pEcbInfo )
            pEcbInfo->bFailed = TRUE;

        bError = TRUE;

        sprintf(szTmp, "Error: SQLSVR(%d): %s", msgno, msgtext);

        TMLog("%2.2d/%2.2d/%2.2d %2.2d:%2.2d:%2.2d\n\n%s\n\n",
            systemTime.wYear, systemTime.wMonth, systemTime.wDay,
            systemTime.wHour, systemTime.wMinute,
            systemTime.wSecond,
            szTmp);
    }
    return INT_CANCEL;
}

/* FUNCTION: BOOL GetParameters(int argc, char *argv[])
*
* PURPOSE:   This function parses the command line passed in to the delivery execut
initializing
*
*            and filling in global variable parameters.
*
* ARGUMENTS: int          argc      number of command line arguments pas
to delivery
*            char        *argv[]   array of command line argume
pointers
*
* RETURNS:   BOOL        FALSE     parameter read successfull
*            TRUE        TRUE      user has requested
parameter information screen be displayed.
*
* COMMENTS:  None
*/

static BOOL GetParameters(int argc, char *argv[])
{
    int i;

```

```

szServer[0]          = 0;
szPassword[0]       = 0;
bFlush              = FALSE;
strcpy(szDatabase, "tpcc");
strcpy(szUser, "sa");

for(i=0; i<argc; i++)
{
    if ( argv[i][0] == '-' || argv[i][0] == '/' )
    {
        switch(argv[i][1])
        {
            case 'S':
            case 's':
                strcpy(szServer, argv[i]+2);
                break;

            case 'V':
            case 'v':
                verbose = TRUE;
                break;

            case 'F':
            case 'f':
                bFlush = TRUE;    //turn on delilog flush

            case '?':
                break;

            default:
                return TRUE;
        }
    }
}

return FALSE;
}

/* FUNCTION: void PrintParameters(void)
 *
 * PURPOSE:      This function displays the supported command line flags.
 *
 * ARGUMENTS:    None
 *
 * RETURNS:      None
 *
 * COMMENTS:     None
 */

static void PrintParameters(void)
{

```

```

    TMLog("Performance Tuning Corporation Tuxedo Kit");
    TMLog(" www.perftuning.com (281) 251-3495 ");
    TMLog("Delivery: -S Server [-v (verbose)] [-F (Flush delilog)]");
    TMLog("Delivery: Server %s Flush %d.", szServer, bFlush);
}

```

A.3 HttpExt.h

```

/*****
 *
 * Copyright (c) 1995 Process Software Corporation
 *
 * Copyright (c) 1995 Microsoft Corporation
 *
 * Module Name : HttpExt.h
 *
 * Abstract :
 *
 * This module contains the structure definitions and prototypes for the
 * version 1.0 HTTP Server Extension interface.
 *
 *****/

#ifndef _HTTPEXT_H_
#define _HTTPEXT_H_

#include <windows.h>

#ifdef __cplusplus
extern "C" {
#endif

#define HSE_VERSION_MAJOR 1 // major version of this spec
#define HSE_VERSION_MINOR 0 // minor version of this spec
#define HSE_LOG_BUFFER_LEN 80
#define HSE_MAX_EXT_DLL_NAME_LEN 256

typedef LPVOID HCONN;

// the following are the status codes returned by the Extension DLL

#define HSE_STATUS_SUCCESS 1
#define HSE_STATUS_SUCCESS_AND_KEEP_CONN 2
#define HSE_STATUS_PENDING 3
#define HSE_STATUS_ERROR 4

```

```

// The following are the values to request services with the ServerSupportFunction.
// Values from 0 to 1000 are reserved for future versions of the interface

#define HSE_REQ_BASE 0
#define HSE_REQ_SEND_URL_REDIRECT_RESP (HSE_REQ_BASE + 1)
#define HSE_REQ_SEND_URL (HSE_REQ_BASE + 2)
#define HSE_REQ_SEND_RESPONSE_HEADER (HSE_REQ_BASE + 3)
#define HSE_REQ_DONE_WITH_SESSION (HSE_REQ_BASE + 4)
#define HSE_REQ_END_RESERVED 1000

//
// These are Microsoft specific extensions
//

#define HSE_REQ_MAP_URL_TO_PATH (HSE_REQ_END_RESERVED+1)
#define HSE_REQ_GET_SSPI_INFO (HSE_REQ_END_RESERVED+2)

//
// passed to GetExtensionVersion
//

typedef struct _HSE_VERSION_INFO {

    DWORD dwExtensionVersion;
    CHAR lpszExtensionDesc[HSE_MAX_EXT_DLL_NAME_LEN];
} HSE_VERSION_INFO, *LPHSE_VERSION_INFO;

//
// passed to extension procedure on a new request
//

typedef struct _EXTENSION_CONTROL_BLOCK {

    DWORD cbSize; // size of this struct.
    DWORD dwVersion; // version info of this spec
    HCONN ConnID; // Context number not to be modified!
    DWORD dwHttpStatusCode; // HTTP Status code
    CHAR lpszLogData[HSE_LOG_BUFFER_LEN]; // null terminated log info specific to this
    Extension DLL

    LPSTR lpszMethod; // REQUEST_METHOD
    LPSTR lpszQueryString; // QUERY_STRING
    LPSTR lpszPathInfo; // PATH_INFO
    LPSTR lpszPathTranslated; // PATH_TRANSLATED

    DWORD cbTotalBytes; // Total bytes indicated from client
    
```

```

DWORD cbAvailable; // Available number of bytes
LPBYTE lpbData; // pointer to cbAvailable bytes

LPSTR lpszContentType; // Content type of client data

BOOL (WINAPI * GetServerVariable) ( HCONN hConn,
    LPSTR lpszVariableName,

    LPVOID lpvBuffer,
    LPDWORD lpdwSize );

BOOL (WINAPI * WriteClient) ( HCONN ConnID,
    LPVOID Buffer,
    LPDWORD lpdwBytes,
    DWORD dwReserved );

BOOL (WINAPI * ReadClient) ( HCONN ConnID,
    LPVOID lpvBuffer,
    LPDWORD lpdwSize );

BOOL (WINAPI * ServerSupportFunction)( HCONN hConn,
    DWORD dwHSERRequest,
    LPVOID lpvBuffer,
    LPDWORD lpdwSize,
    LPDWORD lpdwDataType );

} EXTENSION_CONTROL_BLOCK, *LPEXTENSION_CONTROL_BLOCK;

//
// these are the prototypes that must be exported from the extension DLL
//

BOOL WINAPI GetExtensionVersion( HSE_VERSION_INFO *pVer );
DWORD WINAPI HttpExtensionProc( EXTENSION_CONTROL_BLOCK *pECB );

// the following type declarations is for the server side

typedef BOOL (WINAPI * PFN_GETEXTENSIONVERSION)( HSE_VERSION_INFO *pVer );
typedef DWORD (WINAPI * PFN_HTTPEXTENSIONPROC )(
    EXTENSION_CONTROL_BLOCK *pECB );

#ifdef __cplusplus
}
#endif

#endif // end definition _HTTPEXT_H_
    
```

A.4 install.c

```

/* FILE:          INSTALL.C
 *                Microsoft TPC-C Kit Ver. 3.00.000
 *                Audited 08/23/96, By Francois Raab
 *
 *                Copyright Microsoft, 1996
 *
 * PURPOSE:       Automated installation application for TPC-C Web Kit
 * Author:        Philip Durr
 *                philipdu@Microsoft.com
 */

#include <windows.h>
#include <direct.h>
#include <iio.h>
#include <stdlib.h>
#include <stdio.h>
#include <commctrl.h>
#include "install.h"

HICON          hIcon;
HINSTANCE      hInst;

DWORD          versionExeMS;
DWORD          versionExeLS;
DWORD          versionDllMS;
DWORD          versionDllLS;

static BOOL    bLog;
static int     iThreads;
static int     iMaxWareHouse;
static int     iDelayMs;
static int     iDeadlockRetry;
static int     iMaxConnections;
static int     iPoolThreadsLimit;
static int     iThreadTimeout;
static int     iListenBackLog;
static int     iAcceptExOutstanding;
static int     iQSlotts;

static int     iMaxPhysicalMemory;           //max physical memory

BOOL CALLBACK UpdatedDlgProc(HWND hwnd, UINT uMsg, WPARAM wParam,
LPARAM lParam);
BOOL CALLBACK MainDlgProc(HWND hwnd, UINT uMsg, WPARAM wParam,
LPARAM lParam);

```

```

BOOL CALLBACK CopyDlgProc(HWND hwnd, UINT uMsg, WPARAM wParam,
LPARAM lParam);
static void ReadRegistrySettings(void);
static void WriteRegistrySettings(char *szDllPath);
static int CopyFiles(HWND hDlg, char *szDllPath);
static BOOL GetInstallPath(char *szDllPath);
static void GetVersionInfo(char *szDllPath, char *szExePath);
static BOOL StartWWWService(void);
static BOOL StopWWWService(void);
static void UpdateDialog(HWND hDlg);

int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR
lpCmdLine, int nCmdShow )
{
    int iRc;

    hInst = hInstance;

    InitCommonControls();

    hIcon = LoadIcon(hInstance, MAKEINTRESOURCE(IDI_ICON1));

    iRc = DialogBox(hInstance, MAKEINTRESOURCE(IDD_DIALOG1),
GetDesktopWindow(), MainDlgProc);
    if ( iRc )
        DialogBoxParam(hInstance,
MAKEINTRESOURCE(IDD_DIALOG2), GetDesktopWindow(), UpdatedDlgProc,
(LPARAM)iRc);
    DestroyIcon(hIcon);

    return 0;
}

BOOL CALLBACK UpdatedDlgProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM
lParam)
{
    switch(uMsg)
    {
        case WM_INITDIALOG:
            if ( lParam == 1 )
                SetDlgItemText(hwnd, IDC_RESULTS, "HTML TPC
Installation Successful");
            else
                SetDlgItemText(hwnd, IDC_RESULTS, "HTML TPC
Registry Updated");
            return TRUE;
        case WM_COMMAND:
            if ( wParam == IDOK )

```



```

        EndDialog(hwnd, TRUE);
    default:
        break;
    }
    return FALSE;
}

BOOL CALLBACK MainDlgProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
    PAINTSTRUCT ps;
    MEMORYSTATUS memoryStatus;
    int d;
    int rc;
    HWND hDlg;
    char szTmp[256];
    static char szDllPath[256];
    static char szExePath[256];

    switch(uMsg)
    {
        case WM_INITDIALOG:
            GlobalMemoryStatus(&memoryStatus);
            iMaxPhysicalMemory = (memoryStatus.dwTotalPhys/
1048576);

            if ( GetInstallPath(szDllPath) )
            {
                MessageBox(hwnd, "Error internet service inetsrv is
not installed.", NULL, MB_ICONSTOP | MB_OK);
                EndDialog(hwnd, FALSE);
                return TRUE;
            }

            bLog = FALSE;
            iThreads = 4;
            iMaxWareHouse = 500;
            iDelayMs = 500;
            iDeadlockRetry = 3;
            iMaxConnections = 25;
            iPoolThreadsLimit = iMaxPhysicalMemory * 2;
            iThreadTimeout = 86400;
            iListenBackLog = 15;
            iAcceptExOutstandng = 40;

            ReadRegistrySettings();
    }
}

```

```

    GetModuleFileName(hInst, szExePath, sizeof(szExePath));
    GetVersionInfo(szDllPath, szExePath);
    if ( bLog )
        CheckDlgButton(hwnd, BN_LOG, 1);

    wsprintf(szTmp, "Version %d.00.%3d", versionExeMS,
versionExeLS);

    SetDlgItemText(hwnd, IDC_VERSION, szTmp);

    SetDlgItemText(hwnd, IDC_PATH, szDllPath);
    SetDlgItemInt(hwnd, ED_MAXWARE, iMaxWareHouse,
FALSE);

    SetDlgItemInt(hwnd, ED_THREADS, iThreads, FALSE);
    SetDlgItemInt(hwnd, ED_MAXCONNECTION,
iMaxConnections, FALSE);

    SetDlgItemInt(hwnd, ED_IIS_MAX_THREAD_POOL_LIMIT,
iPoolThreadsLimit, FALSE);

    SetDlgItemInt(hwnd, ED_IIS_THREAD_TIMEOUT,
iThreadTimeout, FALSE);

    SetDlgItemInt(hwnd, ED_IIS_LISTEN_BACKLOG,
iListenBackLog, FALSE);

    SetDlgItemInt(hwnd,
ED_WEB_SERVICE_BACKLOG_QUEUE_SIZE, iAcceptExOutstanding, FALSE);

    return TRUE;
case WM_PAINT:
    if ( !IsIconic(hwnd) )
    {
        BeginPaint(hwnd, &ps);
        DrawIcon(ps.hdc, 0, 0, hIcon);
        EndPaint(hwnd, &ps);
        return TRUE;
    }
    break;
case WM_COMMAND:
    if ( wParam == IDOK )
    {
        if ( !IsDlgButtonChecked(hwnd, BN_LOG) )
            bLog = TRUE;
        else
            bLog = FALSE;

        iThreads = GetDlgItemInt(hwnd, ED_THREADS, &
FALSE);

        iMaxWareHouse = GetDlgItemInt(hwnd,
ED_MAXWARE, &d, FALSE);

        iMaxConnections = GetDlgItemInt(hwnd,
ED_MAXCONNECTION, &d, FALSE);
    }
}

```

```

        iPoolThreadsLimit = GetDlgItemInt(hwnd,
ED_IIS_MAX_THREAD_POOL_LIMIT, &d, FALSE);
        iThreadTimeout = GetDlgItemInt(hwnd,
ED_IIS_THREAD_TIMEOUT, &d, FALSE);
        iListenBackLog = GetDlgItemInt(hwnd,
ED_IIS_LISTEN_BACKLOG, &d, FALSE);
        iAcceptExOutstanding = GetDlgItemInt(hwnd,
ED_WEB_SERVICE_BACKLOG_QUEUE_SIZE, &d, FALSE);

        ShowWindow(hwnd, SW_HIDE);
        hDlg = CreateDialog(hInst,
MAKEINTRESOURCE(IDD_DIALOG3), hwnd, CopyDlgProc);
        ShowWindow(hDlg, SW_SHOWNA);
        UpdateDialog(hDlg);
        rc = CopyFiles(hDlg, szDIIPath);
        if ( !rc )
        {
            ShowWindow(hwnd, SW_SHOWNA);
            DestroyWindow(hDlg);
            MessageBox(hwnd, "Error(s) ocured when
creating tpcc.dll", NULL, MB_ICONSTOP | MB_OK);
            EndDialog(hwnd, 0);
            return TRUE;
        }
        SetDlgItemText(hDlg, IDC_STATUS, "Updating
Registry.");
        SendDlgItemMessage(hDlg, IDC_PROGRESS1,
PBM_STEPIT, 0, 0);

        UpdateDialog(hDlg);

        WriteRegistrySettings(szDIIPath);

        Sleep(100);

        ShowWindow(hwnd, SW_SHOWNA);
        DestroyWindow(hDlg);

        EndDialog(hwnd, rc);
        return TRUE;
    }
    if ( wParam == IDCANCEL )
    {
        EndDialog(hwnd, FALSE);
        return TRUE;
    }
    break;
default:

```

```

        break;
    }
    return FALSE;
}

static void ReadRegistrySettings(void)
{
    HKEY    hKey;
    DWORD   size;
    DWORD   type;
    char    szTmp[256];

    if ( RegOpenKeyEx(HKEY_LOCAL_MACHINE, "SOFTWARE\\Microsoft\\TPCC", 1,
KEY_READ, &hKey) == ERROR_SUCCESS )
    {
        size = sizeof(szTmp);

        bLog = FALSE;
        if ( RegQueryValueEx(hKey, "LOG", 0, &type, szTmp, &size) ==
ERROR_SUCCESS )
            if ( !strcmp(szTmp, "ON") )
                bLog = TRUE;

        iThreads = 4;
        size = sizeof(szTmp);
        if ( RegQueryValueEx(hKey, "NumberOfDeliveryThreads", 0, &type, szT
&size) == ERROR_SUCCESS )
            iThreads = atoi(szTmp);
        if ( iThreads == 0 )
            iThreads = 4;

        iMaxWareHouse = 500;
        size = sizeof(szTmp);
        if ( RegQueryValueEx(hKey, "MaximumWarehouses", 0, &type, szTmp,
&size) == ERROR_SUCCESS )
            iMaxWareHouse = atoi(szTmp);
        if ( iMaxWareHouse == 0 )
            iMaxWareHouse = 500;

        iDelayMs = 500;
        size = sizeof(szTmp);
        if ( RegQueryValueEx(hKey, "BackoffDelay", 0, &type, szTmp, &size) ==
ERROR_SUCCESS )
            iDelayMs = atoi(szTmp);
        if ( iDelayMs == 0 )
            iDelayMs = 500;

        iDeadlockRetry = 3;
        size = sizeof(szTmp);

```

```

        if ( RegQueryValueEx(hKey, "DeadlockRetry", 0, &type, szTmp, &size) ==
ERROR_SUCCESS )
            iDeadlockRetry = atoi(szTmp);
            if ( !iDeadlockRetry )
                iDeadlockRetry = 3;

            iMaxConnections = 25;
            size = sizeof(szTmp);
            if ( RegQueryValueEx(hKey, "MaxConnections", 0, &type, szTmp, &size)
== ERROR_SUCCESS )
                iMaxConnections = atoi(szTmp);
                if ( !iMaxConnections )
                    iMaxConnections = 25;

            iQSlotts = 3000;
            size = sizeof(szTmp);
            if ( RegQueryValueEx(hKey, "QueueSlotts", 0, &type, szTmp, &size) ==
ERROR_SUCCESS )
                iQSlotts = atoi(szTmp);
                if ( iQSlotts == 0 )
                    iQSlotts = 3000;

            RegCloseKey(hKey);

            if ( RegOpenKeyEx(HKEY_LOCAL_MACHINE,
"SYSTEM\\CurrentControlSet\\Services\\Inetinfo\\Parameters", 0, KEY_READ, &hKey) ==
ERROR_SUCCESS )
            {
                iPoolThreadsLimit = iMaxPhysicalMemory * 2;
                size = sizeof(iPoolThreadsLimit);
                if ( RegQueryValueEx(hKey, "PoolThreadsLimit", 0, &type, (char
*)&iPoolThreadsLimit, &size) == ERROR_SUCCESS )
                    if ( !iPoolThreadsLimit )
                        iPoolThreadsLimit = iMaxPhysicalMemory * 2;

                iThreadTimeout = 86400;
                size = sizeof(iThreadTimeout);
                if ( RegQueryValueEx(hKey, "ThreadTimeout", 0, &type, (char
*)&iThreadTimeout, &size) == ERROR_SUCCESS )
                    if ( !iThreadTimeout )
                        iThreadTimeout = 86400;

                iListenBackLog = 15;
                size = sizeof(iListenBackLog);
                if ( RegQueryValueEx(hKey, "ListenBackLog", 0, &type, (char
*)&iListenBackLog, &size) == ERROR_SUCCESS )
                    if ( !iListenBackLog )
                        iListenBackLog = 15;
            }
    
```

```

        }
        RegCloseKey(hKey);

        if ( RegOpenKeyEx(HKEY_LOCAL_MACHINE,
"SYSTEM\\CurrentControlSet\\Services\\W3SVC\\Parameters", 0, KEY_READ, &hKey) ==
ERROR_SUCCESS )
        {
            iAcceptExOutstanding = 40;
            size = sizeof(iAcceptExOutstanding);
            if ( RegQueryValueEx(hKey, "AcceptExOutstanding", 0, &type
(char *)&iAcceptExOutstanding, &size) == ERROR_SUCCESS )
                if ( !iAcceptExOutstanding )
                    iAcceptExOutstanding = 40;
        }

        RegCloseKey(hKey);
    }

    return;
}

static void WriteRegistrySettings(char *szDllPath)
{
    HKEY    hKey;
    DWORD   dwDisposition;
    char    szTmp[256];
    char    *ptr;
    int     iRc;

    if ( RegCreateKeyEx(HKEY_LOCAL_MACHINE, "SOFTWARE\\Microsoft\\TPCC",
NULL, REG_OPTION_NON_VOLATILE, KEY_ALL_ACCESS, NULL, &hKey, &dwDispositio
== ERROR_SUCCESS )
        {
            strcpy(szTmp, szDllPath);
            ptr = strstr(szTmp, "tpcc");
            if ( ptr )
                *ptr = 0;

            RegSetValueEx(hKey, "PATH", 0, REG_SZ, szTmp, strlen(szTmp));

            if ( bLog )
                RegSetValueEx(hKey, "LOG", 0, REG_SZ, "ON", 2);
            else
                RegSetValueEx(hKey, "LOG", 0, REG_SZ, "OFF", 3);

            itoa(iThreads, szTmp, 10);
    }
    
```

```

        RegSetValueEx(hKey, "NumberOfDeliveryThreads", 0, REG_SZ, szTmp,
strlen(szTmp));

        itoa(iMaxWareHouse, szTmp, 10);
        RegSetValueEx(hKey, "MaximumWarehouses", 0, REG_SZ, szTmp,
strlen(szTmp));

        itoa(iDelayMs, szTmp, 10);
        RegSetValueEx(hKey, "BackoffDelay", 0, REG_SZ, szTmp,
strlen(szTmp));

        itoa(iDeadlockRetry, szTmp, 10);
        RegSetValueEx(hKey, "DeadlockRetry", 0, REG_SZ, szTmp,
strlen(szTmp));

        itoa(iMaxConnections, szTmp, 10);
        RegSetValueEx(hKey, "MaxConnections", 0, REG_SZ, szTmp,
strlen(szTmp));

        itoa(iQSlots, szTmp, 10);
        RegSetValueEx(hKey, "QueueSlots", 0, REG_SZ, szTmp, strlen(szTmp));

        RegFlushKey(hKey);

        RegCloseKey(hKey);
    }

    if ( (iRc=RegCreateKeyEx(HKEY_LOCAL_MACHINE,
"SYSTEM\\CurrentControlSet\\Services\\Inetinfo\\Parameters", 0, NULL,
REG_OPTION_NON_VOLATILE, KEY_ALL_ACCESS, NULL, &hKey, &dwDisposition)) ==
ERROR_SUCCESS )
    {
        RegSetValueEx(hKey, "PoolThreadsLimit", 0, REG_DWORD, (char
*)&iPoolThreadsLimit, sizeof(iPoolThreadsLimit));
        RegSetValueEx(hKey, "ThreadTimeout", 0, REG_DWORD, (char
*)&iThreadTimeout, sizeof(iThreadTimeout));
        RegSetValueEx(hKey, "ListenBackLog", 0, REG_DWORD, (char
*)&iListenBackLog, sizeof(iListenBackLog));

        RegFlushKey(hKey);
        RegCloseKey(hKey);
    }

    if ( (iRc=RegCreateKeyEx(HKEY_LOCAL_MACHINE,
"SYSTEM\\CurrentControlSet\\Services\\W3SVC\\Parameters", 0, NULL,
REG_OPTION_NON_VOLATILE, KEY_ALL_ACCESS, NULL, &hKey, &dwDisposition)) ==
ERROR_SUCCESS )
    {

```

```

        RegSetValueEx(hKey, "AcceptExOutstanding", 0, REG_DWORD, (char
*)&iAcceptExOutstanding, sizeof(iAcceptExOutstanding));

        RegFlushKey(hKey);
        RegCloseKey(hKey);
    }

    return;
}

BOOL CALLBACK CopyDlgProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM
lParam)
{
    if ( uMsg == WM_INITDIALOG )
    {
        SendDlgItemMessage(hwnd, IDC_PROGRESS1, PBM_SETRANGE, 0,
MAKELPARAM(0, 8));
        SendDlgItemMessage(hwnd, IDC_PROGRESS1, PBM_SETSTEP,
(WPARAM)1, 0);
        return TRUE;
    }
    return FALSE;
}

static int CopyFiles(HWND hDlg, char *szDllPath)
{
    HGLOBAL          hDLL;
    HGLOBAL          hExe;
    HRSRC            hResInfo;
    BYTE             *pSrc;
    HANDLE           hFile;
    DWORD            dwSize;
    DWORD            d;
    char              szTmp[256];
    char             *ptr;
    BOOL             bSvcRunning;

    SetDlgItemText(hDlg, IDC_STATUS, "Stopping Web Service.");
    SendDlgItemMessage(hDlg, IDC_PROGRESS1, PBM_STEPIT, 0, 0);
    UpdateDialog(hDlg);

    bSvcRunning = !StopWWWWebService();
    SendDlgItemMessage(hDlg, IDC_PROGRESS1, PBM_STEPIT, 0, 0);
    UpdateDialog(hDlg);

    hResInfo = FindResource(hInst, MAKEINTRESOURCE(IDR_TPCCDLL1),
"TPCCDLL");
    SetDlgItemText(hDlg, IDC_STATUS, "Copying Files...");

```

```

SendDlgItemMessage(hDlg, IDC_PROGRESS1, PBM_STEPIT, 0, 0);
UpdateDialog(hDlg);

dwSize = SizeofResource(hInst, hResInfo);
hDLL = LoadResource(hInst, hResInfo );
pSrc = (BYTE *)LockResource(hDLL);
remove(szDllPath);

if ( !(hFile = CreateFile(szDllPath, GENERIC_WRITE, 0, NULL, CREATE_ALWAYS,
FILE_ATTRIBUTE_NORMAL, NULL)) )
    return 0;

if ( !WriteFile(hFile, pSrc, dwSize, &d, NULL) )
    return 0;

CloseHandle(hFile);

UnlockResource(hDLL);
FreeResource(hDLL);

SendDlgItemMessage(hDlg, IDC_PROGRESS1, PBM_STEPIT, 0, 0);
UpdateDialog(hDlg);

hResInfo = FindResource(hInst, MAKEINTRESOURCE(IDR_DELIVERY1),
"DELIVERY");

dwSize = SizeofResource(hInst, hResInfo);
hExe = LoadResource(hInst, hResInfo );
pSrc = (BYTE *)LockResource(hExe);

strcpy(szTmp, szDllPath);
ptr = strstr(szTmp, "tpcc");
if ( ptr )
    *ptr = 0;
strcat(szTmp, "delisrv.exe");

remove(szTmp);

if ( !(hFile = CreateFile(szTmp, GENERIC_WRITE, 0, NULL, CREATE_ALWAYS,
FILE_ATTRIBUTE_NORMAL, NULL)) )
    return 0;

if ( !WriteFile(hFile, pSrc, dwSize, &d, NULL) )
    return 0;

CloseHandle(hFile);

UnlockResource(hExe);
    
```

```

FreeResource(hExe);

SendDlgItemMessage(hDlg, IDC_PROGRESS1, PBM_STEPIT, 0, 0);
UpdateDialog(hDlg);

//if we stopped service restart it.
if ( !bSvcRunning )
{
    SetDlgItemText(hDlg, IDC_STATUS, "Starting Web Service.");
    SendDlgItemMessage(hDlg, IDC_PROGRESS1, PBM_STEPIT, 0, 0);
    UpdateDialog(hDlg);
    StartWWWWebService();
}

SendDlgItemMessage(hDlg, IDC_PROGRESS1, PBM_STEPIT, 0, 0);
UpdateDialog(hDlg);

return 1;
}

static BOOL GetInstallPath(char *szDllPath)
{
    HKEY hKey;
    BYTE szTmp[256];
    char szKey[256];
    DWORD size;
    DWORD sv;
    BOOL bRc;
    int len;
    char *ptr;

    szDllPath[0] = 0;
    bRc = TRUE;
    if ( RegOpenKeyEx(HKEY_LOCAL_MACHINE
"SYSTEM\\CurrentControlSet\\Services\\W3SVC\\Parameters\\Virtual Roots", 0,
KEY_ALL_ACCESS, &hKey) == ERROR_SUCCESS )
    {
        sv = sizeof(szKey);
        size = sizeof(szTmp);

        if ( RegEnumValue(hKey, 0, szKey, &sv, NULL, NULL, szTmp, &size) =
ERROR_SUCCESS )
        {
            strcpy(szDllPath, szTmp);
            bRc = FALSE;
        }
        RegCloseKey(hKey);
    }
}
    
```

```

    if ( (ptr = strchr(szDllPath, ',')) )
        *ptr = 0;

    len = strlen(szDllPath);
    if ( szDllPath[len-1] != '\\ ' )
    {
        szDllPath[len] = '\\';
        szDllPath[len+1] = 0;
    }
    strcat(szDllPath, "tpcc.dll");

    return bRc;
}

static void GetVersionInfo(char *szDLLPath, char *szExePath)
{
    DWORD          d;
    DWORD          dwSize;
    DWORD          dwBytes;
    char           *ptr;
    VS_FIXEDFILEINFO *vs;

    versionDllMS = 0;
    versionDllLS = 0;
    if ( _access(szDLLPath, 00) == 0 )
    {
        dwSize = GetFileVersionInfoSize(szDLLPath, &d);
        if ( dwSize )
        {
            ptr = (char *)malloc(dwSize);
            GetFileVersionInfo(szDLLPath, 0, dwSize, ptr);
            VerQueryValue(ptr, "\\",&vs, &dwBytes);
            versionDllMS = vs->dwProductVersionMS;
            versionDllLS = vs->dwProductVersionLS;
            free(ptr);
        }
    }

    versionExeMS = 0x7FFF;
    versionExeLS = 0x7FFF;
    dwSize = GetFileVersionInfoSize(szExePath, &d);
    if ( dwSize )
    {
        ptr = (char *)malloc(dwSize);
        GetFileVersionInfo(szExePath, 0, dwSize, ptr);
        VerQueryValue(ptr, "\\",&vs, &dwBytes);

        versionExeMS = vs->dwProductVersionMS;
        versionExeLS = vs->dwProductVersionLS;
        free(ptr);
    }
}

static BOOL StartWWWWebService(void)
{
    SC_HANDLE      schSCManager;
    SC_HANDLE      schService;
    SERVICE_STATUS ssStatus;
    DWORD          dwOldCheckPoint;

    schSCManager = OpenSCManager(NULL, NULL, SC_MANAGER_ALL_ACCESS);
    schService = OpenService(schSCManager, TEXT("W3SVC"),
SERVICE_ALL_ACCESS);
    if (schService == NULL)
        return FALSE;

    if (! StartService(schService, 0, NULL) )
        goto StartWWWWebErr;
    //start Service pending, Check the status until the service is running.
    if (! QueryServiceStatus(schService, &ssStatus) )
        goto StartWWWWebErr;
    while( ssStatus.dwCurrentState != SERVICE_RUNNING)
    {
        dwOldCheckPoint = ssStatus.dwCheckPoint;
        //Save the current checkpoint.
        Sleep(ssStatus.dwWaitHint);
        //Wait for the specified interval.
        if ( !QueryServiceStatus(schService, &ssStatus) ) //Check the status
again.
            break;
        if (dwOldCheckPoint >= ssStatus.dwCheckPoint) //Break if
the checkpoint has not been incremented.
            break;
    }

    if (ssStatus.dwCurrentState == SERVICE_RUNNING)
        goto StartWWWWebErr;

    CloseServiceHandle(schService);
    return TRUE;
}

StartWWWWebErr:
    CloseServiceHandle(schService);
    return FALSE;
}

```

```

}

static BOOL StopWWWWebService(void)
{
    SC_HANDLE          schSCManager;
    SC_HANDLE          schService;
    SERVICE_STATUS     ssStatus;
    DWORD              dwOldCheckPoint;

    schSCManager = OpenSCManager(NULL, NULL, SC_MANAGER_ALL_ACCESS);
    schService = OpenService(schSCManager, TEXT(W3SVC"),
SERVICE_ALL_ACCESS);
    if (schService == NULL)
        return FALSE;

    if (! QueryServiceStatus(schService, &ssStatus) )
        goto StopWWWWebErr;

    if ( !ControlService(schService, SERVICE_CONTROL_STOP, &ssStatus) )
        goto StopWWWWebErr;
    //start Service pending. Check the status until the service is running.
    if (! QueryServiceStatus(schService, &ssStatus) )
        goto StopWWWWebErr;
    while( ssStatus.dwCurrentState == SERVICE_RUNNING)
    {
        dwOldCheckPoint = ssStatus.dwCheckPoint;
        //Save the current checkpoint.
        Sleep(ssStatus.dwWaitHint);
        //Wait for the specified interval.
        if ( !QueryServiceStatus(schService, &ssStatus) ) //Check the status
again.
            break;
        if (dwOldCheckPoint >= ssStatus.dwCheckPoint) //Break if
the checkpoint has not been incremented.
            break;
    }

    if (ssStatus.dwCurrentState == SERVICE_RUNNING)
        goto StopWWWWebErr;

    CloseServiceHandle(schService);
    return TRUE;

StopWWWWebErr:
    CloseServiceHandle(schService);
    return FALSE;
}

```

```

static void UpdateDialog(HWND hDlg)
{
    MSG msg;

    UpdateWindow(hDlg);
    while( PeekMessage(&msg, hDlg, 0, 0, PM_REMOVE) )
    {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
    Sleep(250);
    return;
}

```

A.5 install.h

```

//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.
// Used by install.rc
//
#define IDD_DIALOG1          101
#define IDI_ICON1           102
#define IDR_TPCCDLL1        103
#define IDD_DIALOG2         104
#define IDI_ICON2           105
#define IDR_DELIVERY1       109
#define IDD_DIALOG3         110
#define BN_LOG               1001
#define ED_KEEP              1002
#define ED_THREADS          1003
#define ED_THREADS2         1004
#define ED_MAXWARE          1006
#define IDC_PATH             1007
#define IDC_VERSION         1009
#define IDC_RESULTS         1010
#define IDC_PROGRESS1       1011
#define IDC_STATUS          1012
#define IDC_BUTTON1         1013
#define ED_MAXCONNECTION    1014
#define ED_IIS_MAX_THREAD_POOL_LIMIT 1015
#define ED_WEB_SERVICE_BACKLOG_QUEUE_SIZE 1017
#define ED_IIS_THREAD_TIMEOUT 1018
#define ED_IIS_LISTEN_BACKLOG 1019

// Next default values for new objects
//

```

```

#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE        111
#define _APS_NEXT_COMMAND_VALUE        40001
#define _APS_NEXT_CONTROL_VALUE        1015
#define _APS_NEXT_SYMED_VALUE          101
#endif
#endif

A.6 install.rc

//Microsoft Developer Studio generated resource script.
//
#include "install.h"

#define APSTUDIO_READONLY_SYMBOLS
////////////////////////////////////
//
// Generated from the TEXTINCLUDE 2 resource.
//
#include "afxres.h"

////////////////////////////////////
#undef APSTUDIO_READONLY_SYMBOLS

////////////////////////////////////
// English (U.S.) resources

#if !defined(AFX_RESOURCE_DLL) || defined(AFX_TARG_ENU)
#ifdef _WIN32
LANGUAGE LANG_ENGLISH, SUBLANG_ENGLISH_US
#pragma code_page(1252)
#endif // _WIN32

////////////////////////////////////
//
// Dialog

IDD_DIALOG1 DIALOGEX 0, 0, 234, 174
STYLE DS_MODALFRAME | DS_CENTER | WS_MINIMIZEBOX | WS_POPUP |
WS_CAPTION |
WS_SYSMENU
CAPTION "TPCC Web Client Installation Utility"
FONT 8, "MS Sans Serif"
BEGIN
EDITTEXT    ED_MAXWARE,199,37,21,12,ES_NUMBER,WS_EX_RTREADING

```

```

CONTROL     "",BN_LOG,"Button",BS_AUTOCHECKBOX | BS_LEFTTEXT |
BS_LEFT | BS_VCENTER | WS_TABSTOP,205,51,15,13,
WS_EX_STATICEDGE
EDITTEXT    ED_THREADS,205,66,15,12,ES_NUMBER,WS_EX_RTREADING
EDITTEXT
ED_MAXCONNECTION,186,80,34,12,ES_NUMBER,WS_EX_RTREADING
EDITTEXT    ED_IIS_MAX_THREAD_POOL_LIMIT,186,94,34,12,ES_NUMBER,
WS_EX_RTREADING
EDITTEXT    ED_WEB_SERVICE_BACKLOG_QUEUE_SIZE,186,108,34,12,
ES_NUMBER,WS_EX_RTREADING
EDITTEXT    ED_IIS_THREAD_TIMEOUT,186,122,34,12,ES_NUMBER,
WS_EX_RTREADING
EDITTEXT    ED_IIS_LISTEN_BACKLOG,186,136,34,12,ES_NUMBER,
WS_EX_RTREADING
DEFPUSHBUTTON "OK",IDOK,59,153,50,14
PUSHBUTTON   "Cancel",IDCANCEL,125,153,50,14
EDITTEXT    IDC_PATH,42,22,178,13,ES_AUTOHSCROLL | ES_READONLY
LTEXT       "Max Number of Warehouses:",IDC_STATIC,42,37,115,12,
SS_SUNKEN
LTEXT       "Write HTML To Log file:",IDC_STATIC,42,51,115,12,
SS_SUNKEN
LTEXT       "Number of Delivery Threads:",IDC_STATIC,42,66,115,12,
SS_SUNKEN
LTEXT       "Max Number of Connections:",IDC_STATIC,41,80,115,12,
SS_SUNKEN
CTEXT       "Version 1.00.001",IDC_VERSION,42,6,178,14,SS_SUNKEN |
WS_BORDER,WS_EX_CLIENTEDGE
ICON        IDI_ICON1,IDC_STATIC,9,6,21,20,0,WS_EX_CLIENTEDGE
LTEXT       "IIS Max Thread Pool Limit:",IDC_STATIC,41,94,115,12,
SS_SUNKEN
LTEXT       "Web Service Backlog Queue Size:",IDC_STATIC,41,108,115,
12,SS_SUNKEN
LTEXT       "IIS Thread Timeout:",IDC_STATIC,41,122,115,12,SS_SUNKEN
LTEXT       "IIS Listen Backlog:",IDC_STATIC,41,136,115,12,SS_SUNKEN
END

IDD_DIALOG2 DIALOGEX 0, 0, 117, 62
STYLE DS_SETFOREGROUND | DS_3DLOOK | DS_CENTER | WS_POPUP |
WS_BORDER
EXSTYLE WS_EX_STATICEDGE
FONT 12, "MS Sans Serif", 0, 0, 0x1
BEGIN
DEFPUSHBUTTON "OK",IDOK,33,45,50,9
CTEXT       "HTML TPCC Installation Successfull",IDC_RESULTS,7,22,
102,18,0,WS_EX_CLIENTEDGE
ICON        IDI_ICON2,IDC_STATIC,50,7,18,20,SS_REALSIZEIMAGE,
WS_EX_TRANSPARENT
END

```



```

VS_VERSION_INFO VERSIONINFO
FILEVERSION 0,3,0,2
PRODUCTVERSION 0,3,0,2
FILEFLAGSMASK 0x3fL
#ifdef _DEBUG
FILEFLAGS 0x1L
#else
FILEFLAGS 0x0L
#endif
FILEOS 0x40004L
FILETYPE 0x1L
FILESUBTYPE 0x0L
BEGIN
    BLOCK "StringFileInfo"
    BEGIN
        BLOCK "040904b0"
        BEGIN
            VALUE "CompanyName", "Microsoft\0"
            VALUE "FileDescription", "instal\0"
            VALUE "FileVersion", "0, 3, 0, 2\0"
            VALUE "InternalName", "instal\0"
            VALUE "LegalCopyright", "Copyright © 1996\0"
            VALUE "OriginalFilename", "install.exe\0"
            VALUE "ProductName", "Microsoft instal\0"
            VALUE "ProductVersion", "0, 3, 0, 2\0"
        END
    END
    BLOCK "VarFileInfo"
    BEGIN
        VALUE "Translation", 0x409, 1200
    END
END

#endif // !_MAC

////////////////////////////////////
//
// DELIVERY
//

IDR_DELIVERY1      DELIVERY DISCARDABLE  "delisrv.exe"
#endif // English (U.S.) resources
////////////////////////////////////

```

```

#ifndef APSTUDIO_INVOKED
////////////////////////////////////
//
// Generated from the TEXTINCLUDE 3 resource.
//

```

```

////////////////////////////////////
#endif // not APSTUDIO_INVOKED

```

A.7 resource.H

```

//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.
// Used by TPCC.rc
//

```

```

// Next default values for new objects
//

```

```

#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE        101
#define _APS_NEXT_COMMAND_VALUE        40001
#define _APS_NEXT_CONTROL_VALUE        1000
#define _APS_NEXT_SYMED_VALUE          101
#endif
#endif

```

A.8 makefile

```

!IF "$(CFG)" == ""
CFG=Release
!MESSAGE No configuration specified. Defaulting to Debug
!ENDIF

!IF "$(SQL_LOC)" == ""
SQL_LOC=D:\mssql\sql\lib
!MESSAGE No SQL_LOC specified. Defaulting to D:\MSSQL\SQLLIB
!ENDIF

!IF "$(TUXDIR)" == ""
TUXDIR = E:\TUXEDO
!MESSAGE No TUXDIR specified. Defaulting to E:\TUXEDO
!ENDIF

```

```

IIF "$(CFG)" != "Release" && "$(CFG)" != "Debug"
MESSAGE Invalid configuration "$(CFG)" specified.
MESSAGE You can specify a configuration when running NMAKE on this makefile
MESSAGE by defining the macro CFG on the command line. For example:
MESSAGE
MESSAGE NMAKE CFG="Debug"
MESSAGE
MESSAGE Possible choices for configuration are:
MESSAGE
MESSAGE "Release"
MESSAGE "Debug"
MESSAGE
MESSAGE
MESSAGE An invalid configuration is specified.
IENDIF

OUTDIR      = .
SRCDIR     = .\Src
OBJDIR     = .\Objs
OUTDIR     = .\Bin
DBLIB      = $(SQL_LOC)
DBLIBINC   = $(DBLIB)\include
DBLIBDIR   = $(DBLIB)\lib

IIF "$(CFG)" != "Debug"
LDEBUG     =
CDEBUG     =
LDEBUG_RG  =
CDEBUG_RG  =
DEBUG     =
FLAGS      = /D "WIN32" /D "_WINDOWS" /D "_TMSTHREADS"
#FLAGS     = /D "WIN32" /D "_WINDOWS" /D "_TMSTHREADS" /D "LOCAL_ALLOC"
CFLAGS     = /D "WIN32" /D "_WINDOWS" /D "_TMSTHREADS"
OPT        = /Ot
ELSE
LDEBUG     = /debug /pdb:$(OBJDIR)\tpcc.pdb
CDEBUG     = /Zi /Yd /Fd$(OBJDIR)\tpcc.pdb
LDEBUG_RG  = /debug /pdb:$(OBJDIR)\install.pdb
CDEBUG_RG  = /Zi /Yd /Fd$(OBJDIR)\install.pdb
FLAGS      = /D "WIN32" /D "_WINDOWS" /D "_TMSTHREADS"
#FLAGS     = /D "WIN32" /D "_WINDOWS" /D "_TMSTHREADS" /D "LOCAL_ALLOC"
CFLAGS     = /D "WIN32" /D "_WINDOWS" /D "_TMSTHREADS"
OPT        = /Od
ENDIF

LINK32_LIBS = user32.lib msacm32.lib advapi32.lib $(DBLIBDIR)\ntwdblib.lib
    
```

```

TUX_LIBS      = $(TUXDIR)\lib\libtux.lib $(TUXDIR)\lib\libbuft.lib $(TUXDIR)\lib\libtux2.l
\
$(TUXDIR)\lib\libfml.lib $(TUXDIR)\lib\libfml32.lib $(TUXDIR)\lib\libgp.lib
OTHER_LIBS    = wsock32.lib kernel32.lib gdi32.lib comdlg32.lib winspool.lib
LINK32_OBJS   = "$(OBJDIR)\tpcc.obj" "$(OBJDIR)\tpcc.res"
LINK32_DEF    = "$(SRCDIR)\tpcc.def"
LINK32_FLAGS  = /nologo /subsystem:windows /dll /incremental:no $(LDEBUG)
/def:"$(LINK32_DEF)" /out:"$(OUTDIR)\tpcc.dll"

LINK32_LIBS_RG = user32.lib gdi32.lib advapi32.lib version.lib comctl32.lib
LINK32_OBJS_RG = "$(OBJDIR)\install.obj" "$(OBJDIR)\install.res"
LINK32_FLAGS_RG = /nologo /subsystem:windows /incremental:no $(LDEBUG_RG)
/out:$(OUTDIR)\install.exe

ALL: $(OBJDIR)\. $(OUTDIR)\. $(OUTDIR)\tpcc.dll $(OUTDIR)\Neworder.exe
$(OUTDIR)\Payment.exe \
    $(OUTDIR)\Stocklevel.exe $(OUTDIR)\Orderstatus.exe $(OUTDIR)\Delivery.exe
$(OUTDIR)\Delirpt.exe

$(OBJDIR)\.:
    if not exist $(OBJDIR) md $(OBJDIR)

$(OUTDIR)\.:
    if not exist $(OUTDIR) md $(OUTDIR)

"$$(OBJDIR)\tpcc.obj": "$$(SRCDIR)\tpcc.c" "$$(SRCDIR)\tpcc.h"
    cl.exe /nologo /MT /W3 $(CDEBUG) $(OPT) /I $(DBLIBINC) /I $(TUXDIR)\include
$(FLAGS) /Fo$(OBJDIR)\tpcc.obj /c "$$(SRCDIR)\tpcc.c"

$(OBJDIR)\tpcc.res: $(SRCDIR)\tpcc.rc
    rc.exe /I 0x409 /fo $(OBJDIR)\tpcc.res $(FLAGS) $(SRCDIR)\tpcc.rc

$(OUTDIR)\tpcc.dll: $(LINK32_OBJS) $(LINK32_DEF)
    link.exe $(LINK32_FLAGS) $(LINK32_OBJS) $(LINK32_LIBS) $(TUX_LIBS)
$(OTHER_LIBS)

$(OUTDIR)\Neworder.exe: $(SRCDIR)\neworder.c
    $(TUXDIR)\bin\buildserver /f $(SRCDIR)\neworder.c /o $(OUTDIR)\Neworder.exe
/s NEWORDER /I "$(LINK32_LIBS) /I $(DBLIBINC)"
    del neworder.obj

$(OUTDIR)\Payment.exe: $(SRCDIR)\payment.c
    $(TUXDIR)\bin\buildserver /f $(SRCDIR)\payment.c /o $(OUTDIR)\Payment.exe \
/s PAYMENT /I "$(LINK32_LIBS) /I $(DBLIBINC)"
    del payment.obj

$(OUTDIR)\Orderstatus.exe: $(SRCDIR)\orderstatus.c
    
```

```

$(TUXDIR)\bin\buildserver /f $(SRCDIR)\orderstatus.c /o $(OUTDIR)\Orderstatus.exe
\
/s ORDERSTATUS /I "$(LINK32_LIBS) /I $(DBLIBINC)"
del orderstatus.obj

$(OUTDIR)\Stocklevel.exe: $(SRCDIR)\stocklevel.c
$(TUXDIR)\bin\buildserver /f $(SRCDIR)\stocklevel.c /o $(OUTDIR)\Stocklevel.exe \
/s STOCKLEVEL /I "$(LINK32_LIBS) /I $(DBLIBINC)"
del stocklevel.obj

$(OUTDIR)\Delivery.exe: $(SRCDIR)\delivery.c
$(TUXDIR)\bin\buildserver /f $(SRCDIR)\delivery.c /o $(OUTDIR)\Delivery.exe \
/s DELIVERY /I "$(LINK32_LIBS) /I $(DBLIBINC)"
del delivery.obj

$(OUTDIR)\Delirpt.exe: $(SRCDIR)\delirpt.c
cl.exe $(SRCDIR)\delirpt.c /o "$(OUTDIR)\Delirpt.exe"
del delirpt.obj

```

A.9 neworder.c

```

/* FILE: NEWORDER.C
*
* Based on: Microsoft TPC-C Kit Ver. 3.00.000
*
* Copyright Microsoft, 1996
* Copyright Performance Tuning Corporation, 1997
*
* PURPOSE: New Order Tuxedo Server.
* Author: Philip Durr
* philipdu@Microsoft.com
*
* MODIFIED Changed for modularity and to allow for the Tuxedo TM
*
* Author: Edward Whalen
* Performance Tuning Corporation
* ewhalen@perftuning.com
*/

#include <windows.h>
#include <process.h>
#include <stdio.h>
#include <stdarg.h>
#include <malloc.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <sys/timeb.h>

```

```

#include <io.h>

#define DBNTWIN32
#include <sqlfront.h>
#include <sqldb.h>

#include "trans.h" //tpckit transaction
header contains definitions of structures specific to TPC-C //ISAPI DLL information header
#include "httpext.h" //this dlls specific

#include "tpcc.h" //this dlls specific
structure, value e.t. header.

#include <tmenv.h>
#include <xa.h>
#include <atmi.h>

BOOL bLog = FALSE;
BOOL bFlush; //Flush
delivery log info when written.
BOOL verbose = FALSE;
BOOL bError = FALSE;

int iThreads = 5;
int iMaxWareHouses = 500;
int iDelayMs = 100;
short iMaxConnections = (short)1;
short iDeadlockRetry = (short)3;

DBPROCESS *pdbproc;

static char szServer[32];
//SQL server name
static char szDatabase[32];
//tpcc database name
static char szUser[32];
//user name
static char szPassword[32];
//user password
int spld;

#ifdef LOCAL_ALLOC
NEW_ORDER_DATA NewOrderData;
#else
TUX_DATA TuxData;
#endif
TERM Term;

```

```

static char szTpccLogPath[256]; //path to html log file if logging turned on in registry.
static char szErrorLogPath[256]; //path to error log file.

static CRITICAL_SECTION CriticalSection;
static CRITICAL_SECTION ErrorLogCriticalSection;
static EXTENSION_CONTROL_BLOCK *gpECB = NULL;
static int bTpccExit;
//exit delivery disconnect loop as dll exiting.

extern void TMLog();
extern BOOL SQLInit();
extern void UtilStrCpy();
extern void UtilStrCpy();
extern BOOL SQLOpenConnection();
extern BOOL SQLCloseConnection();
extern BOOL SQLDetectDeadlock();

/* FUNCTION: tpsvrinit ( int argc, char *argv[] )
*
* PURPOSE: Initialize the Server to Database connection.
*
* RETURNS: int 0 Success
           -1 Failure
*
* COMMENTS: None
*/

int tpsvrinit ( int argc, char *argv[] )
{
    if ( GetParameters(argc, argv) )
    {
        PrintParameters();
        return -1;
    }

    if ( verbose )
        TMLog("TPSVRINIT: NewOrder: Server %s, Database %s, User %s,
Password %s, Flush %d.",
            szServer, szDatabase, szUser, szPassword, bFlush);

    if ( ! SQLInit() )
    {
        TMLog( "NEWORDER: SQLInit Failed" );
        return -1;
    }
}

```

```

}

if ( SQLOpenConnection ( NULL, 0, 0, &pdbproc, szServer, szDatabase, szUser,
szPassword, szDatabase, &spld))
{
    TMLog ( "NEWORDER: SQLOpenConnection Failed" );
    dbexit();
    return -1;
}

return 0;
}

/* FUNCTION: tpsvrdone ( void )
*
* PURPOSE: Initialize the Server to Database connection.
*
* RETURNS: int 0 Success
           -1 Failure
*
* COMMENTS: None
*/

void tpsvrdone ( void )
{
    SQLCloseConnection( NULL, pdbproc);
    dbexit();
}

/* FUNCTION: NEWORDER ( TPSVCINFO *rqst )
*
* PURPOSE: Process a New Order request.
*
* RETURNS: int 0 Success
           -1 Failure
*
* COMMENTS: None
*/

void NEWORDER ( TPSVCINFO *rqst )
{
    PECBINFO pECBInfo = dbgetuserdata(pdbproc);
    int size = rqst->len;

#ifdef LOCAL_ALLOC
}

```

```

memcpy(&NewOrderData, rqst->data, size);

if (verbose )
{
    TMLog(" NEWORDER: w_id %d ", NewOrderData.w_id);
    TMLog(" NEWORDER: d_id %d ", NewOrderData.d_id);
    TMLog(" NEWORDER: c_id %d ", NewOrderData.c_id);
}

bError = FALSE;

NewOrderData.retval = SQLNewOrder( NULL, 0, 0, pdbproc, &NewOrderData,
iDeadlockRetry);

if (bError == TRUE)
    NewOrderData.retval = -1;

if (verbose )
    TMLog(" NEWORDER: Return Value %d", NewOrderData.retval);

memcpy( rqst->data, &NewOrderData, size);
#else
memcpy(&TuxData, rqst->data, size);

if (verbose )
{
    TMLog(" NEWORDER: w_id %d ", TuxData.NewOrderData.w_id);
    TMLog(" NEWORDER: d_id %d ", TuxData.NewOrderData.d_id);
    TMLog(" NEWORDER: c_id %d ", TuxData.NewOrderData.c_id);
}

bError = FALSE;

TuxData.NewOrderData.retval = SQLNewOrder( NULL, 0, 0, pdbproc,
&TuxData.NewOrderData, iDeadlockRetry);

if (bError == TRUE)
    TuxData.NewOrderData.retval = -1;

if (verbose )
    TMLog(" NEWORDER: Return Value %d", TuxDataNewOrderData.retval);

memcpy( rqst->data, &TuxData.NewOrderData, size);
#endif
treturn( TPSUCCESS, 0, rqst->data, size, 0);
}

```

```

/* FUNCTION: int SQLNewOrder(EXTENSION_CONTROL_BLOCK *pECB, int iTermId, int
iSyncl, int iTermId, int iSyncl, DBPROCESS *dbproc, NEW_ORDER_DATA *pNewOrder
short deadlock_retry)
*
* PURPOSE:      This function handles the new order transaction.
*
* ARGUMENTS:   EXTENSION_CONTROL_BLOCK      *pECB
                passed in structure pointer from inetsrv.
*
*              int
                iTermId      terminal id of browser
*
                int
                iSyncl      sync id of browser
*
                DBPROCESS
                *dbproc      connection db process id
*
                NEW_ORDER_DATA
                *pNewOrder   pointer to new order structure for input/output data
                short
*
                deadlock_retry  retry count if deadlocked
*
* RETURNS:     int      TRUE  transaction committed
*              FALSE  item number not valid
*              -1      deadlock max retry
reached
*
*
* COMMENTS:    None
*
*/

static int SQLNewOrder(EXTENSION_CONTROL_BLOCK *pECB, int iTermId, int iSyncl,
DBPROCESS *dbproc, NEW_ORDER_DATA *pNewOrder, short deadlock_retry)
{
    RETCODE          rc;
    int              i;
    DBINT            commit_flag;
    int              tryit;
    char             printbuff[25];
    char             tmpbuf[30];
    DBDATETIME      datetime;
    BYTE             *pData;
    PECBINFO         pEcbInfo;

    if ( (pEcbInfo = (PECBINFO)dbgetuserdata(dbproc))
    {
        pEcbInfo->pECB = pECB;
        pEcbInfo->bFailed = FALSE;
        pEcbInfo->iTermId = iTermId;
        pEcbInfo->iSyncl = iSyncl;

```

```

}
pNewOrder->num_deadlocks = 0;
strcpy(tmpbuf, "tpcc_neworder");
for (tryit=0; tryit < deadlock_retry; tryit++)
{
    if (dbrpcinit(dbproc, tmpbuf, 0) == SUCCEED)
    {
        dbrpcparam(dbproc, NULL, 0, SQLINT2, -1, -1, (BYTE *)
&pNewOrder->w_id);
        dbrpcparam(dbproc, NULL, 0, SQLINT1, -1, -1, (BYTE *)
&pNewOrder->d_id);
        dbrpcparam(dbproc, NULL, 0, SQLINT4, -1, -1, (BYTE *)
&pNewOrder->c_id);
        dbrpcparam(dbproc, NULL, 0, SQLINT1, -1, -1, (BYTE *)
&pNewOrder->o_ol_cnt);
        //
        dbrpcparam(dbproc, NULL, 0, SQLINT1, -1, -1, (BYTE *)
&pNewOrder->o_all_local);

        pNewOrder->o_all_local = 1;
        for (i = 0; i < pNewOrder->o_ol_cnt; i++)
        {
            if ( pNewOrder->o_all_local && pNewOrder-
>Ol[i].ol_supply_w_id != pNewOrder->w_id )
                pNewOrder->o_all_local = 0;
        }
        dbrpcparam(dbproc, NULL, 0, SQLINT1, -1, -1 (BYTE *)
&pNewOrder->o_all_local);

        for (i = 0; i < pNewOrder->o_ol_cnt; i++)
        {
            dbrpcparam(dbproc, NULL, 0, SQLINT4, -1, -1,
(BYTE *) &pNewOrder->Ol[i].ol_i_id);
            dbrpcparam(dbproc, NULL, 0, SQLINT2, -1, -1,
(BYTE *) &pNewOrder->Ol[i].ol_supply_w_id);
            dbrpcparam(dbproc, NULL, 0, SQLINT2, -1, -1,
(BYTE *) &pNewOrder->Ol[i].ol_quantity);
        }

        if (dbrpcexec(dbproc) == SUCCEED)
        {
            pNewOrder->total_amount=0;

            // Get results from order line
            for (i = 0; i<pNewOrder->o_ol_cnt; i++)
            {

```

```

if (((rc = dbresults(dbproc)) !=
NO_MORE_RESULTS) && (rc != FAIL))
{
    if (DBROWS(dbproc) &&
(dbnumcols(dbproc) == 5))
    {
        while
        (dbnextrow(dbproc) != NO_MORE_ROWS)
        {
            if(pData=dbdata(dbproc, 1))
                UtilStrCpy(pNewOrder->Ol[i].ol_i_name, pData, dbdatlen(dbproc, 1));
            if(pData=dbdata(dbproc, 2))
                pNewOrder->Ol[i].ol_stock = (*(DBSMALLINT *) pData);
            if(pData=dbdata(dbproc, 3))
                UtilStrCpy(pNewOrder->Ol[i].ol_brand_generic, pData, dbdatlen(dbproc, 3));
            if(pData=dbdata(dbproc, 4))
                pNewOrder->Ol[i].ol_i_price = (*(DBFLT8 *) pData);

            if(pData=dbdata(dbproc, 5))

                pNewOrder->Ol[i].ol_amount = (*(DBFLT8 *) pData);

            pNewOrder-
>total_amount = pNewOrder->total_amount + pNewOrder->Ol[i].ol_amount;
        }
    }
}
while (((rc = dbresults(dbproc)) !=
NO_MORE_RESULTS) && (rc != FAIL))
{
    if (DBROWS(dbproc) &&
(dbnumcols(dbproc) == 8))
    {
        while (((rc = dbnextrow(dbproc)
!= NO_MORE_ROWS) && (rc != FAIL))

```

```

{
    if(pData=dbdata(dbproc, 1))
        >w_tax = (*(DBFLT8 *) pData);
    if(pData=dbdata(dbproc, 2))
        >d_tax = (*(DBFLT8 *) pData);
    if(pData=dbdata(dbproc, 3))
        >o_id = (*(DBINT *) pData);
    if(pData=dbdata(dbproc, 4))
        UtilStrCpy(pNewOrder->c_last, pData, dbdatlen(dbproc, 4));
    if(pData=dbdata(dbproc, 5))
        >c_discount = (*(DBFLT8 *) pData);
    if(pData=dbdata(dbproc, 6))
        UtilStrCpy(pNewOrder->c_credit, pData, dbdatlen(dbproc, 6));
    if(pData=dbdata(dbproc, 7))
        {
            *((DBDATETIME *) pData);
            dbdatecrack(dbproc, &pNewOrder->o_entry_d, &datetime);
        }
}

if(pData=dbdata(dbproc, 8))commit_flag = (*(DBTINYINT *) pData);
}
}
}
if (SQLDetectDeadlock(dbproc))
{
    pNewOrder->num_deadlocks++;
    sprintf(printbuf,"deadlock: retry: %d",pNewOrder-
>num_deadlocks);
    Sleep(DEADLOCKWAIT*tryit);
}
else
{
    if (commit_flag == 1)
    {
        pNewOrder->total_amount = pNewOrder-
>total_amount * ((1 + pNewOrder->w_tax + pNewOrder->d_tax) * (1 - pNewOrder->c_discou
strcpy(pNewOrder->execution_status,"Transaction
committed.");
        return TRUE;
    }
    else
    {
        strcpy(pNewOrder->execution_status,"Item numbe
not valid.");
        pNewOrder->error=ERR_BAD_ITEM_ID;
        return FALSE;
    }
}
}

// If we reached here, it means we quit after MAX_RETRY deadlocks
strcpy(pNewOrder->execution_status,"Hit deadlock max. ");
pNewOrder->error=ERR_TYPE_DEADLOCK;
if ( verbose )
    TMLog(" NEWORDER: SQLNewOrder Max Deadlocks %d", tryit);

return -1; // "deadlock max retry reached!"
}
/*
 * Common Code for all Servers
 */
/* FUNCTION: BOOL SQLInit()

```



```

*
* PURPOSE:      This function initializes SQL Server for later use.
*
*
* RETURNS:      BOOL    FALSE  if successfull
*                TRUE    if an error occurs and
connection cannot be established.
*
* COMMENTS:     None
*/
BOOL SQLInit ()
{
    dbinit();

    if ( dbgetmaxprocs() < iMaxConnections )
    {
        if ( dbsetmaxprocs(iMaxConnections) == FAIL )
        {
            //set for fail error message when HttpExtensionProc() is called
            //at this point we don't have a pECB so no way to show error
            //message.
            iMaxConnections = -1;
        }

        // install error and message handlers
        dbmsghandle((DBMSGHANDLE_PROC)msg_handler);
        dberrhandle((DBERRHANDLE_PROC)err_handler);
    }

    return TRUE;
}

```

/* FUNCTION: BOOL SQLOpenConnection(EXTENSION_CONTROL_BLOCK *pECB, int iTermId, int iSynclD, DBPROCESS **dbproc, char *server, char *database, char *user, char *password, char *app, int *spid, long *pack_size)

```

*
* PURPOSE:      This function opens the sql connection for use.
*
* ARGUMENTS:    EXTENSION_CONTROL_BLOCK *pECB  passed in structure
pointer from inetsrv.
*                int iTermId
*                int iSynclD
*                sync id of browser

```

```

*                DBPROCESS **dbproc pointer to
returned DBPROCESS
*                char *server
*                SQL server name
*                char *databaseSQL serv
database
*                char *user
*                user name
*                char *password
*                user password
*                char *app
*                pointer to returned application array
*                int *spid
*                pointer to returned spid
*                long *pack_size
*                pointer to returned default pack size
*
* RETURNS:      BOOL    FALSE  if successfull
*                TRUE    if an error occurs
*
* COMMENTS:     None
*/

#ifdef USE_ODBC
static BOOL SQLOpenConnection(EXTENSION_CONTROL_BLOCK *pECB, int
iTermId, int iSynclD, DBPROCESS **dbproc, char *server, char *database, char *user, char
*password, char *app, int *spid, long *pack_size)
{
    RETCODE rc;
    char buffer[30];

    *dbproc = (DBPROCESS *)malloc(sizeof(DBPROCESS));
    if ( !*dbproc )
        return TRUE;

    //set pECB data into dbproc
    (*dbproc)->bDeadlock = FALSE;
    (*dbproc)->bFailed = FALSE;
    (*dbproc)->pECB = pECB;
    (*dbproc)->iTermId = iTermId;
    (*dbproc)->iSynclD = iSynclD;

    if ( SQLAllocConnect(henv,&(*dbproc)->hdbc) == SQL_ERROR )
        return TRUE;
}

```

```

        if ( SQLSetConnectOption((*dbproc)->hdbc, SQL_PACKET_SIZE,
pack_size) == SQL_ERROR )
            return TRUE;

        rc = SQLConnect((*dbproc)->hdbc, server, SQL_NTS, user, SQL_NTS,
password, SQL_NTS);
        if (rc != SQL_SUCCESS && rc != SQL_SUCCESS_WITH_INFO)
            return TRUE;
        rc = SQLAllocStmt((*dbproc)->hdbc, &(*dbproc)->hstmt);
        if (rc == SQL_ERROR)
            return TRUE;

        sprintf(buffer,"use %s", Client->database);

        rc = SQLExecDirect((*dbproc)->hstmt, buffer, SQL_NTS);
        if (rc != SQL_SUCCESS && rc != SQL_SUCCESS_WITH_INFO)
            return TRUE;

        SQLFreeStmt((*dbproc)->hstmt, SQL_CLOSE);
        sprintf(buffer,"set nocount on");
        rc = SQLExecDirect((*dbproc)->hstmt, buffer, SQL_NTS);
        if (rc != SQL_SUCCESS && rc != SQL_SUCCESS_WITH_INFO)
            return TRUE;
        SQLFreeStmt((*dbproc)->hstmt, SQL_CLOSE);

        sprintf(buffer,"select @@spid");

        rc = SQLExecDirect((*dbproc)->hstmt, buffer, SQL_NTS);
        if (rc != SQL_SUCCESS && rc != SQL_SUCCESS_WITH_INFO)
            return TRUE;

        if ( SQLBindCol((*dbproc)->hstmt, 1, SQL_C_SSHORT, &(*dbproc)->spid,
0, NULL) == SQL_ERROR )
            return TRUE;

        if ( SQLFetch((*dbproc)->hstmt) == SQL_ERROR )
            return TRUE;

        SQLFreeStmt((*dbproc)->hstmt, SQL_CLOSE);

        return FALSE;
    }
}

```

#else

```

        static BOOL SQLOpenConnection(EXTENSION_CONTROL_BLOCK *pECB, int
iTermId, int iSyncId, DBPROCESS **dbproc, char *server, char *database, char *user, char
*password, char *app, int *spid)
        {
            LOGINREC      *login;
            PECBINFO      pEcbInfo;

            //set local msg proc for login record
            //attach pECB record

            //this is necessary as dlib provides no way to pass user data in a login
structure. So until
            //there is an allocated dbproc we need to use a static which means that
login attempt must
            //be serialized.

            gpECB = pECB;

            login = dblogin();
            if ( !*user )
                DBSETLUSER(login, "sa");
            else
                DBSETLUSER(login, user);

            DBSETLPWD(login, password);
            DBSETLHOST(login, app);

            // Do not set the packet size. Use the size set up in SQL Server.
            // DBSETLPACKET(login, (unsigned short)DEFCLPACKSIZE);

            // This can potentially cut down on data conversion
            DBSETLVERSION(login, DBVER60);

            if ((*dbproc = dbopen(login, server )) == NULL)
                return TRUE;

            //set pECB data into dbproc
            pEcbInfo = (PECBINFO)malloc(sizeof(PECBINFO));
            pEcbInfo->bDeadlock = FALSE;
            pEcbInfo->pECB = pECB;
            pEcbInfo->iTermId = iTermId;
            pEcbInfo->iSyncId = iSyncId;
            dbsetuserdata(*dbproc, pEcbInfo);

            // Use the the right database
            dbuse(*dbproc, database);

            dbcmd(*dbproc, "select @@spid");
        }
    }
}

```

```

    dbsqlexec(*dbproc);
    while (dbresults(*dbproc) != NO_MORE_RESULTS)
    {
        dbbind(*dbproc, 1, SMALLBIND, (DBINT) 0, (BYTE *) spid);
        while (dbnextrow(*dbproc) != NO_MORE_ROWS)
            ;
    }
    dbcmd(*dbproc, "set nocount on");

    dbsqlexec(*dbproc);
    while (dbresults(*dbproc) != NO_MORE_RESULTS)
    {
        while (dbnextrow(*dbproc) != NO_MORE_ROWS)
            ;
    }

    //rollback transaction on abort
    dbcmd(*dbproc, "set XACT_ABORT ON");

    dbsqlexec(*dbproc);
    while (dbresults(*dbproc) != NO_MORE_RESULTS)
    {
        while (dbnextrow(*dbproc) != NO_MORE_ROWS)
            ;
    }

    return FALSE;
}

#endif

/* FUNCTION: BOOL SQLCloseConnection(EXTENSION_CONTROL_BLOCK *pECB,
DBPROCESS *dbproc)
*
* PURPOSE:      This function closes the sql connection.
*
* ARGUMENTS:   EXTENSION_CONTROL_BLOCK    *pECB    passed in structure
pointer from inetsrv.
*              DBPROCESS                  *dbproc   pointer to
DBPROCESS
*
* RETURNS:     BOOL    FALSE    if successfull
*              TRUE     if an error occurs
*
* COMMENTS:    None
*
*/

```

```

#ifndef USE_ODBC
    static BOOL SQLCloseConnection(EXTENSION_CONTROL_BLOCK *pECB,
DBPROCESS *dbproc)
    {
        if ( dbproc )
        {
            SQLFreeStmt(dbproc->hstmt, SQL_DROP);
            SQLDisconnect(dbproc->hdbc);
            SQLFreeConnect(dbproc->hdbc);
            free(dbproc);
            dbproc = NULL;
        }
        return FALSE;
    }
#else
    static BOOL SQLCloseConnection(EXTENSION_CONTROL_BLOCK *pECB,
DBPROCESS *dbproc)
    {
        if (dbclose(dbproc) == FAIL)
            return TRUE;
        return FALSE;
    }
#endif

/* FUNCTION: BOOL SQLDetectDeadlock(DBPROCESS *dbproc)
*
* PURPOSE:      This function checks to see if a sql server deadlock condition exists.
*
* ARGUMENTS:   DBPROCESS                  *dbproc
connection db process id to check
*
* RETURNS:     BOOL    FALSE    no deadlock detected
*              TRUE     deadlock
condition exists
*
* COMMENTS:    None
*
*/

BOOL SQLDetectDeadlock(DBPROCESS *dbproc)
{
    PECBINFO    pEcbInfo;

    if ( ( pEcbInfo = (PECBINFO)dbgetuserdata(dbproc) ) )
    {
        if ( pEcbInfo->bDeadlock )
        {

```

```

        pEcbInfo->bDeadlock = FALSE;
        return TRUE;
    }
}
return FALSE;
}

// Lifted from HP FDR since they did such a nice job
void TMLLog( char *format, ... )
{
    va_list args;
    char buf[4096];
    int len;
    va_start( args, format );
    _strtime( buf );
    strcat( buf, " " );
    len = strlen( buf );
    (void)_vsprintf( buf+ len, sizeof( buf ) - len - 1, format, args);
    buf[sizeof( buf ) - 1]= '\0';
    va_end( args );
    userlog( buf );
}

/* FUNCTION: void UtilStrCpy(char *pDest, char *pSrc, int n)
 *
 * PURPOSE:      This function copies n characters from string pSrc to pDst and places a
 *               null character at the end of the destination string.
 *
 * ARGUMENTS:   char          *pDest  destination string pointer
 *               char          *pSrc   source
 *               int           n       number of characters to copy
 *
 * RETURNS:     None
 *
 * COMMENTS:    Unlike strncpy this function ensures that the result string is
 *               always null terminated.
 */

static void UtilStrCpy(char *pDest, char *pSrc, int n)
{
    strncpy(pDest, pSrc, n);
    pDest[n] = '\0';

    return;
}

```

```

/* FUNCTION: int err_handler(DBPROCESS *dbproc, int severity, int dberr, int oserr, char
 *dberrstr, char *oserrstr)
 *
 * PURPOSE:      This function handles DB-Library errors
 *
 * ARGUMENTS:   DBPROCESS          *dbproc
 *               DBPROCESS id pointer
 *               int                 severity
 *               severity of error
 *               int                 dberr
 *               error id
 *               int                 oserr
 *               operating system specific error code
 *               char                *dberrstr
 *               printable error description of dberr
 *               char                *oserrstr
 *               printable error description of oserr
 *
 * RETURNS:     int                 INT_CONTINUE
 *               continue if error is SQLETIME else INT_CANCEL action
 *
 * COMMENTS:    None
 */

int err_handler(DBPROCESS *dbproc, int severity, int dberr, int oserr, char *dberrstr, char
*oserrstr)
{
    PECBINFO
    EXTENSION_CONTROL_BLOCK *pECB;
    FILE
    SYSTEMTIME
    char
    int
    int
    pEcbInfo;
    *fp;
    systemTime;
    szTmp[256];
    iTermId;
    iSyncId;

    pEcbInfo = NULL;

    if ((dbproc == NULL) || (DBDEAD(dbproc)))
    {
        TMLLog("DBPROC is invalid");
        return INT_CANCEL;
    }

    if ( !(pEcbInfo = (PECBINFO)dbgetuserdata(dbproc)) )
    {

```

```

        pECB = gpECB;
        iTermId = 0;
        iSynclId = 0;
    }
    else
    {
        pECB = pEcblInfo->pECB;
        iTermId = pEcblInfo->iTermId;
        iSynclId = pEcblInfo->iSynclId;
    }

    if ( pEcblInfo && pEcblInfo->bFailed )
    {
        bError == FALSE;
        return INT_CANCEL;
    }

    if ( oserr != DBNOERR )
    {
        TMLog("DBLIB Error %s", oserrstr);
        if ( pEcblInfo )
        {
            pEcblInfo->bFailed = TRUE;
            bError = TRUE;
        }

        GetLocalTime(&systemTime);
        fp = fopen(szErrorLogPath, "ab");

        sprintf(szTmp, "ErrorHandler: DBLIB(%d): %s", oserr, oserrstr);

        TMLog("%2.2d/%2.2d/%2.2d %2.2d:%2.2d:%2.2d\n\n%s\n\n",
            systemTime.wYear, systemTime.wMonth, systemTime.wDay,
            systemTime.wHour, systemTime.wMinute,
            systemTime.wSecond,
            szTmp);

        fclose(fp);
    }

    return INT_CANCEL;
}

```

/* FUNCTION: int msg_handler(DBPROCESS *dbproc, DBINT msgno, int msgstate, int severity, char *msgtext)

* PURPOSE: This function handles DB-Library SQL Server error messages

```

*
* ARGUMENTS:  DBPROCESS          *dbproc
              DBPROCESS id pointer
              message number      DBINT          msgno
              message state        int           msgstate
              message severity     int           severity
              printable message description char      *msgtext
*
* RETURNS:    int                INT_CONTINUE
              continue if error is SQLETIME else INT_CANCEL action
*
              INT_CANCEL          cancel operation
*
* COMMENTS:   This function also sets the dead lock dbproc variable if necessary.
*/

int msg_handler(DBPROCESS *dbproc, DBINT msgno, int msgstate, int severity, char
*msgtext)
{
    PECBINFO          pEcblInfo;
    EXTENSION_CONTROL_BLOCK *pECB;
    FILE              *fp;
    SYSTEMTIME        systemTime;
    char               szTmp[256];
    int               iTermId;
    int               iSynclId;

    if ( !(pEcblInfo = (PECBINFO)dbgetuserdata(dbproc)) )
    {
        pECB = gpECB;
        iTermId = 0;
        iSynclId = 0;
    }
    else
    {
        pECB = pEcblInfo->pECB;
        iTermId = pEcblInfo->iTermId;
        iSynclId = pEcblInfo->iSynclId;
    }

    if ( ( msgno == 5701 ) || ( msgno == 2528 ) || ( msgno == 5703 ) || ( msgno == 6006 ) )
        return INT_CONTINUE;
}

```

```

// deadlock message
if (msgno == 1205)
{
    // set the deadlock indicator
    if ( pEcblInfo )
        pEcblInfo->bDeadlock = TRUE;
    else
        TMLog("Error, dbgetuserdata returned NULL.");
    return INT_CONTINUE;
}
if ( pEcblInfo && pEcblInfo->bFailed )
{
    TMLog("SQL Error ");
    return INT_CANCEL;
}

if (msgno == 0)
    return INT_CONTINUE;
else
{
    TMLog("MsgHandler: SQL Error %s", msgtext);

    if ( pEcblInfo )
        pEcblInfo->bFailed = TRUE;

    bError = TRUE;

    sprintf(szTmp, "Error: SQLSVR(%d): %s", msgno, msgtext);

    TMLog("%2.2d/%2.2d/%2.2d %2.2d:%2.2d:%2.2d\n\n%s\n\n\n",
        systemTime.wYear, systemTime.wMonth, systemTime.wDay,
        systemTime.wHour, systemTime.wMinute,
        systemTime.wSecond,
        szTmp);
    }
    return INT_CANCEL;
}

/* FUNCTION: BOOL GetParameters(int argc, char *argv[])
*
* PURPOSE:      This function parses the command line passed in to the delivery executable,
initializing
*
*               and filling in global variable parameters.
*
* ARGUMENTS:   int          argc      number of command line arguments passed
to delivery

```

```

*
*               char      *argv[]   array of command line arguments
pointers
*
* RETURNS:      BOOL      FALSE   parameter read successful
*               TRUE      user has requested
parameter information screen be displayed.
*
* COMMENTS:     None
*
*/

static BOOL GetParameters(int argc, char *argv[])
{
    int i;

    szServer[0]          = 0;
    szPassword[0]        = 0;
    bFlush               = FALSE;
    strcpy(szDatabase, "tpcc");
    strcpy(szUser, "sa");

    for(i=0; i<argc; i++)
    {
        if ( argv[i][0] == '-' || argv[i][0] == '/' )
        {
            switch(argv[i][1])
            {
                case 'S':
                case 's':
                    strcpy(szServer, argv[i]+2);
                    break;

                case 'V':
                case 'v':
                    verbose = TRUE;
                    break;

                case '?':
                    return TRUE;
            }
        }
    }

    return FALSE;
}

/* FUNCTION: void PrintParameters(void)
*
* PURPOSE:      This function displays the supported command line flags.
*
* ARGUMENTS:     None

```

```

*
* RETURNS:          None
*
* COMMENTS:       None
*/

static void PrintParameters(void)
{
    TMLog("Performance Tuning Corporation Tuxedo Kit");
    TMLog(" www.perftuning.com (281) 251-3495 ");
    TMLog("NewOrder: -S Server [-v (verbose)]");
    TMLog("NewOrder: Server %s", szServer);
}

```

A.10 payment.c

```

/* FILE:          PAYMENT.C
*
* Based on:Microsoft TPC-C Kit Ver. 3.00.000
*
* Copyright Microsoft, 1996
* Copyright Performance Tuning Corporation, 1997
*
* PURPOSE:       New Order Tuxedo Server.
* Author:        Philip Durr
*                philipdu@Microsoft.com
*
* MODIFIED      Changed for modularity and to allow for the Tuxedo TM
*
* Author:        Edward Whalen
*                Performance Tuning Corporation
*                ewhalen@perftuning.com
*/

#include <windows.h>
#include <process.h>
#include <stdio.h>
#include <stdarg.h>
#include <malloc.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <sys\timeb.h>
#include <io.h>

#define DBNTWIN32

```

```

#include <sqlfront.h>
#include <sqldb.h>

#include "trans.h" //tpckit transaction
header contains definations of structures specific to TPC-C
#include "httpext.h" //ISAPI DLL information header

#include "tpcc.h" //this dlls specific
structure, value e.t. header.

#include <tmenv.h>
#include <xa.h>
#include <atmi.h>

BOOL bLog = FALSE;
BOOL bFlush; //Flush
delivery log info when written.
BOOL verbose = FALSE;
BOOL bError = FALSE;

int iThreads = 5;
int iMaxWareHouses = 500;
int iDelayMs = 100;
short iMaxConnections = (short)1;
short iDeadlockRetry = (short)3;

DBPROCESS *pdbproc;

char szServer[32];
//SQL server name
char szDatabase[32];
//tpcc database name
char szUser[32];
//user name
char szPassword[32];
//user password
int spld;

#ifdef LOCAL_ALLOC
PAYMENT_DATA PaymentData;
#else
TUX_DATA TuxData;
#endif
TERM Term;

static charszTpccLogPath[256]; //path to html log file if logging turned on in registry.
static char szErrorLogPath[256]; //path to error log file.

```

```

static CRITICAL_SECTION CriticalSection;
static CRITICAL_SECTION ErrorLogCriticalSection;
static EXTENSION_CONTROL_BLOCK *gpECB = NULL;
static int bTpccExit;
//exit delivery disconnect loop as dll exiting.

extern void TMLog();
extern BOOL SQLInit();
extern void UtilStrCpy();
extern void UtilStrCpy();
extern BOOL SQLOpenConnection();
extern BOOL SQLCloseConnection();
extern BOOL SQLDetectDeadlock();

/* FUNCTION: tpsvrinit ( int argc, char *argv[] )
 *
 * PURPOSE: Initialize the Server to Database connection.
 *
 * RETURNS: int 0 Success
 *          -1 Failure
 *
 * COMMENTS: None
 */

int tpsvrinit ( int argc, char *argv[] )
{
    if ( GetParameters(argc, argv) )
    {
        PrintParameters();
        return -1;
    }

    if ( verbose )
        TMLog("TPSVRINIT: Payment: Server %s, Database %s, User %s,
Password %s, Flush %d.",
            szServer, szDatabase, szUser, szPassword, bFlush);

    if ( ! SQLInit() )
    {
        TMLog( "PAYMENT: SQLInit Failed" );
        return -1;
    }
}

```

```

    if ( SQLOpenConnection ( NULL, 0,0, &pdbproc, szServer, szDatabase, szUser,
szPassword, szDatabase, &spld) )
    {
        TMLog ( "PAYMENT: SQLOpenConnection Failed" );
        dbexit();
        return -1;
    }
    return 0;
}

/* FUNCTION: tpsvrdone ( void )
 *
 * PURPOSE: Initialize the Server to Database connection.
 *
 * RETURNS: int 0 Success
 *          -1 Failure
 *
 * COMMENTS: None
 */

void tpsvrdone ( void )
{
    SQLCloseConnection( NULL, pdbproc);
    dbexit();
}

/* FUNCTION: PAYMENT ( TPSVCINFO *rqst )
 *
 * PURPOSE: Process a Payment request.
 *
 * RETURNS: int 0 Success
 *          -1 Failure
 *
 * COMMENTS: None
 */

void PAYMENT ( TPSVCINFO *rqst )
{
    PECBINFO pECBInfo = dbgetuserdata(pdbproc);
    int size = rqst->len;

#ifdef LOCAL_ALLOC
    memcpy(&PaymentData, rqst->data, size);
}

```



```

    if (verbose )
    {
        TMLog(" PAYMENT: w_id %d ", PaymentData.w_id);
        TMLog(" PAYMENT: d_id %d ", PaymentData.d_id);
    }

    bError = FALSE;

    PaymentData.retval = SQLPayment( NULL, 0, 0, pdbproc, &PaymentData,
iDeadlockRetry);

    if (bError == TRUE)
        PaymentData.retval = -1;

    if (verbose )
        TMLog(" PAYMENT: Return Value %d", PaymentData.retval);

#else
    memcpy( rqst->data, &PaymentData, size);
    memcpy(&TuxData, rqst->data, size);

    if (verbose )
    {
        TMLog(" PAYMENT: w_id %d ", TuxData.PaymentData.w_id);
        TMLog(" PAYMENT: d_id %d ", TuxData.PaymentData.d_id);
    }

    bError = FALSE;

    TuxData.PaymentData.retval = SQLPayment( NULL, 0, 0, pdbproc,
&TuxData.PaymentData, iDeadlockRetry);

    if (bError == TRUE)
        TuxData.PaymentData.retval = -1;

    if (verbose )
        TMLog(" PAYMENT: Return Value %d error %d",
TuxData.PaymentData.retval, TuxData.PaymentData.error);

    memcpy( rqst->data, &TuxData, size);
#endif
    tpreturn( TPSUCCESS, 0, rqst->data, size, 0);
}

/* FUNCTION: int SQLPayment(EXTENSION_CONTROL_BLOCK *pECB, int iTermId, int
iSynclId, DBPROCESS *dbproc, PAYMENT_DATA *pPayment, short deadlock_retry)
*
* PURPOSE:      This function handles the payment transaction.

```

```

*
* ARGUMENTS:  EXTENSION_CONTROL_BLOCK      *pECB
              passed in structure pointer from inetsrv.
*
*           iTermId          int
              terminal id of browser
*
*           iSynclId        int
              sync id of browser
*
*           *dbproc         DBPROCESS
              connection db process id
*
*           PAYMENT_DATA    *pPayment
              pointer to payment input/output data structure
*
*           deadlock_retry  short
              deadlock retry count
*
* RETURNS:    int      TRUE      success
*
*           -1          max
              deadlocked reached
*
* COMMENTS:   None
*
*/

static int SQLPayment(EXTENSION_CONTROL_BLOCK *pECB, int iTermId, int iSynclId,
DBPROCESS *dbproc, PAYMENT_DATA *pPayment, short deadlock_retry)
{
    RETCODE          rc;
    int              tryit;
    char             printbuf[26];
    // BOOL          by_name;
    DBDATETIME      datetime;
    BYTE            *pData;
    PECBINFO        pEcbInfo;

    if ( (pEcbInfo = (PECBINFO)dbgetuserdata(dbproc)) )
    {
        pEcbInfo->pECB = pECB;
        pEcbInfo->bFailed = FALSE;
        pEcbInfo->iTermId = iTermId;
        pEcbInfo->iSynclId = iSynclId;
    }

    pPayment->num_deadlocks = 0;

    // if (pPayment->c_id == 0)
    //     by_name = TRUE;
    // else
    //     by_name = FALSE;
    //

```

```

for (tryit=0; tryit < deadlock_retry; tryit++)
{
    if (dbrpcinit(dbproc, "tpcc_payment", 0) == SUCCEED)
    {
        dbrpcparam(dbproc, NULL, 0, SQLINT2, -1, -1, (BYTE *)
&pPayment->w_id);
        dbrpcparam(dbproc, NULL, 0, SQLINT2, -1, -1, (BYTE *)
&pPayment->c_w_id);
        dbrpcparam(dbproc, NULL, 0, SQLFLT8, -1, -1, (BYTE *)
&pPayment->h_amount);
        dbrpcparam(dbproc, NULL, 0, SQLINT1, -1, -1, (BYTE *)
&pPayment->d_id);
        dbrpcparam(dbproc, NULL, 0, SQLINT1, -1, -1, (BYTE *)
&pPayment->c_d_id);
        dbrpcparam(dbproc, NULL, 0, SQLINT4, -1, -1, (BYTE *)
&pPayment->c_id);
        if (pPayment->c_id == 0)
        {
            dbrpcparam(dbproc, NULL, 0, SQLCHAR, -1,
strlen(pPayment->c_last), pPayment->c_last);
        }
        if (dbrpcexec(dbproc) == SUCCEED)
        {
            while (((rc = dbresults(dbproc)) != NO_MORE_RESULTS) &&
(rc != FAIL))
            {
                if (DBROWS(dbproc) && (dbnumcols(dbproc) == 27))
                {
                    while (((rc = dbnextrow(dbproc)) !=
NO_MORE_ROWS) && (rc != FAIL))
                    {
                        if(pData=dbdata(dbproc, 1))
                            pPayment->c_id =
*((DBINT *) pData);
                        if(pData=dbdata(dbproc, 2))
                            UtilStrCpy(pPayment-
>c_last, pData, dbdatlen(dbproc, 2));
                        if(pData=dbdata(dbproc, 3))
                        {
                            datetime =
dbdatecrack(dbproc,
*((DBDATETIME *) pData);
                            &pPayment->h_date, &datetime);
                        }
                        if(pData=dbdata(dbproc, 4))
                            UtilStrCpy(pPayment-
>w_street_1, pData, dbdatlen(dbproc, 4));
                        if(pData=dbdata(dbproc, 5))
                            UtilStrCpy(pPaymer
>w_street_2, pData, dbdatlen(dbproc, 5));
                        if(pData=dbdata(dbproc, 6))
                            UtilStrCpy(pPaymer
>w_city, pData, dbdatlen(dbproc, 6));
                        if(pData=dbdata(dbproc, 7))
                            UtilStrCpy(pPaymer
>w_state, pData, dbdatlen(dbproc, 7));
                        if(pData=dbdata(dbproc, 8))
                            UtilStrCpy(pPaymer
>w_zip, pData, dbdatlen(dbproc, 8));
                        if(pData=dbdata(dbproc, 9))
                            UtilStrCpy(pPaymer
>d_street_1, pData, dbdatlen(dbproc, 9));
                        if(pData=dbdata(dbproc, 10))
                            UtilStrCpy(pPaymer
>d_street_2, pData, dbdatlen(dbproc, 10));
                        if(pData=dbdata(dbproc, 11))
                            UtilStrCpy(pPaymer
>d_city, pData, dbdatlen(dbproc, 11));
                        if(pData=dbdata(dbproc, 12))
                            UtilStrCpy(pPaymer
>d_state, pData, dbdatlen(dbproc, 12));
                        if(pData=dbdata(dbproc, 13))
                            UtilStrCpy(pPaymer
>d_zip, pData, dbdatlen(dbproc, 13));
                        if(pData=dbdata(dbproc, 14))
                            UtilStrCpy(pPaymer
>c_first, pData, dbdatlen(dbproc, 14));
                        if(pData=dbdata(dbproc, 15))
                            UtilStrCpy(pPaymer
>c_middle, pData, dbdatlen(dbproc, 15));
                        if(pData=dbdata(dbproc, 16))
                            UtilStrCpy(pPaymer
>c_street_1, pData, dbdatlen(dbproc, 16));
                        if(pData=dbdata(dbproc, 17))
                            UtilStrCpy(pPaymer
>c_street_2, pData, dbdatlen(dbproc, 17));
                        if(pData=dbdata(dbproc, 18))
                            UtilStrCpy(pPaymer
>c_city, pData, dbdatlen(dbproc, 18));
                        if(pData=dbdata(dbproc, 19))
                            UtilStrCpy(pPaymer
>c_state, pData, dbdatlen(dbproc, 19));
                        if(pData=dbdata(dbproc, 20))
                            UtilStrCpy(pPaymer
>c_zip, pData, dbdatlen(dbproc, 20));
                    }
                }
            }
        }
    }
}

```

```

        if(pData=dbdata(dbproc, 21))
            UtilStrCpy(pPayment-
>c_phone, pData, dbdatlen(dbproc, 21));
        if(pData=dbdata(dbproc, 22))
        {
            datetime =
                dbdatecrack(dbproc,
        }
        if(pData=dbdata(dbproc, 23))
            UtilStrCpy(pPayment-
>c_credit, pData, dbdatlen(dbproc, 23));
        if(pData=dbdata(dbproc, 24))
            pPayment-
>c_credit_lim = (*(DBFLT8 *) pData);
        if(pData=dbdata(dbproc, 25))
            pPayment->c_discount
        = (*(DBFLT8 *) pData);
        if(pData=dbdata(dbproc, 26))
            pPayment->c_balance
        = (*(DBFLT8 *) pData);
        if(pData=dbdata(dbproc, 27))
            UtilStrCpy(pPayment-
>c_data, pData, dbdatlen(dbproc, 27));
    }
}
if (SQLDetectDeadlock(dbproc))
{
    pPayment->num_deadlocks++;
    sprintf(printbuf,"deadlock: retry: %d",pPayment-
>num_deadlocks);
    Sleep(DEADLOCKWAIT*tryit);
}
else
{
    if ( pPayment->c_id == 0 )
    {
        strcpy(pPayment->execution_status,"Invalid Customer
id,name.");
        pPayment->error=ERR_NOSUCH_CUSTOMER;
        TMLog(" PAYMENT: No such customer ");
        return 0;
    }
    else

```

```

        strcpy(pPayment->execution_status,"Transaction
committed.");
        return TRUE;
    }
}
// If we reached here, it means we quit after MAX_RETRY deadlocks
strcpy(pPayment->execution_status,"Hit deadlock max. ");
pPayment->error=ERR_TYPE_DEADLOCK;
return -1; //"deadlock max retry reached!"
}
/*
 * Common Code for all Servers
 */
/* FUNCTION: BOOL SQLInit()
 *
 * PURPOSE: This function initializes SQL Server for later use.
 *
 * RETURNS: BOOL FALSE if successfull
           TRUE if an error occurs an
connection cannot be established.
 *
 * COMMENTS: None
 */
BOOL SQLInit ()
{
    dbinit();
    if ( dbgetmaxprocs() < iMaxConnections )
    {
        if ( dbsetmaxprocs(iMaxConnections) == FAIL )
        {
            //set for fail error message whenHttpExtensionProc() is called
            //at this point we don't have a pECB so no way to show error
            iMaxConnections = -1;
        }
    }
    // install error and message handlers
    dbmsghandle((DBMSGHANDLE_PROC)msg_handler);
    dberrhandle((DBERRHANDLE_PROC)err_handler);

```

```

        return TRUE;
    }

/* FUNCTION: BOOL SQLOpenConnection(EXTENSION_CONTROL_BLOCK *pECB, int
iTermId, int iSyncl, DBPROCESS **dbproc, char *server, char *database, char *user, char
*password, char *app, int *spid, long *pack_size)
*
* PURPOSE:      This function opens the sql connection for use.
*
* ARGUMENTS:   EXTENSION_CONTROL_BLOCK *pECB passed in structure
pointer from inetsrv.
*
*              int iTermId
terminal id of browser
*
*              int iSyncl
sync id of browser
*
*              DBPROCESS **dbproc pointer to
returned DBPROCESS
*
*              char *server
SQL server name
*
*              char *database
SQL server
*
*              char *user
user name
*
*              char *password
user password
*
*              char *app
pointer to returned application array
*
*              int *spid
pointer to returned spid
*
*              long *pack_size
pointer to returned default pack size
*
* RETURNS:     BOOL FALSE if successfull
*              TRUE  if an error occurs
*
* COMMENTS:   None
*/

#ifdef USE_ODBC
static BOOL SQLOpenConnection(EXTENSION_CONTROL_BLOCK *pECB, int
iTermId, int iSyncl, DBPROCESS **dbproc, char *server, char *database, char *user, char
*password, char *app, int *spid, long *pack_size)
{

```

```

    RETCODE rc;
    char buffer[30];

    *dbproc = (DBPROCESS *)malloc(sizeof(DBPROCESS));
    if (!*dbproc)
        return TRUE;

    //set pECB data into dbproc
    (*dbproc)->bDeadlock = FALSE;
    (*dbproc)->bFailed = FALSE;
    (*dbproc)->pECB = pECB;
    (*dbproc)->iTermId = iTermId;
    (*dbproc)->iSyncl = iSyncl;

    if ( SQLAllocConnect(henv, &(*dbproc)->hdbc) == SQL_ERROR )
        return TRUE;

    if ( SQLSetConnectOption((*dbproc)->hdbc, SQL_PACKET_SIZE,
pack_size) == SQL_ERROR )
        return TRUE;

    rc = SQLConnect((*dbproc)->hdbc, server, SQL_NTS, user, SQL_NTS,
password, SQL_NTS);
    if (rc != SQL_SUCCESS && rc != SQL_SUCCESS_WITH_INFO)
        return TRUE;
    rc = SQLAllocStmnt((*dbproc)->hdbc, &(*dbproc)->hstmt);
    if (rc == SQL_ERROR)
        return TRUE;

    sprintf(buffer,"use %s", Client->database);

    rc = SQLExecDirect((*dbproc)->hstmt, buffer, SQL_NTS);
    if (rc != SQL_SUCCESS && rc != SQL_SUCCESS_WITH_INFO)
        return TRUE;

    SQLFreeStmnt((*dbproc)->hstmt, SQL_CLOSE);
    sprintf(buffer,"set nocount on");
    rc = SQLExecDirect((*dbproc)->hstmt, buffer, SQL_NTS);
    if (rc != SQL_SUCCESS && rc != SQL_SUCCESS_WITH_INFO)
        return TRUE;
    SQLFreeStmnt((*dbproc)->hstmt, SQL_CLOSE);

    sprintf(buffer,"select @@spid");

    rc = SQLExecDirect((*dbproc)->hstmt, buffer, SQL_NTS);
    if (rc != SQL_SUCCESS && rc != SQL_SUCCESS_WITH_INFO)
        return TRUE;

```

```

        if ( SQLBindCol((*dbproc)->hstmt, 1, SQL_C_SSHORT, &((*dbproc)->spid,
0, NULL) == SQL_ERROR )
            return TRUE;

        if ( SQLFetch((*dbproc)->hstmt) == SQL_ERROR )
            return TRUE;

        SQLFreeStmt((*dbproc)->hstmt, SQL_CLOSE);

        return FALSE;
    }

#else

    static BOOL SQLOpenConnecton(EXTENSION_CONTROL_BLOCK *pECB, int
iTermId, int iSyncId, DBPROCESS **dbproc, char *server, char *database, char *user, char
*password, char *app, int *spid)
    {
        LOGINREC      *login;
        PECBINFO      pEcbInfo;

        //set local msg proc for login record
        //attach pECB record

        //this is necessary as dblib provides no way to pass user data in a login
structure. So until
        //there is an allocated dbproc we need to use a static which means that the
login attempt must
        //be serialized.

        gpECB = pECB;

        login = dblogin();
        if ( !*user )
            DBSETLUSER(login, "sa");
        else
            DBSETLUSER(login, user);

        DBSETLPWD(login, password);
        DBSETLHOST(login, app);

        // Do not set the packet size. Use the size set up in SQL Server.
        // DBSETLPACKET(login, (unsigned short)DEFCLPACKSIZE);

        // This can potentially cut down on data conversion
        DBSETLVERSION(login, DBVER60);
    }

```

```

if ((*dbproc = dbopen(login, server )) == NULL)
    return TRUE;

//set pECB data into dbproc
pEcbInfo = (PECBINFO)malloc(sizeof(ECBINFO));
pEcbInfo->bDeadlock = FALSE;
pEcbInfo->pECB = pECB;
pEcbInfo->iTermId = iTermId;
pEcbInfo->iSyncId = iSyncId;
dbsetuserdata(*dbproc, pEcbInfo);

// Use the the right database
dbuse(*dbproc, database);

dbcmd(*dbproc, "select @@spid");

dbsqlexec(*dbproc);
while (dbresults(*dbproc) != NO_MORE_RESULTS)
{
    dbbind(*dbproc, 1, SMALLBIND, (DBINT) 0, (BYTE *) spid);
    while (dbnextrow(*dbproc) != NO_MORE_ROWS)
        ;
}
dbcmd(*dbproc, "set nocount on");

dbsqlexec(*dbproc);
while (dbresults(*dbproc) != NO_MORE_RESULTS)
{
    while (dbnextrow(*dbproc) != NO_MORE_ROWS)
        ;
}

//rollback transaction on abort
dbcmd(*dbproc, "set XACT_ABORT ON");

dbsqlexec(*dbproc);
while (dbresults(*dbproc) != NO_MORE_RESULTS)
{
    while (dbnextrow(*dbproc) != NO_MORE_ROWS)
        ;
}

return FALSE;
}

```



```

*
*          char          *pSrc  source
string pointer
*          int          n
*          number of characters to copy
* RETURNS:      None
* COMMENTS:    Unlike strncpy this function ensures that the result string is
*              always null terminated.
*/

static void UtilStrCpy(char *pDest, char *pSrc, int n)
{
    strncpy(pDest, pSrc, n);
    pDest[n] = '\0';

    return;
}

/* FUNCTION: int err_handler(DBPROCESS *dbproc, int severity, int dberr, int oserr, char
 *dberrstr, char *oserrstr)
*
* PURPOSE:     This function handles DB-Library errors
*
* ARGUMENTS:  DBPROCESS          *dbproc
*              DBPROCESS id pointer
*              int                severity
*              severity of error
*              int                dberr
*              error id
*              int                oserr
*              operating system specific error code
*              char               *dberrstr
*              printable error description of dberr
*              char               *oserrstr
*              printable error description of oserr
*
* RETURNS:    int                INT_CONTINUE
*              continue if error is SQLETIME else INT_CANCEL action
*
* COMMENTS:  None
*/

int err_handler(DBPROCESS *dbproc, int severity, int dberr, int oserr, char *dberrstr, char
*oserrstr)

```

```

{
    PECBINFO          pEcbInfo;
    EXTENSION_CONTROL_BLOCK *pECB;
    FILE              *fp;
    SYSTEMTIME        systemTime;
    char              szTmp[256];
    int               iTermId;
    int               iSyncId;

    pEcbInfo = NULL;

    if ((dbproc == NULL) || (DBDEAD(dbproc)))
    {
        TMLog("DBPROC is invalid");
        return INT_CANCEL;
    }

    if ( !(pEcbInfo = (PECBINFO)dbgetuserdata(dbproc)) )
    {
        pECB = gpECB;
        iTermId = 0;
        iSyncId = 0;
    }
    else
    {
        pECB = pEcbInfo->pECB;
        iTermId = pEcbInfo->iTermId;
        iSyncId = pEcbInfo->iSyncId;
    }

    if ( pEcbInfo && pEcbInfo->bFailed )
    {
        bError == FALSE;
        return INT_CANCEL;
    }

    if ( oserr != DBNOERR )
    {
        TMLog("DBLIB Error %s", oserrstr);
        if ( pEcbInfo )
        {
            pEcbInfo->bFailed = TRUE;
            bError = TRUE;
        }

        GetLocalTime(&systemTime);
        fp = fopen(szErrorLogPath, "ab");
    }
}

```

```

        sprintf(szTmp, "ErrorHandler: DBLIB(%d): %s", oserr, oserrstr);

        TMLog("%2.2d/%2.2d/%2.2d %2.2d:%2.2d:%2.2d\n\n%s\n\n",
            systemTime.wYear, systemTime.wMonth, systemTime.wDay,
            systemTime.wHour, systemTime.wMinute,
            systemTime.wSecond,
            szTmp);

        fclose(fp);
    }

    return INT_CANCEL;
}

```

/* FUNCTION: int msg_handler(DBPROCESS *dbproc, DBINT msgno, int msgstate, int severity, char *msgtext)

* PURPOSE: This function handles DB-Library SQL Server error messages

* ARGUMENTS: DBPROCESS *dbproc
 DBPROCESS id pointer
 DBINT msgno
 message number
 int msgstate
 message state
 int severity
 message severity
 char *msgtext
 printable message description

* RETURNS: int INT_CONTINUE
 continue if error is SQLETIME else INT_CANCEL action

* INT_CANCEL cancel operation

* COMMENTS: This function also sets the dead lock dbproc variable if necessary.

*/

```

int msg_handler(DBPROCESS *dbproc, DBINT msgno, int msgstate, int severity, char *msgtext)
{
    PECBINFO pEcbInfo;
    EXTENSION_CONTROL_BLOCK *pECB;
    FILE *fp;
    SYSTEMTIME systemTime;
    char szTmp[256];

```

```

    int iTermId;
    int iSyncId;

    if ( !(pEcbInfo = (PECBINFO)dbgetuserdata(dbproc)) )
    {
        pECB = gpECB;
        iTermId = 0;
        iSyncId = 0;
    }
    else
    {
        pECB = pEcbInfo->pECB;
        iTermId = pEcbInfo->iTermId;
        iSyncId = pEcbInfo->iSyncId;
    }

    if ( (msgno == 5701) || (msgno == 2528) || (msgno == 5703) || (msgno == 6006) )
        return INT_CONTINUE;

    // deadlock message
    if (msgno == 1205)
    {
        // set the deadlock indicator
        if ( pEcbInfo )
            pEcbInfo->bDeadlock = TRUE;
        else
            TMLog("Error, dbgetuserdata returned NULL.");
        return INT_CONTINUE;
    }
    if ( pEcbInfo && pEcbInfo->bFailed )
    {
        TMLog("SQL Error ");
        return INT_CANCEL;
    }

    if (msgno == 0)
        return INT_CONTINUE;
    else
    {
        TMLog("MsgHandler: SQL Error %s", msgtext);

        if ( pEcbInfo )
            pEcbInfo->bFailed = TRUE;

        bError = TRUE;

        sprintf(szTmp, "Error: SQLSVR(%d): %s", msgno, msgtext);

```



```

        TMLog("%2.2d/%2.2d/%2.2d %2.2d:%2.2d:%2.2d\r\n\r\n%s\r\n\r\n",
            systemTime.wYear, systemTime.wMonth, systemTime.wDay,
            systemTime.wHour, systemTime.wMinute,
            systemTime.wSecond,
            szTmp);
    }
    return INT_CANCEL;
}

/* FUNCTION: BOOL GetParameters(int argc, char *argv[])
 *
 * PURPOSE:      This function parses the command line passed in to the delivery executable,
initializing
 *
 *               and filling in global variable parameters.
 *
 * ARGUMENTS:   int          argc      number of command line arguments passed
to delivery
 *               char        *argv[]   array of command line argument
pointers
 *
 * RETURNS:     BOOL        FALSE     parameter read successful
 *               TRUE        TRUE      user has requested
parameter information screen be displayed.
 *
 * COMMENTS:    None
 *
 */

static BOOL GetParameters(int argc, char *argv[])
{
    int i;

    szServer[0]          = 0;
    szPassword[0]        = 0;
    bFlush               = FALSE;
    strcpy(szDatabase, "tpcc");
    strcpy(szUser, "sa");

    for(i=0; i<argc; i++)
    {
        if ( argv[i][0] == '-' || argv[i][0] == '/' )
        {
            switch(argv[i][1])
            {
                case 'S':
                case 's':

```

```

                strcpy(szServer, argv[i]+2);
                break;
            case 'V':
            case 'v':
                verbose = TRUE;
                break;
            case '?':
                return TRUE;
        }
    }
    return FALSE;
}

/* FUNCTION: void PrintParameters(void)
 *
 * PURPOSE:      This function displays the supported command line flags.
 *
 * ARGUMENTS:    None
 *
 * RETURNS:      None
 *
 * COMMENTS:     None
 *
 */

static void PrintParameters(void)
{
    TMLog("Performance Tuning Corporation Tuxedo Kit");
    TMLog(" www.perftuning.com (281) 251-3495 ");
    TMLog("Payment: -S Server [-v (verbose)]");
    TMLog("Payment: Server %s", szServer);
}


```

A.11 orderstatus.c

```

/* FILE:          ORDERSTATUS.C
 *
 * Based on: Microsoft TPC-C Kit Ver. 3.00.000
 *
 * Copyright Microsoft, 1996
 * Copyright Performance Tuning Corporation, 1997
 *
 * PURPOSE:       New Order Tuxedo Server.
 * Author:        Philip Durr
 *                philipdu@Microsoft.com

```

```

*
* MODIFIED      Changed for modularity and to allow for the Tuxedo TM
*
* Author:       Edward Whalen
*               Performance Tuning Corporation
*               ewhalen@perftuning.com
*/

#include <windows.h>
#include <process.h>
#include <stdio.h>
#include <stdarg.h>
#include <malloc.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <sys\timeb.h>
#include <io.h>

#define DBNTWIN32
#include <sqlfront.h>
#include <sqldb.h>

#include "trans.h"                //tpckit transaction
header contains definitions of structures specific to TPC-C
#include "httpext.h"              //ISAPI DLL information header

#include "tpcc.h"                 //this dlls specific
structure, value e.t. header.

#include <tmenv.h>
#include <xa.h>
#include <atmi.h>

BOOL    bLog                    = FALSE;
BOOL    bFlush                  //Flush
delivery log info when written.
BOOL    verbose                 = FALSE;
BOOL    bError                  = FALSE;

int      iThreads                = 5;
int      iMaxWareHouses         = 500;
int      iDelayMs               = 100;
short    iMaxConnections = (short)1;
short    iDeadlockRetry        = (short)3;

DBPROCESS *pdbproc;

```

```

char      szServer[32];
        //SQL server name

char      szDatabase[32];
        //tpcc database name

char      szUser[32];
        //user name

char      szPassword[32];
        //user password

int spld;

#ifdef LOCAL_ALLOC
ORDER_STATUS_DATA OrderStatusData;
#else
TUX_DATA TuxData;
#endif
TERM Term;

static charszTpccLogPath[256];    //path to html log file if logging turned on in registry.
static char szErrorLogPath[256]; //path to error log file.

static CRITICAL_SECTION CriticalSection;
static CRITICAL_SECTION ErrorLogCriticalSection;
static EXTENSION_CONTROL_BLOCK *gpECB = NULL;
static int bTpccExit;
        //exit delivery disconnect loop as dll exiting.

extern void TMLog();
extern BOOL SQLInit();
extern void UtilStrCpy();
extern void UtilStrCpy();
extern BOOL SQLOpenConnection();
extern BOOL SQLCloseConnection();
extern BOOL SQLDetectDeadlock();

/* FUNCTION: tpsvrinit ( int argc, char *argv[] )
*
* PURPOSE:      Initialize the Server to Database connection.
*
* RETURNS:      int      0      Success
*               -1      Failure
*
* COMMENTS:    None
*/

int tpsvrinit ( int argc, char *argv[] )
{

```

```

    if ( GetParameters(argc, argv) )
    {
        PrintParameters();
        return -1;
    }

    if ( verbose )
        TMLog("TPSVRINIT: OrderStatus: Server %s, Database %s, User %s,
Password %s, Flush %d.",
            szServer, szDatabase, szUser, szPassword, bFlush);

    if ( ! SQLInit() )
    {
        TMLog( "ORDERSTATUS: SQLInit Failed" );
        return -1;
    }
    if ( SQLOpenConnection( NULL, 0, 0, &pdbproc, szServer, szDatabase, szUser,
szPassword, szDatabase, &spld) )
    {
        TMLog ( "ORDERSTATUS: SQLOpenConnection Failed" );
        dbexit();
        return -1;
    }
    return 0;
}

/* FUNCTION: tpsvrdone ( void )
*
* PURPOSE:      Initialize the Server toDatabase connection.
*
* RETURNS:      int      0      Success
*               -1      Failure
*
* COMMENTS:     None
*/

void tpsvrdone ( void )
{
    SQLCloseConnection( NULL, pdbproc);
    dbexit();
}

/* FUNCTION: ORDERSTATUS ( TPSVCINFO *rqst )
*

```

```

* PURPOSE:      Process an OrderStatus request.
*
* RETURNS:      int      0      Success
*               -1      Failure
*
* COMMENTS:     None
*/

void ORDERSTATUS ( TPSVCINFO *rqst )
{
    PECBINFO pECBInfo = dbgetuserdata(pdbproc);
    int size = rqst->len;

#ifdef LOCAL_ALLOC
    memcpy(&OrderStatusData, rqst->data, size);

    if (verbose)
    {
        TMLog(" ORDERSTATUS: w_id %d ", OrderStatusData.w_id);
        TMLog(" ORDERSTATUS: d_id %d ", OrderStatusData.d_id);
        TMLog(" ORDERSTATUS: c_id %d ", OrderStatusData.c_id);
    }

    bError = FALSE;

    OrderStatusData.retval = SQLOrderStatus( NULL, 0, 0, pdbproc, &OrderStatusData,
iDeadlockRetry);

    if (bError == TRUE)
        OrderStatusData.retval = -1;

    if ( verbose )
        TMLog(" ORDERSTATUS: Return Value %d", OrderStatusData.retval);

    memcpy( rqst->data, &OrderStatusData, size);
#else
    memcpy(&TuxData, rqst->data, size);

    if (verbose)
    {
        TMLog(" ORDERSTATUS: w_id %d ", TuxData.OrderStatusData.w_id);
        TMLog(" ORDERSTATUS: d_id %d ", TuxData.OrderStatusData.d_id);
        TMLog(" ORDERSTATUS: c_id %d ", TuxData.OrderStatusData.c_id);
    }

    bError = FALSE;

```

```

    TuxData.OrderStatusData.retval = SQLOrderStatus( NULL, 0, 0, pdbproc,
&TuxData.OrderStatusData, iDeadlockRetry);

    if (bError == TRUE)
        TuxData.OrderStatusData.retval = -1;

    if ( verbose )
        TMLLog(" ORDERSTATUS: Return Value %d error =%d",
            TuxData.OrderStatusData.retval,
TuxData.OrderStatusData.error);

    memcpy( rqst->data, &TuxData, size);
#endif
    tpreturn( TPSUCCESS, 0, rqst->data, size, 0);
}

/* FUNCTION: int SQLOrderStatus(EXTENSION_CONTROL_BLOCK *pECB, int iTermId, int
iSyncId, DBPROCESS *dbproc, ORDER_STATUS_DATA *pOrderStatus, short
deadlock_retry)
*
* PURPOSE:      This function processes the Order Status transaction.
*
* ARGUMENTS:   EXTENSION_CONTROL_BLOCK      *pECB
                passed in structure pointer from inetsrv.
*
*              int
                terminal id of browser
*              int
                sync id of browser
*              DBPROCESS
                connection db process id
*              ORDER_STATUS_DATA
                ORDER_STATUS_DATA
*              pointer to Order Status data input/output structure
*              short
                ORDER_STATUS_DATA
*              deadlock_retry count
                deadlock_retry count
*
* RETURNS:     int      -1          max deadlock reached
*              int      0          No orders found for
customer
*              int      1          Transaction
successfull
*
* COMMENTS:   None
*/

static int SQLOrderStatus(EXTENSION_CONTROL_BLOCK *pECB, int iTermId, int iSyncId,
DBPROCESS *dbproc, ORDER_STATUS_DATA *pOrderStatus, short deadlock_retry)

```

```

{
    RETCODE          rc;
    int              tryit;
    int              i;
    char             printbuf[25];
    //              BOOL          by_name;
    DBDATETIME      datetime;
    BYTE             *pData;
    PECBINFO         pEcbInfo;

    if ( ( pEcbInfo = (PECBINFO)dbgetuserdata(dbproc) )
    {
        pEcbInfo->pECB = pECB;
        pEcbInfo->bFailed = FALSE;
        pEcbInfo->iTermId = iTermId;
        pEcbInfo->iSyncId = iSyncId;
    }

    pOrderStatus->num_deadlocks = 0;
    // if (pOrderStatus->c_id == 0)
    //     by_name = TRUE;
    // else
    //     by_name = FALSE;

    for (tryit=0; tryit < deadlock_retry; tryit++)
    {
        if (dbrpcinit(dbproc, "tpcc_orderstatus", 0) == SUCCEED)
        {
            dbrpcparam(dbproc, NULL, 0, SQLINT2, -1, -1, (BYTE *)
&pOrderStatus->w_id);
            dbrpcparam(dbproc, NULL, 0, SQLINT1, -1, -1, (BYTE *)
&pOrderStatus->d_id);
            dbrpcparam(dbproc, NULL, 0, SQLINT4, -1, -1, (BYTE *)
&pOrderStatus->c_id);
            if (pOrderStatus->c_id == 0)
            {
                dbrpcparam(dbproc, NULL, 0, SQLCHAR, -1,
strlen(pOrderStatus->c_last), pOrderStatus->c_last);
            }
            if (dbrpcexec(dbproc) == SUCCEED)
            {
                while (((rc = dbresults(dbproc)) != NO_MORE_RESULTS) &&
(rc != FAIL))
                {
                    if (DBROWS(dbproc) && (dbnumcols(dbproc) == 5)
                    {
                        i=0;

```

```

while (((rc = dbnextrow(dbproc)) !=
NO_MORE_ROWS) && (rc != FAIL))
{
    if(pData=dbdata(dbproc, 1))
        pOrderStatus-
>OIOrderStatusData[i].ol_supply_w_id = (*(DBSMALLINT *) pData);
    if(pData=dbdata(dbproc, 2))
        pOrderStatus-
>OIOrderStatusData[i].ol_i_id = (*(DBINT *) pData);
    if(pData=dbdata(dbproc, 3))
        pOrderStatus-
>OIOrderStatusData[i].ol_quantity = (*(DBSMALLINT *) pData);
    if(pData=dbdata(dbproc, 4))
        pOrderStatus-
>OIOrderStatusData[i].ol_amount = (*(DBFLT8 *) pData);
    if(pData=dbdata(dbproc, 5))
    {
        datetime =
        dbdatecrack(dbproc,
        &pOrderStatus->OIOrderStatusData[i].ol_delivery_d, &datetime);
    }
    i++;
    pOrderStatus->o_ol_cnt = i;
}
else if (DBROWS(dbproc) && (dbnumcols(dbproc) ==
8))
{
    while (((rc = dbnextrow(dbproc)) !=
NO_MORE_ROWS) && (rc != FAIL))
    {
        if(pData=dbdata(dbproc, 1))
            pOrderStatus->c_id =
            (*(DBINT *) pData);
        if(pData=dbdata(dbproc, 2))
            UtilStrCpy(pOrderStatus->c_last, pData, dbdatlen(dbproc,2));
        if(pData=dbdata(dbproc, 3))
            UtilStrCpy(pOrderStatus->c_first, pData, dbdatlen(dbproc,3));
        if(pData=dbdata(dbproc, 4))
            UtilStrCpy(pOrderStatus->c_middle, pData, dbdatlen(dbproc, 4));
        if(pData=dbdata(dbproc, 5))
        {
            datetime =
            (*(DBDATETIME *) pData);

```

```

        dbdatecrack(dbproc
        &pOrderStatus->o_entry_d, &datetime);
    }
    if(pData=dbdata(dbproc, 6))
        pOrderStatus-
>o_carrier_id = (*(DBSMALLINT *) pData);
    if(pData=dbdata(dbproc, 7))
        pOrderStatus-
>c_balance = (*(DBFLT8 *) pData);
    if(pData=dbdata(dbproc, 8))
        pOrderStatus->o_id
        (*(DBINT *) pData);
    }
    if (i==0)
        return 0; /*No orders found for customer
    }
}
if (SQLDetectDeadlock(dbproc))
{
    pOrderStatus->num_deadlocks++;
    sprintf(printbuf,"deadlock: retry: %d",pOrderStatus-
>num_deadlocks);
    Sleep(DEADLOCKWAIT*tryit);
}
else
{
    if (pOrderStatus->c_id == 0 && pOrderStatus->c_last[0] == 0)
    {
        strcpy(pOrderStatus->execution_status,"Invalid
Customer id,name.");
        pOrderStatus->error=ERR_NOSUCH_CUSTOMEF
        TMLog(" ORDERSTATUS: No such customer ");
    }
    else
        strcpy(pOrderStatus->execution_status,"Transactic
committed.");
    return 1;
}
}
// If we reached here, it means we quit after MAX_RETRY deadlocks
strcpy(pOrderStatus->execution_status,"Hit deadlock max. ");
pOrderStatus->error=ERR_TYPE_DEADLOCK;
return -1; /*deadlock max retry reached!
}
/*

```

```

*      Common Code for all Servers
*/

/* FUNCTION: BOOL SQLInit()
*
* PURPOSE:      This function initializes SQL Server for later use.
*
* RETURNS:      BOOL   FALSE   if successfull
*               TRUE    if an error occurs and
connection cannot be established.
*
* COMMENTS:     None
*/
BOOL SQLInit ()
{
    dbinit();

    if ( dbgetmaxprocs() < iMaxConnections )
    {
        if ( dbsetmaxprocs(iMaxConnections) == FAIL )
        {
            //set for fail error message when HttpExtensionProc() is called
            //at this point we don't have a pECB so no way to show error
            //message.
            iMaxConnections = -1;
        }
    }

    // install error and message handlers
    dbmsghandle((DBMSGHANDLE_PROC)msg_handler);
    dberrhandle((DBERRHANDLE_PROC)err_handler);

    return TRUE;
}

/* FUNCTION: BOOL SQLOpenConnection(EXTENSION_CONTROL_BLOCK *pECB, int
iTermId, int iSynclD, DBPROCESS **dbproc, char *server, char *database, char *user, char
*password, char *app, int *spid, long *pack_size)
*
* PURPOSE:      This function opens the sqlconnection for use.
*
* ARGUMENTS:    EXTENSION_CONTROL_BLOCK *pECB passed in structure
pointer from inetsrv.

```

```

*      terminal id of browser          int          iTermId
*      sync id of browser             int          iSynclD
*      DBPROCESS **dbproc pointer to
returned DBPROCESS
*      SQL server name                char         *server
*      database                        char         *databaseSQL serv
*      user name                       char         *user
*      user password                   char         *password
*      pointer to returned application array char         *app
*      pointer to returned spid         int          *spid
*      pointer to returned default pack size long        *pack_size
*
* RETURNS:      BOOL   FALSE   if successfull
*               TRUE    if an error occurs
*
* COMMENTS:     None
*/

#ifdef USE_ODBC
    static BOOL SQLOpenConnection(EXTENSION_CONTROL_BLOCK *pECB, int
iTermId, int iSynclD, DBPROCESS **dbproc, char *server, char *database, char *user, char
*password, char *app, int *spid, long *pack_size)
    {
        RETCODE rc;
        char buffer[30];

        *dbproc = (DBPROCESS *)malloc(sizeof(DBPROCESS));
        if ( !*dbproc )
            return TRUE;

        //set pECB data into dbproc
        (*dbproc)->bDeadlock = FALSE;
        (*dbproc)->bFailed = FALSE;
        (*dbproc)->pECB = pECB;
        (*dbproc)->iTermId = iTermId;
        (*dbproc)->iSynclD = iSynclD;
    }

```

```

        if ( SQLAllocConnect(henv, &(*dbproc)->hdbc) == SQL_ERROR )
            return TRUE;

        if ( SQLSetConnectOption((*dbproc)->hdbc, SQL_PACKET_SIZE,
pack_size) == SQL_ERROR )
            return TRUE;

        rc = SQLConnect((*dbproc)->hdbc, server, SQL_NTS, user, SQL_NTS,
password, SQL_NTS);
        if (rc != SQL_SUCCESS && rc != SQL_SUCCESS_WITH_INFO)
            return TRUE;
        rc = SQLAllocStmt((*dbproc)->hdbc, &(*dbproc)->hstmt);
        if (rc == SQL_ERROR)
            return TRUE;

        sprintf(buffer,"use %s", Client->database);

        rc = SQLExecDirect((*dbproc)->hstmt, buffer, SQL_NTS);
        if (rc != SQL_SUCCESS && rc != SQL_SUCCESS_WITH_INFO)
            return TRUE;

        SQLFreeStmt((*dbproc)->hstmt, SQL_CLOSE);
        sprintf(buffer,"set nocount on");
        rc = SQLExecDirect((*dbproc)->hstmt, buffer, SQL_NTS);
        if (rc != SQL_SUCCESS && rc != SQL_SUCCESS_WITH_INFO)
            return TRUE;
        SQLFreeStmt((*dbproc)->hstmt, SQL_CLOSE);

        sprintf(buffer,"select @@spid");

        rc = SQLExecDirect((*dbproc)->hstmt, buffer, SQL_NTS);
        if (rc != SQL_SUCCESS && rc != SQL_SUCCESS_WITH_INFO)
            return TRUE;

        if ( SQLBindCol((*dbproc)->hstmt, 1, SQL_C_SSHORT, &(*dbproc)->spid,
0, NULL) == SQL_ERROR )
            return TRUE;

        if ( SQLFetch((*dbproc)->hstmt) == SQL_ERROR )
            return TRUE;

        SQLFreeStmt((*dbproc)->hstmt, SQL_CLOSE);

        return FALSE;
    }

```

```

#else

        static BOOL SQLOpenConnection(EXTENSION_CONTROL_BLOCK *pECB,int
iTermId, int iSyncId, DBPROCESS **dbproc, char *server, char *database, char *user, char
*password, char *app, int *spid)
        {
            LOGINREC      *login;
            PECBINFO      pEcbInfo;

            //set local msg proc for login record
            //attach pECB record

            //this is necessary as dblib provides no way to pass user data in a login
            structure. So until
            //there is an allocated dbproc we need to use a static which means that
            login attempt must
            //be serialized.

            gpECB = pECB;

            login = dblogin();
            if ( !*user )
                DBSETLUSER(login, "sa");
            else
                DBSETLUSER(login, user);

            DBSETLPWD(login, password);
            DBSETLHOST(login, app);

            // Do not set the packet size. Use the size set up in SQL Server.
            // DBSETLPACKET(login, (unsigned short)DEFCLPACKSIZE);

            // This can potentially cut down on data conversion
            // DBSETLVERSION(login, DBVER60);

            if ((*dbproc = dbopen(login, server )) == NULL)
                return TRUE;

            //set pECB data into dbproc
            pEcbInfo = (PECBINFO)malloc(sizeof(ECBINFO));
            pEcbInfo->bDeadlock = FALSE;
            pEcbInfo->pECB = pECB;
            pEcbInfo->iTermId = iTermId;
            pEcbInfo->iSyncId = iSyncId;
            dbsetuserdata(*dbproc, pEcbInfo);

            // Use the the right database
            dbuse(*dbproc, database);

```

```

        dbcmd(*dbproc, "select @@spid");

        dbsqlexec(*dbproc);
        while (dbresults(*dbproc) != NO_MORE_RESULTS)
        {
            dbbind(*dbproc, 1, SMALLBIND, (DBINT) 0, (BYTE *) spid);
            while (dbnextrow(*dbproc) != NO_MORE_ROWS)
                ;
        }
        dbcmd(*dbproc, "set nocount on");
        dbsqlexec(*dbproc);
        while (dbresults(*dbproc) != NO_MORE_RESULTS)
        {
            while (dbnextrow(*dbproc) != NO_MORE_ROWS)
                ;
        }

        //rollback transaction on abort
        dbcmd(*dbproc, "set XACT_ABORT ON");

        dbsqlexec(*dbproc);
        while (dbresults(*dbproc) != NO_MORE_RESULTS)
        {
            while (dbnextrow(*dbproc) != NO_MORE_ROWS)
                ;
        }

        return FALSE;
    }

#endif

/* FUNCTION: BOOL SQLCloseConnection(EXTENSION_CONTROL_BLOCK *pECB,
DBPROCESS *dbproc)
*
* PURPOSE:      This function closes the sql connection.
*
* ARGUMENTS:   EXTENSION_CONTROL_BLOCK    *pECB    passed in structure
pointer from inetsrv.
                DBPROCESS                  *dbproc   pointer to
DBPROCESS
*
* RETURNS:     BOOL    FALSE    if successfull
                TRUE     if an error occurs
*
* COMMENTS:    None
*

```

```

*/
#ifdef USE_ODBC
    static BOOL SQLCloseConnection(EXTENSION_CONTROL_BLOCK *pECB,
DBPROCESS *dbproc)
    {
        if ( dbproc )
        {
            SQLFreeStmt(dbproc->hstmt, SQL_DROP);
            SQLDisconnect(dbproc->hdbc);
            SQLFreeConnect(dbproc->hdbc);
            free(dbproc);
            dbproc = NULL;
        }
        return FALSE;
    }
#else
    static BOOL SQLCloseConnection(EXTENSION_CONTROL_BLOCK *pECB,
DBPROCESS *dbproc)
    {
        if (dbclose(dbproc) == FAIL)
            return TRUE;
        return FALSE;
    }
#endif

/* FUNCTION: BOOL SQLDetectDeadlock(DBPROCESS *dbproc)
*
* PURPOSE:      This function checks to see if a sql server deadlock condition exists.
*
* ARGUMENTS:   DBPROCESS                  *dbproc
                connection db process id to check
*
* RETURNS:     BOOL    FALSE    no deadlock detected
                TRUE     deadlock
condition exists
*
* COMMENTS:    None
*
*/

BOOL SQLDetectDeadlock(DBPROCESS *dbproc)
{
    PECBINFO      pEcbInfo;

    if ( ( pEcbInfo = (PECBINFO)dbgetuserdata(dbproc) ) )
    {
        if ( pEcbInfo->bDeadlock )

```



```

        {
            pEcbInfo->bDeadlock = FALSE;
            return TRUE;
        }
    }
    return FALSE;
}

// Lifted from HP FDR since they did such a nice job
void TMLog( char *format, ... )
{
    va_list args;
    char buf[4096];
    int len;
    va_start( args, format );
    _strtime( buf );
    strcat( buf, " ");
    len = strlen( buf );
    (void)_vsprintf( buf+ len, sizeof( buf ) - len - 1, format, args);
    buf[sizeof( buf ) - 1]= '\0';
    va_end( args );
    userlog( buf );
}

/* FUNCTION: void UtilStrCpy(char *pDest, char *pSrc, int n)
*
* PURPOSE:      This function copies n characters from string pSrc to pDst and places a
*               null character at the end of the destination string.
*
* ARGUMENTS:   char          *pDest  destination string pointer
*               char          *pSrc   source
*               string pointer
*               int           n
*               number of characters to copy
*
* RETURNS:     None
*
* COMMENTS:    Unlike strncpy this function ensures that the result string is
*               always null terminated.
*
*/

static void UtilStrCpy(char *pDest, char *pSrc, int n)
{
    strncpy(pDest, pSrc, n);
    pDest[n] = '\0';

    return;
}

```

```

}

/* FUNCTION: int err_handler(DBPROCESS *dbproc, int severity, int dberr, int oserr, char
*dberrstr, char *oserrstr)
*
* PURPOSE:      This function handles DB-Library errors
*
* ARGUMENTS:   DBPROCESS          *dbproc
*               DBPROCESS id pointer
*               int                severity
*               severity of error
*               int                dberr
*               error id
*               int                oserr
*               operating system specific error code
*               char               *dberrstr
*               printable error description of dberr
*               char               *oserrstr
*               printable error description of oserr
*
* RETURNS:     int                INT_CONTINUE
*               continue if error is SQLETIME else INT_CANCEL action
*
* COMMENTS:    None
*
*/

int err_handler(DBPROCESS *dbproc, int severity, int dberr, int oserr, char *dberrstr, char
*oserrstr)
{
    PECBINFO
    EXTENSION_CONTROL_BLOCK    *pECB;
    FILE
    SYSTEMTIME
    char
    int
    int
    pEcbInfo;
    *fp;
    systemTime;
    szTmp[256];
    iTermId;
    iSyncId;

    pEcbInfo = NULL;

    if ((dbproc == NULL) || (DBDEAD(dbproc)))
    {
        TMLog("DBPROC is invalid");
        return INT_CANCEL;
    }

    if ( !(pEcbInfo = (PECBINFO)dbgetuserdata(dbproc)) )

```

```

{
    pECB = gpECB;
    iTermId = 0;
    iSynclId = 0;
}
else
{
    pECB = pEcbInfo->pECB;
    iTermId = pEcbInfo->iTermId;
    iSynclId = pEcbInfo->iSynclId;
}

if ( pEcbInfo && pEcbInfo->bFailed )
{
    bError == FALSE;
    return INT_CANCEL;
}

if ( oserr != DBNOERR )
{
    TMLog("DBLIB Error %s", oserrstr);
    if ( pEcbInfo )
    {
        pEcbInfo->bFailed = TRUE;
        bError = TRUE;
    }

    GetLocalTime(&systemTime);
    fp = fopen(szErrorLogPath, "ab");

    sprintf(szTmp, "ErrorHandler: DBLIB(%d): %s", oserr, oserrstr);

    TMLog("%2.2d/%2.2d/%2.2d %2.2d:%2.2d:%2.2d\n\n\n%s\n\n\n",
        systemTime.wYear, systemTime.wMonth, systemTime.wDay,
        systemTime.wHour, systemTime.wMinute,
        systemTime.wSecond,
        szTmp);

    fclose(fp);
}
return INT_CANCEL;
}

```

/* FUNCTION: int msg_handler(DBPROCESS *dbproc, DBINT msgno, int msgstate, int severity, char *msgtext)

* PURPOSE: This function handles DB-Library SQL Server error message

```

*
* ARGUMENTS:  DBPROCESS          *dbproc
              DBPROCESS id pointer
*
              message number      DBINT          msgno
*
              message state        int           msgstate
*
              message severity     int           severity
*
              printable message description char          *msgtext
*
* RETURNS:    int                 INT_CONTINUE
              continue if error is SQLETIME else INT_CANCEL action
*
              INT_CANCEL          cancel operation
*
* COMMENTS:   This function also sets the dead lock dbproc variable if necessary.
*/

int msg_handler(DBPROCESS *dbproc, DBINT msgno, int msgstate, int severity, char
*msgtext)
{
    PECBINFO          pEcbInfo;
    EXTENSION_CONTROL_BLOCK *pECB;
    FILE              *fp;
    SYSTEMTIME        systemTime;
    char               szTmp[256];
    int               iTermId;
    int               iSynclId;

    if ( !(pEcbInfo = (PECBINFO)dbgetuserdata(dbproc)) )
    {
        pECB = gpECB;
        iTermId = 0;
        iSynclId = 0;
    }
    else
    {
        pECB = pEcbInfo->pECB;
        iTermId = pEcbInfo->iTermId;
        iSynclId = pEcbInfo->iSynclId;
    }

    if ( ( msgno == 5701 ) || ( msgno == 2528 ) || ( msgno == 5703 ) || ( msgno == 6006 ) )
        return INT_CONTINUE;
}

```

```

// deadlockmessage
if (msgno == 1205)
{
    // set the deadlock indicator
    if ( pEcblInfo )
        pEcblInfo->bDeadlock = TRUE;
    else
        TMLog("Error, dbgetuserdata returned NULL.");
    return INT_CONTINUE;
}
if ( pEcblInfo && pEcblInfo->bFailed )
{
    TMLog("SQL Error");
    return INT_CANCEL;
}

if (msgno == 0)
    return INT_CONTINUE;
else
{
    TMLog("MsgHandler: SQL Error %s", msgtext);

    if ( pEcblInfo )
        pEcblInfo->bFailed = TRUE;

    bError = TRUE;

    sprintf(szTmp, "Error: SQLSVR(%d): %s", msgno, msgtext);

    TMLog("%2.2d/%2.2d/%2.2d %2.2d:%2.2d:%2.2d\n\n%s\n\n",
        systemTime.wYear, systemTime.wMonth, systemTime.wDay,
        systemTime.wHour, systemTime.wMinute,
        systemTime.wSecond,
        szTmp);
    }
return INT_CANCEL;
}

/* FUNCTION: BOOL GetParameters(int argc, char *argv[])
*
* PURPOSE: This function parses the command line passed in to the delivery executable,
initializing
*
* and filling in global variable parameters.
*
* ARGUMENTS: int argc number of command line arguments passed
to delivery

```

```

*
* char *argv[] array of command line arguments
*
* pointers
*
* RETURNS: BOOL FALSE parameter read successful
*          TRUE user has requested
parameter information screen be displayed.
*
* COMMENTS: None
*
*/

static BOOL GetParameters(int argc, char *argv[])
{
    int i;

    szServer[0] = 0;
    szPassword[0] = 0;
    bFlush = FALSE;
    strcpy(szDatabase, "tpcc");
    strcpy(szUser, "sa");

    for(i=0; i<argc; i++)
    {
        if ( argv[i][0] == '-' || argv[i][0] == '/' )
        {
            switch(argv[i][1])
            {
                case 'S':
                case 's':
                    strcpy(szServer, argv[i]+2);
                    break;

                case 'V':
                case 'v':
                    verbose = TRUE;
                    break;

                case '?':
                    return TRUE;
            }
        }
    }
    return FALSE;
}

/* FUNCTION: void PrintParameters(void)
*
* PURPOSE: This function displays the supported command line flags.
*

```

```
* ARGUMENTS:  None
*
* RETURNS:    None
*
* COMMENTS:   None
*
*/
```

```
static void PrintParameters(void)
{
    TMLog("Performance Tuning Corporation Tuxedo Kit");
    TMLog(" www.perftuning.com (281) 251-3495 ");
    TMLog("OrderStatus: -S Server [-v (verbose)]");
    TMLog("OrderStatus: Server %s",      szServer);
}
}
```

A.12 stocklevel.c

```
/*      FILE:          STOCKLEVEL.C
*
*      Based on:Microsoft TPC-C Kit Ver. 3.00.000
*
*      Copyright Microsoft, 1996
*      Copyright Performance Tuning Corporation, 1997
*
*      PURPOSE:       New Order Tuxedo Server.
*      Author:        Philip Durr
*                    philipdu@Microsoft.com
*
*      MODIFIED      Changed for modularity and to allow for the Tuxedo TM
*
*      Author:       Edward Whalen
*                    Performance Tuning Corporation
*                    ewhalen@peftuning.com
*/

#include <windows.h>
#include <process.h>
#include <stdio.h>
#include <stdarg.h>
#include <malloc.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <sys\timeb.h>
#include <io.h>
```

```
#define DBNTWIN32
#include <sqlfront.h>
#include <sqldb.h>

#include "trans.h" //tpckit transaction
header contains definitions of structures specific to TPC-C
#include "httpext.h" //ISAPI DLL information header

#include "tpcc.h" //this dlls specific
structure, value e.t. header.

#include <tmenv.h>
#include <xa.h>
#include <atmi.h>

BOOL    bLog                = FALSE;
BOOL    bFlush              = FALSE; //Flush
delivery log info when written.
BOOL    verbose             = FALSE;
BOOL    bError              = FALSE;

int      iThreads           = 5;
int      iMaxWareHouses    = 500;
int      iDelayMs           = 100;
short    iMaxConnections   = (short)1;
short    iDeadlockRetry    = (short)3;

DBPROCESS *pdbproc;

char     szServer[32];
//SQL server name

char     szDatabase[32];
//tpcc database name

char     szUser[32];
//user name

char     szPassword[32];
//user password

int spld;

#ifdef LOCAL_ALLOC
STOCK_LEVEL_DATA StockLevelData;
#else
TUX_DATA TuxData;
#endif
TERM Term;

static charszTpccLogPath[256]; //path to html log file if logging turned on in registry.
```

```

static char szErrorLogPath[256];          //path to error log file

static CRITICAL_SECTION CriticalSection;
static CRITICAL_SECTION ErrorLogCriticalSection;
static EXTENSION_CONTROL_BLOCK *gpECB = NULL;
static int bTpccExit;
//exit delivery disconnect loop as dll exiting.

extern void TMLog();
extern BOOL SQLInit();
extern void UtilStrCpy();
extern void UtilStrCpy();
extern BOOL SQLOpenConnection();
extern BOOL SQLCloseConnection();
extern BOOL SQLDetectDeadlock();

/* FUNCTION: tpsvrinit ( int argc, char *argv[] )
*
* PURPOSE:      Initialize the Server to Database connection.
*
* RETURNS:      int      0      Success
*               -1      Failure
*
* COMMENTS:     None
*/

int tpsvrinit ( int argc, char *argv[] )
{
    if ( GetParameters(argc, argv) )
    {
        PrintParameters();
        return -1;
    }

    if ( verbose )
        TMLog("TPSVRINIT: StockLevel: Server %s, Database %s, User %s,
Password %s, Flush %d.",
            szServer, szDatabase, szUser, szPassword, bFlush);

    if ( ! SQLInit() )
    {
        TMLog( "STOCKLEVEL: SQLInit Failed" );
        return -1;
    }
}

```

```

        if ( SQLOpenConnection ( NULL, 0, 0, &pdbproc, szServer, szDatabase, szUser,
szPassword, szDatabase, &spld) )
        {
            TMLog ( "STOCKLEVEL: SQLOpenConnection Failed" );
            dbexit();
            return -1;
        }
        return 0;
    }

/* FUNCTION: tpsvrdone ( void )
*
* PURPOSE:      Initialize the Server to Database connection.
*
* RETURNS:      int      0      Success
*               -1      Failure
*
* COMMENTS:     None
*/

void tpsvrdone ( void )
{
    SQLCloseConnection( NULL, pdbproc);
    dbexit();
}

/* FUNCTION: STOCKLEVEL ( TPSVCINFO *rqst )
*
* PURPOSE:      Process a Stock Level request.
*
* RETURNS:      int      0      Success
*               -1      Failure
*
* COMMENTS:     None
*/

void STOCKLEVEL ( TPSVCINFO *rqst )
{
    PECBINFO pECBInfo = dbgetuserdata(pdbproc);
    int size = rqst->len;

#ifdef LOCAL_ALLOC

```

```

memcpy(&StockLevelData, rqst->data, size);

if (verbose )
{
    TMLog(" STOCKLEVEL: w_id %d ", StockLevelData.w_id);
    TMLog(" STOCKLEVEL: d_id %d ", StockLevelData.d_id);
    TMLog(" STOCKLEVEL: c_id %d ", StockLevelData.thresh_hold);
}

bError = FALSE;

StockLevelData.retval = SQLStockLevel( NULL, 0, 0, pdbproc, &StockLevelData,
iDeadlockRetry);

if (bError == TRUE)
    StockLevelData.retval = -1;

if ( verbose )
    TMLog(" STOCKLEVEL: Return Value %d", StockLevelData.retval);

memcpy( rqst->data, &StockLevelData, size);
#else
memcpy(&TuxData, rqst->data, size);

if (verbose )
{
    TMLog(" STOCKLEVEL: w_id %d ", TuxData.StockLevelData.w_id);
    TMLog(" STOCKLEVEL: d_id %d ", TuxData.StockLevelData.d_id);
    TMLog(" STOCKLEVEL: c_id %d ", TuxData.StockLevelData.thresh_hold);
}

bError = FALSE;

TuxData.StockLevelData.retval = SQLStockLevel( NULL, 0, 0, pdbproc,
&TuxData.StockLevelData, iDeadlockRetry);

if (bError == TRUE)
    TuxData.StockLevelData.retval = -1;

if ( verbose )
    TMLog(" STOCKLEVEL: Return Value %d",
TuxData.StockLevelData.retval);

memcpy( rqst->data, &TuxData, size);
#endif
tpreturn( TPSUCCESS, 0, rqst->data, size, 0);

```

```

}

/* FUNCTION: SQLStockLevel(EXTENSION_CONTROL_BLOCK *pECB, int iTermId,
iSyncl, DBPROCESS *dbproc, STOCK_LEVEL_DATA *pStockLevel, short deadlock_retry)
*
* PURPOSE:      This function handles the stock level transaction.
*
* ARGUMENTS:   EXTENSION_CONTROL_BLOCK *pECB
                passed in structure pointer from inetsrv.
*
*              int
                iTermId      terminal id of browser
*              int
                iSyncl      sync id of browser
*              DBPROCESS
                *dbproc      connection db process id
*              STOCK_LEVEL_DATA
                *pStockLevel stock level input / output data structure
*              short
                deadlock_retry  retry count if deadlocked
*
* RETURNS:     BOOL  FALSE      if successfull
                TRUE       if
deadlocked
*
* COMMENTS:    None
*
*/

static BOOL SQLStockLevel(EXTENSION_CONTROL_BLOCK *pECB, int iTermId, int iSyncl,
DBPROCESS *dbproc, STOCK_LEVEL_DATA *pStockLevel, short deadlock_retry)
{
    int          tryit;
    RETCODE      rc;
    char         printbuf[25];
    BYTE         *pData;
    PECBINFO     pEcbInfo;

    //update pECB and bFailed flag
    if ( ( pEcbInfo = (PECBINFO)dbgetuserdata(dbproc) ) )
    {
        pEcbInfo->pECB = pECB;
        pEcbInfo->bFailed = FALSE;
        pEcbInfo->iTermId = iTermId;
        pEcbInfo->iSyncl = iSyncl;
    }

    pStockLevel->num_deadlocks = 0;

```

```

for (tryit=0; tryit < dbadlock_retry; tryit++)
{
    if (dbrpcinit(dbproc, "tpcc_stocklevel", 0) == SUCCEED)
    {
        dbrpcparam(dbproc, NULL, 0, SQLINT2, -1, -1, (BYTE *)
&pStockLevel->w_id);
        dbrpcparam(dbproc, NULL, 0, SQLINT1, -1, -1, (BYTE *)
&pStockLevel->d_id);
        dbrpcparam(dbproc, NULL, 0, SQLINT2, -1, -1, (BYTE *)
&pStockLevel->thresh_hold);

        if (dbrpcexec(dbproc) == SUCCEED)
        {
            while (((rc = dbresults(dbproc)) !=
NO_MORE_RESULTS) && (rc != FAIL))
            {
                if (DBROWS(dbproc))
                {
                    while (((rc = dbnextrow(dbproc))
!= NO_MORE_ROWS) && (rc != FAIL))
                    {
                        if(pData=dbdata(dbproc, 1))
                            pStockLevel->low_stock = *((long *) pData);
                    }
                }
            }
            if (SQLDetectDeadlock(dbproc))
            {
                pStockLevel->num_deadlocks++;
                sprintf(printbuf,"deadlock: retry: %d",pStockLevel-
>num_deadlocks);
                Sleep(10 * tryit);
            }
            else
            {
                strcpy(pStockLevel->execution_status, "Transaction committed.");
                return TRUE;
            }
        }
    }

    // If we reached here, it means we quit after MAX_RETRY deadlocks
    strcpy(pStockLevel->execution_status, "Hit deadlock max. ");
    pStockLevel->error=ERR_TYPE_DEADLOCK;
    return -1;
}

```

```

}
/*
 * Common Code for all Servers
 */

/* FUNCTION: BOOL SQLInit()
 *
 * PURPOSE: This function initializes SQL Server for later use.
 *
 * RETURNS: BOOL FALSE if successfull
 *          TRUE if an error occurs an
connection cannot be established.
 *
 * COMMENTS: None
 */
BOOL SQLInit ()
{
    dbinit();

    if ( dbgetmaxprocs() < iMaxConnections )
    {
        if ( dbsetmaxprocs(iMaxConnections) == FAIL )
        {
            //set for fail error message when HttpExtensionProc() is calle
because
            //at this point we don't have a pECB so no way to show error
message.
            iMaxConnections = -1;
        }
    }

    // install error and message handlers
    dbmsghandle((DBMSGHANDLE_PROC)msg_handler);
    dberrhandle((DBERRHANDLE_PROC)err_handler);

    return TRUE;
}

/* FUNCTION: BOOL SQLOpenConnection(EXTENSION_CONTROL_BLOCK *pECB, in
iTermId, int iSynclD, DBPROCESS **dbproc, char *server, char *database, char *user, char
*password, char *app, int *spid, long *pack_size)
 *
 * PURPOSE: This function opens the sql connection for use.

```

```

*
* ARGUMENTS:  EXTENSION_CONTROL_BLOCK  *pECB  passed in structure
pointer from inetsrv.
*
*          terminal id of browser          int          iTermId
*
*          sync id of browser             int          iSyncId
*
*          DBPROCESS                      **dbproc  pointer to
returned DBPROCESS
*
*          SQL server name                char         *server
*
*          database                       char         *databaseSQL server
*
*          user name                      char         *user
*
*          user password                  char         *password
*
*          pointer to returned application array  char         *app
*
*          pointer to returned spid        int          *spid
*
*          pointer to returned default pack size  long         *pack_size
*
* RETURNS:          BOOL  FALSE  if successfull
*                  TRUE   if an error occurs
*
* COMMENTS:  None
*/

#ifdef USE_ODBC
static BOOL SQLOpenConnection(EXTENSION_CONTROL_BLOCK *pECB, int
iTermId, int iSyncId, DBPROCESS **dbproc, char *server, char *database, char *user, char
*password, char *app, int *spid, long *pack_size)
{
    RETCODE      rc;
    char         buffer[30];

    *dbproc = (DBPROCESS *)malloc(sizeof(DBPROCESS));
    if ( !*dbproc )
        return TRUE;

    //set pECB data into dbproc
    (*dbproc)->bDeadlock = FALSE;
    (*dbproc)->bFailed = FALSE;

```

```

    (*dbproc)->pECB = pECB;
    (*dbproc)->iTermId = iTermId;
    (*dbproc)->iSyncId = iSyncId;

    if ( SQLAllocConnect(henv, &(*dbproc)->hdbc) == SQL_ERROR )
        return TRUE;

    if ( SQLSetConnectOption((*dbproc)->hdbc, SQL_PACKET_SIZE,
pack_size) == SQL_ERROR )
        return TRUE;

    rc = SQLConnect((*dbproc)->hdbc, server, SQL_NTS, user, SQL_NTS,
password, SQL_NTS);
    if (rc != SQL_SUCCESS && rc != SQL_SUCCESS_WITH_INFO)
        return TRUE;
    rc = SQLAllocStmt((*dbproc)->hdbc, &(*dbproc)->hstmt);
    if (rc == SQL_ERROR)
        return TRUE;

    sprintf(buffer,"use %s", Client->database);

    rc = SQLExecDirect((*dbproc)->hstmt, buffer, SQL_NTS);
    if (rc != SQL_SUCCESS && rc != SQL_SUCCESS_WITH_INFO)
        return TRUE;

    SQLFreeStmt((*dbproc)->hstmt, SQL_CLOSE);
    sprintf(buffer,"set nocount on");
    rc = SQLExecDirect((*dbproc)->hstmt, buffer, SQL_NTS);
    if (rc != SQL_SUCCESS && rc != SQL_SUCCESS_WITH_INFO)
        return TRUE;
    SQLFreeStmt((*dbproc)->hstmt, SQL_CLOSE);

    sprintf(buffer,"select @@spid");

    rc = SQLExecDirect((*dbproc)->hstmt, buffer, SQL_NTS);
    if (rc != SQL_SUCCESS && rc != SQL_SUCCESS_WITH_INFO)
        return TRUE;

    if ( SQLBindCol((*dbproc)->hstmt, 1, SQL_C_SSHORT, &(*dbproc)->sp
0, NULL) == SQL_ERROR )
        return TRUE;

    if ( SQLFetch((*dbproc)->hstmt) == SQL_ERROR )
        return TRUE;

    SQLFreeStmt((*dbproc)->hstmt, SQL_CLOSE);

```



```

        return FALSE;
    }

#else

    static BOOL SQLOpenConnection(EXTENSION_CONTROL_BLOCK *pECB, int
iTermId, int iSyncId, DBPROCESS **dbproc, char *server, char *database, char *user, char
*password, char *app, int *spid)
    {
        LOGINREC      *login;
        PECBINFO      pEcbInfo;

        //set local msg proc for login ecord
        //attach pECB record

        //this is necessary as dblib provides no way to pass user data in a login
structure. So until
        //there is an allocated dbproc we need to use a static which means that the
login attempt must
        //be serialized.

        gpECB = pECB;

        login = dblogin();
        if ( !*user )
            DBSETLUSER(login, "sa");
        else
            DBSETLUSER(login, user);

        DBSETLPWD(login, password);
        DBSETLHOST(login, app);

//      Do not set the packet size. Use the size set up in SQL Server.
//      DBSETLPACKET(login, (unsigned short)DEFCLPACKSIZE);

//      This can potentially cut down on data conversion
        DBSETLVERSION(login, DBVER60);

        if ((*dbproc = dbopen(login, server )) == NULL)
            return TRUE;

        //set pECB data into dbproc
        pEcbInfo = (PECBINFO)malloc(sizeof(PECBINFO));
        pEcbInfo->bDeadlock = FALSE;
        pEcbInfo->pECB = pECB;
        pEcbInfo->iTermId = iTermId;
        pEcbInfo->iSyncId = iSyncId;
    }

```

```

        dbsetuserdata(*dbproc, pEcbInfo);

        // Use the the right database
        dbuse(*dbproc, database);

        dbcmd(*dbproc, "select @@spid");

        dbsqlexec(*dbproc);
        while (dbresults(*dbproc) != NO_MORE_RESULTS)
        {
            dbbind(*dbproc, 1, SMALLBIND, (DBINT) 0, (BYTE *) spid);
            while (dbnextrow(*dbproc) != NO_MORE_ROWS)
                ;
        }
        dbcmd(*dbproc, "set nocount on");

        dbsqlexec(*dbproc);
        while (dbresults(*dbproc) != NO_MORE_RESULTS)
        {
            while (dbnextrow(*dbproc) != NO_MORE_ROWS)
                ;
        }

        //rollback transaction on abort
        dbcmd(*dbproc, "set XACT_ABORT ON");

        dbsqlexec(*dbproc);
        while (dbresults(*dbproc) != NO_MORE_RESULTS)
        {
            while (dbnextrow(*dbproc) != NO_MORE_ROWS)
                ;
        }

        return FALSE;
    }

#endif

/* FUNCTION: BOOL SQLCloseConnection(EXTENSION_CONTROL_BLOCK *pECB,
DBPROCESS *dbproc)
*
* PURPOSE:      This function closes the sql connection.
*
* ARGUMENTS:   EXTENSION_CONTROL_BLOCK      *pECB      passed in structure
pointer from inetsrv.

```

```

*
*          DBPROCESS          *dbproc  pointer to
DBPROCESS
*
* RETURNS:          BOOL  FALSE  if successfull
*                  TRUE   if an error occurs
*
* COMMENTS:        None
*/

#ifdef USE_ODBC
    static BOOL SQLCloseConnection(EXTENSION_CONTROL_BLOCK *pECB,
DBPROCESS *dbproc)
    {
        if ( dbproc )
        {
            SQLFreeStmt(dbproc->hstmt, SQL_DROP);
            SQLDisconnect(dbproc->hdbc);
            SQLFreeConnect(dbproc->hdbc);
            free(dbproc);
            dbproc = NULL;
        }
        return FALSE;
    }
#else
    static BOOL SQLCloseConnection(EXTENSION_CONTROL_BLOCK *pECB,
DBPROCESS *dbproc)
    {
        if (dbclose(dbproc) == FAIL)
            return TRUE;
        return FALSE;
    }
#endif

/* FUNCTION: BOOL SQLDetectDeadlock(DBPROCESS *dbproc)
*
* PURPOSE:        This function checks to see if a sql server deadlock condition exists.
*
* ARGUMENTS:     DBPROCESS          *dbproc
*                 connection db process id to check
*
* RETURNS:       BOOL  FALSE          no deadlock detected
*                 TRUE   deadlock
*                 condition exists
*
* COMMENTS:      None
*/

```

```

BOOL SQLDetectDeadlock(DBPROCESS *dbproc)
{
    PECBINFO      pEcbInfo;

    if ( (pEcbInfo = (PECBINFO)dbgetuserdata(dbproc)) )
    {
        if ( pEcbInfo->bDeadlock )
        {
            pEcbInfo->bDeadlock = FALSE;
            return TRUE;
        }
    }
    return FALSE;
}

// Lifted from HP FDR since they did such a nice job
void TMLLog( char *format, ... )
{
    va_list args;
    char buf[4096];
    int len;
    va_start( args, format );
    _strtime( buf );
    strcat( buf, " ");
    len = strlen( buf );
    (void)_vsprintf( buf+ len, sizeof( buf) - len - 1, format, args);
    buf[sizeof( buf) - 1] = '\0';
    va_end( args );
    userlog( buf );
}

```

```

/* FUNCTION: void UtilStrCpy(char *pDest, char *pSrc, int n)
*
* PURPOSE:        This function copies n characters from string pSrc to pDst and places a
*                 null character at the end of the destination string.
*
* ARGUMENTS:     char          *pDest  destination string pointer
*                 char          *pSrc   source
*                 string pointer
*                 int          n
*                 number of characters to copy
*
* RETURNS:       None
*
* COMMENTS:      Unlike strncpy this function ensures that the result string is
*                 always null terminated.
*/

```

```

*/
static void UtilStrCpy(char *pDest, char *pSrc, int n)
{
    strncpy(pDest, pSrc, n);
    pDest[n] = '\0';

    return;
}

/* FUNCTION: int err_handler(DBPROCESS *dbproc, int severity, int dberr, int oserr, char
 *dberrstr, char *oserrstr)
 *
 * PURPOSE:      This function handles DB-Library errors
 *
 * ARGUMENTS:   DBPROCESS          *dbproc
                DBPROCESS id pointer
                int                  severity
                severity of error
                int                  dberr
                error id
                int                  oserr
                operating system specific error code
                char                  *dberrstr
                printable error description of dberr
                char                  *oserrstr
                printable error description of oserr
 *
 * RETURNS:    int                  INT_CONTINUE
                continue if error is SQLETIME else INT_CANCEL action
 *
 * COMMENTS:   None
 */

int err_handler(DBPROCESS *dbproc, int severity, int dberr, int oserr, char *dberrstr, char
 *oserrstr)
{
    PECBINFO          pEcblInfo;
    EXTENSION_CONTROL_BLOCK *pECB;
    FILE              *fp;
    SYSTEMTIME        systemTime;
    char              szTmp[256];
    int               iTermId;
    int               iSynclId;

    pEcblInfo = NULL;

```

```

if ((dbproc == NULL) || (DBDEAD(dbproc)))
{
    TMLog("DBPROC is invalid");
    return INT_CANCEL;
}

if ( !(pEcblInfo = (PECBINFO)dbgetuserdata(dbproc)) )
{
    pECB = gpECB;
    iTermId = 0;
    iSynclId = 0;
}
else
{
    pECB = pEcblInfo->pECB;
    iTermId = pEcblInfo->iTermId;
    iSynclId = pEcblInfo->iSynclId;
}

if ( pEcblInfo && pEcblInfo->bFailed )
{
    bError == FALSE;
    return INT_CANCEL;
}

if ( oserr != DBNOERR )
{
    TMLog("DBLIB Error %s", oserrstr);
    if ( pEcblInfo )
    {
        pEcblInfo->bFailed = TRUE;
        bError = TRUE;
    }

    GetLocalTime(&systemTime);
    fp = fopen(szErrorLogPath, "ab");

    sprintf(szTmp, "ErrorHandler: DBLIB(%d): %s", oserr, oserrstr);

    TMLog("%2.2d/%2.2d/%2.2d %2.2d:%2.2d:%2.2d\r\n\r\n%s\r\n\r\n",
        systemTime.wYear, systemTime.wMoTh, systemTime.wDay,
        systemTime.wSecond,
        szTmp);

    fclose(fp);
}

```

```

        return INT_CANCEL;
    }

/* FUNCTION: int msg_handler(DBPROCESS *dbproc, DBINT msgno, int msgstate, int
severity, char *msgtext)
*
* PURPOSE:      This function handles DB-Library SQL Server error messages
*
* ARGUMENTS:   DBPROCESS          *dbproc
                DBPROCESS id pointer
*
                message number      DBINT          msgno
*
                message state       int           msgstate
*
                message severity    int           severity
*
                message severity    char          *msgtext
*
                printable message description
*
* RETURNS:     int                  INT_CONTINUE
                continue if error is SQLETIME else INT_CANCEL action
*
                INT_CANCEL          cancel operation
*
* COMMENTS:    This function also sets the dead lock dbproc variable if necessary.
*/

int msg_handler(DBPROCESS *dbproc, DBINT msgno, int msgstate, int severity, char
*msgtext)
{
    PECBINFO
    EXTENSION_CONTROL_BLOCK    *pECB;
    FILE
    SYSTEMTIME
    char                        szTmp[256];
    int                         iTermId;
    int                         iSyncId;

    if ( !(pECBInfo = (PECBINFO)dbgetuserdata(dbproc)) )
    {
        pECB = gpECB;
        iTermId = 0;
        iSyncId = 0;
    }
    else

```

```

    {
        pECB = pECBInfo->pECB;
        iTermId = pECBInfo->iTermId;
        iSyncId = pECBInfo->iSyncId;
    }

    if ( (msgno == 5701) || (msgno == 2528) || (msgno == 5703) || (msgno == 6006) )
        return INT_CONTINUE;

    // deadlock message
    if (msgno == 1205)
    {
        // set the deadlock indicator
        if ( pECBInfo )
            pECBInfo->bDeadlock = TRUE;
        else
            TMLog("Error, dbgetuserdata returned NULL.");
        return INT_CONTINUE;
    }
    if ( pECBInfo && pECBInfo->bFailed )
    {
        TMLog("SQL Error ");
        return INT_CANCEL;
    }

    if (msgno == 0)
        return INT_CONTINUE;
    else
    {
        TMLog("MsgHandler: SQL Error %s", msgtext);

        if ( pECBInfo )
            pECBInfo->bFailed = TRUE;

        bError = TRUE;

        sprintf(szTmp, "Error: SQLSVR(%d): %s", msgno, msgtext);

        TMLog("%2.2d/%2.2d/%2.2d %2.2d:%2.2d:%2.2d\n\n%s\n\n",
            systemTime.wYear, systemTime.wMonth, systemTime.wDay,
            systemTime.wHour, systemTime.wMinute,
            systemTime.wSecond,
            szTmp);
    }
    return INT_CANCEL;
}

```

```

/* FUNCTION: BOOL GetParameters(int argc, char *argv[])
 *
 * PURPOSE:      This function parses the command line passed in to the delivery executable,
initializing
 *                and filling in global variable parameters.
 *
 * ARGUMENTS:   int          argc    number of command line arguments passed
to delivery
 *                char      *argv[]  array of command line argument
pointers
 *
 * RETURNS:     BOOL   FALSE  parameter read successfull
 *                TRUE   user has requested
parameter information screen be displayed.
 *
 * COMMENTS:    None
 */

```

```

static BOOL GetParameters(int argc, char *argv[])
{
    int i;

    szServer[0]          = 0;
    szPassword[0]       = 0;
    bFlush               = FALSE;
    strcpy(szDatabase, "tpcc");
    strcpy(szUser, "sa");

    for(i=0; i<argc; i++)
    {
        if ( argv[i][0] == '-' || argv[i][0] == '/' )
        {
            switch(argv[i][1])
            {
                case 'S':
                case 's':
                    strcpy(szServer, argv[i]+2);
                    break;

                case 'V':
                case 'v':
                    verbose = TRUE;
                    break;

                case '?':
                    return TRUE;
            }
        }
    }
}

```

```

        return FALSE;
    }

/* FUNCTION: void PrintParameters(void)
 *
 * PURPOSE:      This function displays the supported command line flags.
 *
 * ARGUMENTS:    None
 *
 * RETURNS:      None
 *
 * COMMENTS:     None
 */

```

```

static void PrintParameters(void)
{
    TMLog("Performance Tuning Corporation Tuxedo Kit");
    TMLog(" www.perftuning.com (281) 251-3495 ");
    TMLog("StockLevel: -S Server [-v (verbose)]");
    TMLog("StockLevel: Server %s", szServer);
}

```

A.13 tpcc.c

```

/*      FILE:          TPCC.C
 *
 *      Based on:Microsoft TPC-C Kit Ver. 3.00.000
 *
 *      Copyright Microsoft, 1996
 *      Copyright Performance Tuning Corporation, 1997
 *
 *      PURPOSE:      TPC-C main program.
 *      Author:       Philip Durr
 *                   philipdu@Microsoft.com
 *
 *      MODIFIED     Changed for modularity and to allow for the Tuxedo TM
 *
 *      Author:      Edward Whalen
 *                   Performance Turing Corporation
 *                   ewhalen@perftuning.com
 */

```

```

#include <windows.h>
#include <process.h>

```

```

#include <stdio.h>
#include <stdarg.h>
#include <malloc.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <sys\timeb.h>
#include <io.h>

#define DBNTWIN32
#include <sqlfront.h>
#include <sqlldb.h>

#include "trans.h" //tpckit transaction
header contains definations of structures specific to TPC-C
#include "httpext.h" //ISAPI DLL information header

#include "tpcc.h" //this dlls specific
structure, value e.t. header.

#include <tmenv.h>
#include <xa.h>
#include <atmi.h>

static TPINIT *tpinf;
static DWORD TLSIsTpInitedKey;
static int ThrTPlnit();

char szServer[32] = "EDW"; //global variables used with this DLL
char szUser[32] = "sa";
char szPassword[32] = "";
char szDatabase[32] = "tpcc";

BOOL bLog = FALSE;
BOOL dLog = FALSE;

int iThreads = 5;
int iMaxWareHouses = 500;
int iQSlotts = 3000;
int iDelayMs = 100;
int iConnectDelay = 500;
short iDeadlockRetry = (short)3;
short iMaxConnections = (short)25;
int iErrVal = 0;

//char buffer[256];

//allowable client command strings i.e. CMD=command

```

```

char *szCmds[] =
{
    "..NewOrder..", "..Payment..", "..Delivery..", "..Order-Status..", "..Stock-Level..",
    "..Exit..",
    "Submit", "Begin", "Process", "Menu", "Clear", "Users", ""
};

//defined command string functions, called via CMD=command http string from html client.

void (*DoCmd[])(EXTENSION_CONTROL_BLOCK *pECB, int iFormId, int iTermId, int iSync
=
{
    NewOrderForm,
    PaymentForm,
    DeliveryForm,
    OrderStatusForm,
    StockLevelForm,
    Exitcmd,
    SubmitCmd,
    BeginCmd,
    ProcessCmd,
    MenuCmd,
    ClearCmd,
    NumberOfConnectionsCmd
};

//Terminal client id structure and interface defination
TERM Term = { 0, 0, 0, FALSE, NULL, TermInit, TermAllocate, TermRestore, TermAdd,
TermDelete };

//welcome to tpc-c html form buffer, this is first form client sees.
static char*szWelcomeForm = "<HTML>"

    "<HEAD><TITLE>Welcome To TPC-C</TITLE></HEAD><BODY>"
    "Please
    Identify your Warehouse and District for this session.<BR>"
    "<FORM
    ACTION=\"tpcc.dll\" METHOD=\"GET\">"
    "<INPUT
    TYPE=\"hidden\" NAME=\"STATUSID\" VALUE=\"0\">"
    "<INPUT
    TYPE=\"hidden\" NAME=\"ERROR\" VALUE=\"0\">"
    "<INPUT
    TYPE=\"hidden\" NAME=\"FORMID\" VALUE=\"1\">"
    "<INPUT
    TYPE=\"hidden\" NAME=\"TERMINID\" VALUE=\"-2\">"
    "<INPUT

```

```

TYPE="hidden" NAME="SYNCD" VALUE="0"
ID <INPUT NAME="w_id" SIZE=4><BR>
<INPUT NAME="d_id" SIZE=2><BR>
TYPE="submit" NAME="CMD" VALUE="Submit"
"</FORM><BODY>"
"</HTML>";

```

```

static charszTpccLogPath[256]; //path to html log file if logging turned on in registry.
static char szErrorLogPath[256]; //path to error log file.

static CRITICAL_SECTION CriticalSection;
static CRITICAL_SECTION ErrorLogCriticalSection;

static EXTENSION_CONTROL_BLOCK *gpECB;
static int bTpccExit;
//exit delivery disconnect loop as dll exiting.

/* FUNCTION: BOOL APIENTRY DIIMain(HANDLE hModule, DWORD ul_reason_for_call,
LPVOID lpReserved)
*
* PURPOSE: This function is the entry point for the DLL this implementation is based on
the
* fact that DLL_PROCESS_ATTACH is only called from the inet
service once. Connections
* are sent to this function as thread attachments.
*
* ARGUMENTS: HANDLE hModule module handle
* DWORD ul_reason_for_call reason for call
* LPVOID lpReserved
reserved for future use
*
* RETURNS: BOOL FALSE errors
occured in initialization
TRUE
DLL successfully initialized
*
* COMMENTS: None
*/

BOOL APIENTRY DIIMain(HANDLE hModule, DWORD ul_reason_for_call, LPVOID
lpReserved)

```

```

{
int i;
static SECURITY_ATTRIBUTES sa;
static PSECURITY_DESCRIPTOR pSD;

switch( ul_reason_for_call )
{
case DLL_PROCESS_ATTACH:
if ( ReadRegistrySettings() )
{
MessageBox(NULL, "Cannot Find TPCC Key in
registry (run install.exe).", "Init", MB_OK | MB_ICONSTOP);
return FALSE;
}
InitializeCriticalSection(&CriticalSection);
InitializeCriticalSection(&ErrorLogCriticalSection);

(*Term.Init());
if ( !(*Term.Allocate()) )
{
MessageBox(NULL, "Error Trm.Allocate().", "Init",
MB_OK | MB_ICONSTOP);
return FALSE;
}
for(i=Term.iNext; i<Term.iAvailable; i++)
Term.pClientData[i].inUse = 0;
Term.pClientData[0].inUse = 1;

TLSSlsTpnitedKey = TlsAlloc(); // check for failure later
// assumption: value initied to 0
break;
case DLL_THREAD_ATTACH:
break;
case DLL_THREAD_DETACH:
if ( dLog )
{
SYSTEMTIME systemTime;
FILE *fp;

GetLocalTime(&systemTime);
fp = fopen(szErrorLogPath, "ab");
fprintf(fp, "\n\nError: %2.2d/%2.2d/%2.2d
%2.2d:%2.2d:%2.2d\r\n",
systemTime.wYear, systemTime.wMonth,
systemTime.wDay,
systemTime.wHour, systemTime.wMinut,
systemTime.wSecond);
}
}
}

```

```

                fprintf(fp, "DLL_THREAD_DETACH \r\n");
                fclose(fp);
            }
            break;
        case DLL_PROCESS_DETACH:
            if ( pSD )
                free( pSD );

            bTpccExit = TRUE;

            (*Term.Restore());

            DeleteCriticalSection(&CriticalSection);
            DeleteCriticalSection(&ErrorLogCriticalSection);

            TlsFree(TLSIsTpInitedKey);
            break;
    }
    return TRUE;
}

```

```

/* FUNCTION: BOOL WINAPI GetExtensionVersion(HSE_VERSION_INFO *pVer)

```

```

*
* PURPOSE:      This function is called by the inet service when the DLL is first loaded.
*
* ARGUMENTS:   HSE_VERSION_INFO      *pVer      passed in structure in which to
place expected version number.
*
* RETURNS:     TRUE      inet service expected return value.
*
* COMMENTS:   None
*
*/

```

```

BOOL WINAPI GetExtensionVersion(HSE_VERSION_INFO *pVer)
{

```

```

    pVer->dwExtensionVersion = MAKELONG(HSE_VERSION_MINOR,
HSE_VERSION_MAJOR);
    lstrcpy(pVer->lpszExtensionDesc, "TPC-C Server.",
HSE_MAX_EXT_DLL_NAME_LEN);

    return TRUE;
}

```

```

/* FUNCTION: DWORD WINAPI HttpExtensionProc(EXTENSION_CONTROL_BLOCK
*pECB)

```

```

*
* PURPOSE:      This function is the main entry point for the TPCC DLL. The internet ser
calls this function passing in the http string.
*
* ARGUMENTS:   EXTENSION_CONTROL_BLOCK      *pECB      structure pointer to
passed in internet
*
                service information.
*
* RETURNS:     DWORD HSE_STATUS_SUCCESS
                connection can be dropped if error
*
                HSE_STATUS_SUCCESS_AND_KEEP_CONN      keep connect valid comment s
*
* COMMENTS:   None
*
*/

```

```

DWORD WINAPI HttpExtensionProc(EXTENSION_CONTROL_BLOCK *pECB)

```

```

{
    int iCmd, FormId, TermId, iSyncld;
    FILE *fp;

    //      static BOOL bReadRegistry = FALSE;

    if ( iMaxConnections == -1 )
    {
        ErrorMessage(pECB, ERR_CAN_NOT_SET_MAX_CONNECTIONS,
ERR_TYPE_WEBDLL, NULL, -1, -1);
        return HSE_STATUS_SUCCESS;
    }

    //if registry setting is for html logging then show http string passed in.
    if ( bLog )
    {
        SYSTEMTIME      systemTime;

        fp = fopen(szTpccLogPath, "ab");

        GetLocalTime(&systemTime);

        fprintf(fp, "** QUERY * %2.2d/%2.2d/%2.2d
%2.2d:%2.2d:%2.2d\r\n\r\n%s\r\n\r\n",
                systemTime.wYear, systemTime.wMonth, systemTime.wDay
                systemTime.wHour, systemTime.wMinute,
                systemTime.wSecond,
                pECB->lpszQueryString);
        fclose(fp);
    }
}

```



```

    }

    //process http query
    if ( !ProcessQueryString(pECB, &iCmd, &FormId, &TermId, &iSyncId) )
    {
        if ( TermId < 0 )
            ErrorMessage(pECB, ERR_INVALID_TERMID,
ERR_TYPE_WEBDLL, NULL, TermId, iSyncId);
        else
            ErrorMessage(pECB, ERR_COMMAND_UNDEFINED,
ERR_TYPE_WEBDLL, NULL, TermId, iSyncId);
        return HSE_STATUS_SUCCESS_AND_KEEP_CONN;
    }

    if ( TermId != 0 )
    {
        if ( !IsValidTermId(TermId) )
        {
            ErrorMessage(pECB, ERR_INVALID_TERMID,
ERR_TYPE_WEBDLL, NULL, TermId, iSyncId);
            return HSE_STATUS_SUCCESS_AND_KEEP_CONN;
        }

        //must have a valid syncid here since termid is valid
        if ( iSyncId < 1 || iSyncId != Term.pClientData[TermId].iSyncId )
        {
            ErrorMessage(pECB, ERR_INVALID_SYNC_CONNECTION,
ERR_TYPE_WEBDLL, NULL, TermId, iSyncId);
            return HSE_STATUS_SUCCESS_AND_KEEP_CONN;
        }
    }

    //set use time
    Term.pClientData[TermId].iTickCount = GetTickCount();

    //go execute http: command
    (*DoCmd[iCmd])(pECB, FormId, TermId, iSyncId);

    //finish up and keep connection
    return HSE_STATUS_SUCCESS_AND_KEEP_CONN;
}

/* FUNCTION: static BOOL IsValidTermId(int TermId)
*
* PURPOSE:      This function checks to see of the passed in terminal id is valid.
*

```

```

* ARGUMENTS:   int          TermId
               client terminal id
*
* RETURNS:     BOOL   FALSE
               Terminal ID Invalid
               TRUE
               Terminal ID valid
* COMMENTS:    None
*/

static BOOL IsValidTermId(int TermId)
{
    return (BOOL) ( TermId > 0 && TermId <= Term.iAvailable &&
Term.pClientData[TermId].inUse );
}

/* FUNCTION: BOOL ProcessQueryString(EXTENSION_CONTROL_BLOCK *pECB, int
*pCmd, int *pFormId, int *pTermId, int *pSyncId)
*
* PURPOSE:     This function extracts the relevent information out of the http command
passed in from
               the browser.
*
* ARGUMENTS:  EXTENSION_CONTROL_BLOCK *pECB      structure
               pointer to passed in internet
               service information.
               int *pCmd
               returned command id
               int *pFormId
               returned active form client browser is on
               int *pTermId
               returned client terminal id
* RETURNS:    BOOL   FALSE
               success
               TRUE
               command passed in is invalid
* COMMENTS:   If this is the initial connection i.e. client is at welcome screen then
               there will not be a terminal id or current form id if th
the case
               then the pTermid and pFormid return values are
               undefined.
*/

```

```

BOOL ProcessQueryString(EXTENSION_CONTROL_BLOCK *pECB, int *pCmd, int *pFormId,
int *pTermId, int *pSyncId)
{
    char *ptr;
    char szBuffer[25];
    char szTmp[25];
    char *dest = szBuffer;
    int i;

    if ( (ptr = strstr(pECB->lpszQueryString, "FORMID=")) )
        *pFormId = *(ptr+7) & 0x0F;

    if ( (ptr = strstr(pECB->lpszQueryString, "TERMID=")) )
    {
        *pTermId = atoi((ptr+7));
        if ( *pTermId == 0 ) //terminal id 0 used internally
            *pTermId = -1;
        if ( *pTermId == -2 ) //login screen
            *pTermId = 0;
    }
    else
        *pTermId = 0;

    if ( (ptr = strstr(pECB->lpszQueryString, "SYNCID=")) )
        *pSyncId = atoi((ptr+7));
    else
        *pSyncId = 0;

    if ( !(ptr = strstr(pECB->lpszQueryString, "CMD=")) )
    {
        ptr = szBuffer;
        if ( !strcmp(szBuffer, "Default") )
            strcpy(szBuffer, "CMD=Begin");
        switch( *pFormId )
        {
            case WELCOME_FORM:
                strcpy(szBuffer, "CMD=Submit");
                break;
            case MAIN_MENU_FORM:
                strcpy(szBuffer, "CMD=NewOrder");
                break;
            case NEW_ORDER_FORM:
            case PAYMENT_FORM:
            case DELIVERY_FORM:
            case ORDER_STATUS_FORM:
            case STOCK_LEVEL_FORM:

```

```

        if ( !(*pTermId) )
            return FALSE;
        if ( GetKeyValue(pECB->lpszQueryString, "PI*",
            szTmp, sizeof(szTmp)) )
            strcpy(szBuffer, "CMD=Process");
        else
        {
            strcpy(szBuffer, "CMD=");
            strcat(szBuffer, szCmds[*pFormId] -
                NEW_ORDER_FORM]);
        }
        break;
    default:
        return FALSE;
    }
}

ptr += 4;

while( *ptr && *ptr != '&' )
    *dest++ = *ptr++;
*dest = 0;

for(i=0; szCmds[i][0]; i++)
{
    if ( !strcmp(szCmds[i], szBuffer) )
    {
        *pCmd = i;
        return TRUE;
    }
}
return FALSE;
}

/* FUNCTION: void NewOrderForm(EXTENSION_CONTROL_BLOCK *pECB, int iFormId, int
iTermId, int iSyncId)
*
* PURPOSE:      This function wraps the functionality needed for the TPC-C New Order
Form.
*
* ARGUMENTS:   int          iFormId
                unused
                int
iTermId        id of calling browser, i.e. TERMID= from http command line
                EXTENSION_CONTROL_BLOCK *pECB
                structure pointer to passed in internet
*
                service information.

```

```

*
* RETURNS:          None
*
* COMMENTS:        None
*
*/

void NewOrderForm(EXTENSION_CONTROL_BLOCK *pECB, int iFormId, int iTermId, int iSynclId)
{
    WriteZString(pECB, MakeNewOrderForm(iTermId, iSynclId, FALSE, TRUE, FALSE));

    UNUSEDPARAM(iFormId);

    return;
}

/* FUNCTION: void PaymentForm(EXTENSION_CONTROL_BLOCK *pECB, int iFormId, int iTermId, int iSynclId)
*
* PURPOSE:          This function wraps the functionality needed for the TPC-C Payment Form.
*
* ARGUMENTS:        int          iFormId
                    unused
*
                    int
                    iTermId      id of calling browser, i.e. TERMID= from http command line
*
                    int
                    iSynclId     sync id of calling browser
*
                    EXTENSION_CONTROL_BLOCK *pECB
                    structure pointer to passed in internet
*
                    service information.
* RETURNS:          None
*
* COMMENTS:        None
*
*/

void PaymentForm(EXTENSION_CONTROL_BLOCK *pECB, int iFormId, int iTermId, int iSynclId)
{
    WriteZString(pECB, MakePaymentForm(iTermId, iSynclId, TRUE) );

    UNUSEDPARAM(iFormId);
}

```

```

/* FUNCTION: void DeliveryForm(EXTENSION_CONTROL_BLOCK *pECB, int iFormId, int iTermId, int iSynclId)
*
* PURPOSE:          This function wraps the functionality needed for the TPC-C Delivery Form
*
* ARGUMENTS:        int          iFormId
                    unused
*
                    int
                    iTermId      id of calling browser, i.e. TERMID= from http command line
*
                    int
                    iSynclId     sync id of calling browser
*
                    EXTENSION_CONTROL_BLOCK *pECB
                    structure pointer to passed in internet
*
                    service information.
* RETURNS:          None
*
* COMMENTS:        None
*
*/

void DeliveryForm(EXTENSION_CONTROL_BLOCK *pECB, int iFormId, int iTermId, int iSynclId)
{
    // WriteZString(pECB, MakeDeliveryForm(iTermId, iSynclId, TRUE) );
    WriteZString(pECB, MakeDeliveryForm(iTermId, iSynclId, TRUE, TRUE) );

    UNUSEDPARAM(iFormId);
}

/* FUNCTION: void OrderStatusForm(EXTENSION_CONTROL_BLOCK *pECB, int iFormId, int iTermId, int iSynclId)
*
* PURPOSE:          This function wraps the functionality needed for the TPC-C Order Status Form.
*
* ARGUMENTS:        int          iFormId
                    unused
*
                    int
                    iTermId      id of calling browser, i.e. TERMID= from http command line
*
                    int
                    iSynclId     sync id of calling browser
*
                    EXTENSION_CONTROL_BLOCK *pECB
                    structure pointer to passed in internet
*
                    service information.
* RETURNS:          None
*

```

```

* COMMENTS:    None
*
*/

void OrderStatusForm(EXTENSION_CONTROL_BLOCK *pECB, int iFormId, int iTermId, int
iSynclId)
{
    WriteZString(pECB, MakeOrderStatusForm(iTermId, iSynclId, TRUE) );
    UNUSEDPARAM(iFormId);
}

/* FUNCTION: void StockLevelForm(EXTENSION_CONTROL_BLOCK *pECB, int iFormId, int
iTermId, int iSynclId)
*
* PURPOSE:     This function wraps the functionality needed for the TPC-C Stock Level
Form.
*
* ARGUMENTS:  int                                iFormId
              unused
              int
              iTermId        id of calling browser, i.e. TERMIID= from http command line
              int
              iSynclId      sync id of calling browser
              EXTENSION_CONTROL_BLOCK *pECB
              structure pointer to passed in internet
*
              service information.
* RETURNS:    None
*
* COMMENTS:   None
*
*/

void StockLevelForm(EXTENSION_CONTROL_BLOCK *pECB, int iFormId, int iTermId, int
iSynclId)
{
    WriteZString(pECB, MakeStockLevelForm(iTermId, iSynclId, TRUE) );
    return;
}

/* FUNCTION: void Exitcmd(EXTENSION_CONTROL_BLOCK *pECB, int iFormId, int iTermId,
int iSynclId)
*
* PURPOSE:     This function removes a terminal id from use, the allocated structure
however remains

```

```

*
              valid so the next request for a new client will not require a new
memory allocation.
*
* ARGUMENTS:  int                                iFormId
              unused
              int
              iTermId        id of calling browser, i.e. TERMIID= from http command line
              int
              iSynclId      sync id of calling browser
              EXTENSION_CONTROL_BLOCK *pECB
              structure pointer to passed in internet
*
              service information.
* RETURNS:    None
*
* COMMENTS:   None
*
*/

void Exitcmd(EXTENSION_CONTROL_BLOCK *pECB, int iFormId, int iTermId, int iSynclId)
{
    (*Term.Delete)(pECB, iTermId);
    WriteZString(pECB, MakeWelcomeForm() );
    UNUSEDPARAM(iFormId);
    UNUSEDPARAM(iSynclId);
    return;
}

/* FUNCTION: void SubmitCmd(EXTENSION_CONTROL_BLOCK *pECB, int iFormId, int
iTermId, int iSynclId)
*
* PURPOSE:     This function allocated a new terminal id in the Term structure array.
*
* ARGUMENTS:  int                                iFormId
              unused
              int
              iTermId        id of calling browser, i.e. TERMIID= from http command line
              int
              iSynclId      sync id of calling browser
              EXTENSION_CONTROL_BLOCK *pECB
              structure pointer to passed in internet
*
              service information.
* RETURNS:    None
*

```

```

* COMMENTS:   A terminal id can be allocated but still be invalid if the requested warehouse
number
*             is outside the range specified in the registry. This then
will force the client id
*             to be invalid and an error message sent to the users
browser.
*/

```

```

void SubmitCmd(EXTENSION_CONTROL_BLOCK *pECB, int iFormId, int iTermId, int iSyncId)
{
    int iCurrent;

    if ( (iCurrent = (*Term.Add)(pECB, pECB->lpszQueryString)) < 0 )
    {
        ErrorMessage(pECB, ERR_CANNOT_INIT_TERMINAL,
ERR_TYPE_WEBDLL, NULL, iCurrent, iSyncId);
        return;
    }

    if ( Term.pClientData[iCurrent].w_id > iMaxWareHouses ||
Term.pClientData[iCurrent].w_id < 1 )
    {
        ErrorMessage(pECB, ERR_W_ID_INVALID, ERR_TYPE_WEBDLL,
NULL, iCurrent, iSyncId);
        (*Term.Delete)(pECB, iCurrent);
        return;
    }

    if ( Term.pClientData[iCurrent].d_id < 1 || Term.pClientData[iCurrent].d_id > 10 )
    {
        ErrorMessage(pECB, ERR_D_ID_INVALID, ERR_TYPE_WEBDLL,
NULL, iCurrent, iSyncId);
        (*Term.Delete)(pECB, iCurrent);
        return;
    }

    WriteZString(pECB, MakeMainMenuForm(iCurrent,
Term.pClientData[iCurrent].iSyncId) );

    return;
}

```

```

/* FUNCTION: void BeginCmd(EXTENSION_CONTROL_BLOCK *pECB, int iFormId, int
iTermId, int iSyncId)
*
* PURPOSE:   This function is the first command executed. It is executed with the
command
*             CMD=Begin?Server=xxx from the http command line.
*

```

```

* ARGUMENTS:  int iFormId
unused
*             int
iTermId       id of calling browser, i.e. TERMIID= from http command line
*             int
iSyncId       sync id of calling browser
*             EXTENSION_CONTROL_BLOCK *pECB
structure pointer to passed in internet
*

```

```

service information.
* RETURNS:    None
*
* COMMENTS:   SQL server must be specified, however the user and password paramet
are optional.
*             The complete command line is
CMD=Begin&Server=server&User=sa&Psw=&. The & are used
*             to separate parameters which is internet browser
standard.
*/

```

```

void BeginCmd(EXTENSION_CONTROL_BLOCK *pECB, int iFormId, int iTermId, int iSyncId)
{
    LPSTR pQueryString;

    pQueryString = pECB->lpszQueryString;

    WriteZString(pECB, MakeWelcomeForm() );

    UNUSEDPARAM(iFormId);

    return;
}

```

```

/* FUNCTION: void ProcessCmd(EXTENSION_CONTROL_BLOCK *pECB, int iFormId, int
iTermId, int iSyncId)
*
* PURPOSE:   This function process the passed in http command
*
* ARGUMENTS:  int iFormId
unused
*             int
iTermId       id of calling browser, i.e. TERMIID= from http command line
*             int
iSyncId       sync id of calling browser
*             EXTENSION_CONTROL_BLOCK *pECB
structure pointer to passed in internet
*
service information.

```

```

* RETURNS:          None
*
* COMMENTS:        None
*
*/

void ProcessCmd(EXTENSION_CONTROL_BLOCK *pECB, int iFormId, int iTermId, int
iSynclId)
{
    switch( iFormId )
    {
        case WELCOME_FORM:
            return;
        case MAIN_MENU_FORM:
            return;
        case NEW_ORDER_FORM:
            ProcessNewOrderForm(pECB, iTermId, iSynclId);
            return;
        case PAYMENT_FORM:
            ProcessPaymentForm(pECB, iTermId, iSynclId);
            return;
        case DELIVERY_FORM:
            ProcessDeliveryForm(pECB, iTermId, iSynclId);
            return;
        case ORDER_STATUS_FORM:
            ProcessOrderStatusForm(pECB, iTermId, iSynclId);
            return;
        case STOCK_LEVEL_FORM:
            ProcessStockLevelForm(pECB, iTermId, iSynclId);
            return;
    }
}

/* FUNCTION: void ClearCmd(EXTENSION_CONTROL_BLOCK *pECB, int iFormId, int
iTermId, int iSynclId)
*
* PURPOSE:          This function frees all currently logged in terminal ids.
*
* ARGUMENTS:        int                                iFormId
                    unused
                    int
                    iTermId        id of calling browser, i.e. TERMID= from http command line
                    int
                    iSynclId       sync id of calling browser
                    EXTENSION_CONTROL_BLOCK *pECB
                    structure pointer to passed in internet
*
                    service information.

```

```

* RETURNS:          None
*
* COMMENTS:        Use this function with caution, it may cause unpredictable results
                    if existing browsers attempt to use the web client w
out
*
                    beginning at the login screen for each client.
*/

void ClearCmd(EXTENSION_CONTROL_BLOCK *pECB, int iFormId, int iTermId, int iSyncl
Id)
{
    int i;

    EnterCriticalSection(&CriticalSection);

    for(i=0; i<Term.iAvailable; i++)
    {
        if ( Term.pClientData[i].inUse )
            (*Term.Delete)(pECB, i);
    }

    Term.iNext = 0;
    Term.iAvailable = 0;
    Term.iMasterSynclId = 1;

    if ( Term.pClientData )
        free(Term.pClientData);
    Term.pClientData = NULL;
    Term.bInited = FALSE;

    (*Term.Init)();
    if ( !(*Term.Allocate)() )
    {
        ErrorMessage(pECB, ERR_MAX_CONNECT_PARAM,
ERR_TYPE_WEBDLL, NULL, iTermId, iSynclId);
        return;
    }
    for(i=Term.iNext; i<Term.iAvailable; i++)
        Term.pClientData[i].inUse = 0;
    Term.pClientData[0].inUse = 1;

    LeaveCriticalSection(&CriticalSection);

    WriteZString(pECB, MakeWelcomeForm() );

    return;
}

/* FUNCTION: void MenuCmd(EXTENSION_CONTROL_BLOCK *pECB, int iFormId, int

```

```

iTermId, int iSyncId)
*
* PURPOSE:      This function causes an exit to the main menu
*
* ARGUMENTS:   int          iFormId
               unused
*
               int
*      iTermId   id of calling browser, i.e. TERMID= from http command line
               int
*      iSyncId   sync id of calling browser
               EXTENSION_CONTROL_BLOCK *pECB
*      structure pointer to passed in internet
*
               service information.
* RETURNS:      None
*
* COMMENTS:     None
*/

void MenuCmd(EXTENSION_CONTROL_BLOCK *pECB, int iFormId, int iTermId, int iSyncId)
{
    WriteZString(pECB, MakeMainMenuForm(iTermId, iSyncId));

    return;
}

/* FUNCTION: void NumberOfConnectionsCmd(EXTENSION_CONTROL_BLOCK *pECB, int
iFormId, int iTermId, int iSyncId)
*
* PURPOSE:      This function returns to the browser the total number of active terminal ids
*
* ARGUMENTS:   int          iFormId
               unused
*
               int
*      iTermId   id of calling browser, i.e. TERMID= from http command line
               int
*      iSyncId   sync id of calling browser
               EXTENSION_CONTROL_BLOCK *pECB
*      structure pointer to passed in internet
*
               service information.
* RETURNS:      None
*
* COMMENTS:     None
*/

```

```

void NumberOfConnectionsCmd(EXTENSION_CONTROL_BLOCK *pECB, int iFormId, int
iTermId, int iSyncId)
{
    int i;
    int iTotal;

    // EnterCriticalSection(&CriticalSection);

    iTotal = 0;

    for(i=0; i<Term.iAvailable; i++)
    {
        if ( Term.pClientData[i].iUse )
            iTotal++;
    }

    // LeaveCriticalSection(&CriticalSection);

    h_printf(pECB, "Total Active Connections: %d", iTotal);

    return;
}

/* FUNCTION: void WriteZString(EXTENSION_CONTROL_BLOCK *pECB, char *szStr)
*
* PURPOSE:      This function is the low level output function. It writes a string of text back
the
*
               client browser.
*
* ARGUMENTS:   EXTENSION_CONTROL_BLOCK *pECB passed in structure
               char *szStr
               string to display in the client browser.
*
* RETURNS:      None
*
* COMMENTS:     This function assumes that the string to be written to the client browser has
               been formatted in an HTML manner.
*/

static void WriteZString(EXTENSION_CONTROL_BLOCK *pECB, char *szStr)
{
    FILE *fp;
    int lpbSize;
    int iSize;
    char szHeader[128];
    char szHeader1[128];
}

```

```

        lpbSize = strlen(szStr)+1;

        if ( bLog )
        {
            SYSTEMTIME    systemTime;

            fp = fopen(szTpccLogPath, "ab");

            GetLocalTime(&systemTime);

            fprintf(fp, " * HTML PAGE * %2.2d/%2.2d/%2.2d
%2.2d:%2.2d:%2.2d\r\n\r\n%s\r\n\r\n",
                systemTime.wYear, systemTime.wMonth, systemTime.wDay,
                systemTime.wHour, systemTime.wMinute,
systemTime.wSecond,
                szStr);

            fclose(fp);
        }

        iSize = sprintf(szHeader, "200 Ok");
        sprintf(szHeader1, "Connection: keep-alive\r\nContent-type: text/html\r\nContent-
length: %d\r\n\r\n", lpbSize);

#ifdef PURE_PERFORMIX
        (*pECB->ServerSupportFunction)(pECB->ConnID,
HSE_REQ_DONE_WITH_SESSION, NULL, 0, 0);
#else
        (*pECB->ServerSupportFunction)(pECB->ConnID,
HSE_REQ_SEND_RESPONSE_HEADER, szHeader, &iSize, (LPDWORD)szHeader1);
#endif

        (*pECB->WriteClient)(pECB->ConnID, szStr, &lpbSize, 0);

        return;
    }

/* FUNCTION: void h_printf(EXTENSION_CONTROL_BLOCK *pECB, char *format, ...)
 *
 * PURPOSE:      This function forms a high level printf for an HTML browser
 *
 * ARGUMENTS:   EXTENSION_CONTROL_BLOCK    *pECB    passed in structure
pointer from inetsrv.
 *
 *               char                        *format
 *
 *               printf style format string
 *
 *               ...
 *               other arguments as required by printf style format string.

```

```

 *
 * RETURNS:      None
 *
 * COMMENTS:    This function is mainly used for developmental support.
 */

static void h_printf(EXTENSION_CONTROL_BLOCK *pECB, char *format, ...)
{
    //      int lpbSize;
    //      char szBuff[512];
    //      char szTmp[512];

    va_list marker;
    va_start( marker, format );
    vsprintf(szTmp, format, marker);
    va_end( marker );

    //      lpbSize = wsprintf(szBuff, "<html>%s</html>", szTmp) + 1;
    //
    //      (*pECB->WriteClient)(pECB->ConnID, szBuff, &lpbSize, 0);

    wsprintf(szBuff, "<html>%s</html>", szTmp) + 1;

    WriteZString(pECB, szBuff);

    return;
}

void LogTuxError( int TpErrno, char *ErrorMessage )
{
    FILE *fp;
    SYSTEMTIME    systemTime;

    GetLocalTime(&systemTime);

    fp = fopen(szErrorLogPath, "ab");
    fprintf(fp, "\r\nError: %2.2d/%2.2d/%2.2d %2.2d:%2.2d:%2.2d\r\n",
        systemTime.wYear, systemTime.wMonth, systemTime.wDay,
        systemTime.wHour, systemTime.wMinute, systemTime.wSecond);
    fprintf(fp, "Thread %d: TPCCWEB(%d): %s: %s",
        GetCurrentThreadId(), TpErrno, tpstrerror(TpErrno), ErrorMessage);
    fclose(fp);
}

/* FUNCTION: void ErrorMessage(EXTENSION_CONTROL_BLOCK *pECB, int iError, int
iErrorType, char *szMsg)
 *
 * PURPOSE:      This function displays an error message in the client browser.

```



```

*
* ARGUMENTS:  EXTENSION_CONTROL_BLOCK  *pECB  passed in structure
pointer from inetsrv.
*
*           int           iError
           id of error message
*
*           int           iErrorType
           error type, ERR_TYPE_SQL, ERR_TYPE_DBLIB, or
ERR_TYPE_WEBDLL
*
*           int           iTermId
           terminal id from browser
*
*           int           iSyncId
           sync id from browser
*
*           char *        szMsg
           optional error message string used with ERR_TYPE_SQL and
ERR_TYPE_DBLIB
*
* RETURNS:    None
*
* COMMENTS:   If the error type is ERR_TYPE_WEBDLL the szmsg parameter may be
NULL because it
*
*             is ignored. If the error type is ERR_TYPE_SQL or
ERR_TYPE_DBLIB then the szMsg
*
*             parameter contains the text of the error message, so
the szMsg parameter cannot
*
*             be NULL.
*/

void ErrorMessage(EXTENSION_CONTROL_BLOCK *pECB, int iError, int iErrorType, char
*szMsg, int iTermId, int iSyncId)
{
    int i;
    SYSTEMTIME    systemTime;

    static SERRORMSG errorMsgs[] =
    {
        { ERR_SUCCESS,
          "Success, no error."
        },
        { ERR_COMMAND_UNDEFINED,
          "Command undefined."
        },
        { ERR_NOT_IMPLEMENTED_YET,
          "Not Implemented Yet."
        }
    },

```

```

        { ERR_CANNOT_INIT_TERMINAL,
          "Cannot initialize clientconnection."
        },
        { ERR_OUT_OF_MEMORY,
          "insufficient memory."
        },
        { ERR_NEW_ORDER_NOT_PROCESSED,
          "Cannot process new Order form."
        },
        { ERR_PAYMENT_NOT_PROCESSED,
          "Cannot process payment form."
        },
        { ERR_NO_SERVER_SPECIFIED,
          "No Server name specified."
        },
        { ERR_ORDER_STATUS_NOT_PROCESSED,
          "Cannot process order status form."
        },
        { ERR_W_ID_INVALID,
          "Invalid Warehouse ID."
        },
        { ERR_CAN_NOT_SET_MAX_CONNECTIONS,
          "Insufficient memory to allocate # connections."
        },
        { ERR_NOSUCH_CUSTOMER,
          "No such customer."
        },
        { ERR_D_ID_INVALID,
          "Invalid District ID Must be 1 to 10."
        },
        { ERR_MAX_CONNECT_PARAM,
          "Max client connections exceeded, r# install to increase."
        },
        { ERR_INVALID_SYNC_CONNECTION,
          "Invalid Terminal Sync ID."
        },
        { ERR_INVALID_TERMID,
          "Invalid Terminal ID."
        },
        { ERR_PAYMENT_INVALID_CUSTOMER,
          "Payment Form, No such Customer."
        },
        { ERR_SQL_OPEN_CONNECTION,
          "SQLOpenConnection API Failed."
        },
        { ERR_STOCKLEVEL_MISSING_THRESHOLD_KEY,
          "Stock Level missing Threshold key \"TT\"."
        }
    },

```

```

    {      ERR_STOCKLEVEL_THRESHOLD_INVALID,
"Stock Level Threshold invalid data type range = 499."      },
    {      ERR_STOCKLEVEL_THRESHOLD_RANGE,
"Stock Level Threshold out of range, range must be 1 - 99."  },
    {      ERR_STOCKLEVEL_NOT_PROCESSED,
"Stock Level not processed."                                },
    },
    {      ERR_NEWORDER_FORM_MISSING_DID,
"New Order missing District key \"DID*\"."                  },
    },
    {      ERR_NEWORDER_DISTRICT_INVALID,
"New Order District ID Invalid range 1 - 10."                },
    },
    {      ERR_NEWORDER_DISTRICT_RANGE,
"New Order District ID out of Range. Range = 1 - 10."      },
    },
    {      ERR_NEWORDER_CUSTOMER_KEY,
"New Order missing Customer key \"CID*\"."                  },
    },
    {      ERR_NEWORDER_CUSTOMER_INVALID,
"New Order customer id invalid data type, range = 1 to 3000." },
    {      ERR_NEWORDER_CUSTOMER_RANGE,
"New Order customer id out of range, range = 1 to 3000."    },
    {      ERR_NEWORDER_MISSING_IID_KEY,
"New Order missing Item Id key \"IID*\"."                    },
    },
    {      ERR_NEWORDER_ITEM_BLANK_LINES,
"New Order blank order lines all orders must be continuous." },
    {      ERR_NEWORDER_ITEMID_INVALID,
"New Order Item Id is wrong data type, must be numeric."    },
    },
    {      ERR_NEWORDER_MISSING_SUPPW_KEY,
"New Order missing Supp_W key \"SP##*\"."                    },
    },
    {      ERR_NEWORDER_SUPPW_INVALID,
"New Order Supp_W invalid data type must be numeric."        },
    },
    {      ERR_NEWORDER_MISSING_QTY_KEY,
"New Order Missing Qty key \"Qty##*\"."                      },
    },
    {      ERR_NEWORDER_QTY_INVALID,
"New Order Qty invalid must be numeric range 1 - 99."        },
    },
    {      ERR_NEWORDER_SUPPW_RANGE,
"New Order Supp_W value out of range range = 1 - Max Warehouses." },
    {      ERR_NEWORDER_ITEMID_RANGE,
"New Order Item Id is out of range. Range = 1 to 999999."    },
    },
    {      ERR_NEWORDER_QTY_RANGE,
"New Order Qty is out of range. Range = 1 to 99."            },
    },
    {      ERR_PAYMENT_DISTRICT_INVALID,
"Payment District ID is invalid must be 1 - 10."              },
    },
    {      ERR_NEWORDER_SUPPW_WITHOUT_ITEMID,
"New Order Supp_W field entered without a corresponding Item_Id." },
    {      ERR_NEWORDER_QTY_WITHOUT_ITEMID,
"New Order Qty entered without a corresponding Item_Id."      },
    {      ERR_NEWORDER_NOITEMS_ENTERED,
"New Order Blank Items between items, items must be continuous." },
    {      ERR_PAYMENT_MISSING_DID_KEY,
"Payment missing District Key \"DID*\"."                      },
    },
    {      ERR_PAYMENT_DISTRICT_RANGE,
"Payment District Out of range, range = 1 - 10."              },
    },
    {      ERR_PAYMENT_MISSING_CID_KEY,
"Payment missing Customer Key \"CID*\"."                      },
    },
    {      ERR_PAYMENT_CUSTOMER_INVALID,
"Payment Customer data type invalid, must be numeric."        },
    },
    {      ERR_PAYMENT_MISSING_CLT,
"Payment missing Customer Last Name Key \"CLT*\"."            },
    },
    {      ERR_PAYMENT_LAST_NAME_TO_LONG,
"Payment Customer last name longer than 16 characters."        },
    },
    {      ERR_PAYMENT_CUSTOMER_RANGE,
"Payment Customer ID out of range, must be 1 to 3000."        },
    },
    {      ERR_PAYMENT_CID_AND_CLT,
"Payment Customer ID and Last Name entered must be one or other." },
    {      ERR_PAYMENT_MISSING_CDI_KEY,
"Payment missing Customer district key \"CDI*\"."              },
    },
    {      ERR_PAYMENT_CDI_INVALID,
"Payment Customer district invalid must be numeric."          },
    },
    {      ERR_PAYMENT_CDI_RANGE,
"Payment Customerdistrict out of range must be 1 - 10."        },
    },
    {      ERR_PAYMENT_MISSING_CWI_KEY,
"Payment missing Customer Warehouse key \"CWI*\"."            },
    },
    {      ERR_PAYMENT_CWI_INVALID,

```

```

"Payment Customer Warehouse invalid must be numeric."
},
    {
        ERR_PAYMENT_CWI_RANGE,
        "Payment Customer Warehouse out of range, 1 to Max Warehouses."
    },
    {
        ERR_PAYMENT_MISSING_HAM_KEY,
        "Payment missing Amount key \"HAM*\"."
    },
    {
        ERR_PAYMENT_HAM_INVALID,
        "Payment Amount invalid data type must be numeric."
    },
    {
        ERR_PAYMENT_HAM_RANGE,
        "Payment Amount out of range, 0 - 9999.99."
    },
    {
        ERR_ORDERSTATUS_MISSING_DID_KEY,
        "Order Status missing District key \"DID*\"."
    },
    {
        ERR_ORDERSTATUS_DID_INVALID,
        "Order Status District invalid, value must be numericid - 10."
    },
    {
        ERR_ORDERSTATUS_DID_RANGE,
        "Order Status District out of range must be 1 - 10."
    },
    {
        ERR_ORDERSTATUS_MISSING_CID_KEY,
        "Order Status missing Customer key \"CID*\"."
    },
    {
        ERR_ORDERSTATUS_MISSING_CLT_KEY,
        "Order Status missing Customer Last Name key \"CLT*\"."
    },
    {
        ERR_ORDERSTATUS_CLT_RANGE,
        "Order Status Customer last name longer than 16 characters."
    },
    {
        ERR_ORDERSTATUS_CID_INVALID,
        "Order Status Customer ID invalid, range must be numeric 1 - 3000."
    },
    {
        ERR_ORDERSTATUS_CID_RANGE,
        "Order Status Customer ID out of range must be 1 - 3000."
    },
    {
        ERR_ORDERSTATUS_CID_AND_CLT,
        "Order Status Customer ID and LastName entered must be only one."
    },
    {
        ERR_DELIVERY_MISSING_OCD_KEY,
        "Delivery missing Carrier IDkey \"OCD*\"."
    },
    {
        ERR_DELIVERY_CARRIER_INVALID,
        "Delivery Carrier ID invalid must be numeric 1 - 10."
    },
    {
        ERR_DELIVERY_CARRIER_ID_RANGE,
        "Delivery Carrier ID out of range must be 1 - 10."
    },
    {
        ERR_PAYMENT_MISSING_CLT_KEY,
        "Payment missing Customer Last Name key \"CLT*\"."
    },
    {
        0,
        ""
    }

```

```

    }
};

static char szNoMsg[] = "";
char *szForm;

GetLocalTime(&systemTime);

if ( !szMsg )
    szMsg = szNoMsg;

if ( iTermId > 0 && IsValidTermId(iTermId) )
    szForm = Term.pClientData[iTermId].szBuffer; //if termid valid use comr
terminal static buffer.
else
    szForm = Term.pClientData[0].szBuffer; //else term id invalid so use
common terminal static buffer.
switch(iErrorType)
{
    case ERR_TYPE_WEBDLL:
        for(i=0; errorMsgs[i].szMsg[0]; i++)
        {
            if ( iError == errorMsgs[i].iError )
                break;
        }
        if ( !errorMsgs[i].szMsg[0] )
            i = 1;
        strcpy(szForm, "<HTML><HEAD><TITLE>Welcome To
TPC-C</TITLE></HEAD><BODY><FORM ACTION=\"tpcc.dll\" METHOD=\"GET\">");
        sprintf(szForm+strlen(szForm), "<INPUT TYPE=\"hidden\"
NAME=\"STATUSID\" VALUE=\"%d\">", iError);
        sprintf(szForm+strlen(szForm), "<INPUT TYPE=\"hidden\"
NAME=\"ERROR\" VALUE=\"%d\">", iErrorType);
        sprintf(szForm+strlen(szForm), "<INPUT TYPE=\"hidden\"
NAME=\"TERMID\" VALUE=\"%d\">", iTermId);
        sprintf(szForm+strlen(szForm), "<INPUT TYPE=\"hidden\"
NAME=\"SYNCID\" VALUE=\"%d\">", iSyncId);
        sprintf(szForm+strlen(szForm), "Error: TPCCWEB(%d): %s'
iError, errorMsgs[i].szMsg);
        strcat(szForm, "</FORM><BODY></HTML>");
        WriteZString(pECB, szForm);
        break;
    case ERR_TYPE_SQL:
        strcpy(szForm, "<HTML><HEAD><TITLE>Welcome To
TPC-C</TITLE></HEAD><BODY><FORM ACTION=\"tpcc.dll\" METHOD=\"GET\">");
        sprintf(szForm+strlen(szForm), "<INPUT TYPE=\"hidden\"
NAME=\"STATUSID\" VALUE=\"%d\">", iError);

```

```

        sprintf(szForm+strlen(szForm), "<INPUT TYPE=\"hidden\"
NAME=\"ERROR\" VALUE=\"%d\">", iErrorType);
        sprintf(szForm+strlen(szForm), "<INPUT TYPE=\"hidden\"
NAME=\"TERMid\" VALUE=\"%d\">", iTermId);
        sprintf(szForm+strlen(szForm), "<INPUT TYPE=\"hidden\"
NAME=\"SYNCID\" VALUE=\"%d\">", iSyncId);
        sprintf(szForm+strlen(szForm), "Error: SQLSVR(%d): %s",
iError, szMsg);

        strcat(szForm, "</FORM><BODY></HTML>");
        WriteZString(pECB, szForm);
        break;
    case ERR_TYPE_DBLIB:
        strcpy(szForm, "<HTML><HEAD><TITLE>Welcome To
TPC-C</TITLE></HEAD><BODY><FORM ACTION=\"tpcc.dll\" METHOD=\"GET\">");
        sprintf(szForm+strlen(szForm), "<INPUT TYPE=\"hidden\"
NAME=\"STATUSID\" VALUE=\"%d\">", iError);
        sprintf(szForm+strlen(szForm), "<INPUT TYPE=\"hidden\"
NAME=\"ERROR\" VALUE=\"%d\">", iErrorType);
        sprintf(szForm+strlen(szForm), "<INPUT TYPE=\"hidden\"
NAME=\"TERMid\" VALUE=\"%d\">", iTermId);
        sprintf(szForm+strlen(szForm), "<INPUT TYPE=\"hidden\"
NAME=\"SYNCID\" VALUE=\"%d\">", iSyncId);
        sprintf(szForm+strlen(szForm), "Error: DBLIB(%d): %s", iError,
szMsg);

        strcat(szForm, "</FORM><BODY></HTML>");
        WriteZString(pECB, szForm);
        break;
    case ERR_TYPE_ODBC:
        strcpy(szForm, "<HTML><HEAD><TITLE>Welcome To
TPC-C</TITLE></HEAD><BODY><FORM ACTION=\"tpcc.dll\" METHOD=\"GET\">");
        sprintf(szForm+strlen(szForm), "<INPUT TYPE=\"hidden\"
NAME=\"STATUSID\" VALUE=\"%d\">", iError);
        sprintf(szForm+strlen(szForm), "<INPUT TYPE=\"hidden\"
NAME=\"ERROR\" VALUE=\"%d\">", iErrorType);
        sprintf(szForm+strlen(szForm), "<INPUT TYPE=\"hidden\"
NAME=\"TERMid\" VALUE=\"%d\">", iTermId);
        sprintf(szForm+strlen(szForm), "<INPUT TYPE=\"hidden\"
NAME=\"SYNCID\" VALUE=\"%d\">", iSyncId);
        sprintf(szForm+strlen(szForm), "Error: ODBC");
        strcat(szForm, "</FORM><BODY></HTML>");
        WriteZString(pECB, szForm);
        break;
    case ERR_TYPE_SOCKET:
        strcpy(szForm, "<HTML><HEAD><TITLE>Welcome To
TPC-C</TITLE></HEAD><BODY><FORM ACTION=\"tpcc.dll\" METHOD=\"GET\">");
        sprintf(szForm+strlen(szForm), "<INPUT TYPE=\"hidden\"
NAME=\"STATUSID\" VALUE=\"%d\">", iError);

```

```

        sprintf(szForm+strlen(szForm), "<INPUT TYPE=\"hidden\"
NAME=\"ERROR\" VALUE=\"%d\">", iErrorType);
        sprintf(szForm+strlen(szForm), "<INPUT TYPE=\"hidden\"
NAME=\"TERMid\" VALUE=\"%d\">", iTermId);
        sprintf(szForm+strlen(szForm), "<INPUT TYPE=\"hidden\"
NAME=\"SYNCID\" VALUE=\"%d\">", iSyncId);
        sprintf(szForm+strlen(szForm), "Error: SOCKET");
        strcat(szForm, "</FORM><BODY></HTML>");
        WriteZString(pECB, szForm);
        break;
    case ERR_TYPE_DEADLOCK:
        strcpy(szForm, "<HTML><HEAD><TITLE>Welcome To
TPC-C</TITLE></HEAD><BODY><FORM ACTION=\"tpcc.dll\" METHOD=\"GET\">");
        sprintf(szForm+strlen(szForm), "<INPUT TYPE=\"hidden\"
NAME=\"STATUSID\" VALUE=\"%d\">", iError);
        sprintf(szForm+strlen(szForm), "<INPUT TYPE=\"hidden\"
NAME=\"ERROR\" VALUE=\"%d\">", iErrorType);
        sprintf(szForm+strlen(szForm), "<INPUT TYPE=\"hidden\"
NAME=\"TERMid\" VALUE=\"%d\">", iTermId);
        sprintf(szForm+strlen(szForm), "<INPUT TYPE=\"hidden\"
NAME=\"SYNCID\" VALUE=\"%d\">", iSyncId);
        sprintf(szForm+strlen(szForm), "Error: Deadlock");
        strcat(szForm, "</FORM><BODY></HTML>");
        WriteZString(pECB, szForm);
        break;
    }
    return;
}

#ifdef OLD_ERROR
void ErrorMessage(EXTENSION_CONTROL_BLOCK *pECB, int iError, int iErrorType, char
*szMsg, int iTermId, int iSyncId)
{
    int i;

    static SERRORMSG errorMsgs[] =
    {
        { ERR_SUCCESS,
          "Success, no error."
        },
        { ERR_COMMAND_UNDEFINED,
          "Command undefined."
        },
        { ERR_NOT_IMPLEMENTED_YET,
          "Not Implemented Yet."
        },
        { ERR_CANNOT_INIT_TERMINAL,

```

```

"Cannot initialize client connection."
    },
    {
        ERR_OUT_OF_MEMORY,
        "insufficient memory."
    },
    {
        ERR_NEW_ORDER_NOT_PROCESSED,
        "Cannot process new Order form."
    },
    {
        ERR_PAYMENT_NOT_PROCESSED,
        "Cannot process payment form."
    },
    {
        ERR_NO_SERVER_SPECIFIED,
        "No Server name specified."
    },
    {
        ERR_ORDER_STATUS_NOT_PROCESSED,
        "Cannot process order status form."
    },
    {
        ERR_W_ID_INVALID,
        "Invalid Warehouse ID."
    },
    {
        ERR_CAN_NOT_SET_MAX_CONNECTIONS,
        "Insufficient memory to allocate # connections."
    },
    {
        ERR_NOSUCH_CUSTOMER,
        "No such customer."
    },
    {
        ERR_D_ID_INVALID,
        "Invalid District ID Must be 1 to 10."
    },
    {
        ERR_MAX_CONNECT_PARAM,
        "Max client connections exceeded, run install to increase."
    },
    {
        ERR_INVALID_SYNC_CONNECTION,
        "Invalid Terminal Sync ID."
    },
    {
        ERR_INVALID_TERMID,
        "Invalid Terminal ID."
    },
    {
        ERR_PAYMENT_INVALID_CUSTOMER,
        "Payment Form, No such Customer."
    },
    {
        ERR_SQL_OPEN_CONNECTION,
        "SQLOpenConnection API Failed."
    },
    {
        ERR_STOCKLEVEL_MISSING_THRESHOLD_KEY,
        "Stock Level missing Threshold key \"TT\"."
    },
    },

```

```

    {
        ERR_STOCKLEVEL_THRESHOLD_INVALID,
        "Stock Level Threshold invalid data type range = 1 - 99."
    },
    {
        ERR_STOCKLEVEL_THRESHOLD_RANGE,
        "Stock Level Threshold out of range, range must be 1 - 99."
    },
    {
        ERR_STOCKLEVEL_NOT_PROCESSED,
        "Stock Level not processed."
    },
    {
        ERR_NEWORDER_FORM_MISSING_DID,
        "New Order missing District key \"DID\"."
    },
    {
        ERR_NEWORDER_DISTRICT_INVALID,
        "New Order District ID Invalid range 1 - 10."
    },
    {
        ERR_NEWORDER_DISTRICT_RANGE,
        "New Order District ID out of Range. Range = 1 - 10."
    },
    {
        ERR_NEWORDER_CUSTOMER_KEY,
        "New Order missing Customer key \"CID\"."
    },
    {
        ERR_NEWORDER_CUSTOMER_INVALID,
        "New Order customer id invalid data type, range = 1 to 3000."
    },
    {
        ERR_NEWORDER_CUSTOMER_RANGE,
        "New Order customer id out of range, range = 1 to 3000."
    },
    {
        ERR_NEWORDER_MISSING_IID_KEY,
        "New Order missing Item Id key \"IID\"."
    },
    {
        ERR_NEWORDER_ITEM_BLANK_LINES,
        "New Order blank order lines all orders must be continuous."
    },
    {
        ERR_NEWORDER_ITEMID_INVALID,
        "New Order Item Id is wrong data type, must be numeric."
    },
    {
        ERR_NEWORDER_MISSING_SUPPW_KEY,
        "New Order missing Supp_W key \"SP##\"."
    },
    {
        ERR_NEWORDER_SUPPW_INVALID,
        "New Order Supp_W invalid data type must be numeric."
    },
    {
        ERR_NEWORDER_MISSING_QTY_KEY,
        "New Order Missing Qty key \"Qty##\"."
    },
    {
        ERR_NEWORDER_QTY_INVALID,
        "New Order Qty invalid must be numeric range 1 - 99."
    },
    {
        ERR_NEWORDER_SUPPW_RANGE,
        "New Order Supp_W value out of range range = 1 - Max Warehouses."
    },
    {
        ERR_NEWORDER_ITEMID_RANGE,
        "New Order Item Id is out of range. Range = 1 to 999999."
    },
    },

```

```

        {
            ERR_NEWORDER_QTY_RANGE,
            "New Order Qty is out of range. Range = 1 to 99."
        },
        {
            ERR_PAYMENT_DISTRICT_INVALID,
            "Payment District ID is invalid must be 1 - 10."
        },
    },
    {
        ERR_NEWORDER_SUPPW_WITHOUT_ITEMID,
        "New Order Supp_W field entered without a corresponding Item Id." },
        {
            ERR_NEWORDER_QTY_WITHOUT_ITEMID,
            "New Order Qty entered without a corresponding Item Id." },
        {
            ERR_NEWORDER_NOITEMS_ENTERED,
            "New Order Blank Items between items, items must be continuous." },
        {
            ERR_PAYMENT_MISSING_DID_KEY,
            "Payment missing District Key \"DID\"."
        },
    },
    {
        ERR_PAYMENT_DISTRICT_RANGE,
        "Payment District Out of range, range = 1 - 10."
    },
    {
        ERR_PAYMENT_MISSING_CID_KEY,
        "Payment missing Customer Key \"CID\"."
    },
    {
        ERR_PAYMENT_CUSTOMER_INVALID,
        "Payment Customer data type invalid, must be numeric."
    },
    {
        ERR_PAYMENT_MISSING_CLT,
        "Payment missing Customer Last Name Key \"CLT\"."
    },
    {
        ERR_PAYMENT_LAST_NAME_TO_LONG,
        "Payment Customer last name longer than 16 characters."
    },
    {
        ERR_PAYMENT_CUSTOMER_RANGE,
        "Payment Customer ID out of range, must be 1 to 3000."
    },
    {
        ERR_PAYMENT_CID_AND_CLT,
        "Payment Customer ID and Last Name entered must be one or other." },
        {
            ERR_PAYMENT_MISSING_CDI_KEY,
            "Payment missing Customer districtkey \"CDI\"."
        },
    },
    {
        ERR_PAYMENT_CDI_INVALID,
        "Payment Customer district invalid must be numeric."
    },
    {
        ERR_PAYMENT_CDI_RANGE,
        "Payment Customer district out of range must be 1 - 10."
    },
    {
        ERR_PAYMENT_MISSING_CWI_KEY,
        "Payment missing Customer Warehouse key \"CWI\"."
    },
    },
    {
        ERR_PAYMENT_CWI_INVALID,
        "Payment Customer Warehouse invalid must be numeric."
    },
    {
        ERR_PAYMENT_CWI_RANGE,
        "Payment Customer Warehouse out of range, 1 to Max Warehouses."
    },
    {
        ERR_PAYMENT_MISSING_HAM_KEY,
        "Payment missing Amount key \"HAM\"."
    },
    {
        ERR_PAYMENT_HAM_INVALID,
        "Payment Amount invalid data type must be numeric."
    },
    {
        ERR_PAYMENT_HAM_RANGE,
        "Payment Amount out of range, 0 - 9999.99."
    },
    },
    {
        ERR_ORDERSTATUS_MISSING_DID_KEY,
        "Order Status missing District key \"DID\"."
    },
    {
        ERR_ORDERSTATUS_DID_INVALID,
        "Order Status District invalid, value must be numeric 1 - 10." },
    {
        ERR_ORDERSTATUS_DID_RANGE,
        "Order Status District out of range must be 1- 10."
    },
    {
        ERR_ORDERSTATUS_MISSING_CID_KEY,
        "Order Status missing Customer key \"CID\"."
    },
    {
        ERR_ORDERSTATUS_MISSING_CLT_KEY,
        "Order Status missing Customer Last Name key \"CLT\"."
    },
    {
        ERR_ORDERSTATUS_CLT_RANGE,
        "Order Status Customer last name longer than 16 characters." },
    {
        ERR_ORDERSTATUS_CID_INVALID,
        "Order Status Customer ID invalid, range must be numeric 1 - 3000." },
    {
        ERR_ORDERSTATUS_CID_RANGE,
        "Order Status Customer ID out of range must be 1 - 3000."
    },
    {
        ERR_ORDERSTATUS_CID_AND_CLT,
        "Order Status Customer ID and LastName entered must be only one." },
    {
        ERR_DELIVERY_MISSING_OCD_KEY,
        "Delivery missing Carrier ID key \"OCD\"."
    },
    {
        ERR_DELIVERY_CARRIER_INVALID,
        "Delivery Carrier ID invalid must be numeric 1 - 10."
    },
    {
        ERR_DELIVERY_CARRIER_ID_RANGE,
        "Delivery Carrier ID out of range must be 1 - 10."
    },
    {
        ERR_PAYMENT_MISSING_CLT_KEY,
        "Payment missing Customer Last Name key \"CLT\"."
    },
    },
    {
        0,
    },

```

```

        ""
    };
};

static char szNoMsg[] = "";
char      *szForm;

if ( !szMsg )
    szMsg = szNoMsg;

if ( iTermId > 0 && IsValidTermId(iTermId) )
    szForm = Term.pClientData[iTermId].szBuffer; //if termid valid use common
terminal static buffer.
else
    szForm = Term.pClientData[0].szBuffer; //else term id invalid so use
common terminal static buffer.
switch(iErrorType)
{
    case ERR_TYPE_WEBDDL:
        for(i=0; errorMsgs[i].szMsg[0]; i++)
        {
            if ( iError == errorMsgs[i].iError )
                break;
        }
        if ( !errorMsgs[i].szMsg[0] )
            i = 1;
        strcpy(szForm, "<HTML><HEAD><TITLE>Welcome To
TPC-C</TITLE></HEAD><BODY><FORM ACTION=\"tpcc.dll\" METHOD=\"GET\">");
        sprintf(szForm+strlen(szForm), "<INPUT TYPE=\"hidden\"
NAME=\"STATUSID\" VALUE=\"%d\">", iError);
        sprintf(szForm+strlen(szForm), "<INPUT TYPE=\"hidden\"
NAME=\"ERROR\" VALUE=\"%d\">", iErrorType);
        sprintf(szForm+strlen(szForm), "<INPUT TYPE=\"hidden\"
NAME=\"TERMINID\" VALUE=\"%d\">", iTermId);
        sprintf(szForm+strlen(szForm), "<INPUT TYPE=\"hidden\"
NAME=\"SYNCID\" VALUE=\"%d\">", iSyncId);
        sprintf(szForm+strlen(szForm), "Error: TPCCWEB(%d): %s",
iError, errorMsgs[i].szMsg);
        strcat(szForm, "</FORM><BODY></HTML>");
        WriteZString(pECB, szForm);
        break;
    case ERR_TYPE_SQL:
        strcpy(szForm, "<HTML><HEAD><TITLE>Welcome To
TPC-C</TITLE></HEAD><BODY><FORM ACTION=\"tpcc.dll\" METHOD=\"GET\">");
        sprintf(szForm+strlen(szForm), "<INPUT TYPE=\"hidden\"
NAME=\"STATUSID\" VALUE=\"%d\">", iError);

```

```

        sprintf(szForm+strlen(szForm), "<INPUT TYPE=\"hidden\"
NAME=\"ERROR\" VALUE=\"%d\">", iErrorType);
        sprintf(szForm+strlen(szForm), "<INPUT TYPE=\"hidden\"
NAME=\"TERMINID\" VALUE=\"%d\">", iTermId);
        sprintf(szForm+strlen(szForm), "<INPUT TYPE=\"hidden\"
NAME=\"SYNCID\" VALUE=\"%d\">", iSyncId);
        sprintf(szForm+strlen(szForm), "Error: SQLSVR(%d): %s",
iError, szMsg);
        strcat(szForm, "</FORM><BODY></HTML>");
        WriteZString(pECB, szForm);
        break;
    case ERR_TYPE_DBLIB:
        strcpy(szForm, "<HTML><HEAD><TITLE>Welcome To
TPC-C</TITLE></HEAD><BODY><FORM ACTION=\"tpcc.dll\" METHOD=\"GET\">");
        sprintf(szForm+strlen(szForm), "<INPUT TYPE=\"hidden\"
NAME=\"STATUSID\" VALUE=\"%d\">", iError);
        sprintf(szForm+strlen(szForm), "<INPUT TYPE=\"hidden\"
NAME=\"ERROR\" VALUE=\"%d\">", iErrorType);
        sprintf(szForm+strlen(szForm), "<INPUT TYPE=\"hidden\"
NAME=\"TERMINID\" VALUE=\"%d\">", iTermId);
        sprintf(szForm+strlen(szForm), "<INPUT TYPE=\"hidden\"
NAME=\"SYNCID\" VALUE=\"%d\">", iSyncId);
        sprintf(szForm+strlen(szForm), "Error: DBLIB(%d): %s", iEr
szMsg);
        strcat(szForm, "</FORM><BODY></HTML>");
        WriteZString(pECB, szForm);
        break;
    case ERR_TYPE_ODBC:
        strcpy(szForm, "<HTML><HEAD><TITLE>Welcome To
TPC-C</TITLE></HEAD><BODY><FORM ACTION=\"tpcc.dll\" METHOD=\"GET\">");
        sprintf(szForm+strlen(szForm), "<INPUT TYPE=\"hidden\"
NAME=\"STATUSID\" VALUE=\"%d\">", iError);
        sprintf(szForm+strlen(szForm), "<INPUT TYPE=\"hidden\"
NAME=\"ERROR\" VALUE=\"%d\">", iErrorType);
        sprintf(szForm+strlen(szForm), "<INPUT TYPE=\"hidden\"
NAME=\"TERMINID\" VALUE=\"%d\">", iTermId);
        sprintf(szForm+strlen(szForm), "<INPUT TYPE=\"hidden\"
NAME=\"SYNCID\" VALUE=\"%d\">", iSyncId);
        sprintf(szForm+strlen(szForm), "Error: ODBC(%d): %s", iEr
szMsg);
        strcat(szForm, "</FORM><BODY></HTML>");
        WriteZString(pECB, szForm);
        break;
    }
}
return;
}
#endif

```

```

/* FUNCTION: BOOL GetKeyValue(char *pQueryString, char *pKey, char *pValue, int iMax)
*
* PURPOSE:      This function parses a http formatted string for specific key values.
*
* ARGUMENTS:   char          *pQueryString    http string from client
browser
*              char          *pKey
*              key value to look for
*              char          *pValue
*              character array into which to place key's value
*              int          iMax
*              maximum length of key value array.
*
* RETURNS:     BOOL   FALSE   key value not found
*              TRUE    key valud found
*
* COMMENTS:    http keys are formatted either KEY=value& or KEY=value\0. This DLL
formats
*              TPC-C input fields in such a manner that the keys can
be extracted in the
*              above manner.
*/

static BOOL GetKeyValue(char *pQueryString, char *pKey, char *pValue, int iMax)
{
    char *ptr;

    if ( !(ptr=strstr(pQueryString, pKey)) )
        return FALSE;
    if ( !(ptr=strchr(ptr, '=') ) )
        return FALSE;
    ptr++;
    iMax--;
    while( *ptr && *ptr != '&' && iMax)
    {
        *pValue++ = *ptr++;
        iMax--;
    }
    *pValue = 0;
    return TRUE;
}

/* FUNCTION: void TermInit(void)
*
* PURPOSE:      This function initializes the client terminal structure it is called when the
TPCC.DLL

```

```

*              is first loaded by the inet service.
*
* ARGUMENTS:   none
*
* RETURNS:     None
*
* COMMENTS:    None
*/

static void TermInit(void)
{
    if ( Term.bInit )
        return;
    Term.iNext = 0;
    Term.iMasterSyncId = 1;

    Term.iAvailable = 0;
    Term.pClientData = NULL;
    Term.bInit = TRUE;

    return;
}

/* FUNCTION: void TermRestore(void)
*
* PURPOSE:      This function frees allocated resources associated with the terminal
structure.
*
* ARGUMENTS:   none
*
* RETURNS:     None
*
* COMMENTS:    This function is called only with the inet service unloads the TPCC.DLL
*/

static void TermRestore(void)
{
    Term.iNext = 0;
    Term.iAvailable = 0;
    Term.iMasterSyncId = 0;
    if ( Term.pClientData )
        free(Term.pClientData);
    Term.pClientData = NULL;
    Term.bInit = FALSE;
}

```



```

        return;
    }

/* FUNCTION: int TermAllocate(void)
 *
 * PURPOSE:      This function allocates more terminal array entries in the Term structure.
 *
 * ARGUMENTS:    None
 *
 * RETURNS:      int      TRUE or 1 if sucessfull
 *               int      FALSE or 0 if terminal id cannot be
allocated.
 *
 * COMMENTS:     None
 */

static int TermAllocate(void)
{
    Term.iAvailable += 32;
    if ( !Term.pClientData )
        Term.pClientData = (PCLIENTDATA)malloc(Term.iAvailable *
sizeof(CLIENTDATA));
    else
        Term.pClientData = (PCLIENTDATA)realloc(Term.pClientData,
Term.iAvailable * sizeof(CLIENTDATA));
    return ( Term.pClientData ) ? 1: 0;
}

/* FUNCTION: int TermAdd(EXTENSION_CONTROL_BLOCK *pECB, char *pQueryString)
 *
 * PURPOSE:      This function assigns a terminal id which is used to identify a client browser.
 *
 * ARGUMENTS:    EXTENSION_CONTROL_BLOCK      *pECB
 *               char
 *               *pQueryString      http query string passed to this DLL.
 *
 * RETURNS:      int      assigned terminal id
 *               -1      cannot assign id error occured.
 *
 * COMMENTS:     if the terminal id cannot be assigned it is because of insufficient memory or
the
 *               SQL connection cannot be allocated.
 */

```

```

static int TermAdd(EXTENSION_CONTROL_BLOCK *pECB, char *pQueryString)
{
    char      szTmp[32];
    int       i, iCurrent, iTotConnections, iTickCount;

    EnterCriticalSection(&CriticalSection);

    for(i=0, iTotConnections = 0; i<Term.iAvailable; i++)
    {
        if ( Term.pClientData[i].inUse )
            iTotConnections++;
    }

    if ( iTotConnections >= iMaxConnections )
    {
        for(iCurrent = 1, i=1, iTickCount = 0x7FFFFFFF; i<iMaxConnections; i++)
        {
            if ( iTickCount > Term.pClientData[i].iTickCount )
            {
                iTickCount = Term.pClientData[i].iTickCount;
                iCurrent = i;
            }
        }
    }
    else
    {
        for(i=0; i<Term.iAvailable; i++)
        {
            if ( !Term.pClientData[i].inUse )
                break;
        }
        iCurrent = i;
    }

    if ( i == Term.iAvailable )
    {
        Term.iNext = Term.iAvailable;
        if ( !(*Term.Allocate)() )
            goto TermAddErr1;
        for(i=Term.iNext; i<Term.iAvailable; i++)
            Term.pClientData[i].inUse = 0;
        iCurrent = Term.iNext;
    }

    Term.pClientData[iCurrent].inUse = 1;

    if ( !GetKeyValue(pQueryString, "w_id", szTmp, sizeof(szTmp)) )
        goto TermAddErr1;
}

```

```

Term.pClientData[iCurrent].w_id = (short)atoi(szTmp);

if ( !GetKeyValue(pQueryString, "d_id", szTmp, sizeof(szTmp)) )
    goto TermAddErr1;

Term.pClientData[iCurrent].d_id = atoi(szTmp);

Term.pClientData[iCurrent].iTickCount = GetTickCount();
Term.pClientData[iCurrent].iSyncId = Term.iMasterSyncId++;

if ( Init(pECB, iCurrent, Term.pClientData[iCurrent].iSyncId, szServer, szUser,
szPassword, szDatabase) )
{
    (*Term.Delete)(pECB, iCurrent);
    goto TermAddErr1;
}

LeaveCriticalSection(&CriticalSection);
return iCurrent;

TermAddErr1:
LeaveCriticalSection(&CriticalSection);
return -1; //terminal unsuccessfully added
}

/* FUNCTION: void TermDelete(EXTENSION_CONTROL_BLOCK *pECB, int id)
*
* PURPOSE:      This function makes a terminal entry in the Term array available for reuse.
*
* ARGUMENTS:   EXTENSION_CONTROL_BLOCK    *pECB    passed in structure
pointer from inetsrv.
*
*              int
*              id            Terminal id of client exiting
*
* RETURNS:     None
*
* COMMENTS:    None
*/

static void TermDelete(EXTENSION_CONTROL_BLOCK *pECB, int id)
{
    if ( id >= 0 && id < Term.iAvailable )
    {
        Close(pECB, id, -1);
        Term.pClientData[id].inUse = 0;
    }
}

```

```

#ifdef LOCAL_ALLOC
    tpfree((char *)Term.pClientData[id]TuxDataPtr);
#endif // Not LOCAL_ALLOC

}

return;
}

/* FUNCTION: BOOL Init(EXTENSION_CONTROL_BLOCK *pECB, int iTermId, int iSyncId,
char *szServer, char *szUser, char *szPassword, char *szDatabase)
*
* PURPOSE:      This function initializes the sql connection for use.
*
* ARGUMENTS:   EXTENSION_CONTROL_BLOCK    *pECB            passed in
structure pointer from inetsrv.
*
*              int
*              iTermId        id of browser client that this connection is for.
*              int
*              iSyncId        sync id for this client session
*              char
*              *szServer      sql server name
*              char
*              *szUser        user name
*              char
*              *szPassword    user password
*              char
*              *szDatabase    database to use
*
* RETURNS:     BOOL    FALSE    if successful
*              TRUE     if an error occurs an
connection cannot be established.
*
* COMMENTS:    None
*/

BOOL Init(EXTENSION_CONTROL_BLOCK *pECB, int iTermId, int iSyncId, char *szServer
char *szUser, char *szPassword, char *szDatabase)
{
    char    szApp[32];
#ifdef LOCAL_ALLOC
    char    buf[64];
    int     TpRc;
#endif // Not LOCAL_ALLOC

    sprintf(szApp, "TPCC:%ld", (int)iTermId);

```

```

Term.pClientData[iTermId].dbproc = NULL;

#ifndef LOCAL_ALLOC // Globally allocate tuxedo structures

    if ( dLog )
    {
        FILE *fp;
        fp = fopen(szTpccLogPath, "ab");
        fprintf(fp, "Sizeof(TUX_DATA) %d \r\n", sizeof(TUX_DATA));
        fprintf(fp, "Sizeof(NEW_ORDER_DATA) %d \r\n",
sizeof(NEW_ORDER_DATA));
        fprintf(fp, "Sizeof(PAYMENT_DATA) %d \r\n",
sizeof(PAYMENT_DATA));
        fprintf(fp, "Sizeof(ORDER_STATUS_DATA) %d \r\n",
sizeof(ORDER_STATUS_DATA));
        fprintf(fp, "Sizeof(DELIVERY_DATA) %d \r\n",
sizeof(DELIVERY_DATA));
        fprintf(fp, "Sizeof(STOCK_LEVEL_DATA) %d \r\n",
sizeof(STOCK_LEVEL_DATA));
        fclose(fp);
    }

// Add initialization of Tuxedo Structures

    if ((Term.pClientData[iTermId].TuxDataPtr = (TUX_DATA
*)tpalloc("CARRAY", NULL, sizeof(TUX_DATA))) == NULL)
    {
        TpRc = tperrno;
        sprintf(buf, "Tuxedo tpalloc failed:");
        LogTuxError(TpRc, buf);
        ErrorMessage(pECB, ERR_CANNOT_INIT_TERMINAL,
ERR_TYPE_WEBDLL, NULL, iTermId, iSynclD);
        return TRUE;
    }

    if ( dLog )
    {
        FILE *fp;

        fp = fopen(szTpccLogPath, "ab");

        fprintf(fp, "Thread %d iTermId %d * TuxDataPtr: %x \r\n",
GetCurrentThreadId(), iTermId,
&Term.pClientData[iTermId].TuxDataPtr);
        fclose(fp);
    }

#endif // LOCAL_ALLOC

```

```

    return FALSE;
}

/* FUNCTION: BOOL Close(EXTENSION_CONTROL_BLOCK *pECB, int iTermId,
iSynclD)
*
* PURPOSE: This function closes the sql connection for use.
*
* ARGUMENTS: EXTENSION_CONTROL_BLOCK *pECB passed in structure
pointer from inetsrv.
*
* iTermId id of browser client that this connection is for.
*
* iSynclD sync id of client browser
*
* RETURNS: BOOL FALSE if successfull
*
* TRUE if an error occurs an
connection cannot be terminated.
*
* COMMENTS: None
*
*/

static BOOL Close(EXTENSION_CONTROL_BLOCK *pECB, int iTermId, int iSynclD)
{
    PECBINFO pEcbInfo;

    if (Term.pClientData[iTermId].dbproc != NULL)
    {
        if ( (pEcbInfo =
(PECBINFO)dbgetuserdata(Term.pClientData[iTermId].dbproc) )
        {
            pEcbInfo->iTermId = -1;
            pEcbInfo->iSynclD = -1;
            free(pEcbInfo); //free up user info
        }

// return SQLCloseConnection(pECB, Term.pClientData[iTermId].dbproc)
return 1;
    }

    UNUSEDPARAM(iSynclD);
}

/* FUNCTION: void FormatString(char *szDest, char *szPic, char *szSrc)
*
* PURPOSE: This function formats a character string for inclusion in the

```

```

*           HTML formatted page being constructed.
*
* ARGUMENTS:  char          *szDest  Destination buffer where formatted string is
to be placed
*           char          *szPic   picture string which
describes how character value is to be
*           formatted.
*           char          *szSrc   character string value.
*
* RETURNS:    None
*
* COMMENTS:   This functions is used to format TPC-C phone and zip value strings.
*/

static void FormatString(char *szDest, char *szPic, char *szSrc)
{
    while( *szPic )
    {
        if ( *szPic == 'X' )
        {
            if ( *szSrc )
                *szDest++ = *szSrc++;
            else
                *szDest++ = ' ';
        }
        else
            *szDest++ = *szPic;
        szPic++;
    }
    *szDest = 0;

    return;
}

/* FUNCTION: char *MakeStockLevelForm(int iTermId, int iSyncId, BOOL bInput)
*
* PURPOSE:    This function constructs the Stock Level HTML page.
*
* ARGUMENTS:  int          iTermId  client browser terminal
id
*           int          iSyncId
client browser sync id
*           BOOL         bInput    TRUE if
form is being constructed for input else FALSE
*/

```

```

* RETURNS:    char *           A pointer to buffer inside client
structure where HTML form is built.
*
* COMMENTS:   The internal client buffer is created when the terminal id is assigned and
should not
*           be freed except when the client terminal id is no longer
needed.
*/

static char *MakeStockLevelForm(int iTermId, int iSyncId, BOOL bInput)
{
    char      *szForm;

    szForm = (char *)Term.pClientData[iTermId].szBuffer;

    Term.pClientData[iTermId].StockLevelData.w_id           =
(short)Term.pClientData[iTermId].w_id;
    Term.pClientData[iTermId].StockLevelData.d_id           =
(short)Term.pClientData[iTermId].d_id;
    Term.pClientData[iTermId].StockLevelData.num_deadlocks = 0;

    strcpy(szForm, "<HTML><HEAD><TITLE>TPC-C Stock Level</TITLE></HEAD>");
    strcat(szForm, "<FORM ACTION=\"tpcc.dll\" METHOD=\"GET\">");
    if ( bInput )
        strcat(szForm, "<INPUT TYPE=\"hidden\" NAME=\"PI*\" VALUE=\"\">");
    strcat(szForm, "<INPUT TYPE=\"hidden\" NAME=\"STATUSID\" VALUE=\"0\">");
    strcat(szForm, "<INPUT TYPE=\"hidden\" NAME=\"ERROR\" VALUE=\"0\">");
    sprintf(szForm+strlen(szForm), "<INPUT TYPE=\"hidden\" NAME=\"FORMID\"
VALUE=\"%d\">", STOCK_LEVEL_FORM);
    sprintf(szForm+strlen(szForm), "<INPUT TYPE=\"hidden\" NAME=\"TERMIID\"
VALUE=\"%d\">", iTermId);
    sprintf(szForm+strlen(szForm), "<INPUT TYPE=\"hidden\" NAME=\"SYNCID\"
VALUE=\"%d\">", iSyncId);
    strcat(szForm, "<PRE>                               Stock-Level<BR>");
    sprintf(szForm+strlen(szForm), "Warehouse: %4.4d District: %2.2d<BR><BR>";
Term.pClientData[iTermId].StockLevelData.w_id,
Term.pClientData[iTermId].StockLevelData.d_id);
    if ( bInput )
    {
        strcat(szForm, "Stock Level Threshold: <INPUT NAME=\"TT*\"
SIZE=2<<BR><BR>"
"low stock: <BR><HR>"
"<INPUT TYPE=\"submit\"\"
NAME=\"CMD\" VALUE=\"Process\">"
"<INPUT TYPE=\"submit\"\"
NAME=\"CMD\" VALUE=\"Menu\">");
    }
    else

```

```

    {
        sprintf(szForm+strlen(szForm), "Stock Level Threshold:
        %2.2d<BR><BR>", Term.pClientData[iTermId].StockLevelData.thresh_hold);

        sprintf(szForm+strlen(szForm), "low stock: %3.3d<PRE><BR><HR>",
        Term.pClientData[iTermId].StockLevelData.low_stock);
        strcat(szForm, "<INPUT TYPE='submit' NAME='CMD'
        VALUE='..NewOrder..'>"
                "<INPUT TYPE='submit'
        NAME='CMD' VALUE='..Payment..'>"
                "<INPUT TYPE='submit'
        NAME='CMD' VALUE='..Delivery..'>"
                "<INPUT TYPE='submit'
        NAME='CMD' VALUE='..Order-Status..'>"
                "<INPUT TYPE='submit'
        NAME='CMD' VALUE='..Stock-Level..'>"
                "<INPUT TYPE='submit'
        NAME='CMD' VALUE='..Exit..'>" );
    }

    strcat(szForm, "</FORM></HTML>");

    return szForm;
}

/* FUNCTION: char *MakeMainMenuForm(int iTermId, int iSyncId)
 *
 * PURPOSE:      This function
 *
 * ARGUMENTS:   int iTermId client browser terminal
 *              int iSyncId client browser sync id
 *
 * RETURNS:     char * A pointer to buffer inside client
 *              structure where HTML form is built.
 *
 * COMMENTS:    The internal client buffer is created when the terminal id is assigned and
 *              should not be freed except when the client terminal id is no longer
 *              needed.
 */

static char *MakeMainMenuForm(int iTermId, int iSyncId)
{
    char *szForm;

    szForm = (char *)Term.pClientData[iTermId].szBuffer;

```

```

        strcpy(szForm, "<HTML><HEAD><TITLE>TPC-C Main
        Menu</TITLE></HEAD><BODY>"
                "Select Desired Transaction.<BR><HR>"
                "<FORM ACTION='tpcc.dll'"
        METHOD="GET">");
        strcat(szForm, "<INPUT TYPE='hidden' NAME='STATUSID' VALUE='0'>");
        strcat(szForm, "<INPUT TYPE='hidden' NAME='ERROR' VALUE='0'>");
        sprintf(szForm+strlen(szForm), "<INPUT TYPE='hidden' NAME='TERMINID'"
        VALUE='%d'>", iTermId);
        sprintf(szForm+strlen(szForm), "<INPUT TYPE='hidden' NAME='SYNCDID'"
        VALUE='%d'>", iSyncId);
        sprintf(szForm+strlen(szForm), "<INPUT TYPE='hidden' NAME='FORMID'"
        VALUE='%d'>", MAIN_MENU_FORM);
        strcat(szForm, "<INPUT TYPE='submit' NAME='CMD'"
        VALUE='..NewOrder..'>"
                "<INPUT TYPE='submit' NAME='CMD'"
        VALUE='..Payment..'>"
                "<INPUT TYPE='submit' NAME='CMD'"
        VALUE='..Delivery..'>"
                "<INPUT TYPE='submit' NAME='CMD'"
        VALUE='..Order-Status..'>"
                "<INPUT TYPE='submit' NAME='CMD'"
        VALUE='..Stock-Level..'>"
                "<INPUT TYPE='submit' NAME='CMD'"
        VALUE='..Exit..'>"
                "</FORM>"
                "</HTML>");

    return szForm;
}

/* FUNCTION: char *MakeWelcomeForm(void)
 *
 * PURPOSE:      This function
 *
 * ARGUMENTS:   None
 *
 * RETURNS:     char * A pointer to the static HTML
 *              welcome form.
 *
 * COMMENTS:    The welcome form is static.
 */

static char *MakeWelcomeForm(void)
{
    return szWelcomeForm;
}

```

```

/* FUNCTION: char *MakeNewOrderForm(int iTermId, int iSyncId, BOOL Rollback, BOOL
bInput, BOOL bValid)
*
* PURPOSE:      This function
*
* ARGUMENTS:   int          iTermId  client browser terminal
id
*              int          iSyncId
client browser sync id
*              BOOL         bInput   TRUE if
form is being constructed for input else FALSE
*              BOOL         bValid   TRUE if
NewOrderData valid, ELSE FALSE effects output only
*
* RETURNS:     char *        A pointer to buffer inside client
structure where HTML form is built.
*
* COMMENTS:    The internal client buffer is created when the terminal id is assigned and
should not
*              be freed except when the client terminal id is no longer
needed.
*/

static char *MakeNewOrderForm(int iTermId, int iSyncId, BOOL Rollback, BOOL bInput, BOOL
bValid)
{
    char    *szForm;
    char    szName[146];
    char    szCredit[14];
    int     i;

    szForm = (char *)Term.pClientData[iTermId].szBuffer;

    Term.pClientData[iTermId].NewOrderData.w_id = Term.pClientData[iTermId].w_id;

    strcpy(szForm, "<HTML>"
Order</TITLE></HEAD><BODY>"
METHOD="GET">");

    if ( bInput )
    {
        strcat(szForm, "<INPUT TYPE='hidden' NAME='PI' VALUE='\">");
        strcat(szForm, "<INPUT TYPE='hidden' NAME='STATUSID"
VALUE="0">");
    }

```

```

else
{
    if ( bValid )
        strcat(szForm, "<INPUT TYPE='hidden\"
NAME='STATUSID' VALUE='\">");
    else
        sprintf(szForm+strlen(szForm), "<INPUT TYPE='hidden\"
NAME='STATUSID' VALUE='\"%d\">", ERR_BAD_ITEM_ID);
}

    if (Rollback == FALSE)
        strcat(szForm, "<INPUT TYPE='hidden\" NAME='ERROR\"
VALUE='\">");
    else
        strcat(szForm, "<INPUT TYPE='hidden\" NAME='ERROR\"
VALUE='\"1\">");

        sprintf(szForm+strlen(szForm), "<INPUT TYPE='hidden\" NAME='FORMID\"
VALUE='\"%d\">", NEW_ORDER_FORM);
        sprintf(szForm+strlen(szForm),
NAME="TERMINID" VALUE="\"%d\">", iTermId);
        sprintf(szForm+strlen(szForm), "<INPUT TYPE='hidden\" NAME='SYNCDID\"
VALUE='\"%d\">", iSyncId);
        strcat(szForm, "<PRE>
New Order<BR>");

    if ( bInput )
    {
        sprintf(szForm+strlen(szForm), "Warehouse:%4.4d District: <INPUT
NAME='DID' SIZE=1> Date:<BR>",
Term.pClientData[iTermId].NewOrderData.w_id);
        strcat(szForm, "Customer: <INPUT NAME='CID' SIZE=4>
Name:
Credit: %Disc:<BR>"
"Order Number: Number
Lines: W_tax: D_tax:<BR><BR>"
" Supp_W Item_Id Item Name
Qty Stock B/G Price Amount<BR>"
" <INPUT NAME='SP00'\"
<INPUT NAME='Qty00'\"
SIZE=4> <INPUT NAME='IID00'\" SIZE=6>
SIZE=1><BR>"
" <INPUT NAME='SP01'\"
<INPUT NAME='Qty01'\"
SIZE=4> <INPUT NAME='IID01'\" SIZE=6>
SIZE=1><BR>"
" <INPUT NAME='SP02'\"
<INPUT NAME='Qty02'\"
SIZE=4> <INPUT NAME='IID02'\" SIZE=6>
SIZE=1><BR>"
" <INPUT NAME='SP03'\"
<INPUT NAME='Qty03'\"
SIZE=4> <INPUT NAME='IID03'\" SIZE=6>
SIZE=1><BR>"

```

```

        " <INPUT NAME=\"SP04*\"
        <INPUT NAME=\"Qty04*\"
    SIZE=4> <INPUT NAME=\"IID04*\" SIZE=6>
    SIZE=1><BR>"
        " <INPUT NAME=\"SP05*\"
        <INPUT NAME=\"Qty05*\"
    SIZE=4> <INPUT NAME=\"IID05*\" SIZE=6>
    SIZE=1><BR>"
        " <INPUT NAME=\"SP06*\"
        <INPUT NAME=\"Qty06*\"
    SIZE=4> <INPUT NAME=\"IID06*\" SIZE=6>
    SIZE=1><BR>"
        " <INPUT NAME=\"SP07*\"
        <INPUT NAME=\"Qty07*\"
    SIZE=4> <INPUT NAME=\"IID07*\" SIZE=6>
    SIZE=1><BR>"
        " <INPUT NAME=\"SP08*\"
        <INPUT NAME=\"Qty08*\"
    SIZE=4> <INPUT NAME=\"IID08*\" SIZE=6>
    SIZE=1><BR>"
        " <INPUT NAME=\"SP09*\"
        <INPUT NAME=\"Qty09*\"
    SIZE=4> <INPUT NAME=\"IID09*\" SIZE=6>
    SIZE=1><BR>"
        " <INPUT NAME=\"SP10*\"
        <INPUT NAME=\"Qty10*\"
    SIZE=4> <INPUT NAME=\"IID10*\" SIZE=6>
    SIZE=1><BR>"
        " <INPUT NAME=\"SP11*\"
        <INPUT NAME=\"Qty11*\"
    SIZE=4> <INPUT NAME=\"IID11*\" SIZE=6>
    SIZE=1><BR>"
        " <INPUT NAME=\"SP12*\"
        <INPUT NAME=\"Qty12*\"
    SIZE=4> <INPUT NAME=\"IID12*\" SIZE=6>
    SIZE=1><BR>"
        " <INPUT NAME=\"SP13*\"
        <INPUT NAME=\"Qty13*\"
    SIZE=4> <INPUT NAME=\"IID13*\" SIZE=6>
    SIZE=1><BR>"
        " <INPUT NAME=\"SP14*\"
        <INPUT NAME=\"Qty14*\"
    SIZE=4> <INPUT NAME=\"IID14*\" SIZE=6>
    SIZE=1><BR>"
    Total:<BR><HR>"
    NAME=\"CMD\" VALUE=\"Process\">"
    NAME=\"CMD\" VALUE=\"Menu\">"
    }
    else
    {
        if ( bValid )
        {

```

```

        wsprintf(szForm+strlen(szForm), "Warehouse: %4.4d Distric
    %2.2d Date: %2.2d-%2.2d-%4.4d %2.2d:%2.2d:%2.2d <BR>",
        Term.pClientData[iTermId].NewOrderData.w_id,
        Term.pClientData[iTermId].NewOrderData.d_id,
        Term.pClientData[iTermId].NewOrderData.o_entry_d.day,
        Term.pClientData[iTermId].NewOrderData.o_entry_d.month,
        Term.pClientData[iTermId].NewOrderData.o_entry_d.year,
        Term.pClientData[iTermId].NewOrderData.o_entry_d.hour,
        Term.pClientData[iTermId].NewOrderData.o_entry_d.minute,
        Term.pClientData[iTermId].NewOrderData.o_entry_d.second);
    }
    else
    {
        wsprintf(szForm+strlen(szForm), "Warehouse: %4.4d Distric
    %2.2d Date:<BR>",
        Term.pClientData[iTermId].NewOrderData.w_id,
        Term.pClientData[iTermId].NewOrderData.d_id);
    }
    FormatHTMLString(szName,
    Term.pClientData[iTermId].NewOrderData.c_last, 16),
    FormatHTMLString(szCredit,
    Term.pClientData[iTermId].NewOrderData.c_credit, 2);
    wsprintf(szForm+strlen(szForm), "Customer: %4.4d Name: %s Cred
    %s ",
        Term.pClientData[iTermId].NewOrderData.c_id, szName,
    szCredit);
    if ( bValid )
    {
        sprintf(szForm+strlen(szForm), "%%%Disc: %5.2f <BR>
    Term.pClientData[iTermId].NewOrderData.c_discount);
        sprintf(szForm+strlen(szForm), "Order Number: %8.8d Numt
    of Lines: %2.2d W_tax: %5.2f D_tax: %5.2f <BR><BR>",
        Term.pClientData[iTermId].NewOrderData.o_id,
        Term.pClientData[iTermId].NewOrderData.o_ol_cn
        Term.pClientData[iTermId].NewOrderData.w_tax,
        Term.pClientData[iTermId].NewOrderData.d_tax);

```

```

                strcat(szForm, " Supp_W Item_Id Item Name          Qty
Stock B/G Price  Amount<BR>");
                for(i=0; i<Term.pClientData[iTermId].NewOrderData.o_o_cnt;
i++)
                {
                    FormatHTMLString(szName,
Term.pClientData[iTermId].NewOrderData.Ol[i].ol_i_name, 24);

                    sprintf(szForm+strlen(szForm), " %4.4d %66d %s
%2.2d %3.3d %1.1s $%6.2f $%7.2f <BR>",

Term.pClientData[iTermId].NewOrderData.Ol[i].ol_supply_w_id,
Term.pClientData[iTermId].NewOrderData.Ol[i].ol_i_id,
szName,
Term.pClientData[iTermId].NewOrderData.Ol[i].ol_quantity,
Term.pClientData[iTermId].NewOrderData.Ol[i].ol_stock,
Term.pClientData[iTermId].NewOrderData.Ol[i].ol_brand_generic,
Term.pClientData[iTermId].NewOrderData.Ol[i].ol_i_price,
Term.pClientData[iTermId].NewOrderData.Ol[i].ol_amount );
                }
                else
                {
                    strcat(szForm, "%Disc:<BR>");
                    sprintf(szForm+strlen(szForm), "Order Number: %8.8d Number
of Lines:      W_tax:      D_tax:<BR><BR>",
Term.pClientData[iTermId].NewOrderData.o_id);

                    strcat(szForm, " Supp_W Item_Id Item Name          Qty
Stock B/G Price  Amount<BR>");

                    i = 0;
                }
                for(; i<15; i++)
                strcat(szForm, "<BR>");

                if ( bValid )
                {
                    sprintf(szForm+strlen(szForm), "Execution Status: %24.24s
Total: $%8.2f ",

Term.pClientData[iTermId].NewOrderData.execution_status,

```

```

Term.pClientData[iTermId].NewOrderData.total_amount);
                }
                else
                {
                    sprintf(szForm+strlen(szForm), "Execution Status: %24.24s
Total:",

Term.pClientData[iTermId].NewOrderData.execution_status);
                }

                strcat(szForm, "</PRE><HR><BR>"
NAME="CMD" VALUE="..NewOrder..">"
NAME="CMD" VALUE="..Payment..">"
NAME="CMD" VALUE="..Delivery..">"
NAME="CMD" VALUE="..Order-Status..">"
NAME="CMD" VALUE="..Stock-Level..">"
NAME="CMD" VALUE="..Exit..">" );
                strcat(szForm, "</FORM></HTML>");
                }

                return szForm;
            }

/* FUNCTION: char *MakePaymentForm(int iTermId, int iSynclD, BOOL bInput)
*
* PURPOSE:      This function
*
* ARGUMENTS:   int iTermId client browser termin
id
*              int iSynclD
*              client browser sync id
*              BOOL bInput TRUE if
form is being constructed for input else FALSE
*
* RETURNS:     char * A pointer to buffer inside client
structure where HTML form is built.
*
* COMMENTS:    The internal client buffer is created when the terminal id is assigned and
should not
*              be freed except when the client terminal id is no longer

```


needed.

*/

static char *MakePaymentForm(int iTermId, int iSyncId, BOOL bInput)

```
{
    char    *szForm;
    char    *ptr;
    char    szTmp[64];
    char    szW_Zip[26];
    char    szD_Zip[26];
    char    szC_Zip[26];
    char    szC_Phone[26];
    char    szTmpStr1[122];
    char    szTmpStr2[122];
    char    szTmpStr3[122];
    char    szTmpStr4[122];
    int     i;
    int     l;
    char    *szZipPic = "XXXXX-XXXX";

    szForm = (char *)Term.pClientData[iTermId].szBuffer;

    Term.pClientData[iTermId].PaymentData.w_id = Term.pClientData[iTermId].w_id;

    strcpy(szForm, "<HTML><HEAD><TITLE>TPC-C
Payment</TITLE></HEAD><BODY>"
          "<FORM ACTION=\"tpcc.dll\"
METHOD=\"GET\">");
    if ( bInput )
        strcat(szForm, "<INPUT TYPE=\"hidden\" NAME=\"PI\" VALUE=\"\">");

    strcat(szForm, "<INPUT TYPE=\"hidden\" NAME=\"STATUSID\" VALUE=\"0\">");
    strcat(szForm, "<INPUT TYPE=\"hidden\" NAME=\"ERROR\" VALUE=\"0\">");
    sprintf(szForm+strlen(szForm), "<INPUT TYPE=\"hidden\" NAME=\"FORMID\"
VALUE=\"%d\">", PAYMENT_FORM);
    sprintf(szForm+strlen(szForm), "<INPUT TYPE=\"hidden\" NAME=\"TERMIN\"
VALUE=\"%d\">", iTermId);
    sprintf(szForm+strlen(szForm), "<INPUT TYPE=\"hidden\" NAME=\"SYNCID\"
VALUE=\"%d\">", iSyncId);

    strcat(szForm, "<PRE>
Payment<BR>");

    if ( bInput )
        strcat(szForm, "Date:<BR><BR>");
    else
    {
        sprintf(szForm+strlen(szForm), "Date: %2.2d-%2.2d-%4.4d
%2.2d:%2.2d:%2.2d <BR><BR>",
```

```
Term.pClientData[iTermId].PaymentData.h_date.day,
Term.pClientData[iTermId].PaymentData.h_date.month,
Term.pClientData[iTermId].PaymentData.h_date.year,
Term.pClientData[iTermId].PaymentData.h_date.hour,
Term.pClientData[iTermId].PaymentData.h_date.minute,
Term.pClientData[iTermId].PaymentData.h_date.second);
    }
    sprintf(szForm+strlen(szForm), "Warehouse: %4.4d",
Term.pClientData[iTermId].PaymentData.w_id);

    if ( bInput )
    {
        strcat(szForm, "
District: <INPUT NAME=\"DID\"
SIZE=1><BR><BR><BR><BR><BR>"
                "Customer: <INPUT
NAME=\"CID\" SIZE=4>"
                "Cust-Warehouse: <INPUT
NAME=\"CWI\" SIZE=4> "
                "Cust-District: <INPUT
NAME=\"CDI\" SIZE=1><BR>"
                "Name: <INPUT
NAME=\"CLT\" SIZE=16> Since:<BR>"
                "
Credit:<BR>"
                "
Disc:<BR>"
                "
Phone:<BR><BR>"
                "Amount Paid: $<INPUT
NAME=\"HAM\" SIZE=7> New Cust Balance:<BR>"
                "Credit Limit:<BR><BR>Cust-
Data: <BR><BR><BR><BR></PRE><HR>"
                "<INPUT TYPE=\"submit\"
NAME=\"CMD\" VALUE=\"Process\"><INPUT TYPE=\"submit\" NAME=\"CMD\"
VALUE=\"Menu\">"
                "</BODY></FORM></HTML>"
    }
    else
    {
        sprintf(szForm+strlen(szForm),
                "District: %2.2d<BR>",
                Term.pClientData[iTermId].PaymentData.d_id);

        FormatHTMLString(szTmpStr1,
Term.pClientData[iTermId].PaymentData.w_street_1, 20);
        FormatHTMLString(szTmpStr2,
Term.pClientData[iTermId].PaymentData.d_street_1, 20);
```

```

    sprintf(szForm+strlen(szForm),"%s          %s<BR>", szTmpStr1,
szTmpStr2);

    FormatHTMLString(szTmpStr1,
Term.pClientData[iTermId].PaymentData.w_street_2, 20);
    FormatHTMLString(szTmpStr2,
Term.pClientData[iTermId].PaymentData.d_street_2, 20);

    sprintf(szForm+strlen(szForm),"%s          %s<BR>", szTmpStr1,
szTmpStr2);

    FormatString(szW_Zip, szZipPic,
Term.pClientData[iTermId].PaymentData.w_zip);
    FormatString(szD_Zip, szZipPic,
Term.pClientData[iTermId].PaymentData.d_zip);

    FormatHTMLString(szTmpStr1,
Term.pClientData[iTermId].PaymentData.w_city, 20);
    FormatHTMLString(szTmpStr2,
Term.pClientData[iTermId].PaymentData.w_state, 2);
    FormatHTMLString(szTmpStr3,
Term.pClientData[iTermId].PaymentData.d_city, 20);
    FormatHTMLString(szTmpStr4,
Term.pClientData[iTermId].PaymentData.d_state, 2);

    wsprintf(szForm+strlen(szForm), "%s %s %10.10s    %s %s
%10.10s<BR><BR>",
            szTmpStr1, szTmpStr2, szW_Zip, szTmpStr3, szTmpStr4,
szD_Zip );

    wsprintf(szForm+strlen(szForm), "Customer: %4.4d Cust-Warehouse:
%4.4d Cust-District: %2.2d<BR>",
            Term.pClientData[iTermId].PaymentData.c_id,
            Term.pClientData[iTermId].PaymentData.c_w_id,
            Term.pClientData[iTermId].PaymentData.c_d_id);

    FormatHTMLString(szTmpStr1,
Term.pClientData[iTermId].PaymentData.c_first, 16);
    FormatHTMLString(szTmpStr2,
Term.pClientData[iTermId].PaymentData.c_middle, 2);
    FormatHTMLString(szTmpStr3,
Term.pClientData[iTermId].PaymentData.c_last, 16);

    wsprintf(szForm+strlen(szForm), "Name: %s %s %s Since: %2.2d-
%2.2d-%4.4d<BR>",
            szTmpStr1, szTmpStr2, szTmpStr3,
            Term.pClientData[iTermId].PaymentData.c_since.day,
            Term.pClientData[iTermId].PaymentData.c_since.month,
            Term.pClientData[iTermId].PaymentData.c_since.year);

    FormatHTMLString(szTmpStr1,
Term.pClientData[iTermId].PaymentData.c_street_1, 20);
    FormatHTMLString(szTmpStr2,
Term.pClientData[iTermId].PaymentData.c_credit, 2);

    wsprintf(szForm+strlen(szForm), "    %s          Credit: %s<BR>", szTmpStr1,
szTmpStr2);

    FormatHTMLString(szTmpStr1,
Term.pClientData[iTermId].PaymentData.d_street_2, 20);
    sprintf(szForm+strlen(szForm), "    %s          %%Disc: %5.2f<BR>",
            szTmpStr1,
            Term.pClientData[iTermId].PaymentData.c_discount);

    FormatString(szC_Zip, szZipPic,
Term.pClientData[iTermId].PaymentData.c_zip);
    FormatString(szC_Phone, "XXXXXX-XXX-XXX-XXXX",
Term.pClientData[iTermId].PaymentData.c_phone);

    FormatHTMLString(szTmpStr1,
Term.pClientData[iTermId].PaymentData.c_city, 20);
    FormatHTMLString(szTmpStr2,
Term.pClientData[iTermId].PaymentData.c_state, 2);

    wsprintf(szForm+strlen(szForm), "    %s %s %10.10s    Phone: %-
19.19s<BR><BR>",
            szTmpStr1, szTmpStr2, szC_Zip, szC_Phone );

    sprintf(szForm+strlen(szForm), "Amount Paid:    $%7.2f    New Cu-
Balance: $%14.2f<BR>",
            Term.pClientData[iTermId].PaymentData.h_amount,
            Term.pClientData[iTermId].PaymentData.c_balance);

    sprintf(szForm+strlen(szForm), "Credit Limit: $%13.2f<BR><BR>",
            Term.pClientData[iTermId].PaymentData.c_credit_lim);

    ptr = Term.pClientData[iTermId].PaymentData.c_credit;
    if ( *ptr == 'B' && *(ptr+1) == 'C' )
    {
        ptr = Term.pClientData[iTermId].PaymentData.c_data;
        l = strlen( ptr ) / 50;
        for(i=0; i<4; i++, ptr += 50)
        {
            if ( i <= l )
                UtilStrCpy(szTmp, ptr, 50);
            else

```

```

        szTmp[0] = 0;
        if ( li )
        {
            FormatHTMLString(szTmpStr1, szTmp,
50);
            Data: %s<BR>", szTmpStr1);
        }
        else
        {
            FormatHTMLString(szTmpStr1, szTmp,
50);
            %s<BR>", szTmpStr1);
        }
    }
    else
        strcat(szForm, "Cust-Data: <BR><BR><BR><BR>");

        strcat(szForm, "</PRE><HR><BR>"
            "NAME=\"CMD\" VALUE=\"..NewOrder..\">"
            "NAME=\"CMD\" VALUE=\"..Payment..\">"
            "NAME=\"CMD\" VALUE=\"..Delivery..\">"
            "NAME=\"CMD\" VALUE=\"..Order-Status..\">"
            "NAME=\"CMD\" VALUE=\"..Stock-Level..\">"
            "NAME=\"CMD\" VALUE=\"..Exit..\">"
            "</BODY></FORM></HTML>");
    }
    return szForm;
}

/* FUNCTION: char *MakeOrderStatusForm(int iTermId, int iSyncId, BOOL bInput)
 *
 * PURPOSE:      This function
 *
 * ARGUMENTS:   int iTermId client browser terminal
 *              int iSyncId client browser sync id

```

```

 *
 *          BOOL bInput TRUE if
form is being constructed for input else FALSE
 *
 * RETURNS:      char * A pointer to buffer inside client
structure where HTML form is built.
 *
 * COMMENTS:    The internal client buffer is created when the terminal id is assigned and
should not be freed except when the client terminal id is no longer
needed.
 */

static char *MakeOrderStatusForm(int iTermId, int iSyncId, BOOL bInput)
{
    char *szForm;
    char c_first[98];
    char c_middle[14];
    char c_last[98];
    int i;

    szForm = (char *)Term.pClientData[iTermId].szBuffer;

    Term.pClientData[iTermId].OrderStatusData.w_id = Term.pClientData[iTermId].w_

    strcpy(szForm, "<HTML><HEAD><TITLE>TPC-C Order-
Status</TITLE></HEAD><BODY>"
            "<FORM ACTION=\"tpcc.dll"
METHOD=\"GET\">");

    if ( bInput )
        strcat(szForm, "<INPUT TYPE=\"hidden\" NAME=\"PI*\" VALUE=\"\">");

    strcat(szForm, "<INPUT TYPE=\"hidden\" NAME=\"STATUSID\" VALUE=\"0\">");
    strcat(szForm, "<INPUT TYPE=\"hidden\" NAME=\"ERROR\" VALUE=\"0\">");
    sprintf(szForm+strlen(szForm), "<INPUT TYPE=\"hidden\" NAME=\"FORMID\"
VALUE=\"%d\">", ORDER_STATUS_FORM);
    sprintf(szForm+strlen(szForm), "<INPUT TYPE=\"hidden\" NAME=\"TERMINID\"
VALUE=\"%d\">", iTermId);
    sprintf(szForm+strlen(szForm), "<INPUT TYPE=\"hidden\" NAME=\"SYNCID\"
VALUE=\"%d\">", iSyncId);

    strcat(szForm, "<PRE> Order-Status<BR> ");
    sprintf(szForm+strlen(szForm), "Warehouse: %4.4d ",
Term.pClientData[iTermId].OrderStatusData.w_id);

    if ( bInput )
    {
        strcat(szForm, "District: <INPUT NAME=\"DID*\" SIZE=1><BR>"

```

```

NAME="CID" SIZE=4> Name:          "Customer: <INPUT
                                <INPUT NAME="CLT" SIZE=23><BR>"
                                "Cust-Balance:<BR><BR>"
                                "Order-Number:      Entry-
Date:          Carrier-Number:<BR>"
                                "Supply-W  Item-Id  Qty
Amount  Delivery-Date<BR></PRE>"
                                "<HR><INPUT TYPE="submit"
NAME="CMD" VALUE="Process"><INPUT TYPE="submit" NAME="CMD"
VALUE="Menu">"
                                "</BODY></FORM></HTML>" );
    }
    else
    {
        wsprintf(szForm+strlen(szForm), "District: %2.2d<BR>",
Term.pClientData[iTermId].OrderStatusData.d_id);

        FormatHTMLString(c_first,
Term.pClientData[iTermId].OrderStatusData.c_first, 16);
        FormatHTMLString(c_middle,
Term.pClientData[iTermId].OrderStatusData.c_middle, 2);
        FormatHTMLString(c_last,
Term.pClientData[iTermId].OrderStatusData.c_last, 16);

        wsprintf(szForm+strlen(szForm), "Customer: %4.4d  Name: %s %s
%s<BR>",
                Term.pClientData[iTermId].OrderStatusData.c_id, c_first,
c_middle, c_last);

        sprintf(szForm+strlen(szForm), "Cust-Balance: %9.2f<BR><BR>",
                Term.pClientData[iTermId].OrderStatusData.c_balance);

        wsprintf(szForm+strlen(szForm), "Order-Number: %8.8d  Entry-Date:
%2.2d-%2.2d-%4.4d %2.2d-%2.2d-%2.2d  Carrier-Number: %2.2d<BR>",
                Term.pClientData[iTermId].OrderStatusData.o_id,
                Term.pClientData[iTermId].OrderStatusData.o_entry_d.day,
                Term.pClientData[iTermId].OrderStatusData.o_entry_d.month,
                Term.pClientData[iTermId].OrderStatusData.o_entry_d.year,
                Term.pClientData[iTermId].OrderStatusData.o_entry_d.hour,
                Term.pClientData[iTermId].OrderStatusData.o_entry_d.minute,
                Term.pClientData[iTermId].OrderStatusData.o_entry_d.second,
                Term.pClientData[iTermId].OrderStatusData.o_carrier_id);
        strcat(szForm+strlen(szForm), "Supply-W  Item-Id  Qty  Amount
Delivery-Date<BR>");

        for(i=0; i<Term.pClientData[iTermId].OrderStatusData.o_o_cnt; i++)
    {

```

```

                                sprintf(szForm+strlen(szForm), " %4.4d  %6.6d  %2.2d
%8.2f  %2.2d-%2.2d-%4.4d<BR>",
                Term.pClientData[iTermId].OrderStatusData.OIOrderStatusData[i].ol_supply_w_id,
                Term.pClientData[iTermId].OrderStatusData.OIOrderStatusData[i].ol_i_id,
                Term.pClientData[iTermId].OrderStatusData.OIOrderStatusData[i].ol_quantity,
                Term.pClientData[iTermId].OrderStatusData.OIOrderStatusData[i].ol_amount,
                Term.pClientData[iTermId].OrderStatusData.OIOrderStatusData[i].ol_delivery_d.d
                Term.pClientData[iTermId].OrderStatusData.OIOrderStatusData[i].ol_delivery_d.mc
h,
                Term.pClientData[iTermId].OrderStatusData.OIOrderStatusData[i].ol_delivery_d.y
                }

        strcat(szForm,      "<BR></PRE><HR><INPUT TYPE="submit"
NAME="CMD" VALUE="..NewOrder..">"
                "<INPUT TYPE="submit"
NAME="CMD" VALUE="..Payment..">"
                "<INPUT TYPE="submit"
NAME="CMD" VALUE="..Delivery..">"
                "<INPUT TYPE="submit"
NAME="CMD" VALUE="..Order-Status..">"
                "<INPUT TYPE="submit"
NAME="CMD" VALUE="..Stock-Level..">"
                "<INPUT TYPE="submit"
NAME="CMD" VALUE="..Exit..">"
                "</BODY></FORM></HTML>"
    }

    return szForm;
}

/* FUNCTION: char *MakeDeliveryForm(int iTermId, int iSyncl, BOOL bInput, BOOL
bSuccess)
*
* PURPOSE:      This function
*
* ARGUMENTS:   int iTermId  client browser termid
*              int iSyncl
*              client browser sync id
*              BOOL bInput  TRUE if
form is being constructed for input else FALSE

```

```

*                               BOOL                               bSuccess TRUE if
Delivery succeeded else FALSE
*
* RETURNS:      char *      A pointer to buffer inside client
structure where HTML form is built.
*
* COMMENTS:    The internal client buffer is created when the terminal id is assigned and
should not
*              be freed except when the client terminal id is no longer
needed.
*/

static char *MakeDeliveryForm(int iTermId, int iSynclId, BOOL bInput, BOOL bSuccess)
{
    char    *szForm;

    szForm = (char *)Term.pClientData[iTermId].szBuffer;

    Term.pClientData[iTermId].DeliveryData.w_id = Term.pClientData[iTermId].w_id;

    strcpy( szForm, "<HTML><HEAD><TITLE>TPC-C
Delivery</TITLE></HEAD><BODY>"
           "<FORM ACTION=\"tpcc.dll\"
METHOD=\"GET\">");

    if ( bInput )
    {
        strcat(szForm, "<INPUT TYPE=\"hidden\" NAME=\"PI\" VALUE=\"\">");
        strcat(szForm, "<INPUT TYPE=\"hidden\" NAME=\"STATUSID\"
VALUE=\"0\">");
    }
    else
    {
        if ( !bSuccess )
        {
            sprintf(szForm+strlen(szForm), "<INPUT TYPE=\"hidden\"
NAME=\"STATUSID\" VALUE=\"%d\">", ERR_TYPE_DELIVERY_POST);
            strcat(szForm, "<INPUT TYPE=\"hidden\" NAME=\"ERROR\"
VALUE=\"2\">");
        }
        else
        {
            strcat(szForm, "<INPUT TYPE=\"hidden\"
NAME=\"STATUSID\" VALUE=\"0\">");
            strcat(szForm, "<INPUT TYPE=\"hidden\" NAME=\"ERROR\"
VALUE=\"0\">");
        }
    }
}

```

```

        wsprintf(szForm+strlen(szForm), "<INPUT TYPE=\"hidden\" NAME=\"FORMID\"
VALUE=\"%d\">", DELIVERY_FORM);
        wsprintf(szForm+strlen(szForm), "<INPUT TYPE=\"hidden\" NAME=\"TERMIID\"
VALUE=\"%d\">", iTermId);
        wsprintf(szForm+strlen(szForm), "<INPUT TYPE=\"hidden\" NAME=\"SYNCID\"
VALUE=\"%d\">", iSynclId);

        strcat(szForm, "<PRE>
Delivery<BR>");

        wsprintf(szForm+strlen(szForm), "Warehouse: %4.4d<BR><BR>",
Term.pClientData[iTermId].DeliveryData.w_id);

        if ( bInput )
            strcat( szForm, "Carrier Number: <INPUT NAME=\"OCD\"
SIZE=1><BR><BR>");
        else
        {
            wsprintf(szForm+strlen(szForm), "Carrier Number: %2.2d<BR><BR>",
Term.pClientData[iTermId].DeliveryData.o_carrier_id);
        }
        if ( bInput )
        {
            strcat( szForm, "Execution Status:<BR></PRE>"
                   "<HR><INPUT TYPE=\"submit\"
NAME=\"CMD\" VALUE=\"Process\">"
                   "<INPUT TYPE=\"submit\"
NAME=\"CMD\" VALUE=\"Menu\">");
        }
        else
        {
            wsprintf(szForm+strlen(szForm), "Execution Status:
%25.25s<BR></PRE>",
Term.pClientData[iTermId].DeliveryData.execution_status);

            strcat(szForm, "<HR><INPUT TYPE=\"submit\" NAME=\"CMD\"
VALUE=\"..NewOrder..\">"
                   "<INPUT TYPE=\"submit\"
NAME=\"CMD\" VALUE=\"..Payment..\">"
                   "<INPUT TYPE=\"submit\"
NAME=\"CMD\" VALUE=\"..Delivery..\">"
                   "<INPUT TYPE=\"submit\"
NAME=\"CMD\" VALUE=\"..Order-Status..\">"
                   "<INPUT TYPE=\"submit\"
NAME=\"CMD\" VALUE=\"..Stock-Level..\">"
                   "<INPUT TYPE=\"submit\"
NAME=\"CMD\" VALUE=\"..Exit..\">");
        }
    }
}

```

```

        strcat( szForm,      "</BODY></FORM></HTML>" );

        return szForm;
    }

/* FUNCTION: void UtilStrCpy(char * pDest, char * pSrc, int n)
 *
 * PURPOSE:      This function copies n characters from string pSc to pDst and places a
 *                null character at the end of the destination string.
 *
 * ARGUMENTS:   char          *pDest  destination string pointer
 *                char          *pSrc   source
 *                string pointer
 *                int           n
 *                number of characters to copy
 *
 * RETURNS:      None
 *
 * COMMENTS:     Unlike strncpy this function ensures that the result string is
 *                always null terminated.
 *
 */

static void UtilStrCpy(char * pDest, char * pSrc, int n)
{
    strncpy(pDest, pSrc, n);
    pDest[n] = '\0';

    return;
}

/* FUNCTION: void ProcessNewOrderForm(EXTENSION_CONTROL_BLOCK *pECB, int
iTermId, int iSyncId)
 *
 * PURPOSE:      This function gets and validates the input data from the new order form
 *                filling in the required input variables. it then calls the
SQLNewOrder
 *                transaction, constructs the output form and writes it back to client
 *                browser.
 *
 * ARGUMENTS:   EXTENSION_CONTROL_BLOCK *pECB  passed in structure
 *                pointer from inetsrv.
 *                int
 *                iTermId  client browser terminal id
 *                int
 *                iSyncId  client browser sync id
 *

```

```

 * RETURNS:      None
 *
 * COMMENTS:     None
 *
 */

#ifdef LOCAL_ALLOC // Allocate the tmalloc structure for each transaction
// This saves on some memory at the expense of some CPU cycles.
static void ProcessNewOrderForm(EXTENSION_CONTROL_BLOCK *pECB, int iTermId, i
iSyncId)
{
    int      TpRc, iRc, iError;
    long     ilen, *olen;
    char     buf[128];

    Buffer    NEW_ORDER_DATA          *NewOrderDataPtr; //New Order Tuxedc

    if((iRc = ThrTpInit()) <0)
    {
        // This is bad
        sprintf(buf, "ProcessNewOrder Failed ThrTpInit: iRc = %d", iRc);
        LogTuxError(0, buf);
        ErrorMessage(pECB, ERR_NEW_ORDER_NOT_PROCESSED,
ERR_TYPE_WEBDLL, NULL, iTermId, iSyncId);
        return;
    }

    memset(&Term.pClientData[iTermId].NewOrderData, 0,
sizeof(NEW_ORDER_DATA));

    Term.pClientData[iTermId].NewOrderData.w_id = Term.pClientData[iTermId].w_id;

    if ( (iError=GetNewOrderData(pECB->lpszQueryString,
&Term.pClientData[iTermId].NewOrderData) != ERR_SUCCESS )
    {
        ErrorMessage(pECB, iError, ERR_TYPE_WEBDLL, NULL, iTermId,
iSyncId);
        return;
    }

    if ((NewOrderDataPtr = (NEW_ORDER_DATA *)tpalloc("CARRAY", NULL,
sizeof(NEW_ORDER_DATA))) == NULL)
    {
        TpRc = tperrno;
        sprintf(buf, "ProcessNewOrder Tpcalloc Failed");
        LogTuxError(TpRc, buf);
        ErrorMessage(pECB, ERR_NEW_ORDER_NOT_PROCESSED,

```

```

ERR_TYPE_WEBDLL, NULL, iTermId, iSyncId);
    return;
}

*NewOrderDataPtr = Term.pClientData[iTermId].NewOrderData;

ilen = sizeof(NEW_ORDER_DATA);
olen = &ilen;

if ( dLog )
{
    FILE *fp;

    fp = fopen(szTpccLogPath, "ab");
    fprintf(fp, "** ProcessNewOrderL Thread %d iTermId %d NewOrderDataPtr:
%x size %d \r\n",
        GetCurrentThreadId(), iTermId, &NewOrderDataPtr,
        sizeof(*NewOrderDataPtr));
    fclose(fp);
}

if ((iRc = tpcall("NEWORDER", (char *)NewOrderDataPtr, ilen,
    (char **)&NewOrderDataPtr, (long*)olen, TPSIGRSTRT)) == -1)
{
    TpRc = tperrno;
    sprintf(buf, "Neworder tpcall failed");
    LogTuxError(TpRc, buf);
    ErrorMessage(pECB, ERR_NEW_ORDER_NOT_PROCESSED,
ERR_TYPE_WEBDLL, NULL, iTermId, iSyncId);
    return;
}

if ( dLog )
{
    FILE *fp;

    fp = fopen(szTpccLogPath, "ab");
    fprintf(fp, "*** ProcessNewOrderL Thread %d iTermId %d
NewOrderDataPtr: %x size %d\r\n",
        GetCurrentThreadId(), iTermId, &NewOrderDataPtr,
        sizeof(*NewOrderDataPtr));
    fclose(fp);
}

Term.pClientData[iTermId].NewOrderData = *NewOrderDataPtr;

iRc = NewOrderDataPtr->retval;
iError = NewOrderDataPtr->error;

```

```

tpfree((char *)NewOrderDataPtr);

if ( iRc < 0 )
{
    if (iError == ERR_TYPE_DEADLOCK)
        ErrorMessage(pECB,
ERR_NEW_ORDER_NOT_PROCESSED, ERR_TYPE_DEADLOCK, NULL, iTermId,
iSyncId);
    else
        ErrorMessage(pECB,
ERR_NEW_ORDER_NOT_PROCESSED, ERR_TYPE_WEBDLL, NULL, iTermId, iSyncId);
}
else
    if ( iError == ERR_BAD_ITEM_ID)
        WriteZString(pECB, MakeNewOrderForm(iTermId, iSyncId,
TRUE, FALSE, (BOOL)iRc) );
    else
        WriteZString(pECB, MakeNewOrderForm(iTermId, iSyncId,
FALSE, FALSE, (BOOL)iRc) );

return;
}
#else // Not LOCAL_ALLOC
static void ProcessNewOrderForm(EXTENSION_CONTROL_BLOCK *pECB, int iTermId,
iSyncId)
{
    int TpRc, iRc, iError;
    long ilen, *olen;
    char buf[128];

    if((iRc = ThrTplnit()) <0)
    {
        // This is bad
        sprintf(buf, "ProcessNewOrder Failed ThrTplnit: iRc = %d", iRc);
        LogTuxError(0, buf);
        ErrorMessage(pECB, ERR_NEW_ORDER_NOT_PROCESSED,
ERR_TYPE_WEBDLL, NULL, iTermId, iSyncId);
        return;
    }

    memset(&Term.pClientData[iTermId].NewOrderData, 0,
sizeof(NEW_ORDER_DATA));

    Term.pClientData[iTermId].NewOrderData.w_id = Term.pClientData[iTermId].w_ic

    if ( (iError=GetNewOrderData(pECB->lpszQueryString,
&Term.pClientData[iTermId].NewOrderData)) != ERR_SUCCESS )

```

```

    {
        ErrorMessage(pECB, iError, ERR_TYPE_WEBDLL, NULL, iTermId,
iSyncId);
        return;
    }

    Term.pClientData[iTermId].TuxDataPtr->NewOrderData =
Term.pClientData[iTermId].NewOrderData;

    ilen = sizeof(TUX_DATA);
    olen = &ilen;

    if ( dLog )
    {
        FILE *fp;

        fp = fopen(szTpccLogPath, "ab");
        fprintf(fp, "*** ProcessNewOrder Thread %d iTermId %d TuxDataPtr: %x
size %d \r\n",
                GetCurrentThreadId(), iTermId,
&Term.pClientData[iTermId].TuxDataPtr,
                sizeof(*Term.pClientData[iTermId].TuxDataPtr));
        fclose(fp);
    }

    if ((iRc = tpcall("NEWORDER", (char *)Term.pClientData[iTermId].TuxDataPtr, ilen,
(char **)&Term.pClientData[iTermId].TuxDataPtr, (long *)olen,
TPSIGRSTRT)) == -1)
    {
        TpRc = tperrno;
        sprintf(buf, "Neworder tpcall failed");
        LogTuxError(TpRc, buf);
        ErrorMessage(pECB, ERR_NEW_ORDER_NOT_PROCESSED,
ERR_TYPE_WEBDLL, NULL, iTermId, iSyncId);
        return;
    }

    if ( dLog )
    {
        FILE *fp;

        fp = fopen(szTpccLogPath, "ab");
        fprintf(fp, "*** ProcessNewOrder Thread %d iTermId %d TuxDataPtr: %x
size %d\r\n",
                GetCurrentThreadId(), iTermId,
&Term.pClientData[iTermId].TuxDataPtr,
                sizeof(*Term.pClientData[iTermId].TuxDataPtr));
        fclose(fp);
    }

```

```

    }

    Term.pClientData[iTermId].NewOrderData = Term.pClientData[iTermId].TuxDataPtr-
>NewOrderData;

    iRc = Term.pClientData[iTermId].TuxDataPtr->NewOrderData.retval;
    iError = Term.pClientData[iTermId].TuxDataPtr->NewOrderData.error;

    if ( iRc < 0 )
    {
        if (iError == ERR_TYPE_DEADLOCK)
            ErrorMessage(pECB,
ERR_NEW_ORDER_NOT_PROCESSED, ERR_TYPE_DEADLOCK, NULL, iTermId,
iSyncId);
        else
            ErrorMessage(pECB,
ERR_NEW_ORDER_NOT_PROCESSED, ERR_TYPE_WEBDLL, NULL, iTermId, iSyncId);
    }
    else
        if ( iError == ERR_BAD_ITEM_ID)
            WriteZString(pECB, MakeNewOrderForm(iTermId, iSyncId,
TRUE, FALSE, (BOOL)iRc) );
        else
            WriteZString(pECB, MakeNewOrderForm(iTermId, iSyncId,
FALSE, FALSE, (BOOL)iRc) );

    return;
}
#endif // LOCAL_ALLOC

/* FUNCTION: void ProcessPaymentForm(EXTENSION_CONTROL_BLOCK *pECB, int
iTermId, int iSyncId)
*
* PURPOSE:      This function gets and validates the input data from the payment form
                filling in the required input variables. It then calls the
SQLPayment
*
                transaction, constructs the output form and writes it back to cli
                browser.
*
* ARGUMENTS:   EXTENSION_CONTROL_BLOCK      *pECB      passed in structure
                pointer from inetsrv.
*
                int
                iTermId  client browser terminal id
                int
                iSyncId  client browser sync id
*
* RETURNS:     None
*

```



```

* COMMENTS:    None
*
*/

#ifdef LOCAL_ALLOC // Allocate the tmalloc structure for each transaction
//      This saves on some memory at the expense of some CPU cycles.
static void ProcessPaymentForm(EXTENSION_CONTROL_BLOCK *pECB, int iTermId, int
iSynclId)
{
    int          TpRc, iRc, iError;
    long         ilen, *olen;
    char         buf[128];

    PAYMENT_DATA      *PaymentDataPtr; //Payment Tuxedo Buffer

    if((iRc = ThrTPlnit()) <0)
    {
        // This is bad
        sprintf(buf, "ProcessPayment Failed ThrTPlnit: iRc = %d", iRc);
        LogTuxError(0, buf);
        ErrorMessage(pECB, ERR_PAYMENT_NOT_PROCESSED,
ERR_TYPE_WEBDLL, NULL, iTermId, iSynclId);
        return;
    }

    memset(&Term.pClientData[iTermId].PaymentData, 0, sizeof(PAYMENT_DATA));

    Term.pClientData[iTermId].PaymentData.w_id = Term.pClientData[iTermId].w_id;

    if ( (iError=GetPaymentData(pECB->lpszQueryString,
&Term.pClientData[iTermId].PaymentData) != ERR_SUCCESS )
    {
        ErrorMessage(pECB, iError, ERR_TYPE_WEBDLL, NULL, iTermId, iSynclId);
        return;
    }

    if ((PaymentDataPtr = (PAYMENT_DATA *)tpalloc("CARRAY", NULL,
sizeof(PAYMENT_DATA))) == NULL)
    {
        TpRc = tperrno;
        sprintf(buf, "ProcessPayment Tpcalloc Failed");
        LogTuxError(TpRc, buf);
        ErrorMessage(pECB, ERR_PAYMENT_NOT_PROCESSED,
ERR_TYPE_WEBDLL, NULL, iTermId, iSynclId);
        return;
    }

    *PaymentDataPtr = Term.pClientData[iTermId].PaymentData;

```

```

        ilen = sizeof(PAYMENT_DATA);
        olen = &ilen;

        if ( dLog )
        {
            FILE *fp;

            fp = fopen(szTpccLogPath, "ab");
            fprintf(fp, "** ProcessPayment Thread %d iTermId %d PaymentDataPtr: %
\n",
                GetCurrentThreadId(), iTermId, &PaymentDataPtr);
            fclose(fp);
        }

        if (( iRc = tpcall("PAYMENT", (char *)PaymentDataPtr, ilen,
(char **)&PaymentDataPtr, (long *)olen, TPSIGRSTRT)) == -1)
        {
            TpRc = tperrno;
            sprintf(buf, "ProcessPayment tpcall failed");
            LogTuxError(TpRc, buf);
            ErrorMessage(pECB, ERR_PAYMENT_NOT_PROCESSED,
ERR_TYPE_WEBDLL, NULL, iTermId, iSynclId);
            return;
        }

        if ( dLog )
        {
            FILE *fp;

            fp = fopen(szTpccLogPath, "ab");
            fprintf(fp, "*** ProcessPayment Thread %d iTermId %d PaymentDataPtr:
\n",
                GetCurrentThreadId(), iTermId, &PaymentDataPtr);
            fclose(fp);
        }

        Term.pClientData[iTermId].PaymentData = *PaymentDataPtr;

        iRc = PaymentDataPtr->retval;
        iError = PaymentDataPtr->error;

        tpfree((char *)PaymentDataPtr);

        if ( iRc < 0 )
        {
            if (iError == ERR_TYPE_DEADLOCK )

```

```

        ErrorMessage(pECB, ERR_PAYMENT_NOT_PROCESSED,
ERR_TYPE_DEADLOCK, NULL, iTermId, iSynclD);
        else if (iError == ERR_NOSUCH_CUSTOMER)
            ErrorMessage(pECB, ERR_PAYMENT_INVALID_CUSTOMER,
ERR_TYPE_WEBDLL, NULL, iTermId, iSynclD);
        else
            ErrorMessage(pECB, ERR_PAYMENT_NOT_PROCESSED,
ERR_TYPE_WEBDLL, NULL, iTermId, iSynclD);
    }
    else
        WriteZString(pECB, MakePaymentForm(iTermId, iSynclD, FALSE) );

    return;
}
#else // Not LOCAL_ALLOC
static void ProcessPaymentForm(EXTENSION_CONTROL_BLOCK *pECB, int iTermId, int
iSynclD)
{
    int            TpRc, iRc, iError;
    long           ilen, *olen;
    char           buf[128];

    if((iRc = ThrTplnit()) <0)
    {
        // This is bad
        sprintf(buf, "ProcessPayment Failed ThrTplnit: iRc = %d", iRc);
        LogTuxError(0, buf);
        ErrorMessage(pECB, ERR_PAYMENT_NOT_PROCESSED,
ERR_TYPE_WEBDLL, NULL, iTermId, iSynclD);
        return;
    }

    memset(&Term.pClientData[iTermId].PaymentData, 0, sizeof(PAYMENT_DATA));

    Term.pClientData[iTermId].PaymentData.w_id = Term.pClientData[iTermId].w_id;

    if ( (iError=GetPaymentData(pECB->lpszQueryString,
&Term.pClientData[iTermId].PaymentData)) != ERR_SUCCESS )
    {
        ErrorMessage(pECB, iError, ERR_TYPE_WEBDLL, NULL, iTermId, iSynclD);
        return;
    }

    Term.pClientData[iTermId].TuxDataPtr->PaymentData =
Term.pClientData[iTermId].PaymentData;

    ilen = sizeof(TUX_DATA);

```

```

    olen = &ilen;

    if ( dLog )
    {
        FILE *fp;

        fp = fopen(szTpccLogPath, "ab");
        fprintf(fp, "** ProcessPayment Thread %d iTermId %d PaymentDataPtr:
\n",
                GetCurrentThreadId(), iTermId,
&Term.pClientData[iTermId].TuxDataPtr);
        fclose(fp);
    }

    if (( iRc = tpcall("PAYMENT", (char *)Term.pClientData[iTermId].TuxDataPtr, ilen,
(char **)&Term.pClientData[iTermId].TuxDataPtr, (long *)olen,
TPSIGRSTRT)) == -1)
    {
        TpRc = tperrno;
        sprintf(buf, "ProcessPayment tpcall failed");
        LogTuxError(TpRc, buf);
        ErrorMessage(pECB, ERR_PAYMENT_NOT_PROCESSED,
ERR_TYPE_WEBDLL, NULL, iTermId, iSynclD);
        return;
    }

    if ( dLog )
    {
        FILE *fp;

        fp = fopen(szTpccLogPath, "ab");
        fprintf(fp, "*** ProcessPayment Thread %d iTermId %d TuxDataPtr: %x
\n",
                GetCurrentThreadId(), iTermId,
&Term.pClientData[iTermId].TuxDataPtr);
        fclose(fp);
    }

    Term.pClientData[iTermId].PaymentData = Term.pClientData[iTermId].TuxDataPtr-
>PaymentData;

    iRc = Term.pClientData[iTermId].TuxDataPtr->PaymentData.retval;
    iError = Term.pClientData[iTermId].TuxDataPtr->PaymentData.error;

    if ( iRc < 0 )
    {
        if (iError == ERR_TYPE_DEADLOCK )
            ErrorMessage(pECB, ERR_PAYMENT_NOT_PROCESSED,

```

```

ERR_TYPE_DEADLOCK, NULL, iTermId, iSyncId);
    else if (iError == ERR_NOSUCH_CUSTOMER)
        ErrorMessage(pECB, ERR_PAYMENT_INVALID_CUSTOMER,
ERR_TYPE_WEBDLL, NULL, iTermId, iSyncId);
    else
        ErrorMessage(pECB, ERR_PAYMENT_NOT_PROCESSED,
ERR_TYPE_WEBDLL, NULL, iTermId, iSyncId);
}
else
    WriteZString(pECB, MakePaymentForm(iTermId, iSyncId, FALSE) );

return;
}
#endif // LOCAL_ALLOC

/* FUNCTION: void ProcessOrderStatusForm(EXTENSION_CONTROL_BLOCK *pECB, int
iTermId, int iSyncId)
*
* PURPOSE:      This function gets and validates the input data from the Order Status
*               form filling in the required input variables. It then calls the
*               SQLOrderStatus transaction, constructs the output form and
writes it
*               back to client browser.
*
* ARGUMENTS:   EXTENSION_CONTROL_BLOCK *pECB passed in structure
pointer from inetsrv.
*               int
iTermId client browser terminal id
*               int
iSyncId client browser sync id
*
* RETURNS:     None
*
* COMMENTS:   None
*/

#ifdef LOCAL_ALLOC // Allocate the tmalloc structure for each transaction
// This saves on some memory at the expense of some CPU cycles.
static void ProcessOrderStatusForm(EXTENSION_CONTROL_BLOCK *pECB, int iTermId, int
iSyncId)
{
    int TpRc, iRc, iError;
    long ilen, *olen;
    char buf[128];

    ORDER_STATUS_DATA *OrderStatusDataPtr; //Order Status Tuxedo
Buffer

```

```

if((iRc = ThrTPlnit()) <0)
{
    // This is bad
    sprintf(buf, "ProcessOrderStatus Failed ThrTPlnit: iRc = %d", iRc);
    LogTuxError(0, buf);
    ErrorMessage(pECB, ERR_ORDER_STATUS_NOT_PROCESSED,
ERR_TYPE_WEBDLL, NULL, iTermId, iSyncId);
    return;
}

memset(&Term.pClientData[iTermId].OrderStatusData, 0,
sizeof(ORDER_STATUS_DATA));

Term.pClientData[iTermId].OrderStatusData.w_id = Term.pClientData[iTermId].w_id;

if ( (iError=GetOrderStatusData(pECB->lpszQueryString,
&Term.pClientData[iTermId].OrderStatusData) != ERR_SUCCESS )
{
    ErrorMessage(pECB, iError, ERR_TYPE_WEBDLL, NULL, iTermId, iSyncId);
    return;
}

if ((OrderStatusDataPtr = (ORDER_STATUS_DATA *)tpalloc("CARRAY", NULL,
sizeof(ORDER_STATUS_DATA))) == NULL)
{
    TpRc = tperrno;
    sprintf(buf, "ProcessOrderStatus Tpcalloc Failed");
    LogTuxError(TpRc, buf);
    ErrorMessage(pECB, ERR_ORDER_STATUS_NOT_PROCESSED,
ERR_TYPE_WEBDLL, NULL, iTermId, iSyncId);
    return;
}

*OrderStatusDataPtr = Term.pClientData[iTermId].OrderStatusData;

ilen = sizeof(ORDER_STATUS_DATA);
olen = &ilen;

if ( dLog )
{
    FILE *fp;

    fp = fopen(szTpccLogPath, "ab");
    fprintf(fp, "** ProcessOrderStatus Thread %d iTermId %d
OrderStatusDataPtr: %x \r\n",
GetCurrentThreadId(), iTermId, &OrderStatusDataPtr);
fclose(fp);
}

```

```

    }

    if (( iRc = tpcall("ORDERSTATUS", (char *)OrderStatusDataPtr, ilen,
                    (char **)&OrderStatusDataPtr, (long *)olen, TPSIGRSTRT)) == -1)
    {
        TpRc = tperrno;
        sprintf(buf, "ProcessOrderStatus tpcall failed");
        LogTuxError(TpRc, buf);
        ErrorMessage(pECB, ERR_ORDER_STATUS_NOT_PROCESSED,
ERR_TYPE_WEBDLL, NULL, iTermId, iSyncId);
        return;
    }

    if ( dLog )
    {
        FILE *fp;

        fp = fopen(szTpccLogPath, "ab");
        fprintf(fp, "*** ProcessOrderStatus Thread %d iTermId %d
OrderStatusDataPtr: %x \n\n",
                GetCurrentThreadId(), iTermId, &OrderStatusDataPtr);
        fclose(fp);
    }

    Term.pClientData[iTermId].OrderStatusData = *OrderStatusDataPtr;

    iRc = OrderStatusDataPtr->retval;
    iError = OrderStatusDataPtr->error;

    tpfree((char *)OrderStatusDataPtr);

    if ( iRc < 0 )
    {
        if ( iError == ERR_TYPE_DEADLOCK )
            ErrorMessage(pECB,
ERR_ORDER_STATUS_NOT_PROCESSED, ERR_TYPE_DEADLOCK, NULL, iTermId,
iSyncId);
        else if (iError == ERR_NOSUCH_CUSTOMER)
            ErrorMessage(pECB, ERR_NOSUCH_CUSTOMER,
ERR_TYPE_WEBDLL, NULL, iTermId, iSyncId);
        else
            ErrorMessage(pECB,
ERR_ORDER_STATUS_NOT_PROCESSED, ERR_TYPE_WEBDLL, NULL, iTermId,
iSyncId);
    }
    else
        WriteZString(pECB, MakeOrderStatusForm(iTermId, iSyncId, FALSE) );

```

```

        return;
    }
    #else // Not LOCAL_ALLOC
    static void ProcessOrderStatusForm(EXTENSION_CONTROL_BLOCK *pECB, int iTermId,
iSyncId)
    {
        int            TpRc, iRc, iError;
        long           ilen, *olen;
        char           buf[128];

        if((iRc = ThrTpInIt()) <0)
        {
            // This is bad
            sprintf(buf, "ProcessOrderStatus Failed ThrTpInIt: iRc = %d", iRc);
            LogTuxError(0, buf);
            ErrorMessage(pECB, ERR_ORDER_STATUS_NOT_PROCESSED,
ERR_TYPE_WEBDLL, NULL, iTermId, iSyncId);
            return;
        }

        memset(&Term.pClientData[iTermId].OrderStatusData, 0,
sizeof(ORDER_STATUS_DATA));

        Term.pClientData[iTermId].OrderStatusData.w_id = Term.pClientData[iTermId].w_id;

        if ( (iError=GetOrderStatusData(pECB->lpszQueryString,
&Term.pClientData[iTermId].OrderStatusData) != ERR_SUCCESS )
        {
            ErrorMessage(pECB, iError, ERR_TYPE_WEBDLL, NULL, iTermId, iSyncId);
            return;
        }

        Term.pClientData[iTermId].TuxDataPtr->OrderStatusData =
Term.pClientData[iTermId].OrderStatusData;

        ilen = sizeof(TUX_DATA);
        olen = &ilen;

        if ( dLog )
        {
            FILE *fp;

            fp = fopen(szTpccLogPath, "ab");
            fprintf(fp, "** ProcessOrderStatus Thread %d iTermId %d TuxDataPtr: %
\n\n",
                    GetCurrentThreadId(), iTermId,
&Term.pClientData[iTermId].TuxDataPtr);
            fclose(fp);
        }
    }

```

```

    }

    if (( iRc = tpcall("ORDERSTATUS", (char *)Term.pClientData[iTermId].TuxDataPtr, ilen,
                    (char **)&Term.pClientData[iTermId].TuxDataPtr, (long *)olen,
TPSIGRSTRT)) == -1)
    {
        TpRc = tperrno;
        sprintf(buf, "ProcessOrderStatus tpcall failed");
        LogTuxError(TpRc, buf);
        ErrorMessage(pECB, ERR_ORDER_STATUS_NOT_PROCESSED,
ERR_TYPE_WEBDLL, NULL, iTermId, iSyncId);
        return;
    }

    if ( dLog )
    {
        FILE *fp;

        fp = fopen(szTpccLogPath, "ab");
        fprintf(fp, "*** ProcessOrderStatus Thread %d iTermId %d TuxDataPtr: %x
\n",
                GetCurrentThreadId(), iTermId,
&Term.pClientData[iTermId].TuxDataPtr);
        fclose(fp);
    }

    Term.pClientData[iTermId].OrderStatusData = Term.pClientData[iTermId].TuxDataPtr-
>OrderStatusData;

    iRc = Term.pClientData[iTermId].TuxDataPtr->OrderStatusData.retval;
    iError = Term.pClientData[iTermId].TuxDataPtr->OrderStatusData.error;

    if ( iRc < 0 )
    {
        if ( iError == ERR_TYPE_DEADLOCK )
            ErrorMessage(pECB,
ERR_ORDER_STATUS_NOT_PROCESSED, ERR_TYPE_DEADLOCK, NULL, iTermId,
iSyncId);
        else if (iError == ERR_NOSUCH_CUSTOMER)
            ErrorMessage(pECB, ERR_NOSUCH_CUSTOMER,
ERR_TYPE_WEBDLL, NULL, iTermId, iSyncId);
        else
            ErrorMessage(pECB,
ERR_ORDER_STATUS_NOT_PROCESSED, ERR_TYPE_WEBDLL, NULL, iTermId,
iSyncId);
    }
    else
        WriteZString(pECB, MakeOrderStatusForm(iTermId, iSyncId, FALSE));

```

```

        return;
    }
#endif // LOCAL_ALLOC

/* FUNCTION: void ProcessDeliveryForm(EXTENSION_CONTROL_BLOCK *pECB, int
iTermId, int iSyncId)
*
* PURPOSE:      This function gets and validates the input data from the delivery form
*                filling in the required input variables. It then calls the
PostDeliveryInfo
*                Api, The client is then informed that the transaction has been
posted.
*
* ARGUMENTS:   EXTENSION_CONTROL_BLOCK      *pECB   passed in structure
pointer from inetsrv.
*                int
iTermId   client browser terminal id
*                int
iSyncId   clinet browser sync id
*
* RETURNS:     None
*
* COMMENTS:    None
*/

#ifdef LOCAL_ALLOC // Allocate the talloc structure for each transaction
// This saves on some memory at the expense of some CPU cycles.
static void ProcessDeliveryForm(EXTENSION_CONTROL_BLOCK *pECB, int iTermId, int
iSyncId)
{
    int                TpRc, iRc;
    char               szTmp[26];
    BOOL               bSuccess;
    long               ilen, *olen;
    char               buf[128];

    DELIVERY_DATA      *DeliveryDataPtr; //Delivery Tuxedo Buffer

    if((iRc = ThrTplnit()) <0)
    {
        // This is bad
        sprintf(buf, "ProcessDelivery Failed ThrTplnit: iRc = %d", iRc);
        LogTuxError(0, buf);
        ErrorMessage(pECB, ERR_DELIVERY_MISSING_OCD_KEY,
ERR_TYPE_WEBDLL, NULL, iTermId, iSyncId);
    }
}

```

```

        return;
    }

    memset(&Term.pClientData[iTermId].DeliveryData, 0, sizeof(DELIVERY_DATA));

    if ( !GetKeyValue(pECB->lpszQueryString, "OCD*", szTmp, sizeof(szTmp)) )
    {
        ErrorMessage(pECB, ERR_DELIVERY_MISSING_OCD_KEY,
ERR_TYPE_WEBDLL, NULL, iTermId, iSyncId);
        return;
    }

    if ( !IsNumeric(szTmp) )
    {
        ErrorMessage(pECB, ERR_DELIVERY_CARRIER_INVALID,
ERR_TYPE_WEBDLL, NULL, iTermId, iSyncId);
        return;
    }

    Term.pClientData[iTermId].DeliveryData.o_carrier_id = atoi(szTmp);

    if ( Term.pClientData[iTermId].DeliveryData.o_carrier_id > 10 ||
Term.pClientData[iTermId].DeliveryData.o_carrier_id < 1 )
    {
        ErrorMessage(pECB, ERR_DELIVERY_CARRIER_ID_RANGE,
ERR_TYPE_WEBDLL, NULL, iTermId, iSyncId);
        return;
    }

    Term.pClientData[iTermId].DeliveryData.w_id = Term.pClientData[iTermId].w_id;

    GetLocalTime(&Term.pClientData[iTermId].DeliveryData.queue_time);

    if ((DeliveryDataPtr = (DELIVERY_DATA *)tpalloc("CARRAY", NULL,
sizeof(DELIVERY_DATA))) == NULL)
    {
        TpRc = tperrno;
        sprintf(buf, "ProcessDelivery Tpcalloc Failed");
        LogTuxError(TpRc, buf);
        strcpy(Term.pClientData[iTermId].DeliveryData.execution_status, "Delivery
Post Failed");

        bSuccess = FALSE;
        WriteZString(pECB, MakeDeliveryForm(iTermId, iSyncId, FALSE,
bSuccess) );
        return;
    }

    *DeliveryDataPtr = Term.pClientData[iTermId].DeliveryData;

```

```

    ilen = sizeof(DELIVERY_DATA);
    olen = &ilen;

    if ( dLog )
    {
        FILE *fp;

        fp = fopen(szTpccLogPath, "ab");
        fprintf(fp, "** ProcessDelivery Thread %d iTermId %d DeliveryDataPtr: %d
\n",
                GetCurrentThreadId(), iTermId, &DeliveryDataPtr);
        fclose(fp);
    }

    if (( iRc = tpacall("DELIVERY", (char *)DeliveryDataPtr, ilen, TPNOREPLY)) == -1)
    {
        TpRc = tperrno;
        sprintf(buf, "ProcessDelivery Tpcalloc Failed");
        LogTuxError(TpRc, buf);
        strcpy(Term.pClientData[iTermId].DeliveryData.execution_status, "Delivery
Post Failed");

        bSuccess = FALSE;
        WriteZString(pECB, MakeDeliveryForm(iTermId, iSyncId, FALSE,
bSuccess) );
        return;
    }

    if ( dLog )
    {
        FILE *fp;

        fp = fopen(szTpccLogPath, "ab");
        fprintf(fp, "*** ProcessDelivery Thread %d iTermId %d DeliveryDataPtr: %d
TpRc = %d \n",
                GetCurrentThreadId(), iTermId, &DeliveryDataPtr, TpRc);
        fclose(fp);
    }

    tpfree((char *)DeliveryDataPtr);

    strcpy(Term.pClientData[iTermId].DeliveryData.execution_status, "Delivery has be
queued.");
    bSuccess = TRUE;

    WriteZString(pECB, MakeDeliveryForm(iTermId, iSyncId, FALSE, bSuccess) );

    return;

```

```

}
#else // Not LOCAL_ALLOC
static void ProcessDeliveryForm(EXTENSION_CONTROL_BLOCK *pECB, int iTermId, int
iSynclD)
{
    int                TpRc, iRc;
    char              szTmp[26];
    BOOL              bSuccess;
    long              ilen, *olen;
    char              buf[128];

    if((iRc = ThrTpInit()) <0)
    {
        // This is bad
        sprintf(buf, "ProcessDelivery Failed ThrTpInit: iRc = %d", iRc);
        LogTuxError(0, buf);
        ErrorMessage(pECB, ERR_DELIVERY_MISSING_OCD_KEY,
ERR_TYPE_WEBDLL, NULL, iTermId, iSynclD);
        return;
    }

    memset(&Term.pClientData[iTermId].DeliveryData, 0, sizeof(DELIVERY_DATA));

    if ( !GetKeyValue(pECB->lpszQueryString, "OCD*", szTmp, sizeof(szTmp)) )
    {
        ErrorMessage(pECB, ERR_DELIVERY_MISSING_OCD_KEY,
ERR_TYPE_WEBDLL, NULL, iTermId, iSynclD);
        return;
    }

    if ( !IsNumeric(szTmp) )
    {
        ErrorMessage(pECB, ERR_DELIVERY_CARRIER_INVALID,
ERR_TYPE_WEBDLL, NULL, iTermId, iSynclD);
        return;
    }

    Term.pClientData[iTermId].DeliveryData.o_carrier_id = atoi(szTmp);

    if ( Term.pClientData[iTermId].DeliveryData.o_carrier_id > 10 ||
Term.pClientData[iTermId].DeliveryData.o_carrier_id < 1 )
    {
        ErrorMessage(pECB, ERR_DELIVERY_CARRIER_ID_RANGE,
ERR_TYPE_WEBDLL, NULL, iTermId, iSynclD);
        return;
    }
}

```

```

Term.pClientData[iTermId].DeliveryData.w_id = Term.pClientData[iTermId].w_id;

    GetLocalTime(&Term.pClientData[iTermId].DeliveryData.queue_time);

    Term.pClientData[iTermId].TuxDataPtr->DeliveryData =
Term.pClientData[iTermId].DeliveryData;

    ilen = sizeof(TUX_DATA);
    olen = &ilen;

    if ( dLog )
    {
        FILE *fp;

        fp = fopen(szTpccLogPath, "ab");
        fprintf(fp, "*** ProcessDelivery Thread %d iTermId %d TuxDataPtr: %x \n",
            GetCurrentThreadId(), iTermId,
&Term.pClientData[iTermId].TuxDataPtr);
        fclose(fp);
    }

    if (( iRc = tpacall("DELIVERY", (char *)Term.pClientData[iTermId].TuxDataPtr, ilen,
TPNOREPLY)) == -1)
    {
        TpRc = tperrno;
        sprintf(buf, "ProcessDelivery Tpcalloc Failed");
        LogTuxError(TpRc, buf);
        strcpy(Term.pClientData[iTermId].DeliveryData.execution_status, "Deliv
Post Failed");

        bSuccess = FALSE;
        WriteZString(pECB, MakeDeliveryForm(iTermId, iSynclD, FALSE,
bSuccess) );
        return;
    }

    if ( dLog )
    {
        FILE *fp;

        fp = fopen(szTpccLogPath, "ab");
        fprintf(fp, "*** ProcessDelivery Thread %d iTermId %d TuxDataPtr: %x
TpRc = %d \n",
            GetCurrentThreadId(), iTermId,
&Term.pClientData[iTermId].TuxDataPtr, TpRc);
        fclose(fp);
    }
}

```

```

        strcpy(Term.pClientData[iTermId].DeliveryData.execution_status, "Delivery has been
queued.");
        bSuccess = TRUE;

        WriteZString(pECB, MakeDeliveryForm(iTermId, iSynclD, FALSE, bSuccess) );
    return;
}
#endif // LOCAL_ALLOC

/* FUNCTION: void ProcessStockLevelForm(EXTENSION_CONTROL_BLOCK *pECB, int
iTermId, int iSynclD)
*
* PURPOSE:      This function gets and validates the input data from the Stock Level
*               form filling in the required input variables. It then calls the
*               SQLStockLevel transaction, constructs the output form and
writes it
*               back to client browser.
*
* ARGUMENTS:   EXTENSION_CONTROL_BLOCK *pECB passed in structure
pointer from inetsrv.
*               int
*               iTermId client browser terminal id
*               int
*               iSynclD client browser sync id
*
* RETURNS:     None
*
* COMMENTS:    None
*/

#ifdef LOCAL_ALLOC // Allocate the tpalloc structure for each transaction
// This saves on some memory at the expense of some CPU cycles.
static void ProcessStockLevelForm(EXTENSION_CONTROL_BLOCK *pECB, int iTermId, int
iSynclD)
{
    int          TpRc, iRc, iError;
    char         szTmp[26];
    long         ilen, *olen;
    char         buf[128];

    STOCK_LEVEL_DATA *StockLevelDataPtr; //Stock Level Tuxedo Buffer

    if((iRc = ThrTpInit()) <0)
    {
        // This is bad
        sprintf(buf, "ProcessStockLevel Failed ThrTpInit: iRc = %d", iRc);

```

```

        LogTuxError(0, buf);
        ErrorMessage(pECB, ERR_STOCKLEVEL_NOT_PROCESSED,
ERR_TYPE_WEBDLL, NULL, iTermId, iSynclD);
        return;
    }

    memset(&Term.pClientData[iTermId].StockLevelData, 0, sizeof(STOCK_LEVEL_DATA))

    Term.pClientData[iTermId].StockLevelData.w_id = Term.pClientData[iTermId].w_id;
    Term.pClientData[iTermId].StockLevelData.d_id = Term.pClientData[iTermId].d_id;

    if ( !GetKeyValue(pECB->lpszQueryString, "TT*", szTmp, sizeof(szTmp)) )
    {
        ErrorMessage(pECB, ERR_STOCKLEVEL_MISSING_THRESHOLD_KEY,
ERR_TYPE_WEBDLL, NULL, iTermId, iSynclD);
        return;
    }

    if ( !IsNumeric(szTmp) )
    {
        ErrorMessage(pECB, ERR_STOCKLEVEL_THRESHOLD_INVALID,
ERR_TYPE_WEBDLL, NULL, iTermId, iSynclD);
        return;
    }

    Term.pClientData[iTermId].StockLevelData.thresh_hold = atoi(szTmp);

    if ( Term.pClientData[iTermId].StockLevelData.thresh_hold >= 100 ||
Term.pClientData[iTermId].StockLevelData.thresh_hold < 0 )
    {
        ErrorMessage(pECB, ERR_STOCKLEVEL_THRESHOLD_RANGE,
ERR_TYPE_WEBDLL, NULL, iTermId, iSynclD);
        return;
    }

    if ((StockLevelDataPtr = (STOCK_LEVEL_DATA *)tpalloc("CARRAY", NULL,
sizeof(STOCK_LEVEL_DATA))) == NULL)
    {
        TpRc = tperrno;
        sprintf(buf, "ProcessStockLevel Tpcalloc Failed");
        LogTuxError(TpRc, buf);
        ErrorMessage(pECB, ERR_STOCKLEVEL_NOT_PROCESSED,
ERR_TYPE_WEBDLL, NULL, iTermId, iSynclD);
        return;
    }

    *StockLevelDataPtr = Term.pClientData[iTermId].StockLevelData;

```



```

ilen = sizeof(STOCK_LEVEL_DATA);
olen = &ilen;

    if ( dLog )
    {
        FILE *fp;

        fp = fopen(szTpccLogPath, "ab");
        fprintf(fp, "** ProcessStockLevel Thread %d iTermId %d StockLevelDataPtr:
%x \r\n",
                GetCurrentThreadId(), iTermId, &StockLevelDataPtr);
        fclose(fp);
    }

    if (( iRc = tpcall("STOCKLEVEL", (char *)StockLevelDataPtr, ilen,
                    (char **)&StockLevelDataPtr, (long *) olen, TPSIGRSTRT)) == -1)
    {
        TpRc = tperrno;
        sprintf(buf, "ProcessStockLevel tpcall failed");
        LogTuxError(TpRc, buf);
        ErrorMessage(pECB, ERR_STOCKLEVEL_NOT_PROCESSED,
ERR_TYPE_WEBDLL, NULL, iTermId, iSyncId);
        return;
    }

    if ( dLog )
    {
        FILE *fp;

        fp = fopen(szTpccLogPath, "ab");
        fprintf(fp, "*** ProcessStockLevel Thread %d iTermId %d
StockLevelDataPtr: %x \r\n",
                GetCurrentThreadId(), iTermId, &StockLevelDataPtr);
        fclose(fp);
    }

    Term.pClientData[iTermId].StockLevelData = *StockLevelDataPtr;

    iRc = StockLevelDataPtr->retval;
    iError = StockLevelDataPtr->error;

    tpfree((char *)StockLevelDataPtr);

    if ( iRc == 0 )
    {
        if ( iError == ERR_TYPE_DEADLOCK )

```

```

        ErrorMessage(pECB,
ERR_STOCKLEVEL_NOT_PROCESSED, ERR_TYPE_DEADLOCK, NULL, iTermId,
iSyncId);
    }
    else
        ErrorMessage(pECB,
ERR_STOCKLEVEL_NOT_PROCESSED, ERR_TYPE_WEBDLL, NULL, iTermId, iSyncId);
    }
    else
        WriteZString(pECB, MakeStockLevelForm(iTermId, iSyncId, FALSE) );

    return;
}
#else // Not LOCAL_ALLOC
static void ProcessStockLevelForm(EXTENSION_CONTROL_BLOCK *pECB, int iTermId, ir
iSyncId)
{
    int            TpRc, iRc, iError;
    char          szTmp[26];
    long          ilen, *olen;
    char          buf[128];

    if((iRc = ThrTpInit()) <0)
    {
        // This is bad
        sprintf(buf, "ProcessStockLevel Failed ThrTpInit: iRc = %d", iRc);
        LogTuxError(0, buf);
        ErrorMessage(pECB, ERR_STOCKLEVEL_NOT_PROCESSED,
ERR_TYPE_WEBDLL, NULL, iTermId, iSyncId);
        return;
    }

    memset(&Term.pClientData[iTermId].StockLevelData, 0, sizeof(STOCK_LEVEL_DATA))

    Term.pClientData[iTermId].StockLevelData.w_id = Term.pClientData[iTermId].w_id;
    Term.pClientData[iTermId].StockLevelData.d_id = Term.pClientData[iTermId].d_id;

    if ( !GetKeyValue(pECB->lpszQueryString, "TT*", szTmp, sizeof(szTmp)) )
    {
        ErrorMessage(pECB, ERR_STOCKLEVEL_MISSING_THRESHOLD_KEY,
ERR_TYPE_WEBDLL, NULL, iTermId, iSyncId);
        return;
    }

    if ( !IsNumeric(szTmp) )
    {
        ErrorMessage(pECB, ERR_STOCKLEVEL_THRESHOLD_INVALID,
ERR_TYPE_WEBDLL, NULL, iTermId, iSyncId);

```

```

    return;
}

Term.pClientData[iTermId].StockLevelData.thresh_hold = atoi(szTmp);

if ( Term.pClientData[iTermId].StockLevelData.thresh_hold >= 100 ||
Term.pClientData[iTermId].StockLevelData.thresh_hold < 0 )
{
    ErrorMessage(pECB, ERR_STOCKLEVEL_THRESHOLD_RANGE,
ERR_TYPE_WEBDLL, NULL, iTermId, iSyncId);
    return;
}

Term.pClientData[iTermId].TuxDataPtr->StockLevelData =
Term.pClientData[iTermId].StockLevelData;

ilen = sizeof(TUX_DATA);
olen = &ilen;

    if ( dLog )
    {
        FILE *fp;

        fp = fopen(szTpccLogPath, "ab");
        fprintf(fp, "*** ProcessStockLevel Thread %d iTermId %d TuxDataPtr: %x
\n",
                GetCurrentThreadId(), iTermId,
&Term.pClientData[iTermId].TuxDataPtr);
        fclose(fp);
    }

    if (( iRc = tpcall("STOCKLEVEL", (char *)Term.pClientData[iTermId].TuxDataPtr, ilen,
(char **)&Term.pClientData[iTermId].TuxDataPtr, (long *) olen,
TPSIGRSTRT)) == -1)
    {
        TpRc = tperrno;
        sprintf(buf, "ProcessStockLevel tpcall failed");
        LogTuxError(TpRc, buf);
        ErrorMessage(pECB, ERR_STOCKLEVEL_NOT_PROCESSED,
ERR_TYPE_WEBDLL, NULL, iTermId, iSyncId);
        return;
    }

    if ( dLog )
    {
        FILE *fp;

        fp = fopen(szTpccLogPath, "ab");

```

```

        fprintf(fp, "*** ProcessStockLevel Thread %d iTermId %d TuxDataPtr: %:
\n",
                GetCurrentThreadId(), iTermId,
&Term.pClientData[iTermId].TuxDataPtr);
        fclose(fp);
    }

    Term.pClientData[iTermId].StockLevelData = Term.pClientData[iTermId].TuxDataPtr-
>StockLevelData;

    iRc = Term.pClientData[iTermId].TuxDataPtr->StockLevelData.retval;
    iError = Term.pClientData[iTermId].TuxDataPtr->StockLevelData.error;

    if ( iRc == 0 )
    {
        if ( iError == ERR_TYPE_DEADLOCK )
            ErrorMessage(pECB,
ERR_STOCKLEVEL_NOT_PROCESSED, ERR_TYPE_DEADLOCK, NULL, iTermId,
iSyncId);
        else
            ErrorMessage(pECB,
ERR_STOCKLEVEL_NOT_PROCESSED, ERR_TYPE_WEBDLL, NULL, iTermId, iSyncId);
    }
    else
        WriteZString(pECB, MakeStockLevelForm(iTermId, iSyncId, FALSE) );

    return;
}
#endif // LOCAL_ALLOC

/* FUNCTION: int GetNewOrderData(LPSTR lpszQueryString, NEW_ORDER_DATA
*pNewOrderData)
*
* PURPOSE:      This function extracts and validates the new order form data from an htt
command string.
*
* ARGUMENTS:   LPSTR                lpszQueryString                client
browser http command string
*
*               NEW_ORDER_DATA    *pNewOrderData
*               pointer to new order data structure
*
* RETURNS:     int
*               error code indicating reason for failure
*               ERR_SUCCESS
*               new order input data successfully parsed
*
*
*

```

```

* COMMENTS:   None
*
*/

static int GetNewOrderData(LPSTR lpszQueryString, NEW_ORDER_DATA *pNewOrderData)
{
    char    szTmp[26];
    char    szKey[26];
    int     i;
    short   items;
    BOOL    bCheck;

    if ( !GetKeyValue(lpszQueryString, "DD*", szTmp, sizeof(szTmp)) )
        return ERR_NEWORDER_FORM_MISSING_DID;

    if ( !IsNumeric(szTmp) )
        return ERR_NEWORDER_DISTRICT_INVALID;

    pNewOrderData->d_id = atoi(szTmp);

    if ( !GetKeyValue(lpszQueryString, "CID*", szTmp, sizeof(szTmp)) )
        return ERR_NEWORDER_CUSTOMER_KEY;

    if ( !IsNumeric(szTmp) )
        return ERR_NEWORDER_CUSTOMER_INVALID;
    pNewOrderData->c_id = atoi(szTmp);

    bCheck = FALSE;
    for(i=0, items=0; i<15; i++)
    {
        wsprintf(szKey, "IID%2.2d*", i);
        if ( !GetKeyValue(lpszQueryString, szKey, szTmp, sizeof(szTmp)) )
            return ERR_NEWORDER_MISSING_IID_KEY;
        if ( szTmp[0] )
        {
            //if blank lines between item ids
            if ( bCheck )
                return ERR_NEWORDER_ITEM_BLANK_LINES;
            if ( !IsNumeric(szTmp) )
                return ERR_NEWORDER_ITEMID_INVALID;
            pNewOrderData->Ol[i].ol_i_id = atoi(szTmp);

            wsprintf(szKey, "SP%2.2d*", i);
            if ( !GetKeyValue(lpszQueryString, szKey, szTmp,
sizeof(szTmp)) )
                return ERR_NEWORDER_MISSING_SUPPW_KEY;
            if ( !IsNumeric(szTmp) )
                return ERR_NEWORDER_SUPPW_INVALID;

```

```

pNewOrderData->Ol[i].ol_supply_w_id = (short)atoi(szTmp);

wsprintf(szKey, "Qty%2.2d*", i);
if ( !GetKeyValue(lpszQueryString, szKey, szTmp,
sizeof(szTmp)) )
    return ERR_NEWORDER_MISSING_QTY_KEY;

if ( !IsNumeric(szTmp) )
    return ERR_NEWORDER_QTY_INVALID;

pNewOrderData->Ol[i].ol_quantity = atoi(szTmp);
items++;

if ( pNewOrderData->Ol[i].ol_i_id >= 1000000 || pNewOrderD
>Ol[i].ol_i_id < 1 )
    return ERR_NEWORDER_ITEMID_RANGE;
if ( pNewOrderData->Ol[i].ol_quantity >= 100 || pNewOrderDat
>Ol[i].ol_quantity < 1 )
    return ERR_NEWORDER_QTY_RANGE;
        }
    else
    {
        wsprintf(szKey, "SP%2.2d*", i);
        if ( !GetKeyValue(lpszQueryString, szKey, szTmp,
sizeof(szTmp)) )
            return ERR_NEWORDER_MISSING_QTY_KEY;

        if ( szTmp[0] )
            return
ERR_NEWORDER_SUPPW_WITHOUT_ITEMID;

        wsprintf(szKey, "Qty%2.2d*", i);
        if ( !GetKeyValue(lpszQueryString, szKey, szTmp,
sizeof(szTmp)) )
            return ERR_NEWORDER_MISSING_QTY_KEY;

        if ( szTmp[0] )
            return
ERR_NEWORDER_QTY_WITHOUT_ITEMID;

        bCheck = TRUE;
    }
}
if ( items == 0 )
    return ERR_NEWORDER_NOITEMS_ENTERED;

pNewOrderData->o_ol_cnt = items;

```

```

        return ERR_SUCCESS;
    }

/* FUNCTION: int GetPaymentData(LPSTR lpszQueryString, PAYMENT_DATA
 *pPaymentData)
 *
 * PURPOSE:      This function extracts and validates the payment form data from an http
command string.
 *
 * ARGUMENTS:   LPSTR                lpszQueryString        client
browser http command string
 *              PAYMENT_DATA *pPaymentData
 *              pointer to payment data structure
 *
 * RETURNS:     int
 *              error code indicating reason for failure
 *              ERR_SUCCESS
 *              all input data successfully parsed
 *
 * COMMENTS:    None
 */

static int GetPaymentData(LPSTR lpszQueryString, PAYMENT_DATA *pPaymentData)
{
    char    szTmp[26];
    char    *ptr;

    if ( !GetKeyValue(lpszQueryString, "DID*", szTmp, sizeof(szTmp)) )
        return ERR_PAYMENT_MISSING_DID_KEY;
    if ( !IsNumeric(szTmp) )
        return ERR_PAYMENT_DISTRICT_INVALID;
    pPaymentData->d_id = atoi(szTmp);

    if ( !GetKeyValue(lpszQueryString, "CID*", szTmp, sizeof(szTmp)) )
        return ERR_PAYMENT_MISSING_CID_KEY;

    if ( szTmp[0] && !IsNumeric(szTmp) )
        return ERR_PAYMENT_CUSTOMER_INVALID;

    pPaymentData->c_id = atoi(szTmp);

    if ( szTmp[0] == 0 )
    {
        if ( !GetKeyValue(lpszQueryString, "CLT*", szTmp, sizeof(szTmp)) )
            return ERR_PAYMENT_MISSING_CLT;
        _strupr( szTmp );

        strcpy(pPaymentData->c_last, szTmp);
        if ( strlen(pPaymentData->c_last) > 16 )
            return ERR_PAYMENT_LAST_NAME_TO_LONG;
    }
    else
    {
        if ( !GetKeyValue(lpszQueryString, "CLT*", szTmp, sizeof(szTmp)) )
            return ERR_PAYMENT_MISSING_CLT_KEY;
        if ( szTmp[0] )
            return ERR_PAYMENT_CID_AND_CLT;
    }

    if ( !GetKeyValue(lpszQueryString, "CDI*", szTmp, sizeof(szTmp)) )
        return ERR_PAYMENT_MISSING_CDI_KEY;
    if ( !IsNumeric(szTmp) )
        return ERR_PAYMENT_CDI_INVALID;
    pPaymentData->c_d_id = atoi(szTmp);

    if ( !GetKeyValue(lpszQueryString, "CWI*", szTmp, sizeof(szTmp)) )
        return ERR_PAYMENT_MISSING_CWI_KEY;

    if ( !IsNumeric(szTmp) )
        return ERR_PAYMENT_CWI_INVALID;

    pPaymentData->c_w_id = atoi(szTmp);

    if ( !GetKeyValue(lpszQueryString, "HAM*", szTmp, sizeof(szTmp)) )
        return ERR_PAYMENT_MISSING_HAM_KEY;

    ptr = szTmp;

    while( *ptr )
    {
        if ( *ptr == '.' )
        {
            ptr++;
            if ( !*ptr )
                break;
            if ( *ptr < '0' || *ptr > '9' )
                return ERR_PAYMENT_HAM_INVALID;
            ptr++;
            if ( !*ptr )
                break;
            if ( *ptr < '0' || *ptr > '9' )
                return ERR_PAYMENT_HAM_INVALID;
            if ( !*ptr )
                break;
        }
    }
}

```

```

        return ERR_PAYMENT_HAM_INVALID;
    }
    else if ( *ptr < '0' || *ptr > '9' )
        return ERR_PAYMENT_HAM_INVALID;
    ptr++;
}

pPaymentData->h_amount = atof(szTmp);
if ( pPaymentData->h_amount >= 10000.00 || pPaymentData->h_amount < 0 )
    return ERR_PAYMENT_HAM_RANGE;

return ERR_SUCCESS;
}

/* FUNCTION: int GetOrderStatusData(LPSTR lpszQueryString, ORDER_STATUS_DATA
 *pOrderStatusData)
 *
 * PURPOSE:      This function extracts and validates the payment form data from an http
command string.
 *
 * ARGUMENTS:   LPSTR                lpszQueryString
                client browser http command string
                ORDER_STATUS_DATA    *pOrderStatusData
                pointer to order statusdata structure
 *
 * RETURNS:     int
                error code indicating reason for failure
                ERR_SUCCESS
                successfully parsed all required input data
 *
 * COMMENTS:    None
 */
static int GetOrderStatusData(LPSTR lpszQueryString, ORDER_STATUS_DATA
 *pOrderStatusData)
{
    char    szTmp[26];

    if ( !GetKeyValue(lpszQueryString, "DID*", szTmp, sizeof(szTmp)) )
        return ERR_ORDERSTATUS_MISSING_DID_KEY;
    if ( !IsNumeric(szTmp) )
        return ERR_ORDERSTATUS_DID_INVALID;
    pOrderStatusData->d_id = atoi(szTmp);

    if ( !GetKeyValue(lpszQueryString, "CID*", szTmp, sizeof(szTmp)) )
        return ERR_ORDERSTATUS_MISSING_CID_KEY;

    if ( szTmp[0] == 0 )

```

```

    {
        pOrderStatusData->c_id = 0;
        if ( !GetKeyValue(lpszQueryString, "CLT*", szTmp, sizeof(szTmp)) )
            return ERR_ORDERSTATUS_MISSING_CLT_KEY;
        _strupr( szTmp );
        strcpy(pOrderStatusData->c_last, szTmp);
        if ( strlen(pOrderStatusData->c_last) > 16 )
            return ERR_ORDERSTATUS_CLT_RANGE;
    }
    else
    {
        if ( !IsNumeric(szTmp) )
            return ERR_ORDERSTATUS_CID_INVALID;
        pOrderStatusData->c_id = atoi(szTmp);
        if ( !GetKeyValue(lpszQueryString, "CLT*", szTmp, sizeof(szTmp)) )
            return ERR_ORDERSTATUS_MISSING_CLT_KEY;
        if ( szTmp[0] )
            return ERR_ORDERSTATUS_CID_AND_CLT;
    }

    return ERR_SUCCESS;
}

/* FUNCTION: BOOL ReadRegistrySettings(void)
 *
 * PURPOSE:      This function reads the NT registry for startup parameters. There
parameters are
                under the TPCC key.
 *
 * ARGUMENTS:    None
 *
 * RETURNS:      None
 *
 * COMMENTS:     This function also sets up required operation variables to the default val
so if registry is not setup the default values will be
used.
 */
static BOOL ReadRegistrySettings(void)
{
    HKEY    hKey;
    DWORD   size;
    DWORD   type;
    char    szTmp[256];

    bLog    = FALSE;
    dLog    = FALSE;

```

```

iMaxWareHouses = 500;
iThreads       = 5;
iQSlotts      = 3000;
iDelayMs      = 100;
iDeadlockRetry = (short)3;
strcpy(szTpccLogPath, "tpcclog.");
strcpy(szErrorLogPath, "tpccerr.");

if ( RegOpenKeyEx(HKEY_LOCAL_MACHINE, "SOFTWARE\\Microsoft\\TPCC", 0,
KEY_READ, &hKey) != ERROR_SUCCESS )
    return TRUE;
size = sizeof(szTmp);

if ( RegQueryValueEx(hKey, "PATH", 0, &type, szTmp, &size) ==
ERROR_SUCCESS )
{
    strcpy(szTpccLogPath, szTmp);
    strcat(szTpccLogPath, "tpcclog.");
    strcpy(szErrorLogPath, szTmp);
    strcat(szErrorLogPath, "tpccerr.");
}

size = sizeof(szTmp);
if ( RegQueryValueEx(hKey, "LOG", 0, &type, szTmp, &size) == ERROR_SUCCESS
)
{
    if ( !strcmp(szTmp, "ON") )
        bLog = TRUE;
}

size = sizeof(szTmp);
if ( RegQueryValueEx(hKey, "DEBUG", 0, &type, szTmp, &size) ==
ERROR_SUCCESS )
{
    if ( !strcmp(szTmp, "ON") )
        dLog = TRUE;
}

size = sizeof(szTmp);
if ( RegQueryValueEx(hKey, "MaximumWarehouses", 0, &type, szTmp, &size) ==
ERROR_SUCCESS )
{
    iMaxWareHouses = atoi(szTmp);
    if ( iMaxWareHouses == 0 )
        iMaxWareHouses = 500;
}

size = sizeof(szTmp);
if ( RegQueryValueEx(hKey, "NumberOfDeliveryThreads", 0, &type, szTmp, &size) ==
ERROR_SUCCESS )
    iThreads = atoi(szTmp);
if ( !iThreads )
    iThreads = 5;

size = sizeof(szTmp);
if ( RegQueryValueEx(hKey, "QueueSlits", 0, &type, szTmp, &size) ==
ERROR_SUCCESS )
    iQSlotts = atoi(szTmp);
if ( !iQSlotts )
    iQSlotts = 3000;

size = sizeof(szTmp);
if ( RegQueryValueEx(hKey, "BackoffDelay", 0, &type, szTmp, &size) ==
ERROR_SUCCESS )
    iDelayMs = atoi(szTmp);
if ( !iDelayMs )
    iDelayMs = 100;

size = sizeof(szTmp);
if ( RegQueryValueEx(hKey, "DeadlockRetry", 0, &type, szTmp, &size) ==
ERROR_SUCCESS )
    iDeadlockRetry = (short)atoi(szTmp);
if ( !iDeadlockRetry )
    iDeadlockRetry = (short)3;

size = sizeof(szTmp);
if ( RegQueryValueEx(hKey, "MaxConnections", 0, &type, szTmp, &size) ==
ERROR_SUCCESS )
    iMaxConnections = (short)atoi(szTmp);
if ( !iMaxConnections )
    iMaxConnections = (short)25;

RegCloseKey(hKey);

return FALSE;
}

/* FUNCTION: BOOL IsNumeric(char *ptr)
 *
 * PURPOSE:      This function determines if a string is numeric. It fails if any characters
other
 *
 *               than numeric and null terminator are present.

```

```

*
* ARGUMENTS:  char          *ptr    pointer to string to check.
*
* RETURNS:    BOOL  FALSE  if string is not all numeric
*             TRUE   if string contains only
numeric characters i.e. '0' - '9'
*
* COMMENTS:   None
*
*/

static BOOL IsNumeric(char *ptr)
{
    if ( *ptr == 0 )
        return FALSE;

    while( *ptr && isdigit(*ptr) )
        ptr++;
    return ( !*ptr );
}

/* FUNCTION: void FormatHTMLString(char *szBuff, int iLen, char *szStr)
*
* PURPOSE:    This function Handles translation of HTML specific character field data
              when an HTML output form is generated.
*
* ARGUMENTS:  char    *szBuff  Returned string information
              char    *szStr  input string to be formatted.
              int     iLen    Length of returned
string
*
* RETURNS:    none
*
* COMMENTS:   The length paramter is the absolute length of the returned string in
              HTML characters. For example the input string>
would be returned as
              &gt; which would be counted as 1 character.If the
number of input
              characters is less than the iLen parameter spaces are
appended to
              the end of the string to ensure that at least iLen
characters are
              returned in the szBuff parameter.
*
*/

static void FormatHTMLString(char *szBuff, char *szStr, int iLen)

```

```

{
    while( iLen && *szStr )
    {
        switch( *szStr )
        {
            case '>':
                *szBuff++ = '&';
                *szBuff++ = 'g';
                *szBuff++ = 't';
                *szBuff++ = ';';
                szStr++;
                break;

            case '<':
                *szBuff++ = '&';
                *szBuff++ = 'l';
                *szBuff++ = 't';
                *szBuff++ = ';';
                szStr++;
                break;

            case '&':
                *szBuff++ = '&';
                *szBuff++ = 'a';
                *szBuff++ = 'm';
                *szBuff++ = 'p';
                *szBuff++ = ';';
                szStr++;
                break;

            case '\":
                *szBuff++ = '&';
                *szBuff++ = 'q';
                *szBuff++ = 'u';
                *szBuff++ = 'o';
                *szBuff++ = 't';
                *szBuff++ = ';';
                szStr++;
                break;

            default:
                *szBuff++ = *szStr++;
                break;
        }
        iLen--;
    }
    while( iLen-- )
        *szBuff++ = ' ';

    *szBuff = 0;

    return;
}

```

```

}

static int ThrTpInit()
{
    static int num_tpinit=0;
    static int x=1;
    static int once=0;
    static CRITICAL_SECTION TpCriticalSection;
    int lasterr, iRc, TpRc;
    int retry = 0;
    BOOL Success = FALSE;

    if(!TlsGetValue(TLSIsTpInitKey))
    {
        if (!once)
        {
            InitializeCriticalSection(&TpCriticalSection);
            once=1;
        }

        if ( dLog )
        {
            FILE *fp;

            fp = fopen(szTpccLogPath, "ab");
            fprintf(fp, "** In ThrTpInit Thread %d * \n",
GetCurrentThreadId());
            fclose(fp);
        }

        while ( retry < TP_MAX_RETRIES )
        {
            EnterCriticalSection(&TpCriticalSection);
            if(tpinf == NULL)
            {
                if ((tpinf = ( TPINIT *)tpalloc("TPINIT", NULL,
sizeof(TPINIT))) == NULL)
                {
                    LeaveCriticalSection(&TpCriticalSection);
                    TpRc = tperrno;
                    {
                        FILE *fp;

                        fp = fopen(szErrorLogPath, "ab");
                        fprintf(fp, ">>>> ThrTpInit:%d :
tpalloc of tpinf failed: %d : %s\n",

```

```

GetCurrentThreadId
TpRc, tpsterror(TpRc));
                                fclose(fp);
                                }
                                retry++;
                                continue;
                                }
                                tpinf->flags|=TPMULTICONTEXTS;
                                }

                                if (retry == 0)
                                    itoa(++num_tpinit, tpinf->clname, 10);

// Do the TPINIT

                                iRc = tpinit(tpinf);
                                TpRc = tperrno;

                                // check tpalloc() ?
                                if (iRc < 0)
                                {
                                    LeaveCriticalSection(&TpCriticalSection);
                                    retry++;
                                    lasterr = GetLastError();
                                    TpRc = tperrno;
                                    {
                                        FILE *fp;

                                        fp = fopen(szErrorLogPath, "ab");
                                        fprintf(fp, ">>>> ThrTpInit:%d : tpinit faile
GetCurrentThreadId(), iRc,
                                        fclose(fp);
                                    }
                                }
                                else
                                {
                                    Success = TRUE;
                                    LeaveCriticalSection(&TpCriticalSection);
                                    break;
                                }

                                Sleep(50); // Relinquish thread timeslice
                                } // retry the tpinit if it failed the first time

                                if ( Success == FALSE )
                                {

```



```

        {
            char ebuf[128];

            sprintf(ebuf, ">>>> ThrTplnit %d : Cannot tpinit after
%d tries iRc = %d LastErr = %d \r\n",
lasterr);
            GetCurrentThreadId(), TP_MAX_RETRIES, iRc,
            LogTuxError(TpRc, ebuf);
        }
        return -1;
    }
    if ( Success == TRUE )
    {
        if ( retry > 0 )
        {
            char ebuf[128];

            sprintf(ebuf, ">>>> ThrTplnit %d : Cannot tpinit after
%d tries iRc = %d LastErr = %d \r\n",
lasterr);

            count %d with LastErr = %d * \r\n",
                sprintf(ebuf, "** ThrTplnit Thread %d Success retry
                GetCurrentThreadId(), retry, lasterr);
                LogTuxError(TpRc, ebuf);
            }
            if ( ( iRc=TlsSetValue(TLSIsTpInitedKey,&x) ==0)
            {
                {
                    FILE *fp;

                    fp = fopen(szErrorLogPath, "ab");
                    fprintf(fp, ">>>> ThrTplnit %d : TlsSetValue

                    GetCurrentThreadId(), iRc);
                    fclose(fp);
                }
            }
        }
    }
    else
    {
        if ( dLog )
        {
            FILE *fp;

```

```

        fp = fopen(szTpccLogPath, "ab");
        fprintf(fp, "** ThrTplnit Thread %d already tpinited * \r\n",
        fclose(fp);
    }
}
return 0;
}

```

A.14 tpcc.def

LIBRARY TPCC.DLL

EXPORTS

```

        GetExtensionVersion      @1
        HttpExtensionProc @2

```

A.15 tpcc.h

```

/*      FILE:          TPCC.H
*
*      Microsoft TPC-C Kit Ver. 3.00.001
*      Audited 08/23/96, By Francois Raab
*
*      Copyright Microsoft, 1996
*
*      PURPOSE:       Header file for ISAPI TPCC.DLL, defines structures and
functions used in the isapi tpcc.dll.
*      Author:        Philip Durr
*                    philipdu@Microsoft.com
*/

#define LOCAL_ALLOC 1

//VERSION RESOURCE DEFINES
#define _APS_NEXT_RESOURCE_VALUE          101
#define _APS_NEXT_COMMAND_VALUE
40001
#define _APS_NEXT_CONTROL_VALUE
1000
#define _APS_NEXT_SYMED_VALUE          101

#define TP_MAX_RETRIES
50

```

#define ERR_BAD_ITEM_ID	1	//expected abort record in txnRecord	#define ERR_INVALID_SYNC_CONNECTION	1014	//Invalid Terminal Sync ID.
#define ERR_TYPE_DELIVERY_POST	2	//expected delivery post failed	#define ERR_INVALID_TERMID	1015	//Invalid Terminal ID.
#define ERR_TYPE_WEBDLL		//tpcc web generated error	#define ERR_PAYMENT_INVALID_CUSTOMER	1016	//Payment Form, No such Customer.
#define ERR_TYPE_SQL	4	//sql server generated error	#define ERR_SQL_OPEN_CONNECTION	1017	//SQLOpenConnection API Failed.
#define ERR_TYPE_DBLIB	5	//dblib generated error	#define ERR_STOCKLEVEL_MISSING_THRESHOLD_KEY	1018	//Stock Level missing Threshold key "TT".
#define ERR_TYPE_ODBC	6	//odbc generated error	#define ERR_STOCKLEVEL_THRESHOLD_INVALID	1019	//Stock Level Threshold invalid data type range = 1 - 99.
#define ERR_TYPE_SOCKET	7	//error on communication socket client rte only	#define ERR_STOCKLEVEL_THRESHOLD_RANGE	1020	//Stock Level Threshold out of range, range must be 1 - 99.
#define ERR_TYPE_DEADLOCK	8	//dblib and odbc only deadlock condition	#define ERR_STOCKLEVEL_NOT_PROCESSED	1021	//Stock Level not processed.
#define ERR_TYPE_TUXEDO	9	//tuxedo error	#define ERR_NEWORDER_FORM_MISSING_DID	1022	//New Order missing District key "DID".
#define ERR_SUCCESS	1000	//Success, no error.	#define ERR_NEWORDER_DISTRICT_INVALID	1023	//New Order District ID Invalid range 1 - 10.
#define ERR_COMMAND_UNDEFINED	1001	//Command undefined.	#define ERR_NEWORDER_DISTRICT_RANGE	1024	//New Order District ID out of Range. Range = 1 - 10.
#define ERR_NOT_IMPLEMENTED_YET	1002	//Not Implemented Yet.	#define ERR_NEWORDER_CUSTOMER_KEY	1025	//New Order missing Customer key "CID".
#define ERR_CANNOT_INIT_TERMINAL	1003	//Cannot initialize client connection.	#define ERR_NEWORDER_CUSTOMER_INVALID	1026	//New Order customer id invalid data type, range = 1 to 3000.
#define ERR_OUT_OF_MEMORY	1004	//insufficient memory.	#define ERR_NEWORDER_CUSTOMER_RANGE	1027	//New Order customer id out of range, range = 1 to 3000.
#define ERR_NEW_ORDER_NOT_PROCESSED	1005	//Cannot process new Order form.	#define ERR_NEWORDER_MISSING_IID_KEY	1028	//New Order missing Item Id key "IID".
#define ERR_PAYMENT_NOT_PROCESSED	1006	//Cannot process payment form.	#define ERR_NEWORDER_ITEM_BLANK_LINES	1029	//New Order blank order lines all orders must be continuous.
#define ERR_NO_SERVER_SPECIFIED	1007	//No Server name specified.	#define ERR_NEWORDER_ITEMID_INVALID	1030	//New Order Item Id is wrong data type, must be numeric.
#define ERR_ORDER_STATUS_NOT_PROCESSED	1008	//Cannot process order status form.	#define ERR_NEWORDER_MISSING_SUPPW_KEY	1031	//New Order missing Supp_W key "SP##".
#define ERR_W_ID_INVALID	1009	//Invalid Warehouse ID.	#define ERR_NEWORDER_SUPPW_INVALID	1032	//New Order Supp_W invalid data type must be numeric.
#define ERR_CAN_NOT_SET_MAX_CONNECTIONS	1010	//Insufficient memory to allocate # connections.	#define ERR_NEWORDER_MISSING_QTY_KEY	1033	//New Order Missing Qty key "Qty##".
#define ERR_NOSUCH_CUSTOMER	1011	//No such customer.	#define ERR_NEWORDER_QTY_INVALID	1034	//New Order Qty invalid must be numeric range 1 - 99.
#define ERR_D_ID_INVALID	1012	//Invalid District ID Must be 1 to 10.	#define ERR_NEWORDER_SUPPW_RANGE	1035	//New Order Supp_W value out of range range = 1 - Max Warehouses.
#define ERR_MAX_CONNECT_PARAM	1013	//Max client connections exceeded, run install to increase.	#define ERR_NEWORDER_ITEMID_RANGE	1036	//New Order Item Id is out of range. Range = 1 to 999999.
			#define ERR_NEWORDER_QTY_RANGE	1037	//New Order Qty is out of range. Range = 1 to 99.

```

#define ERR_PAYMENT_DISTRICT_INVALID 1038 //Payment
District ID is invalid must be 1 - 10.
#define ERR_NEWORDER_SUPPW_WITHOUT_ITEMID 1039 //New
Order Supp_W field entered without a corresponding Item_Id.
#define ERR_NEWORDER_QTY_WITHOUT_ITEMID 1040
//New Order Qty entered without a corresponding Item_Id.
#define ERR_NEWORDER_NOITEMS_ENTERED 1041
//New Order Blank Items between items, items must be continuous.
#define ERR_PAYMENT_MISSING_DID_KEY 1042
//Payment missing District Key "DID*".
#define ERR_PAYMENT_DISTRICT_RANGE 1043
//Payment District Out of range, range = 1 - 10.
#define ERR_PAYMENT_MISSING_CID_KEY 1044
//Payment missing Customer Key "CID*".
#define ERR_PAYMENT_CUSTOMER_INVALID 1045
//Payment Customer data type invalid, must be numeric.
#define ERR_PAYMENT_MISSING_CLT 1046
//Payment missing Customer Last Name Key "CLT*".
#define ERR_PAYMENT_LAST_NAME_TO_LONG 1047
//Payment Customer last name longer than 16 characters.
#define ERR_PAYMENT_CUSTOMER_RANGE 1048
//Payment Customer ID out of range, must be 1 to 3000.
#define ERR_PAYMENT_CID_AND_CLT 1049
//Payment Customer ID and Last Name entered must be one or other.
#define ERR_PAYMENT_MISSING_CDI_KEY 1050
//Payment missing Customer district key "CDI*".
#define ERR_PAYMENT_CDI_INVALID 1051
//Payment Customer district invalid must be numeric.
#define ERR_PAYMENT_CDI_RANGE 1052
//Payment Customer district out of range must be 1 - 10.
#define ERR_PAYMENT_MISSING_CWI_KEY 1053
//Payment missing Customer Warehouse key "CWI*".
#define ERR_PAYMENT_CWI_INVALID 1054
//Payment Customer Warehouse invalid must be numeric.
#define ERR_PAYMENT_CWI_RANGE 1055
//Payment Customer Warehouse out of range, 1 to Max Warehouses.
#define ERR_PAYMENT_MISSING_HAM_KEY 1056
//Payment missing Amount key "HAM*".
#define ERR_PAYMENT_HAM_INVALID 1057
//Payment Amount invalid data type must be numeric.
#define ERR_PAYMENT_HAM_RANGE 1058
//Payment Amount out of range, 0 - 9999.99.
#define ERR_ORDERSTATUS_MISSING_DID_KEY 1059
//Order Status missing District key "DID*".
#define ERR_ORDERSTATUS_DID_INVALID 1060
//Order Status District invalid, value must be numeric 1 - 10.
#define ERR_ORDERSTATUS_DID_RANGE 1061
//Order Status District out of range must be 1 - 10.

#define ERR_ORDERSTATUS_MISSING_CID_KEY 1062
//Order Status missing Customer key "CID*".
#define ERR_ORDERSTATUS_MISSING_CLT_KEY 1063
//Order Status missing Customer Last Name key "CLT*".
#define ERR_ORDERSTATUS_CLT_RANGE 1064
//Order Status Customer last name longer than 16 characters.
#define ERR_ORDERSTATUS_CID_INVALID 1065
//Order Status Customer ID invalid, range must be numeric 1 - 3000.
#define ERR_ORDERSTATUS_CID_RANGE 1066
//Order Status Customer ID out of range must be 1 - 300.
#define ERR_ORDERSTATUS_CID_AND_CLT 1067
//Order Status Customer ID and LastName entered must be only one."
#define ERR_DELIVERY_MISSING_OCD_KEY 1068 //Delivery
missing Carrier ID key "OCD*".
#define ERR_DELIVERY_CARRIER_INVALID 1069 //Delivery
Carrier ID invalid must be numeric 1 - 10.
#define ERR_DELIVERY_CARRIER_ID_RANGE 1070
//Delivery Carrier ID out of range must be 1 - 10.
#define ERR_PAYMENT_MISSING_CLT_KEY 1071
//Payment missing Customer Last Name key "CLT*".
#define ERR_TPINIT_BAD 5001 //Bad TPINIT"
#define ERR_TPALLOC_BAD 5002 //Bad TPALLOC"
#define ERR_TPCALL_BAD 5003 //Bad TPCALL"

//note that the welcome form must be processed first as terminal ids assigned here, once the
//terminal id is assigned then the forms can be processed in any order.
#define WELCOME_FORM 1
//beginning form no term id assigned, form id
#define MAIN_MENU_FORM 2
//term id assigned main menu form id
#define NEW_ORDER_FORM 3
//new order form id
#define PAYMENT_FORM 4
//payment form id
#define DELIVERY_FORM 5
//delivery form id
#define ORDER_STATUS_FORM 6
//order status id
#define STOCK_LEVEL_FORM 7
//stock level form id

//This macro is used to prevent the compiler error unused formal parameter
#define UNUSEDPARAM(x) (x = x)

//error message structure used in ErrorMessage API

```

```

typedef struct _SERRORMSG
{
    int          iError;                //error id of message
    char        szMsg[80];              //message to sent to browser
} SERRORMSG;

//This structure is used for posting delivery transactions
typedef struct _DELIVERY_TRANSACTION
{
    SYSTEMTIME   queue;                 //time delivery transaction queued
    short        w_id;                  //delivery warehouse
    short        o_carrier_id;          //carrier id
} DELIVERY_TRANSACTION;

#ifdef USE_ODBC
    typedef struct _DBPROCESS
    {
        HDBC     hdbc;
        HSTMT    hstmt;
        int       spid;
        void      *uPtr;
    } DBPROCESS, *PDBPROCESS;

    //dblib error message return values
    #define INT_EXIT      0
    #define INT_CONTINUE 1
    #define INT_CANCEL   2
#endif

//This structure defines the data necessary to keep distinct for each terminal or client connection.
typedef struct _CLIENTDATA
{
    int          inUse;
    //in use flag allows client entries to be reused
    int          w_id;
    //warehouse id assigned at welcome form
    int          d_id;
    //district id assigned at welcome form

    PDBPROCESS   dbproc;                //dblib
    connection pointer
    int          spid;
    //spid assigned from dblib
    int          iSyncId;
    //synchronization id
    int          iTickCount;
    //time of last access;

```

```

    int          iTermId;                //terminal
    of http stream connection

    char        szBuffer[4096];         //form buffer each
    HTML form is built for a client in here

    NEW_ORDER_DATA      NewOrderData;    //new order
    form data

    PAYMENT_DATA        PaymentData;      //payment form data
    ORDER_STATUS_DATA   OrderStatusData;  //order status form data
    DELIVERY_DATA        DeliveryData;     //delivery form data
    STOCK_LEVEL_DATA    StockLevelData;   //stock level form data

#ifdef LOCAL_ALLOC
    TUX_DATA             *TuxDataPtr;      //Tuxedo
    Data Structure for all transactions
#endif // LOCAL_ALLOC

} CLIENTDATA;

typedef CLIENTDATA *PCLIENTDATA;        //pointer to client structure

//This structure is used to define the operational interface for terminal id support
typedef struct _TERM
{
    int          iAvailable;
    //total allocated terminal array entries
    int          iNext;
    //next available terminal array element

    int          iMasterSyncId;
    //synchronization id
    BOOL         bInit;
    //structure has been initialized flag
    CLIENTDATA   *pClientData;
    //pointer to allocated client data
    void          (*Init)(void);
    //API to initialize this structure
    int          (*Allocate)(void);
    //API to allocate a new terminal entry array id returned
    void          (*Restore)(void);      //API to free

    terminal data
    int          (*Add)(EXTENSION_CONTROL_BLOCK *pECB,
    char *pQueryString); //API to add a terminal id to array, this context will

    //be passed from the browser to the tpcc.dll in the

    //TERMINID= key in the HTTP string.
    void          (*Delete)(EXTENSION_CONTROL_BLOCK *pECB, int id);

```

```

        //API to free resources used by a terminal array entry
    } TERM;

typedef TERM *PTERM;
        //pointer to terminal structure type

//this structure allows the EXTENSION CONTROL BLOCK to be passed to the msg and error
handlers.
typedef struct _ECBINFO
{
        int                                iTermId; //terminal id
        int                                iSynclId; //browser
    sync id
        BOOL                                bDeadlock; //deadlock
    condition flag
        BOOL                                bFailed; //cleared before sql
    transaction, set in err handlers if an error occurs
        EXTENSION_CONTROL_BLOCK *pECB; //inetsrv current
    connection structure information
} ECBINFO, *PECBINFO;

//function prototypes

BOOL APIENTRY DIIMain(HANDLE hModule, DWORD ul_reason_for_call, LPVOID
lpReserved);
static void DeliveryDisconnect(void *ptr);
static BOOL IsValidTermId(int TermId);
BOOL ProcessQueryString(EXTENSION_CONTROL_BLOCK *pECB, int *pCmd, int *pFormId,
int *pTermId, int *pSynclId);
void NewOrderForm(EXTENSION_CONTROL_BLOCK *pECB, int iFormId, int iTermId, int
iSynclId);
void PaymentForm(EXTENSION_CONTROL_BLOCK *pECB, int iFormId, int iTermId, int
iSynclId);
void DeliveryForm(EXTENSION_CONTROL_BLOCK *pECB, int iFormId, int iTermId, int
iSynclId);
void OrderStatusForm(EXTENSION_CONTROL_BLOCK *pECB, int iFormId, int iTermId, int
iSynclId);
void StockLevelForm(EXTENSION_CONTROL_BLOCK *pECB, int iFormId, int iTermId, int
iSynclId);
void Exitcmd(EXTENSION_CONTROL_BLOCK *pECB, int iFormId, int iTermId, int iSynclId);
void SubmitCmd(EXTENSION_CONTROL_BLOCK *pECB, int iFormId, int iTermId, int
iSynclId);
void BeginCmd(EXTENSION_CONTROL_BLOCK *pECB, int iFormId, int iTermId, int iSynclId);
void ProcessCmd(EXTENSION_CONTROL_BLOCK *pECB, int iFormId, int iTermId, int
iSynclId);
void ClearCmd(EXTENSION_CONTROL_BLOCK *pECB, int iFormId, int iTermId, int iSynclId);
void MenuCmd(EXTENSION_CONTROL_BLOCK *pECB, int iFormId, int iTermId, int iSynclId);

```

```

void NumberOfConnectionsCmd(EXTENSION_CONTROL_BLOCK *pECB, int iFormId, int
iTermId, int iSynclId);
static void WriteZString(EXTENSION_CONTROL_BLOCK *pECB, char *szStr);
static void h_printf(EXTENSION_CONTROL_BLOCK *pECB, char *format, ...);
void LogTuxError(int TpRc, char *ErrMsg);
void ErrorMessage(EXTENSION_CONTROL_BLOCK *pECB, int iError, int iErrorType, char
*szMsg, int iTermId, int iSynclId);
static BOOL GetKeyValue(char *pQueryString, char *pKey, char *pValue, int iMax);
static void TermInit(void);
int err_handler(DBPROCESS *dbproc, int severity, int dberr, int oserr, char *dberrstr, char
*oserrstr);
int msg_handler(DBPROCESS *dbproc, DBINT msgno, int msgstate, int severity, char
*msgtext);
static void TermRestore(void);
static int TermAllocate(void);
static int TermAdd(EXTENSION_CONTROL_BLOCK *pECB, char *pQueryString);
static void TermDelete(EXTENSION_CONTROL_BLOCK *pECB, int id);
BOOL Init(EXTENSION_CONTROL_BLOCK *pECB, int iTermId, int iSynclId, char *szServer
char *szUser, char *szPassword, char *szDatabase);
static BOOL Close(EXTENSION_CONTROL_BLOCK *pECB, int iTermId, int iSynclId);
static BOOL SQLOpenConnection(EXTENSION_CONTROL_BLOCK *pECB, int iTermId, int
iSynclId, DBPROCESS **dbproc, char *server, char *database, char *user, char *password,
i
*app, int *spid);
static BOOL SQLCloseConnection(EXTENSION_CONTROL_BLOCK *pECB, DBPROCESS
*dbproc);
static BOOL SQLStockLevel(EXTENSION_CONTROL_BLOCK *pECB, int iTermId, int iSyncl
DBPROCESS *dbproc, STOCK_LEVEL_DATA *pStockLevel, short deadlock_retry);
static int SQLNewOrder(EXTENSION_CONTROL_BLOCK *pECB, int iTermId, int iSynclId
DBPROCESS *dbproc, NEW_ORDER_DATA *pNewOrder, short deadlock_retry);
static int SQLPayment(EXTENSION_CONTROL_BLOCK *pECB, int iTermId, int iSynclId,
DBPROCESS *dbproc, PAYMENT_DATA *pPayment, short deadlock_retry);
static int SQLOrderStatus(EXTENSION_CONTROL_BLOCK *pECB, int iTermId, int iSynclId
DBPROCESS *dbproc, ORDER_STATUS_DATA *pOrderStatus, short deadlock_retry);
static int SQLDelivery(DBPROCESS *dbproc, DELIVERY_DATA *pDelivery, short
deadlock_retry);
static void WriteLog(DELIVERY_DATA *pDelivery, SYSTEMTIME *trans_end);
static void CalculateElapsedTime(int *pElapsed, LPSYSTEMTIME lpBegin, LPSYSTEMTIME
lpEnd);
BOOL SQLDetectDeadlock(DBPROCESS *dbproc);
static void FormatString(char *szDest, char *szPic, char *szSrc);
static char *MakeStockLevelForm(int iTermId, int iSynclId, BOOL bInput);
static char *MakeMainMenuForm(int iTermId, int iSynclId);
static char *MakeWelcomeForm(void);
static char *MakeNewOrderForm(int iTermId, int iSynclId, BOOL Rollback, BOOL bInput, BO
bValid);
static char *MakePaymentForm(int iTermId, int iSynclId, BOOL bInput);
static char *MakeOrderStatusForm(int iTermId, int iSynclId, BOOL bInput);
static char *MakeDeliveryForm(int iTermId, int iSynclId, BOOL bInput, BOOL bSuccess);

```

```

static void UtilStrCpy(char * pDest, char * pSrc, int n);
static void ProcessNewOrderForm(EXTENSION_CONTROL_BLOCK *pECB, int iTermlId, int
iSynclId);
static void ProcessPaymentForm(EXTENSION_CONTROL_BLOCK *pECB, int iTermlId, int
iSynclId);
static void ProcessOrderStatusForm(EXTENSION_CONTROL_BLOCK *pECB, int iTermlId, int
iSynclId);
static void ProcessDeliveryForm(EXTENSION_CONTROL_BLOCK *pECB, int iTermlId, int
iSynclId);
static void ProcessStockLevelForm(EXTENSION_CONTROL_BLOCK *pECB, int iTermlId, int
iSynclId);
static int GetNewOrderData(LPSTR lpszQueryString, NEW_ORDER_DATA *pNewOrderData);
static int GetPaymentData(LPSTR lpszQueryString, PAYMENT_DATA *pPaymentData);
static int GetOrderStatusData(LPSTR lpszQueryString, ORDER_STATUS_DATA
*pOrderStatusData);
static BOOL ReadRegistrySettings(void);
static BOOL PostDeliveryInfo(short w_id, short o_carrier_id);
static BOOL IsNumeric(char *ptr);
static void FormatHTMLString(char *szBuff, char *szStr, int iLen);
static void PrintParameters(void);

#ifdef USE_ODBC
    void dbsetuserdata(PDBPROCESS dbproc, void *uPtr);
    void *dbgetuserdata(PDBPROCESS dbproc);
    void BindParameter(PDBPROCESS dbproc, UWORD ipar, SWORD fCType,
SWORD fSqlType, UDWORD cbColDef, SWORD ibScale, PTR rgbValue, SDWORD
cbValueMax);
    void ODBCError(PDBPROCESS dbproc);
    BOOL ExecuteStatement(PDBPROCESS dbproc, char *szStatement);
    BOOL BindColumn(PDBPROCESS dbproc, SQLUSMALLINT icol, SQLSMALLINT
fCType, SQLPOINTER rgbValue, SQLINTEGER cbValueMax);
    BOOL GetResults(PDBPROCESS dbproc);
    BOOL MoreResults(PDBPROCESS dbproc);
    BOOL ReopenConnexion(PDBPROCESS dbproc);
#endif

```

A.16 tpcc.rc

```

//Microsoft Developer Studio generated resource script.
//
#include "resource.h"

#define APSTUDIO_READONLY_SYMBOLS
//
// Generated from the TEXTINCLUDE 2 resource.
//

```

```

#include "afxres.h"

//
//
// undef APSTUDIO_READONLY_SYMBOLS
//
//
// English (U.S.) resources
//
//
// if !defined(AFX_RESOURCE_DLL) || defined(AFX_TARG_ENU)
// #ifdef _WIN32
LANGUAGE LANG_ENGLISH, SUBLANG_ENGLISH_US
#pragma code_page(1252)
// #endif // _WIN32
//
// #ifndef _MAC
//
//
// Version
//
//
VS_VERSION_INFO VERSIONINFO
FILEVERSION 0,3,0,2
PRODUCTVERSION 0,3,0,2
FILEFLAGSMASK 0x3fL
#ifdef _DEBUG
FILEFLAGS 0x1L
#else
FILEFLAGS 0x0L
#endif
FILEOS 0x40004L
FILETYPE 0x2L
FILESUBTYPE 0x0L
BEGIN
    BLOCK "StringFileInfo"
    BEGIN
        BLOCK "040904b0"
        BEGIN
            VALUE "Comments", "TPC-C HTML DLL Server\0"
            VALUE "CompanyName", "Microsoft\0"
            VALUE "FileDescription", "tpcc\0"
            VALUE "FileVersion", "0, 3, 0, 2\0"
            VALUE "InternalName", "tpcc\0"
            VALUE "LegalCopyright", "Copyright © 1996\0"
            VALUE "OriginalFilename", "tpcc.dll\0"
            VALUE "ProductName", "Microsoft tpcc\0"
            VALUE "ProductVersion", "0, 3, 0, 2\0"
        END
    END
END

```

```

BLOCK "VarFileInfo"
BEGIN
  VALUE "Translation", 0x409, 1200
END
END

#endif // !_MAC

#ifdef APSTUDIO_INVOKED
////////////////////////////////////
//
// TEXTINCLUDE
//

1 TEXTINCLUDE DISCARDABLE
BEGIN
  "resource.h\0"
END

2 TEXTINCLUDE DISCARDABLE
BEGIN
  "#include ""afxres.h""\r\n"
  "\0"
END

3 TEXTINCLUDE DISCARDABLE
BEGIN
  "\r\n"
  "\0"
END

#endif // APSTUDIO_INVOKED

#endif // English (U.S.) resources
////////////////////////////////////

#ifdef APSTUDIO_INVOKED
////////////////////////////////////
//
// Generated from the TEXTINCLUDE 3 resource.
//

////////////////////////////////////
#endif // not APSTUDIO_INVOKED

```

A.17 trans.h

```

/*      FILE:          TRANS.H
 *
 *      Microsoft TPC-C Kit Ver. 3.00.000
 *      Audited 08/23/96   By Francois Raab
 *
 *      PURPOSE:      Header file for ISAPI TPCC.DLL, defines structures and
 *      functions used in the isapi tpcc.dll.
 *
 *
 *      Copyright Microsoft inc. 1996, All Rights Reserved
 *
 *      Author:       PhilipDu, from tpcc.h by DamienL
 *                  DamienL@Microsoft.com
 *                  philipdu@Microsoft.com
 */

#ifdef _INC_TRANS

#define _INC_TRANS

#ifdef USE_ODBC
    #ifndef TIMESTAMP_STRUCT
        #include <sqltypes.h>
    #endif
#else
    #ifndef _INC_SQLFRONT
        #include <sqlfront.h>
    #endif
#endif

#ifdef DBINT
    typedef long DBINT;
#endif

#define DEFCLPACKSIZE          4096
#define DEADLOCKWAIT          10

// String length constants
#define SERVER_NAME_LEN        20
#define DATABASE_NAME_LEN     20
#define USER_NAME_LEN          20
#define PASSWORD_LEN           20
#define TABLE_NAME_LEN       20
#define I_DATA_LEN             50
#define I_NAME_LEN             24
#define BRAND_LEN              1
#define LAST_NAME_LEN          16

```

```

#define W_NAME_LEN      10
#define ADDRESS_LEN     20
#define STATE_LEN       2
#define ZIP_LEN         9
#define S_DIST_LEN     24
#define S_DATA_LEN     50
#define D_NAME_LEN     10
#define FIRST_NAME_LEN 16
#define MIDDLE_NAME_LEN 2
#define PHONE_LEN      16
#define DATETIME_LEN   30
#define CREDIT_LEN     2
#define C_DATA_LEN     250
#define H_DATA_LEN     24
#define DIST_INFO_LEN  24
#define MAX_OL_NEW_ORDER_ITEMS 15
#define MAX_OL_ORDER_STATUS_ITEMS 15
#define STATUS_LEN     25
#define OL_DIST_INFO_LEN 24

// transaction structures

typedef struct
{
    short      ol_supply_w_id;
    long       ol_i_id;
    char       ol_i_name[1_NAME_LEN+1];
    short      ol_quantity;
    char
ol_brand_generic[BRAND_LEN+1];
    double     ol_i_price;
    double     ol_amount;
    short      ol_stock;
    short      num_warehouses;
} OL_NEW_ORDER_DATA;

typedef struct
{
    short      w_id;
    short      d_id;
    long       c_id;
    short      o_ol_cnt;
    char       c_last[1LAST_NAME_LEN+1];
    char       c_credit[1CREDIT_LEN+1];
    double     c_discount;
    double     w_tax;
    double     d_tax;
    long       o_id;

```

```

    short      o_commit_flag;
#ifdef USE_ODBC
    TIMESTAMP_STRUCT o_entry_d;
#else
    DBDATAREC      o_entry_d;
#endif
    short      o_all_local;
    double     total_amount;
    long       num_deadlocks;
    int        retval;
    int        error;
    char
execution_status[1STATUS_LEN];
    OL_NEW_ORDER_DATA OI[1MAX_OL_NEW_ORDER_ITEMS];
} NEW_ORDER_DATA;

typedef struct
{
    short      w_id;
    short      d_id;
    long       c_id;
    short      c_d_id;
    short      c_w_id;
    double     h_amount;
#ifdef USE_ODBC
    TIMESTAMP_STRUCT h_date;
#else
    DBDATAREC      h_date;
#endif
    char       w_street_1[1ADDRESS_LEN+1];
    char       w_street_2[1ADDRESS_LEN+1];
    char       w_city[1ADDRESS_LEN+1];
    char       w_state[1STATE_LEN+1];
    char       w_zip[1ZIP_LEN+1];
    char       d_street_1[1ADDRESS_LEN+1];
    char       d_street_2[1ADDRESS_LEN+1];
    char       d_city[1ADDRESS_LEN+1];
    char       d_state[1STATE_LEN+1];
    char       d_zip[1ZIP_LEN+1];
    char       c_first[1FIRST_NAME_LEN+1];
    char       c_middle[1MIDDLE_NAME_LEN + 1];
    char       c_last[1LAST_NAME_LEN+1];
    char       c_street_1[1ADDRESS_LEN+1];
    char       c_street_2[1ADDRESS_LEN+1];
    char       c_city[1ADDRESS_LEN+1];
    char       c_state[1STATE_LEN+1];
    char       c_zip[1ZIP_LEN+1];
    char       c_phone[1PHONE_LEN+1];

```



```

#ifdef USE_ODBC
    TIMESTAMP_STRUCT    c_since;
#else
    DBDATEREC          c_since;
#endif
char                    c_credit[CREDIT_LEN+1];
double                 c_credit_lim;
double                 c_discount;
double                 c_balance;
char                    c_data[200+1];
long                   num_deadlocks;
int                     retval;
int                     error;
char
execution_status[STATUS_LEN];
} PAYMENT_DATA;

typedef struct
{
    long                ol_i_id;
    short               ol_supply_w_id;
    short               ol_quantity;
    double              ol_amount;
#ifdef USE_ODBC
    TIMESTAMP_STRUCT    ol_delivery_d;
#else
    DBDATEREC          ol_delivery_d;
#endif
} OL_ORDER_STATUS_DATA;

typedef struct
{
    short              w_id;
    short              d_id;
    long               c_id;
    char               c_first[FIRST_NAME_LEN+1];
    char               c_middle[MIDDLE_NAME_LEN+1];
    char               c_last[LAST_NAME_LEN+1];
    double             c_balance;
    long               o_id;
#ifdef USE_ODBC
    TIMESTAMP_STRUCT    o_entry_d;
#else
    DBDATEREC          o_entry_d;
#endif
    short              o_carrier_id;
} OL_ORDER_STATUS_DATA
OIOrderStatusData[MAX_OL_ORDER_STATUS_ITEMS];

```

```

short                o_ol_cnt;
long                num_deadlocks;
int                 retval;
int                 error;
char                execution_status[STATUS_LEN];
} ORDER_STATUS_DATA;

typedef struct
{
    long                o_id;
} DEL_ITEM;

typedef struct
{
    short              w_id;
    short              o_carrier_id;
    SYSTEMTIME         queue_time;
    long               num_deadlocks;
    long                o_id[10];
    int                 retval;
    int                 error;
    char                execution_status[STATUS_LEN];
} DELIVERY_DATA;

typedef struct
{
    short              w_id;
    short              d_id;
    short              thresh_hold;
    long               low_stock;
    long               num_deadlocks;
    int                 retval;
    int                 error;
    char                execution_status[STATUS_LEN];
} STOCK_LEVEL_DATA;

typedef struct
{
    NEW_ORDER_DATA     NewOrderData;
//new order form data
    PAYMENT_DATA       PaymentData;           //payment
form data
    ORDER_STATUS_DATA  OrderStatusData;      //order status form
data
    DELIVERY_DATA      DeliveryData;         //delivery
form data

```

```
form data      STOCK_LEVEL_DATA      StockLevelData;      //stock level
              } TUX_DATA;
#endif
```

Appendix B Server Source Code

The TPC-C database was created using the following Tranact-SQL scripts

B.1 createdb.sql

```

/* TPC-C Benchmark Kit          */
/*                               */
/* CREATEDB.SQL                 */
/*                               */
/* This script is used to create the database */

```

```

use master
go

```

```

if exists ( select name from sysdatabases where name = "tpcc" )
    drop database tpcc

```

```

go

```

```

create database tpcc

```

```

    on tpc_misc1    =    655,
    tpc_misc2     =    655,
    tpc_misc3     =    655,
    tpc_misc4     =    655,
    tpc_misc5     =    880,
    tpc_cs1       =    9952,
    tpc_cs2       =    9952,
    tpc_cs3       =    9952,
    tpc_cs4       =    9952,
    tpc_cs5       =    9952,
    tpc_cs6       =    13240,
    tpc_ol1       =    4549,
    tpc_ol2       =    4549,
    tpc_ol3       =    4549,
    tpc_ol4       =    4549,
    tpc_ol5       =    7000

```

```

    log on v_log_dev = 8699

```

```

go

```

```

alter database tpcc on q_log_dev=8699

```

```

go

```

```

alter database tpcc on y_log_dev=8699

```

```

go

```

```

alter database tpcc on x_log_dev=8699
go

```

```

alter database tpcc on w_log_dev=8699
go

```

```

sp_logdevice tpcc, q_log_dev
go

```

```

sp_logdevice tpcc, y_log_dev
go

```

```

sp_logdevice tpcc, x_log_dev
go

```

```

sp_logdevice tpcc, w_log_dev
go

```

B.2 dbopt1.sql

```

/* TPC-C Benchmark Kit          */
/*                               */
/*                               */
/* Set database options for database load */

```

```

use master
go

```

```

sp_dboption tpcc,'select into/bulkcopy',true
go

```

```

sp_dboption tpcc,'trunc. log on chkpt.',true
go

```

```

use tpcc
go

```

```

checkpoint
go

```

```

use tpcc_admin
go

sp_dboption tpcc,'trunc. log on chkpt.',true
go

```

B.3 dbopt2.sql

```

/* TPC-C Benchmark Kit          */
/*                               */
/*                               */
/* Reset database options after database load */

```

```

use master
go

sp_dboption tpcc,'select ',false
go

sp_dboption tpcc,'trunc. ',false
go

use tpcc
go

checkpoint
go

```

B.4 diskinit.sql

```

/* TPC-C Benchmark Kit          */
/*                               */
/* DISKINIT.SQL                 */
/*                               */
/* This script is used create the database devices */

```

```

use master
go

disk init name = "v_log_dev",

```

```

    physname = "V:",
    vdevno = 10,
    size = 4453888
go

disk init name = "q_log_dev",
    physname = "Q:",
    vdevno = 11,
    size = 4453888
go

disk init name = "y_log_dev",
    physname = "Y:",
    vdevno = 12,
    size = 4453888
go

disk init name = "x_log_dev",
    physname = "X:",
    vdevno = 14,
    size = 4453888
go

disk init name = "w_log_dev",
    physname = "W:",
    vdevno = 31,
    size = 4453888
go

disk init name = "tpc_misc1",
    physname = "G:",
    vdevno = 15,
    size = 335360
go

disk init name = "tpc_misc2",
    physname = "H:",
    vdevno = 16,
    size = 335360
go

disk init name = "tpc_misc3",
    physname = "I:",
    vdevno = 17,
    size = 335360
go

disk init name = "tpc_misc4",

```

```

        physname = "F:",
        vdevno = 18,
        size = 335360
go

disk init name = "tpc_misc5",
        physname = "E:",
        vdevno = 19,
        size = 450560
go

disk init name = "tpc_cs1",
        physname = "L:",
        vdevno = 20,
        size = 5095424
go

disk init name = "tpc_cs2",
        physname = "M:",
        vdevno = 21,
        size = 5095424
go

disk init name = "tpc_cs3",
        physname = "N:",
        vdevno = 22,
        size = 5095424
go

disk init name = "tpc_cs4",
        physname = "O:",
        vdevno = 23,
        size = 5095424
go

disk init name = "tpc_cs5",
        physname = "J:",
        vdevno = 24,
        size = 5095424
go

disk init name = "tpc_cs6",
        physname = "K:",
        vdevno = 25,
        size = 6778880
go

disk init name = "tpc_ol1",

```

```

        physname = "T:",
        vdevno = 26,
        size = 2329088
go

disk init name = "tpc_ol2",
        physname = "U:",
        vdevno = 27,
        size = 2329088
go

disk init name = "tpc_ol3",
        physname = "R:",
        vdevno = 28,
        size = 2329088
go

disk init name = "tpc_ol4",
        physname = "S:",
        vdevno = 29,
        size = 2329088
go

disk init name = "tpc_ol5",
        physname = "P:",
        vdevno = 30,
        size = 3584000
go

```

B.5 idxcuscl.sql

```

/* TPC-C Benchmark Kit */
/*
/* IDXCUSCL.SQL */
/* Creates clustered index on customer (seg) */

use tpcc
go

if exists ( select name from sysindexes where name = 'customer_c1' )
drop index customer.customer_c1
go

```

```

select getdate()
go
create unique clustered index customer_c1 on customer(c_w_id, c_d_id, c_id)
    with sorted_data on big_seg
go
select getdate()
go

```

B.6 idxcusnc.sql

```

/* TPC-C Benchmark Kit          */
/*                               */
/* IDXCUSNC.SQL                 */
/*                               */
/* Creates non-clustered index on customer (seg)          */

```

```

use tpcc
go

```

```

if exists ( select name from sysindexes where name = 'customer_nc1' )
    drop index customer.customer_nc1

```

```
go
```

```

select getdate()
go
create unique nonclustered index customer_nc1 on customer(c_w_id, c_d_id, c_last, c_first,
c_id)
    on big_seg
go
select getdate()
go

```

B.7 idxdiscl.sql

```

/* TPC-C Benchmark Kit          */
/*                               */
/* IDXDISCL.SQL                 */
/*                               */
/* Creates clustered index on district (seg)              */

```

```

use tpcc
go

```

```

if exists ( select name from sysindexes where name = 'district_c1' )
    drop index district.district_c1

```

```
go
```

```

select getdate()
go
create unique clustered index district_c1 on district(d_w_id, d_id)
    with fillfactor=1 on misc_seg

```

```
go
```

```
select getdate()
```

```
go
```

B.8 idxitmcl.sql

```

/* TPC-C Benchmark Kit          */
/*                               */
/* IDXITMCL.SQL                 */
/*                               */
/* Creates clustered index on item (seg)                  */

```

```

use tpcc
go

```

```

if exists ( select name from sysindexes where name = 'item_c1' )
    drop index item.item_c1

```

```
go
```

```

select getdate()
go
create unique clustered index item_c1 on item(i_id)
    with sorted_data on misc_seg

```

```
go
```

```
select getdate()
```

```
go
```

B.9 idxnodcl.sql

```

/* TPC-C Benchmark Kit          */
/*                               */
/* IDXNODCL.SQL                 */
/*                               */

```

```

/* Creates clustered index on new-order (seg)          */
                                                    */

use tpcc
go

if exists ( select name from sysindexes where name = 'new_order_c1' )
    drop index new_order.new_order_c1
go

select getdate()
go
create unique clustered index new_order_c1 on new_order(no_w_id, no_d_id, no_o_id)
    with sorted_data on misc_seg
go
select getdate()
go

```

B.10 idxodlcl.sql

```

/* TPC-C Benchmark Kit                               */
/*                                                    */
/* IDXODLCL.SQL                                     */
/*                                                    */
/* Creates clustered index on order-line (seg)      */
                                                    */

use tpcc
go

if exists ( select name from sysindexes where name = 'order_line_c1' )
    drop index order_line.order_line_c1
go

select getdate()
go
create unique clustered index order_line_c1 on order_line(ol_w_id, ol_d_id, ol_o_id, ol_number)
    with sorted_data on ordln_seg
go
select getdate()
go

```

B.11 idxordcl.sql

```

/* TPC-C Benchmark Kit                               */
/*                                                    */
/* IDXORDCL.SQL                                     */
/*                                                    */
/* Creates clustered index on orders (seg)          */
                                                    */

use tpcc
go

if exists ( select name from sysindexes where name = 'orders_c1' )
    drop index orders.orders_c1
go

select getdate()
go
create unique clustered index orders_c1 on orders(o_w_id, o_d_id, o_id)
    with sorted_data on misc_seg
go
select getdate()
go

```

B.12 idxstkcl.sql

```

/* TPC-C Benchmark Kit                               */
/*                                                    */
/* IDXSTKCL.SQL                                     */
/*                                                    */
/* Creates clustered index on stock (seg)           */
                                                    */

use tpcc
go

if exists ( select name from sysindexes where name = 'stock_c1' )
    drop index stock.stock_c1
go

select getdate()
go
create unique clustered index stock_c1 on stock(s_i_id, s_w_id)
    with sorted_data on big_seg
go

```

```
select getdate()
go
```

B.13 idxwarcl.sql

```
/* TPC-C Benchmark Kit          */
/*                               */
/* IDXWARCL.SQL                 */
/*                               */
/* Creates clustered index on warehouse (seg) */

use tpcc
go

if exists ( select name from sysindexes where name = 'warehouse_c1' )
    drop index warehouse.warehouse_c1
go

select getdate()
go
create unique clustered index warehouse_c1 on warehouse(w_id)
    with fillfactor=1 on misc_seg
go
select getdate()
go
```

B.14 pintable.sql

```
/* TPC-C Benchmark Kit          */
/*                               */
/* This script file is used to 'pin' certain tables in the data cache */

use tpcc
go

exec sp_tableoption "district","pintable",true
exec sp_tableoption "warehouse","pintable",true
exec sp_tableoption "new_order","pintable",true
exec sp_tableoption "item","pintable",true
go
```

B.15 segment.sql

```
/* TPC-C Benchmark Kit          */
/*                               */
/* SEGMENT.SQL                 */
/*                               */
/* This script is used to create the database segments */
```

```
use tpcc
go

exec sp_dropsegment misc_seg
go
exec sp_dropsegment ol_seg
go
exec sp_dropsegment cs_seg
go

sp_addsegment misc_seg, tpc_misc1
go
sp_extendsegment misc_seg, tpc_misc2
go
sp_extendsegment misc_seg, tpc_misc3
go
sp_extendsegment misc_seg, tpc_misc4
go
sp_extendsegment misc_seg, tpc_misc5
go

sp_addsegment cs_seg, tpc_cs1
go
sp_extendsegment cs_seg, tpc_cs2
go
sp_extendsegment cs_seg, tpc_cs3
go
sp_extendsegment cs_seg, tpc_cs4
go
sp_extendsegment cs_seg, tpc_cs5
go
sp_extendsegment cs_seg, tpc_cs6
go

sp_addsegment ol_seg, tpc_ol1
go
sp_extendsegment ol_seg, tpc_ol2
go
sp_extendsegment ol_seg, tpc_ol3
go
sp_extendsegment ol_seg, tpc_ol4
```



```
go
sp_extendsegment ol_seg, tpc_ol5
go
```

B.16 tables.sql

```
/* TPC-C Benchmark Kit          */
/*                               */
/* TABLES.SQL                  */
/*                               */
/* Creates TPC-C tables (seg)   */
/*                               */
```

```
use tpcc
go
```

```
checkpoint
go
```

```
if exists ( select name from sysobjects where name = 'warehouse' )
drop table warehouse
go
```

```
create table warehouse
(
    w_id          smallint,
    w_name        char(10),
    w_street_1    char(20),
    w_street_2    char(20),
    w_city        char(20),
    w_state       char(2),
    w_zip         char(9),
    w_tax         numeric(4,4),
    w_ytd         numeric(12,2)
) on misc_seg
go
```

```
if exists ( select name from sysobjects where name = 'district' )
drop table district
go
```

```
create table district
(
    d_id          tinyint,
```

```
    d_w_id       smallint,
    d_name        char(10),
    d_street_1    char(20),
    d_street_2    char(20),
    d_city        char(20),
    d_state       char(2),
    d_zip         char(9),
    d_tax         numeric(4,4),
    d_ytd         numeric(12,2),
    d_next_o_id   int
) on misc_seg
go
```

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

```
go
```

```
create table customer
(
    c_id          int,
    c_d_id        tinyint,
    c_w_id        smallint,
    c_first       char(16),
    c_middle      char(2),
    c_last        char(16),
    c_street_1    char(20),
    c_street_2    char(20),
    c_city        char(20),
    c_state       char(2),
    c_zip         char(9),
    c_phone       char(16),
    c_since       datetime,
    c_credit      char(2),
    c_credit_lim  numeric(12,2),
    c_discount    numeric(4,4),
    c_balance     numeric(12,2),
    c_ytd_payment numeric(12,2),
    c_payment_cnt smallint,
    c_delivery_cnt smallint,
    c_data_1      char(250),
    c_data_2      char(250)
) on cs_seg
go
```

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

```

go

create table history
(
    h_c_id          int,
    h_c_d_id        tinyint,
    h_c_w_id        smallint,
    h_d_id          tinyint,
    h_w_id          smallint,
    h_date          datetime,
    h_amount        numeric(6,2),
    h_data          char(24)
) on misc_seg
go

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

create table new_order
(
    no_o_id          int,
    no_d_id          tinyint,
    no_w_id          smallint
) on misc_seg
go

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

create table orders
(
    o_id            int,
    o_d_id          tinyint,
    o_w_id          smallint,
    o_c_id          int,
    o_entry_d       datetime,
    o_carrier_id    tinyint,
    o_ol_cnt        tinyint,
    o_all_local     tinyint
) on misc_seg
go

if exists ( select name from sysobjects where name = 'order_line' )

```

```

    drop table order_line
go

create table order_line
(
    ol_o_id         int,
    ol_d_id         tinyint,
    ol_w_id         smallint,
    ol_number       tinyint,
    ol_i_id         int,
    ol_supply_w_id  smallint,
    ol_delivery_d   datetime,
    ol_quantity     smallint,
    ol_amount       numeric(6,2),
    ol_dist_info    char(24)
) on ol_seg
go

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

create table item
(
    i_id            int,
    i_im_id         int,
    i_name          char(24),
    i_price         numeric(5,2),
    i_data          char(50)
) on misc_seg
go

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

create table stock
(
    s_i_id          int,
    s_w_id          smallint,
    s_quantity      smallint,
    s_dist_01       char(24),
    s_dist_02       char(24),
    s_dist_03       char(24),
    s_dist_04       char(24),
    s_dist_05       char(24),

```

```

s_dist_06      char(24),
s_dist_07      char(24),
s_dist_08      char(24),
s_dist_09      char(24),
s_dist_10      char(24),
s_ytd          int,
s_order_cnt    smallint,
s_remote_cnt   smallint,
s_data         char(50)
) on cs_seg
go

```

B.17 tpccbcpl.sql

```

/* TPC-C Benchmark Kit */
/* */
/* This script file sets the table lock option for bulk load */

use tpcc
go

exec sp_tableoption "warehouse","table lock on bulk load",true
exec sp_tableoption "district","table lock on bulk load",true
exec sp_tableoption "stock","table lock on bulk load",true
exec sp_tableoption "item","table lock on bulk load",true
exec sp_tableoption "customer","table lock on bulk load",true
exec sp_tableoption "history","table lock on bulk load",true
exec sp_tableoption "orders","table lock on bulk load",true
exec sp_tableoption "order_line","table lock on bulk load",true
exec sp_tableoption "new_order","table lock on bulk load",true
go

```

B.18 tpccirl.sql

```

/* TPC-C Benchmark Kit */
/* */
/* This script file sets the insert row lock option on selected tables */

use tpcc
go

exec sp_tableoption "history","insert row lock",true
exec sp_tableoption "new_order","insert row lock",true

```

```

exec sp_tableoption "orders","insert row lock",true
exec sp_tableoption "order_line","insert row lock",true
go

```

Stored Procedures

B.19 neword.sql

```

/* File:      NEWORD.SQL */
/* Microsoft TPC-C Kit Ver. 3.00.000 */
/* Audited 08/23/96, By Francois Raab */
/* Copyright Microsoft, 1996 */
/* Purpose:   New-Order transaction for Microsoft TPC-C Benchmark Kit */
/* Author:    Damien Lindauer */
/* damienl@Microsoft.com */

```

```

use tpcc
go

/* new-order transaction stored procedure */

if exists ( select name from sysobjects where name = "tpcc_neworder" )
drop procedure tpcc_neworder
go

/* Modified by rick vicik, 2/4/97 */
/* Combined initialization of local variables into district update statement */
/* Combined 3 huge case select statements into a single one */

```

```

create proc tpcc_neworder

@s_w_id1 smallint = 0, @ol_qty1 smallint = 0,
@s_w_id2 smallint = 0, @ol_qty2 smallint = 0,
@s_w_id3 smallint = 0, @ol_qty3 smallint = 0,
@s_w_id4 smallint = 0, @ol_qty4 smallint = 0,

@w_id      smallint,
@d_id      tinyint,
@c_id      int,
@o_ol_cnt  tinyint,
@o_all_local tinyint,
@i_id1     int = 0,
@i_id2     int = 0,
@i_id3     int = 0,
@i_id4     int = 0,

```

```

@s_w_id5 smallint = 0, @ol_qty5 smallint = 0,
@s_w_id6 smallint = 0, @ol_qty6 smallint = 0,
@s_w_id7 smallint = 0, @ol_qty7 smallint = 0,
@s_w_id8 smallint = 0, @ol_qty8 smallint = 0,
@s_w_id9 smallint = 0, @ol_qty9 smallint = 0,
@s_w_id10 smallint = 0, @ol_qty10 smallint = 0,
@s_w_id11 smallint = 0, @ol_qty11 smallint = 0,
@s_w_id12 smallint = 0, @ol_qty12 smallint = 0,
@s_w_id13 smallint = 0, @ol_qty13 smallint = 0,
@s_w_id14 smallint = 0, @ol_qty14 smallint = 0,
@s_w_id15 smallint = 0, @ol_qty15 smallint = 0

as
declare @w_tax      numeric(4,4),
        @d_tax      numeric(4,4),
        @c_last     char(16),
        @c_credit   char(2),
        @c_discount numeric(4,4),
        @i_price    numeric(5,2),
        @i_name     char(24),
        @i_data     char(50),
        @o_entry_d  datetime,
        @remote_flag int,
        @s_quantity smallint,
        @s_data     char(50),
        @s_dist     char(24),
        @li_no      int,
        @o_id       int,
        @commit_flag int,
        @li_id      int,
        @li_s_w_id  smallint,
        @li_qty     smallint,
        @ol_number  int,
        @c_id_local int

begin
        @i_id5 int = 0,
        @i_id6 int = 0,
        @i_id7 int = 0,
        @i_id8 int = 0,
        @i_id9 int = 0,
        @i_id10 int = 0,
        @i_id11 int = 0,
        @i_id12 int = 0,
        @i_id13 int = 0,
        @i_id14 int = 0,
        @i_id15 int = 0,

begin transaction n
/* get district tax and next available order id and update */
/* plus initialize local variables */

update district
    set @d_tax = d_tax,
        @o_id = d_next_o_id,
            d_next_o_id = d_next_o_id + 1,
        @o_entry_d = getdate(),
        @li_no=0,
        @commit_flag = 1
    where d_w_id = @w_id and
          d_id = @d_id

/* process orderlines */
while (@li_no < @o_ol_cnt)
begin

    select @li_no = @li_no + 1

/* Set i_id, s_w_id, and qty for this lineitem */

    select @li_id = case @li_no
        when 1 then @i_id1
        when 2 then @i_id2
        when 3 then @i_id3
        when 4 then @i_id4
        when 5 then @i_id5
        when 6 then @i_id6
        when 7 then @i_id7
        when 8 then @i_id8
        when 9 then @i_id9
        when 10 then @i_id10
        when 11 then @i_id11
        when 12 then @i_id12
        when 13 then @i_id13
        when 14 then @i_id14
        when 15 then @i_id15
    end,

        @li_s_w_id = case @li_no
        when 1 then @s_w_id1
        when 2 then @s_w_id2
        when 3 then @s_w_id3
        when 4 then @s_w_id4
        when 5 then @s_w_id5

```

```

when 6 then @s_w_id6
when 7 then @s_w_id7
when 8 then @s_w_id8
when 9 then @s_w_id9
when 10 then @s_w_id10
when 11 then @s_w_id11
when 12 then @s_w_id12
when 13 then @s_w_id13
when 14 then @s_w_id14
when 15 then @s_w_id15
end,

@li_qty = case @li_no
when 1 then @ol_qty1
when 2 then @ol_qty2
when 3 then @ol_qty3
when 4 then @ol_qty4
when 5 then @ol_qty5
when 6 then @ol_qty6
when 7 then @ol_qty7
when 8 then @ol_qty8
when 9 then @ol_qty9
when 10 then @ol_qty10
when 11 then @ol_qty11
when 12 then @ol_qty12
when 13 then @ol_qty13
when 14 then @ol_qty14
when 15 then @ol_qty15
end

/* get item data (no one updates item) */

select @i_price = i_price,
       @i_name = i_name,
       @i_data = i_data
from item (tablock holdlock)
where i_id = @li_id

/* if there actually is an item with this id, go to work */

if (@@rowcount > 0)
begin
update stock set s_ytd      = s_ytd + @li_qty,
               @s_quantity = s_quantity,
               s_quantity  = s_quantity - @li_qty +
               case when (s_quantity - @li_qty < 10) then 91 else 0 end,
               s_order_cnt = s_order_cnt + 1,
               s_remote_cnt = s_remote_cnt + case

```

```

when (@li_s_w_id = @w_id) then 0 else 1 end,
@s_data = s_data,
@s_dist = case @d_id
when 1 then s_dist_01
when 2 then s_dist_02
when 3 then s_dist_03
when 4 then s_dist_04
when 5 then s_dist_05
when 6 then s_dist_06
when 7 then s_dist_07
when 8 then s_dist_08
when 9 then s_dist_09
when 10 then s_dist_10
end
where s_i_id = @li_id and
       s_w_id = @li_s_w_id

/* insert order_line data (using data from item and stock) */

insert into order_line values(@o_id,      /* from district update */
                              @d_id,      /* input param      */
                              @w_id,      /* input param      */
                              @li_no,     /* orderline number */
                              @li_id,     /* lineitem id      */
                              @li_s_w_id, /* lineitem warehouse */
                              "jan 1, 1900", /* constant        */
                              @li_qty,    /* lineitem qty     */
                              @i_price * @li_qty, /* ol_amount      */
                              @s_dist)    /* from stock       */

/* send line-item data to client */

select @i_name,
       @s_quantity,
       b_g = case when (patindex("%ORIGINAL%",@i_data) > 0) and
                    (patindex("%ORIGINAL%",@s_data) > 0) )
       then "B" else "G" end,
       @i_price,
       @i_price * @li_qty

end
else
begin

/* no item found - triggers rollback condition */

select "",0,"",0,0
select @commit_flag = 0

```

```

        end
    end

    /* get customer last name, discount, and credit rating */

    select @c_last = c_last,
           @c_discount = c_discount,
           @c_credit = c_credit,
           @c_id_local = c_id
    from customer holdlock
    where c_id = @c_id and
           c_w_id = @w_id and
           c_d_id = @d_id

    /* insert fresh row into orders table */

    insert into orders values (@o_id,
                              @d_id,
                              @w_id,
                              @c_id_local,
                              @o_entry_d,
                              0,
                              @o_ol_cnt,
                              @o_all_local)

    /* insert corresponding row into new-order table */

    insert into new_order values (@o_id,
                                  @d_id,
                                  @w_id)

    /* select warehouse tax */

    select @w_tax = w_tax
    from warehouse holdlock
    where w_id = @w_id

    if (@commit_flag = 1)
        commit transaction n
    else
        /* all that work for nuthin!!! */
        rollback transaction n

    /* return order data to client */
    select @w_tax,
           @d_tax,
           @o_id,

```

```

@c_last,
@c_discount,
@c_credit,
@o_entry_d,
@commit_flag

```

```
end
```

```
go
```

B.20 ordstat.sql

```

/* File:   ORDSTAT.SQL                               */
/* Microsoft TPC-C Kit Ver. 3.00.000                 */
/* Audited 08/23/96, By Francois Raab                */
/* Copyright Microsoft, 1996                          */
/* Purpose: Order-Status transaction for Microsoft TPC-C Benchmark Kit */
/* Author:  Damien Lindauer                          */
/* damienl@Microsoft.com                             */

use tpcc
go

if exists ( select name from sysobjects where name = "tpcc_orderstatus" )
    drop procedure tpcc_orderstatus
go

/* Modified by rick vicik, 2/4/97 */
/* Eliminated @val local variable */

create proc tpcc_orderstatus @w_id          smallint,
                              @d_id        int,
                              @c_id        int,
                              @c_last char(16) =
''

as

declare @c_balance    numeric(12,2),
        @c_first      char(16),
        @c_middle     char(2),
        @o_id         int,

```

```

@o_entry_d      datetime,
@o_carrier_id  smallint,
@cnt           smallint

begin tran o

  if (@c_id = 0)
  begin
    /* get customer id and info using last name */

    select @cnt = (count(*)+1)/2
    from customer holdlock
    where c_last= @c_last and
           c_w_id = @w_id and
    c_d_id = @d_id
    set rowcount @cnt

    select @c_id = c_id,
           @c_balance = c_balance,
           @c_first  = c_first,
           @c_last   = c_last,
           @c_middle = c_middle
    from customer hotblock
    where c_last = @c_last and
           c_w_id = @w_id and
    c_d_id = @d_id
    order by c_w_id, c_d_id, c_last, c_first

    set rowcount 0
  end

  else
  begin

    /* get customer info if by id*/

    select @c_balance = c_balance,
           @c_first  = c_first,
           @c_middle = c_middle,
           @c_last   = c_last
    from customer holdlock
    where c_id = @c_id and
           c_d_id = @d_id and
           c_w_id = @w_id

    select @cnt = @@rowcount

  end

```

```

/* if no such customer */
if (@cnt = 0)
begin
  raiserror("Customer not found",18,1)
  goto custnotfound
end

/* get order info */

select @o_id = o_id,
       @o_entry_d = o_entry_d,
       @o_carrier_id = o_carrier_id
from orders hotblock
where o_w_id = @w_id and
      o_d_id = @d_id and
      o_c_id = @c_id

/* select order lines for the current order */

select ol_supply_w_id,
       ol_i_id,
       ol_quantity,
       ol_amount,
       ol_delivery_d
from order_line holdlock
where ol_o_id = @o_id and
      ol_d_id = @d_id and
      ol_w_id = @w_id

custnotfound:

commit tran o

/* return data to client */

select @c_id,
       @c_last,
       @c_first,
       @c_middle,
       @o_entry_d,
       @o_carrier_id,
       @c_balance,
       @o_id

go

```

B.21 payment.sql

```

/* File:      PAYMENT.SQL                */
/*
/* Microsoft TPC-C Kit Ver. 3.00.000    */
/* Audited 08/23/96, By Francois Raab   */
/*                                         */
/* Copyright Microsoft, 1996            */
/*                                         */
/* Purpose:   Payment transaction for Microsoft TPC-C Benchmark Kit */
/* Author:    Damien Lindauer           */
/*           damienl@Microsoft.com      */

use tpcc
go

if exists (select name from sysobjects where name = "tpcc_payment")
    drop procedure tpcc_payment
go

create proc tpcc_payment @w_id          smallint,
                        @c_w_id        smallint,
                        @h_amount       numeric(6,2),
                        @d_id           tinyint,
                        @c_d_id         tinyint,
                        @c_id           int,
                        @c_last         char(16) = ""

as
declare @w_street_1 char(20),
        @w_street_2 char(20),
        @w_city     char(20),
        @w_state    char(2),
        @w_zip      char(9),
        @w_name     char(10),
        @d_street_1 char(20),
        @d_street_2 char(20),
        @d_city     char(20),
        @d_state    char(2),
        @d_zip      char(9),
        @d_name     char(10),
        @c_first    char(16),
        @c_middle   char(2),
        @c_street_1 char(20),
        @c_street_2 char(20),
        @c_city     char(20),
        @c_state    char(2),

        @c_zip      char(9),
        @c_phone    char(16),
        @c_since    datetime,
        @c_credit   char(2),
        @c_credit_lim numeric(12,2),
        @c_balance  numeric(12,2),
        @c_discount numeric(4,4),
        @data1     char(250),
        @data2     char(250),
        @c_data_1  char(250),
        @c_data_2  char(250),
        @datetime  datetime,
        @w_ytd     numeric(12,2),
        @d_ytd     numeric(12,2),
        @cnt       smallint,
        @val       smallint,
        @screen_data char(200),
        @d_id_local tinyint,
        @w_id_local smallint,
        @c_id_local int

select @screen_data = ""

begin tran p

/* get payment date */

select @datetime = getdate()

if (@c_id = 0)
begin
/* get customer id and info using last name */

select @cnt = count(*)
from customer holdlock
where c_last = @c_last and
c_w_id = @c_w_id and
c_d_id = @c_d_id

select @val = (@cnt + 1) / 2
set rowcount @val

select @c_id = c_id
from customer holdlock
where c_last = @c_last and
c_w_id = @c_w_id and
c_d_id = @c_d_id
order by c_w_id, c_d_id, c_last, c_first

```



```

        set rowcount 0
    end

/* get customer info and update balances */

update customer set
    @c_balance = c_balance - @h_amount,
    c_payment_cnt = c_payment_cnt + 1,
    c_ytd_payment = c_ytd_payment + @h_amount,
    @c_first = c_first,
    @c_middle = c_middle,
    @c_last = c_last,
    @c_street_1 = c_street_1,
    @c_street_2 = c_street_2,
    @c_city = c_city,
    @c_state = c_state,
    @c_zip = c_zip,
    @c_phone = c_phone,
    @c_credit = c_credit,
    @c_credit_lim = c_credit_lim,
    @c_discount = c_discount,
    @c_since = c_since,
    @data1 = c_data_1,
    @data2 = c_data_2,
    @c_id_local = c_id
where c_id = @c_id and
    c_w_id = @c_w_id and
    c_d_id = @c_d_id

/* if customer has bad credit get some more info */

if (@c_credit = "BC")
begin

    /* compute new info */

    select @c_data_2 = substring(@data1,209,42) +
        substring(@data2, 1, 208)
    select @c_data_1 = convert(char(5),@c_id) +
        convert(char(4),@c_d_id) +
        convert(char(5),@c_w_id) +
        convert(char(4),@d_id) +
        convert(char(5),@w_id) +
        convert(char(19),@h_amount) +
        substring(@data1, 1, 208)

    /* update customer info */

```

```

        update customer set
            c_data_1 = @c_data_1,
            c_data_2 = @c_data_2
        where c_id = @c_id and
            c_w_id = @c_w_id and
            c_d_id = @c_d_id

        select @screen_data = substring (@c_data_1,1,200)
    end

/* get district data and update year-to-date */

update district
    set d_ytd = d_ytd + @h_amount,
        @d_street_1 = d_street_1,
        @d_street_2 = d_street_2,
        @d_city = d_city,
        @d_state = d_state,
        @d_zip = d_zip,
        @d_name = d_name,
        @d_id_local = d_id
    where d_w_id = @w_id and
        d_id = @d_id

/* get warehouse data and update year-to-date */

update warehouse
    set w_ytd = w_ytd + @h_amount,
        @w_street_1 = w_street_1,
        @w_street_2 = w_street_2,
        @w_city = w_city,
        @w_state = w_state,
        @w_zip = w_zip,
        @w_name = w_name,
        @w_id_local = w_id
    where w_id = @w_id

/* create history record */

insert into history values (@c_id_local,
                            @c_d_id,
                            @c_w_id,
                            @d_id_local,

```

```

@w_id_local,

@h_amount,

+ " " + @d_name)

commit tran p

/* return data to client */

select @c_id,
        @c_last,
        @datetime,
        @w_street_1,
        @w_street_2,
        @w_city,
        @w_state,
        @w_zip,
        @d_street_1,
        @d_street_2,
        @d_city,
        @d_state,
        @d_zip,
        @c_first,
        @c_middle,
        @c_street_1,
        @c_street_2,
        @c_city,
        @c_state,
        @c_zip,
        @c_phone,
        @c_since,
        @c_credit,
        @c_credit_lim,
        @c_discount,
        @c_balance,
        @screen_data

go

```

B.22 delivery.sql

```

/* File:    DELIVERY.SQL                */
/* Microsoft TPC-C Kit Ver. 3.00.000    */
/* Audited 08/23/96, By Francois Raab   */
/*                                          */

```

```

/* Copyright Microsoft, 1996            */
/*                                     */
/* Purpose:  Delivery transaction for Microsoft TPC-C Benchmark Kit  */
/* Author:   Damien Lindauer           */
/*          damienl@Microsoft.com      */
/*                                     */

use tpcc
go

/* delivery transaction */

if exists (select name from sysobjects where name = "tpcc_delivery" )
    drop procedure tpcc_delivery
go

create proc tpcc_delivery @w_id smallint,
                          @o_carrier_id smallint
as

declare @d_id tinyint,
        @o_id int,
        @c_id int,
        @total numeric(12,2),
        @oid1 int,
        @oid2 int,
        @oid3 int,
        @oid4 int,
        @oid5 int,
        @oid6 int,
        @oid7 int,
        @oid8 int,
        @oid9 int,
        @oid10 int

select @d_id = 0

begin tran d

    while (@d_id < 10)
    begin

        select @d_id = @d_id + 1,
               @total = 0,
               @o_id = 0

        select @o_id = min(no_o_id)

```

```

from new_order holdlock
where no_w_id = @w_id and
      no_d_id = @d_id

if (@@rowcount <> 0)
begin

  /* claim the order for this district */

  delete new_order
  where no_w_id = @w_id and
        no_d_id = @d_id and
        no_o_id = @o_id

  /* set carrier_id on this order (and get customer id) */

  update orders
    set o_carrier_id = @o_carrier_id,
        @c_id       = o_c_id
  where o_w_id = @w_id and
        o_d_id = @d_id and
        o_id   = @o_id

  /* set date in all lineitems for this order (and sum amounts) */

  update order_line
    set ol_delivery_d = getdate(),
        @total       = @total + ol_amount
  where ol_w_id = @w_id and
        ol_d_id = @d_id and
        ol_o_id = @o_id

  /* accumulate lineitem amounts for this order into customer */

  update customer
    set c_balance   = c_balance + @total,
        c_delivery_cnt = c_delivery_cnt + 1
  where c_w_id = @w_id and
        c_d_id = @d_id and
        c_id   = @c_id

end

select @oid1 = case @d_id when 1 then @o_id else @oid1 end,
       @oid2 = case @d_id when 2 then @o_id else @oid2 end,
       @oid3 = case @d_id when 3 then @o_id else @oid3 end,
       @oid4 = case @d_id when 4 then @o_id else @oid4 end,
       @oid5 = case @d_id when 5 then @o_id else @oid5 end,

```

```

       @oid6 = case @d_id when 6 then @o_id else @oid6 end,
       @oid7 = case @d_id when 7 then @o_id else @oid7 end,
       @oid8 = case @d_id when 8 then @o_id else @oid8 end,
       @oid9 = case @d_id when 9 then @o_id else @oid9 end,
       @oid10 = case @d_id when 10 then @o_id else @oid10 end

end

commit tran d

select @oid1,
       @oid2,
       @oid3,
       @oid4,
       @oid5,
       @oid6,
       @oid7,
       @oid8,
       @oid9,
       @oid10

go

```

B.23 stocklev.sql

```

/* File:      STOCKLEV.SQL                               */
/* Microsoft TPC-C Kit Ver. 3.00.000                     */
/* Audited 08/23/96, By Francois Raab                   */
/*                                                    */
/* Copyright Microsoft, 1996                             */
/*                                                    */
/* Purpose:   Stock-Level transaction for Microsoft TPC-C Benchmark Kit */
/* Author:    Damien Lindauer                             */
/* damienl@Microsoft.com                                */

use tpcc
go

/* stock-level transaction stored procedure */

if exists (select name from sysobjects where name = "tpcc_stocklevel")
  drop procedure tpcc_stocklevel

go

/* Modified by rick vicik, 2/4/97 */

```

```
/* Eliminate 1 local variable, use derived table to eliminate duplicate item#'s */
```

```
create proc tpcc_stocklevel @w_id          smallint,
                           @d_id          tinyint,
                           @threshold     smallint
as
declare @o_id int

select @o_id = d_next_o_id
from district
where d_w_id = @w_id and
      d_id   = @d_id

select count(*) from stock,
      (select distinct(ol_i_id) from order_line
       where ol_w_id = @w_id and
             ol_d_id = @d_id and
             ol_o_id between (@o_id-20) and (@o_id-1)) OL

where s_w_id = @w_id and
      s_i_id = OL.ol_i_id and
      s_quantity < @threshold
go
```

The TPC-C database was loaded using the LOAD.EXE executable, the source code is listed below.

B.24 makefile

```
!include $(TPC_DIR)\build\ntintel\tpc.inc

CUR_DIR = $(TPC_DIR)\src

CLIENT_EXE = $(EXE_DIR)\client.exe
MASTER_EXE = $(EXE_DIR)\master.exe
TPCCCLR_EXE = $(EXE_DIR)\tpccldr.exe
DELIVERY_EXE = $(EXE_DIR)\delivery.exe
sqlstat_EXE = $(EXE_DIR)\sqlstat.exe

all : $(CLIENT_EXE) $(MASTER_EXE) $(TPCCCLR_EXE) $(DELIVERY_EXE) $(sqlstat_EXE)

$(OBJ_DIR)\client.obj : $(CUR_DIR)\client.c $(INC_DIR)\tpcc.h
$(CC) $(CFLAGS) /Fo$(OBJ_DIR)\client.obj $(CUR_DIR)\client.c
```

```
$(OBJ_DIR)\master.obj : $(CUR_DIR)\master.c $(INC_DIR)\tpcc.h
$(CC) $(CFLAGS) /Fo$(OBJ_DIR)\master.obj $(CUR_DIR)\master.c
```

```
$(OBJ_DIR)\tpccldr.obj : $(CUR_DIR)\tpccldr.c $(INC_DIR)\tpcc.h
$(CC) $(CFLAGS) /Fo$(OBJ_DIR)\tpccldr.obj $(CUR_DIR)\tpccldr.c
```

```
$(OBJ_DIR)\stats.obj : $(CUR_DIR)\stats.c $(INC_DIR)\tpcc.h
$(CC) $(CFLAGS) /Fo$(OBJ_DIR)\stats.obj $(CUR_DIR)\stats.c
```

```
$(OBJ_DIR)\getargs.obj : $(CUR_DIR)\getargs.c $(INC_DIR)\tpcc.h
$(CC) $(CFLAGS) /Fo$(OBJ_DIR)\getargs.obj $(CUR_DIR)\getargs.c
```

```
$(OBJ_DIR)\util.obj : $(CUR_DIR)\util.c $(INC_DIR)\tpcc.h
$(CC) $(CFLAGS) /Fo$(OBJ_DIR)\util.obj $(CUR_DIR)\util.c
```

```
$(OBJ_DIR)\time.obj : $(CUR_DIR)\time.c $(INC_DIR)\tpcc.h
$(CC) $(CFLAGS) /Fo$(OBJ_DIR)\time.obj $(CUR_DIR)\time.c
```

```
$(OBJ_DIR)\random.obj : $(CUR_DIR)\random.c $(INC_DIR)\tpcc.h
$(CC) $(CFLAGS) /Fo$(OBJ_DIR)\random.obj $(CUR_DIR)\random.c
```

```
$(OBJ_DIR)\strings.obj : $(CUR_DIR)\strings.c $(INC_DIR)\tpcc.h
$(CC) $(CFLAGS) /Fo$(OBJ_DIR)\strings.obj $(CUR_DIR)\strings.c
```

```
$(OBJ_DIR)\sqlfuncs.obj : $(CUR_DIR)\sqlfuncs.c $(INC_DIR)\tpcc.h
$(CC) $(CFLAGS) /Fo$(OBJ_DIR)\sqlfuncs.obj $(CUR_DIR)\sqlfuncs.c
```

```
$(OBJ_DIR)\tran.obj : $(CUR_DIR)\tran.c $(INC_DIR)\tpcc.h
$(CC) $(CFLAGS) /Fo$(OBJ_DIR)\tran.obj $(CUR_DIR)\tran.c
```

```
$(OBJ_DIR)\data.obj : $(CUR_DIR)\data.c $(INC_DIR)\tpcc.h
$(CC) $(CFLAGS) /Fo$(OBJ_DIR)\data.obj $(CUR_DIR)\data.c
```

```
$(OBJ_DIR)\delivery.obj : $(CUR_DIR)\delivery.c $(INC_DIR)\tpcc.h
$(CC) $(CFLAGS) /Fo$(OBJ_DIR)\delivery.obj $(CUR_DIR)\delivery.c
```

```
$(OBJ_DIR)\sqlstat.obj : $(CUR_DIR)\sqlstat.c $(INC_DIR)\tpcc.h
$(CC) $(CFLAGS) /Fo$(OBJ_DIR)\sqlstat.obj $(CUR_DIR)\sqlstat.c
```

```
$(EXE_DIR)\client.exe : $(OBJ_DIR)\client.obj $(OBJ_DIR)\tran.obj $(OBJ_DIR)\sqlfuncs.obj
$(OBJ_DIR)\random.obj $(OBJ_DIR)\util.obj $(OBJ_DIR)\data.obj $(OBJ_DIR)\getargs.obj
$(OBJ_DIR)\time.obj $(OBJ_DIR)\stats.obj $(OBJ_DIR)\strings.obj
$(LL) -entry:mainCRTStartup -out:$(EXE_DIR)\client.exe \
$(OBJ_DIR)\client.obj $(OBJ_DIR)\tran.obj $(OBJ_DIR)\sqlfuncs.obj \
$(OBJ_DIR)\random.obj $(OBJ_DIR)\util.obj $(OBJ_DIR)\data.obj \
$(OBJ_DIR)\getargs.obj $(OBJ_DIR)\time.obj $(OBJ_DIR)\stats.obj \
$(OBJ_DIR)\strings.obj \
```

```

$(DB_LIB)\ntwdblib.lib $(NTLIBS)

$(EXE_DIR)\master.exe : $(OBJ_DIR)\master.obj $(OBJ_DIR)\sqlfuncs.obj $(OBJ_DIR)\util.obj
$(OBJ_DIR)\getargs.obj $(OBJ_DIR)\time.obj $(OBJ_DIR)\stats.obj
$(LL) -entry:mainCRTStartup -out:$(EXE_DIR)\master.exe \
$(OBJ_DIR)\master.obj $(OBJ_DIR)\sqlfuncs.obj $(OBJ_DIR)\util.obj \
$(OBJ_DIR)\getargs.obj $(OBJ_DIR)\time.obj $(OBJ_DIR)\stats.obj \
$(DB_LIB)\ntwdblib.lib $(NTLIBS)

$(EXE_DIR)\tpccldr.exe : $(OBJ_DIR)\tpccldr.obj $(OBJ_DIR)\getargs.obj $(OBJ_DIR)\util.obj
$(OBJ_DIR)\time.obj $(OBJ_DIR)\random.obj $(OBJ_DIR)\strings.obj
$(LL) -entry:mainCRTStartup -out:$(EXE_DIR)\tpccldr.exe \
$(OBJ_DIR)\tpccldr.obj $(OBJ_DIR)\getargs.obj $(OBJ_DIR)\strings.obj \
$(OBJ_DIR)\util.obj $(OBJ_DIR)\time.obj $(OBJ_DIR)\random.obj \
$(DB_LIB)\ntwdblib.lib $(NTLIBS)

$(EXE_DIR)\delivery.exe : $(OBJ_DIR)\delivery.obj $(OBJ_DIR)\sqlfuncs.obj
$(OBJ_DIR)\util.obj $(OBJ_DIR)\getargs.obj $(OBJ_DIR)\time.obj $(OBJ_DIR)\stats.obj
$(LL) -entry:mainCRTStartup -out:$(EXE_DIR)\delivery.exe \
$(OBJ_DIR)\delivery.obj $(OBJ_DIR)\sqlfuncs.obj $(OBJ_DIR)\util.obj \
$(OBJ_DIR)\getargs.obj $(OBJ_DIR)\time.obj $(OBJ_DIR)\stats.obj \
$(DB_LIB)\ntwdblib.lib $(NTLIBS)

$(EXE_DIR)\sqlstat.exe : $(OBJ_DIR)\sqlstat.obj $(OBJ_DIR)\sqlfuncs.obj $(OBJ_DIR)\util.obj
$(OBJ_DIR)\getargs.obj $(OBJ_DIR)\time.obj $(OBJ_DIR)\stats.obj
$(LL) -entry:mainCRTStartup -out:$(EXE_DIR)\sqlstat.exe \
$(OBJ_DIR)\sqlstat.obj $(OBJ_DIR)\sqlfuncs.obj $(OBJ_DIR)\util.obj \
$(OBJ_DIR)\getargs.obj $(OBJ_DIR)\time.obj $(OBJ_DIR)\stats.obj \
$(DB_LIB)\ntwdblib.lib $(NTLIBS)

```

Loader Source

B.25 tpccldr.c

```

/* FILE: TPCCLDR.C
* Microsoft TPC-C Kit Ver. 3.00.000
* Audited 08/23/96, By Francois Raab
*
* Copyright Microsoft, 1996
*
* PURPOSE: Database loader for Microsoft TPC-C Benchmark Kit
* Author: Damien Lindauer
* damienl@Microsoft.com
*/

```

```

// Includes
#include "tpcc.h"
#include "search.h"

// Defines
#define MAXITEMS 10000
#define CUSTOMERS_PER_DISTRICT 3000
#define DISTRICT_PER_WAREHOUSE 10
#define ORDERS_PER_DISTRICT 3000
#define MAX_CUSTOMER_THREADS 2
#define MAX_ORDER_THREADS 3
#define MAX_MAIN_THREADS 4

// Functions declarations
long NURand();
void LoadItem();
void LoadWarehouse();

void Stock();
void District();

void LoadCustomer();
void CustomerBufInit();
void CustomerBufLoad();
void LoadCustomerTable();
void LoadHistoryTable();

void LoadOrders();
void OrdersBufInit();
void OrdersBufLoad();
void LoadOrdersTable();
void LoadNewOrderTable();
void LoadOrderLineTable();
void GetPermutation();
void CheckForCommit();
void OpenConnections();

void BuildIndex();

void CurrentDate();

// Shared memory structures
typedef struct
{
    long ol;
    long ol_i_id;
    short ol_supply_w_id;
}

```

```

short      ol_quantity;
double     ol_amount;
char       ol_dist_info[DIST_INFO_LEN+1];
           // Added to insure ol_delivery_d set properly during load
char       ol_delivery_d[30];
} ORDER_LINE_STRUCT;

typedef struct
{
    long      o_id;
    short     o_d_id;
    short     o_w_id;
    long      o_c_id;
    short     o_carrier_id;
    short     o_ol_cnt;
    short     o_all_local;
    ORDER_LINE_STRUCT  o_ol[15];
} ORDERS_STRUCT;

typedef struct
{
    long      c_id;
    short     c_d_id;
    short     c_w_id;
    char      c_first[FIRST_NAME_LEN+1];
    char      c_middle[MIDDLE_NAME_LEN+1];
    char      c_last[LAST_NAME_LEN+1];
    char      c_street_1[ADDRESS_LEN+1];
    char      c_street_2[ADDRESS_LEN+1];
    char      c_city[ADDRESS_LEN+1];
    char      c_state[STATE_LEN+1];
    char      c_zip[ZIP_LEN+1];
    char      c_phone[PHONE_LEN+1];
    char      c_credit[CREDIT_LEN+1];
    double    c_credit_lim;
    double    c_discount;
    double    c_balance;
    double    c_ytd_payment;
    short     c_payment_cnt;
    short     c_delivery_cnt;
    char      c_data_1[C_DATA_LEN+1];
    char      c_data_2[C_DATA_LEN+1];
    double    h_amount;
    char      h_data[H_DATA_LEN+1];
} CUSTOMER_STRUCT;

typedef struct
{

```

```

char       c_last[LAST_NAME_LEN+1];
char       c_first[FIRST_NAME_LEN+1];
long      c_id;
} CUSTOMER_SORT_STRUCT;

typedef struct
{
    long      time_start;
} LOADER_TIME_STRUCT;

// Global variables
char      errfile[20];
DBPROCESS *i_dbproc1;
DBPROCESS *w_dbproc1, *w_dbproc2;
DBPROCESS *c_dbproc1, *c_dbproc2;
DBPROCESS *o_dbproc1, *o_dbproc2, *o_dbproc3;
ORDERS_STRUCT  orders_buf[ORDERS_PER_DISTRICT];
CUSTOMER_STRUCT customer_buf[CUSTOMERS_PER_DISTRICT];
long      main_threads_completed;
long      customer_threads_completed;
long      order_threads_completed;
long      orders_rows_loaded;
long      new_order_rows_loaded;
long      order_line_rows_loaded;
long      history_rows_loaded;
long      customer_rows_loaded;
long      stock_rows_loaded;
long      district_rows_loaded;
long      item_rows_loaded;
long      warehouse_rows_loaded;
long      main_time_start;
long      main_time_end;
TPCC_LDR_ARGS *aptr, args;

//=====
//
// Function name: main
//
//=====

int main(int argc, char **argv)
{
    DWORD    dwThreadID[MAX_MAIN_THREADS];
    HANDLE   hThread[MAX_MAIN_THREADS];
    FILE     *fLoader;

```



```

        if (hThread[1] == NULL)
        {
            printf("Error, failed in creating creating thread = 1.\n");
            exit(-1);
        }
        main_threads_started++;
    }
    if ((aptr->table == NULL) || !(strcmp(aptr->table,"customer")))
    {
        fprintf(fLoader, "Starting bader threads for: customer\n");
        hThread[2] = CreateThread(NULL,
                                0,
                                (LPTHREAD_START_ROUTINE) LoadCustomer,
                                NULL,
                                0,
                                &dwThreadID[2]);
        if (hThread[2] == NULL)
        {
            printf("Error, failed in creating creating main thread = 2.\n");
            exit(-1);
        }
        main_threads_started++;
    }
    if ((aptr->table == NULL) || !(strcmp(aptr->table,"orders")))
    {
        fprintf(fLoader, "Starting loader threads for: orders\n");
        hThread[3] = CreateThread(NULL,
                                0,
                                (LPTHREAD_START_ROUTINE) LoadOrders,
                                NULL,
                                0,
                                &dwThreadID[3]);
        if (hThread[3] == NULL)
        {
            printf("Error, failed in creating creating main thread = 3.\n");
            exit(-1);

```

```

        }
        main_threads_started++;
    }
    while (main_threads_completed != main_threads_started)
        Sleep(1000L);
    main_time_end = (TimeNow() / MILLI);
    sprintf(buffer, "\nTPC-C load completed successfully in %ld minutes.\n",
            (main_time_end - main_time_start)/60);
    printf("%s",buffer);
    fprintf(fLoader, "%s", buffer);
    fclose(fLoader);
    dbexit();
    exit(0);
}
//=====
//
// Function name: LoadItem
//
//=====
void LoadItem()
{
    long i_id;
    long i_im_id;
    char i_name[I_NAME_LEN+1];
    double i_price;
    char i_data[I_DATA_LEN+1];
    char name[20];
    long time_start;

    printf("\nLoading item table...\n");

    // Seed with unique number
    seed(1);

    InitString(i_name, I_NAME_LEN+1);
    InitString(i_data, I_DATA_LEN+1);

```



```

sprintf(name, "%s..%s", aptr->database, "item");
bcp_init(i_dbproc1, name, NULL, "logs\\item.err", DB_IN);

bcp_bind(i_dbproc1, (BYTE *) &i_id, 0, -1, NULL, 0, 0, 1);
bcp_bind(i_dbproc1, (BYTE *) &i_im_id, 0, -1, NULL, 0, 0, 2);
bcp_bind(i_dbproc1, (BYTE *) i_name, 0, I_NAME_LEN, NULL, 0, 0, 3);
bcp_bind(i_dbproc1, (BYTE *) &i_price, 0, -1, NULL, 0, SQLFLT8, 4);
bcp_bind(i_dbproc1, (BYTE *) i_data, 0, I_DATA_LEN, NULL, 0, 0, 5);

time_start = (TimeNow() / MILLI);

item_rows_loaded = 0;

for (i_id = 1; i_id <= MAXITEMS; i_id++)
{
    i_im_id = RandomNumber(1L, 10000L);

    MakeAlphaString(14, 24, I_NAME_LEN, i_name);

    i_price = ((float) RandomNumber(100L, 10000L))/100.0;

    MakeOriginalAlphaString(26, 50, I_DATA_LEN, i_data, 10);

    if (!bcp_sendrow(i_dbproc1))
        printf("Error, LoadItem() failed calling bcp_sendrow(). Check
error file.\n");

    item_rows_loaded++;
    CheckForCommit(i_dbproc1, item_rows_loaded, "item", &time_start);
}

bcp_done(i_dbproc1);
dbclose(i_dbproc1);

printf("Finished loading item table.\n");

if (aptr->build_index == 1)
    BuildIndex("idxitmcl");

InterlockedIncrement(&main_threads_completed);
}

//=====
//
// Function : LoadWarehouse
//
// Loads WAREHOUSE table and loads Stock and District as Warehouses are created

```

```

//
//=====

void LoadWarehouse()
{
    short w_id;
    char w_name[W_NAME_LEN+1];
    char w_street_1[ADDRESS_LEN+1];
    char w_street_2[ADDRESS_LEN+1];
    char w_city[ADDRESS_LEN+1];
    char w_state[STATE_LEN+1];
    char w_zip[ZIP_LEN+1];
    double w_tax;
    double w_ytd;
    char name[20];
    long time_start;

    printf("\nLoading warehouse table...\n");

    // Seed with unique number
    seed(2);

    InitString(w_name, W_NAME_LEN+1);
    InitAddress(w_street_1, w_street_2, w_city, w_state, w_zip);

    sprintf(name, "%s..%s", aptr->database, "warehouse");
    bcp_init(w_dbproc1, name, NULL, "logs\\whouse.err", DB_IN);

    bcp_bind(w_dbproc1, (BYTE *) &w_id, 0, -1, NULL, 0, 0, 1);
    bcp_bind(w_dbproc1, (BYTE *) w_name, 0, W_NAME_LEN, NULL, 0, 0, 2);
    bcp_bind(w_dbproc1, (BYTE *) w_street_1, 0, ADDRESS_LEN, NULL, 0, 0, 3);
    bcp_bind(w_dbproc1, (BYTE *) w_street_2, 0, ADDRESS_LEN, NULL, 0, 0, 4);
    bcp_bind(w_dbproc1, (BYTE *) w_city, 0, ADDRESS_LEN, NULL, 0, 0, 5);
    bcp_bind(w_dbproc1, (BYTE *) w_state, 0, STATE_LEN, NULL, 0, 0, 6);
    bcp_bind(w_dbproc1, (BYTE *) w_zip, 0, ZIP_LEN, NULL, 0, 0, 7);
    bcp_bind(w_dbproc1, (BYTE *) &w_tax, 0, -1, NULL, 0, SQLFLT8, 8);
    bcp_bind(w_dbproc1, (BYTE *) &w_ytd, 0, -1, NULL, 0, SQLFLT8, 9);

    time_start = (TimeNow() / MILLI);

    warehouse_rows_loaded = 0;

    for (w_id = aptr->starting_warehouse; w_id < aptr->num_warehouses+1; w_id++)
    {

        MakeAlphaString(6,10, W_NAME_LEN, w_name);
    }
}

```

```

        MakeAddress(w_street_1, w_street_2, w_city, w_state, w_zip);

        w_tax = ((float) RandomNumber(0,2000L))/10000.00;

        w_ytd = 300000.00;

        if (!bcp_sendrow(w_dbproc1))
            printf("Error, LoadWarehouse() failed calling bcp_sendrow(). Check error
file.\n");

        warehouse_rows_loaded++;
        CheckForCommit(i_dbproc1, warehouse_rows_loaded'warehouse",
&time_start);
    }

    bcp_done(w_dbproc1);
    dbcclose(w_dbproc1);

    printf("Finished loading warehouse table.\n");

    if (aptr->build_index == 1)
        BuildIndex("idxwarcl");

    stock_rows_loaded = 0;
    district_rows_loaded = 0;

    District(w_id);
    Stock(w_id);

    InterlockedIncrement(&main_threads_completed);
}

//=====
//
// Function : District
//
//=====

void District()
{
    short d_id;
    short d_w_id;
    char d_name[D_NAME_LEN+1];
    char d_street_1[ADDRESS_LEN+1];
    char d_street_2[ADDRESS_LEN+1];
    char d_city[ADDRESS_LEN+1];
    char d_state[STATE_LEN+1];
    char d_zip[ZIP_LEN+1];

```

```

double d_tax;
double d_ytd;
    char name[20];
    long d_next_o_id;
    int rc;
    long time_start;
    int w_id;

    for (w_id = aptr->starting_warehouse; w_id < aptr->num_warehouses+1; w_id++)
    {

        printf("...Loading district table: w_id = %d\n", w_id);

        // Seed with unique number
        seed(4);

        InitString(d_name, D_NAME_LEN+1);

        InitAddress(d_street_1, d_street_2, d_city, d_state, d_zip);

        sprintf(name, "%s..%s", aptr->database, "district");
        rc = bcp_init(w_dbproc2, name, NULL, "logs\district.err", DB_IN);

        bcp_bind(w_dbproc2, (BYTE *) &d_id, 0, -1, NULL, 0, 0, 1)
        bcp_bind(w_dbproc2, (BYTE *) &d_w_id, 0, -1, NULL, 0, 0, 2)
        bcp_bind(w_dbproc2, (BYTE *) d_name, 0, D_NAME_LEN, NULL
0, 0, 3);

        bcp_bind(w_dbproc2, (BYTE *) d_street_1, 0, ADDRESS_LEN, NU
0, 0, 4);

        bcp_bind(w_dbproc2, (BYTE *) d_street_2, 0, ADDRESS_LEN, NU
0, 0, 5);

        bcp_bind(w_dbproc2, (BYTE *) d_city, 0, ADDRESS_LEN, NULL
0, 6);

        bcp_bind(w_dbproc2, (BYTE *) d_state, 0, STATE_LEN, NULL, 1
0, 7);

        bcp_bind(w_dbproc2, (BYTE *) d_zip, 0, ZIP_LEN, NULL, 0, 0
8);

        bcp_bind(w_dbproc2, (BYTE *) &d_tax, 0, -1, NULL, 0,
SQLFLT8, 9);

        bcp_bind(w_dbproc2, (BYTE *) &d_ytd, 0, -1, NULL, 0,
SQLFLT8, 10);

        bcp_bind(w_dbproc2, (BYTE *) &d_next_o_id, 0, -1, NULL, 0, 0
11);

        d_w_id = w_id;

        d_ytd = 30000.0;

```

```

        d_next_o_id = 3001L;

        time_start = (TimeNow() / MILLI);

        for (d_id = 1; d_id <= DISTRICT_PER_WAREHOUSE; d_id++)
        {
            MakeAlphaString(6,10,D_NAME_LEN, d_name);

            MakeAddress(d_street_1, d_street_2, d_city, d_state, d_zip);

            d_tax = ((float) RandomNumber(0L,2000L))/10000.00;

            if (!bcp_sendrow(w_dbproc2))
                printf("Error, District() failed calling bcp_sendrow().

Check error file.\n");

            district_rows_loaded++;
            CheckForCommit(w_dbproc2, district_rows_loaded, "district",
&time_start);
        }

        rc = bcp_done(w_dbproc2);
    }

    printf("Finished loading district table.\n");

    if (aptr->build_index == 1)
        BuildIndex("idxdiscl");

    return;
}

//=====
//
// Function : Stock
//
//=====

void Stock()
{
    long s_i_id;
    short s_w_id;
    short s_quantity;
    char s_dist_01[S_DIST_LEN+1];
    char s_dist_02[S_DIST_LEN+1];
    char s_dist_03[S_DIST_LEN+1];
    char s_dist_04[S_DIST_LEN+1];
    char s_dist_05[S_DIST_LEN+1];

```

```

    char s_dist_06[S_DIST_LEN+1];
    char s_dist_07[S_DIST_LEN+1];
    char s_dist_08[S_DIST_LEN+1];
    char s_dist_09[S_DIST_LEN+1];
    char s_dist_10[S_DIST_LEN+1];
    long s_ytd;
    short s_order_cnt;
    short s_remote_cnt;
    char s_data[S_DATA_LEN+1];
    short i;
    short len;
    int rc;

    char name[20];
    long time_start;

    // Seed with unique number
    seed(3);

    sprintf(name, "%s..%s", aptr->database, "stock");
    rc = bcp_init(w_dbproc2, name, NULL, "logs\\stock.err", DB_IN);

    bcp_bind(w_dbproc2, (BYTE *) &s_i_id, 0, -1, NULL, 0, 0, 1);
    bcp_bind(w_dbproc2, (BYTE *) &s_w_id, 0, -1, NULL, 0, 0, 2);
    bcp_bind(w_dbproc2, (BYTE *) &s_quantity, 0, -1, NULL, 0, 0, 3);
    bcp_bind(w_dbproc2, (BYTE *) s_dist_01, 0, S_DIST_LEN, NULL, 0, 0, 4);
    bcp_bind(w_dbproc2, (BYTE *) s_dist_02, 0, S_DIST_LEN, NULL, 0, 0, 5);
    bcp_bind(w_dbproc2, (BYTE *) s_dist_03, 0, S_DIST_LEN, NULL, 0, 0, 6);
    bcp_bind(w_dbproc2, (BYTE *) s_dist_04, 0, S_DIST_LEN, NULL, 0, 0, 7);
    bcp_bind(w_dbproc2, (BYTE *) s_dist_05, 0, S_DIST_LEN, NULL, 0, 0, 8);
    bcp_bind(w_dbproc2, (BYTE *) s_dist_06, 0, S_DIST_LEN, NULL, 0, 0, 9);
    bcp_bind(w_dbproc2, (BYTE *) s_dist_07, 0, S_DIST_LEN, NULL, 0, 0, 10);
    bcp_bind(w_dbproc2, (BYTE *) s_dist_08, 0, S_DIST_LEN, NULL, 0, 0, 11);
    bcp_bind(w_dbproc2, (BYTE *) s_dist_09, 0, S_DIST_LEN, NULL, 0, 0, 12);
    bcp_bind(w_dbproc2, (BYTE *) s_dist_10, 0, S_DIST_LEN, NULL, 0, 0, 13);
    bcp_bind(w_dbproc2, (BYTE *) &s_ytd, 0, -1, NULL, 0, 0, 14);
    bcp_bind(w_dbproc2, (BYTE *) &s_order_cnt, 0, -1, NULL, 0, 0, 15);
    bcp_bind(w_dbproc2, (BYTE *) &s_remote_cnt, 0, -1, NULL, 0, 0, 16);
    bcp_bind(w_dbproc2, (BYTE *) s_data, 0, S_DATA_LEN, NULL, 0, 0, 17);

    s_ytd = s_order_cnt = s_remote_cnt = 0;

    time_start = (TimeNow() / MILLI);

    printf("...Loading stock table\n");

    for (s_i_id=1; s_i_id <= MAXITEMS; s_i_id++)
    {

```



```

0,
(LPTHREAD_START_ROUTINE) LoadCustomerTable,
&customer_time_start,
0,
&dwThreadID[0]);

        if (hThread[0] == NULL)
        {
                printf("Error, failed in creating creating thread = 0.\n");
                exit(-1);
        }

        // Start History table thread
        printf("...Loading history table for: d_id = %d, w_id = %d\n", d_id,
w_id);

        hThread[1] = CreateThread(NULL,

0,
(LPTHREAD_START_ROUTINE) LoadHistoryTable,
&history_time_start,
0,
&dwThreadID[1]);

        if (hThread[1] == NULL)
        {
                printf("Error, failed in creating creating thread = 1.\n");
                exit(-1);
        }

        while (customer_threads_completed != 2)
                Sleep(1000L);
    }

    // flush the bulk connection
    bcp_done(c_dbproc1);
    bcp_done(c_dbproc2);

```

```

        sprintf(buf,"update customer set c_first = 'C_LOAD = %d' where c_id = 1 and c_w_id = 1 &
c_d_id = 1",LOADER_NURAND_C);
        dbcmd(c_dbproc1, buf);
        dbsqlxec(c_dbproc1);
        while (dbresults(c_dbproc1) != NO_MORE_RESULTS);

        dbclose(c_dbproc1);
        dbclose(c_dbproc2);

        printf("Finished loading customer table.\n");

        if (aptr->build_index == 1)
                BuildIndex("idxcuscl");

        if (aptr->build_index == 1)
                BuildIndex("idxcusnc");

        InterlockedIncrement(&main_threads_completed);

        return;
}

//=====
//
// Function : CustomerBufInit
//
//=====

void CustomerBufInit()
{
        int i;

        for (i=0;i<CUSTOMERS_PER_DISTRICT;i++)
        {
                customer_buf[i].c_id = 0;
                customer_buf[i].c_d_id = 0;
                customer_buf[i].c_w_id = 0;

                strcpy(customer_buf[i].c_first,"");
                strcpy(customer_buf[i].c_middle,"");
                strcpy(customer_buf[i].c_last,"");
                strcpy(customer_buf[i].c_street_1,"");
                strcpy(customer_buf[i].c_street_2,"");
                strcpy(customer_buf[i].c_city,"");
                strcpy(customer_buf[i].c_state,"");
                strcpy(customer_buf[i].c_zip,"");

```

```

        strcpy(customer_buf[i].c_phone,"");
        strcpy(customer_buf[i].c_credit,"");

        customer_buf[i].c_credit_lim = 0;
        customer_buf[i].c_discount = (float) 0;
        customer_buf[i].c_balance = 0;
        customer_buf[i].c_ytd_payment = 0;
        customer_buf[i].c_payment_cnt = 0;
        customer_buf[i].c_delivery_cnt = 0;

        strcpy(customer_buf[i].c_data_1,"");
        strcpy(customer_buf[i].c_data_2,"");

        customer_buf[i].h_amount = 0;

        strcpy(customer_buf[i].h_data,"");
    }
}

//=====
//
// Function : CustomerBufLoad
//
// Fills shared buffer for HISTORY and CUSTOMER
//=====

void CustomerBufLoad(int d_id, int w_id)
{
    long i;
    CUSTOMER_SORT_STRUCT c[CUSTOMERS_PER_DISTRICT];

    for (i=0;i<CUSTOMERS_PER_DISTRICT;i++)
    {
        if (i < 1000)
            LastName(i, c[i].c_last);
        else
            LastName(NURand(255,0,999,LOADER_NURAND_C),
c[i].c_last);

        MakeAlphaString(8,16,FIRST_NAME_LEN, c[i].c_first);

        c[i].c_id = i+1;
    }
}

}

printf("...Loading customer bufer for: d_id = %d, w_id = %d\n",
        d_id, w_id);

for (i=0;i<CUSTOMERS_PER_DISTRICT;i++)
{
    customer_buf[i].c_d_id = d_id;
    customer_buf[i].c_w_id = w_id;
    customer_buf[i].h_amount = 10.0;
    customer_buf[i].c_ytd_payment = 0.0;
    customer_buf[i].c_payment_cnt = 1;
    customer_buf[i].c_delivery_cnt = 0;

    // Generate CUSTOMER and HISTORY data

    customer_buf[i].c_id = c[i].c_id;

    strcpy(customer_buf[i].c_first, c[i].c_first);
    strcpy(customer_buf[i].c_last, c[i].c_last);

    customer_buf[i].c_middle[0] = 'O';
    customer_buf[i].c_middle[1] = 'E';

    MakeAddress(customer_buf[i].c_street_1,
                customer_buf[i].c_street_2,
                customer_buf[i].c_city,
                customer_buf[i].c_state,
                customer_buf[i].c_zip);

    MakeNumberString(16, 16, PHONE_LEN, customer_buf[i].c_phone);

    if (RandomNumber(1L, 100L) > 10)
        customer_buf[i].c_credit[0] = 'G';
    else
        customer_buf[i].c_credit[0] = 'B';
    customer_buf[i].c_credit[1] = 'C';

    customer_buf[i].c_credit_lim = 50000.0;
    customer_buf[i].c_discount = ((float) RandomNumber(0L, 5000L)) /
10000.0;

    customer_buf[i].c_balance = -10.0;

    MakeAlphaString(250, 250, C_DATA_LEN, customer_buf[i].c_data_1);
    MakeAlphaString(50, 250, C_DATA_LEN, customer_buf[i].c_data_2);

    // Generate HISTORY data
}

```

```

        MakeAlphaString(12, 24, H_DATA_LEN, customer_buf[i].h_data);
    }
}

//=====
//
// Function : LoadCustomerTable
//
//=====

void LoadCustomerTable(LOADER_TIME_STRUCT *customer_time_start)
{
    int i;
    long c_id;
    short c_d_id;
    short c_w_id;
    char c_first[FIRST_NAME_LEN+1];
    char c_middle[MIDDLE_NAME_LEN+1];
    char c_last[LAST_NAME_LEN+1];
    char c_street_1[ADDRESS_LEN+1];
    char c_street_2[ADDRESS_LEN+1];
    char c_city[ADDRESS_LEN+1];
    char c_state[STATE_LEN+1];
    char c_zip[ZIP_LEN+1];
    char c_phone[PHONE_LEN+1];
    char c_credit[CREDIT_LEN+1];
    double c_credit_lim;
    double c_discount;
    double c_balance;
    double c_ytd_payment;
    short c_payment_cnt;
    short c_delivery_cnt;
    char c_data_1[C_DATA_LEN+1];
    char c_data_2[C_DATA_LEN+1];
    char name[20];
    char c_since[50];

    bcp_bind(c_dbproc1, (BYTE *) &c_id, 0, -1, NULL,0,0, 1);
    bcp_bind(c_dbproc1, (BYTE *) &c_d_id, 0, -1, NULL,0,0, 2);
    bcp_bind(c_dbproc1, (BYTE *) &c_w_id, 0, -1, NULL,0,0, 3);
    bcp_bind(c_dbproc1, (BYTE *) c_first, 0, FIRST_NAME_LEN, NULL,0,0, 4);
    bcp_bind(c_dbproc1, (BYTE *) c_middle, 0, MIDDLE_NAME_LEN, NULL,0,0, 5);
    bcp_bind(c_dbproc1, (BYTE *) c_last, 0, LAST_NAME_LEN, NULL,0,0, 6);
    bcp_bind(c_dbproc1, (BYTE *) c_street_1, 0, ADDRESS_LEN, NULL,0,0, 7);
    bcp_bind(c_dbproc1, (BYTE *) c_street_2, 0, ADDRESS_LEN, NULL,0,0, 8);
    bcp_bind(c_dbproc1, (BYTE *) c_city, 0, ADDRESS_LEN, NULL,0,0, 9);
    bcp_bind(c_dbproc1, (BYTE *) c_state, 0, STATE_LEN, NULL,0,0,10);

```

```

    bcp_bind(c_dbproc1, (BYTE *) c_zip, 0, ZIP_LEN, NULL,0,0,11);
    bcp_bind(c_dbproc1, (BYTE *) c_phone, 0, PHONE_LEN, NULL,0,0,12);
    bcp_bind(c_dbproc1, (BYTE *) c_since, 0, 50, NULL,0,SQLCHAR,13);
    bcp_bind(c_dbproc1, (BYTE *) c_credit, 0, CREDIT_LEN, NULL,0,0,14);
    bcp_bind(c_dbproc1, (BYTE *) &c_credit_lim, 0, -1, NULL,0,SQLFLT8,15);
    bcp_bind(c_dbproc1, (BYTE *) &c_discount, 0, -1, NULL,0,SQLFLT8,16);
    bcp_bind(c_dbproc1, (BYTE *) &c_balance, 0, -1, NULL,0,SQLFLT8,17);
    bcp_bind(c_dbproc1, (BYTE *) &c_ytd_payment, 0, -1, NULL,0,SQLFLT8,18);
    bcp_bind(c_dbproc1, (BYTE *) &c_payment_cnt, 0, -1, NULL,0,0,19);
    bcp_bind(c_dbproc1, (BYTE *) &c_delivery_cnt,0, -1, NULL,0,0,20);
    bcp_bind(c_dbproc1, (BYTE *) c_data_1, 0, C_DATA_LEN, NULL,0,0,21);
    bcp_bind(c_dbproc1, (BYTE *) c_data_2, 0, C_DATA_LEN, NULL,0,0,22);

    for (i = 0; i < CUSTOMERS_PER_DISTRICT; i++)
    {
        c_id = customer_buf[i].c_id;
        c_d_id = customer_buf[i].c_d_id;
        c_w_id = customer_buf[i].c_w_id;

        strcpy(c_first, customer_buf[i].c_first);
        strcpy(c_middle, customer_buf[i].c_middle);
        strcpy(c_last, customer_buf[i].c_last);
        strcpy(c_street_1, customer_buf[i].c_street_1);
        strcpy(c_street_2, customer_buf[i].c_street_2);
        strcpy(c_city, customer_buf[i].c_city);
        strcpy(c_state, customer_buf[i].c_state);
        strcpy(c_zip, customer_buf[i].c_zip);
        strcpy(c_phone, customer_buf[i].c_phone);
        strcpy(c_credit, customer_buf[i].c_credit);

        CurrentDate(&c_since);

        c_credit_lim = customer_buf[i].c_credit_lim;
        c_discount = customer_buf[i].c_discount;
        c_balance = customer_buf[i].c_balance;
        c_ytd_payment = customer_buf[i].c_ytd_payment;
        c_payment_cnt = customer_buf[i].c_payment_cnt;
        c_delivery_cnt = customer_buf[i].c_delivery_cnt;

        strcpy(c_data_1, customer_buf[i].c_data_1);
        strcpy(c_data_2, customer_buf[i].c_data_2);

        // Send data to server
        if (!bcp_sendrow(c_dbproc1))
            printf("Error, LoadCustomerTable() failed calling bcp_sendrow(). Check
            error file.\n");

        customer_rows_loaded++;
    }

```

```

        CheckForCommit(c_dbproc1, customer_rows_loaded, "customer",
&customer_time_start->time_start);
    }

    InterlockedIncrement(&customer_threads_completed);

}

//=====
//
// Function : LoadHistoryTable
//
//=====

void LoadHistoryTable(LOADER_TIME_STRUCT *history_time_start)
{
    int i;
    long c_id;
    short c_d_id;
    short c_w_id;
    double h_amount;
    char h_data[H_DATA_LEN+1];
    char h_date[50];

    bcp_bind(c_dbproc2, (BYTE *) &c_id, 0, -1, NULL, 0, 0, 1);
    bcp_bind(c_dbproc2, (BYTE *) &c_d_id, 0, -1, NULL, 0, 0, 2);
    bcp_bind(c_dbproc2, (BYTE *) &c_w_id, 0, -1, NULL, 0, 0, 3);
    bcp_bind(c_dbproc2, (BYTE *) &c_d_id, 0, -1, NULL, 0, 0, 4);
    bcp_bind(c_dbproc2, (BYTE *) &c_w_id, 0, -1, NULL, 0, 0, 5);
    bcp_bind(c_dbproc2, (BYTE *) h_date, 0, 50, NULL, 0, SQLCHAR, 6);
    bcp_bind(c_dbproc2, (BYTE *) &h_amount, 0, -1, NULL, 0, SQLFLT8, 7);
    bcp_bind(c_dbproc2, (BYTE *) h_data, 0, H_DATA_LEN, NULL, 0, 0, 8);

    for (i = 0; i < CUSTOMERS_PER_DISTRICT; i++)
    {
        c_id = customer_buff[i].c_id;
        c_d_id = customer_buff[i].c_d_id;
        c_w_id = customer_buff[i].c_w_id;
        h_amount = customer_buff[i].h_amount;
        strcpy(h_data, customer_buff[i].h_data);
        CurrentDate(&h_date);

        // send to server
        if (!bcp_sendrow(c_dbproc2))
            printf("Error, LoadHistoryTable() failed calling bcp_sendrow(). Check error
file.\n");

        history_rows_loaded++;
    }
}

```

```

        CheckForCommit(c_dbproc2, history_rows_loaded, "history",
&history_time_start->time_start);
    }

    InterlockedIncrement(&customer_threads_completed);

}

//=====
//
// Function : LoadOrders
//
//=====

void LoadOrders()
{
    LOADER_TIME_STRUCT orders_time_start;
    LOADER_TIME_STRUCT new_order_time_start;
    LOADER_TIME_STRUCT order_line_time_start;
    short w_id;

    short d_id;
    DWORD dwThreadID[MAX_ORDER_THREADS];
    HANDLE hThread[MAX_ORDER_THREADS];
    char name[20];

    printf("\nLoading orders...\n");

    // seed with unique number
    seed(6);

    // initialize bulk copy
    sprintf(name, "%s..%s", aptr->database, "orders");
    bcp_init(o_dbproc1, name, NULL, "logs\\orders.err", DB_IN);

    sprintf(name, "%s..%s", aptr->database, "new_order");
    bcp_init(o_dbproc2, name, NULL, "logs\\neword.err", DB_IN);

    sprintf(name, "%s..%s", aptr->database, "order_line");
    bcp_init(o_dbproc3, name, NULL, "logs\\ordline.err", DB_IN);

    orders_rows_loaded = 0;
    new_order_rows_loaded = 0;
    order_line_rows_loaded = 0;
}

```



```

OrdersBufInit();

orders_time_start.time_start = (TimeNow() / MILLI);
new_order_time_start.time_start = (TimeNow() / MILLI);
order_line_time_start.time_start = (TimeNow() / MILLI);

for (w_id = apr->starting_warehouse; w_id <= apr->num_warehouses; w_id++)
{
    for (d_id = 1L; d_id <= DISTRICT_PER_WAREHOUSE; d_id++)
    {
        OrdersBufLoad(d_id, w_id);

        // start parallel loading threads here...

        order_threads_completed=0;

        // start Orders table thread

        printf("...Loading Order Table for: d_id = %d, w_id = %d\n", d_id,
w_id);

        hThread[0] = CreateThread(NULL,

0,

(LPTHREAD_START_ROUTINE) LoadOrdersTable,

&orders_time_start,

0,

&dwThreadID[0]);

        if (hThread[0] == NULL)
        {
            printf("Error, failed in creating creating thread = 0.\n");
            exit(-1);
        }

        // start NewOrder table thread

        printf("...Loading New-Order Tablefor: d_id = %d, w_id = %d\n",
d_id, w_id);

        hThread[1] = CreateThread(NULL,

0,

```

```

(LPTHREAD_START_ROUTINE) LoadNewOrderTable,

&new_order_time_start,

0,

&dwThreadID[1]);

        if (hThread[1] == NULL)
        {
            printf("Error, failed in creating creating thread = 1.\r
            exit(-1);
        }

        // start Order-Line table thread

        printf("...Loading Order-Line Table for: d_id = %d, w_id = %d\
d_id, w_id);

        hThread[2] = CreateThread(NULL,

0,

(LPTHREAD_START_ROUTINE) LoadOrderLineTable,

&order_line_time_start,

0,

&dwThreadID[2]);

        if (hThread[2] == NULL)
        {
            printf("Error, failed in creating creating thread = 2.\r
            exit(-1);
        }

        while (order_threads_completed != 3)
            Sleep(1000L);

    }

    printf("Finished loading orders.\n");

    InterlockedIncrement(&main_threads_completed);

```

```

return;
}

//=====
//
// Function : OrdersBufInit
//
// Clears shared buffer for ORDERS, NEWORDER, and ORDERLINE
//
//=====

void OrdersBufInit()
{
    int i;
    int j;

    for (i=0;i<ORDERS_PER_DISTRICT;i++)
    {
        orders_buf[i].o_id = 0;
        orders_buf[i].o_d_id = 0;
        orders_buf[i].o_w_id = 0;
        orders_buf[i].o_c_id = 0;
        orders_buf[i].o_carrier_id = 0;
        orders_buf[i].o_ol_cnt = 0;
        orders_buf[i].o_all_local = 0;

        for (j=0;j<=14;j++)
        {
            orders_buf[i].o_ol[j].ol = 0;
            orders_buf[i].o_ol[j].ol_i_id = 0;
            orders_buf[i].o_ol[j].ol_supply_w_id = 0;
            orders_buf[i].o_ol[j].ol_quantity = 0;
            orders_buf[i].o_ol[j].ol_amount = 0;
            strcpy(orders_buf[i].o_ol[j].ol_dist_info, "");
        }
    }
}

//=====
//
// Function : OrdersBufLoad
//
// Fills shared buffer for ORDERS, NEWORDER, and ORDERLINE
//

```

```

//=====

void OrdersBufLoad(int d_id, int w_id)
{
    int cust[ORDERS_PER_DIST+1];
    long o_id;
    short ol;

    printf("...Loading Order Buffer for: d_id = %d, w_id = %d\n",
           d_id, w_id);

    GetPermutation(cust, ORDERS_PER_DIST);

    for (o_id=0;o_id<ORDERS_PER_DISTRICT;o_id++)
    {
        // Generate ORDER and NEW-ORDER data

        orders_buf[o_id].o_d_id = d_id;
        orders_buf[o_id].o_w_id = w_id;
        orders_buf[o_id].o_id = o_id+1;
        orders_buf[o_id].o_c_id = cust[o_id+1];
        orders_buf[o_id].o_ol_cnt = RandomNumber(5L, 15L);

        if (o_id < 2100)
        {
            orders_buf[o_id].o_carrier_id = RandomNumber(1L, 10L);
            orders_buf[o_id].o_all_local = 1;
        }
        else
        {
            orders_buf[o_id].o_carrier_id = 0;
            orders_buf[o_id].o_all_local = 1;
        }

        for (ol=0;ol<orders_buf[o_id].o_ol_cnt;ol++)
        {

            orders_buf[o_id].o_ol[ol].ol = ol+1;
            orders_buf[o_id].o_ol[ol].ol_i_id = RandomNumber(1L,
MAXITEMS);

            orders_buf[o_id].o_ol[ol].ol_supply_w_id = w_id;
            orders_buf[o_id].o_ol[ol].ol_quantity = 5;
            MakeAlphaString(24, 24, OL_DIST_INFO_LEN,
&orders_buf[o_id].o_ol[ol].ol_dist_info);

            // Generate ORDER-LINE data

```

```

        if (o_id < 2100)
        {
            orders_buf[o_id].o_ol[o].ol_amount = 0;
            // Added to insure ol_delivery_d set properly during
load      CurrentDate(&orders_buf[o_id].o_ol[o].ol_delivery_d);
        }
        else
        {
            orders_buf[o_id].o_ol[o].ol_amount =
load      RandomNumber(1,999999)/100.0;
            // Added to insure ol_delivery_d set properly during
load      strcpy(orders_buf[o_id].o_ol[o].ol_delivery_d,"Dec 31,
1889");
        }
    }
}

```

```

//=====
//
// Function : LoadOrdersTable
//
//=====

```

```

void LoadOrdersTable(LOADER_TIME_STRUCT *orders_time_start)
{
    int i;
    long o_id;
    short o_d_id;
    short o_w_id;
    long o_c_id;
    short o_carrier_id;
    short o_ol_cnt;
    short o_all_local;
    char o_entry_d[50];

    // bind ORDER data
    bcp_bind(o_dbproc1, (BYTE *) &o_id, 0, -1, NULL, 0, 0, 1);
    bcp_bind(o_dbproc1, (BYTE *) &o_d_id, 0, -1, NULL, 0, 0, 2);
    bcp_bind(o_dbproc1, (BYTE *) &o_w_id, 0, -1, NULL, 0, 0, 3);
    bcp_bind(o_dbproc1, (BYTE *) &o_c_id, 0, -1, NULL, 0, 0, 4);
    bcp_bind(o_dbproc1, (BYTE *) o_entry_d, 0, 50, NULL, 0, SQLCHAR, 5);
    bcp_bind(o_dbproc1, (BYTE *) &o_carrier_id, 0, -1, NULL, 0, 0, 6);
    bcp_bind(o_dbproc1, (BYTE *) &o_ol_cnt, 0, -1, NULL, 0, 0, 7);
    bcp_bind(o_dbproc1, (BYTE *) &o_all_local, 0, -1, NULL, 0, 0, 8);
}

```

```

for (i = 0; i < ORDERS_PER_DISTRICT; i++)
{
    o_id = orders_buf[i].o_id;
    o_d_id = orders_buf[i].o_d_id;
    o_w_id = orders_buf[i].o_w_id;
    o_c_id = orders_buf[i].o_c_id;
    o_carrier_id = orders_buf[i].o_carrier_id;
    o_ol_cnt = orders_buf[i].o_ol_cnt;
    o_all_local = orders_buf[i].o_all_local;
    CurrentDate(&o_entry_d);

    // send data to server
    if (!bcp_sendrow(o_dbproc1))
        printf("Error, LoadOrdersTable() failed calling bcp_sendrow(). Check err
file.\n");

    orders_rows_loaded++;
    CheckForCommit(o_dbproc1, orders_rows_loaded, "ORDERS",
&orders_time_start->time_start);
}

if ((o_w_id == apr->num_warehouses) && (o_d_id == 10))
{
    bcp_done(o_dbproc1);
    dbclose(o_dbproc1);

    if (apr->build_index == 1)
        BuildIndex("idxordcl");
}

InterlockedIncrement(&order_threads_completed);
}

```

```

//=====
//
// Function : LoadNewOrderTable
//
//=====

```

```

void LoadNewOrderTable(LOADER_TIME_STRUCT *new_order_time_start)
{
    int i;
    long o_id;
    short o_d_id;
    short o_w_id;
}

```

```

// Bind NEW-ORDER data
bcp_bind(o_dbproc2, (BYTE *) &o_id,      0, -1, NULL, 0, 0, 1);
bcp_bind(o_dbproc2, (BYTE *) &o_d_id,    0, -1, NULL, 0, 0, 2);
bcp_bind(o_dbproc2, (BYTE *) &o_w_id,    0, -1, NULL, 0, 0, 3);

for (i = 2100; i < 3000; i++)
{
    o_id = orders_buf[i].o_id;
    o_d_id = orders_buf[i].o_d_id;
    o_w_id = orders_buf[i].o_w_id;

    if (!bcp_sendrow(o_dbproc2))
        printf("Error, LoadNewOrderTable() failed calling bcp_sendrow(). Check
error file.\n");
    new_order_rows_loaded++;
    CheckForCommit(o_dbproc2, new_order_rows_loaded, "NEW_ORDER",
&new_order_time_start->time_start);
}

if ((o_w_id == aptr->num_warehouses) && (o_d_id == 10))
{
    bcp_done(o_dbproc2);
    dbclose(o_dbproc2);

    if (aptr->build_index == 1)
        BuildIndex("idxnodc!");
}

InterlockedIncrement(&order_threads_completed);
}

//=====================================================
//
// Function : LoadOrderLineTable
//
//=====================================================

void LoadOrderLineTable(LOADER_TIME_STRUCT *order_line_time_start)
{
    int i,j;
    long o_id;
    short o_d_id;
    short o_w_id;
    long ol;
    long ol_i_id;

```

```

short ol_supply_w_id;
short ol_quantity;
double ol_amount;
short o_all_local;
char ol_dist_info[DIST_INFO_LEN+1];
char ol_delivery_d[50];

// bind ORDER-LINE data
bcp_bind(o_dbproc3, (BYTE *) &o_id,      0, -1, NULL, 0, 0, 1);
bcp_bind(o_dbproc3, (BYTE *) &o_d_id,    0, -1, NULL, 0, 0, 2);
bcp_bind(o_dbproc3, (BYTE *) &o_w_id,    0, -1, NULL, 0, 0, 3);
bcp_bind(o_dbproc3, (BYTE *) &ol,      0, -1, NULL, 0, 0, 4);
bcp_bind(o_dbproc3, (BYTE *) &ol_i_id,  0, -1, NULL, 0, 0, 5);
bcp_bind(o_dbproc3, (BYTE *) &ol_supply_w_id, 0, -1, NULL, 0, 0, 6);
bcp_bind(o_dbproc3, (BYTE *) ol_delivery_d, 0, 50,
NULL, 0, SQLCHAR, 7);
bcp_bind(o_dbproc3, (BYTE *) &ol_quantity, 0, -1, NULL, 0, 0, 8);
bcp_bind(o_dbproc3, (BYTE *) &ol_amount, 0, -1, NULL, 0, SQLFLT8, 9);
bcp_bind(o_dbproc3, (BYTE *) ol_dist_info, 0, DIST_INFO_LEN, NULL, 0, 0, 10);

for (i = 0; i < ORDERS_PER_DISTRICT; i++)
{
    o_id = orders_buf[i].o_id;
    o_d_id = orders_buf[i].o_d_id;
    o_w_id = orders_buf[i].o_w_id;

    for (j=0; j < orders_buf[i].o_ol_cnt; j++)
    {
        ol = orders_buf[i].o_ol[j].ol;
        ol_i_id = orders_buf[i].o_ol[j].ol_i_id;
        ol_supply_w_id = orders_buf[i].o_ol[j].ol_supply_w_id;
        ol_quantity = orders_buf[i].o_ol[j].ol_quantity;
        ol_amount = orders_buf[i].o_ol[j].ol_amount;
        // Changed to insure ol_delivery_d set properly (now set in
OrdersBufLoad)
        // CurrentDate(&ol_delivery_d);
        strcpy(ol_delivery_d,orders_buf[i].o_ol[j].ol_delivery_d);

        strcpy(ol_dist_info,orders_buf[i].o_ol[j].ol_dist_info);

        if (!bcp_sendrow(o_dbproc3))
            printf("Error, LoadOrderLineTable() failed calling
bcp_sendrow(). Check error file.\n");
        order_line_rows_loaded++;
        CheckForCommit(o_dbproc3, order_line_rows_loaded,
"ORDER_LINE", &order_line_time_start->time_start);
    }
}

```

```

    }

    if ((o_w_id == apr->num_warehouses) && (o_d_id == 10))
    {
        bcp_done(o_dbproc3);
        dbclose(o_dbproc3);

        if (aptr->build_index == 1)
            BuildIndex("idxodlcl");
    }

    InterlockedIncrement(&order_threads_completed);
}

//=====
//
// Function : GetPermutation
//
//=====

void GetPermutation(int perm[], int n)
{
    int i, r, t;

    for (i=1; i<=n; i++)
        perm[i] = i;

    for (i=1; i<=n; i++)
    {
        r = RandomNumber(i,n);
        t = perm[i];
        perm[i] = perm[r];
        perm[r] = t;
    }
}

//=====
//
// Function : CheckForCommit
//
//=====

void CheckForCommit(DBPROCESS *dbproc,
                    int rows_loaded,

```

```

                    char *table_name,
                    long *time_start)
{
    long time_end, time_diff;

    // commit every "batch" rows

    if ( !(rows_loaded % apr->batch) )
    {
        bcp_batch(dbproc);

        time_end = (TimeNow() / MILLI);
        time_diff = time_end - *time_start;

        printf("-> Loaded %ld rows into %s in %ld sec - Total = %d (%.2f rps)\n"
               apr->batch,
               table_name,
               time_diff,
               rows_loaded,
               (float) apr->batch / (time_diff ? time_diff : 1L));

        *time_start = time_end;
    }

    return;
}

//=====
//
// Function : OpenConnections
//
//=====

void OpenConnections()
{
    RETCODE retcode;

    LOGINREC *login;

    login = dblogin();

    retcode = DBSETLUSER(login, apr->user);
    if (retcode == FAIL)
    {
        printf("DBSETLUSER failed.\n");
    }
}

```

```

retcode = DBSETLPWD(login, apr->password);
if (retcode == FAIL)
{
    printf("DBSETLPWD failed.\n");
}

retcode = DBSETLPACKET(login, (USHORT) apr->pack_size);
if (retcode == FAIL)
{
    printf("DBSETLPACKET failed.\n");
}

printf("DB-Library packet size: %ld\n", apr->pack_size);

// turn connection into a BCP connection
retcode = BCP_SETL(login, TRUE);
if (retcode == FAIL)
{
    printf("BCP_SETL failed.\n");
}

// open connections to SQL Server */
if ((i_dbproc1 = dbopen(login, apr->server)) == NULL)
{
    printf("Error on login 1 to server %s.\n", apr->server);
    exit(-1);
}

if ((w_dbproc1 = dbopen(login, apr->server)) == NULL)
{
    printf("Error on login 2 to server %s.\n", apr->server);
    exit(-1);
}

if ((w_dbproc2 = dbopen(login, apr->server)) == NULL)
{
    printf("Error on login 3 to server %s.\n", apr->server);
    exit(-1);
}

if ((c_dbproc1 = dbopen(login, apr->server)) == NULL)
{
    printf("Error on login 4 to server %s.\n", apr->server);
    exit(-1);
}

if ((c_dbproc2 = dbopen(login, apr->server)) == NULL)

```

```

{
    printf("Error on login 5 to server %s.\n", apr->server);
    exit(-1);
}

if ((o_dbproc1 = dbopen(login, apr->server)) == NULL)
{
    printf("Error on login 6 to server %s.\n", apr->server);
    exit(-1);
}

if ((o_dbproc2 = dbopen(login, apr->server)) == NULL)
{
    printf("Error on login 7 to server %s.\n", apr->server);
    exit(-1);
}

if ((o_dbproc3 = dbopen(login, apr->server)) == NULL)
{
    printf("Error on login 8 to server %s.\n", apr->server);
    exit(-1);
}
}

//=====
//
// Function name: SQLErrHandler
//
//=====

int SQLErrHandler(SQLCONN *dbproc,
                  int severity,
                  int err,
                  int oserr,
                  char *dberrstr,
                  char *oserrstr)
{
    char msg[256];
    FILE *fp1;
    char timebuf[128];
    char datebuf[128];

    _strtime(timebuf);
    _strdate(datebuf);

```

```

        sprintf(msg, "%s %s : DBLibrary (%ld) %s\n", datebuf, timebuf, err, dberrstr);
        printf("%s",msg);

        fp1 = fopen("logs\tpccldr.err","a");
        if (fp1 == NULL)
        {
            printf("Error in opening errorlog file.\n");
        }
        else
        {
            fprintf(fp1, msg);
            fclose(fp1);
        }

        if (oserr != DBNOERR)
        {
            sprintf(msg, "%s %s : OSErrror (%ld) %s\n", datebuf, timebuf, oserr,
oserrstr);
            printf("%s",msg);

            fp1 = fopen("logs\tpccldr.err","a");
            if (fp1 == NULL)
            {
                printf("Error in opening errorlog file.\n");
            }
            else
            {
                fprintf(fp1, msg);
                fclose(fp1);
            }
        }

        if ((dbproc == NULL) || (DBDEAD(dbproc)))
        {
            exit(-1);
        }

        return (INT_CANCEL);
    }

//=====
//
// Function name: SQLMsgHandler
//
//=====

```

```

int SQLMsgHandler(SQLCONN *dbproc,
                  DBINT msgno,
                  int msgstate,
                  int severity,
                  char *msgtext)
{
    char msg[256];
    FILE *fp1;
    char timebuf[128];
    char datebuf[128];

    if ( (msgno == 5701) || (msgno == 2528) || (msgno == 5703) || (msgno == 6006) )
    {
        return(INT_CONTINUE);
    }

    if (msgno == 0)
    {
        return(INT_CONTINUE);
    }
    else
    {
        _strtime(timebuf);
        _strdate(datebuf);

        sprintf(msg, "%s %s : SQLServer (%ld) %s\n", datebuf, timebuf, msgno,
msgtext);

        printf("%s",msg);

        fp1 = fopen("logs\tpccldr.err","a");
        if (fp1 == NULL)
        {
            printf("Error in opening errorlog file.\n");
        }
        else
        {
            fprintf(fp1, msg);
            fclose(fp1);
        }

        exit(-1);
    }

    return (INT_CANCEL);
}

```

```

=====
//
// Function name: CurrentDate
//
=====

void CurrentDate(char      *datetime)
{
    char timebuf[128];
    char datebuf[128];

    _strtime(timebuf);
    _strdate(datebuf);

    sprintf(datetime, "%s %s", datebuf, timebuf);
}

=====
//
// Function name: BuildIndex
//
=====

void BuildIndex(char*index_script)
{
    char    cmd[256];

    printf("Starting index creation: %s\n",index_script);

    sprintf(cmd, "isql -S%s -U%s -P%s -e -i%s\\%s.sql >> logs\\%s.out",
            aptr->server,
            aptr->user,
            aptr->password,
            aptr->index_script_path,
            index_script,
            index_script);

    system(cmd);

    printf("Finished index creation: %s\n",index_script);
}

```

B.26 getargs.c

```

// TPC-C Benchmark Kit
//
// Module: GETARGS.C
// Author: DamienL

// Includes
#include "tpcc.h"

=====
//
// Function name: GetArgsLoader
//
=====

void GetArgsLoader(int argc, char **argv, TPCCLDR_ARGS *pargs)
{
    int          i;
    char        *ptr;

#ifdef DEBUG
    printf("[%ld]DBG: Entering GetArgsLoader()\n", (int) GetCurrentThreadId());
#endif

    /* init args struct with some useful values */
    pargs->server          = SERVER;
    pargs->user             = USER;
    pargs->password         = PASSWORD;
    pargs->database         = DATABASE;
    pargs->batch            = BATCH;
    pargs->num_warehouses   = UNDEF;
    pargs->table            = NULL;
    pargs->loader_res_file  = LOADER_RES_FILE;
    pargs->pack_size        = DEF_LDPPACKSIZE;
    pargs->starting_warehouse = DEF_STARTING_WAREHOUSE;
    pargs->build_index      = BUILD_INDEX;
    pargs->index_script_path = INDEX_SCRIPT_PATH;

    /* check for zero command line args */
    if ( argc == 1 )
        GetArgsLoaderUsage();

    for (i = 1; i < argc; ++i)
    {
        if (argv[i][0] != '-' && argv[i][0] != '/')
        {

```



```

printf("\nUnrecognized command");
GetArgsLoaderUsage();
exit(1);
}

ptr = argv[i];

switch (ptr[1])
{
case 'h': /* Fall through */
case 'H':
            GetArgsLoaderUsage();
            break;

case 'D':
            pargs->database = ptr+2;
            break;

case 'P':
            pargs->password = ptr+2;
            break;

case 'S':
            pargs->server = ptr+2;
            break;

case 'U':
            pargs->user = ptr+2;
            break;

case 'b':
            pargs->batch = atol(ptr+2);
            break;

case 'W':
            pargs->num_warehouses =atol(ptr+2);
            break;

case 's':
            pargs->starting_warehouse = atol(ptr+2);
            break;

case 't':
            pargs->table = ptr+2;
            break;

case 'f':
            pargs->loader_res_file = ptr+2;

```

```

            break;

case 'p':
            pargs->pack_size =atol(ptr+2);
            break;

case 'i':
            pargs->build_index = atol(ptr+2);

case 'd':
            pargs->index_script_path = ptr+2;
            break;

default:
            GetArgsLoaderUsage();
            exit(-1);
            break;
}

}

/* check for required args */
if (pargs->num_warehouses == UNDEF )
{
    printf("Number of Warehouses is required\n");
    exit(-2);
}

return;
}

//=====
//
// Function name: GetArgsLoaderUsage
//
//=====

void GetArgsLoaderUsage()
{
#ifdef DEBUG
    printf("[%ld]DBG: Entering GetArgsLoaderUsage()\n", (int) GetCurrentThreadId());
#endif

    printf("TPCCLDR:\n\n");
    printf("Parameter                                Default\n");

```

```

        printf("-----\n");
printf("-W Number of Warehouses to Load      Required \n");
printf("-S Server                               %s\n", SERVER);
printf("-U Username                             %s\n", USER);
printf("-P Password                             %s\n", PASSWORD);
printf("-D Database                             %s\n", DATABASE);
printf("-b Batch Size                             %ld\n", (long) BATCH);
printf("-p TDS packet size                       %ld\n", (long) DEFLDPACKSIZE);
printf("-f Loader Results Output Filename       %s\n", LOADER_RES_FILE);
printf("-s Starting Warehouse                   %ld\n", (long)
DEF_STARTING_WAREHOUSE);
printf("-i Build Option (data = 0, data and index = 1) %ld\n", (long)
BUILD_INDEX);
printf("-d Index Script Path                     %s\n", INDEX_SCRIPT_PATH);
printf("-t Table to Load                         all tables \n");

printf(" [itemwarehousecustomerorders]\n");

printf("\nNote: Command line switches are case sensitive.\n");

exit(0);
}

//=====
//
// Function name: GetArgsMaster
//
//=====

void GetArgsMaster(int argc, char **argv, MASTER_DATA *pargs)
{
    int      i;
    char    *ptr;

#ifdef DEBUG
printf("[%d]DBG: Entering GetArgsMaster()\n", (int) GetCurrentThreadId());
#endif

    pargs->server      = SERVER;
    pargs->database    = DATABASE;
    pargs->admin_database = ADMIN_DATABASE;
    pargs->user        = USER;
    pargs->password    = PASSWORD;
    pargs->ramp_up     = RAMP_UP;
    pargs->steady_state = STEADY_STATE;
    pargs->ramp_down   = RAMP_DOWN;
    pargs->num_users   = NUM_USERS;
    pargs->num_warehouses = NUM_WAREHOUSES;

```

```

pargs->think_times    = THINK_TIMES;
pargs->display_data    = DISPLAY_DATA;
    pargs->deadlock_retry = DEADLOCK_RETRY;
pargs->tran            = TRANSACTION;
    pargs->client_mode    = CLIENT_MODE;
    pargs->comment        = NULL;
    pargs->load_multiplier = DEF_LOAD_MULTIPLIER;
    pargs->checkpoint_interval = DEF_CHECKPOINT_INTERVAL;
    pargs->first_checkpoint = DEF_FIRST_CHECKPOINT;
    pargs->delivery_backoff = DELIVERY_BACKOFF;
    pargs->num_deliveries   = NUM_DELIVERIES;
    pargs->disable_90th    = DISABLE_90TH;
    pargs->enable_sqlstat  = ENABLE_SQLSTAT;
    pargs->resfilename     = RESFILENAME;
    pargs->sqlstat_filename = SQLSTAT_FILENAME;
    pargs->sqlstat_perdb  = SQLSTAT_PERIOD;
    pargs->shutdown_server = SHUTDOWN_SERVER;
    pargs->auto_run        = AUTO_RUN;
    pargs->disable_sqlperf = DISABLE_SQLPERF;

```

```
/* check for zero command line args */
```

```
if ( argc == 1 )
```

```
    GetArgsMasterUsage();
```

```
for ( i = 1; i < argc; ++i)
```

```
{
```

```
    if (argv[i][0] != '-' && argv[i][0] != '/')
```

```
    {
```

```
        printf("\nUnrecognized command");
```

```
        GetArgsMasterUsage();
```

```
        exit(1);
```

```
    }
```

```
    ptr = argv[i];
```

```
    switch (ptr[1])
```

```
    {
```

```
        case 'h': /* Fall through */
```

```
            GetArgsMasterUsage();
```

```
            break;
```

```
        case 'S':
```

```
            pargs->server = ptr+2;
```

```
            break;
```

```
        case 'D':
```

```
            pargs->database = ptr+2;
```

```
            break;
```

```

case 'A':
    pargs->admin_database = ptr+2;
    break;

case 'U':
    pargs->user = ptr+2;
    break;

case 'P':
    pargs->password = ptr+2;
    break;

case 'u':
    pargs->ramp_up = atol(ptr+2);
    break;

case 's':
    pargs->steady_state = atol(ptr+2);
    break;

case 'd':
    pargs->ramp_down = atol(ptr+2);
    break;

case 'c':
    pargs->num_users = atol(ptr+2);
    break;

case 'w':
    pargs->num_warehouses = atol(ptr+2);
    break;

case 'T':
    pargs->think_times = atol(ptr+2);
    break;

case 'o':
    pargs->display_data = atol(ptr+2);
    break;

case 'm':
    pargs->load_multiplier = atof(ptr+2);
    break;

case 'f':
    pargs->first_checkpoint = atol(ptr+2);
    break;

```

```

case 'i':
    pargs->checkpoint_interval = atol(ptr+2);
    break;

case 'C':
    pargs->comment = ptr+2;
    break;

case 'B':
    pargs->client_mode = atol(ptr+2);
    break;

case 'n':
    pargs->num_deliveries = atol(ptr+2);
    break;

case 'b':
    pargs->delivery_backoff = atol(ptr+2);
    break;

case 'r':
    pargs->deadlock_retry = (short) atol(ptr+2);
    break;

case 't':
    pargs->tran = atol(ptr+2);
    break;

case 'E':
    pargs->enable_sqlstat = atol(ptr+2);
    break;

case 'e':
    pargs->sqlstat_filename = ptr+2;
    break;

case 'g':
    pargs->shutdown_server = atol(ptr+2);
    break;

case 'F':
    pargs->resfilename = ptr+2;
    break;

case 'N':
    pargs->disable_90th = atol(ptr+2);
    break;

```

```

        case 'a':
            pargs->auto_run = atol(ptr+2);
            break;

        case 'q':
            pargs->disable_sqlperf = atol(ptr+2);
            break;

        case 'W':
            pargs->sqlstat_period = atol(ptr+2);
            break;

        default:
            GetArgsMasterUsage();
            exit(-1);
            break;
    }

    return;
}

//=====
//
// Function name: GetArgsMasterUsage
//
//=====

void GetArgsMasterUsage()
{
#ifdef DEBUG
    printf("[%ld]DBG: Entering GetArgsMasterUsage()\n", (int) GetCurrentThreadId());
#endif

    printf("MASTER:\n\n");
    printf("Parameter                Default\n");
    printf("-----\n");
    printf("-S Server                    %s\n", SERVER);
    printf("-D Database                   %s\n", DATABASE);
    printf("-A Admin Database             %s\n", ADMIN_DATABASE);
    printf("-U Username                   %s\n", USER);
    printf("-P Password                   %s\n", PASSWORD);
    printf("-u Ramp Up Time (seconds)    %ld\n", (long)
RAMP_UP);

    printf("-s Steady State Time (seconds) %ld\n", (long)
STEADY_STATE);
    printf("-d Ramp Down Time (seconds)   %ld\n", (long)
RAMP_DOWN);
    printf("-c Number of Users             %ld\n", (long) NUM_USERS);
    printf("-w Number of Warehouses       %ld\n", (long)
NUM_WAREHOUSES);
    printf("-f First Checkpoint (seconds) %ld\n", (long)
DEF_FIRST_CHECKPOINT);
    printf("-i Checkpoint Interval (seconds) %ld\n", (long)
DEF_CHECKPOINT_INTERVAL);
    printf("-B Client mode (TPC-C Scaled = 0, TPC-C Batch = 1) %ld\n", (long)
CLIENT_MODE);
    printf("-n Number of Delivery Threads per Client Driver %ld\n", (long)
NUM_DELIVERIES);
    printf("-b Delivery Queue Backoff Delay (seconds) %ld\n", (long)
DELIVERY_BACKOFF);
    printf("-r Deadlock Retries           %ld\n", (long)
DEADLOCK_RETRY);
    printf("-T Use Think Times (no = 0, yes = 1) %ld\n", (long)
THINK_TIMES);
    printf("-m Think Time Load Multiplier %0.4f\n",
DEF_LOAD_MULTIPLIER);
    printf("-o Display Data to Console (no = 0, yes = 1) %ld\n", (long)
DISPLAY_DATA);
    printf("-t Transaction (0, 1, 2, 3, 4, 5) %ld\n", (long)
TRANSACTION);
    printf("-N Disable 90th Per. Calc. (no = 0, yes = 1) %ld\n", (long)
DISABLE_90TH);
    printf("-E Enable Steady State Sqlstats Collection (no = 0, yes = 1) %ld\n", (long)
ENABLE_SQLSTAT);
    printf("-W Sqlstats Collection Period (seconds) %ld\n", (long)
SQLSTAT_PERIOD);
    printf("-e Sqlstats File Name         %s\n",
SQLSTAT_FILENAME);
    printf("-g Shutdown SQL Server at End of Test (no = 0, yes = 1) %ld\n", (long)
SHUTDOWN_SERVER);
    printf("-F Result File Name           %s\n", RESFILENAME);
    printf("-a Automated Test Run (no = 0, yes = 1) %ld\n", (long)
AUTO_RUN);
    printf("-C Comment to Include in Result File None\n");
    printf("\nNote: Command line switches are case sensitive.\n");

    exit(0);
}

//=====
//

```

```
// Function name: GetArgsClient
//
//=====
void GetArgsClient(int argc, char **argv, GLOBAL_CLIENT_DATA *pClient)
{
    int          i;
    char        *ptr;

#ifdef DEBUG
    printf("[%d]DBG: Entering GetArgsClient()\n", (int) GetCurrentThreadId());
#endif

    pClient->num_threads      = NUM_THREADS;
    pClient->server           = SERVER;
    pClient->database         = DATABASE;
    pClient->admin_database   = ADMIN_DATABASE;
    pClient->user             = USER;
    pClient->password        = PASSWORD;
    pClient->pack_size        = (long) DEFCLPACKSIZE;
    pClient->synch_servername = SYNCH_SERVERNAME;
    pClient->disable_delivery_resfiles = DISABLE_DELIVERY_RESFILES;
    pClient->enable_qj        = ENABLE_QJ;

    /* check for 1 or more command line args */
    if ( argc != 1 )
    {
        for (i = 1; i < argc; ++i)
        {
            if (argv[i][0] != '-' && argv[i][0] != '/')
            {
                printf("\nUnrecognized command");
                GetArgsClientUsage();
                exit(1);
            }

            ptr = argv[i]

            switch (ptr[1])
            {
                case 'S':
                    pClient->server = ptr+2;
                    break;

                case 'D':
                    pClient->database = ptr+2;
                    break;
            }
        }

        return;
    }
}
//=====
//
// Function name: GetArgsClientUsage
```

```

case 'A':
    pClient->admin_database = ptr+2;
    break;

case 'U':
    pClient->user = ptr+2;
    break;

case 'P':
    pClient->password = ptr+2;
    break;

case 'c':
    pClient->num_threads = atol(ptr+2);
    break;

case 'p':
    pClient->pack_size = atol(ptr+2);
    break;

case 'd':
    pClient->disable_delivery_resfiles =
    atol(ptr+2);
    break;

case 's':
    pClient->synch_servername = ptr+2;
    break;

case 'q':
    pClient->enable_qj = atol(ptr+2);
    break;

default:
    GetArgsClientUsage();
    exit(-1);
    break;
    }
}

return;
}
//=====
//
// Function name: GetArgsClientUsage
```

```
//
//=====
void GetArgsClientUsage()
{
#ifdef DEBUG
    printf("[%d]DBG: Entering GetArgsClientUsage()\n", (int) GetCurrentThreadId());
#endif

    printf("CLIENT:\n\n");
    printf("Parameter                Default\n");
    printf("-----\n");
    printf("-S Server                %s\n", SERVER);
    printf("-D Database                %s\n", DATABASE);
    printf("-A Admin Database        %s\n", ADMIN_DATABASE);
    printf("-U Username                %s\n", USER);
    printf("-P Password                %s\n", PASSWORD);
    printf("-c Number of User Connctions    %ld\n", (long) NUM_THREADS);
    printf("-p TDS Packet Size            %ld\n", (long) DEFCLPACKSIZE);
    printf("-d Disable Delivery Result Files (no = 0, yes = 1) %ld\n", (long)
DISABLE_DELIVERY_RESFILES);
    printf("-s Master Driver Servername    %s\n", SYNCH_SERVERNAME);

    printf("\nNote: Command line switches are case sensitive.\n");

    exit(0);
}

//=====
//
// Function name: GetArgsDelivery
//
//=====
void GetArgsDelivery(int argc, char **argv, DELIVERY_ARGS *pDelivery)
{
    int            i;
    char        *ptr;

#ifdef DEBUG
    printf("[%d]DBG: Entering GetArgsDelivery()\n", (int) GetCurrentThreadId());
#endif

    pDelivery->pipe_num = 0;

    /* check for 1 or more command line args */

```

```

if ( argc != 1 )
    {
        for (i = 1; i < argc; ++i)
        {
            if (argv[i][0] != '-' && argv[i][0] != '/')
            {
                printf("\nUnrecognized command");
                GetArgsClientUsage();
                exit(1);
            }

            ptr = argv[i];

            switch (ptr[1])
            {
                case 'p':
                    pDelivery->pipe_num = (long) atol(ptr+2)
                    break;

                default:
                    printf("ERROR: No pipe number
specified.");
                    exit(-1);
                    break;
            }
        }

        return;
    }

//=====
//
// Function name: GetArgsSQLStat
//
//=====

void GetArgsSQLStat(int argc, char **argv, SQLSTAT_ARGS *pargs)
{
    int            i;
    char        *ptr;

    /* init args struct with some useful values */
    pargs->server            = SERVER;
    pargs->user                = USER;
    pargs->password            = PASSWORD;
    pargs->admin_database     = ADMIN_DATABASE;

```



```

*                               Copyright Microsoft, 1996
*
*   PURPOSE:      Random number generation functions for Microsoft TPC-C
Benchmark Kit
*   Author:       Damien Lindauer
*                               damienl@Microsoft.com
*/

// Includes
#include "tpcc.h"
#include "math.h"

// Defines
#define A      16807
#define M      2147483647
#define Q      127773 /* M div A */
#define R      2836 /* M mod A */
#define Thread __declspec(thread)

// Globals
long Thread Seed = 0; /* thread local seed */

/*****
*
* random -
* Implements a GOOD pseudo random number generator. This generator
* will/should? run the complete period before repeating.
*
* Copied from:
* Random Numbers Generators: Good Ones Are Hard to Find.
* Communications of the ACM - October 1988 Volume 31 Number 10
*
* Machine Dependencies:
* long must be 2 ^ 31 - 1 or greater.
*
*****/

/*****
* seed - load the Seed value used in irand and drand. Should be used before
* first call to irand or drand.
*****/

void seed(long val)
{
#ifdef DEBUG
printf("[%d]DBG: Entering seed()...\n", (int) GetCurrentThreadId());
printf("Old Seed %ld New Seed %ld\n",Seed, val);

```

```

#endif

if ( val < 0 )
    val = abs(val);

Seed = val;
}

/*****
*
* irand - returns a 32 bit integer pseudo random number with a period of
* 1 to 2 ^ 32 - 1.
*
* parameters:
* none.
*
* returns:
* 32 bit integer - defined as long ( see above ).
*
* side effects:
* seed get recomputed.
*****/

long irand()
{
    register long s; /* copy of seed */
    register long test; /* test flag */
    register long hi; /* tmp value for speed */
    register long lo; /* tmp value for speed */

#ifdef DEBUG
printf("[%d]DBG: Entering irand()...\n", (int) GetCurrentThreadId());
#endif

    s = Seed;
    hi = s / Q;
    lo = s % Q;

    test = A * lo - R * hi;
    if ( test > 0 )
        Seed = test;
    else
        Seed = test + M;

    return( Seed );
}

```



```

/******
 *
 * drand - returns a double pseudo random number between 0.0 and 1.0.
 * See irand.
 *****/
double drand()
{
#ifdef DEBUG
    printf("[%d]DBG: Entering drand()...\n", (int) GetCurrentThreadId());
#endif

    return( (double)irand() / 2147483647.0);
}

//=====
// Function : RandomNumber
//
// Description:
//=====
long RandomNumber(long lower, long upper)
{
    long rand_num;

#ifdef DEBUG
    printf("[%d]DBG: Entering RandomNumber()...\n", (int) GetCurrentThreadId());
#endif

    if ( upper == lower ) /* pgd 08-13-96 perf enhancement */
        return lower;

    upper++;

    if ( upper <= lower )
        rand_num = upper;
    else
        rand_num = lower + irand() % (upper - lower); /* pgd 08-13-96 perf
enhancement */

#ifdef DEBUG
    printf("[%d]DBG: RandomNumber between %ld & %ld ==> %ld\n",
           (int) GetCurrentThreadId(), lower, upper, rand_num);
#endif
}

```

```

    return rand_num;
}

#if 0
//Original code pgd 08/13/96
long RandomNumber(long lower,
                  long upper)
{
    long rand_num;

#ifdef DEBUG
    printf("[%d]DBG: Entering RandomNumber()...\n", (int) GetCurrentThreadId());
#endif

    upper++;

    if ((upper <= lower))
        rand_num = upper;
    else
        rand_num = lower + irand() % ((upper > lower) ? upper - lower : upper);

#ifdef DEBUG
    printf("[%d]DBG: RandomNumber between %ld & %ld ==> %ld\n",
           (int) GetCurrentThreadId(), lower, upper, rand_nurr);
#endif

    return rand_num;
}
#endif

//=====
// Function : NURand
//
// Description:
//=====
long NURand(int iConst,
            long x,
            long y,
            long C)
{
    long rand_num;
}

```

```

#ifdef DEBUG
    printf("[%d]DBG: Entering NURand(...)\n", (int) GetCurrentThreadId());
#endif

    rand_num = (((RandomNumber(0,iConst) | RandomNumber(x,y)) + C) % (y-x+1))+x;

#ifdef DEBUG
    printf("[%d]DBG: NURand: num = %d\n", (int) GetCurrentThreadId(), rand_num);
#endif

    return rand_num;
}

```

B.28 util.c

```

// TPC-C Benchmark Kit
//
// Module: UTIL.C
// Author: DamienL

// Includes
#include "tpcc.h"

//=====
//
// Function name: UtilSleep
//
//=====

void UtilSleep(long delay)
{
#ifdef DEBUG
    printf("[%d]DBG: Entering UtilSleep()\n", (int) GetCurrentThreadId());
#endif

#ifdef DEBUG
    printf("[%d]DBG: Sleeping for %ld seconds...\n", (int) GetCurrentThreadId(), delay);
#endif

    Sleep(delay * 1000);
}

//=====

```

```

//
// Function name: UtilSleep
//
//=====

void UtilSleepMs(long delay)
{
#ifdef DEBUG
    printf("[%d]DBG: Entering UtilSleepMs()\n", (int) GetCurrentThreadId());
#endif

#ifdef DEBUG
    printf("[%d]DBG: Sleeping for %ld milliseconds...\n", (int) GetCurrentThreadId(), delay);
#endif

    Sleep(delay);
}

//=====
//
// Function name: UtilPrintNewOrder
//
//=====

void UtilPrintNewOrder(NEW_ORDER_DATA *pNewOrder)
{
    int i;

#ifdef DEBUG
    printf("[%d]DBG: Entering UtilPrintNewOrder()\n", (int) GetCurrentThreadId());
#endif

    EnterCriticalSection(&ConsoleCritSec);

    printf("\n[%04d]tNewOrder Transaction\n\n", (int) GetCurrentThreadId());

    printf("Warehouse: %d\n"
           "District: %d\n"
           "Date: %02d/%02d/%04d %02d:%02d:%02d\n\n"
           "Customer Number: %d\n"
           "Customer Name: %s\n"
           "Customer Credit: %s\n"
           "Cusotmer Discount: %02.2f%%\n\n"
           "Order Number: %d\n"
           "Warehouse Tax: %02.2f%%\n\n"
           "District Tax: %02.2f%%\n\n"
           "Number of Order Lines: %d\n\n",

```

```

        (int) pNewOrder->w_id,
        (int) pNewOrder->d_id,
        (char *) pNewOrder->o_entry_d.month,
        (char *) pNewOrder->o_entry_d.day,
        (char *) pNewOrder->o_entry_d.year,
        (char *) pNewOrder->o_entry_d.hour,
        (char *) pNewOrder->o_entry_d.minute,
        (char *) pNewOrder->o_entry_d.second,
        (int) pNewOrder->c_id,
        (char *) pNewOrder->c_last,
        (char *) pNewOrder->c_credit,
        (float) pNewOrder->c_discount,
        (int) pNewOrder->o_id,
        (float) pNewOrder->w_tax,
        (float) pNewOrder->d_tax,
        (int) pNewOrder->o_ol_cnt);

    printf("Supp_W Item_Id Item Name          Qty Stock B/G Price  Amount  \n");
    printf("-----\n");

    for (i=0;i < pNewOrder->o_ol_cnt;i++)
    {
        printf("%04ld %06ld %24s %02ld %08ld %1s %8.2f %9.2f\n",
            (int) pNewOrder->Ol[i].ol_supply_w_id,
            (int) pNewOrder->Ol[i].ol_i_id,
            (char *) pNewOrder->Ol[i].ol_i_name,
            (int) pNewOrder->Ol[i].ol_quantity,
            (int) pNewOrder->Ol[i].ol_stock,
            (char *) pNewOrder->Ol[i].ol_brand_generic,
            (float) pNewOrder->Ol[i].ol_i_price,
            (float) pNewOrder->Ol[i].ol_amount);
    }

    printf("\nTotal: $%05.2f\n",
        (float) pNewOrder->total_amount);

    printf("Execution Status: %s\n",
        (char *) pNewOrder->execution_status);

    LeaveCriticalSection(&ConsoleCritSec);
}

//=====
//
// Function name: UtilPrintPayment
//

```

```

//=====
void UtilPrintPayment(PAYMENT_DATA *pPayment)
{
    char tmp_data[201];
    char data_line_1[51];
    char data_line_2[51];
    char data_line_3[51];
    char data_line_4[51];

#ifdef DEBUG
    printf("[%ld]DBG: Entering UtilPrintPayment()\n", (int) GetCurrentThreadId());
#endif

    EnterCriticalSection(&ConsoleCritSec);

    printf("\n[%04ld]tPayment Transaction\n\n", (int) GetCurrentThreadId());

    printf("Date: %02ld/%02ld/%04ld %02ld:%02ld:%02ld\n",
        (int) pPayment->h_date.month,
        (int) pPayment->h_date.day,
        (int) pPayment->h_date.year,
        (int) pPayment->h_date.hour,
        (int) pPayment->h_date.minute,
        (int) pPayment->h_date.second);

    printf("Warehouse: %ld\n",
        "District: %ld\n",
        (int) pPayment->w_id,
        (int) pPayment->d_id);

    printf("Warehouse Address Street 1: %s\n",
        "Warehouse Address Street 2: %s\n",
        (char *) pPayment->w_street_1,
        (char *) pPayment->w_street_2);

    printf("Warehouse Address City: %s\n",
        "Warehouse Address State: %s\n",
        "Warehouse Address Zip: %s\n",
        (char *) pPayment->w_city,
        (char *) pPayment->w_state,
        (char *) pPayment->w_zip);

    printf("District Address Street 1: %s\n",
        "District Address Street 2: %s\n",
        (char *) pPayment->d_street_1,
        (char *) pPayment->d_street_2);
}

```

```

printf("District Address City: %s\n"
      "District Address State: %s\n"
      "District Address Zip: %s\n\n",
      (char *) pPayment->d_city,
      (char *) pPayment->d_state,
      (char *) pPayment->d_zip);

printf("Customer Number: %ld\n"
      "Customer Warehouse: %ld\n"
      "Customer District: %ld\n",
      (int) pPayment->c_id,
      (int) pPayment->c_w_id,
      (int) pPayment->c_d_id);

printf("Customer Name: %s %s %s\n"
      "Customer Since: %02ld-%02ld-%04ld\n",
      (char *) pPayment->c_first,
      (char *) pPayment->c_middle,
      (char *) pPayment->c_last,
      (int) pPayment->c_since.month,
      (int) pPayment->c_since.day,
      (int) pPayment->c_since.year);

printf("Customer Address Street 1: %s\n"
      "Customer Address Street 2: %s\n"
      "Customer Address City: %s\n"
      "Customer Address State: %s\n"
      "Customer Address Zip: %s\n"
      "Customer Phone Number: %s\n\n"
      "Customer Credit: %s\n"
      "Customer Discount: %02.2f%%\n",
      (char *) pPayment->c_street_1,
      (char *) pPayment->c_street_2,
      (char *) pPayment->c_city,
      (char *) pPayment->c_state,
      (char *) pPayment->c_zip,
      (char *) pPayment->c_phone,
      (char *) pPayment->c_credit,
      (double) pPayment->c_discount);

printf("Amount Paid: $%04.2f\n"
      "New Customer Balance: $%10.2f\n",
      (float) pPayment->h_amount,
      (double) pPayment->c_balance);

printf("Credit Limit: $%10.2f\n\n",
      (double) pPayment->c_credit_lim);

```

```

if (strcmp(pPayment->c_data, "") != 0)
{
    strcpy(tmp_data, pPayment->c_data);
    strncpy(data_line_1, tmp_data, 50); data_line_1[50] = '\0';
    strncpy(data_line_2, &tmp_data[50], 50); data_line_2[50] = '\0';
    strncpy(data_line_3, &tmp_data[100], 50); data_line_3[50] = '\0';
    strncpy(data_line_4, &tmp_data[150], 50); data_line_4[50] = '\0';
}
else
{
    strcpy(data_line_1, ""); strcpy(data_line_2, "");
    strcpy(data_line_3, ""); strcpy(data_line_4, "");
}

printf("-----\n");
printf("Customer Data: l%50s\n", data_line_1);
printf("                l%50s\n", data_line_2);
printf("                l%50s\n", data_line_3);
printf("                l%50s\n", data_line_4);
printf("-----\n\n");

printf("Execution Status: %s\n\n",
      (char *) pPayment->execution_status);

LeaveCriticalSection(&ConsoleCritSec);
}

//=====
//
// Function name: UtilPrintOrderStatus
//
//=====

void UtilPrintOrderStatus(ORDER_STATUS_DATA *pOrderStatus)
{
    int i;

#ifdef DEBUG
    printf("[%ld]DBG: Entering UtilPrintOrderStatus()\n", (int) GetCurrentThreadId());
#endif

    EnterCriticalSection(&ConsoleCritSec);

    printf("\n[%04ld]tOrder-Status Transaction\n\n", (int) GetCurrentThreadId());

```

```

printf("Warehouse: %ld\n"
      "District: %ld\n\n",
      (int) pOrderStatus->w_id,
      (int) pOrderStatus->d_id);

printf("Customer Number: %ld\n"
      "Customer Name: %s %s %s\n\n",
      (int) pOrderStatus->c_id,
      (char *) pOrderStatus->c_first,
      (char *) pOrderStatus->c_middle,
      (char *) pOrderStatus->c_last);

printf("Customer Balance: $%5.2f\n\n",
      (double) pOrderStatus->c_balance);

printf("Order Number: %ld\n"
      "Entry Date: %02ld/%02ld/%04ld %02ld:%02ld:%02ld\n"
      "Carrier Number: %ld\n\n"
      "Number of order lines: %ld\n\n",
      (int) pOrderStatus->o_id,
      (int) pOrderStatus->o_entry_d.month,
      (int) pOrderStatus->o_entry_d.day,
      (int) pOrderStatus->o_entry_d.year,
      (int) pOrderStatus->o_entry_d.hour,
      (int) pOrderStatus->o_entry_d.minute,
      (int) pOrderStatus->o_entry_d.second,
      (int) pOrderStatus->o_carrier_id,
      (int) pOrderStatus->o_ol_cnt);

printf("Supply-W Item-Id Delivery-Date Qty Amount \n");
printf ("----- ----- ----- --- -----\n");

for (i=0; i < pOrderStatus->o_ol_cnt; i++)
{
    printf("%04ld %06ld %02ld/%02ld/%04ld %02ld %9.2f\n",
          (int) pOrderStatus-
>OIOrderStatusData[i].ol_supply_w_id,
          (int) pOrderStatus->OIOrderStatusData[i].ol_i_id,
          (int) pOrderStatus-
>OIOrderStatusData[i].ol_delivery_d.month,
          (int) pOrderStatus-
>OIOrderStatusData[i].ol_delivery_d.day,
          (int) pOrderStatus-
>OIOrderStatusData[i].ol_delivery_d.year,
          (int) pOrderStatus->OIOrderStatusData[i].ol_quantity,
          (double) pOrderStatus-
>OIOrderStatusData[i].ol_amount);

```

```

}

if (pOrderStatus->o_ol_cnt == 0)
    printf("\nNo Order-Status items.\n\n");

printf("\nExecution Status: %s\n\n",
      (char *) pOrderStatus->execution_status);

LeaveCriticalSection(&ConsoleCritSec);
}

//=====
//
// Function name: UtilPrintDelivery
//
//=====

void UtilPrintDelivery(DELIVERY_DATA *pQueuedDelivery)
{
#ifdef DEBUG
    printf("[%ld]DBG: Entering UtilPrintDelivery()\n", (int) GetCurrentThreadId());
#endif

    EnterCriticalSection(&ConsoleCritSec);

    printf("\n[%04ld]tDelivery Transaction\n\n", (int) GetCurrentThreadId());

    printf("Warehouse: %ld\n", (int) pQueuedDelivery->w_id);

    printf("Carrier Number: %ld\n", (int) pQueuedDelivery->o_carrier_id);

    printf("Execution Status: %s\n\n", (char *) pQueuedDelivery->execution_status);

    LeaveCriticalSection(&ConsoleCritSec);
}

//=====
//
// Function name: UtilPrintStockLevel
//
//=====

void UtilPrintStockLevel(STOCK_LEVEL_DATA *pStockLevel)

```

```

{
#ifdef DEBUG
printf("[%d]DBG: Entering UtilPrintStockLevel()\n", (int) GetCurrentThreadId());
#endif

EnterCriticalSection(&ConsoleCritSec);

printf("\n[%04d]\tStock-Level Transaction\n\n", (int) GetCurrentThreadId());

printf("Warehouse: %d\nDistrict: %d\n",
      (int) pStockLevel->w_id,
      (int) pStockLevel->d_id);

printf("Stock Level Threshold: %d\n\n", (int) pStockLevel->thresh_hold);

printf("Low Stock Count: %d\n\n", (int) pStockLevel->low_stock);

printf("Execution Status: %s\n\n", (char *) pStockLevel->execution_status);

LeaveCriticalSection(&ConsoleCritSec);
}

//=====
//
// Function name: UtilError
//
//=====
void UtilError(long threadid, char * header, char *msg)
{
#ifdef DEBUG
printf("[%d]DBG: Entering UtilError()\n", (int) GetCurrentThreadId());
#endif

printf("[%d] %s: %s\n", (int) threadid, header, msg);
}

//=====
//
// Function name: UtilFatalError
//
//=====
void UtilFatalError(long threadid, char * header, char *msg)
{

```

```

#ifdef DEBUG
printf("[%d]DBG: Entering UtilFatalError()\n", (int) GetCurrentThreadId());
#endif

printf("[Thread: %d]... %s: %s\n", (int) threadid, header, msg);
exit(-1);
}

//=====
//
// Function name: UtilStrCpy
//
//=====
void UtilStrCpy(char * pDest, char * pSrc, int n)
{
#ifdef DEBUG
printf("[%d]DBG: Entering UtilStrCpy()\n", (int) GetCurrentThreadId());
#endif

strncpy(pDest, pSrc, n);
pDest[n] = '\0';
}

#ifdef USE_CONMON
//=====
//
// Function name: WriteConsoleString
//
//=====
void WriteConsoleString(HANDLE hConMon, char *str, short x, short y, short color, BOOL p;
{
COORD dwWriteCoord = {0, 0};
DWORD cCharsWritten;
LPVOID dummy;
int len, i;

#ifdef DEBUG
printf("[%d]DBG: Entering WriteConsoleString()\n", (int) GetCurrentThreadId());
#endif

dwWriteCoord.X = x;
dwWriteCoord.Y = y;

if (pad)
{

```

```

        len = strlen(str);
        if (len < CON_LINE_SIZE)
        {
            for(i=1;i<CON_LINE_SIZE-len;i++)
            {
                strcat(str, " ");
            }
        }
        EnterCriticalSection(&ConsoleCritSec);

        switch (color)
        {
            case YELLOW:
                SetConsoleTextAttribute(hConMon,
                    FOREGROUND_INTENSITY |
                    FOREGROUND_GREEN | FOREGROUND_RED | BACKGROUND_BLUE);
                break;

            case RED:
                SetConsoleTextAttribute(hConMon,
                    FOREGROUND_INTENSITY |
                    FOREGROUND_RED | BACKGROUND_BLUE);
                break;

            case GREEN:
                SetConsoleTextAttribute(hConMon,
                    FOREGROUND_INTENSITY |
                    FOREGROUND_GREEN | BACKGROUND_BLUE);
                break;
        }

        SetConsoleCursorPosition(hConMon, dwWriteCoord);
        WriteConsole(hConMon, str, strlen(str), &cCharsWritten, dummy);

        LeaveCriticalSection(&ConsoleCritSec);
    }
#endif

```

```

//=====
//
// Function name: AddDeliveryQueueNode
//
//=====

```

```

BOOL AddDeliveryQueueNode(DELIVERY_PTR node_to_add)
{
    DELIVERY_PTR    local_node;
#ifdef DEBUG
    DELIVERY_PTR    ptrtmp;
    short           i;
#endif

    EnterCriticalSection(&QueuedDeliveryCritSec);

    if ((local_node = malloc(sizeof(struct delivery_node)) ) == NULL)
    {
        printf("ERROR: problem allocating memory for delivery queue.\n");
        exit(-1);
    }
    else
    {
        memcpy(local_node, node_to_add, sizeof (struct delivery_node));

        if (queued_delivery_cnt == 0)
        {
            delivery_head = local_node
            delivery_head->next_delivery = NULL;
            delivery_tail = delivery_head;
        }
        else
        {
            local_node->next_delivery = NULL;
            delivery_tail->next_delivery = local_node;
            delivery_tail = local_node;
        }
    }

    queued_delivery_cnt++;

#ifdef DEBUG
    i=0;
    printf("Add to delivery list: %ld\n",queued_delivery_cnt);
    ptrtmp=delivery_head;
    while (ptrtmp != NULL)
    {
        i++;
        printf("%ld - w_id %ld - o_carrier_id %ld - queue_time %d/%d/%d
%d:%d:%d:%d\n",
            i, ptrtmp->w_id, ptrtmp->o_carrier_id,
            ptrtmp->queue_time.wMonth,
            ptrtmp->queue_time.wDay,

```

```

                ptrtmp->queue_time.wYear,
                ptrtmp->queue_time.wHour,
                ptrtmp->queue_time.wMinute,
                ptrtmp->queue_time.wSecond,
                ptrtmp->queue_time.wMilliseconds);
        ptrtmp=ptrtmp->next_delivery;
    }
#endif

    LeaveCriticalSection(&QueuedDeliveryCritSec);

    return TRUE;
}

//=====
//
// Function name: GetDeliveryQueueNode
//
//=====

BOOL GetDeliveryQueueNode(DELIVERY_PTR node_to_get)
{
    DELIVERY_PTR    local_node;
    BOOL            rc;
#ifdef DEBUG
    DELIVERY_PTR    ptrtmp;
    short           i;
#endif

    EnterCriticalSection(&QueuedDeliveryCritSec);

    if (queued_delivery_cnt == 0)
    {
#ifdef DEBUG
        printf("No delivery nodes found.\n");
#endif
        rc = FALSE;
    }
    else
    {
        memcopy(node_to_get, delivery_head, sizeof(struct delivery_node));

                ptrtmp->queue_time.wYear,
                ptrtmp->queue_time.wHour,
                ptrtmp->queue_time.wMinute,
                ptrtmp->queue_time.wSecond,
                ptrtmp->queue_time.wMilliseconds);
        ptrtmp=ptrtmp->next_delivery;
    }
}

//=====
// Function name: GetDeliveryQueueNode
//
//=====

if (queued_delivery_cnt == 1)
{
    free(delivery_head);
    delivery_head = NULL;
    queued_delivery_cnt = 0;
}
else
{
    local_node = delivery_head;
    delivery_head = delivery_head->next_delivery;
    free(local_node);
    queued_delivery_cnt--;
}

#ifdef DEBUG
i=0;
printf("Get from delivery list: %ld\n",queued_delivery_cnt);
ptrtmp=delivery_head;
while (ptrtmp != NULL)
{
    i++;
    printf("%ld - w_id %ld - o_carrier_id %ld - queue_time
%d/%d/%d %d:%d:%d:%d\n",
                i, ptrtmp->w_id, ptrtmp->o_carrier_id,
                ptrtmp->queue_time.wMonth,
                ptrtmp->queue_time.wDay,
                ptrtmp->queue_time.wYear,
                ptrtmp->queue_time.wHour,
                ptrtmp->queue_time.wMinute,
                ptrtmp->queue_time.wSecond,
                ptrtmp->queue_time.wMilliseconds);

        ptrtmp=ptrtmp->next_delivery;
}
#endif

rc = TRUE;
}

LeaveCriticalSection(&QueuedDeliveryCritSec);

return rc;
}

//=====

```



```
//  
// Function name: WriteDeliveryString  
//  
//=====
```

```
void WriteDeliveryString(char buf[255])  
{  
    DWORD bytesWritten;  
    DWORD retCode;  
  
#ifdef DEBUG  
    printf("[%ld]DBG: Entering UtilDeliveryMsg()\n", (int) GetCurrentThreadId());  
#endif  
  
    EnterCriticalSection(&WriteDeliveryCritSec);  
  
    retCode = WriteFile (hDeliveryMonPipe, buf, PLEASE_WRITE,  
                        &bytesWritten, NULL);  
  
    LeaveCriticalSection(&WriteDeliveryCritSec);  
  
}
```

Appendix C Benchcraft RTE Configuration

Profile: 1098

File Path: C:\bench106\1098.pro

Version: 1.0.1

Number of Engines: 9

Name: DRIVER1
Description: herbert driver engine
Directory: c:\benchcrflogs\bench.log
Machine: HERBERT
Parameter Set: spec3.3
Index: 0
Seed: 87043
Configured Users: 1220
Pipe Name: DRIVER11562454610
Connect Rate: 200
Start Rate: 0
CLIENT_NURAND: 233
CPU: 0

Name: DRIVER2
Description: Hill driver engine
Directory: c:\benchcrflogs\bench.log
Machine: HILL
Parameter Set: spec3.3
Index: 100000
Seed: 87043
Configured Users: 1220
Pipe Name: DRIVER21562509179
Connect Rate: 200
Start Rate: 0
CLIENT_NURAND: 233
CPU: 0

Name: DRIVER3
Description: Prost driver engine
Directory: c:\benchcrflogs\bench.log
Machine: PROST
Parameter Set: spec3.3
Index: 200000
Seed: 87043
Configured Users: 1220
Pipe Name: DRIVER31562544880

Connect Rate: 200
Start Rate: 0
CLIENT_NURAND: 233
CPU: 0

Name: DRIVER4
Description: Dunlop driver engine
Directory: c:\benchcrflogs\bench.log
Machine: DUNLOP
Parameter Set: spec3.3
Index: 300000
Seed: 87043
Configured Users: 1220
Pipe Name: DRIVER41562629512
Connect Rate: 200
Start Rate: 0
CLIENT_NURAND: 233
CPU: 0

Name: DRIVER5
Description: Senna driver engine
Directory: c:\benchcrflogs\bench.log
Machine: SENNA
Parameter Set: spec3.3
Index: 400000
Seed: 87043
Configured Users: 1220
Pipe Name: DRIVER5252107451
Connect Rate: 200
Start Rate: 0
CLIENT_NURAND: 233
CPU: 0

Name: DRIVER6
Description: Senna number2
Directory: c:\benchcrflogs\bench2.log
Machine: SENNA
Parameter Set: spec3.3
Index: 500000
Seed: 87043
Configured Users: 1220
Pipe Name: DRIVER65114684
Connect Rate: 200
Start Rate: 0
CLIENT_NURAND: 233
CPU: 0

Name: DRIVER7

Description: prost2
 Directory: c:\benchcrf\logs\bench2.log
 Machine: PROST
 Parameter Set: spec3.3
 Index: 600000
 Seed: 87043
 Configured Users: 1220
 Pipe Name: DRIVER731455801
 Connect Rate: 200
 Start Rate: 0
 CLIENT_NURAND: 233
 CPU: 0

Name: DRIVER8
 Description: Dunlop 2
 Directory: c:\benchcrf\logs\bench2.log
 Machine: DUNLOP
 Parameter Set: spec3.3
 Index: 700000
 Seed: 87043
 Configured Users: 1220
 Pipe Name: DRIVER831518661
 Connect Rate: 200
 Start Rate: 0
 CLIENT_NURAND: 233
 CPU: 0

Name: DRIVER9
 Description: Hill 2
 Directory: c:\benchcrf\logs\bench2.log
 Machine: HILL
 Parameter Set: spec3.3
 Index: 800000
 Seed: 87043
 Configured Users: 1220
 Pipe Name: DRIVER921968018
 Connect Rate: 200
 Start Rate: 0
 CLIENT_NURAND: 233
 CPU: 0

Number of User groups: 9

Driver Engine: DRIVER1
 IIS Server: FLESH
 SQL Server: garbage
 User: sa
 Protocol: Html

w_id Range: 1 - 122
 w_id Max Warehouse: 1098
 Scale: Normal
 User Count: 1220
 District id: 1
 Scale Down: No

Driver Engine: DRIVER2
 IIS Server: COFFEE
 SQL Server: garbage
 User: sa
 Protocol: Html
 w_id Range: 123 - 244
 w_id Max Warehouse: 1098
 Scale: Normal
 User Count: 1220
 District id: 1
 Scale Down: No

Driver Engine: DRIVER3
 IIS Server: HAPPY
 SQL Server: garbage
 User: sa
 Protocol: Html
 w_id Range: 245 - 366
 w_id Max Warehouse: 1098
 Scale: Normal
 User Count: 1220
 District id: 1
 Scale Down: No

Driver Engine: DRIVER4
 IIS Server: TEA
 SQL Server: garbage
 User: sa
 Protocol: Html
 w_id Range: 367 - 488
 w_id Max Warehouse: 1098
 Scale: Normal
 User Count: 1220
 District id: 1
 Scale Down: No

Driver Engine: DRIVER5
 IIS Server: BEER
 SQL Server: garbage
 User: sa
 Protocol: Html

w_id Range: 489 - 610
 w_id Max Warehouse: 1098
 Scale: Normal
 User Count: 1220
 District id: 1
 Scale Down: No

Driver Engine: DRIVER6
 IIS Server: COLA
 SQL Server: garbage
 User: sa
 Protocol: Html
 w_id Range: 611 - 732
 w_id Max Warehouse:1098
 Scale: Normal
 User Count: 1220
 District id: 1
 Scale Down: No

Driver Engine: DRIVER7
 IIS Server: booze
 SQL Server: garbage
 User: sa
 Protocol: Html
 w_id Range: 733 - 854
 w_id Max Warehouse: 1098
 Scale: Normal
 User Count: 1220
 District id: 1
 Scale Down: No

Driver Engine: DRIVER8
 IIS Server: vodka
 SQL Server: garbage
 User: sa
 Protocol: Html
 w_id Range: 855 - 976
 w_id Max Warehouse: 1098
 Scale: Normal
 User Count: 1220
 District id: 1
 Scale Down: No

Driver Engine: DRIVER9
 IIS Server: METHS
 SQL Server: garbage
 User: sa
 Protocol: Html

w_id Range: 977 - 1098
 w_id Max Warehouse: 1098
 Scale: Normal
 User Count: 1220
 District id: 1
 Scale Down: No

Number of Parameter Sets: 2

~Default

Default Parameter Set

	Txn	Think	Key	RT	RT	Menu	
	Weight	Time	Time	Delay	Fence	Delay	
0.10	New Order	10.00		12.05	18.01	0.10	5.00
0.10	Payment	10.00		12.05	3.01	0.10	5.00
0.10	Delivery	1.00	5.05	2.01	0.10	5.00	0.10
0.10	Stock Level	1.00		5.05	2.01	0.10	20.00
0.10	Order Status	1.00		10.05	2.01	0.10	5.00

spec3.3

tpcc spec version 33 transaction mix

	Txn	Think	Key	RT	RT	Menu	
	Weight	Time	Time	Delay	Fence	Delay	
0.10	New Order	44.50		12.10	18.01	0.10	5.00
0.10	Payment	43.20		12.10	3.01	0.10	5.00
0.10	Delivery	4.10	5.10	2.01	0.10	5.00	0.10
0.10	Stock Level	4.10		5.05	2.01	0.10	20.00
0.10	Order Status	4.10		10.20	2.01	0.10	5.00

Appendix D Tunable Parameters

D.1 Microsoft Windows NT Server 4.0

[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\MSSQLServer\MSSQLServer\Parameters]

"SQLArg0"="-dC:\MSSQL\DATA\MASTER.DAT"
 "SQLArg1"="-eC:\MSSQL\LOG\ERRORLOG"

D.2 Microsoft Windows NT Client 4.0

[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\TPCC]

"PATH"="C:\InetPub\wwwroot\
 "LOG"="OFF"
 "NumberOfDeliveryThreads"="30"
 "MaximumWarehouses"="1010"
 "BackoffDelay"="500"
 "DeadlockRetry"="3"
 "MaxConnections"="2000"
 "QueueSlots"="3000"
 "ConnectionPooling"="OFF"
 "ConnectionPoolRetryTime"="500"
 "LastInstalledVersion"="DBLIB"
 "DEBUG"="OFF"

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\InetInfo]

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\InetInfo\Parameters]

"BandwidthLevel"=dword:ffffff
 "ListenBackLog"=dword:00000800
 "PoolThreadsLimit"=dword:00000100
 "ThreadTimeout"=dword:00015180
 "PoolThreadLimit"=dword:00000100

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\InetInfo\Parameters\Filter]

"FilterType"=dword:00000000
 "NumGrantSites"=dword:00000000
 "NumDenySites"=dword:00000000

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\InetInfo\Parameters\Mime
]
 "text/html,htm,,h"=""
 "image/gif,gif,,g"=""
 "image/jpeg,jpg,,:"=""
 "text/plain,txt,,0"=""
 "text/html,html,,h"=""
 "image/jpeg,jpeg,,:"=""
 "image/jpeg,jpe,,:"=""
 "image/bmp,bmp,,:"=""
 "application/octet-stream,*,,5"=""
 "application/pdf,pdf,,5"=""
 "application/octet-stream,bin,,5"=""
 "application/oda,oda,,5"=""
 "application/zip,zip,,9"=""
 "application/rtf,rtf,,5"=""
 "application/postscript,ps,,5"=""
 "application/postscript,ai,,5"=""
 "application/postscript,eps,,5"=""
 "application/mac-binhex40,hqx,,4"=""
 "application/msword,doc,,5"=""
 "application/msword,dot,,5"=""
 "application/winhelp,hlp,,5"=""
 "video/mpeg,mpeg,,:"=""
 "video/mpeg,mpg,,:"=""
 "video/mpeg,mpe,,:"=""
 "video/x-msvideo,avi,,<"=""
 "video/quicktime,qt,,:"=""
 "video/quicktime,mov,,:"=""
 "video/x-sgi-movie,movie,,<"=""
 "x-world/x-vrml,wrl,,5"=""
 "x-world/x-vrml,xaf,,5"=""
 "x-world/x-vrml,xof,,5"=""
 "x-world/x-vrml,flr,,5"=""
 "x-world/x-vrml,wrz,,5"=""
 "application/x-director,dcr,,5"=""
 "application/x-director,dir,,5"=""
 "application/x-director,dxr,,5"=""
 "image/cis-cod,cod,,5"=""
 "image/x-cmx,cmx,,5"=""
 "application/envoy,evy,,5"=""
 "application/x-msaccess,mdb,,5"=""
 "application/x-mscardfile,crd,,5"=""
 "application/x-msclip,clip,,5"=""
 "application/octet-stream,exe,,5"=""
 "application/x-msexcel,xla,,5"=""
 "application/x-msexcel,xlc,,5"=""
 "application/x-msexcel,xlm,,5"=""

```

"application/x-msexcel,xls,,5"=""
"application/x-msexcel,xlt,,5"=""
"application/x-msexcel,xlw,,5"=""
"application/x-msmediaview,m13,,5"=""
"application/x-msmediaview,m14,,5"=""
"application/x-msmoney,mny,,5"=""
"application/x-mspowerpoint,ppt,,5"=""
"application/x-msproject,mpp,,5"=""
"application/x-mspublisher,pub,,5"=""
"application/x-msterminal,trm,,5"=""
"application/x-msworks,wks,,5"=""
"application/x-mswrite,wri,,5"=""
"application/x-msmetafile,wmf,,5"=""
"application/x-csh,csh,,5"=""
"application/x-dvi,dvi,,5"=""
"application/x-hdf,hdf,,5"=""
"application/x-latex,latex,,5"=""
"application/x-netcdf,nc,,5"=""
"application/x-netcdf,cdf,,5"=""
"application/x-sh,sh,,5"=""
"application/x-tcl,tcl,,5"=""
"application/x-tex,tex,,5"=""
"application/x-texinfo,texinfo,,5"=""
"application/x-texinfo,texi,,5"=""
"application/x-troff,t,,5"=""
"application/x-troff,tr,,5"=""
"application/x-troff,roff,,5"=""
"application/x-troff-man,man,,5"=""
"application/x-troff-me,me,,5"=""
"application/x-troff-ms,ms,,5"=""
"application/x-wais-source,src,,7"=""
"application/x-bcpio,bcpio,,5"=""
"application/x-cpio,cpio,,5"=""
"application/x-gtar,gtar,,9"=""
"application/x-shar,shar,,5"=""
"application/x-sv4cpio,sv4cpio,,5"=""
"application/x-sv4crc,sv4crc,,5"=""
"application/x-tar,tar,,5"=""
"application/x-ustar,ustar,,5"=""
"audio/basic,au,,<"=""
"audio/basic,snd,,<"=""
"audio/x-aiff,aif,,<"=""
"audio/x-aiff,aiff,,<"=""
"audio/x-aiff,aifc,,<"=""
"audio/x-wav,wav,,<"=""
"audio/x-pn-realaudio,ram,,<"=""
"image/ief,ief,,:"=""
"image/tiff,tiff,,:"=""

```

```

"image/tiff,tif,,:"=""
"image/x-cmu-raster,ras,,:"=""
"image/x-portable-anymap,pnm,,:"=""
"image/x-portable-bitmap,pbm,,:"=""
"image/x-portable-graymap,pgm,,:"=""
"image/x-portable-pixmap,ppm,,:"=""
"image/x-rgb,rgb,,:"=""
"image/x-xbitmap,xbm,,:"=""
"image/x-xpixmap,xpm,,:"=""
"image/x-xwindwdump,xwd,,:"=""
"text/html,stm,,h"=""
"text/plain,bas,,0"=""
"text/plain,c,,0"=""
"text/plain,h,,0"=""
"text/richtext,rtx,,0"=""
"text/tab-separated-values,tsv,,0"=""
"text/x-setext,etx,,0"=""
"application/x-perfmon,pmc,,5"=""
"application/x-perfmon,pma,,5"=""
"application/x-perfmon,pmr,,5"=""
"application/x-perfmon,pml,,5"=""
"application/x-perfmon,pmw,,5"=""

```

```

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\InetInfo\Performance]
"Library"="infoctrs.DLL"
"Open"="OpenINFOPerformanceData"
"Close"="CloseINFOPerformanceData"
"Collect"="CollectINFOPerformanceData"
"Last Counter"=dword:00000756
"Last Help"=dword:00000757
"First Counter"=dword:00000738
"First Help"=dword:00000739

```

```

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\W3SVC]
"Type"=dword:00000020
"Start"=dword:00000002
"ErrorControl"=dword:00000000
"ImagePath"=hex(2):43,3a,5c,57,49,4e,4e,54,5c,53,79,73,74,65,6d,33,32,5c,69,6e,\
65,74,73,72,76,5c,69,6e,65,74,69,6e,66,6f,2e,65,78,65,00
"DisplayName"="World Wide Web Publishing Service"
"DependOnService"=hex(7):52,50,43,53,53,00,4e,54,4c,4d,53,53,50,00,00
"DependOnGroup"=hex(7):00
"ObjectName"="LocalSystem"

```

```

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\W3SVC\HTMLA]

```

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\W3SVC\Parameters]

"MajorVersion"=dword:00000002
 "MinorVersion"=dword:00000000
 "AdminName"="Administrator"
 "AdminEmail"="Admin@corp.com"
 "MaxConnections"=dword:000186a0
 "LogType"=dword:00000000
 "LogFileDirectory"=hex(2):25,53,79,73,74,65,6d,52,6f,6f,74,25,5c,53,79,73,74,\
 65,6d,33,32,5c,4c,6f,67,46,69,6c,65,73,00
 "LogFileTruncateSize"=dword:01388000
 "LogFilePeriod"=dword:00000001
 "LogFileFormat"=dword:00000000
 "LogSqlDataSource"="HTTPLOG"
 "LogSqlTableName"="Internetlog"
 "LogSqlUserName"="InternetAdmin"
 "LogSqlPassword"="sqllog"
 "Authorization"=dword:00000005
 "AnonymousUserName"="IUSR_BEER"
 "Default Load File"="Default.htm"
 "Dir Browse Control"=dword:4000001e
 "CheckForWAISDB"=dword:00000000
 "CacheExtensions"=dword:00000001
 "GlobalExpire"=dword:ffffff
 "ServerSideIncludesEnabled"=dword:00000001
 "ServerSideIncludesExtension"=".stm"
 "DebugFlags"=dword:00000008
 "ScriptTimeout"=dword:00000384
 "ConnectionTimeOut"=dword:00000a8c
 "InstallPath"="C:\WINNT\System32\inetnsrv"
 "SecurePort"=dword:000001bb
 "Filter DLLs"="C:\WINNT\System32\inetnsrv\sspifilt.dll"
 "AccessDeniedMessage"="Error: Access is Denied."
 "NTAuthenticationProviders"="NTLM"
 "AcceptExOutstanding"=dword:00000800
 "ServerComment"=""

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\W3SVC\Parameters\Deny IP List]

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\W3SVC\Parameters\Grant IP List]

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\W3SVC\Parameters\Script Map]

".idc"="C:\WINNT\System32\inetnsrv\httpodbc.dll"

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\W3SVC\Parameters\Virtual Roots]

"/, "C:\inetpub\wwwroot,,5"
 "/Scripts, "C:\inetpub\scripts,,4"
 "/iisadmin, "C:\WINNT\System32\inetnsrv\iisadmin,,1"

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\W3SVC\Performance]

"Library"="w3ctrs.DLL"
 "Open"="OpenW3PerformanceData"
 "Close"="CloseW3PerformanceData"
 "Collect"="CollectW3PerformanceData"
 "Last Counter"=dword:00000790
 "Last Help"=dword:00000791
 "First Counter"=dword:00000758
 "First Help"=dword:00000759

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\W3SVC\Security]

"Security"=hex:01,00,14,80,c0,00,00,00,cc,00,00,00,14,00,00,00,34,00,00,02,\
 00,20,00,01,00,00,02,80,18,00,ff,01,0f,00,01,01,00,00,00,00,01,00,00,\
 00,00,20,02,00,02,00,8c,00,05,00,00,00,00,18,00,8d,01,02,00,01,01,00,\
 00,00,00,00,01,00,00,00,74,00,73,00,00,00,1c,00,fd,01,02,00,01,02,00,00,\
 00,00,00,05,20,00,00,23,02,00,00,76,00,63,00,00,00,1c,00,ff,01,0f,00,01,\
 02,00,00,00,00,05,20,00,00,20,02,00,00,76,00,63,00,00,00,1c,00,ff,01,\
 0f,00,01,02,00,00,00,00,05,20,00,00,25,02,00,00,76,00,63,00,00,18,\
 00,fd,01,02,00,01,01,00,00,00,00,05,12,00,00,00,25,02,00,00,01,01,00,00,\
 00,00,00,05,12,00,00,00,01,01,00,00,00,00,05,12,00,00,00

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\W3SVC\W3SAMP]

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\W3SVC\Enum]

"0"="Root\LEGACY_W3SVC\0000"
 "Count"=dword:00000001
 "NextInstance"=dword:00000001

D.3 Tuxedo Configuration

*RESOURCES

IPCKEY 133133

MAXACCESSERS 500

MAXSERVERS 50

MAXSERVICES 50

MODEL SHM

MASTER tpcc

LDBAL Y
SCANUNIT 15
BLOCKTIME 60
BBLQUERY 60

*MACHINES

DEFAULT:

"COFFEE" LMID= tpcc
TUXDIR="C:\tuxedo"
APPDIR="C:\inetPub\wwwroot"
TUXCONFIG="C:\inetPub\wwwroot\tuxconfig"
ULOGPFX="C:\inetPub\wwwroot\ULOG"
TYPE="WinNT"
UID= 0
GID= 0

*GROUPS

GROUPNO

LMID=tpcc GRPNO=1 OPENINFO=NONE

GROUPPAY

LMID=tpcc GRPNO=2 OPENINFO=NONE

GROUPPOS

LMID=tpcc GRPNO=3 OPENINFO=NONE

GROUPSL

LMID=tpcc GRPNO=4 OPENINFO=NONE

GROUPDEL

LMID=tpcc GRPNO=5 OPENINFO=NONE

*SERVERS

DEFAULT:

neworder SRVGRP=GROUPNO

SRVID=100

MIN=16 MAX=20

CLOPT="-A -- -Sgarbage"

RQADDR=newq REPLYQ=Y

payment SRVGRP=GROUPPAY

SRVID=200

MIN=11 MAX=15

CLOPT="-A -- -Sgarbage"

RQADDR=payq REPLYQ=Y

orderstatus SRVGRP=GROUPPOS

SRVID=300

MIN=4 MAX=5

CLOPT="-A -- -Sgarbage"

RQADDR=ordq REPLYQ=Y

stocklevel SRVGRP=GROUPSL

SRVID=400

MIN=6 MAX=10

CLOPT="-A -- -Sgarbage"

RQADDR=stkq REPLYQ=Y

delivery SRVGRP=GROUPDEL

SRVID=500

MIN=3 MAX=5

CLOPT="-A -- -Sgarbage -F"

RQADDR=delq REPLYQ=N

*SERVICES

D.4 Microsoft SQL Server 6.5 Parameters

name	minimum	maximum	config_value	run_value
affinity mask	0	2147483647	63	63
allow updates	0	1	1	1
backup buffer size	1	32	1	1
backup threads	0	32	5	5
cursor threshold	-1	2147483647	-1	-1
database size	2	10000	2	2
default language	0	9999	0	0
default sortorder id	0	255	50	50
fill factor	0	100	0	0
free buffers	20	524288	4000	4000
hash buckets	4999	16777216	749011	749011
language in cache	3	100	3	3
LE threshold maximum	2	500000	301	301
LE threshold minimum	2	500000	20	20
LE threshold percent	1	100	0	0
locks	5000	2147483647	5000	5000
LogLRU buffers	0	2147483647	2100	2100
logwrite sleep (ms)	-1	500	-1	-1
max async IO	1	1024	25	25
max lazywrite IO	1	1024	100	100
max text repl size	0	2147483647	65536	65536
max worker threads	10	1024	180	180
media retention	0	365	0	0
memory	2800	1048576	900000	900000
nested triggers	0	1	1	1
network packet size	512	32767	4096	4096
open databases	5	32767	20	20
open objects	100	2147483647	500	500
priority boost	0	1	0	0
procedure cache	1	99	2	2
Protection cache size	1	8192	15	15
RA cache hit limit	1	255	4	4
RA cache miss limit	1	255	3	3
RA delay	0	500	15	15
RA pre-fetches	1	1000	3	3
RA slots per thread	1	255	5	5
RA worker threads	0	255	0	0
recovery flags	0	1	0	0
recovery interval	1	32767	32767	32767
remote access	0	1	0	0
remote conn timeout	-1	32767	10	10
remote login timeout	0	2147483647	5	5
remote proc trans	0	1	0	0
remote query timeout	0	2147483647	0	0

remote sites	0	256	0	0
resource timeout	5	2147483647	10	10
set working set size	0	1	0	0
show advanced options	0	1	1	1
SMP concurrency	-1	64	-1	-1
sort pages	64	511	64	64
spin counter	1	2147483647	10000	10000
tempdb in ram (MB)	0	2044	5	5
time slice	50	1000	100	100
user connections	5	32767	370	370
user options	0	4095	0	0

D.5 Microsoft SQL Server 6.5 Startup Parameters

Configuration Parameters

No Windows NT Server Registry parameters were changed from their default settings.

The following services were stopped on the server and clinet machines:

Computer Browser, Licence Logging Service, Messenger, Net Logon, Plug and Play, RPC locator, Server, Spooler, TCP/IP NetBIOS helper.

Microsoft SQL Server Startup Parameters

```
C:\mssql\bin\sqlservr -c -x -t1081 -T812 -T3502 -T1140 -Cd1440000 -Cp4500
```

The above parameters where:

- c Start SQL Server independently of the Service Control Manager
- x This disables the keeping of the CPU time and cache hit ratio statistics
- t1081 Allows the index pages to travel through the cache a second time
- T812 This disablesthe checkpoint buffer sorting
- T3502 Writes a message to the SQL Server Errorlog showing the beginning and ending tim each checkpoint
- T1140 Optimizes the free space allocation
- Cd1440000 Defines 2KB of database cache buffers
- Cp4500 Defines a number of buffers to be used by the procedure cache

Microsoft Server Stack Size

Microsoft SQL Server 6.5 Enterprize Edition default stack size was changed when using the EDITBIN utility.

The EDITBIN utility is available with Microsoft Visual C++ V4.0 and the command used to change the stack size is editbin /S: 65536 sqlservr.exe

The above command editbin /S: 65536 sqlservr.exe can be found as an article in the Microsoft Knowledge Base on the Microsoft Web Site at www.microsoft.com/support

The DBCC GAMINIT

After execution of the benchmark, the following script was run to proactively to populate the Global Allocation Map (GAM) rather than allowing it to be populated on a as needed basis.

```
Use tpcc
go
dbcc gaminit
go
```

The above commands are an article in the Microsoft Knowledge Base on the Microsoft Web Site www.microsoft.com/support

Cache'Column of Sysobjects Table

Before the execution of the benchmark, the following script was used and SQL Server was restarted to improve the cache performance of tables which are accessed non-uniformly. Us this feature is documented as an article in the Microsoft Knowledge Base on the Microsoft W Site at www.microsoft.com/support

```
use tpcc
go
update sysobjects set cache=2 from sysobjects where name = 'stock'
go
update sysobjects set cache=5 from sysobjects where name = 'customer'
go
```

BOOT.INI

In the Boot.ini file /3gb"switch was added to allow the use of 3GB of user and 1GB of kernel virtual address space to be used rather than the usual 2GB.

Appendix F Third Party Letters



November 20, 1997

Mr. D. A. Howarth
FUJITSU ICL Computers Ltd.,
Lowelace Road, Bracknell, Berkshire,
England RG12 8SN

Dear Mr. Howarth:

Per your request I am enclosing the pricing information regarding TUXEDO 6.x that you requested. This pricing applies to Tuxedo 6.1, 6.2 and 6.3. Please note that Tuxedo 6.3 is our most recent version of Tuxedo but that all 6.x releases are generally available. Core functionality services pricing is appropriate for your activities. As per the table below the FUJITSU system is classified as either a Tier 1 or Tier 2 server depending on the CPU Capacity of the system. If the CPU capacity is 2 CPUs then the system is a tier 1 system. If the CPU capacity is 4 CPUs then the systems is a tier 2 system.

Tuxedo Core Functionality Services (CFS) Program Product Pricing and

Description

TUX CFS is a limited function product from BEA that offers a subset of TUXEDO 6.3/6.2/6.1 capabilities. The following standard 6.x features ARE NOT AVAILABLE WITH TUX-Core AND CANNOT BE SUPPORTED BY TUX-Core:

1. /WS
2. /Q
3. /COBOL
4. /DCE
5. Transaction
6. Events
7. DOMAINS
8. Admin API
9. ACLS (Access Control List security option)
10. Link Level Encryption
11. Web Browser GUI Admin

TUX-CFS provides a basic level of middleware support for distributed computing, and is best used by organizations with substantial resources and knowledge for advanced distributed computing implementations.

TUX-CFS prices are server only and are based on the overall performance characteristics of the server and uses the same five tier computer classification as TUXEDO 6.3. Prices range from \$3,000 for Tier 1 to \$250,000 for Tier 5. Under this pricing option EVERY system running TUX-CFS at the user site must have a TUXEDO license installed and pay the appropriate per server license fees.

11/21/97 FRI 10:10 PM 300 000 000

10/31/97

BEA SYSTEMS, INC.

BEA Tux/CFS Unlimited User License Fees Per Server

Unlimited User License fees per server	Number of Users	Dollar Amount	Maintenance (5 x 8) per year	Maintenance (7 x 24) per year
Tier 1 -- PC Servers with 1 or 2 CPUs, entry level RISC Uniprocessor workstations and servers (Class 1 and Class 2)	Unlimited	\$3,000.00	\$450.00	\$660.00
Tier 2 - PC Servers with 3 or 4 CPUs, Midrange RISC Uniprocessor servers and workstations (class 3)	Unlimited	\$12,000.00	\$1,800.00	\$2,640.00
Tier 3 - Midrange Multiprocessors, up to 8 CPUs per system capacity (Class 4 and 5)	Unlimited	\$30,000.00	\$4,500.00	\$6,600.00
Tier 4 - Large (more than 8, less than 32 CPUs) and Mainframe Systems (Class 6)	Unlimited	\$100,000.00	\$15,000.00	\$22,000.00
Tier 5 - Massively Parallel Systems, > 32 processors	Unlimited	\$250,000.00	\$37,500.00	\$55,000.00

Intel based server tier classifications:

Platform	Operating System	Tier 1 Class 1	Tier 1 Class 2	Tier 2 Class 3	Tier 3 Class 4	Tier 3 Class 5
Intel Pentium/ Pentium Pro PCs	Interactive R3.2 ESIX SVR 4.0 SCO UNIX 3.2.2 and 3.2.4 SCO ODT 2.x,3.x Solaris x86 2X UnixWare, Windows NT 3.5/4.0	All 386/486 PCs are Class 1	ALL Pentium and Pentium Pro PCs with 1 or 2 CPUs capacity are Tier 1	ALL Pentium and Pentium Pro PCs with 3 or 4 CPUs capacity are Tier 2	ALL Pentium and Pentium Pro PCs with 5,6,7, or 8 CPUs are Tier 3	ALL Pentium and Pentium Pro PCs with 5,6,7, or 8 CPUs are Tier 3

Very Truly Yours,



Lewis D. Brentano,
Director Market Planning



Fax

To: Ronald Chou **From:** Joseen Ruisena, Manager - Cyberstore Division

Fax: +44 1344 472060 **Pages:** 1 (including cover sheet)

Phone: +44 1344 472697 **Date:** February 5, 1998

Re: Price Quote **CC:** N/A

Urgent For Review Please Comment Please Reply Please Recycle

Comments:

Compx MicroHub WX1205 100TX, 5 Port Hub
 Each Unit 5186.77 X4 units= \$747.08
 Compx MicroHub/5 10 port hub & RAM5 18NC Stackable
 Each Unit 534.42 X1515 units= \$82,831.10

TOTAL \$83,378.18 - Cash Price (Check, Cashier's Check, Wire Transfer)

Shipping to Customs - NO CHARGE

TOTAL \$85,513.31 - Credit Card Price

I thank you for the opportunity to give you a quote. If you find a better price, we would like to try to beat it. It has been a pleasure serving you.

Voice 913.385.3888 x112 Fax 913.385.3360 www.netss.es.net jrsen@netss.es.net

02 04 93 NEW YORK FAX 8001366 TELEPHONE NUMBER IN U.S.
 Microsoft Corporation TEL 212 853 8100
 One Microsoft Way TSN 16150
 Redmond, WA 98072-5099 Fax 425363729



February 6, 1998

Mr. David Howarth
 Performance Benchmarking Manager
 Services Division
 Fujitsu/SQL Computers Ltd.
 Lovelace Road, Bicknell, Berkshire
 England RG9 1J 8SN

via FAX +44 7344 477000

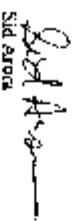
Dear Dave,

Here is the information you requested regarding US pricing of certain Microsoft products:

Microsoft SQL Server, Enterprise Edition 4.3, unlimited user license	\$28599
Microsoft Windows NT Server, Enterprise Edition 4.0, incl 2x CALs	\$3999
Windows NT Server 4.0, incl 5 CALs	\$809
Microsoft SQL Workstation (includes programmer's toolkit)	\$499
Visual C++ 32-bit edition (includes libraries)	\$499
3-yr maintenance for above software @ \$209/yr	\$10475

This quote is valid for the next 90 days. Please let me know if I can be of any further assistance.

Very regards,



Sid Avra
 Product Manager, Microsoft SQL Server
 Personal and Business Systems Group

P. O. Box 513
 Great Ashenden,
 Bucks, England, HP18 0P5
 Tel: +44 1494 896381
 Fax: +44 1494 890378
 E Mail: Ron@mylex.com



Fax

To: David Howarth From: Ron MacLeod, V. P. For Europe
 Fax: _____ Pages: 2
 Phone: _____ Date: 5 February 1998
 Re: DACPJ Quotation CC: _____
 Urgent For Review Please Comment Please Reply Please Recycle

• Comments:

Dear Mr Howarth

In response to your request Mylex is delighted to submit the following quotation to Fujitsu TCl for the supply of the DACPJ RAID controller.

<u>Item</u>	<u>Product Description</u>	<u>MSRP</u>
1	Mylex DACPJ - 3 - 3B - MY1	\$1,995
3	Channel controller with 8Mb EDO	

Notes

- 1 All prices above are in \$US
 - 2 Normal warranty is 3 years. Estimated maintenance cost for additional 2 years is \$50
 - 3 Above products are available for immediate delivery. For any large quantity please allow a lead-time of 30 days after receipt of order.
 - 4 Above prices are listed on ex works factory Fremont California
- Should there be anything further I can be of assistance or please advise me

Respectfully yours,

Ronald F. MacLeod
 Vice President Europe, Africa & Middle East.

Appendix G 180 Day Space Calculations.

Warehouses

1110

tpmC 13391.13

12.06 tpmC/W

Table	Rows	Data (KB)	Index (KB)	Free (KB)	Daily growth	5% space	
Warehouse	1,110	2,220	12	8		112	
District	11,100	22,200	94	10		1,115	
Item	100,000	9,100	46	6		457	
History	33,300,000	1,665,002	0	6	321,384	0	
Order	33,300,000	865,800	5,222	18	167,120	0	
Customer	33,300,000	22,204,440	1,723,374	26		1,196,391	
New_order	9,990,000	109,890	668	18		5,528	
Order_line	332,999,230	18,514,758	121,098	16	3,573,782	0	
Stock	111,000,000	37,007,400	204,468	20		1,850,371	
Totals		80,400,810	2,054,982	0	128	4,062,286	3,053,973

Catalog et al

26,624

create database tpcc on

tpc_misc1	655					
tpc_misc2	655					
tpc_misc3	655					
tpc_misc4	655					
tpc_misc5	880	3,500				
tpc_ol1	4549					
tpc_ol2	4549					
tpc_ol3	4549					
tpc_ol4	4549					
tpc_ol5	7000	25,196				
tpc_cs1	9952					
tpc_cs2	9952					
tpc_cs3	9952					
tpc_cs4	9952					
tpc_cs5	9952					
tpc_cs6	13240	63,000				
Total DB (KB)	93,896,704					

Dynamic space Sum of Data for Order, Order_Line and History

Static space Sum of all data,index + 5% - above dynamic space

Free space Total space allocated - dynamic and static

Daily growth	4,062,286	(Dynamic space/W * 62.5)* tpmC
Daily spread	87,803,275	Free space - 1.5 * Daily growth (zero if negative)
180 day space	795,675,692	SQLServer NT can be reconfigured to eliminate daily spread, zero assumed
180 day in GB	759	Static space + 180 (daily growth + daily spread)

Log space	8,628544922
used gb	
N-O performed	1575853
8 hr lqa space	35,191

(GB)

log + mr 9gb drives
 Excess log space

180 day space
 OS+SQL Serve
 rNT
 Pageset
 Less excess log space
 Total

List size and quantity of priced disks here:

Disk capacity	Number	Capacity(MB)
4.0518	84	340.35
1.9	1	1.90
8.4785	39	330.66
Total	124.00	672.91

There is not enough disk capacity priced.

Exchange. **18**
 Or add **9GB** **9**
 Plus log/mirror **10**
 4gb **66**
 9gb **67**

Appendix H Attestation Letter



PERFORMANCE METRICS INC.
TPC Certified Auditors

February 4, 1998

Mr. David Howarth
 Performance Benchmarking Manager
 FUJITSU, ICL Computers LTD,
 Lovelace Road
 Bracknell
 Berkshire
 England
 RG12 8SS

I have verified the TPC Benchmark™ C client/server for the following configuration:

Platform: Fujitsu GRANPOWER 5000 Model 670 C/S
 Database Manager: Microsoft SQL Server 6.5
 Enterprise Edition (6.50.258)
 Operating System: Microsoft Windows NT Server 4.0
 Enterprise Edition (SP3)
 Transaction Manager: BEA Tuxedo C/S Version 6.3 Client Threads for NT

Server: Fujitsu GRANPOWER 5000 Model 670 C/S				
CPU's	Memory	Disks	90% Response	tpmc
6 PentiumPro @ 200 Mhz	Main: 3584 MB Cache: 1MB each	49 @ 9GB 84 @ 4 GB 1 @ 5 GB	1.11 sec	13,391.13
9 Clients: ErgoPro e664/233				
1 Pentium II @ 233Mhz	Main: 128 MB Cache: 512K	1 @ 2 GB	na	na

PERFORMANCE METRICS INC.
TPC Certified Auditors

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 transactions were correctly implemented
- The database files were properly sized and populated.
- The database was properly scaled with 1110 warehouses. Prior to the measurements, 12 warehouse rows were deleted.
- The ACID properties were met
- Portions of the ACID tests were performed on a database scaled to 10 warehouses.
- Input data was generated according to the specified percentages
- Eight hours of mirrored log space was present on the tested system.
- Eight hours of growth space for the dynamic tables was present on the tested system.
- The data for the 180-day space calculation was verified. Eighteen 9GB drives were substituted in the piled configuration for 18 4GB drives used in the measurement.
- The steady state portion of the test was 30 minutes.
- One checkpoint was taken before the measured interval.
- One checkpoint was taken during the measured interval
- The checkpoints were verified to be clear of the guard zone.
- The system pricing was checked for major components and maintenance.

Auditor Notes:

The 10 drives used for the database log were 10,000RPM drives.

Sincerely,



Tom Sawyer
Auditor