

**TPC Benchmark™ C
Full Disclosure
Report**

Advanced Logic Research, Inc.

Revolution Quad6 SD512

using

Microsoft NT Server 4.0

and

Microsoft SQL Server 6.5

**Submitted for review
April 1997**

First Edition - April 1997

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

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

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

All performance data contained in this report was obtained in a rigorously controlled environment, and therefore results obtained in other operating environments may vary significantly. ALR and Microsoft Corporation do not warrant or represent that a user can or will achieve similar performance expressed in transactions per minute (tpmC) or normalized price/performance (\$/tpmC). No warranty of system performance or price/performance is expressed or implied in this report.

Copyright © 1997 Advanced Logic Research, Inc.

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

Printed in USA, April 1997.

ALR Part Number: 6597/0040

ALR is a registered trademark of Advanced Logic Research, Inc.

Intel, Pentium 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 IpmC are trademarks of the Transaction Processing Performance Council.

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

Page Status

Page	Issue
i through xiii	Blank
xiv	Blank
0-1 through 0-3	Blank
0-4	Blank
1-1 through 1-1	Blank
1-2	Blank
2-1 through 2-2	Blank
3-1 through 3-3	Blank
3-4	Blank
4-1 through 4-3	Blank
4-4	Blank
5-1 through 5-9	Blank
5-10	Blank
6-1 through 6-2	Blank
7-1 through 7-2	Blank
8-1 through 8-1	Blank
8-2	Blank
9-1 through 9-3	Blank
9-4	Blank
A-1 through A-87	Blank
A-88	Blank
B-1 through B-46	Blank
C-1 through C-14	Blank
D-1 through D-7	Blank
D-8	Blank
E-1 through E-1	Blank
E-2	Blank
F-1 through F-6	Blank

Overview

This report documents the methodology and results of the TPC Benchmark C (TPC-C) conducted on the ALR Revolution Quad6 SD512. The operating systems on both the clients and the server was Microsoft Windows NT Server 4.0 SP1. The DBMS used was Microsoft SQL Server 6.5 SP3. The clients ran Microsoft's Internet Information Server 3.0.

TPC Benchmark Metrics

The standard TPC Benchmark C metrics, tpmC (transaction per minute), price per tpmC (five year capital cost per measured tpmC), and the availability date are reported as required by the benchmark specification.

Executive Summary

The following pages contain the executive summary results of the benchmark.

Auditor

The benchmark configuration, environment, and methodology used to produce and validate the test results, along with the pricing model used to calculate the cost per tpmC, were audited by Richard Gimarc of Performance Metrics, Inc. to verify compliance with the relevant TPC specification.

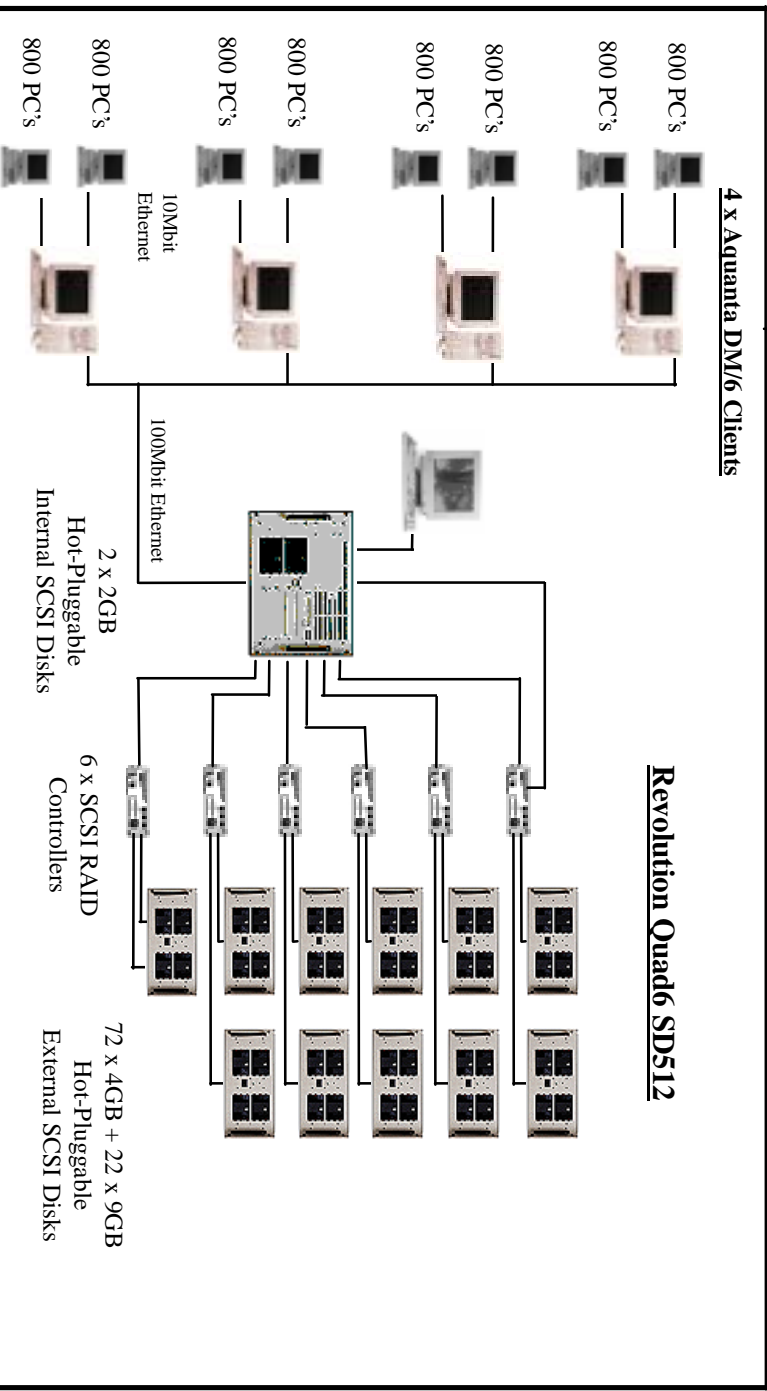
**Advanced Logic
Research, Inc.**

**Revolution Quad6 SD512
Client/Server**

TPC-C Rev. 3.2.3

Report Date:
04-Apr-1997

Total System Cost	TPC-C Throughput	Price/Performance	Availability Date
\$427,949	7407.05 tpmC	\$57.78 per tpmC	30-Apr-1997
Processors	Database Manager	Operating System	Other Software
4 Pentium® Pro 200 MHz 512MB L2 cache	Microsoft SQL Server 6.5 SP3	Microsoft NT Server 4.0 SP1	Microsoft IIS 3.0
			Number of Users
			6400



System Components	Server		Clients	
	Quantity	Type	Quantity	Type
Processors	4	200 MHz Pentium® Pro with 512KB Level 2 Cache	4	200MHz Pentium® Pro with 256KB Level 2 Cache
Memory	1	2048MB	4	128 MB
Disk Controllers	6	SCSI RAID controllers	4	Inbuilt IDE
Disk Drives	2	2GB internal	4	1.5 GB internal
	72	4GB external		
	22	9GB external		
Total Storage		482.25 GB		6 GB
CD-ROM / Tape	1	CD-ROM Drive	4	CD-ROM Drive

Description	Part Number	Third Party Brand Pricing	MRLP Unit Price	Qty.	Extended Price	5 Years Maint.
Server Hardware						
Revolution Quad6 SD512	72460220		\$12,495	1	\$12,495	\$14,314
200-MHz Pentium Pro/512KB Cache	12968621		\$2,295	3	\$6,885	INCL
ECC Memory Board, 0MB Mem	(included w/ 72460220)		\$0	1	\$0	INCL
128-MB Memory Upgrade	12903260		\$2,755	16	\$44,080	INCL
SVGA Video Card	(included w/ 72460220)		\$0	1	\$0	INCL
3-Channel ADAC PCI Ultra SCSI RAID Ctrlr	11900983-10		\$1,315	6	\$7,890	INCL
2-GB SCA F/W Ultra SCSI Hard Drive	11351230		\$955	2	\$1,910	INCL
16X SCSI CD-ROM Drive	11315280		\$168	1	\$168	INCL
10/100 PCI Network Interface Card	CP000169		\$290	1	\$290	INCL
Quick Hot Swap II Storage Drvr + 2 spares	11910190		\$1,795	13	\$23,335	INCL
4-GB SCA F/W Ultra SCSI Hard Drive	11351430		\$1,795	72	\$129,240	INCL
9-GB SCA F/W Ultra SCSI Hard Drive	11351910		\$2,845	22	\$62,590	INCL
14-inch Color Monitor	11001034		\$295	1	\$295	INCL
Keyboard Tray Kit (Kybod, Mouse, Cables)	11910130		\$165	1	\$165	INCL
Server to Disk Drawers SCSI Cable	(two included w/ ea 11910190)		\$0	12	\$0	INCL
Rack Cabinet (44U)	11910170		\$2,195	3	\$6,585	INCL
Rack Coupling Kit (44U)	82321506		\$165	2	\$330	INCL
Rack Cabinet Bezel Kit (44U)	(included w/ 1191070)		\$0	3	\$0	INCL
Rack Cabinet Weight Kit (0U)	(N/A)		\$0	3	\$0	INCL
Monitor Alcove	82321502		\$145	1	\$145	INCL
Keyboard Drawer (1U)	(included w/ 11910130)		\$0	1	\$0	INCL
Rack Cabinet Side Panel Kit (44U)	82321503		\$245	1	\$245	INCL
Rack Cabinet Filler Panel (3U)	82321204		\$30	6	\$180	INCL
Rack Cabinet Filler Panel (1U)	82321202		\$15	2	\$30	INCL
Subtotal					\$296,858	\$14,314
Server Software						
Microsoft Windows NT Server 4.0 (w/ 10cal)	Microsoft	2	\$809	1	\$809	\$0
Microsoft SQL Server 6.5 w/ 5 cal	Microsoft	2	\$1,399	1	\$1,399	\$10,475
MSSQL Internet Conn. License	Microsoft	2	\$2,999	1	\$2,999	\$0
Subtotal					\$5,207	\$10,475
Client Hardware						
SYS: Aquantia DM/6 II, 0 Proc, 0MB Mem	CMT60072-ZFA	Unisys	\$343	4	\$1,372	\$3,272
PRCC:1x200MHz PentiumPro/256KB Cache	PRC 6200-256	Unisys	\$1,087	4	\$4,348	INCL
MEM: 32 MB EDO Memory Upgrade	MPM4-32M	Unisys	\$244	16	\$3,904	INCL
CTRL: SVGA	PCV102-PCI	Unisys	\$176	4	\$704	INCL
DISK: 1.5GB Drive	HDI1600-5	Unisys	\$306	4	\$1,224	INCL
CDROM: Twelve Speed IDE	CDR1200-SI	Unisys	\$357	4	\$1,428	INCL
ETHERNET: 100Mbit/sec, PCI 32-bit	ETH101007-PCI	Unisys	\$156	4	\$624	INCL
ETHERNET: 10Mbit/sec, PCI 32-bit	ETH101-PCI	Unisys	\$153	8	\$1,224	INCL
MONITOR: 14-inch Color	EVG142-COL	Unisys	\$341	4	\$1,364	INCL
K/B: Keyboard/Mouse/Power Cord kit	KBM104-SS	Unisys	\$78	4	\$312	INCL
Subtotal					\$16,504	\$3,272
Client Software						
Microsoft Windows NT Server 4.0	Microsoft	2	\$809	4	\$3,236	\$0
Microsoft SQL Server Programmer's Toolkit	Microsoft	2	\$499	1	\$499	\$0
Microsoft Visual C++ v.4.0	Microsoft	2	\$499	1	\$499	\$0
Subtotal					\$4,234	\$0
User Connectivity						
NetLUX 8-Port 100TX TrueFast Ethernet Hub + 2 spares	NETLUX	3	\$595	3	\$1,785	\$0
NetLUX 24-Port 10Base-T Ethernet Hub + 10% spares	NETLUX	3	\$251	300	\$75,300	\$0
Subtotal					\$77,085	\$0
Total					\$399,888	\$28,061

- Notes:**
- All Microsoft maintenance is covered by the maintenance cost of Microsoft SQL Server.
 - 10% or minimum 2 spares are added in place of onsite service (products have a five year return-to-vendor warranty)
 - Pricing: 1 - Unisys, 2 - Microsoft, 3 - NetLUX

		Five Year Cost of Ownership	TPC-C Throughput	\$/tpmc
		\$427,949		
		7407,05		
		\$57,78		

The benchmark results and test methodology were audited by Richard Gimarc of Performance Metrics, Inc.

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

NUMERICAL QUANTITIES SUMMARY

ALR Revolution Quad6 SD512
Microsoft SQL Server 6.5

MQTth, Computed Maximum Qualified Throughput: **7407.05**
% throughput difference, reported & reproducibility runs: 1.41%

Transaction Mix (% of Total Transactions)

Transaction Mix	
New-Order	44.76%
Payment	43.06%
Delivery	4.05%
Stock-Level	4.06%
Order-Status	4.07%

Response Time (seconds)

Transaction	Average	Maximum	90th %ile
New-Order	2.00	8.37	2.73
Payment	1.83	6.29	2.54
Delivery	0.29	0.97	0.31
Stock-Level	4.48	12.54	6.20
Order-Status	2.48	11.57	3.36
Menu	0.21	1.08	0.32
Delivery (Deferred)	2.97	28.37	3.65

Response time delay added for emulated components (seconds)

RT Response time 0.1
Menu Response time 0.1

Keying/Think Time Times (seconds)

Transaction	Minimum	Average	Maximum
New-Order	18.00/0.00	18.02/12.07	18.48/120.51
Payment	3.00/0.00	3.02/12.03	3.54/120.53
Delivery	2.00/0.00	2.01/5.04	2.34/37.20
Stock-Level	2.00/0.00	2.01/5.06	2.20/42.73
Order-Status	2.00/0.00	2.01/10.02	2.29/95.31

Test Duration:

Ramp up time 1hour, 28 minutes
Measurement interval (M) 60 minutes
Transactions (all types) completed during measurement interval 992,864
Ramp-down time 35.5 minutes
Checkpointing:
Number of checkpoints 2
Checkpoint interval 30 minutes

Table of Contents

Abstract.....	iv
Table of Contents.....	viii
Preface.....	xii
0. General Items	0-1
0.1. Order and Titles	0-1
0.2. Executive Summary Statement	0-1
0.3. Numerical Quantities Summary.....	0-1
0.4. Application Code Disclosure	0-1
0.5. Benchmark Sponsor	0-2
0.6. Parameter Settings	0-2
0.7. Configuration Diagrams.....	0-2
1. Clause 1: Logical Database Design.....	1-1
1.1. Table Definitions	1-1
1.2. Physical Organization of the Database	1-1
1.3. Insert and/or Delete Operations	1-1
1.4. Partitioning	1-1
1.5. Replication, Duplication or Additions	1-1
2. Clause 2: Transaction & Terminal Profiles.....	2-1
2.1. Random Number Generation	2-1
2.2. Input/Output Screen Layout.....	2-1
2.3. Priced Terminal Feature Verification	2-1
2.4. Presentation Managers or Intelligent Terminal	2-1
2.5. Transaction Statistics	2-2
2.6. Queuing Mechanism of Delivery	2-2
3. Clause 3: Transaction & System Properties.....	3-1
3.1. Transaction System Properties (ACID)	3-1
3.2. Atomicity	3-1

3.2.1. Completed Transaction	3-1
3.2.2. Aborted Transactions	3-1
3.3. Consistency	3-2
3.4. Isolation	3-2
3.5. Durability	3-2
3.5.1. Loss of Log and Loss of Data Disk	3-2
3.5.2. Instantaneous Interruption and Loss of Memory	3-3
4. Clause 4: Scaling & Database Population	4-1
4.1. Initial Cardinality of Tables	4-1
4.2. Constant Values	4-1
4.3. Database Layout	4-2
4.4. DBMS: Data Model and DBMS Interface/Access Language	4-2
4.5. DBMS Partitions/Replications	4-2
4.6. DBMS Space Requirements	4-2
5. Clause 5: Performance Metrics & Response Time	5-1
5.1. Measured Throughput (tpmC)	5-1
5.2. Response Times	5-1
5.3. Keying and Think Times	5-1
5.4. Response Time Frequency Distribution Curves	5-2
5.5. New Order Think Time Frequency Distribution Curve	5-5
5.6. Response Time versus Throughput Performance Curve	5-6
5.7. New-Order Throughput vs. Time	5-7
5.8. Determination of "Steady State"	5-7
5.9. Work Performed During Steady State	5-7
5.10. Reproducibility	5-8
5.11. Measurement Interval Duration	5-8
5.12. Regulation of Transaction Mix	5-8
5.13. Transaction Statistics	5-9
5.14. Checkpoint Statistics	5-9
6. Clause 6: SUT, Driver & Communications Definition	6-1
6.1. Remote Terminal Emulator (RTE) Description	6-1
6.2. Emulated Components	6-1
6.3. Functional Diagrams	6-1
6.4. Network Configuration	6-1
6.5. Network Bandwidth	6-1
6.6. Operator Intervention	6-2
7. Clause 7: Pricing	7-1
7.1. Pricing	7-1

7.1.1. System Pricing	7-1
7.1.2. Maintenance Pricing	7-1
7.1.3. Discounts	7-1
7.2. Availability	7-2
7.3. Measured tpmC, Price/Performance, and Availability Date	7-2
7.4. Country-Specific Pricing	7-2
7.5. Usage Pricing.....	7-2
8. Clause 8 : Full Disclosure Availability	8-1
8.1. Availability	8-1
9. Clause 9 : Audit	9-1
9.1. Auditor's Report	9-1
Appendix A - Client/Server Source	A-1
Appendix B - Database Design	B-1
Appendix C - Tunable Parameters.....	C-1
Appendix D - RTE Code.....	D-1
Appendix E - Disk Storage	E-1
Appendix F - Third-Party Price Quotations.....	F-1

Figures

Figure 0.1: Benchmarked Configuration.....	0-3
Figure 0.2: Priced Configuration	0-3
Figure 5.1: New Order Response Time Distribution	5-2
Figure 5.2: Payment Response Time Distribution	5-3
Figure 5.3: Order Status Response Time Distribution.....	5-3
Figure 5.4: Delivery Response Time Distribution	5-4
Figure 5.5: Stock Level Response Time Distribution.....	5-4
Figure 5.6: New Order Think Time Distribution.....	5-5
Figure 5.7: Response Time versus Throughput.....	5-6
Figure 5.8: Throughput (rpmC) versus Time.....	5-7

Tables

Table 2.1: Transaction Statistics	2-2
Table 4.1: Initial Cardinality of Database Table.....	4-1
Table 4.2: Constant C for NURand.....	4-1
Table 4.3: Benchmark Configuration Disk Storage Detail.....	4-3
Table 5.1: Response Time Data.....	5-1
Table 5.2: Keying Times.....	5-1
Table 5.3: Think Times.....	5-2
Table 5.4: Transaction Statistics	5-9

Document Structure

The TPC Benchmark C Standard Specification requires test sponsors to publish, submit to the TPC, and make available to the public, a full disclosure report for any result to be considered compliant with the specification. The required contents of the full disclosure report are specified in Clause 8.

This report is submitted to satisfy the specification's requirement for full disclosure. It documents the compliance of the benchmark implementation and execution reported for the ALR Revolution Quad6 SD512 using Microsoft SQL Server 6.5.

TPC Benchmark C Overview

The TPC Benchmark™ C Standard Specification Revision 3.2.3 was developed by the Transaction Processing Council (TPC). It is the intent of the TPC to develop a suite of benchmark to measure the performance of computer systems executing a wide range of applications.

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 characterized 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 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 the fact that this benchmark offers a rich environment that emulates many OLTP environments, 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.

0.

General Items

0.1. Order and Titles

The order and titles of sections in the Test Sponsor’s Full Disclosure report must correspond with the order and titles of sections from the TPC-C standard specification (i.e., this document). The intent is to make it as easy as possible for readers to compare and contrast material in different Full Disclosure reports.

The order and titles of the sections in this report correspond with those from the TPC-C standard specification.

0.2. Executive Summary Statement

The TPC Executive Summary Statement must be included near the beginning of the Full Disclosure report.

The TPC Executive Summary Statement is included near the beginning of this report.

0.3. Numerical Quantities Summary

The numerical quantities listed below must be summarized near the beginning of the Full Disclosure report :

- *measurement interval in minutes,*
- *number of checkpoints in the measurement interval,*
- *checkpoint interval in minutes,*
- *number of transactions (all types) completed within the measurement interval,*
- *computed Maximum Qualified Throughput in tpmC,*
- *percentage difference between reported throughput and throughput obtained in reproducibility run,*
- *ninetieth percentile, average and maximum response times for the New-Order, Payment, Order-Status, Stock-Level, Delivery (deferred and interactive) and Menu transactions,*
- *time in seconds added to response time to compensate for delays associated with emulated components,*
- *percentage of transaction mix for each transaction type.*

These numerical quantities are summarized near the beginning of this report.

0.4. Application Code Disclosure

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 the client application code used in this TPC-C benchmark. Appendix B contains the SQL stored procedures which implement the TPC-C transactions.

0.5. Benchmark Sponsor

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

This TPC benchmark C was sponsored by ALR. The benchmark test was developed by Microsoft and ALR. The benchmark was conducted at Unisys, Mission Viejo, California.

0.6. Parameter Settings

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

- *Data Base tuning options*
- *Recovery/commit options*
- *Consistency/locking options*
- *Operating system and application configuration parameters*

Appendix C contains the configuration and system parameters used in running these TPC-C tests. It also contains all the client and server OS, and SQL Server tunable parameters.

0.7. Configuration Diagrams

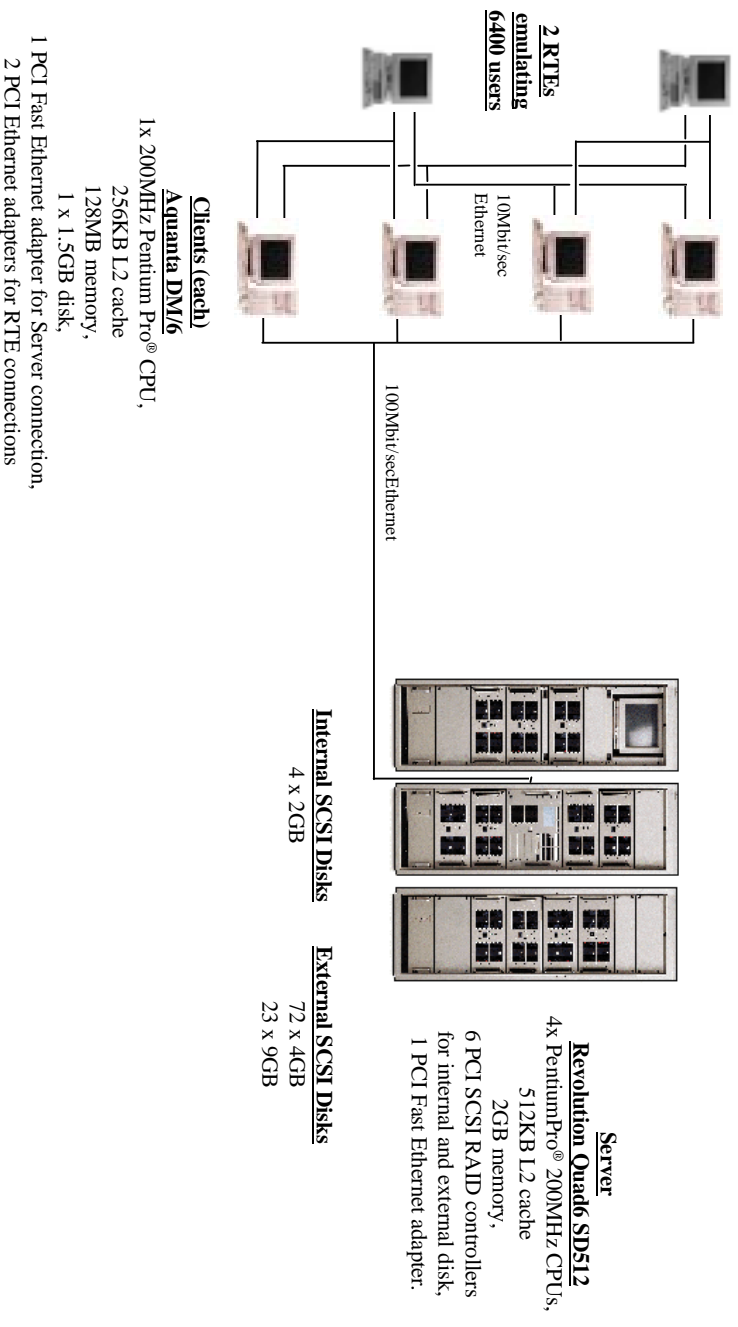
Diagrams of both measured and priced configurations must be provided, accompanied by a description of the differences. This includes, but is not limited to:

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

The Remote Terminal Emulator (RTE) software used for these TPC-C tests is a product developed by Microsoft. The benchmark configuration is illustrated in Figure 0.1. Table 4.3 contains a detailed explanation of the disk configuration.

The priced configuration for the Revolution Quad5 SD512 server is shown in Figure 0.2.

Figure 0.1: Benchmarked Configuration



Revolution Quad6 SD512 - Benchmarked Configuration

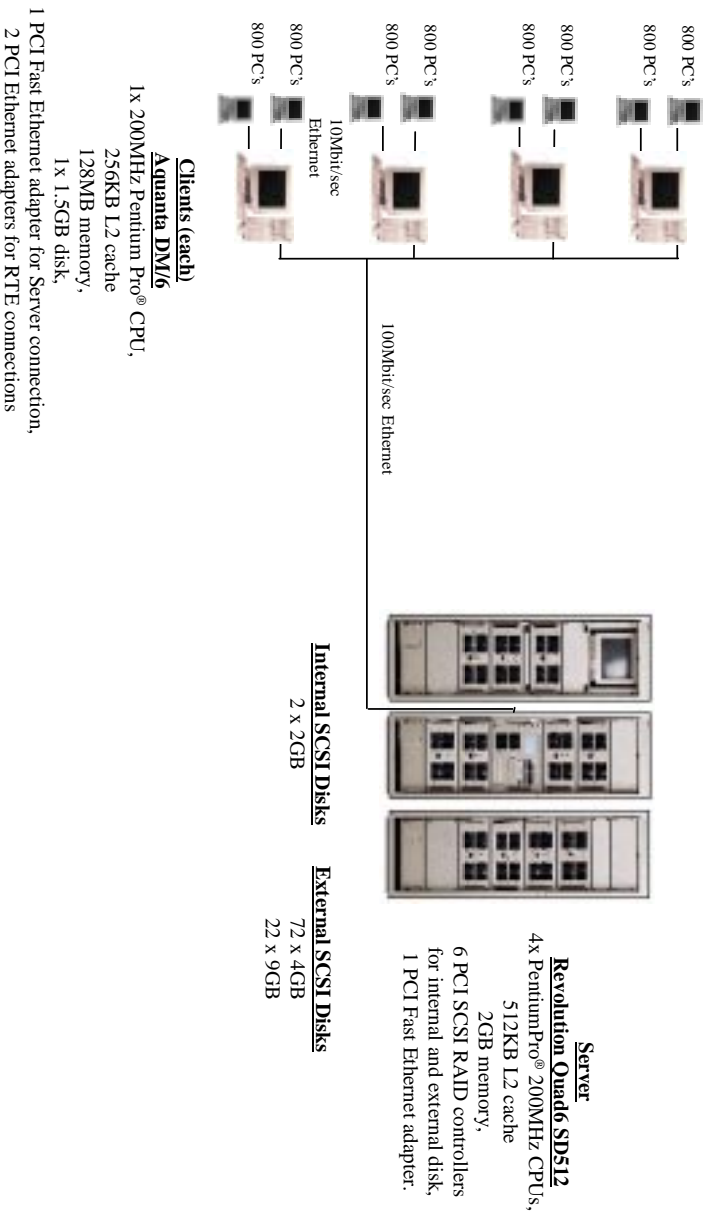


Figure 0.2: Priced Configuration

Revolution Quad6 SD512 - Priced Configuration

1. Clause 1: Logical Database Design

1.1. Table Definitions

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

Appendix B contains the SQL definitions of all the required database devices, tables, indexes and stored procedures, plus a listing of the program used to load the database and establish the required initial populations of each table.

1.2. Physical Organization of the Database

The physical organization of tables and indices, within the data base, must be disclosed.

The disk space was allocated to SQL Server according to the data in Table 4.3. In addition to the SQL definitions in Appendix B, configuration information is given in Appendix C for the six AMI MegaRAID disk controllers.

1.3. 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 data base 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.

There were no restrictions on insert and/or delete operations to any of the tables.

1.4. Partitioning

While there are few restrictions placed upon horizontal or vertical partitioning of tables and rows in the TPC-C benchmark, any such partitioning must be disclosed.

Partitioning was not used for any table in this implementation.

1.5. Replication, Duplication or Additions

Replication of tables, if used, must be disclosed.

Additional and/or duplicate attributes in any table must be disclosed along with a statement on the impact on performance.

No replications, duplications or additional attributes were used in this implementation.

2. Clause 2: Transaction & Terminal Profiles

2.1. Random Number Generation

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

The drivers used the Microsoft BenchCraft RTE program, which was independently audited. The initial population of the database was performed by the loader program from V3-02 of the Microsoft TPCC toolkit, which was also independently audited. Furthermore, the auditor sampled various initial and runtime distributions produced by this implementation to verify correctness.

2.2. Input/Output Screen Layout

The actual layout 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 in appearance due to the use of a web client implementation.

2.3. 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).

This was verified by the auditor by a direct experiment during the onsite audit portion of this benchmark, using Microsoft Internet Explorer 3.0 as the web browser.

2.4. Presentation Managers or Intelligent Terminal

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

Application code running on the client interfaced with Microsoft's Internet Information Service (web server) and implemented the TPC-C user interface. A listing of this code is included in Appendix A. No presentation manager was used on the client, as screen manipulation and data input/output was handled for each user by the Microsoft Internet Explorer web browser running on each user PC.

2.5. Transaction Statistics

The percentage of New-Order transactions 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 transactions must be disclosed.

The percentage of home and remote Payment transactions must be disclosed.

The percentage of Payment and Order-Status transactions that used non-primary key (C_LAST) access to the database must be disclosed.

The percentage of Delivery transactions 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.

Table 2.1 contains all these statistics.

Table 2.1: Transaction Statistics

Transaction Type	Statistics	Value
New Order	Rollback transactions	1.00 %
	Home warehouse	89.99%
	Remote warehouse	1.01 %
	Average Items per Order	10.00
Payment	Home warehouse	84.91%
	Remote warehouse	15.09%
	Non-primary key access	60.07%
	Non-primary key access	59.23%
Order Status	Skipped transactions (Interactive)	0
Delivery	Skipped transaction counts (Deferred)	0
	Skipped District counts (Deferred)	0
Transaction Mix	New Order	44.76%
	Payment	43.06%
	Delivery	4.05%
	Stock-Level	4.06%
	Order-Status	4.07%

2.6. Queuing Mechanism of Delivery

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

The client application process on each client machine submitted delivery transactions through a named pipe to a single delivery server process running on the same machine. This multi-threaded delivery server in turn removed queued delivery transactions from the named pipe and was responsible for submitting them to the database server and awaiting their proper completion. The source code for this delivery process is included in Appendix A.

3. Clause 3: Transaction & System Properties

3.1. Transaction System Properties (ACID)

The results of the ACID tests 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.

The TPC Benchmark C Standard Specification defines a set of transaction processing system properties that a system under test (SUT) must support during the execution of the benchmark. Those properties are Atomicity, Consistency, Isolation, and Durability (ACID).

This section defines each of these properties, describes the steps taken to ensure that they were present during the test and describes a series of tests done to demonstrate compliance with the specification. All ACID property tests were executed successfully.

3.2. Atomicity

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

3.2.1. 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 been changed appropriately.

The balances from a randomly selected warehouse, district, and customer row was retrieved by customer number from a script. A Payment transaction was submitted with the same warehouse, district and customer identifiers for a known amount. After completion of the Payment transaction, the balances of the selected warehouse, district, and customer were again retrieved to verify that the changes had been made correctly.

3.2.2. Aborted Transactions

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.

The balances from a randomly selected warehouse, district, and customer row were retrieved by customer number from a script. A Payment transaction was submitted with the same warehouse, district and customer identifiers that issued a ROLLBACK command rather than a COMMIT. After the transaction completed, the balances of the warehouse, district, and customer row was again retrieved to verify that no changes had been made to the database.

3.3. Consistency

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

The benchmark specification requires explicit demonstration of the following four consistency conditions:

1. The sum of the district balances in a warehouse is equal to the warehouse balance;
 2. For each district, the next order id minus one is equal to maximum order id in the ORDER table and equal to the maximum new order id in the NEW ORDER table;
 3. For each district, the maximum order id minus minimum order id in the ORDER table plus one equals the number of rows in the NEW-ORDER table for that district;
 4. For each district, the sum of the order line counts equals the number of rows in the ORDER-LINE table for that district;
- In order to demonstrate the consistency the following steps were taken:
1. Prior to the start of a benchmark run, the consistency of the database was verified by testing successfully conditions 1-4 described above with a script.
 2. A run under full user load was executed for over 10 minutes with a checkpoint during the run.
 3. After completion of that test, the consistency of the database was again verified by successfully testing using the same consistency script as in step 1.

3.4. 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.

The benchmark specification defines seven required tests to be performed to demonstrate that required levels of transaction isolation are met. These tests, described in Clauses 3.4.2.1 - 3.4.2.7, were all performed from a script and verified by the auditor. In Isolation Test 7, Case A was observed. In addition, the phantom tests and stock level tests were executed and verified to be successful.

3.5. Durability

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

Three durability tests were executed to satisfy the requirements of the specification. The test for loss of memory and instantaneous interruption was combined and performed with a fully scaled database with 6,400 emulated users. The loss of log and loss of data tests were also combined and performed on a separate ten warehouse database with 100 emulated users. To the best of our knowledge, these tests prove that the fully scaled configuration used for the throughput test would also meet all durability tests.

3.5.1. Loss of Log and Loss of Data Disk

The following steps were taken to demonstrate durability in the case of loss of a data disk:

1. The database was backed up to extra disks on a dump device.
2. The D_NEXT_O_ID fields for all rows in the district table were summed up to determine the initial count of orders present in the database.

3. The RTE was started with 100 users. On the driver systems, committed and rolled back New-Order transactions were recorded in a “success” file.
4. One of the mirrored hot-pluggable log disks was removed from the disk cabinet after five minutes of steady state.
5. Since the log disk mirroring is done by a hardware RAID controller, no errors were reported to the system, and the benchmark run continued without interruption.
6. After another 5 minutes of running at steady state, a hot-pluggable database disk was removed from the disk cabinet.
7. NT and SQL Server encountered IO errors due to the missing disk and recorded these errors in the NT event log and SQL Server error log, respectively.
8. First, the RTE and drivers were stopped, then the database server was shutdown and the SUT restarted.
9. SQL Server was restarted and marked the database as ‘suspect’. A dump of the transaction log was taken to extra disks on a dump device.
10. Next, scripts were executed to drop the database and all its devices. Then, SQL Server was shutdown again and the SUT shutdown.
11. The data disk was reinserted in the disk cabinet, the RAID configuration was repaired, and the SUT was restarted. (The removed log disk was never replaced.)
12. SQL Server was restarted. First, the devices and empty database were recreated. After space allocation had finished, the database was recovered by first reloading the entire initial database and log from backup, then by loading and applying the transaction log dump that was taken after the disk failure. The latter restored all committed transactions to the database.
13. Consistency condition 3 of Clause 3.3.2.3 was executed to verify database consistency.
14. Step 2 was repeated to determine the total number of orders. This number was subtracted from the count obtained previously in Step 2 to determine the number of additional orders added to the database.
15. The contents of the “success” files on the drivers were sampled to verify that the records in the “success” file for committed New-Order transactions had corresponding records in the ORDER table and no entries existed for rolled back transactions. Moreover, the counts were matched with those obtained in step 14.

3.5.2. Instantaneous Interruption and Loss of Memory

Instantaneous interruption and loss of memory tests were combined because the loss of power erased the contents of memory. This failure was induced by removing the primary power to the System Under Test while the benchmark was executing.

1. The D_NEXT_O_ID fields for all rows in the district table were summed up to determine the initial count of orders present in the database (count1).
2. On the driver systems, committed and rolled back New-Order transaction were recorded in a “success” file.
3. The benchmark was executed at full load with 6,400 emulated users for a minimum of 10 minutes.
4. Immediately after execution of a checkpoint completed, the system’s primary power was turned off.
5. The test was aborted on the driver systems, and the RTE was shutdown.
6. Power was restored to the SUT, the system rebooted, SQL Server was restarted, and automatic database recovery was performed. The database recovery uses the transaction log to reapply all committed transactions and rollback any (in progress) uncommitted transactions, so that the database disks are correct.
7. After recovery finished, Consistency Condition of Clause 3.3.2.3 (no gaps in NO_O_ID) was executed to verify that the database was consistent..
8. Next, samples of the contents of the “success” file on the driver were compared against corresponding rows of the ORDER table to verify that records in the “success” file for committed New-Order transactions had corresponding records in the ORDER table and no entries existed for rolled back transactions.
9. Finally, step 1 was repeated to determine the total number of orders (count2). Count2 minus count1 was not less than the number of committed New-Order records in the “success” file.

4. Clause 4: Scaling & Database Population

4.1. 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, must be disclosed. If the database was over-scaled and inactive rows of the WAREHOUSE table were deleted (see Clause 4.2.2 and the Auditor's attestation letter) the cardinality of the WAREHOUSE table as initially configured and the number of rows deleted must be disclosed.

The TPC-C database for this test was configured with 650 warehouses. The benchmark run was conducted with 640 warehouses; 10 warehouses were deleted. The cardinality of each table in the database is listed in Table 4.1

Table 4.1: Initial Cardinality of Database Table

Table	Occurrences
Warehouse	650
District	6,500
Customer	19,900,000
History	19,500,000
Order	19,500,000
New Order	5,850,000
Order Line	195,004,800
Stock	65,000,000
Item	100,000

4.2. Constant Values

The following values were used as the constant C input values to the NURand function during Build and Run time for this implementation.

Table 4.2: Constant C for NURand

Function	Value
C_LAST (Build)	123
C_LAST (Run)	233

4.3. Database Layout

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

Table 4.3 lists the distribution of the database over 80 disks and the transaction log over 4 mirrored pairs of disks for the benchmark configuration. In addition, there was one mirrored pair of disks containing Windows NT and SQL Server code and the Master database, plus two striped disks for the paging file. Another 7 disks were used for database backup. The priced system added 8 additional 8.5GB disks for the database to satisfy the 180-day space requirement. It also eliminated the 7 backup disks, 1 of the 4 mirrored pairs of log disks, and 2 of the 4 internal disks (no mirroring on system disk and only one paging disk rather than two).

4.4. DBMS: Data Model and DBMS Interface/Access Language

A statement must be provided that describes:

1. *The data model implemented by the DBMS used (e.g., relational, network, hierarchical).*
2. *The database interface (e.g., embedded, call level) and access language (e.g., SQL, DLI, COBOL, read/write) used to implement the TPC-C transactions. 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 SP3 (6.50.247) is a relational DBMS.

The client software interfaced to SQL Server through Stored Procedures invoked through Remote Procedure Calls embedded in the C application code. Specifically, DBLIB and TCP/IP sockets were used.

4.5. DBMS Partitions/Replications

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

No table partitioning or replication was done.

4.6. DBMS Space Requirements

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

Appendix E lists the space requirements for the 180-day space as well as the logical log space for eight hours.

Table 4.3: Benchmark Configuration Disk Storage Detail

Disk Adapter	Logical Drive	NT Disk #	Drive Letter	Drive Capacity (MB)	File System	SQL Device Name (OS, SQL Server)
1	1	0	C: (unused)	1024	NTFS	(OS, SQL Server)
	2	1	D: (unused)	1012	NTFS	(paging files)
	3	2	E: (unused)	4087	raw	tpc_misc1
			J: (unused)	510	raw	rpc_cs1
			O: (unused)	7805	raw	tpc_ol1
				3302	raw	
				54753	raw	
				510	raw	tpc_misc2
				7805	raw	tpc_cs2
				3302	raw	tpc_ol2
2	1	3	F: (unused)	55453	NTFS	tpc_dmp1
			V: (unused)	36264		
			G: (unused)	510	raw	tpc_misc3
			L: (unused)	7805	raw	tpc_cs3
			Q: (unused)	3302	raw	tpc_ol2
			H: (unused)	54753	raw	tpc_misc4
			M: (unused)	510	raw	tpc_cs4
			R: (unused)	7805	raw	tpc_ol4
				3302	raw	
				54753	raw	
3	1	4		52086	NTFS	tpc_dmp2
			W: (unused)	510	raw	tpc_misc5
			I: (unused)	7805	raw	tpc_cs5
			N: (unused)	3302	raw	tpc_ol5
			S: (unused)	54753	raw	
			T: (unused)	17359	raw	tpc_log1
			U: (unused)	17359	raw	tpc_log2
				17359	raw	
				17359	raw	
				17359	raw	
4	1 & 2	18	4,0GB	Usage of Disks		
	3	4	2,0GB	Database (2 unused)		
	1 & 2	8	4,0GB	OS, SQL Server & paging		
	1 & 2	10	8,5GB	Database		
	1 & 2	18	4,0GB	Database (2 unused)		
	1 & 2	16	4,0GB	Database (2 unused)		
	1 & 2	7	8,5GB	Database		
	1 & 2	16	4,0GB	Backup for database		
	1 & 2	8	8,5GB	Database		
	1 & 2	8	8,5GB	Logs for database		
5	1 & 2	18	4,0GB	Usage of Disks		
	3	4	2,0GB	Database (2 unused)		
	1 & 2	8	4,0GB	OS, SQL Server & paging		
	1 & 2	10	8,5GB	Database		
	1 & 2	18	4,0GB	Database (2 unused)		
	1 & 2	16	4,0GB	Database (2 unused)		
	1 & 2	7	8,5GB	Database		
	1 & 2	16	4,0GB	Backup for database		
	1 & 2	8	8,5GB	Database		
	1 & 2	8	8,5GB	Logs for database		
6	1	8	4,0GB	Usage of Disks		
	2	9	2,0GB	Database (2 unused)		
	1 & 2	18	4,0GB	OS, SQL Server & paging		
	3	4	2,0GB	Database (2 unused)		
	1 & 2	8	4,0GB	Database		
	1 & 2	10	8,5GB	Database (2 unused)		
	1 & 2	18	4,0GB	Database (2 unused)		
	1 & 2	16	4,0GB	Database		
	1 & 2	7	8,5GB	Backup for database		
	1 & 2	16	4,0GB	Database		
Benchmark #	Disks	Size of Disks	Change in # Disks	Reason for Change	Priced # Disks	
	4	2.0GB	-2	unused	2	
	76	4.0GB	-4	unused	72	
	25	8.5GB	-2,-7,-2,+8	unused, backup, too much log, extra for 180-day	22	
	105				96	

5. Clause 5: Performance Metrics & Response Time

5.1. Measured Throughput (tpmC)

Measured tpmC must be reported.

The measured tpmC was 7407.05.

5.2. Response Times

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

Table 5.1: Response Time Data

Transaction	Average	Maximum	90th %ile
New-Order	2.00	8.37	2.73
Payment	1.83	6.29	2.54
Delivery	0.29	0.97	0.31
Stock-Level	4.48	12.54	6.20
Order Status	2.48	11.57	3.36
Menu	0.21	1.08	0.32
Delivery (Deferred)	2.97	28.37	3.65

5.3. Keying and Think Times

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

Table 5.2: Keying Times

Transaction	Minimum	Average	Maximum
New-Order	18.00	18.02	18.48
Payment	3.00	3.02	3.54
Delivery	2.00	2.01	2.34
Stock-Level	2.00	2.01	2.20
Order Status	2.00	2.01	2.29

Table 5.3: Think Times

Transaction	Minimum	Average	Maximum
New-Order	0.00	12.07	120.51
Payment	0.00	12.03	120.53
Delivery	0.00	5.04	37.20
Stock-Level	0.00	5.06	42.73
Order Status	0.00	10.02	95.31

5.4. Response Time Frequency Distribution Curves

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

Figure 5.1: New Order Response Time Distribution

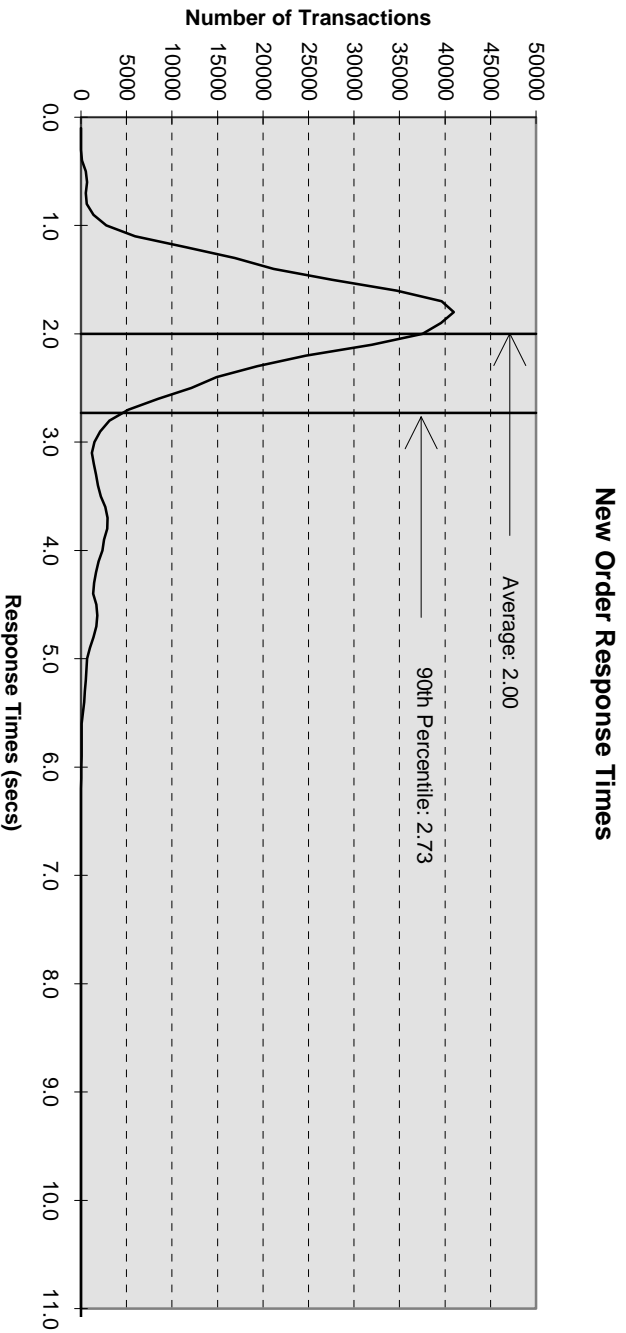


Figure 5.2: Payment Response Time Distribution

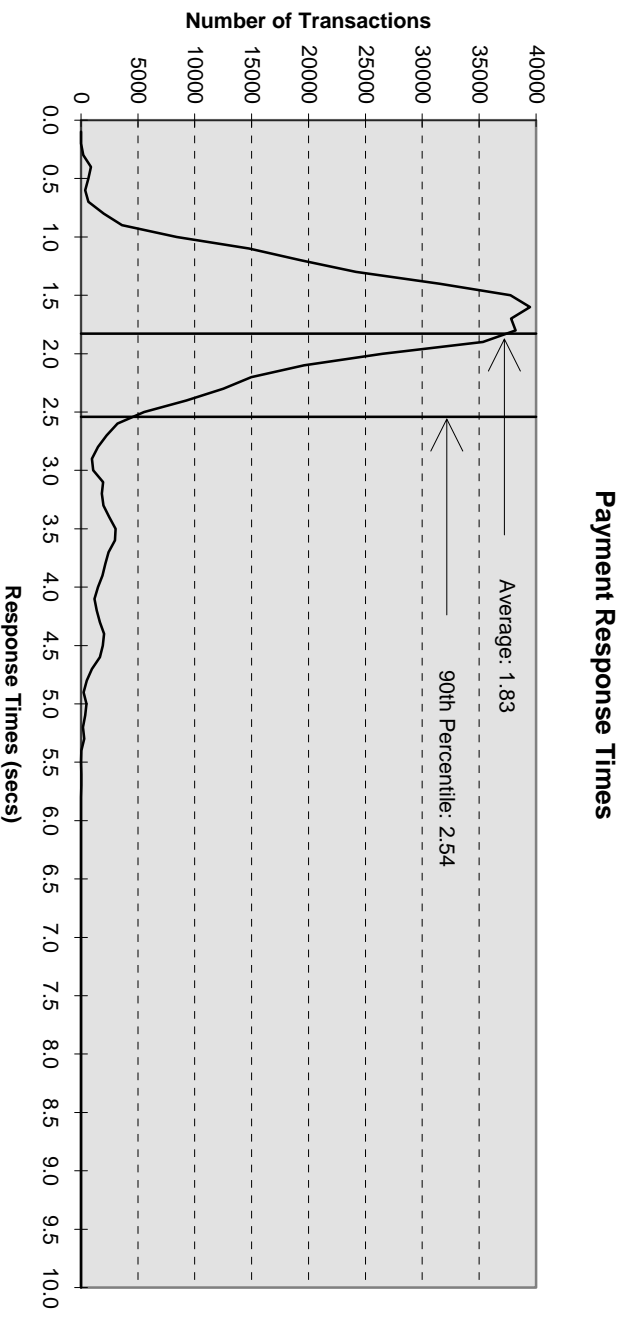


Figure 5.3: Order Status Response Time Distribution

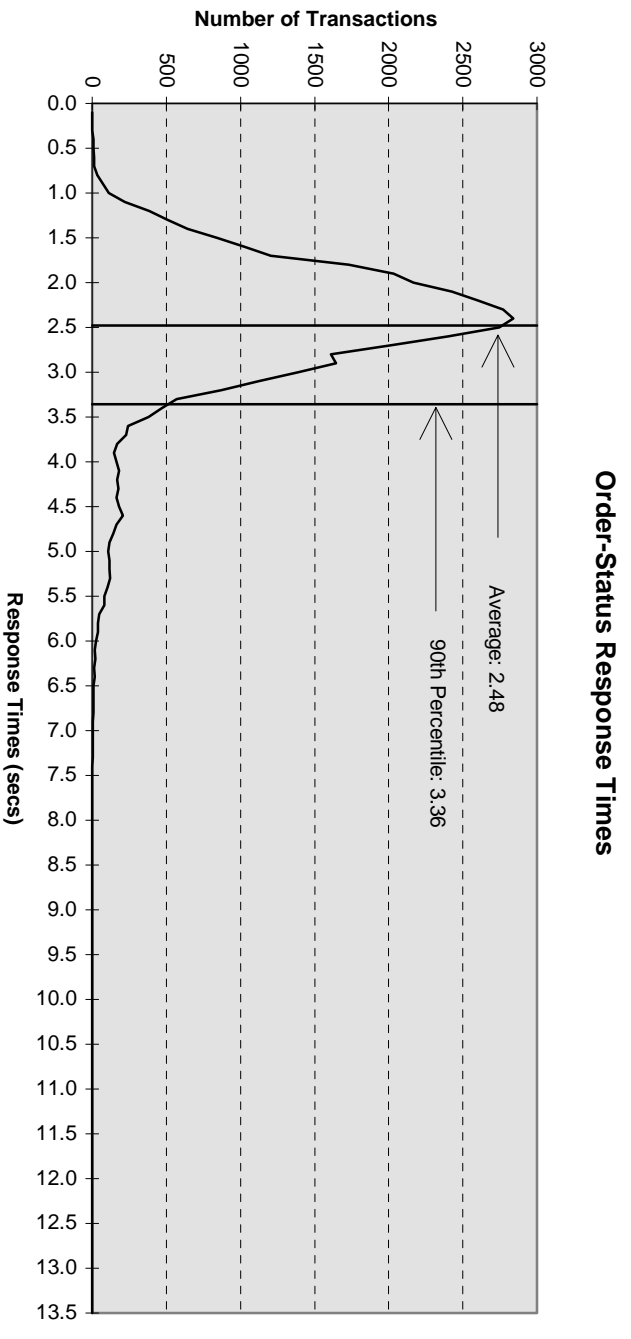


Figure 5.4: Delivery Response Time Distribution

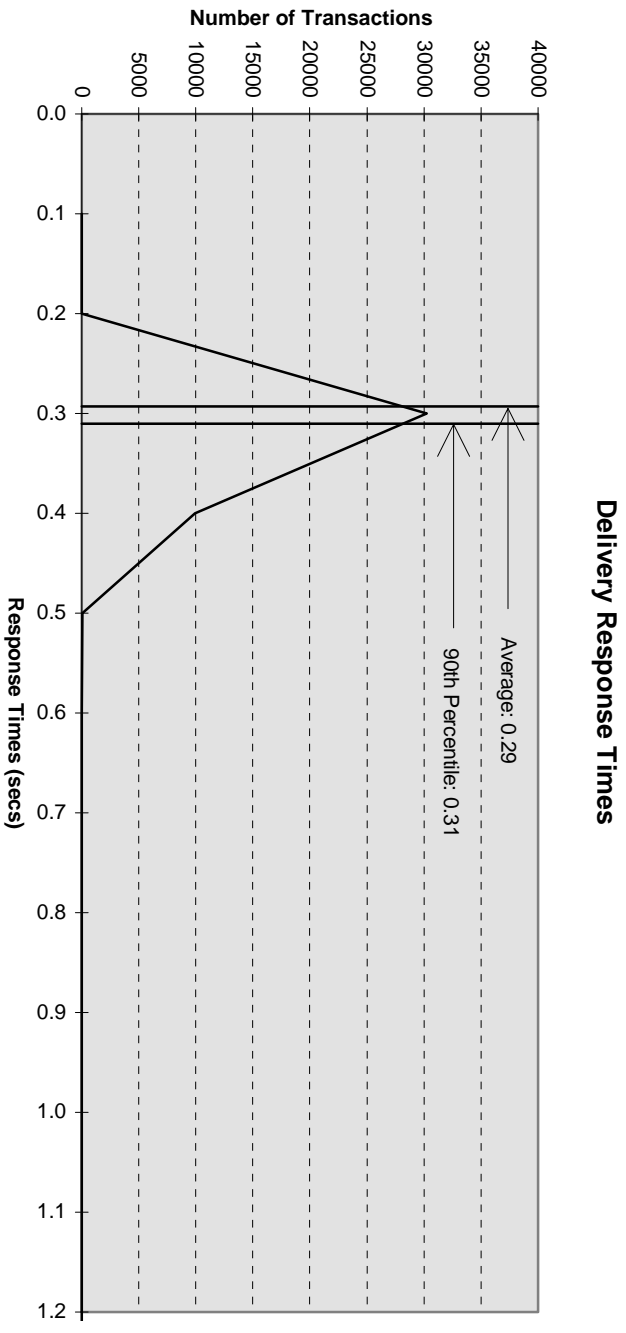
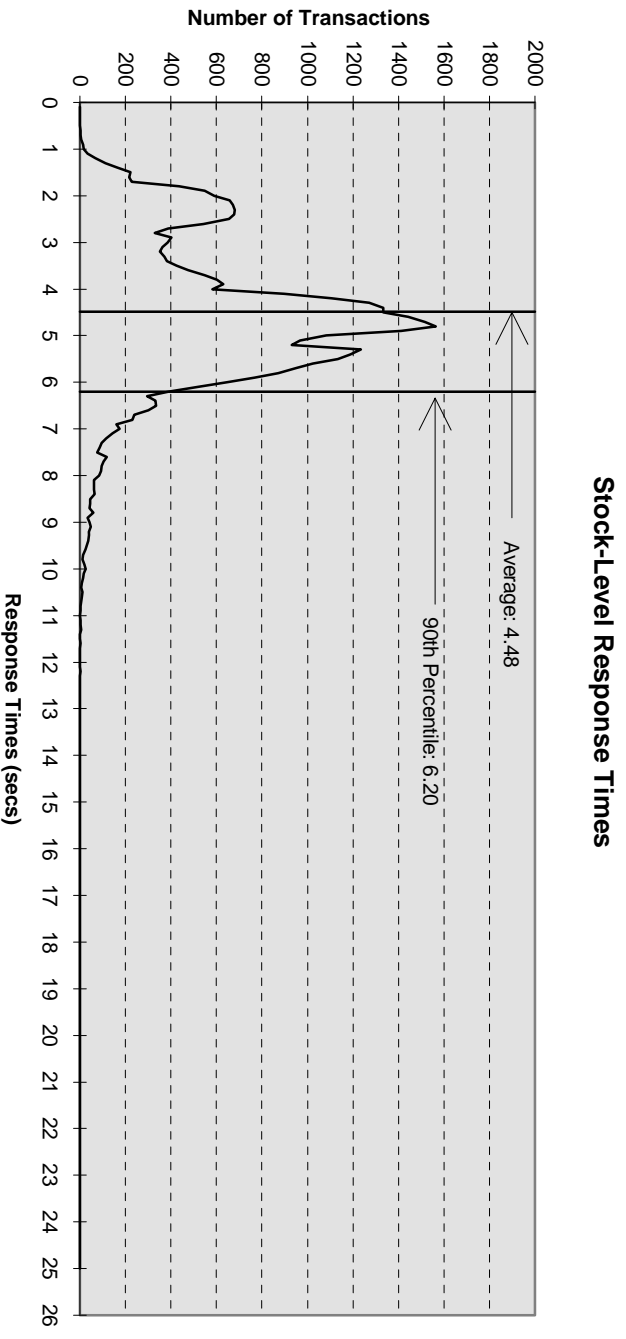


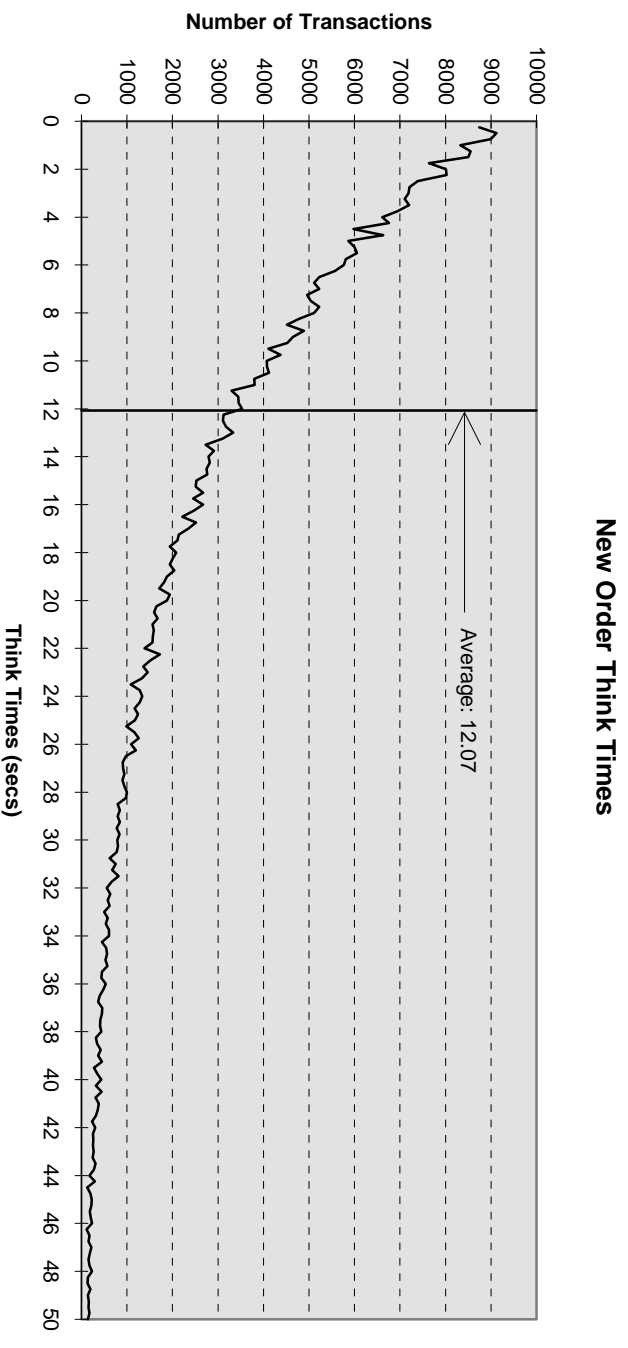
Figure 5.5: Stock Level Response Time Distribution



5.5. New Order Think Time Frequency Distribution Curve

Think Time frequency distribution curve (see Clause 5.6.3) must be reported for the New-Order transaction.

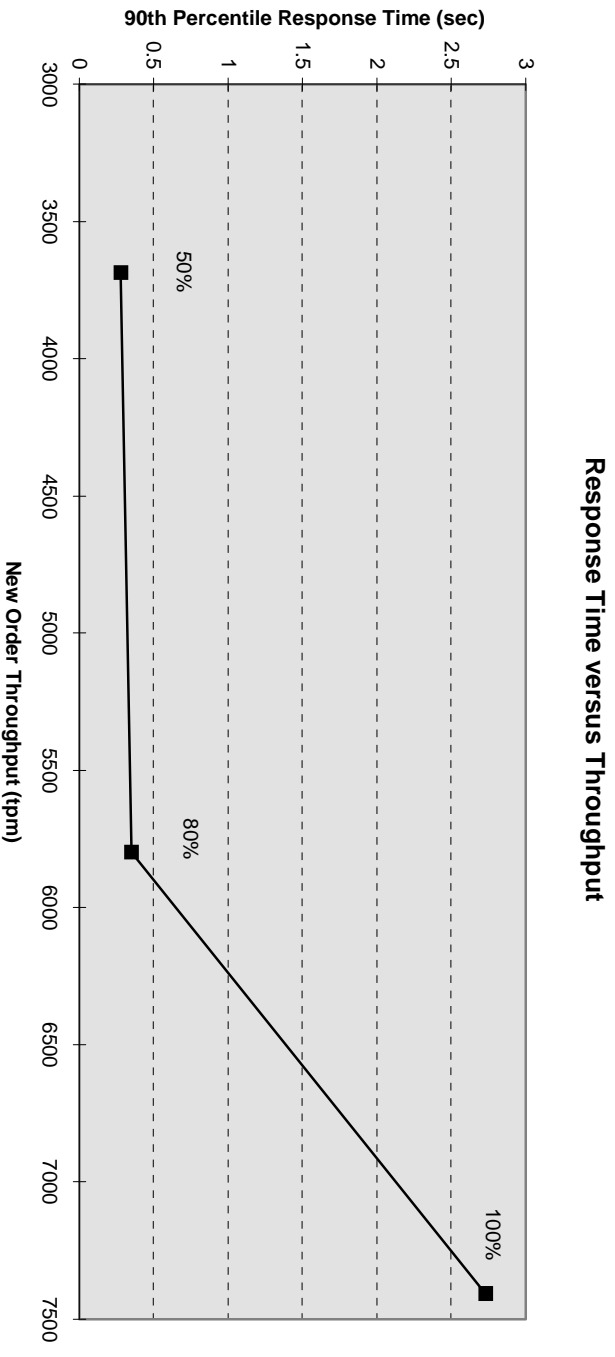
Figure 5.6: New Order Think Time Distribution



5.6. Response Time versus Throughput Performance Curve

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

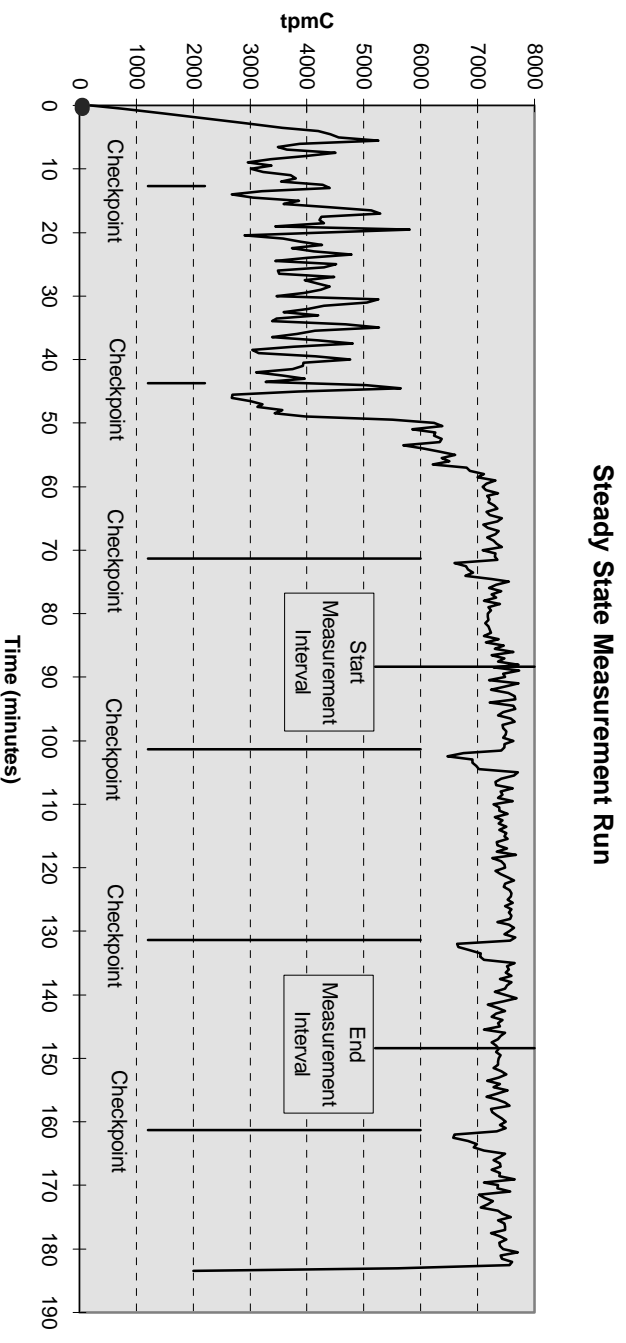
Figure 5.7: Response Time versus Throughput



5.7. New-Order Throughput vs. Time

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

Figure 5.8: Throughput (tpmC) versus Time



5.8. Determination of ‘Steady State’

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.

The transaction throughput rate (tpmC) and response time were relatively constant after the initial ‘ramp up’ period. The throughput and response time behavior were determined by examining data reported for each 30-second interval over the duration of the benchmark. Ramp-up, steady state, and ramp-down regions are discernible in the graph presented in Figure 5.8.

5.9. 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 generated the required input data to choose a transaction from the menu. This data was timestamped and captured in RTE log files before being transmitted. There was one log file for each driver on each client machine. Later, the output screen for the requested transaction was returned and also captured and timestamped in the RTE log files. The difference between these two timestamps is the response time for that transaction. An additional 0.1 second delay was added for the web browser emulation and for the menu delay.

The RTE then waited the required think time interval before repeating the process, starting with the selection of a transaction from the menu.

The RTE emulated web browsers (not terminals) in this client-server implementation, so RTE transmissions were sent (in HTML, using HTTP) to the web server running on the client machine through Ethernet LANs. The web server in turn passed the transaction requests on to the associated client applications. The latter processes then extracted the requests and submitted them to the database server over another Ethernet LAN using Microsoft SQL Server DBLIB and RPC through TCP/IP sockets. When the database transactions were completed, the results were encoded in HTML and send back (through the web server) to the (emulated) web browser (RTE).

All delivery requests were sent by the client application processes through a named pipe to a multi-threaded delivery server process running on the same client machine, which in turn submitted them to the database server and waited for them to complete. A delivery results file was written by the delivery server process to track delivery transaction queuing and database response times.

To perform checkpoints at specific intervals, SQL Server's checkpoint interval was set to the maximum allowable value and a script was written to schedule checkpoints at 30 minute intervals and record the start and end time of each checkpoint. The checkpoint script was started manually on one of the client machines after the RTE had all users logged in and sending transactions and a steady state had been achieved. SQL Server TRACE FLAG #3502 was also set in the Server's startup parameters, so that the start and end of each checkpoint would be recorded in the Server's ERRORLOG. Using this information, the positioning of the checkpoint within the measurement interval was verified to be clear of the guard zones.

At each checkpoint, SQL Server wrote to disk all database pages in memory that had been updated but not yet physically written to the disk. Upon completion of the checkpoint, SQL Server also wrote records to the transaction log indicating that a checkpoint had completed.

5.10. Reproducibility

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

In a repeat test, carried out in the same manner as the primary test, a throughput of 7302.3 tpmC was achieved on the same database during a 60-minute, steady state run. All required transaction statistics were met. See the Auditor's attestation letter for details.

5.11. Measurement Interval Duration

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

The measurement interval was 60 minutes.

5.12. Regulation of Transaction Mix

The method of regulation of the transaction mix (e.g. card decks or weighted random distribution) 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 weighed random distribution which was not adjusted during the run. The transaction weights were defined as follows:

NEW_ORDER_PROB	.4456
PAYMENT_PROB	.4320
DELIVER_PROB	.0408
STOCK_LEVEL_PROB	.0408
ORDER_STATUS_PROB	.0408

5.13. Transaction Statistics

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

The percentage of New-Order transactions 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 transactions must be disclosed.

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

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

Table 5.4 shows this information.

Table 5.4: Transaction Statistics

Transaction Type	Statistics	Value
New Order	Rollback transactions	1.00%
	Home warehouse	89.99%
	Remote warehouse	1.01%
	Average Items per Order	10.00
Payment	Home warehouse	84.91%
	Remote warehouse	15.09%
	Non-primary key access	60.07%
	Non-primary key access	59.23%
Order Status	Skipped transactions (Interactive)	0
Delivery	Skipped transaction counts (Deferred)	0
	Skipped District counts (Deferred)	0
Transaction Mix	New Order	44.76%
	Payment	43.06%
	Delivery	4.05%
	Stock-Level	4.06%
	Order-Status	4.07%

5.14. 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.

There are two checkpoints in the measurement interval. The first checkpoint starts 13 minutes into the measurement interval. The checkpoint interval is 30 minutes (from the start of one to the start of the next) and a checkpoint lasts approximately 3 minutes. In conformance with Clause 5.2.2 there is no checkpoint within a span of 12 minutes before or after the beginning or end of the measurement interval.

6. Clause 6: SUT, Driver & Communications Definition

6.1. Remote Terminal Emulator (RTE) Description

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

The RTE used was Microsoft BenchCraft RTE, version 0.5.2a. Appendix D contains the profile used as input to this RTE.

6.2. Emulated Components

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

There were no emulated components in the benchmark configuration other than the emulated web browsers on the users' PCs.

6.3. Functional Diagrams

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

Section 0.7 describes and shows functional diagrams of the benchmarked and priced systems.

6.4. Network Configuration

The network configuration of both the 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.

Figures 0.1 and 0.2 in Section 0.7 also diagram the network configurations of the benchmark and configured systems and represent the RTEs connected via LAN replacing the user PCs that are directly connected via LAN.

6.5. Network Bandwidth

The bandwidth of the network(s) used in the tested/priced configuration must be disclosed.

Ethernet local area networks (LAN) with a bandwidth of 10 megabits per second are used in the tested/priced configurations between RTE/emulated web browsers and the client machines. A single Ethernet LAN with a bandwidth of 100 megabits per second is used between the client machines and the database server (SUT).

Each of the 4 client machines contains one 100 megabit per second LAN adapter and two 10 megabit per second LAN adapters. The 100 megabit per second LAN adapter is connected to a single LAN segment and to the database server in both priced and tested configurations.

In the priced configuration, the 4 client machines are connected to 8 LAN segments, each supporting 800 workstations (PCs running web browsers).

In the tested configuration, there are 2 RTE machines with two 10 megabit per second LAN adapters each. A total of 4 LAN segments are used to connect the 4 clients machines to the 2 RTE machines. Each LAN adapter on each RTE connects to one LAN adapter on each of two clients, so that each RTE connects to all 4 clients and each client is connected to both RTE machines. Thus, while each LAN segment supports 1,600 users and each client supports 1,600 users, the LAN adapter on each client supports only 800 users, just as in the priced configuration.

6.6. Operator Intervention

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

No operator intervention was required to sustain eight hours of operation at the reported throughput.

7.

Clause 7: Pricing

7.1. 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.

A detailed list of hardware and software components along with their part numbers and prices are given in the Executive Summary near the beginning of this document.

7.1.1. System Pricing

Each priced configuration consists of an integrated system package, additional options, and components. Prices for all ALR products are US list prices.

7.1.2. Maintenance Pricing

The ALR standard factory warranty covers major components for five (5) years, with associated labor costs covered for thirty-six (36) months. All other peripheral components (excluding monitors) are warranted for thirty-six (36) months from time of delivery to the original purchaser. The Microsoft support pricing is based on sixty (60) months of monthly support costs.

ALR ProCare extended coverage plan "C" is a five-year, four-hour response time, same business day plan.

Netlux provide return-to-factory replacement within seven (7) days. Appropriate spares are included in the priced configuration.

7.1.3. Discounts

No discounts were applied to the priced configuration.

7.2. Availability

The committed delivery date for general availability (availability date) of products used in the price calculation 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.

The hardware, software and support/maintenance products priced in this benchmark are detailed on page vi.

All products in the priced system will be available on 30th April 1997.

7.3. Measured tpmC, Price/Performance, and Availability Date

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

ALR Revolution Quad6 SD512, with Microsoft Windows NT 4.0 and SQL Server 6.5, was 7407.05 tpmC at \$57.78 per tpmC. Available on 30th April 1997.

7.4. 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.

None.

7.5. 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:

- Five (5) Microsoft Windows NT 4.0 licenses
- One (1) Microsoft SQL Server v.6.50 license
- One (1) Microsoft Internet Connector License
- One (1) Microsoft SQL Server Programmers Toolkit
- One (1) Microsoft Visual C++ Subscription

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

8.

Clause 8 : Full Disclosure Availability

8.1. Availability

The Full Disclosure Report must be readily available to the public at a reasonable charge, similar to charges for similar documents by that test sponsor.

Copies of this Full Disclosure Report may be obtained by contacting:

TPC Benchmark Administrator
Advanced Logic Research, Inc.
9401 Jeronimo Road
Irvine, CA 92618
USA

9.

Clause 9 : Audit

9.1. Auditor's Report

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

This implementation of the TPC Benchmark C on the ALR Revolution Quad6 SD512 was audited by Richard Gimarc, a TPC certified auditor of:

2229 Benita Drive, Suite 101,
Rancho Cordova, CA 95670.

(916)635-2822 Fax: (916) 858-0109
e-mail: Lorna@PerfMetrics.com

The attestation letter is shown on the next page.

PERFORMANCE METRICS INC.
TPC Certified Auditors

April 3, 1997

Cameron Spears
 Software Engineering Manager
 Advanced Logic Research, Inc.
 9401 Jeronimo
 Irvine, CA 92618

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

Platform: ALR Revolution Quad6 SD512
 Database Manager: Microsoft SQL Server 6.5 SP3 (6.50.247)
 Operating System: Microsoft Windows NT Server 4.0 SP1

CPU's	Memory	Disks	New-Order Response Time @ 90%	tpmc
Server: ALR Revolution Quad6 SD512				
4 Pentium Pro @ 200 MHz	Main: 2 GB L2 Cache: 512 KB	4 @ 1.99 GB 76 @ 4.05 GB 25 @ 8.47 GB	2.73 sec.	7,407.05
4 Clients: Unisys Aquanta DM6				
1 Pentium Pro @ 200 MHz	Main: 128 MB L2 Cache: 256 KB	1 @ 1.51 GB	n.a.	n.a.

In my opinion, these performance results were produced in compliance with the TPC requirements for Revision 3.2.3 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 650 warehouses. Only 640 warehouses were active during measurement. See additional audit note below.
- The ACID properties were met, including phantom protection.
- The durability data loss and log loss tests was performed on a 16-warehouse database.
- Input data was generated according to the specified percentages.

2229 Benita Drive, Suite 101, Rancho Cordova, CA 95670
 (916) 635-2822 Fax: (916) 858-0109 e-mail: Lorna@PerfMetrics.com

Page 1

PERFORMANCE METRICS INC.
TPC Certified Auditors

- Eight hours of mirrored log space was configured on the measured system. Two 8.47 GB log disks were removed from the priced configuration since they were not needed to satisfy the 8 hour requirement.
- The following disks contained backup and other data and were not active during measurement: four 4.05 GB disks and nine 8.47 GB disks.
- Two of the 1.99 GB disks on the server are not priced. One of these disks was the mirror of the boot drive (C:) and the second was part of the 3-disk D: stripe set. Measurement data showed negligible activity on these disks during measurement.
- The data for the 180-day space calculation was verified. Eight 8.47 GB disks were added to the priced configuration.
- Measurement cycle times include a 0.1 second menu and a 0.1 second response time delay for an emulated Web browser.
- The steady state portion of the test was 60 minutes.
- Two checkpoints were taken during the steady state portion of the test.
- Checkpoints were verified to be clear of the guard zones.
- There were 6,400 user contexts present on the system.
- Each emulated user had a different random number stream.
- The NURand constants used for database load and at run time were verified.
- System pricing was checked for major components and maintenance.

Additional Audit Notes:

- The benchmark was executed on a Unisys Aquanta QRV6 server. All active components on the Unisys Aquanta QRV6 and the ALR Revolution Qrv6 SD512 are identical.
- On file primary performance run, rows for warehouses 641 through 650 were not deleted from the Warehouse table prior to measurement. Sufficient data was collected after the run to demonstrate that these warehouses were not active.

Regards,

Richard L. Gimare

Richard L. Gimare
Auditor

2229 Benita Drive, Suite 101, Rancho Cordova, CA 95670
(916) 635-2822 Fax: (916) 858-0109 e-mail: Lorna@PerfMetrics.com

Page 2

Appendix A - Client/Server Source

MAKEFILE

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

!IF "$(SQL_LOC)" == ""
SQL_LOC=D:\MSDEV
!MESSAGE No SQL_LOC specified. Defaulting to C:\MSSQL\DBLIB
!ENDIF

!IF "$(CFG)" != "Release" && "$(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
!ERROR An invalid configuration is specified.
!ENDIF

OUTDIR      = .
SRCDIR     = .\Src
OBJDIR     = .\Objs
OUTDIR     = .\Bin
ODBC       = d:\msdev
DBLIB      = $(SQL_LOC)
DBLIBINC   = $(DBLIB)\INCLUDE
ODBCINCDIR = $(ODBC)\INCLUDE
DBLIBDIR   = $(DBLIB)\LIB
ODBCLIBDIR = $(ODBC)\LIB

!IF "$(CFG)" != "Debug"
LDEBUG     =
CDEBUG     =
LDEBUG_RG  =
CDEBUG_RG  =

```

```
DEBUG      =
FLAGS     = /D "WIN32" /D "_WINDOWS"
OPT       = /Ot
!ELSE
LDEBUG    = /debug /pdb:$(OBJDIR)\tpcc1.pdb
CDEBUG    = /Zi /Yd
LDEBUG_RG = /debug /pdb:$(OBJDIR)\install.pdb
CDEBUG_RG = /Zi /Yd /Fd$(OBJDIR)\install.pdb
FLAGS     = /D "_DEBUG" /D "WIN32" /D "_WINDOWS"
OPT       = /Od
!ENDIF

LINK32_LIBS1 = user32.lib msacm32.lib advapi32.lib
$(DBLIBDIR)\ntwdblib.lib
LINK32_OBJS1 = "$(OBJDIR)\tpcc1.obj" "$(OBJDIR)\tpcc1.res"
LINK32_DEF1  = "$(SRCDIR)\tpcc1.def"
LINK32_FLAGS1 = /nologo /subsystem:windows /dll /incremental:no
$(LDEBUG) /def:"$(LINK32_DEF1)" /out:"$(OBJDIR)\tpcc1.dll"

LINK32_LIBS2 = user32.lib msacm32.lib advapi32.lib
$(ODBCLIBDIR)\odbc32.LIB
LINK32_OBJS2 = "$(OBJDIR)\tpcc2.obj" "$(OBJDIR)\tpcc2.res"
LINK32_DEF2  = "$(SRCDIR)\tpcc2.def"
LINK32_FLAGS2 = /nologo /subsystem:windows /dll /incremental:no
$(LDEBUG) /def:"$(LINK32_DEF2)" /out:"$(OBJDIR)\tpcc2.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)\install.exe

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

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

"$(OBJDIR)\tpcc1.obj": "$(SRCDIR)\tpcc.c" "$(SRCDIR)\tpcc.h"
cl.exe /nologo /MT /W3 $(CDEBUG) $(OPT) /I$(DBLIBINC) $(FLAGS)
/Fd$(OBJDIR)\tpcc1.pdb /Fo$(OBJDIR)\tpcc1.obj /c "$(SRCDIR)\tpcc.c"

$(OBJDIR)\tpcc1.res: $(SRCDIR)\tpcc1.rc
```

```

rc.exe /l 0x409 /fo $(OBJDIR)\tpcc1.res $(FLAGS)
$(SRCDIR)\tpcc1.rc

$(OBJDIR)\tpcc1.dll: $(LINK32_OBJS1) $(LINK32_DEF1)
link.exe $(LINK32_FLAGS1) $(LINK32_OBJS1) $(LINK32_LIBS1)

"$(OBJDIR)\tpcc2.obj": "$(SRCDIR)\tpcc.c" "$(SRCDIR)\tpcc.h"
cl.exe /nologo /MT /W3 $(CDEBUG) $(OPT) /I$(ODBCINCDIR) $(FLAGS)
/Fd$(OBJDIR)\tpcc2.pdb /Fo$(OBJDIR)\tpcc2.obj /c /D"USE_ODBC"
"$(SRCDIR)\tpcc.c"

$(OBJDIR)\tpcc2.res: $(SRCDIR)\tpcc2.rc
rc.exe /l 0x409 /fo $(OBJDIR)\tpcc2.res $(FLAGS)
$(SRCDIR)\tpcc2.rc

$(OBJDIR)\tpcc2.dll: $(LINK32_OBJS2) $(LINK32_DEF2)
link.exe $(LINK32_FLAGS2) $(LINK32_OBJS2) $(LINK32_LIBS2)

$(OBJDIR)\delisrv1.exe: $(SRCDIR)\delisrv.c $(SRCDIR)\delisrv.h
cl.exe /nologo /MT /W3 $(CDEBUG) $(OPT) /I $(DBLIBINC) $(FLAGS)
/Fo$(OBJDIR)\delisrv.obj $(SRCDIR)\delisrv.c /link
/out:$(OBJDIR)\delisrv1.exe $(DBLIBDIR)\ntwdblib.lib msacm32.lib
advapi32.lib

$(OBJDIR)\delisrv2.exe: $(SRCDIR)\delisrv.c $(SRCDIR)\delisrv.h
cl.exe /nologo /MT /W3 $(CDEBUG) $(OPT) /I$(ODBCINCDIR) $(FLAGS)
/Fo$(OBJDIR)\delisrv.obj $(SRCDIR)\delisrv.c /D"USE_ODBC" /link
/out:$(OBJDIR)\delisrv2.exe $(ODBCLIBDIR)\odbc32.lib msacm32.lib
advapi32.lib

$(OBJDIR)\install.res: $(SRCDIR)\install.rc $(OBJDIR)\tpcc1.dll
$(OBJDIR)\tpcc2.dll $(OBJDIR)\delisrv1.exe $(OBJDIR)\delisrv2.exe
rc.exe /l 0x409 /fo$(OBJDIR)\install.res /i $(OBJDIR) /i $(SRCDIR)
$(FLAGS) $(SRCDIR)\install.rc

$(OBJDIR)\install.obj: $(SRCDIR)\install.c $(OBJDIR)\tpcc1.dll
$(OBJDIR)\tpcc2.dll $(OBJDIR)\delisrv1.exe $(OBJDIR)\delisrv2.exe
$(OBJDIR)\install.res
cl -W3 $(CDEBUG_RG) /Fo$(OBJDIR)\install.obj /c
$(SRCDIR)\install.c

$(OUTDIR)\install.exe: $(OBJDIR)\install.obj $(OBJDIR)\install.res
link.exe @<<
$(LINK32_FLAGS_RG) $(LINK32_OBJS_RG) $(LINK32_LIBS_RG)
<<

```

DELISRV.H

```

/* FILE: DELISRV.H, MSTPCC.300
* -----
* Microsoft TPC-C Kit Ver. 3.00.000
* Audited 08/23/96, By Francois Raab
*
* Copyright Microsoft, 1996
*
* PURPOSE: Header file for delivery service executable
* Author: Philip Durr
* philipdu@Microsoft.com
*/

```

```

#define AVAILABLE 0
//queue array element available
#define WRITE_LOCKED 1
//queue array element is being written to
#define READ_LOCKED 2
//queue array element is begin read
#define INUSE 4
//queue array element has information stored in it

#define CTRL_C 3
//<Ctrl> C, exit key code

#define DEFCLPCKSIZE 4096 //default DB
Library SQL Connection pack size

#define ERR_SUCCESS 0
//Success, no error.
#define ERR_CANNOT_CREATE_THREAD 1000 //Cannot create
thread.
#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
results file.
#define ERR_CANNOT_OPEN_PIPE 1006 //Cannot open
delivery pipe.
#define ERR_READ_PIPE 1007 //Error
reading pipe
#define ERR_INSUFFICIENT_MEMORY 1008 //insufficient
memory
#define ERR_ODBC_SQLALLOCENV 1009 //Cannot allocated
ODBC env handle
#define ERR_SQL_ATTR_ODBC_VERSION 1010 //Cannot set ODBC
version
#define ERR_SQL_ATTR_CONNECTION_POOLING 1011 //Cannot set
Connection Pooling

typedef struct _DELIVERY_TRANSACTION
{
    SYSTEMTIME queue; //time delivery transaction
    short w_id; //delivery warehouse
    short o_carrier_id; //carrier id
} DELIVERY_TRANSACTION;

typedef DELIVERY_TRANSACTION *LPDELIVERY_TRANSACTION; //pointer to
delivery transaction queue

typedef struct _DELIVERY_PACKET
{
    BOOL bInUse;
    //entry current in use
    OVERLAPPED ov;
    //pipe io overlapped structure
    DELIVERY_TRANSACTION trans; //delivery
transaction information

```

```

} DELIVERY_PACKET, *LPDELIVERY_PACKET;

typedef struct _SERRORMSG
{
    int          iError;          //error message id
    char        szMsg[80];       //error message
} SERRORMSG;

#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

//delivery transaction structure
typedef struct DELIVERY
{
    short        w_id;           //warehouse id
    short        o_carrier_id;   //carrier id
    int          spid;           //db library spid
    long         o_id[10];       //returned delivery
transaction ids
    DBPROCESS    *dbproc;       //db library DBPROCESS
pointer
    SYSTEMTIME   queue;        //delivery transaction queue
time
    SYSTEMTIME   trans_end;     //delivery transaction
finished time
} DELIVERY;

typedef DELIVERY *LPDELIVERY; //pointer to delivery structure

//function prototypes
void          main(int argc, char *argv[]);
static void   cls(void);
static int    RunDelivery(void);
static void   QuitStatus(void);
static void   AnimateWait1(void);
static void   AnimateWait(void);
static int    Init(void);
static void   Restore(void);
static void   ErrorMessage(int iError);
static BOOL   GetParameters(int argc, char *argv[]);
static void   PrintParameters(void);
static void   PrintHeader(void);
static int    ReadRegistrySettings(void);
static void   CheckKey(void *ptr);
static void   DeliveryHandler( void *ptr );
static void   DeliveryThread( void *ptr );

#endif USE_ODBC

```

```

static int          err_handler(DBPROCESS *dbproc, int severity,
int dberr, int oserr, char *dberrstr, char *oserrstr);
#endif

#ifdef USE_ODBC
#define DBINT int
#endif

static int          msg_handler(DBPROCESS *dbproc, DBINT msgno, int
msgstate, int severity, char *msgtext);
static BOOL        SQLOpenConnection(DBPROCESS **dbproc, char *server, char
*database, char *user, char *password, int *spid);
static void        WriteLog(LPDELIVERY pDelivery);
static void        CalculateElapsedTime(int *pElapsed, LPSYSTEMTIME lpBegin,
LPSYSTEMTIME lpEnd);
static int         SQLDelivery(DELIVERY *pDelivery);
static BOOL        SQLDetectDeadlock(DBPROCESS *dbproc);
static BOOL        ReadDeliveryInfo(short *w_id, short *o_carrier_id);
static BOOL        PostDeliveryInfo(short w_id, short o_carrier_id);
static int         OpenLogFile(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, SQLSMALLINT icol, SQLSMALLINT
fCType, SQLPOINTER rgbValue, SQLINTEGER cbValueMax, SDWORD *piLength);
BOOL GetResults(PDBPROCESS dbproc);
BOOL MoreResults(PDBPROCESS dbproc);
BOOL ReopenConnection(PDBPROCESS dbproc);
#endif

```

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_HTTPPEXTENSIONPROC ) ( EXTENSION_CONTROL_BLOCK
*pECB );

#ifdef __cplusplus
}
#endif
#endif // end definition _HTTPTEXT_H_

```

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 IDR_TPCCDLL2 104
#define IDD_DIALOG2 105
#define IDI_ICON2 106
#define IDR_DELIVERY1 107
#define IDD_DIALOG3 108
#define IDR_DELIVERY2 109
```

```
#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
#define IDC_DBLIB 1021
#define IDC_ODBC 1022
#define IDC_CONNECT_POOL 1023
#define ED_USER_CONNECT_DELAY_TIME 1024
```

```
// Next default values for new objects
```

```
//
#ifdef APSTUDIO_INVOKED
#ifdef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE 111
#define _APS_NEXT_COMMAND_VALUE 40001
#define _APS_NEXT_CONTROL_VALUE 1022
#define _APS_NEXT_SYMED_VALUE 101
#endif
#endif
```

RESOURCE.H

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

```
// Next default values for new objects
```

```
//
#ifdef APSTUDIO_INVOKED
#ifdef 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
```

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
 */
//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 ERR_BAD_ITEM_ID
1 //expected abort record in txnRecord
#define ERR_TYPE_DELIVERY_POST 2
//expected delivery post failed
#define ERR_TYPE_WEBDDL 3
//tpcc web generated error
#define ERR_TYPE_SQL 4
//sql server generated error
#define ERR_TYPE_DBLIB 5
//dblib generated error
#define ERR_TYPE_ODBC 6
//odbc generated error
#define ERR_TYPE_SOCKET 7
//error on communication socket client rte only
#define ERR_TYPE_DEADLOCK 8
//dblib and odbc only deadlock condition

#define ERR_SUCCESS
1000 //Success, no error.
#define ERR_COMMAND_UNDEFINED 1001
//Command undefined.
#define ERR_NOT_IMPLEMENTED_YET 1002
//Not Implemented Yet.
#define ERR_CANNOT_INIT_TERMINAL 1003
//Cannot initialize client connection.
#define ERR_OUT_OF_MEMORY 1004
//insufficient memory.
#define ERR_NEW_ORDER_NOT_PROCESSED 1005
//Cannot process new Order form.
#define ERR_PAYMENT_NOT_PROCESSED 1006
//Cannot process payment form.
#define ERR_NO_SERVER_SPECIFIED 1007
//No Server name specified.
```

#define ERR_ORDER_STATUS_NOT_PROCESSED	1008	1008	//#Payment
//#Cannot process order status form.			
#define ERR_W_ID_INVALID	1009	1038	//#New Order
//#Invalid Warehouse ID.			
#define ERR_CAN_NOT_SET_MAX_CONNECTIONS	1010	1039	//#New Order
//#Insufficient memory to allocate # connections.			
#define ERR_NOSUCH_CUSTOMER	1011	1040	//#New Order
//#No such customer.			
#define ERR_D_ID_INVALID	1012	1041	//#New Order
//#Invalid District ID Must be 1 to 10.			
#define ERR_MAX_CONNECT_PARAM	1013	1042	//#Payment
client connections exceeded, run install to increase.	1013	1043	//#Payment
#define ERR_INVALID_SYNC_CONNECTION	1014	1044	//#Payment
//#Invalid Terminal Sync ID.			
#define ERR_INVALID_TERMID	1015	1045	//#Payment
//#Invalid Terminal ID.			
#define ERR_PAYMENT_INVALID_CUSTOMER	1016	1046	//#Payment
Form, No such Customer.			
#define ERR_SQL_OPEN_CONNECTION	1017	1047	//#Payment
//#SQLOpenConnection API Failed.			
#define ERR_STOCKLEVEL_MISSING_THRESHOLD_KEY	1018	1048	//#Payment
missing Threshold key "TT".			
#define ERR_STOCKLEVEL_THRESHOLD_INVALID	1019	1049	//#Payment
Threshold invalid data type range = 1 - 99.			
#define ERR_STOCKLEVEL_THRESHOLD_RANGE	1020	1050	//#Payment
//#Stock Level Threshold out of range, range must be 1 - 99.			
#define ERR_STOCKLEVEL_NOT_PROCESSED	1021	1051	//#Payment
not processed.			
#define ERR_NEWORDER_FORM_MISSING_DID	1022	1052	//#Payment
missing District key "DID".			
#define ERR_NEWORDER_DISTRICT_INVALID	1023	1053	//#Payment
District ID Invalid range 1 - 10.			
#define ERR_NEWORDER_DISTRICT_RANGE	1024	1054	//#Payment
Order District ID out of Range. Range = 1 - 10.			
#define ERR_NEWORDER_CUSTOMER_KEY	1025	1055	//#Payment
Order missing Customer key "CID".			
#define ERR_NEWORDER_CUSTOMER_INVALID	1026	1056	//#Payment
customer id invalid data type, range = 1 to 3000.			
#define ERR_NEWORDER_CUSTOMER_RANGE	1027	1057	//#Payment
Order customer id out of range, range = 1 to 3000.			
#define ERR_NEWORDER_MISSING_IID_KEY	1028	1058	//#Payment
missing Item Id key "IID".			
#define ERR_NEWORDER_ITEM_BLANK_LINES	1029	1059	//#Payment
blank order lines all orders must be continuous.			
#define ERR_NEWORDER_ITEMID_INVALID	1030	1060	//#Payment
Order Item Id is wrong data type, must be numeric.			
#define ERR_NEWORDER_MISSING_SUPPW_KEY	1031	1061	//#Payment
Order missing Supp W key "SP###".			
#define ERR_NEWORDER_SUPPW_INVALID	1032	1062	//#Payment
Order Supp W invalid data type must be numeric.			
#define ERR_NEWORDER_MISSING_QTY_KEY	1033	1063	//#Payment
Missing Qty key "Qty###".			
#define ERR_NEWORDER_QTY_INVALID	1034	1064	//#Payment
Order Qty invalid must be numeric range 1 - 99.			
#define ERR_NEWORDER_SUPPW_RANGE	1035	1065	//#Payment
Order Supp W value out of range range = 1 - Max Warehouses.			
#define ERR_NEWORDER_ITEMID_RANGE	1036	1066	//#Payment
Order Item Id is out of range. Range = 1 to 999999.			
#define ERR_NEWORDER_QTY_RANGE	1037	1067	//#Payment
Order Qty is out of range. Range = 1 to 99.			
#define ERR_PAYMENT_DISTRICT_INVALID	1038	1038	//#Payment
District ID is invalid must be 1 - 10.			
#define ERR_NEWORDER_SUPPW_WITHOUT_ITEMID	1039	1039	//#New Order
Supp_W field entered without a corresponding Item_Id.			
#define ERR_NEWORDER_QTY_WITHOUT_ITEMID	1040	1040	//#New Order
Order Qty entered without a corresponding Item_Id.			
#define ERR_NEWORDER_NOITEMS_ENTERED	1041	1041	//#New Order
Blank Items between items, items must be continuous.			
#define ERR_PAYMENT_MISSING_DID_KEY	1042	1042	//#Payment
//#Payment missing District Key "DID".			
#define ERR_PAYMENT_DISTRICT_RANGE	1043	1043	//#Payment
//#Payment District Out of range, range = 1 - 10.			
#define ERR_PAYMENT_MISSING_CID_KEY	1044	1044	//#Payment
//#Payment missing Customer Key "CID".			
#define ERR_PAYMENT_CUSTOMER_INVALID	1045	1045	//#Payment
Customer data type invalid, must be numeric.			
#define ERR_PAYMENT_MISSING_CLT	1046	1046	//#Payment
//#Payment missing Customer Last Name Key "CLT".			
#define ERR_PAYMENT_LAST_NAME_TO_LONG	1047	1047	//#Payment
Customer last name longer than 16 characters.			
#define ERR_PAYMENT_CUSTOMER_RANGE	1048	1048	//#Payment
//#Payment Customer ID out of range, must be 1 to 3000.			
#define ERR_PAYMENT_CID_AND_CLT	1049	1049	//#Payment
//#Payment Customer ID and Last Name entered must be one or other.			
#define ERR_PAYMENT_MISSING_CDI_KEY	1050	1050	//#Payment
//#Payment missing Customer district key "CDI".			
#define ERR_PAYMENT_CDI_INVALID	1051	1051	//#Payment
//#Payment Customer district invalid must be numeric.			
#define ERR_PAYMENT_CDI_RANGE	1052	1052	//#Payment
//#Payment Customer district out of range must be 1 - 10.			
#define ERR_PAYMENT_MISSING_CWI_KEY	1053	1053	//#Payment
//#Payment missing Customer Warehouse key "CWI".			
#define ERR_PAYMENT_CWI_INVALID	1054	1054	//#Payment
//#Payment Customer Warehouse invalid must be numeric.			
#define ERR_PAYMENT_CWI_RANGE	1055	1055	//#Payment
//#Payment Customer Warehouse out of range, 1 to Max Warehouses.			
#define ERR_PAYMENT_MISSING_HAM_KEY	1056	1056	//#Payment
//#Payment missing Amount key "HAM".			
#define ERR_PAYMENT_HAM_INVALID	1057	1057	//#Payment
//#Payment Amount invalid data type must be numeric.			
#define ERR_PAYMENT_HAM_RANGE	1058	1058	//#Payment
//#Payment Amount out of range, 0 - 9999.99.			
#define ERR_ORDERSTATUS_MISSING_DID_KEY	1059	1059	//#Payment
//#Order Status missing District key "DID".			
#define ERR_ORDERSTATUS_DID_INVALID	1060	1060	//#Payment
//#Order Status District invalid, value must be numeric 1 - 10.			
#define ERR_ORDERSTATUS_DID_RANGE	1061	1061	//#Payment
//#Order Status District out of range must be 1 - 10.			
#define ERR_ORDERSTATUS_MISSING_CID_KEY	1062	1062	//#Payment
//#Order Status missing Customer key "CID".			
#define ERR_ORDERSTATUS_MISSING_CLT_KEY	1063	1063	//#Payment
//#Order Status missing Customer Last Name key "CLT".			
#define ERR_ORDERSTATUS_CLT_RANGE	1064	1064	//#Payment
//#Order Status Customer last name longer than 16 characters.			
#define ERR_ORDERSTATUS_CID_INVALID	1065	1065	//#Payment
//#Order Status Customer ID invalid, range must be numeric 1 - 3000.			
#define ERR_ORDERSTATUS_CID_RANGE	1066	1066	//#Payment
//#Order Status Customer ID out of range must be 1 - 3000.			


```

#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*\".

//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
    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 id 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
} 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

//TERMID= 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;
    int iSyncId;
    //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 DllMain(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 *pSyncId);
void NewOrderForm(EXTENSION_CONTROL_BLOCK *pECB, int iFormId, int iTermId,
int iSyncId);
void PaymentForm(EXTENSION_CONTROL_BLOCK *pECB, int iFormId, int iTermId,
int iSyncId);
void DeliveryForm(EXTENSION_CONTROL_BLOCK *pECB, int iFormId, int iTermId,
int iSyncId);
void OrderStatusForm(EXTENSION_CONTROL_BLOCK *pECB, int iFormId, int
iTermId, int iSyncId);
void StockLevelForm(EXTENSION_CONTROL_BLOCK *pECB, int iFormId, int
iTermId, int iSyncId);
void ExitCmd(EXTENSION_CONTROL_BLOCK *pECB, int iFormId, int iTermId, int
iSyncId);
void SubmitCmd(EXTENSION_CONTROL_BLOCK *pECB, int iFormId, int iTermId,
int iSyncId);
void BeginCmd(EXTENSION_CONTROL_BLOCK *pECB, int iFormId, int iTermId, int
iSyncId);
void ProcessCmd(EXTENSION_CONTROL_BLOCK *pECB, int iFormId, int iTermId,
int iSyncId);
void ClearCmd(EXTENSION_CONTROL_BLOCK *pECB, int iFormId, int iTermId, int
iSyncId);
void MenuCmd(EXTENSION_CONTROL_BLOCK *pECB, int iFormId, int iTermId, int
iSyncId);

void NumberOfConnectionsCmd(EXTENSION_CONTROL_BLOCK *pECB, int iFormId,
int iTermId, int iSyncId);
static void WriteZString(EXTENSION_CONTROL_BLOCK *pECB, char *szStr);
static void h_printf(EXTENSION_CONTROL_BLOCK *pECB, char *format, ...);
void ErrorMessage(EXTENSION_CONTROL_BLOCK *pECB, int iError, int
iErrorType, char *szMsg, int iTermId, int iSyncId);
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 iSyncId, char
*szServer, char *szUser, char *szPassword, char *szDatabase);
static BOOL Close(EXTENSION_CONTROL_BLOCK *pECB, int iTermId, int
iSyncId);
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);
static BOOL SQLCloseConnection(EXTENSION_CONTROL_BLOCK *pECB, DBPROCESS
*dbproc);
static BOOL SQLStockLevel(EXTENSION_CONTROL_BLOCK *pECB, int iTermId, int
iSyncId, DBPROCESS *dbproc, STOCK_LEVEL_DATA *pStockLevel, short
deadlock_retry);
static int SQLNewOrder(EXTENSION_CONTROL_BLOCK *pECB, int iTermId,
int iSyncId, DBPROCESS *dbproc, NEW_ORDER_DATA *pNewOrder, short
deadlock_retry);
static int SQLPayment(EXTENSION_CONTROL_BLOCK *pECB, int iTermId, int
iSyncId, DBPROCESS *dbproc, PAYMENT_DATA *pPayment, short deadlock_retry);
static int SQLOrderStatus(EXTENSION_CONTROL_BLOCK *pECB, int iTermId, int
iSyncId, DBPROCESS *dbproc, ORDER_STATUS_DATA *pOrderStatus, short
deadlock_retry);
BOOL SQLDetectDeadlock(DBPROCESS *dbproc);
static void FormatString(char *szDest, char *szPic, char *szSrc);
static char *MakeStockLevelForm(int iTermId, int iSyncId, BOOL bInput);
static char *MakeMainMenuForm(int iTermId, int iSyncId);
static char *MakeWelcomeForm(void);
static char *MakeNewOrderForm(int iTermId, int iSyncId, BOOL bInput, BOOL
bValid);
static char *MakePaymentForm(int iTermId, int iSyncId, BOOL bInput);
static char *MakeOrderStatusForm(int iTermId, int iSyncId, BOOL bInput);
static char *MakeDeliveryForm(int iTermId, int iSyncId, BOOL bInput, BOOL
bSuccess);
static void UtilStrCpy(char *pDest, char *pSrc, int n);
static void ProcessNewOrderForm(EXTENSION_CONTROL_BLOCK *pECB, int
iTermId, int iSyncId);
static void ProcessPaymentForm(EXTENSION_CONTROL_BLOCK *pECB, int iTermId,
int iSyncId);
static void ProcessOrderStatusForm(EXTENSION_CONTROL_BLOCK *pECB, int
iTermId, int iSyncId);
static void ProcessDeliveryForm(EXTENSION_CONTROL_BLOCK *pECB, int
iTermId, int iSyncId);
static void ProcessStockLevelForm(EXTENSION_CONTROL_BLOCK *pECB, int
iTermId, int iSyncId);
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);

#ifdef USE_ODBC
void dbsetuserdata(PDBPROCESS dbproc, void *uPtr);
void *dbgetuserdata(PDBPROCESS dbproc);
void BindParameter(PDBPROCESS dbproc, UWORD ipar, SWORD fCType,
SWORD fSqlType, UWORD 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 ReopenConnection(PDBPROCESS dbproc);
#endif

```

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
#ifdef TIMESTAMP_STRUCT
#include <sqltypes.h>
#endif
#else
#ifdef _INC_SQLFRONT
#include <sqlfront.h>
#endif
#endif

#ifdef DBINT
typedef long DBINT;
#endif

#define DEFCLPACKSIZE 2000
#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[I_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[LAST_NAME_LEN+1];
char c_credit[CREDIT_LEN+1];
double c_discount;
double w_tax;
double d_tax;
long o_id;
short o_commit_flag;
} OL_ORDER_DATA;

#ifdef USE_ODBC

```

```

        TIMESTAMP_STRUCT      o_entry_d;
#else
        DBDATERECC           o_entry_d;
#endif
        short                o_all_local;
        double               total_amount;
        long                 num_deadlocks;
        char                 execution_status[STATUS_LEN];
    } 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
        DBDATERECC           h_date;
#endif
        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];
        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];
        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];
#ifdef USE_ODBC
        TIMESTAMP_STRUCT      c_since;
#else
        DBDATERECC           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;
        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
        DBDATERECC           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
        DBDATERECC           o_entry_d;
#endif
        short                o_carrier_id;
        OL_ORDER_STATUS_DATA OlOrderStatusData[MAX_OL_ORDER_STATUS_ITEMS];
        short                o_ol_cnt;
        long                 num_deadlocks;
        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;
        char                 DelItems[10];
        char                 execution_status[STATUS_LEN];
    } DELIVERY_DATA;

typedef struct
{
        short                w_id;
        short                d_id;
        short                thresh_hold;
        long                 low_stock;
        long                 num_deadlocks;
        char                 execution_status[STATUS_LEN];
    } STOCK_LEVEL_DATA;

#endif

```

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 occured 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 milliseconds
    int           w_id;
//delilog report line warehouse id
for delivery
    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 = 4;
//delirpt version
int           versionMM = 0;
int           versionLS = 0;
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 void   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 command line
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 iTotalsResponse;
    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 ( !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      iTotalBuckets;
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;
    }

    iTotalBuckets = iMaxSeconds + 1;

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

    iBucketSize = iTotalBuckets * 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, iTotl = 0.0; iTotl < i90thPercent; iTotl +=
(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 ( !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
withing 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;
    }
    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
*
*/

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 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;
    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.%2.2d.%3.3d
*\n", versionMS, versionMM, versionLS);
printf(" *\n");
printf("*****\n\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
only 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;
}

DELISRV.C

/* FILE: DELISRV.C
*
* Microsoft TPC-C Kit Ver. 3.00.000
* Audited 08/23/96, By Francois Raab
*
* Copyright Microsoft, 1996
*
* PURPOSE: Delivery TPC-C transaction executable
* Author: Philip Durr
* philipdu@Microsoft.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>
#include <conio.h>
#include <ctype.h>

#ifdef USE_ODBC
#include <sql.h>
#include <sqlext.h>
HENV henv;
#else
#define DBNTWIN32
#include <sqlfront.h>
#include <sqldb.h>
#endif

#include "delisrv.h"

```

```

char          szServer[32];
char //SQL server name
char          szDatabase[32];
char //tpcc database name
char          szUser[32];
char //user name
char          szPassword[32];
char //user password
int           iNumThreads = 4;
char //number of threads to create
int           iDelayMs = 1000;
char //delay between delivery queue checks
int           iDeadlockRetry = 3;
char //number of read check retries.
int           iQSlotts = 3000;
char //delivery transaction queues
int           iConnectDelay = 500;
char //delay between re-connect attempts if sql server refuses
connection.

FILE          *fpLog;
char //pointer to log file
CRITICAL_SECTION WriteLogCriticalSection; //critical
section for delivery write log
CRITICAL_SECTION DeliveryCriticalSection; //critical
section for deliveries cache
static LPTSTR lpszPipeName = TEXT("\\\\.\\pipe\\DELISRV");
char //delivery pipe name

HANDLE        hPipe = INVALID_HANDLE_VALUE; //delivery pipe handle
HANDLE        hComPort = INVALID_HANDLE_VALUE; //delivery pipe completion port handle.

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

LPDELIVERY_PACKET pDeliveryCache;

int           versionMS = 4;
char //delivery executable version number.
int           versionMM = 0;
char //formatted as MS.MM.LS, 1.00.005
int           versionLS = 0;

/* 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 command line
argument pointers
 *
 * RETURNS: None
 *
 * COMMENTS: None

```

```

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

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

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

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

    Restore();

    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: int RunDelivery(void)
*
* PURPOSE: This function executes the main delivery executable loop.
*
* ARGUMENTS: None
*
* RETURNS: int ERR_CANNOT_OPEN_PIPE cannot open
named pipe
ERR_CANNOT_CREATE_THREAD cannot
create required threads
ERR_SUCCESS
*
* successfull no error
*
* COMMENTS: None
*/

static int RunDelivery(void)
{
SECURITY_ATTRIBUTES sa;
int i;

cls();

PrintHeader();

printf("\n<Starting Delivery Service with %d Threads.>\n",
iNumThreads);
printf("\nPress <Ctrl>C to exit.\n");

bDone = FALSE;
_beginthread( CheckKey, 0, NULL );

printf("\nWaiting for delivery pipe: ");

while( !bDone )
{
AnimateWait1();
if ( WaitNamedPipe(lpszPipeName, NMPWAIT_USE_DEFAULT_WAIT)

{
sa.nLength = sizeof(sa);
sa.lpSecurityDescriptor = NULL;
sa.bInheritHandle = TRUE;

hPipe = CreateFile(lpszPipeName, GENERIC_READ |
GENERIC_WRITE, FILE_SHARE_READ | FILE_SHARE_WRITE, NULL, OPEN_EXISTING,
FILE_FLAG_OVERLAPPED, NULL);
if ( hPipe == INVALID_HANDLE_VALUE )

```

```

return ERR_CANNOT_OPEN_PIPE;
hComPort = CreateIoCompletionPort(hPipe, NULL, 0,
256);
break;
}
Sleep(100);
}

if ( !bDone )
{
if ( _beginthread( DeliveryHandler, 0, NULL ) == -1 )
return ERR_CANNOT_CREATE_THREAD;

for(i=0; i<iNumThreads; i++)
{
if ( _beginthread( DeliveryThread, 0, NULL ) == -1
return ERR_CANNOT_CREATE_THREAD;
}

printf(" \nRunning : ");

while( !bDone )
AnimateWait();
}

return ERR_SUCCESS;
}

/* FUNCTION: void AnimateWait1(void)
*
* PURPOSE: This function provides a visual indicator that the delivery
executable is waiting for
the delivery pipe to appear.
*
* ARGUMENTS: None
*
* RETURNS: None
*
* COMMENTS: None
*/

static void AnimateWait1(void)
{
const static char szStr[] = "+-|*";
static char *ptr = (char *)szStr;

printf("%c\x8", *ptr);
ptr = (*(ptr+1)) ? ptr + 1 : (char *)szStr;
Sleep(100);

return;
}

/* FUNCTION: void AnimateWait(void)
*
* PURPOSE: This function provides a visual indicator that the delivery
executable is waiting for
and processing transactions.
*

```

```

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

static void AnimateWait(void)
{
    const static char szStr[] = "/-\\|/\\|";
    static char *ptr = (char *)szStr;

    printf("%c\\x8", *ptr);
    ptr = (*(ptr+1)) ? ptr + 1 : (char *)szStr;
    Sleep(100);

    return;
}

/* FUNCTION: int Init(void)
*
* PURPOSE:   This function prepares the delivery executable for
processing.
*
* ARGUMENTS: None
*
* RETURNS:   int      iError      Error code if unsuccessfull
              ERR_SUCCESS  No error successfull
code
*
* COMMENTS:  None
*
*/

static int Init(void)
{
    int      iError;

    InitializeCriticalSection(&WriteLogCriticalSection);
    InitializeCriticalSection(&DeliveryCriticalSection);

    fpLog    = NULL;

    if ( !(pDeliveryCache = malloc(sizeof(DELIVERY_PACKET) *
iQSlots)) )
        return ERR_INSUFFICIENT_MEMORY;

    memset(pDeliveryCache, 0, sizeof(DELIVERY_PACKET) * iQSlots);

    if ( (iError = ReadRegistrySettings()) )
        return iError;

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

    //initialize db library for use
#ifdef USE_ODBC
    if ( SQLAllocEnv(&henv) == SQL_ERROR )

```

```

        return ERR_ODBC_SQLALLOCENV;
#else
    dbinit();

    // install Db Library error and message handlers
    dbmsghandle( (DBMSGHANDLE_PROC)msg_handler);
    dberrhandle( (DBERRHANDLE_PROC)err_handler);
#endif

    return ERR_SUCCESS;
}

/* FUNCTION: void Restore(void)
*
* PURPOSE:   This function cleans up allocated objects to allow for
termination of the
              delivery executable.
*
* ARGUMENTS: None
*
* RETURNS:   None
*
* COMMENTS:  None
*
*/

static void Restore(void)
{
    int      iret, l, d;

    DeleteCriticalSection(&WriteLogCriticalSection);
    DeleteCriticalSection(&DeliveryCriticalSection);

    l = 1;
    iret = WriteFile(hPipe, &l, 1, &d, NULL);

    if ( hPipe != INVALID_HANDLE_VALUE )
        iret = CloseHandle(hPipe);

    if ( fpLog )
        fclose(fpLog);

    fpLog = NULL;

#ifdef USE_ODBC
    SQLFreeEnv(henv);
#else
    dbexit();
#endif

    return;
}

/* 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_CREATE_THREAD,
            "Cannot create thread."
        },
        {
            ERR_DBGETDATA_FAILED,
            "Get data failed."
        },
        {
            ERR_REGISTRY_NOT_SETUP,
            "Registry not setup for tpcc."
        },
        {
            ERR_CANNOT_ACCESS_DELIVERY_FN,
            "Cannot access ReadDelivery cache."
        },
        {
            ERR_CANNOT_ACCESS_REGISTRY,
            "Cannot access registry key TPCC."
        },
        {
            ERR_CANNOT_CREATE_RESULTS_FILE,
            "Cannot create results file."
        },
        {
            ERR_CANNOT_OPEN_PIPE,
            "Cannot open delivery pipe."
        },
        {
            ERR_READ_PIPE,
            "Reading Delivery Pipe."
        },
        {
            ERR_INSUFFICIENT_MEMORY,
            "Insufficient memory."
        },
        {
            ERR_ODBC_SQLALLOCENV,
            "Cannot allocated ODBC env handle."
        },
        {
            ERR_SQL_ATTR_ODBC_VERSION,
            "Cannot set ODBC version."
        },
        {
            ERR_SQL_ATTR_CONNECTION_POOLING,
            "Cannot set Connection Pooling."
        },
        {
            0,
            ""
        }
    };

    for(i=0; errorMsgs[i].szMsg[0]; i++)
    {
        if ( iError == errorMsgs[i].iError )
        {
            printf("\nError(%d): %s", iError,
                errorMsgs[i].szMsg);
            if ( fpLog )
            {
                EnterCriticalSection(&WriteLogCriticalSection);
                fprintf(fpLog, "*Error(%d): %s\r\n", iError,
                    errorMsgs[i].szMsg);
            }
        }
    }
}

```

```

        if ( bFlush )
            fflush(fpLog);

        LeaveCriticalSection(&WriteLogCriticalSection);
    }
    return;
}

printf("Error(%d): Unknown Error.");
EnterCriticalSection(&WriteLogCriticalSection);
fprintf(fpLog, "*Error(%d): Unknown Error.\r\n", iError);
if ( bFlush )
    fflush(fpLog);
LeaveCriticalSection(&WriteLogCriticalSection);

return;
}

/* 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 'D':
                case 'd':

```

```

                strcpy(szDatabase, argv[i]+2);
                break;
        case 'U':
        case 'u':
                strcpy(szUser, argv[i]+2);
                break;
        case 'P':
        case 'p':
                strcpy(szPassword, argv[i]+2);
                break;
        case 'F':
        case 'f':
                bFlush = TRUE; //turn on delilog
                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("DELISRV:\n\n");
    printf("Parameter
Default\n");
    printf("-----\n");
    printf("-S Server
\n");
    printf("-D Database
tpcc \n");
    printf("-U Username
sa \n");
    printf("-P Password
\n");
    printf("-F Flush output to delilog file when written.
OFF \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 executable's banner
information.
 *
 * ARGUMENTS: None
 *
 * RETURNS: None
 *
 * COMMENTS: None
 */

static void PrintHeader(void)
{
    printf("*****\n");
    printf(" * \n");
#ifdef USE_ODBC
    printf(" * Microsoft SQL Server 6.5 (ODBC) * \n");
#else
    printf(" * Microsoft SQL Server 6.5 (DBLIB) * \n");
#endif
    printf(" * \n");
    printf(" * HTML TPC-C BENCHMARK KIT: Delivery Server * \n");
    printf(" * Version %d.%2d.%3d
*\n", versionMS, versionMM, versionLS);
    printf(" * \n");
    printf("*****\n\n");

    return;
}

/* FUNCTION: int ReadRegistrySettings(void)
 *
 * PURPOSE: This function reads the system registry filling in required
key parameters.
 *
 * ARGUMENTS: None
 *
 * RETURNS: int ERR_REGISTRY_NOT_SETUP registry not
setup tpcc.exe needs to be run
 *
 * ERR_SUCCESS
Registry read Successfull, no error
 *
 * COMMENTS: None
 */

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

    if ( RegOpenKeyEx(HKEY_LOCAL_MACHINE, "SOFTWARE\\Microsoft\\TPCC",
0, KEY_READ, &hKey) != ERROR_SUCCESS )
        return ERR_REGISTRY_NOT_SETUP;

    size = sizeof(szTmp);

```

```

        iNumThreads = 4;
        if ( RegQueryValueEx(hKey, "NumberOfDeliveryThreads", 0, &type,
szTmp, &size) == ERROR_SUCCESS )
            iNumThreads = atoi(szTmp);
            if ( !iNumThreads )
                iNumThreads = 4;

        iDelayMs = 1000;
        if ( RegQueryValueEx(hKey, "BackoffDelay", 0, &type, szTmp, &size)
== ERROR_SUCCESS )
            iDelayMs = atoi(szTmp);
            if ( !iDelayMs )
                iDelayMs = 1000;

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

        RegCloseKey(hKey);

        return ERR_SUCCESS;
    }

/* FUNCTION: void CheckKey(void *ptr)
 *
 * PURPOSE: This function checks for a key press on the delivery
executable's console. If the
 *           key press is a Ctrl C then the execution
termination flag variable bDone is set to
 *           TRUE which will start the termination of the
delivery executable.
 *
 * ARGUMENTS: void *ptr dummy argument passed in though thread
manager, unused NULL.
 *
 * RETURNS: None
 *
 * COMMENTS: None
 */

static void CheckKey(void *ptr)
{
    while( _getch() != CTRL_C)
        ;
    bDone = TRUE;

    return;
}

/* FUNCTION: void DeliveryHandler( void *ptr )
 *
 * PURPOSE: This function is executed in it's own thread what it does
is to check for delivery
 *           postings in the delivery named pipe. If any are
present then it pulls them off and
 *           places them in the next available delivery queue
array element.

```

```

 *
 * ARGUMENTS: void *ptr dummy argument passed in though thread
manager, unused NULL.
 *
 * RETURNS: None
 *
 * COMMENTS: None
 */
*/

static void DeliveryHandler( void *ptr )
{
    int i;
    int size;
    int iError;

    while( !bDone )
    {
        for(i=0; i<iQSlots; i++)
        {
            if ( !pDeliveryCache[i].bInUse )
                break;
        }
        if ( i < iQSlots )
        {
            EnterCriticalSection(&DeliveryCriticalSection);
            pDeliveryCache[i].bInUse = TRUE;
            LeaveCriticalSection(&DeliveryCriticalSection);
        }
        else
        {
            EnterCriticalSection(&DeliveryCriticalSection);
            if ( !(pDeliveryCache =
(LPDELIVERY_PACKET)realloc(pDeliveryCache, sizeof(DELIVERY_PACKET) *
(iQSlots+512))) )
                {
                    ErrorMessage(ERR_INSUFFICIENT_MEMORY);

                    LeaveCriticalSection(&DeliveryCriticalSection);
                    return;
                }
            for(i=iQSlots; i<iQSlots+512; i++)
                pDeliveryCache[i].bInUse = FALSE;
            i = iQSlots;
            pDeliveryCache[i].bInUse = TRUE;
            LeaveCriticalSection(&DeliveryCriticalSection);
        }

        pDeliveryCache[i].ov.Offset = i;
        pDeliveryCache[i].ov.Internal = 0;
        pDeliveryCache[i].ov.InternalHigh = 0;
        pDeliveryCache[i].ov.OffsetHigh = 1;
        pDeliveryCache[i].ov.hEvent = NULL;

        while( !bDone )
        {
            if ( ReadFile(hPipe, &pDeliveryCache[i].trans,
sizeof(DELIVERY_TRANSACTION), &size, &pDeliveryCache[i].ov) )
                break;
            if ( bDone )
                break;
        }
    }
}

```

```

        iError = GetLastError();
        if ( iError == ERROR_IO_PENDING )
        {
            while( pDeliveryCache[i].ov.OffsetHigh )
                Sleep(10);
            break;
        }
        else
        {
            ErrorMessage(ERR_READ_PIPE);
            return;
        }
    }
    Sleep(1);
}

return;
}

/* FUNCTION: void DeliveryThread( void *ptr )
 *
 * PURPOSE: This function is executed inside the delivery threads. The
 * queue array is continuously check and if any array elements are
 * in use then the array entry is read, cleared and this function
 * processes it.
 *
 * ARGUMENTS: void *ptr dummy argument passed in though thread
 * manager, unused NULL.
 *
 * RETURNS: None
 *
 * COMMENTS: The registry key HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\TPCC
 * value NumberOfDeliveryThreads controls how
 * many of these functions are running. The
 * HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\TPCC
 * value BackoffDelay controls the amount of
 * time this function waits
 * between checks of the delivery queue.
 */

static void DeliveryThread( void *ptr )
{
    int size;
    int key;
    LPOVERLAPPED pov;
    DELIVERY delivery;
    int iError;

    if ( SQLOpenConnection(&delivery.dbproc, szServer, szDatabase,
szUser, szPassword, &delivery.spid) )
        return; //error posting tbd

    //while delisrv running i.e. user has not requested termination
    while( !bDone )
    {
        if ( GetQueuedCompletionStatus(hComPort, &size, &key, &pov,
(DWORD)-1) )

```

```

        {
            pov->OffsetHigh = 0; //clear to notify delivery
            handler ok to read another entry.
            //some delivery to do so process it
            memcpy(&delivery.queue, &pDeliveryCache[pov-
>Offset].trans.queue, sizeof(SYSTEMTIME));
            delivery.w_id = pDeliveryCache[pov-
>Offset].trans.w_id;
            delivery.o_carrier_id = pDeliveryCache[pov-
>Offset].trans.o_carrier_id;

            if ( (iError=SQLDelivery(&delivery)) )
            {
                ErrorMessage(iError);
                printf("Running : ");
                continue;
            }

            //update log
            WriteLog(&delivery);

            EnterCriticalSection(&DeliveryCriticalSection);
            pDeliveryCache[pov->Offset].bInUse = FALSE;
            LeaveCriticalSection(&DeliveryCriticalSection);
        }
    }

    return;
}

/* FUNCTION: static 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
 */

#ifdef USE_ODBC
static int err_handler(DBPROCESS *dbproc, int severity, int dberr, int
oserr, char *dberrstr, char *oserrstr)
{
    if (oserr != DBNOERR)
        printf("(%d) %s", oserr, oserrstr);
}

```

```

    if ((dbproc == NULL) || (DBDEAD(dbproc)))
        ExitThread((unsigned long)-1);

    return INT_CONTINUE;
}
#endif

/* FUNCTION: static 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.
*/
static int msg_handler(DBPROCESS *dbproc, DBINT msgno, int msgstate, int
severity, char *msgtext)
{
    if ( ( msgno == 5701) || (msgno == 2528) || (msgno == 5703) || (msgno
== 6006) )
        return INT_CONTINUE;

    // deadlock message
    if (msgno == 1205)
    {
        // set the deadlock indicator
        if (dbgetuserdata(dbproc) != NULL)
            *((BOOL *) dbgetuserdata(dbproc)) = TRUE;
        else
            printf("\nError, dbgetuserdata returned NULL.\n");

        return INT_CONTINUE;
    }

    if (msgno == 0)
        return INT_CONTINUE;
    else
        printf("SQL Server Message (%ld) : %s\n", msgno, msgtext);
    return INT_CANCEL;
}

```

```

/* FUNCTION: BOOL SQLOpenConnection(DBPROCESS **dbproc, char *server, char
*database, char *user, char *password, int *spid)
*
* PURPOSE: This function opens the sql connection for use.
*
* ARGUMENTS: DBPROCESS **dbproc pointer to returned
DBPROCESS
*
* char *server SQL
server name
*
* char *database SQL
server database
*
* char *user user
name
*
* char *password user
password
*
* int *spid
*
* pointer to returned spid
*
* RETURNS: BOOL FALSE if successful
*
* TRUE if an error occurs
*
* COMMENTS: None
*/

#ifdef USE_ODBC
    static BOOL SQLOpenConnection(DBPROCESS **dbproc, char *server,
char *database, char *user, char *password, int *spid)
    {

        RETCODE rc;
        char buffer[30];

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

        //set pECB data into dbproc
        dbsetuserdata(*dbproc, malloc(sizeof(BOOL)));
        *((BOOL *)dbgetuserdata(*dbproc)) = FALSE;

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

        if ( SQLSetConnectOption((*dbproc)->hdbc, SQL_PACKET_SIZE,
4096) == 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;

        strcpy(buffer, "use tpcc");

        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(DBPROCESS **dbproc, char *server,
char *database, char *user, char *password, int *spid)
{
    LOGINREC *login;

    login = dblogin();
    DBSETLUSER(login, user);
    DBSETLPWD(login, password);

    DBSETLPACKET(login, (USHORT)DEFCLPACKSIZE);

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

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

    dbsetuserdata(*dbproc, malloc(sizeof(BOOL)));
    *((BOOL *)dbgetuserdata(*dbproc)) = FALSE;

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

    dbsqlxec(*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");

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

```

```

        return FALSE;
    }
#endif
//queue time, end time, elapsed time, w_id, o_carrier_id, o_id1, ...
o_id10
/* FUNCTION: void WriteLog(LPDELIVERY pDelivery)
*
* PURPOSE:      This function writes the delivery results to the delivery
log file.
*
* ARGUMENTS:   LPDELIVERY      pDelivery      Pointer to delivery
information.
*
* RETURNS:     None
*
* COMMENTS:    None
*/

static void WriteLog(LPDELIVERY pDelivery)
{
    int elapsed;

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

    EnterCriticalSection(&WriteLogCriticalSection);

    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\r\n",
pDelivery->trans_end.wYear - 1900, pDelivery-
>trans_end.wMonth, pDelivery->trans_end.wDay,
pDelivery->queue.wHour, pDelivery->queue.wMinute,
pDelivery->queue.wSecond, pDelivery->queue.wMilliseconds,
pDelivery->trans_end.wHour, pDelivery->trans_end.wMinute,
pDelivery->trans_end.wSecond, pDelivery->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);

    LeaveCriticalSection(&WriteLogCriticalSection);

    return;
}

/* FUNCTION: void CalculateElapsedTime(int *pElapsed, LPSYSTEMTIME
lpBegin, LPSYSTEMTIME lpEnd)
*
* PURPOSE:      This function calculates the elapsed time a delivery
transaction took.
*

```

```

* ARGUMENTS: int *pElapsed pointer 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 hour period
however it will not work over 48 hours.
    if ( *pElapsed < 0 )
        *pElapsed = *pElapsed + (24 * 60 * 60 * 1000);

    return;
}

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

#ifdef USE_ODBC
static int SQLDelivery(DELIVERY *pDelivery)
{
    int i;
    int deadlock_count;

```

```

SDWORD iLength[10];
BOOL bDeadlock;

deadlock_count = 0;

// Start new delivery
while ( TRUE )
{
    BindParameter(pDelivery->dbproc, 1, SQL_C_SSHORT,
SQL_SMALLINT, 0, 0, &pDelivery->w_id, 0);
    BindParameter(pDelivery->dbproc, 2, SQL_C_SSHORT,
SQL_SMALLINT, 0, 0, &pDelivery->o_carrier_id, 0);

    if ( ExecuteStatement(pDelivery->dbproc, "{call
tpcc_delivery (?, ?)}") )
        return 1;
    bDeadlock = *((BOOL *)dbgetuserdata(pDelivery-
>dbproc));

    if ( !bDeadlock )
    {
        for (i=0;i<10;i++)
        {
            if ( BindColumn(pDelivery->dbproc,
(UWORD)(i+1), SQL_C_SLONG, &pDelivery->o_id[i], 0, &iLength[i]) )
                return 1;
        }
        if ( GetResults(pDelivery->dbproc) )
            return 1;
        for(i=0; i<10; i++)
        {
            if ( iLength[i] <= 0 )
                pDelivery->o_id[i] = 0;
        }
    }
    SQLFreeStmt(pDelivery->dbproc->hstmt, SQL_CLOSE);
    if ( !SQLDetectDeadlock(pDelivery->dbproc) )
        break;
    deadlock_count++;
    Sleep(10 * deadlock_count);
}
GetLocalTime(&pDelivery->trans_end);

return ERR_SUCCESS;
}

#else
static int SQLDelivery(DELIVERY *pDelivery)
{
    RETCODE rc;
    int i;
    int deadlock_count;
    BYTE *pData;

    deadlock_count = 0;

    // Start new delivery
    while ( TRUE )
    {
        if (dbrpcinit(pDelivery->dbproc, "tpcc_delivery",
0) == SUCCEED)
        {

```

```

        dbrpcparam(pDelivery->dbproc, NULL, 0,
SQLINT2, -1, -1, (BYTE *)&pDelivery->w_id);
        dbrpcparam(pDelivery->dbproc, NULL, 0,
SQLINT1, -1, -1, (BYTE *) &pDelivery->o_carrier_id);

        if (dbrpcexec(pDelivery->dbproc) == SUCCEED)
        {
            while ((rc = dbresults(pDelivery-
>dbproc)) != NO_MORE_RESULTS) && (rc != FAIL))
            {
                while ((rc =
dbnextrow(pDelivery->dbproc)) != NO_MORE_ROWS) && (rc != FAIL))
                {
                    for (i=0;i<10;i++)
                    {
                        if(pData=dbdata(pDelivery->dbproc, i+1))
                        pDelivery->o_id[i] = *((DBINT *)pData);
                        else
                        pDelivery->o_id[i] = 0;
                    }
                }
            }
            if ( !SQLDetectDeadlock(pDelivery->dbproc) )
                break;
            deadlock_count++;
            Sleep(10 * deadlock_count);
        }
        GetLocalTime (&pDelivery->trans_end);
        return ERR_SUCCESS;
    }
#endif

/* 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
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;
}
#endif USE_ODBC
/* FUNCTION: void dbsetuserdata(PDBPROCESS dbproc, void *uPtr)
 *
 * PURPOSE: This function sets a user pointer in a dbproc
structure
 *
 * This functionality is not provided in odbc
so this function
 *
 * provides it.
 *
 * ARGUMENTS: DBRPOCESS dbproc ODBC dbprocess structure
 *
 * void *uPtr returned data
user pointer
 *
 * RETURNS: none
 *
 * COMMENTS: The caller is responsible for the contents of the
uPtr.
 *
 */
void dbsetuserdata(PDBPROCESS dbproc, void *uPtr)
{
    dbproc->uPtr = uPtr;
}
/* FUNCTION: void dbsetuserdata(PDBPROCESS dbproc, void *uPtr)
 *
 * PURPOSE: This function returns the user pointer stored in a
dbproc structure
 *
 * This functionality is not provided in odbc
so this function
 *
 * provides it.
 *
 * ARGUMENTS: DBRPOCESS dbproc ODBC dbprocess structure
 *
 * RETURNS: none
 *
 * COMMENTS: The returned pointer is placed in the dbproc
structure by the dbsetuserdata() API.
 *
 */
void *dbgetuserdata(PDBPROCESS dbproc)
{
    return dbproc->uPtr;
}

```

```

/* FUNCTION: void BindParameter(PDBPROCESS dbproc, UWORD ipar,
SWORD fCType, SWORD fSqlType, UDWORD cbColDef, SWORD ibScale, PTR
rgbValue, SDWORD cbValueMax)
 *
 * PURPOSE: This function wraps the functionality provided by
the SQLBindParameter
 *
 * allowing error process so that each bind
call does not need to provide
 *
 * error and message checking.
 *
 * ARGUMENTS: PDBPROCESS dbproc pointer to odbc
dbprocess structure
 *
 * UWORD ipar Parameter
number, ordered sequentially left to right, starting at 1.
 *
 * SWORD fParamType The type of
the parameter.
 *
 * SWORD fCType The C data
type of the parameter.
 *
 * SWORD fSqlType The SQL data
type of the parameter.
 *
 * UDWORD cbColDef The precision
of the column or expression
 *
 * of the corresponding parameter marker.
 *
 * SWORD ibScale The scale of
the column or expression of the corresponding
 *
 * parameter marker.
 *
 * PTR rgbValue A
pointer to a buffer for the parameter's data.
 *
 * SDWORD cbValueMax Maximum length
of the rgbValue buffer.
 *
 * void *uPtr returned data
user pointer
 *
 * RETURNS: none
 *
 * COMMENTS: The returned pointer is placed in the dbproc
structure by the dbset
 *
 */
void BindParameter(PDBPROCESS dbproc, UWORD ipar, SWORD fCType,
SWORD fSqlType, UDWORD cbColDef, SWORD ibScale, PTR rgbValue, SDWORD
cbValueMax)
{
    RETCODE rc;
    rc = SQLBindParameter(dbproc->hstmt, ipar, SQL_PARAM_INPUT,
fCType, fSqlType, cbColDef, ibScale, rgbValue, cbValueMax, NULL);
    if (rc == SQL_ERROR)
        ODBCError(dbproc);
    return;
}
/* FUNCTION: void ODBCError(PDBPROCESS dbproc)
 *
 * PURPOSE: This function wraps the odbc error call so that the
dblib msg_handler is called.
 *
 * This allows the deadlock flag in the dbproc
user data structure pEcbInfo in

```

```

*          dbproc to be set if necessary.
*
* ARGUMENTS: DBRPOCESS      dbproc ODBC dbprocess structure
*
* RETURNS:      none
*
* COMMENTS:    none
*
*/

void ODBCError(PDBPROCESS dbproc)
{
    SDWORD      lNativeError;
    char        szState[6];
    char        szMsg[SQL_MAX_MESSAGE_LENGTH];

    while( SQLError(henv, dbproc->hdbc, dbproc->hstmt, szState,
&lNativeError, szMsg, sizeof(szMsg), NULL) == SQL_SUCCESS )
    {
        msg_handler(dbproc, lNativeError, 0, 0, szMsg);
        if ( !lNativeError )
        {
            printf("\nODBC Error State = %s, %s\n",
szState, szMsg);
            printf("Running : ");
        }
    }

    return;
}

/* FUNCTION: BOOL ExecuteStatement(PDBPROCESS dbproc, szStatement)
*
* PURPOSE:      This function wraps the odbc SQLExecDirect API so
that error handling and
*
*               and deadlock are taken care of in a common
location.
*
* ARGUMENTS:    DBRPOCESS      dbproc ODBC dbprocess structure
*               char          *szStatement sql
stored procedure statement to be executed.
*
* RETURNS:      none
*
* COMMENTS:    none
*
*/

BOOL ExecuteStatement(PDBPROCESS dbproc, char *szStatement)
{
    RETCODE      rc;

    rc = SQLExecDirect(dbproc->hstmt, szStatement, SQL_NTS);
    if (rc != SQL_SUCCESS && rc != SQL_SUCCESS_WITH_INFO)
    {
        ODBCError(dbproc);
        if ( *((BOOL *)dbgetuserdata(dbproc)) )
            return FALSE;
        return TRUE;
    }
    return FALSE;
}

```

```

}

/* FUNCTION: BOOL BindColumn(PDBPROCESS dbproc, SQLUSMALLINT icol,
SQLSMALLINT fctype, SQLPOINTER rgbValue, SQLINTEGER cbValueMax, SDWORD FAR
*piLength)
*
* PURPOSE:      This function wraps the odbc SQLBindCol API so that
error handling and
*
*               and deadlock are taken care of in a common
location.
*
* ARGUMENTS:    DBRPOCESS      dbproc ODBC dbprocess structure
*               UWORD icol      Column
number of result data, ordered sequentially left to right, starting at 1.
*               SWORD fctype    The C
data type of the result data. SQL_C_BINARY, SQL_C_BIT, SQL_C_BOOKMARK,
*
*               SQL_C_CHAR, SQL_C_DATE, SQL_C_DEFAULT, SQL_C_DOUBLE,
SQL_C_FLOAT, SQL_C_SLONG,
*
*               SQL_C_SSHORT, SQL_C_STINYINT, SQL_C_TIME, SQL_C_TIMESTAMP,
SQL_C_ULONG,
*
*               SQL_C_USHORT, SQL_C_UTINYINT, SQL_C_DEFAULT
*               PTR          rgbValue
the Pointer to storage for the data. If rgbValue is a null pointer,
*
*               driver unbinds the column.
*               SDWORD cbValueMax
Maximum length of the rgbValue buffer. For character data,
rgbValue
*
*               must also include space for the null-termination byte.
*               SDWORD *piLength
Pointer to variable to receive length of returned data.
* RETURNS:      none
*
* COMMENTS:    none
*
*/

BOOL BindColumn(PDBPROCESS dbproc, SQLUSMALLINT icol, SQLSMALLINT
fctype, SQLPOINTER rgbValue, SQLINTEGER cbValueMax, SDWORD *piLength)
{
    RETCODE      rc;

    rc = SQLBindCol(dbproc->hstmt, icol, fctype, rgbValue,
cbValueMax, piLength);
    if ( rc == SQL_ERROR )
    {
        ODBCError(dbproc);
        return TRUE;
    }
    return FALSE;
}

/* FUNCTION: BOOL GetResults(PDBPROCESS dbproc)
*
* PURPOSE:      This function wraps the odbc SQLFetch API so that
error handling and

```

```

*          and deadlock are taken care of in a common
location.
*
* ARGUMENTS: DBRPROCESS      dbproc ODBC dbprocess structure
*
* RETURNS:      none
*
* COMMENTS:    none
*
*/

BOOL GetResults(PDBPROCESS dbproc)
{
    if ( SQLFetch(dbproc->hstmt) == SQL_ERROR )
    {
        ODBCError(dbproc);
        if ( *((BOOL *)dbgetuserdata(dbproc)) )
            return FALSE;
        return TRUE;
    }
    return FALSE;
}

/* FUNCTION: BOOL MoreResults(DBPROCESS dbproc)
*
* PURPOSE:   This function wraps the odbc SQLMoreResults API so
that error handling and
*          and deadlock are taken care of in a common
location.
*
* ARGUMENTS: DBRPROCESS      dbproc ODBC dbprocess structure
*
* RETURNS:      none
*
* COMMENTS:    none
*
*/

BOOL MoreResults(PDBPROCESS dbproc)
{
    if ( SQLMoreResults(dbproc->hstmt) == SQL_ERROR )
    {
        ODBCError(dbproc);
        if ( *((BOOL *)dbgetuserdata(dbproc)) )
            return FALSE;
        return TRUE;
    }
    return FALSE;
}

#endif

```

TPCC.C

```

/* FILE:      TPCC.C
*
* Microsoft TPC-C Kit Ver. 3.00.000
* Audited 08/23/96      By Francois Raab
*
* Copyright Microsoft, 1996
*
*/

```

```

* PURPOSE:    Main module for TPCC.DLL which is an ISAPI service
dll.
* Author:     Philip Durr
*             philipdu@Microsoft.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>

#ifdef USE_ODBC
#include <sqltypes.h>
#include <sql.h>
#include <sqlext.h>
HENV henv;
#else
#define DBNTWIN32
#include <sqlfront.h>
#include <sqldb.h>
#endif

#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.

char  szServer[32]  = { 0 }; //global variables used with
this DLL
char  szUser[32]   = { 0 };
char  szPassword[32] = { 0 };
char  szDatabase[32] = "tpcc";
BOOL  bLog         = FALSE;
int   iThreads    = 5;
int   iMaxWareHouses = 500;
int   iQSlotts    = 3000;
int   iDelayMs    = 100;
int   iConnectDelay = 500;
short iDeadlockRetry = (short)3;
short iMaxConnections = (short)25;

#ifdef USE_ODBC
int   bConnectionPooling = FALSE;
#endif

//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 iSyncId) =
{
    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=\"FORMID\" VALUE=\"1\">"
    "<INPUT
TYPE=\"hidden\" NAME=\"TERMIN\" VALUE=\"-2\">"
    "<INPUT
TYPE=\"hidden\" NAME=\"SYNCID\" VALUE=\"0\">"
    "Warehouse ID
<INPUT NAME=\"w_id\" SIZE=4><BR>"
    "District ID
<INPUT NAME=\"d_id\" SIZE=2><BR>"
    "<HR>"
    "<INPUT
TYPE=\"submit\" NAME=\"CMD\" VALUE=\"Submit\">"
    "</FORM><BODY>"
    "</HTML>";

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 LPTSTR lpszPipeName =
TEXT("\\\\.\\pipe\\DELISRV");

```

```

static HANDLE hDeliveryWrite =
INVALID_HANDLE_VALUE;
static HANDLE hPipe =
INVALID_HANDLE_VALUE;

static EXTENSION_CONTROL_BLOCK *gpECB;
static int bTpccExit; //exit
delivery disconnect loop as dll exiting.

/* FUNCTION: BOOL APIENTRY DllMain(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 occurred
in initialization TRUE
DLL successfully initialized
*
* COMMENTS: None
*/

BOOL APIENTRY DllMain(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;

        // create a security descriptor that allows anyone
to access the pipe...
        pSD = (PSECURITY_DESCRIPTOR)malloc(
SECURITY_DESCRIPTOR_MIN_LENGTH );
        if ( pSD == NULL )
        {
            MessageBox(NULL, "Error
malloc(SECURITY_DESCRIPTOR_MIN_LENGTH)", "Init", MB_OK | MB_ICONSTOP);
            return FALSE;
        }
        if ( !InitializeSecurityDescriptor(pSD,
SECURITY_DESCRIPTOR_REVISION) )
        {
            MessageBox(NULL, "Error
InitializeSecurityDescriptor()", "Init", MB_OK | MB_ICONSTOP);
            return FALSE;
        }
        // add a NULL disc. ACL to the security descriptor.
        if ( !SetSecurityDescriptorDacl(pSD, TRUE, (PACL)
NULL, FALSE) )
        {
            MessageBox(NULL, "Error
SetSecurityDescriptorDacl().", "Init", MB_OK | MB_ICONSTOP);
            return FALSE;
        }

        sa.nLength                = sizeof(sa);
        sa.lpSecurityDescriptor    = pSD;
        sa.bInheritHandle         = TRUE;

        // open delivery named pipe...
        hPipe = CreateNamedPipe(lpszPipeName,
FILE_FLAG_OVERLAPPED | PIPE_ACCESS_DUPLEX,
PIPE_TYPE_BYTE | PIPE_READMODE_BYTE |
PIPE_NOWAIT,
1, 65535, 65535, 250, &sa);

        if ( hPipe == INVALID_HANDLE_VALUE )
        {
            MessageBox(NULL, "Error CreateNamedPipe().",
"Init", MB_OK | MB_ICONSTOP);
            free(pSD);
            return FALSE;
        }

        bTpccExit = FALSE;

        if ( _beginthread( DeliveryDisconnect, 0, NULL ) ==
-1 )
        {
            MessageBox(NULL, "Error _beginthread()",
"Init", MB_OK | MB_ICONSTOP);
            return FALSE;
        }
#ifdef USE_ODBC
        if ( SQLAllocEnv(&henv) == SQL_ERROR )

```

```

        {
            MessageBox(NULL, "Error
SQLAllocEnv()", "Init", MB_OK | MB_ICONSTOP);
            return FALSE;
        }

        #if ( ODBCVER >= 0x0300 )

            if ( bConnectionPooling )
            {
                /* added to make sure we go
into connection pooling mode */
                Beep(100,500);
                Beep(1000,500);

                if ( SQLSetEnvAttr(henv,
SQL_ATTR_ODBC_VERSION, (PTR)SQL_OV_ODBC3, SQL_INTEGER) == SQL_ERROR )
                {
                    MessageBox(NULL,
"Error SQLSetEnvAttr() SQL_ATTR_ODBC_VERSION", "Init", MB_OK |
MB_ICONSTOP);
                    return FALSE;
                }
                if ( SQLSetEnvAttr(henv,
SQL_ATTR_CONNECTION_POOLING, (PTR)SQL_CP_ONE_PER_HENV, SQL_INTEGER) ==
SQL_ERROR )
                {
                    MessageBox(NULL,
"Error SQLSetEnvAttr() SQL_ATTR_CONNECTION_POOLING", "Init", MB_OK |
MB_ICONSTOP);
                    return FALSE;
                }
            }
        }
    #endif

    #else
        dbinit();
        if ( dbgetmaxprocs() < iMaxConnections )
        {
            if ( dbsetmaxprocs(iMaxConnections)
== FAIL )
            {
                //set for fail error message
                when HttpExtensionProc() is called 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);
    #endif
    break;
case DLL_THREAD_ATTACH:
    break;
case DLL_THREAD_DETACH:
    break;
case DLL_PROCESS_DETACH:

```

```

        if ( pSD )
            free( pSD );

        bTpccExit = TRUE;

        if ( hPipe )
            DisconnectNamedPipe( hPipe );

        if ( hPipe != INVALID_HANDLE_VALUE )
            CloseHandle( hPipe );

        (*Term.Restore)();

#ifdef USE_ODBC
        SQLFreeEnv( henv );
#else
        dbexit();
#endif

        DeleteCriticalSection(&CriticalSection);
        DeleteCriticalSection(&ErrorLogCriticalSection);

        break;
    }
    return TRUE;
}

/* FUNCTION: void DeliveryDisconnect(void *ptr)
 *
 * PURPOSE: This function handles disconnecting the server side of the
 *          delivery pipe when the
 *          delivery handler application shuts down.
 *
 * ARGUMENTS: void *ptr void pointer normally NULL passed from
 *          thread handler.
 *
 * RETURNS: None
 *
 * COMMENTS: This function runs as thread which allows the client pipe
 *          to disconnect by
 *          sending a byte back though the pipe to the
 *          server i.e. this DLL.
 */

static void DeliveryDisconnect(void *ptr)
{
    int l, d;
    SECURITY_ATTRIBUTES sa;
    PSECURITY_DESCRIPTOR pSD;

    // create a security descriptor that allows anyone to access the
    pipe...
    pSD = (PSECURITY_DESCRIPTOR)malloc( SECURITY_DESCRIPTOR_MIN_LENGTH
);
    InitializeSecurityDescriptor(pSD, SECURITY_DESCRIPTOR_REVISION);
    SetSecurityDescriptorDacl(pSD, TRUE, (PACL) NULL, FALSE);
    sa.nLength = sizeof(sa);
    sa.lpSecurityDescriptor = pSD;
    sa.bInheritHandle = TRUE;

    while( !bTpccExit )

```

```

    {
        if ( hPipe && ReadFile(hPipe, &l, 1, &d, NULL) )
        {
            DisconnectNamedPipe(hPipe);
            CloseHandle(hPipe);
            // open delivery named pipe...
            hPipe = CreateNamedPipe(lpszPipeName,
FILE_FLAG_OVERLAPPED | PIPE_ACCESS_DUPLEX,
PIPE_NOWAIT,
PIPE_TYPE_BYTE | PIPE_READMODE_BYTE |
1, 65535, 65535, 250, &sa);
        }
        Sleep( 2000 ); //check for delivery application exit once
every 2 seconds.
    }

    free(pSD);

    return;
}

/* 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 service
 *          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
sent
*
* COMMENTS: None
*
*/

DWORD WINAPI HttpExtensionProc(EXTENSION_CONTROL_BLOCK *pECB)
{
    int iCmd, FormId, TermId, iSyncId;
    FILE *fp;

    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.wDay,
systemTime.wYear, systemTime.wMonth,
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 this is 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 iSyncId)
{

```



```

        WriteZString(pECB, MakeNewOrderForm(iTermId, iSyncId, TRUE,
FALSE));
        UNUSEDPARAM(iFormId);
    }
    return;
}

/* FUNCTION: void PaymentForm(EXTENSION_CONTROL_BLOCK *pECB, int iFormId,
int iTermId, int iSyncId)
*
* 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
iSyncId 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 iSyncId)
{
    WriteZString(pECB, MakePaymentForm(iTermId, iSyncId, TRUE) );
    UNUSEDPARAM(iFormId);
}

/* FUNCTION: void DeliveryForm(EXTENSION_CONTROL_BLOCK *pECB, int iFormId,
int iTermId, int iSyncId)
*
* 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
iSyncId 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 iSyncId)
{
    WriteZString(pECB, MakeDeliveryForm(iTermId, iSyncId, TRUE, TRUE)
);
    UNUSEDPARAM(iFormId);
}

/* FUNCTION: void OrderStatusForm(EXTENSION_CONTROL_BLOCK *pECB, int
iFormId, int iTermId, int iSyncId)
*
* 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
iSyncId 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 iSyncId)
{
    WriteZString(pECB, MakeOrderStatusForm(iTermId, iSyncId, TRUE) );
    UNUSEDPARAM(iFormId);
}

/* FUNCTION: void StockLevelForm(EXTENSION_CONTROL_BLOCK *pECB, int
iFormId, int iTermId, int iSyncId)
*
* 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. 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 StockLevelForm(EXTENSION_CONTROL_BLOCK *pECB, int iFormId, int
iTermId, int iSyncId)
{
    WriteZString(pECB, MakeStockLevelForm(iTermId, iSyncId, TRUE) );
    return;
}

/* FUNCTION: void Exitcmd(EXTENSION_CONTROL_BLOCK *pECB, int iFormId, int
iTermId, int iSyncId)
*
* 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. TERMID= from http
command line
*           int
iSyncId      sync id of calling browser
*           EXTENSION_CONTROL_BLOCK *pECB
*           structure pointer to passed in internet
*
* RETURNS:   None
*
* COMMENTS:  None
*
*/

void Exitcmd(EXTENSION_CONTROL_BLOCK *pECB, int iFormId, int iTermId, int
iSyncId)
{
    (*Term.Delete)(pECB, iTermId);

    WriteZString(pECB, MakeWelcomeForm() );

    UNUSEDPARAM(iFormId);
    UNUSEDPARAM(iSyncId);

    return;
}

/* FUNCTION: void SubmitCmd(EXTENSION_CONTROL_BLOCK *pECB, int iFormId,
int iTermId, int iSyncId)
*
* 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. TERMID= from http
command line
*           int
iSyncId      sync id of calling browser
*           EXTENSION_CONTROL_BLOCK *pECB
*           structure pointer to passed in internet
*
* 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. 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:   SQL server must be specified, however the user and password
parameters 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;

    if ( !GetKeyValue(pQueryString, "Server", szServer,
sizeof(szServer)) )
    {
        ErrorMessage(pECB, ERR_NO_SERVER_SPECIFIED,
ERR_TYPE_WEBDLL, NULL, iTermId, iSyncId);
        return;
    }

    if ( !GetKeyValue(pQueryString, "User", szUser, sizeof(szUser)) )
        strcpy(szUser, "sa");

    if ( !GetKeyValue(pQueryString, "Psw", szPassword,
sizeof(szPassword)) )
        strcpy(szPassword, "");

    if ( !GetKeyValue(pQueryString, "Db", szDatabase,
sizeof(szDatabase)) )
        strcpy(szDatabase, "tpcc");

    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. 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
*
*/

```

```

*
              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 ProcessCmd(EXTENSION_CONTROL_BLOCK *pECB, int iFormId, int iTermId,
int iSyncId)
{
    switch( iFormId )
    {
        case WELCOME_FORM:
            return;
        case MAIN_MENU_FORM:
            return;
        case NEW_ORDER_FORM:
            ProcessNewOrderForm(pECB, iTermId, iSyncId);
            return;
        case PAYMENT_FORM:
            ProcessPaymentForm(pECB, iTermId, iSyncId);
            return;
        case DELIVERY_FORM:
            ProcessDeliveryForm(pECB, iTermId, iSyncId);
            return;
        case ORDER_STATUS_FORM:
            ProcessOrderStatusForm(pECB, iTermId, iSyncId);
            return;
        case STOCK_LEVEL_FORM:
            ProcessStockLevelForm(pECB, iTermId, iSyncId);
            return;
    }
}

/* FUNCTION: void ClearCmd(EXTENSION_CONTROL_BLOCK *pECB, int iFormId, int
iTermId, int iSyncId)
*
* 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
iSyncId      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 with out
*           beginning at the login screen for each
client.
*/

```

```

void ClearCmd(EXTENSION_CONTROL_BLOCK *pECB, int iFormId, int iTermId, int
iSyncId)
{
    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.iMasterSyncId = 1;

    if ( Term.pClientData )
        free(Term.pClientData);
    Term.pClientData = NULL;
    Term.bInit       = FALSE;

    (*Term.Init)();
    if ( !(*Term.Allocate)() )
    {
        ErrorMessage(pECB, ERR_MAX_CONNECT_PARAM, ERR_TYPE_WEBDLL,
NULL, iTermId, iSyncId);
        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].inUse )
            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 to the
 * client browser.
 *
 * ARGUMENTS: EXTENSION_CONTROL_BLOCK *pECB passed in structure
 * pointer from inetsrv.
 * char *szStr
 * string to display in the client browser.
 *
 * RETURNS: None
 *
 * COMMENTS: This function assumes that the string to 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);

    (*pECB->ServerSupportFunction)(pECB->ConnID,
HSE_REQ_SEND_RESPONSE_HEADER, szHeader, &iSize, (LPDWORD)szHeader1);
    (*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, ...)
{
    char szBuff[512];
    char szTmp[512];

    va_list marker;
    va_start(marker, format);
    vsprintf(szTmp, format, marker);
    va_end(marker);

    wsprintf(szBuff, "<html>%s</html>", szTmp) + 1;

    WriteZString(pECB, szBuff);

    return;
}
/* 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;

    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_TERMINAL_ID,
      "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_TOO_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 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_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\">");
        wsprintf(szForm+strlen(szForm), "<INPUT
TYPE=\"hidden\" NAME=\"STATUSID\" VALUE=\"%d\">", iErrorType);
        wsprintf(szForm+strlen(szForm), "<INPUT
TYPE=\"hidden\" NAME=\"TERMINID\" VALUE=\"%d\">", iTermId);
        wsprintf(szForm+strlen(szForm), "<INPUT
TYPE=\"hidden\" NAME=\"SYNCID\" VALUE=\"%d\">", iSyncId);
        wsprintf(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\">");
        wsprintf(szForm+strlen(szForm), "<INPUT
TYPE=\"hidden\" NAME=\"STATUSID\" VALUE=\"%d\">", iErrorType);
        wsprintf(szForm+strlen(szForm), "<INPUT
TYPE=\"hidden\" NAME=\"TERMINID\" VALUE=\"%d\">", iTermId);
        wsprintf(szForm+strlen(szForm), "<INPUT
TYPE=\"hidden\" NAME=\"SYNCID\" VALUE=\"%d\">", iSyncId);
        wsprintf(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\">");
        wsprintf(szForm+strlen(szForm), "<INPUT
TYPE=\"hidden\" NAME=\"STATUSID\" VALUE=\"%d\">", iErrorType);
        wsprintf(szForm+strlen(szForm), "<INPUT
TYPE=\"hidden\" NAME=\"TERMINID\" VALUE=\"%d\">", iTermId);
        wsprintf(szForm+strlen(szForm), "<INPUT
TYPE=\"hidden\" NAME=\"SYNCID\" VALUE=\"%d\">", iSyncId);
        wsprintf(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\">");
        wsprintf(szForm+strlen(szForm), "<INPUT
TYPE=\"hidden\" NAME=\"STATUSID\" VALUE=\"%d\">", iErrorType);
        wsprintf(szForm+strlen(szForm), "<INPUT
TYPE=\"hidden\" NAME=\"TERMINID\" VALUE=\"%d\">", iTermId);
        wsprintf(szForm+strlen(szForm), "<INPUT
TYPE=\"hidden\" NAME=\"SYNCID\" VALUE=\"%d\">", iSyncId);
        wsprintf(szForm+strlen(szForm), "Error: ODBC(%d):
%s", iError, szMsg);

```

```

        strcat(szForm, "</FORM><BODY></HTML>");
        WriteZString(pECB, szForm);
        break;
    }
    return;
}
/* 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 value 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: 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
*
*/

#ifdef USE_ODBC
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)))

```

```

{
    ErrorMessage(gpECB, -1, ERR_TYPE_DBLIB, "DBPROC is
invalid.", iTermId, iSyncId);
    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 )
    return INT_CANCEL;

if ( oserr != DBNOERR )
{
    ErrorMessage(pECB, oserr, ERR_TYPE_DBLIB, oserrstr,
iTermId, iSyncId);

    if ( pEcbInfo )
        pEcbInfo->bFailed = TRUE;

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

    EnterCriticalSection(&ErrorLogCriticalSection);

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

    fprintf(fp, "%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);

    LeaveCriticalSection(&ErrorLogCriticalSection);

    fclose(fp);
}

return INT_CANCEL;
}
#endif

/* 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
*          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
            ErrorMessage(pECB, -1, ERR_TYPE_SQL, "Error,
dbgetuserdata returned NULL.", iTermId, iSyncId);
        return INT_CONTINUE;
    }
    if ( pEcbInfo && pEcbInfo->bFailed )
        return INT_CANCEL;
}

```

```

    if (msgno == 0)
        return INT_CONTINUE;
    else
    {
        ErrorMessage(pECB, msgno, ERR_TYPE_SQL, msgtext, iTermId,
iSyncId);

        if ( pEcbInfo )
            pEcbInfo->bFailed = TRUE;

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

        EnterCriticalSection(&ErrorLogCriticalSection);
        sprintf(szTmp, "Error: SQLSVR(%d): %s", msgno, msgtext);
        fprintf(fp, "%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);
        LeaveCriticalSection(&ErrorLogCriticalSection);

        fclose(fp);
    }
    return INT_CANCEL;
}

/* 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            passed
in structure pointer from inetsrv.
               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, iTotalConnections, iTickCount;

    EnterCriticalSection(&CriticalSection);

    for(i=0, iTotalConnections = 0; i<Term.iAvailable; i++)
    {
        if ( Term.pClientData[i].inUse )
            iTotalConnections++;
    }

    if ( iTotalConnections >= 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;
    }

    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 successfull
*
* TRUE if an error occurs
and 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];
    char server[256];
    char database[256];

```

```

char user[256];
char password[256];

sprintf(szApp, "TPCC:%ld", (int)iTermId);

Term.pClientData[iTermId].dbproc = NULL;

sprintf(szApp, "TPCC:%ld", (int)iTermId);

Term.pClientData[iTermId].dbproc = NULL;

strcpy(server, szServer);
strcpy(database, szDatabase);
strcpy(user, szUser);
strcpy(password, szPassword);

if ( SQLOpenConnection(pECB, iTermId, iSyncId,
&Term.pClientData[iTermId].dbproc, server, database, user, password,
szApp, &Term.pClientData[iTermId].spid) )
{
    ErrorMessage(pECB, ERR_SQL_OPEN_CONNECTION,
ERR_TYPE_WEBDDL, NULL, iTermId, iSyncId);
    return TRUE;
}
return FALSE;
}

/* FUNCTION: BOOL Close(EXTENSION_CONTROL_BLOCK *pECB, int iTermId,
int iSyncId)
*
* PURPOSE: This function closes 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 of client browser
*
* RETURNS: BOOL FALSE if successfull
*
* TRUE if an error occurs
and connection cannot be terminated.
*
* COMMENTS: None
*/

static BOOL Close(EXTENSION_CONTROL_BLOCK *pECB, int iTermId, int iSyncId)
{
    PECBINFO pEcbInfo;

    if (Term.pClientData[iTermId].dbproc != NULL)
    {
        if ( (pEcbInfo =
(PECBINFO)dbgetuserdata(Term.pClientData[iTermId].dbproc)) )
        {
            pEcbInfo->iTermId = -1;
            pEcbInfo->iSyncId = -1;
            free(pEcbInfo); //free up user info
        }
    }
}

```

```

        return SQLCloseConnection(pECB,
Term.pClientData[iTermId].dbproc);
    }

    UNUSEDPARAM(iSyncId);
}

/* FUNCTION: 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)
*
* 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
*
* pointer to returned DBPROCESS **dbproc
*
* server name char *server SQL
*
* server database char *database SQL
*
* name char *user user
*
* password char *password user
*
* 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)
{
    RETCODE rc;
    char buffer[30];
    PECBINFO pEcbInfo;

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

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

if ( SQLSetConnectOption((*dbproc)->hdbc, SQL_PACKET_SIZE,
4096) == SQL_ERROR )
{
    ODBCError(*dbproc);
    return TRUE;
}

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

strcpy(buffer, "use tpcc set nocount on set XACT_ABORT
ON");

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

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

if ( SQLFetch((*dbproc)->hstmt) == SQL_ERROR )

```

```

    {
        ODBCError(*dbproc);
        return TRUE;
    }
    SQLFreeStmt((*dbproc)->hstmt, SQL_CLOSE);
    if ( bConnectionPooling )
        SQLDisconnect((*dbproc)->hdbc);
    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);

        DBSETLPACKET(login, (unsigned short)DEFCLPACKSIZE);

        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");

```

```

        dbsqlxexec(*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");

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

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

        dbsqlxexec(*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
DEPROCESS
*
* 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: SQLStockLevel(EXTENSION_CONTROL_BLOCK *pECB, int iTermId,
int iSyncId, 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.
*
* iTermId int terminal id of browser
*
* iSyncId int 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
*
*/

#ifdef USE_ODBC
    static int SQLStockLevel(EXTENSION_CONTROL_BLOCK *pECB, int
    iTermId, int iSyncId, DBPROCESS *dbproc, STOCK_LEVEL_DATA *pStockLevel,
    short deadlock_retry)
    {
        int tryit;
        PECBINFO pEcbInfo;

        //update pECB and bFailed flag
        if ( (pEcbInfo = (PECBINFO)dbgetuserdata(dbproc)) )
        {
            pEcbInfo->pECB = pECB;
            pEcbInfo->bFailed = FALSE;
            pEcbInfo->iTermId = iTermId;
            pEcbInfo->iSyncId = iSyncId;
        }
    }
#endif

#ifdef USE_ODBC
    if ( ReopenConnection(dbproc) )
        return -3;
#endif

```

```

        pStockLevel->num_deadlocks = 0;

        for (tryit=0; tryit<deadlock_retry; tryit++)
        {
            BindParameter(dbproc, 1,SQL_C_SSHORT, SQL_SMALLINT,
            0, 0, &pStockLevel->w_id, 0);
            BindParameter(dbproc, 2,SQL_C_STINYINT,
            SQL_TINYINT, 0, 0, &pStockLevel->d_id, 0);
            BindParameter(dbproc, 3,SQL_C_SSHORT, SQL_SMALLINT,
            0, 0, &pStockLevel->thresh_hold, 0);

            if ( !ExecuteStatement(dbproc, "{call
            tpcc_stocklevel(?,?,?)}") )
            {
                if ( !SQLDetectDeadlock(dbproc) )
                {
                    if ( BindColumn(dbproc, 1,
                    SQL_C_SSHORT, &pStockLevel->low_stock, 0) )
                        return TRUE;

                    if ( GetResults(dbproc) )
                        return TRUE;
                }
            }

            SQLFreeStmt(dbproc->hstmt, SQL_CLOSE);

            if ( SQLDetectDeadlock(dbproc) )
            {
                pStockLevel->num_deadlocks++;
                Sleep(10 * tryit);
            }
            else
            {
                strcpy(pStockLevel->execution_status,
                "Transaction committed.");
                return FALSE;
            }
        }

        // If we reached here, it means we quit after MAX_RETRY
        deadlocks
        strcpy(pStockLevel->execution_status, "Hit deadlock max.");

        return TRUE;
    }
#else
    static BOOL SQLStockLevel(EXTENSION_CONTROL_BLOCK *pECB, int
    iTermId, int iSyncId, 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->iSyncId = iSyncId;
    }

    pStockLevel->num_deadlocks = 0;

    for (tryit=0; tryit < deadlock_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 FALSE;
                }
            }

        // If we reached here, it means we quit after MAX_RETRY
        strcpy(pStockLevel->execution_status, "Hit deadlock max.
");
        return TRUE;
    }
#endif

```

```

/* FUNCTION: int SQLNewOrder(EXTENSION_CONTROL_BLOCK *pECB, int iTermId,
int iSyncId, int iTermId, int iSyncId, 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.
*
* iTermId int terminal id of browser
*
* iSyncId int sync id of browser
*
* *dbproc DBPROCESS 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
*
*/

#ifdef USE_ODBC
static int SQLNewOrder(EXTENSION_CONTROL_BLOCK *pECB, int
iTermId, int iSyncId, DBPROCESS *dbproc, NEW_ORDER_DATA *pNewOrder, short
deadlock_retry)
{
    int i;
    int j;
    int tryit;
    DBINT commit_flag;
    char buffer[255];
    PECBINFO pEcbInfo;

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

    if ( ReopenConnection(dbproc) )
        return -3;

    pNewOrder->num_deadlocks = 0;

    for (tryit=0; tryit<deadlock_retry; tryit++)
    {
        strcpy(buffer, "{call tpcc_neworder(?,?,?,?);");
        for (i=1; i<pNewOrder->o_ol_cnt; i++)
            strcat(buffer, "?,?;");
        strcat(buffer, "?,?;");
    }
}

```



```

        BindParameter(dbproc, 1, SQL_C_SSHORT,
SQL_SMALLINT, 0, 0, &pNewOrder->w_id, 0);
        BindParameter(dbproc, 2, SQL_C_STINYINT,
SQL_TINYINT, 0, 0, &pNewOrder->d_id, 0);
        BindParameter(dbproc, 3, SQL_C_SLONG,
SQL_INTEGER, 0, 0, &pNewOrder->c_id, 0);
        BindParameter(dbproc, 4, SQL_C_STINYINT,
SQL_TINYINT, 0, 0, &pNewOrder->o_ol_cnt, 0);

        pNewOrder->o_all_local = 1;
        for (j=0; j<pNewOrder->o_ol_cnt; j++)
        {
            if ( pNewOrder->o_all_local && pNewOrder-
>Ol[j].ol_supply_w_id != pNewOrder->w_id )
                pNewOrder->o_all_local = 0;
            BindParameter(dbproc, 5, SQL_C_STINYINT,
SQL_TINYINT, 0, 0, &pNewOrder->o_all_local, 0);

            for (j=0, i=0; i<(pNewOrder->o_ol_cnt * 3); i=i+3,
j++)
            {
                BindParameter(dbproc, (UWORD)(i+6),
SQL_C_SLONG, SQL_INTEGER, 0, 0, &pNewOrder->Ol[j].ol_i_id, 0);
                BindParameter(dbproc, (UWORD)(i+7),
SQL_C_SSHORT, SQL_SMALLINT, 0, 0, &pNewOrder->Ol[j].ol_supply_w_id, 0);
                BindParameter(dbproc, (UWORD)(i+8),
SQL_C_SSHORT, SQL_SMALLINT, 0, 0, &pNewOrder->Ol[j].ol_quantity, 0);
            }

            if ( ExecuteStatement(dbproc, buffer) )
                if ( !SQLDetectDeadlock(dbproc) )
                    return -2;

            pNewOrder->total_amount=0;

            for (i = 0; i<pNewOrder->o_ol_cnt; i++)
            {
                if ( BindColumn(dbproc,1, SQL_C_CHAR,
&pNewOrder->Ol[i].ol_i_name, sizeof(pNewOrder->Ol[i].ol_i_name)) )
                    return -2;
                if ( BindColumn(dbproc,2, SQL_C_SSHORT,
&pNewOrder->Ol[i].ol_stock, 0) )
                    return -2;
                if ( BindColumn(dbproc,3, SQL_C_CHAR,
&pNewOrder->Ol[i].ol_brand_generic, sizeof(pNewOrder-
>Ol[i].ol_brand_generic)) )
                    return -2;
                if ( BindColumn(dbproc,4, SQL_C_DOUBLE,
&pNewOrder->Ol[i].ol_i_price, 0) )
                    return -2;
                if ( BindColumn(dbproc,5, SQL_C_DOUBLE,
&pNewOrder->Ol[i].ol_amount, 0) )
                    return -2;

                if ( GetResults(dbproc) )
                    return -2;

                pNewOrder->total_amount = pNewOrder-
>total_amount + pNewOrder->Ol[i].ol_amount;

```

```

        if ( !pEcbInfo->bDeadlock )
        {
            if ( MoreResults(dbproc) )
                return -2;
        }
        if ( pEcbInfo->bDeadlock )
            break;
    }

    if ( !SQLDetectDeadlock(dbproc) )
    {
        if ( BindColumn(dbproc, 1, SQL_C_DOUBLE,
&pNewOrder->w_tax, 0) )
            return -2;
        if ( BindColumn(dbproc, 2, SQL_C_DOUBLE,
&pNewOrder->d_tax, 0) )
            return -2;
        if ( BindColumn(dbproc, 3, SQL_C_SLONG,
&pNewOrder->o_id, 0) )
            return -2;
        if ( BindColumn(dbproc, 4, SQL_C_CHAR,
&pNewOrder->c_last, sizeof(pNewOrder->c_last)) )
            return -2;
        if ( BindColumn(dbproc, 5, SQL_C_DOUBLE,
&pNewOrder->c_discount, 0) )
            return -2;
        if ( BindColumn(dbproc, 6, SQL_C_CHAR,
&pNewOrder->c_credit, sizeof(pNewOrder->c_credit)) )
            return -2;
        if ( BindColumn(dbproc, 7, SQL_C_TIMESTAMP,
&pNewOrder->o_entry_d, 0) )
            return -2;
        if ( BindColumn(dbproc, 8, SQL_C_SLONG,
&commit_flag, 0) )
            return -2;

        if ( GetResults(dbproc) )
            return -2;

        SQLFreeStmt(dbproc->hstmt, SQL_CLOSE);

        if ( commit_flag == 1 )
        {
            pNewOrder->total_amount = pNewOrder-
>total_amount * ((1 + pNewOrder->w_tax + pNewOrder->d_tax) * (1 -
pNewOrder->c_discount));
            strcpy(pNewOrder-
>execution_status,"Transaction committed.");
            return TRUE;
        }
        else
        {
            strcpy(pNewOrder-
>execution_status,"Item number is not valid.");
            return FALSE;
        }
    }
    else
    {
        SQLFreeStmt(dbproc->hstmt, SQL_CLOSE);
        pNewOrder->num_deadlocks++;
    }

```

```

        Sleep(DEADLOCKWAIT*tryit);
    }
}
// If we reached here, it means we quit after MAX_RETRY
deadlocks
strcpy(pNewOrder->execution_status,"Hit deadlock max. ");
return -1;
}
#else
static int SQLNewOrder(EXTENSION_CONTROL_BLOCK *pECB, int
iTermId, int iSyncId, DBPROCESS *dbproc, NEW_ORDER_DATA *pNewOrder, short
deadlock_retry)
{
    RETCODE          rc;
    int              i;
    DBINT            commit_flag;
    int              tryit;
    char             printbuf[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->iSyncId = iSyncId;
    }

    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);

            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))

                                pNewOrder->w_tax = *(DBFLT8 *) pData);

                            if (pData=dbdata(dbproc, 2))

                                pNewOrder->d_tax = *(DBFLT8 *) pData);

                            if (pData=dbdata(dbproc, 3))
                                pNewOrder->o_id = *(DBINT *) pData);
                            if (pData=dbdata(dbproc, 4))
                                UtilStrCpy(pNewOrder->c_last, pData, dbdatlen(dbproc, 4));
                            if (pData=dbdata(dbproc, 5))
                                pNewOrder->c_discount = *(DBFLT8 *) pData);

                            if (pData=dbdata(dbproc, 6))
                                UtilStrCpy(pNewOrder->c_credit, pData, dbdatlen(dbproc, 6));
                            if (pData=dbdata(dbproc, 7))
                                {
                                    datetime = *((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_discount));
                strcpy(pNewOrder-
>execution_status, "Transaction committed.");

                return TRUE;
            }
            else
            {
                strcpy(pNewOrder-
>execution_status, "Item number is not valid.");
                return FALSE;
            }
        }
    }
}

// If we reached here, it means we quit after MAX_RETRY
strcpy(pNewOrder->execution_status, "Hit deadlock max. ");
return -1; // "deadlock max retry reached!"
}
#endif

/* FUNCTION: int SQLPayment(EXTENSION_CONTROL_BLOCK *pECB, int iTermId,
int iSyncId, 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 terminal id of browser
*
* iSyncId sync id of browser
*
* *dbproc DBPROCESS
connection db process id
*
* PAYMENT_DATA *pPayment
pointer to payment input/output data structure
*
* short
deadlock_retry deadlock retry count
*
* RETURNS: int TRUE success
-1 reached max deadlocked
*
* COMMENTS: None
*

```

```

*/
#ifdef USE_ODBC
static int SQLPayment(EXTENSION_CONTROL_BLOCK *pECB, int iTermId,
int iSyncId, DBPROCESS *dbproc, PAYMENT_DATA *pPayment, short
deadlock_retry)
{
    int            tryit;
    char           printbuf[25];
    char           buffer[255];
    BOOL           deadlock_detected;
    PECBINFO       pEcbInfo;

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

    if ( ReopenConnection(dbproc) )
        return -3;

    pPayment->num_deadlocks = 0;

    for (tryit=0; tryit<deadlock_retry; tryit++)
    {
        deadlock_detected = FALSE;

        strcpy(buffer, "{call tpcc_payment(?,?,?,?,?,?)");
        if (pPayment->c_id == 0)
            strcat(buffer, ",?");
        strcat(buffer, "}")";

        BindParameter(dbproc, 1, SQL_C_SSHORT,
SQL_SMALLINT, 0, &pPayment->w_id, 0);
        BindParameter(dbproc, 2, SQL_C_SSHORT,
SQL_SMALLINT, 0, &pPayment->c_w_id, 0);
        BindParameter(dbproc, 3, SQL_C_DOUBLE,
SQL_NUMERIC, 6, &pPayment->h_amount, 0);
        BindParameter(dbproc, 4, SQL_C_STINYINT,
SQL_TINYINT, 0, &pPayment->d_id, 0);
        BindParameter(dbproc, 5, SQL_C_STINYINT,
SQL_TINYINT, 0, &pPayment->c_d_id, 0);
        BindParameter(dbproc, 6, SQL_C_SLONG,
SQL_INTEGER, (UINT)SQL_NTS, 0, &pPayment->c_id, 0);

        if (pPayment->c_id == 0)
            BindParameter(dbproc, 7, SQL_C_CHAR,
SQL_CHAR, (UINT)SQL_NTS, 0, &pPayment->c_last, sizeof(pPayment->c_last));

        if ( ExecuteStatement(dbproc, buffer) )
            if ( !pEcbInfo->bDeadlock )
                return -2;

        if ( !pEcbInfo->bDeadlock )
        {
            if ( BindColumn(dbproc, 1, SQL_C_SLONG,
&pPayment->c_id, 0) )

```

```

                return -2;
            if ( BindColumn(dbproc, 2, SQL_C_CHAR,
&pPayment->c_last, sizeof(pPayment->c_last)) )
                return -2;
            if ( BindColumn(dbproc, 3, SQL_C_TIMESTAMP,
&pPayment->h_date, 0) )
                return -2;
            if ( BindColumn(dbproc, 4, SQL_C_CHAR,
&pPayment->w_street_1, sizeof(pPayment->w_street_1)) )
                return -2;
            if ( BindColumn(dbproc, 5, SQL_C_CHAR,
&pPayment->w_street_2, sizeof(pPayment->w_street_2)) )
                return -2;
            if ( BindColumn(dbproc, 6, SQL_C_CHAR,
&pPayment->w_city, sizeof(pPayment->w_city)) )
                return -2;
            if ( BindColumn(dbproc, 7, SQL_C_CHAR,
&pPayment->w_state, sizeof(pPayment->w_state)) )
                return -2;
            if ( BindColumn(dbproc, 8, SQL_C_CHAR,
&pPayment->w_zip, sizeof(pPayment->w_zip)) )
                return -2;
            if ( BindColumn(dbproc, 9, SQL_C_CHAR,
&pPayment->d_street_1, sizeof(pPayment->d_street_1)) )
                return -2;
            if ( BindColumn(dbproc, 10, SQL_C_CHAR,
&pPayment->d_street_2, sizeof(pPayment->d_street_2)) )
                return -2;
            if ( BindColumn(dbproc, 11, SQL_C_CHAR,
&pPayment->d_city, sizeof(pPayment->d_city)) )
                return -2;
            if ( BindColumn(dbproc, 12, SQL_C_CHAR,
&pPayment->d_state, sizeof(pPayment->d_state)) )
                return -2;
            if ( BindColumn(dbproc, 13, SQL_C_CHAR,
&pPayment->d_zip, sizeof(pPayment->d_zip)) )
                return -2;
            if ( BindColumn(dbproc, 14, SQL_C_CHAR,
&pPayment->c_first, sizeof(pPayment->c_first)) )
                return -2;
            if ( BindColumn(dbproc, 15, SQL_C_CHAR,
&pPayment->c_middle, sizeof(pPayment->c_middle)) )
                return -2;
            if ( BindColumn(dbproc, 16, SQL_C_CHAR,
&pPayment->c_street_1, sizeof(pPayment->c_street_1)) )
                return -2;
            if ( BindColumn(dbproc, 17, SQL_C_CHAR,
&pPayment->c_street_2, sizeof(pPayment->c_street_2)) )
                return -2;
            if ( BindColumn(dbproc, 18, SQL_C_CHAR,
&pPayment->c_city, sizeof(pPayment->c_city)) )
                return -2;
            if ( BindColumn(dbproc, 19, SQL_C_CHAR,
&pPayment->c_state, sizeof(pPayment->c_state)) )
                return -2;
            if ( BindColumn(dbproc, 20, SQL_C_CHAR,
&pPayment->c_zip, sizeof(pPayment->c_zip)) )
                return -2;
            if ( BindColumn(dbproc, 21, SQL_C_CHAR,
&pPayment->c_phone, sizeof(pPayment->c_phone)) )
                return -2;

```

```

    if ( BindColumn(dbproc, 22, SQL_C_TIMESTAMP,
&pPayment->c_since, 0) )
        return -2;
    if ( BindColumn(dbproc, 23, SQL_C_CHAR,
&pPayment->c_credit, sizeof(pPayment->c_credit) ) )
        return -2;
    if ( BindColumn(dbproc, 24, SQL_C_DOUBLE,
&pPayment->c_credit_lim, 0) )
        return -2;
    if ( BindColumn(dbproc, 25, SQL_C_DOUBLE,
&pPayment->c_discount, 0) )
        return -2;
    if ( BindColumn(dbproc, 26, SQL_C_DOUBLE,
&pPayment->c_balance, 0) )
        return -2;
    if ( BindColumn(dbproc, 27, SQL_C_CHAR,
&pPayment->c_data, sizeof(pPayment->c_data) ) )
        return -2;

    if ( GetResults(dbproc) )
        return -2;
}

SQLFreeStmt(dbproc->hstmt, SQL_CLOSE);

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.");
        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. ");
return -1;
}

#else
static int SQLPayment(EXTENSION_CONTROL_BLOCK *pECB, int iTermId,
int iSyncId, DBPROCESS *dbproc, PAYMENT_DATA *pPayment, short
deadlock_retry)
{
    RETCODE    rc;
    int        tryit;
    char       printbuf[26];

```

```

DBDATETIME    datetime;
BYTE          *pData;
PECBINFO      pEcbInfo;

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

pPayment->num_deadlocks = 0;

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 =
*((DBDATETIME *) pData);

```

```

dbdatecrack(dbproc, &pPayment->h_date, &datetime);
        }

if (pData=dbdata(dbproc, 4))
UtilStrCpy(pPayment->w_street_1, pData, dbdatlen(dbproc, 4));
if (pData=dbdata(dbproc, 5))
UtilStrCpy(pPayment->w_street_2, pData, dbdatlen(dbproc, 5));
if (pData=dbdata(dbproc, 6))
UtilStrCpy(pPayment->w_city, pData, dbdatlen(dbproc, 6));
if (pData=dbdata(dbproc, 7))
UtilStrCpy(pPayment->w_state, pData, dbdatlen(dbproc, 7));
if (pData=dbdata(dbproc, 8))
UtilStrCpy(pPayment->w_zip, pData, dbdatlen(dbproc, 8));
if (pData=dbdata(dbproc, 9))
UtilStrCpy(pPayment->d_street_1, pData, dbdatlen(dbproc, 9));
if (pData=dbdata(dbproc, 10))
UtilStrCpy(pPayment->d_street_2, pData, dbdatlen(dbproc, 10));
if (pData=dbdata(dbproc, 11))
UtilStrCpy(pPayment->d_city, pData, dbdatlen(dbproc, 11));
if (pData=dbdata(dbproc, 12))
UtilStrCpy(pPayment->d_state, pData, dbdatlen(dbproc, 12));
if (pData=dbdata(dbproc, 13))
UtilStrCpy(pPayment->d_zip, pData, dbdatlen(dbproc, 13));
if (pData=dbdata(dbproc, 14))
UtilStrCpy(pPayment->c_first, pData, dbdatlen(dbproc, 14));
if (pData=dbdata(dbproc, 15))
UtilStrCpy(pPayment->c_middle, pData, dbdatlen(dbproc, 15));
if (pData=dbdata(dbproc, 16))
UtilStrCpy(pPayment->c_street_1, pData, dbdatlen(dbproc, 16));
if (pData=dbdata(dbproc, 17))
UtilStrCpy(pPayment->c_street_2, pData, dbdatlen(dbproc, 17));
if (pData=dbdata(dbproc, 18))

```

```

UtilStrCpy(pPayment->c_city, pData, dbdatlen(dbproc, 18));
if (pData=dbdata(dbproc, 19))
UtilStrCpy(pPayment->c_state, pData, dbdatlen(dbproc, 19));
if (pData=dbdata(dbproc, 20))
UtilStrCpy(pPayment->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 =
*( (DBDATETIME *) pData);
        dbdatecrack(dbproc, &pPayment->c_since, &datetime);
        }
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.");
                        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. ");
    return -1; //"deadlock max retry reached!"
}
#endif

/* 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.
*
* iTermId int terminal id of browser
*
* iSyncId int sync id of browser
*
* *dbproc DBPROCESS connection db process id
*
* ORDER_STATUS_DATA *pOrderStatus
pointer to Order Status data input/output structure
*
* short
deadlock_retry deadlock_retry count
*
* RETURNS: int -1 max deadlock reached
*
* 0 No orders found for
customer
*
* 1 Transaction
successful
*
* COMMENTS: None
*
*/

#ifndef USE_ODBC
static int SQLOrderStatus(EXTENSION_CONTROL_BLOCK *pECB, int
iTermId, int iSyncId, DBPROCESS *dbproc, ORDER_STATUS_DATA *pOrderStatus,
short deadlock_retry)
{
    int tryit;
    int i;
    BOOL not_done;
    char buffer[255];
    PECBINFO pEcbInfo;

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

```

```

if ( ReopenConnection(dbproc) )
    return -3;

pOrderStatus->num_deadlocks = 0;

for (tryit=0; tryit < deadlock_retry; tryit++)
{
    pEcbInfo->bDeadlock = FALSE;

    strcpy(buffer,"{call tpcc_orderstatus(?,?,?");
    if (pOrderStatus->c_id == 0)
        strcat(buffer,"?");
    strcat(buffer,"}");

    BindParameter(dbproc, 1, SQL_C_SSHORT,
SQL_SMALLINT, 0, 0, &pOrderStatus->w_id, 0);
    BindParameter(dbproc, 2, SQL_C_STINYINT,
SQL_TINYINT, 0, 0, &pOrderStatus->d_id, 0);
    BindParameter(dbproc, 3, SQL_C_SLONG, SQL_INTEGER,
0, 0, &pOrderStatus->c_id, 0);
    if (pOrderStatus->c_id == 0)
        BindParameter(dbproc, 4, SQL_C_CHAR,
SQL_CHAR, (UINT)SQL_NTS, 0, &pOrderStatus->c_last, sizeof(pOrderStatus-
>c_last));

    if ( ExecuteStatement(dbproc, buffer) )
        if ( !SQLDetectDeadlock(dbproc) )
            return -2;

    not_done = TRUE;
    i=0;

    while ( not_done && !pEcbInfo->bDeadlock )
    {
        if ( BindColumn(dbproc, 1, SQL_C_SSHORT,
&pOrderStatus->olOrderStatusData[i].ol_supply_w_id, 0) )
            return -2;
        if ( BindColumn(dbproc, 2, SQL_C_SLONG,
&pOrderStatus->olOrderStatusData[i].ol_i_id, 0) )
            return -2;
        if ( BindColumn(dbproc, 3, SQL_C_SSHORT,
&pOrderStatus->olOrderStatusData[i].ol_quantity, 0) )
            return -2;
        if ( BindColumn(dbproc, 4, SQL_C_DOUBLE,
&pOrderStatus->olOrderStatusData[i].ol_amount, 0) )
            return -2;
        if ( BindColumn(dbproc, 5, SQL_C_TIMESTAMP,
&pOrderStatus->olOrderStatusData[i].ol_delivery_d, 0) )
            return -2;

        switch( SQLFetch(dbproc->hstmt) )
        {
            case SQL_ERROR:
                if ( !pEcbInfo->bDeadlock )
                    return -2;
                break;
            case SQL_NO_DATA_FOUND:
                not_done = FALSE;
                break;
            default:

```

```

        i++;
        break;
    }
}
pOrderStatus->o_ol_cnt = i;
if ( i )
{
    if ( !pEcbInfo->bDeadlock )
    {
        if ( MoreResults(dbproc) )
        {
            if ( !pEcbInfo->bDeadlock )
                return -2;
        }
        else
        {
            if ( !pEcbInfo->bDeadlock )
            {
                if (
                    BindColumn(dbproc, 1, SQL_C_SLONG, &pOrderStatus->c_id, 0) )
                    return -2;
                if (
                    BindColumn(dbproc, 2, SQL_C_CHAR, &pOrderStatus->c_last,
                    sizeof(pOrderStatus->c_last)) )
                    return -2;
                if (
                    BindColumn(dbproc, 3, SQL_C_CHAR, &pOrderStatus->c_first,
                    sizeof(pOrderStatus->c_first)) )
                    return -2;
                if (
                    BindColumn(dbproc, 4, SQL_C_CHAR, &pOrderStatus->c_middle,
                    sizeof(pOrderStatus->c_middle)) )
                    return -2;
                if (
                    BindColumn(dbproc, 5, SQL_C_TIMESTAMP, &pOrderStatus->o_entry_d, 0) )
                    return -2;
                if (
                    BindColumn(dbproc, 6, SQL_C_SSHORT, &pOrderStatus->o_carrier_id, 0) )
                    return -2;
                if (
                    BindColumn(dbproc, 7, SQL_C_DOUBLE, &pOrderStatus->c_balance, 0) )
                    return -2;
                if (
                    BindColumn(dbproc, 8, SQL_C_SLONG, &pOrderStatus->o_id, 0) )
                    return -2;
                if (
                    GetResults(dbproc) )
                    return -2;
            }
        }
    }
}
else
{
    SQLFreeStmt(dbproc->hstmt, SQL_CLOSE);
    return 0; //"No orders found for customer"
}
}

```

```

SQLFreeStmt(dbproc->hstmt, SQL_CLOSE);
if ( pEcbInfo->bDeadlock )
{
    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.");
    else
        strcpy(pOrderStatus->execution_status, "Transaction committed.");
    return 1;
}
}
// If we reached here, it means we quit after MAX_RETRY
strcpy(pOrderStatus->execution_status, "Hit deadlock max.");
return -1;
}
#else
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];
    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;

    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-
>OlOrderStatusData[i].ol_supply_w_id = (*(DBSMALLINT *) pData);
                        if (pData=dbdata(dbproc, 2))
                                pOrderStatus-
>OlOrderStatusData[i].ol_i_id = (*(DBINT *) pData);
                        if (pData=dbdata(dbproc, 3))
                                pOrderStatus-
>OlOrderStatusData[i].ol_quantity = (*(DBSMALLINT *) pData);
                        if (pData=dbdata(dbproc, 4))
                                pOrderStatus-
>OlOrderStatusData[i].ol_amount = (*(DBFLT8 *) pData);
                        if (pData=dbdata(dbproc, 5))
                                {
                                    datetime =
*(DBDATETIME *) pData);
                                    dbdatecrack(dbproc, &pOrderStatus-
>OlOrderStatusData[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.");
                                    else
                                        strcpy(pOrderStatus-
>execution_status, "Transaction committed.");
                                    return 1;
                                }
                            }
                        // If we reached here, it means we quit after MAX_RETRY
                        strcpy(pOrderStatus->execution_status, "Hit deadlock max.
");
                        return -1; //"deadlock max retry reached!"
                    }
                }
            }
        }
    #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;
}

/* 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\">");
        sprintf(szForm+strlen(szForm), "<INPUT TYPE=\"hidden\"
NAME=\"FORMID\" VALUE=\"%d\">", STOCK_LEVEL_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>                               Stock-
Level<BR>");
        wsprintf(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
        {
            wsprintf(szForm+strlen(szForm), "Stock Level Threshold:
%2.2d<BR><BR>", Term.pClientData[iTermId].stockLevelData.thresh_hold);

            wsprintf(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 id
 *
                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\">");
    wsprintf(szForm+strlen(szForm), "<INPUT TYPE=\"hidden\"
NAME=\"TERMINID\" VALUE=\"%d\">", iTermId);
    wsprintf(szForm+strlen(szForm), "<INPUT TYPE=\"hidden\"
NAME=\"SYNCID\" VALUE=\"%d\">", iSyncId);
    wsprintf(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, 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 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=\"0\">");
        else
            sprintf(szForm+strlen(szForm), "<INPUT
TYPE=\"hidden\" NAME=\"STATUSID\" VALUE=\"%d\">", ERR_BAD_ITEM_ID);
    }

```

```

    sprintf(szForm+strlen(szForm), "<INPUT TYPE=\"hidden\"
NAME=\"FORMID\" VALUE=\"%d\">", NEW_ORDER_FORM);
    sprintf(szForm+strlen(szForm), "<INPUT TYPE=\"hidden\"
NAME=\"TERMID\" VALUE=\"%d\">", iTermId);
    sprintf(szForm+strlen(szForm), "<INPUT TYPE=\"hidden\"
NAME=\"SYNCID\" VALUE=\"%d\">", iSyncId);
    strcat(szForm, "<PRE>"
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 of Lines: W_tax: D_tax:<BR><BR>"
" Supp_W Item_Id Item Name
Qty Stock B/G Price Amount<BR>"
" <INPUT NAME=\"SP00*\"
SIZE=4> <INPUT NAME=\"IID00*\" SIZE=6> <INPUT
NAME=\"Qty00*\" SIZE=1><BR>"
" <INPUT NAME=\"SP01*\"
SIZE=4> <INPUT NAME=\"IID01*\" SIZE=6> <INPUT
NAME=\"Qty01*\" SIZE=1><BR>"
" <INPUT NAME=\"SP02*\"
SIZE=4> <INPUT NAME=\"IID02*\" SIZE=6> <INPUT
NAME=\"Qty02*\" SIZE=1><BR>"
" <INPUT NAME=\"SP03*\"
SIZE=4> <INPUT NAME=\"IID03*\" SIZE=6> <INPUT
NAME=\"Qty03*\" SIZE=1><BR>"
" <INPUT NAME=\"SP04*\"
SIZE=4> <INPUT NAME=\"IID04*\" SIZE=6> <INPUT
NAME=\"Qty04*\" SIZE=1><BR>"
" <INPUT NAME=\"SP05*\"
SIZE=4> <INPUT NAME=\"IID05*\" SIZE=6> <INPUT
NAME=\"Qty05*\" SIZE=1><BR>"
" <INPUT NAME=\"SP06*\"
SIZE=4> <INPUT NAME=\"IID06*\" SIZE=6> <INPUT
NAME=\"Qty06*\" SIZE=1><BR>"
" <INPUT NAME=\"SP07*\"
SIZE=4> <INPUT NAME=\"IID07*\" SIZE=6> <INPUT
NAME=\"Qty07*\" SIZE=1><BR>"
" <INPUT NAME=\"SP08*\"
SIZE=4> <INPUT NAME=\"IID08*\" SIZE=6> <INPUT
NAME=\"Qty08*\" SIZE=1><BR>"
" <INPUT NAME=\"SP09*\"
SIZE=4> <INPUT NAME=\"IID09*\" SIZE=6> <INPUT
NAME=\"Qty09*\" SIZE=1><BR>"
" <INPUT NAME=\"SP10*\"
SIZE=4> <INPUT NAME=\"IID10*\" SIZE=6> <INPUT
NAME=\"Qty10*\" SIZE=1><BR>"
" <INPUT NAME=\"SP11*\"
SIZE=4> <INPUT NAME=\"IID11*\" SIZE=6> <INPUT
NAME=\"Qty11*\" SIZE=1><BR>"
" <INPUT NAME=\"SP12*\"
SIZE=4> <INPUT NAME=\"IID12*\" SIZE=6> <INPUT
NAME=\"Qty12*\" SIZE=1><BR>"
" <INPUT NAME=\"SP13*\"
SIZE=4> <INPUT NAME=\"IID13*\" SIZE=6> <INPUT
NAME=\"Qty13*\" SIZE=1><BR>"
" <INPUT NAME=\"SP14*\"
SIZE=4> <INPUT NAME=\"IID14*\" SIZE=6> <INPUT
NAME=\"Qty14*\" SIZE=1><BR>"
"Execution Status:
Total:<BR><HR>"
" <INPUT TYPE=\"submit\"
NAME=\"CMD\" VALUE=\"Process\">"
" <INPUT TYPE=\"submit\"
NAME=\"CMD\" VALUE=\"Menu\">"
"</FORM>"

```

```

        "</HTML>" );
    }
    else
    {
        if ( bValid )
        {
            wsprintf(szForm+strlen(szForm), "Warehouse: %4.4d
District: %2.2d
%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
District: %2.2d
                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 Credit: %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
Number 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_cnt,
            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_ol_cnt; i++)
        {
            FormatHTMLString(szName,
            Term.pClientData[iTermId].newOrderData.Ol[i].ol_i_name, 24);
            sprintf(szForm+strlen(szForm), " %4.4d
%6.6d %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>"
                "<INPUT TYPE=\"submit\" "
                "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 iSyncId, BOOL bInput)
 *
 * 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
 *
 * 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\">");
        sprintf(szForm+strlen(szForm), "<INPUT TYPE=\"hidden\"
NAME=\"FORMID\" VALUE=\"%d\">", PAYMENT_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>
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><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 Cust 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 <= 1 )
                    UtilStrCpy(szTmp, ptr, 50);
                else
                    szTmp[0] = 0;
                if ( !i )
                {
                    FormatHTMLString(szTmpStr1, szTmp,
50);
                    wsprintf(szForm+strlen(szForm),
"Cust-Data: %s<BR>", szTmpStr1);

```

```

    }
    else
    {
        FormatHTMLString(szTmpStr1, szTmp,
50);
        wsprintf(szForm+strlen(szForm), "
%s<BR>", szTmpStr1);
    }
}
else
    strcat(szForm, "Cust-Data: <BR><BR><BR><BR>");
    strcat(szForm, "</PRE><HR><BR>"
        "<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 *MakeOrderStatusForm(int iTermId, int iSyncId, BOOL
bInput)
*
* PURPOSE: This function
*
* ARGUMENTS: int iTermId client browser
terminal id
*
* browser sync id int iSyncId client
*
* 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_id;
        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\">");
            wsprintf(szForm+strlen(szForm), "<INPUT TYPE=\"hidden\"
NAME=\"FORMID\" VALUE=\"%d\">", ORDER_STATUS_FORM);
            wsprintf(szForm+strlen(szForm), "<INPUT TYPE=\"hidden\"
NAME=\"TERMID\" VALUE=\"%d\">", iTermId);
            wsprintf(szForm+strlen(szForm), "<INPUT TYPE=\"hidden\"
NAME=\"SYCID\" VALUE=\"%d\">", iSyncId);
            strcat(szForm, "<PRE> Order-
Status<BR>" );
            wsprintf(szForm+strlen(szForm), "Warehouse: %4.4d ",
Term.pClientData[iTermId].orderStatusData.w_id);
            if ( bInput )
            {
                strcat(szForm, "District: <INPUT NAME=\"DID*\" SIZE=1><BR>"
                    "Customer: <INPUT
NAME=\"CID*\" SIZE=4> Name: <INPUT NAME=\"CLT*\"
SIZE=23><BR>"
                    "Cust-Balance:<BR><BR>"
                    "Order-Number:
Entry-Date: Carrier-Number:
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_ol_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.OlOrderStatusData[i].ol_
            supply_w_id,
            Term.pClientData[iTermId].orderStatusData.OlOrderStatusData[i].ol_
            i_id,
            Term.pClientData[iTermId].orderStatusData.OlOrderStatusData[i].ol_
            quantity,
            Term.pClientData[iTermId].orderStatusData.OlOrderStatusData[i].ol_
            amount,
            Term.pClientData[iTermId].orderStatusData.OlOrderStatusData[i].ol_
            delivery_d.day,
            Term.pClientData[iTermId].orderStatusData.OlOrderStatusData[i].ol_
            delivery_d.month,
            Term.pClientData[iTermId].orderStatusData.OlOrderStatusData[i].ol_
            delivery_d.year);
        }

        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..\">"

```

```

        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 iSyncId, BOOL bInput,
BOOL bSuccess)
*
* PURPOSE: This function
*
* ARGUMENTS: int iTermId client browser
terminal id
*
* browser sync id int iSyncId client
*
* is being constructed for input else FALSE BOOL bInput TRUE if form
*
* Delivery succeeded else FALSE BOOL bSuccess TRUE if
*
* 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 iSyncId, 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);
    else
        strcat(szForm, "<INPUT TYPE=\"hidden\"
NAME=\"STATUSID\" VALUE=\"0\">");
    }

    wsprintf(szForm+strlen(szForm), "<INPUT TYPE=\"hidden\"
NAME=\"FORMID\" VALUE=\"%d\">", DELIVERY_FORM);
    wsprintf(szForm+strlen(szForm), "<INPUT TYPE=\"hidden\"
NAME=\"TERMID\" VALUE=\"%d\">", iTermId);
    wsprintf(szForm+strlen(szForm), "<INPUT TYPE=\"hidden\"
NAME=\"SYNCID\" VALUE=\"%d\">", iSyncId);

    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 pSrc to pDest
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
 */

static void ProcessNewOrderForm(EXTENSION_CONTROL_BLOCK *pECB, int
iTermId, int iSyncId)
{
    int iRc;
    int iError;

```

```

    PECBINFO      pEcbInfo;

    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;
    }

    iRc = SQLNewOrder(pECB, iTermId, iSyncId,
Term.pClientData[iTermId].dbproc, &Term.pClientData[iTermId].newOrderData,
iDeadlockRetry);

#ifdef USE_ODBC
    #if (ODBCVER >= 0x0300)
        if ( bConnectionPooling && iRc != -3 )
            SQLDisconnect(Term.pClientData[iTermId].dbproc-
>hdbc);
    #endif
#endif

    if ( (pEcbInfo =
(PECBINFO)dbgetuserdata(Term.pClientData[iTermId].dbproc)) )
    {
        if ( pEcbInfo->bFailed )
            return;
    }

    if ( iRc < 0 )
        ErrorMessage(pECB, ERR_NEW_ORDER_NOT_PROCESSED,
ERR_TYPE_WEBDLL, NULL, iTermId, iSyncId);
    else
        WriteZString(pECB, MakeNewOrderForm(iTermId, iSyncId,
FALSE, (BOOL)iRc) );

    return;
}

/* 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 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
*
*/

static void ProcessPaymentForm(EXTENSION_CONTROL_BLOCK *pECB, int iTermId,
int iSyncId)
{
    int          iRc;
    int          iError;
    PECBINFO     pEcbInfo;

    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,
iSyncId);
        return;
    }

    iRc = SQLPayment(pECB, iTermId, iSyncId,
Term.pClientData[iTermId].dbproc, &Term.pClientData[iTermId].paymentData,
iDeadlockRetry);

#ifdef USE_ODBC
    #if (ODBCVER >= 0x0300)
        if ( bConnectionPooling && iRc != -3 )
            SQLDisconnect(Term.pClientData[iTermId].dbproc-
>hdbc);
    #endif
#endif

    if ( (pEcbInfo =
(PECBINFO)dbgetuserdata(Term.pClientData[iTermId].dbproc)) )
    {
        if ( pEcbInfo->bFailed )
            return;
    }

    if ( iRc == 0 )
        ErrorMessage(pECB, ERR_PAYMENT_INVALID_CUSTOMER,
ERR_TYPE_WEBDLL, NULL, iTermId, iSyncId);
    else if ( iRc < 0 )
        ErrorMessage(pECB, ERR_PAYMENT_NOT_PROCESSED,
ERR_TYPE_WEBDLL, NULL, iTermId, iSyncId);
    else
        WriteZString(pECB, MakePaymentForm(iTermId, iSyncId, FALSE)
);

    return;
}

```

```

}

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

static void ProcessOrderStatusForm(EXTENSION_CONTROL_BLOCK *pECB, int
iTermId, int iSyncId)
{
    int iRc;
    int iError;
    PECBINFO pEcbInfo;

    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;
    }

    iRc = SQLOrderStatus(pECB, iTermId, iSyncId,
Term.pClientData[iTermId].dbproc,
&Term.pClientData[iTermId].orderStatusData, iDeadlockRetry);

#ifdef USE_ODBC
    #if (ODBCVER >= 0x0300)
        if ( bConnectionPooling && iRc != -3 )
            SQLDisconnect(Term.pClientData[iTermId].dbproc-
>hdbc);
    #endif
#endif
}

```

```

    if ( (pEcbInfo =
(PECBINFO)dbgetuserdata(Term.pClientData[iTermId].dbproc)) )
    {
        if ( pEcbInfo->bFailed )
            return;
    }

    if ( iRc == 0 )
        ErrorMessage(pECB, ERR_NOSUCH_CUSTOMER, ERR_TYPE_WEBDLL,
NULL, iTermId, iSyncId);
    else if ( iRc < 0 )
        ErrorMessage(pECB, ERR_ORDER_STATUS_NOT_PROCESSED,
ERR_TYPE_WEBDLL, NULL, iTermId, iSyncId);
    else
        WriteZString(pECB, MakeOrderStatusForm(iTermId, iSyncId,
FALSE) );

    return;
}

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

static void ProcessDeliveryForm(EXTENSION_CONTROL_BLOCK *pECB, int
iTermId, int iSyncId)
{
    char szTmp[26];
    BOOL bSuccess;

    memset(&Term.pClientData[iTermId].deliveryData, 0,
sizeof(DELIVERY_DATA));

    Term.pClientData[iTermId].deliveryData.w_id =
Term.pClientData[iTermId].w_id;

    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;
    }

    //post delivery info
    if ( PostDeliveryInfo(Term.pClientData[iTermId].deliveryData.w_id,
Term.pClientData[iTermId].deliveryData.o_carrier_id) )
    {
        strcpy(Term.pClientData[iTermId].deliveryData.execution_status,
"Delivery Post Failed");
        bSuccess = FALSE;
    }
    else
    {
        strcpy(Term.pClientData[iTermId].deliveryData.execution_status,
"Delivery has been queued.");
        bSuccess = TRUE;
    }

    WriteZString(pECB, MakeDeliveryForm(iTermId, iSyncId, FALSE,
bSuccess) );

    return;
}

/* FUNCTION: void ProcessStockLevelForm(EXTENSION_CONTROL_BLOCK *pECB, int
iTermId, int iSyncId)
*
* 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 iSyncId client browser sync id
*

```

```

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

static void ProcessStockLevelForm(EXTENSION_CONTROL_BLOCK *pECB, int
iTermId, int iSyncId)
{
    char szTmp[26];
    int iRc;
    PECBINFO pEcbInfo;

    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;
    }

    iRc = SQLStockLevel(pECB, iTermId, iSyncId,
Term.pClientData[iTermId].dbproc,
&Term.pClientData[iTermId].stockLevelData, iDeadlockRetry);

#ifdef USE_ODBC
    #if (ODBCVER >= 0x0300)
        if ( bConnectionPooling && iRc != -3 )
            SQLDisconnect(Term.pClientData[iTermId].dbproc-
>hdbc);
    #endif
#endif
}

```

```

    if ( (pEcbInfo =
(PECBINFO)dbgetuserdata(Term.pClientData[iTermId].dbproc)) )
    {
        if ( pEcbInfo->bFailed )
            return;
    }

    if ( iRc )
        ErrorMessage(pECB, ERR_STOCKLEVEL_NOT_PROCESSED,
ERR_TYPE_WEBDLL, NULL, iTermId, iSyncId);
    else
        WriteZString(pECB, MakeStockLevelForm(iTermId, iSyncId,
FALSE) );

    return;
}

/* FUNCTION: int GetNewOrderData(LPSTR lpszQueryString, NEW_ORDER_DATA
*pNewOrderData)
*
* PURPOSE: This function extracts and validates the new order form
data from an http 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, "DID*", 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 ||
pNewOrderData->Ol[i].ol_i_id < 1 )
                return ERR_NEWORDER_ITEMID_RANGE;
            if ( pNewOrderData->Ol[i].ol_quantity >= 100 ||
pNewOrderData->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 )
                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 status data 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;
_strdupr( 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
their default value
* 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;
iMaxWareHouses = 500;
iThreads = 5;
iQSlotts = 3000;
iDelayMs = 100;
iDeadlockRetry = (short)3;
strcpy(szTpccLogPath, "tpcclog.");

#ifdef USE_ODBC
bConnectionPooling = FALSE;
#endif

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, "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, "QueueSlotts", 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;

#ifdef USE_ODBC
    #if ( ODBCVER >= 0x0300 )
        size = sizeof(szTmp);
        if ( RegQueryValueEx(hKey, "ConnectionPooling", 0, &type,
szTmp, &size) == ERROR_SUCCESS )
            if ( !strcmp(szTmp, "ON") )
                bConnectionPooling = TRUE;

        iConnectDelay = 500;
        size = sizeof(szTmp);
        if ( RegQueryValueEx(hKey, "ConnectionPoolRetryTime", 0,
&type, szTmp, &size) == ERROR_SUCCESS )
            iConnectDelay = atoi(szTmp);
        if ( !iConnectDelay )
            iConnectDelay = 500;
    #endif
#endif

    RegCloseKey(hKey);

```

```

        return FALSE;
    }

/* FUNCTION: BOOL PostDeliveryInfo(short w_id, short o_carrier_id)
 *
 * PURPOSE: This function writes the delivery information to the
delivery pipe. The information is
 *          sent as a long.
 *
 * ARGUMENTS: short w_id warehouse id
 *             short o_carrier_id
 *
 * RETURNS: BOOL FALSE delivery information posted
successfully
 *           TRUE error cannot post
delivery info
 *
 * COMMENTS: The pipe is initially created with 16K buffer size this
should allow for
 *           up to 4096 deliveries to be queued before an
overflow condition would
 *           occur. The only reason that an overflow
would occur is if the delivery
 *           application stopped listening while
deliveries were being posted.
 */

static BOOL PostDeliveryInfo(short w_id, short o_carrier_id)
{
    DELIVERY_TRANSACTION deliveryTransaction;
    int d;
    int i;

    GetLocalTime(&deliveryTransaction.queue);

    deliveryTransaction.w_id = w_id;
    deliveryTransaction.o_carrier_id = o_carrier_id;

    for(i=0; i<4; i++)
    {
        if ( WriteFile(hPipe, &deliveryTransaction,
sizeof(deliveryTransaction), &d, NULL) )
            return FALSE;

        if ( GetLastError() != ERROR_PIPE_BUSY )
            //ERROR_PIPE_LISTENING
            return TRUE;
    }

    return TRUE;
}

/* 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;
}

#ifdef USE_ODBC

/* FUNCTION: void dbsetuserdata(PDBPROCESS dbproc, void *uPtr)
*
* PURPOSE:    This function sets a user pointer in a dbproc
structure
*              This functionality is not provided in odbc
so this function
*              provides it.
*
* ARGUMENTS: DBRPOCESS    dbproc ODBC dbprocess structure
*              void        *uPtr  returned data
user pointer
*
* RETURNS:    none
*
* COMMENTS:   The caller is responsible for the contents of the
uPtr.
*
*/

void dbsetuserdata(PDBPROCESS dbproc, void *uPtr)
{
    dbproc->uPtr = uPtr;
}

```

```

}
/* FUNCTION: void dbsetuserdata(PDBPROCESS dbproc, void *uPtr)
*
* PURPOSE: This function returns the user pointer stored in a
dbproc structure
* This functionality is not provided in odbc
so this function
* provides it.
*
* ARGUMENTS: DBRPOCESS dbproc ODBC dbprocess structure
*
* RETURNS: none
*
* COMMENTS: The returned pointer is placed in the dbproc
structure by the dbsetuserdata() API.
*/

void *dbgetuserdata(PDBPROCESS dbproc)
{
return dbproc->uPtr;
}

/* FUNCTION: void BindParameter(PDBPROCESS dbproc, UWORD ipar,
SWORD fCType, SWORD fSqlType, UDWORD cbColDef, SWORD ibScale, PTR
rgbValue, SDWORD cbValueMax)
*
* PURPOSE: This function wraps the functionality provided by
the SQLBindParameter
* allowing error process so that each bind
call does not need to provide
* error and message checking.
*
* ARGUMENTS: PDBPROCESS dbproc pointer to odbc
dbprocess structure
* UWORD ipar Parameter
number, ordered sequentially left to right, starting at 1.
* SWORD fParamType The type of
the parameter.
* SWORD fCType The C data
type of the parameter.
* SWORD fSqlType The SQL data
type of the parameter.
* UDWORD cbColDef The precision
of the column or expression
* of the corresponding parameter marker.
* SWORD ibScale The scale of
the column or expression of the corresponding
* parameter marker.
* PTR rgbValue A
pointer to a buffer for the parameter's data.
* SDWORD cbValueMax Maximum length
of the rgbValue buffer.
* void *uPtr returned data
user pointer
*
* RETURNS: none
*

```

```

* COMMENTS: The returned pointer is placed in the dbproc
structure by the dbset
*
*/

void BindParameter(PDBPROCESS dbproc, UWORD ipar, SWORD fCType,
SWORD fSqlType, UDWORD cbColDef, SWORD ibScale, PTR rgbValue, SDWORD
cbValueMax)
{
RETCODE rc;

if ( ((PECBINFO)dbgetuserdata(dbproc))->bFailed )
return;
rc = SQLBindParameter(dbproc->hstmt, ipar, SQL_PARAM_INPUT,
fCType, fSqlType, cbColDef, ibScale, rgbValue, cbValueMax, NULL);
if (rc == SQL_ERROR)
ODBCError(dbproc);
return;
}

/* FUNCTION: void ODBCError(PDBPROCESS dbproc)
*
* PURPOSE: This function wraps the odbc error call so that the
dblib msg_handler is called.
* This allows the deadlock flag in the dbproc
user data structure pEcbInfo in
dbproc to be set if necessary.
*
* ARGUMENTS: DBRPOCESS dbproc ODBC dbprocess structure
*
* RETURNS: none
*
* COMMENTS: none
*/

void ODBCError(PDBPROCESS dbproc)
{
SDWORD lNativeError;
char szState[6];
char szMsg[SQL_MAX_MESSAGE_LENGTH];
char szMsgText[256];
PECBINFO pEcbInfo;
char szTmp[256];
FILE *fp;
SYSTEMTIME systemTime;

pEcbInfo = (PECBINFO)dbgetuserdata(dbproc);
while( SQLError(henv, dbproc->hdbc, dbproc->hstmt, szState,
&lNativeError, szMsg, sizeof(szMsg), NULL) == SQL_SUCCESS )
{
msg_handler(dbproc, lNativeError, 0, 0, szMsg);
if ( !lNativeError )
{
sprintf(szMsgText, "State = %s, %s",
szState, szMsg);
ErrorMessage(pEcbInfo->pECB, -1,
ERR_TYPE_ODBC, szMsgText, pEcbInfo->iTermId, pEcbInfo->iSyncId);
pEcbInfo->bFailed = TRUE;

GetLocalTime(&systemTime);
}
}
}

```

```

        fp = fopen(szErrorLogPath, "ab");

        EnterCriticalSection(&ErrorLogCriticalSection);
        sprintf(szTmp, "Error: SQLSVR(): %s",
szMsg);
        fprintf(fp, "%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);

        LeaveCriticalSection(&ErrorLogCriticalSection);

        fclose(fp);
    }
}

return;
}

/* FUNCTION: BOOL ExecuteStatement(PDBPROCESS dbproc, szStatement)
*
* PURPOSE: This function wraps the odbc SQLExecDirect API so
that error handling and
*
* and deadlock are taken care of in a common
location.
*
* ARGUMENTS: DBRPOCESS dbproc ODBC dbprocess structure
* char *szStatement sql
stored procedure statement to be executed.
*
* RETURNS: none
*
* COMMENTS: none
*/

BOOL ExecuteStatement(PDBPROCESS dbproc, char *szStatement)
{
    RETCODE rc;
    PECBINFO pEcbInfo;

    pEcbInfo = (PECBINFO)dbgetuserdata(dbproc);
    if ( pEcbInfo->bFailed )
        return TRUE;

    rc = SQLExecDirect(dbproc->hstmt, szStatement, SQL_NTS);
    if (rc != SQL_SUCCESS && rc != SQL_SUCCESS_WITH_INFO)
    {
        ODBCError(dbproc);
        if ( pEcbInfo->bDeadlock )
            return FALSE;
        return TRUE;
    }
    return FALSE;
}

```

```

/* FUNCTION: BOOL BindColumn(PDBPROCESS dbproc, SQLUSMALLINT icol,
SQLSMALLINT fCType, SQLPOINTER rgbValue, SQLINTEGER cbValueMax)
*
* PURPOSE: This function wraps the odbc SQLBindCol API so that
error handling and
*
* and deadlock are taken care of in a common
location.
*
* ARGUMENTS: DBRPOCESS dbproc ODBC dbprocess structure
* UWORD icol Column
number of result data, ordered sequentially left to right, starting at 1.
* SWORD fCType The C
data type of the result data. SQL_C_BINARY, SQL_C_BIT, SQL_C_BOOKMARK,
*
* SQL_C_CHAR, SQL_C_DATE, SQL_C_DEFAULT, SQL_C_DOUBLE,
SQL_C_FLOAT, SQL_C_SLONG,
*
* SQL_C_SSHORT, SQL_C_STINYINT, SQL_C_TIME, SQL_C_TIMESTAMP,
SQL_C_ULONG,
*
* SQL_C_USHORT, SQL_C_UTINYINT, SQL_C_DEFAULT
* PTR rgbValue
the
*
* Pointer to storage for the data. If rgbValue is a null pointer,
driver unbinds the column.
*
* SDWORD cbValueMax
Maximum length of the rgbValue buffer. For character data,
rgbValue
*
* must also include space for the null-termination byte.
* RETURNS: none
*
* COMMENTS: none
*/

BOOL BindColumn(PDBPROCESS dbproc, SQLUSMALLINT icol, SQLSMALLINT
fCType, SQLPOINTER rgbValue, SQLINTEGER cbValueMax)
{
    RETCODE rc;
    PECBINFO pEcbInfo;

    pEcbInfo = (PECBINFO)dbgetuserdata(dbproc);
    if ( pEcbInfo->bFailed )
        return TRUE;

    rc = SQLBindCol(dbproc->hstmt, icol, fCType, rgbValue,
cbValueMax, NULL);
    if ( rc == SQL_ERROR )
    {
        ODBCError(dbproc);
        return TRUE;
    }
    return FALSE;
}

/* FUNCTION: BOOL GetResults(PDBPROCESS dbproc)
*
* PURPOSE: This function wraps the odbc SQLFetch API so that
error handling and

```

```

*          and deadlock are taken care of in a common
location.
*
* ARGUMENTS:  DBRPOCESS      dbproc ODBC dbprocess structure
*
* RETURNS:    none
*
* COMMENTS:   none
*
*/

BOOL GetResults(PDBPROCESS dbproc)
{
    PECBINFO pEcbInfo;

    pEcbInfo = (PECBINFO)dbgetuserdata(dbproc);
    if ( pEcbInfo->bFailed )
        return TRUE;

    if ( SQLFetch(dbproc->hstmt) == SQL_ERROR )
    {
        ODBCError(dbproc);
        if ( pEcbInfo->bDeadlock )
            return FALSE;
        return TRUE;
    }
    return FALSE;
}

/* FUNCTION: BOOL MoreResults(DBPROCESS dbproc)
*
* PURPOSE:    This function wraps the odbc SQLMoreResults API so
that error handling and
*          and deadlock are taken care of in a common
location.
*
* ARGUMENTS:  DBRPOCESS      dbproc ODBC dbprocess structure
*
* RETURNS:    none
*
* COMMENTS:   none
*
*/

BOOL MoreResults(PDBPROCESS dbproc)
{
    PECBINFO pEcbInfo;

    pEcbInfo = (PECBINFO)dbgetuserdata(dbproc);
    if ( pEcbInfo->bFailed )
        return TRUE;

    if ( SQLMoreResults(dbproc->hstmt) == SQL_ERROR )
    {
        ODBCError(dbproc);
        if ( pEcbInfo->bDeadlock )
            return FALSE;
        return TRUE;
    }
    return FALSE;
}

```

```

/* FUNCTION: BOOL ReopenConnection(PDBPROCESS dbproc)
*
* PURPOSE:    This function is used with connection ODBC pooling
to reissue the
*          close hdbc connection.
*
* ARGUMENTS:  DBRPOCESS      dbproc ODBC dbprocess structure
*
* RETURNS:    FALSE if successfull
*          TRUE if an error occurs
*
* COMMENTS:   none
*
*/

BOOL ReopenConnection(PDBPROCESS dbproc)
{
    RETCODE rc;
    PECBINFO pEcbInfo;
    int iCount;
    FILE *fp;
    SYSTEMTIME systemTime;

    if ( !bConnectionPooling )
        return FALSE;

    pEcbInfo = (PECBINFO)dbgetuserdata(dbproc);

    iCount = 0;

    /* I don't think this is necessary. ODBC connection
pooling should remember this. - damienl

    if ( SQLSetConnectOption(dbproc->hdbc, SQL_PACKET_SIZE,
4096) == SQL_ERROR )
    {
        ODBCError(dbproc);
        return TRUE;
    }
*/

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

    rc = SQLConnect(dbproc->hdbc, szServer, SQL_NTS, szUser,
SQL_NTS, szPassword, SQL_NTS);
    while (rc != SQL_SUCCESS && rc != SQL_SUCCESS_WITH_INFO)
    {

        Sleep(iConnectDelay); //wait and try again

        iCount++;
        if ( (iCount % 1) == 0)
        {
            fp = fopen(szErrorLogPath, "ab");

            GetLocalTime(&systemTime);

```

```

                fprintf(fp, "** CONNECTION POOL *
%2.2d/%2.2d/%2.2d %2.2d:%2.2d:%2.2d TermId = %d, SyncID = %d, Spin Count =
%d\r\n\r\n",
                systemTime.wYear, systemTime.wMonth,
systemTime.wDay,
                systemTime.wHour,
systemTime.wMinute, systemTime.wSecond,
                pEcbInfo->iTermId, pEcbInfo-
>iSyncId, iCount);

                fclose(fp);
        }

        rc = SQLConnect(dbproc->hdbc, szServer, SQL_NTS,
szUser, SQL_NTS, szPassword, SQL_NTS);
    }

    rc = SQLAllocStmt(dbproc->hdbc, &dbproc->hstmt);
    if (rc == SQL_ERROR)
    {
        ODBCError(dbproc);
        return TRUE;
    }

    rc = SQLExecDirect((dbproc->hstmt, "use tpcc set nocount
on set XACT_ABORT ON", SQL_NTS);
    if (rc != SQL_SUCCESS && rc != SQL_SUCCESS_WITH_INFO)
    {
        ODBCError(dbproc);
        return TRUE;
    }
    SQLFreeStmt((dbproc->hstmt, SQL_CLOSE);

    return FALSE;
}
#endif

```

WEBCNCT.C

```

/* FILE: WEBCNCT.C
*
* Microsoft TPC-C Kit Ver. 3.00.000
*
* Copyright Microsoft, 1996
*
* PURPOSE: Example application that connects a specified
number of clients to
* the WEB client TPCC.DLL.
*
* REQUIREMENTS: Requires VC++ Version 4.2 for the wininet internet
library.
*
* Author: Philip Durr
* philipdu@Microsoft.com
*/

#include <windows.h>

```

```

#include <stdio.h>
#include <stdlib.h>
#include <memory.h>
#include <malloc.h>
#include <conio.h>
#include <wininet.h>

char szCmd1[256];
char szCmd2[256];
char szCmd3[256];

int versionMS = 4;
int versionMM = 0;
int versionLS = 0;

int
void          paktc(void);
int          Login(HINTERNET hInternet, int *pTermId, int
*pSyncId);
int          Logout(HINTERNET hInternet, int iTermId, int
iSyncId);
int          ReadUrlPage(HINTERNET hInternet, char *szUrl, char
**ppBuffer, int *piTotalBytes);
static BOOL  GetParameters(int argc, char *argv[]);
static void  PrintParameters(void);
static void  PrintHeader(void);

int          iTotConnections;
int          iTermId[8192];
int          iSyncId[8192];
char        szInetServer[256];
char        szSqlServer[256];

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

int main(int argc, char *argv[])
{
    HINTERNET hInternet;
    int iRc;
    int i;

    PrintHeader();
    if ( GetParameters(argc, argv) )
    {
        PrintParameters();
        return 0;
    }
}

```

```

    sprintf(szCmd1, "http://%s/tpcc.dll?CMD=Begin&Server=%s&",
szInetServer, szSqlServer);
    sprintf(szCmd2,
"http://%s/tpcc.dll?FORMID=1&w_id=1&d_id=6&CMD=Submit", szInetServer);
    sprintf(szCmd3,
"http://%s/tpcc.dll?STATUSID=0&FORMID=2&TERMID=%d&SYNCID=%d&CMD=..Exit..",
", szInetServer);

    printf("About to attempt %d connections.\n", iTotConnections);
    printf("On Internet Server %s.\n", szInetServer);
    printf("On SQL Server %s.\n", szSqlServer);

    paktc();

    if ( !(hInternet = InternetOpen("Web Test Connections",
INTERNET_OPEN_TYPE_DIRECT, NULL, NULL, 0)) )
    {
        printf("Error cannot open internet connection,
GetLastError() = %d\n",
        GetLastError());
        return 0;
    }

    for(i=0; i<iTotConnections; i++)
    {
        if ( (iRc = Login(hInternet, &iTermId[i], &iSyncId[i])) )
        {
            printf("Error = %d\n", iRc);
            goto main_exit;
        }
        printf("Login() TermID = %d SyncID = %d\n", iTermId[i],
iSyncId[i]);
    }

    printf("All %d Connections made.\n", iTotConnections);
    paktc();

    for(i=0; i<iTotConnections; i++)
    {
        printf("Logout() TermID = %d SyncID = %d\n", iTermId[i],
iSyncId[i]);
        if ( (iRc = Logout(hInternet, iTermId[i], iSyncId[i])) )
        {
            printf("Error = %d\n", iRc);
            goto main_exit;
        }
    }

main_exit:

    InternetCloseHandle(hInternet);

    return 0;
}

/* FUNCTION: void paktc(void)
*
* PURPOSE: This function displays a message on the console and waits
for the user to press a key.
*

```

```

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

void paktc(void)
{
    printf("Press any Key to Continue.");
    getch();
}

/* FUNCTION: int Login(HINTERNET hInternet, int *pTermId, int *pSyncId)
*
* PURPOSE: This function logs the specified number of clients into the
IIS TPCC.DLL client.
*
* ARGUMENTS: HINTERNET hInternet internet handle
return from InternetOpen()
*
terminal id array int *pTermId
*
sync id array int *pSyncId
*
* RETURNS: 0 if successfull or error code if an error occurs.
*
* COMMENTS: The TermID and SyncID returned are provided to the logout
function
*
on exit or logout.
*
*/

int Login(HINTERNET hInternet, int *pTermId, int *pSyncId)
{
    int iRc;
    int iPageSize;
    char *ptr;
    char *pBuffer;
    static char *szKeyTermId = "NAME=\"TERMID\" VALUE=\"";
    static char *szKeySyncId = "NAME=\"SYNCID\" VALUE=\"";

    if ( (iRc = ReadUrlPage(hInternet, szCmd1, &pBuffer, NULL)) )
    {
        printf("ERROR: %s\n", pBuffer);
        return iRc;
    }
    free(pBuffer);

    if ( (iRc = ReadUrlPage(hInternet, szCmd2, &pBuffer, &iPageSize)) )
    {
        printf("ERROR: %s\n", pBuffer);
        return iRc;
    }
    ptr = strstr(pBuffer, szKeyTermId);
    if ( ptr )
    {
        ptr += strlen(szKeyTermId);
        *pTermId = atoi(ptr);
    }
}

```

```

ptr = strstr(pBuffer, szKeySyncId);
if ( ptr )
{
    ptr += strlen(szKeySyncId);
    *pSyncId = atoi(ptr);
    free(pBuffer);
    return 0;
}
}
free(pBuffer);
return -1;
}

/* FUNCTION: int Logout(HINTERNET hInternet, int iTermId, int iSyncId)
 *
 * PURPOSE: This function logs the number of clients logged into the
IIS TPCC.DLL client out.
 *
 * ARGUMENTS: HINTERNET hInternet internet handle return from
InternetOpen()
 *
 * terminal id array, provided from Login API.
 *
 * int iTermId
 *
 * int iSyncId sync
id array, provided from Login API.
 *
 * RETURNS: 0 if successfull or error code if an error occurs.
 *
 * COMMENTS: None
 *
 */

int Logout(HINTERNET hInternet, int iTermId, int iSyncId)
{
    char *pBuffer;
    char szTmp[256];
    int iRc;

    sprintf(szTmp, szCmd3, iTermId, iSyncId);

    iRc = ReadUrlPage(hInternet, szTmp, &pBuffer, NULL);
    free(pBuffer);

    return iRc;
}

/* FUNCTION: int ReadUrlPage(HINTERNET hInternet, char *szUrl, char
**ppBuffer, int *piTotalBytes)
 *
 * PURPOSE: This function logs the number of clients logged into the
IIS TPCC.DLL client out.
 *
 * ARGUMENTS: HINTERNET hInternet internet handle return from
InternetOpen()
 *
 * char *szUrl URL string
passed to the internet service
 *
 * char **ppBuffer pointer to
dynamically allocated buffer containing
 *
 * the result HTML page read from the URL.

```

```

 *
 * int *piTotalBytes size
of the returned HTML page in bytes.
 *
 * RETURNS: 0 if successfull or error code if an error occurs.
 *
 * COMMENTS: The caller is responsible for freeing the returned buffer.
 *
 */

int ReadUrlPage(HINTERNET hInternet, char *szUrl, char **ppBuffer, int
*piTotalBytes)
{
    HINTERNET hUrl;
    BOOL bOk;
    int bytesRead;
    int iBuffSize;
    int iCurrentByte;
    char szTmp[512];

    *ppBuffer = (char *)malloc(512);
    iBuffSize = 512;
    iCurrentByte = 0;
    **ppBuffer = 0;
    if ( piTotalBytes )
        *piTotalBytes = 0;

    hUrl = InternetOpenUrl(hInternet, szUrl, NULL, 0,
INTERNET_FLAG_RELOAD, 0);
    if ( !hUrl )
    {
        return GetLastError();
    }
    do
    {
        bOk = InternetReadFile(hUrl, szTmp, sizeof(szTmp),
&bytesRead);
        if ( !bOk )
            break;
        if ( (iBuffSize - iCurrentByte) <= bytesRead )
        {
            iBuffSize += ((bytesRead/512)+1)*512;
            *ppBuffer = realloc(*ppBuffer, iBuffSize );
            if ( !(*ppBuffer) )
            {
                InternetCloseHandle(hUrl);
                return -2;
            }
        }
        memcpy((*ppBuffer)+iCurrentByte, szTmp, bytesRead);
        iCurrentByte += bytesRead;
    } while( bOk && (bytesRead != 0) );

    InternetCloseHandle(hUrl);

    if ( !bOk )
        return GetLastError();

    if ( piTotalBytes )
        *piTotalBytes = iCurrentByte;

```



```

    return 0;
}

/* 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;

    szSqlServer[0] = 0;
    szInetServer[0] = 0;
    iTotConnections = 100;

    for(i=0; i<argc; i++)
    {
        if ( argv[i][0] == '-' || argv[i][0] == '/' )
        {
            switch(argv[i][1])
            {
                case 'S':
                case 's':
                    strcpy(szSqlServer, argv[i]+2);
                    break;
                case 'I':
                case 'i':
                    strcpy(szInetServer, argv[i]+2);
                    break;
                case 'C':
                case 'c':
                    if ( !iTotConnections =
atoi(argv[i]+2) )
                        iTotConnections = 100;
                    break;
                case '?':
                    return TRUE;
            }
        }
    }

    if ( !szInetServer[0] )
        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)
{
    printf("WEBTEST:\n");
    printf("Parameter
Default\n");
    printf("-----\n");
    printf("-S SQL Server for test run.
\n");
    printf("-I Internet server for test run.(Required)
\n");
    printf("-C Total connections for test run.
\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 executable's banner
information.
 *
 * ARGUMENTS: None
 *
 * RETURNS:      None
 *
 * COMMENTS:  None
 */

static void PrintHeader(void)
{
    printf("*****\n");
    printf("*
*\n");
    printf("* Microsoft SQL Server 6.5
*\n");
    printf("*
*\n");
    printf("* HTML TPC-C BENCHMARK KIT: Web Connect Utility *\n");
    printf("* Version %d.%2d.%3d
*\n", versionMS, versionMM, versionLS);
    printf("*
*\n");
    printf("*****\n");

    return;
}

```


Appendix B - Database Design

Build Scripts

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

/* tpcc database size */
/* 2,525 MB on misc segment (5 x 505 MB) */
/* 38,975 MB on cs segment (5 x 7,795 MB) */
/* 16,450 MB on ol segment (5 x 3,290 MB) */
/* 32,700 MB on syslogs segment (2 x 16,350 MB) */
/* total 90,650 MB (88.52 GB) */

/* total of 32 device fragments */

create database tpcc
    on tpc_misc2=310, tpc_misc3=310, tpc_misc4=310, tpc_misc5=310,
    tpc_misc1=310,
    tpc_misc2=195, tpc_misc3=195, tpc_misc4=195, tpc_misc5=195,
    tpc_misc1=195,
    tpc_cs3= 6883, tpc_cs4= 6883, tpc_cs5= 6883, tpc_cs1= 6883,
    tpc_cs2= 6883,
    tpc_cs3= 912, tpc_cs4= 912, tpc_cs5= 912, tpc_cs1= 912,
    tpc_cs2= 912,
    tpc_ol1= 2668, tpc_ol2= 2668, tpc_ol3= 2668, tpc_ol4= 2668,
    tpc_ol5= 2668,
    tpc_ol1= 622, tpc_ol2= 622, tpc_ol3= 622, tpc_ol4= 622,
    tpc_ol5= 622

    log on tpc_log1=16350, tpc_log2=16350
```

DBOPT1.SQL

```
go

/* TPC-C Benchmark Kit
*/
/* */
/* DBOPT1.SQL
*/
/* */
/* 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

/* TPC-C Benchmark Kit
*/
/* */
/* */
```

DBOPT2.SQL

```
/* TPC-C Benchmark Kit
*/
/* */
/* */
```

```

/* DBOPT2.SQL
*/
/*
*/
/* 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

```

DISKINIT.SQL

```

/* TPC-C Benchmark Kit */
/* */
/* DISKINIT.SQL */
/* */
/* This script is used create devices */

use master
go

/* devices for warehouse, district, item, history, */
/* orders and new-order tables */
/* 510MB = 261,120 pages of 2KB per device */
/* 510MB per device x 5 devices = 2,550MB total (2.49GB) */

disk init name = "tpc_misc1",
    physname = "E:",
    vdevno = 15,
    size = 261120

go

disk init name = "tpc_misc2",
    physname = "F:",
    vdevno = 16,
    size = 261120

go

disk init name = "tpc_misc3",
    physname = "G:",
    vdevno = 17,
    size = 261120

go

disk init name = "tpc_misc4",
    physname = "H:",

```

```

    vdevno = 18,
    size = 261120

go

disk init name = "tpc_misc5",
    physname = "I:",
    vdevno = 19,
    size = 261120

go

/* devices for customer and stock tables */
/* 7,800MB = 3,993,600 pages of 2KB per device */
/* 7,800MB per device x 5 devices = 39,000MB total (38.09GB) */

disk init name = "tpc_cs1",
    physname = "J:",
    vdevno = 20,
    size = 3993600

go

disk init name = "tpc_cs2",
    physname = "K:",
    vdevno = 21,
    size = 3993600

go

disk init name = "tpc_cs3",
    physname = "L:",
    vdevno = 22,
    size = 3993600

go

disk init name = "tpc_cs4",
    physname = "M:",
    vdevno = 23,
    size = 3993600

go

disk init name = "tpc_cs5",
    physname = "N:",
    vdevno = 24,
    size = 3993600

go

/* devices for order-line table */
/* 3,300MB = 1,689,600 pages of 2KB per volume */
/* 3,300MB per device x 5 devices = 16,500MB total (16.11GB) */

disk init name = "tpc_ol1",
    physname = "O:",
    vdevno = 25,
    size = 1689600

go

disk init name = "tpc_ol2",
    physname = "P:",
    vdevno = 26,
    size = 1689600

go

```

```

disk init name = "tpc_ol3",
  physname = "Q:",
  vdevno = 27,
  size = 1689600
go

disk init name = "tpc_ol4",
  physname = "R:",
  vdevno = 28,
  size = 1689600
go

disk init name = "tpc_ol5",
  physname = "S:",
  vdevno = 29,
  size = 1689600
go

/* Log devices */
/* 17,359MB per device = 8,887,808 pages of 2KB */
/* 17,359MB per device times 2 devices = 34,718MB total (33.90GB) */

disk init name = "tpc_log1",
  physname = "T:",
  vdevno = 30,
  size = 8887808
go

disk init name = "tpc_log2",
  physname = "U:",
  vdevno = 31,
  size = 8887808
go

```

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 cs_seg
go
select getdate()
go

```

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 cs_seg
go
select getdate()
go

```

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

```

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

```

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

```

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 ol_seg
go
select getdate()
go

```

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_cl' )
    drop index orders.orders_cl
go

select getdate()
go
create unique clustered index orders_cl on orders(o_w_id, o_d_id, o_id)
    with sorted_data on misc_seg
go
select getdate()
go

```

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_cl' )
    drop index stock.stock_cl
go

select getdate()
go
create unique clustered index stock_cl on stock(s_i_id, s_w_id)
    with sorted_data on cs_seg
go
select getdate()
go

```

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_cl' )
    drop index warehouse.warehouse_cl
go

select getdate()
go
create unique clustered index warehouse_cl on warehouse(w_id)
    with fillfactor=1 on misc_seg
go
select getdate()
go

```

PINTABLE.SQL

```

/* TPC-C Benchmark Kit
*/
/*
*/
/* PINTABLE.SQL
*/
/*
*/
/* 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

```

SEGMENT .SQL

```

/* TPC-C Benchmark Kit
*/
/* SEGMENT.SQL
*/
/* This script is used create segments
*/

use tpcc
go

```

```

exec sp_dropsegment misc_seg
go
exec sp_dropsegment cs_seg
go
exec sp_dropsegment ol_seg
go

/* create segment for miscellaneous tables (warehouse, */
/* district, item, orders, new_order, and history) */

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

/* create segment for customer and stock tables */

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

/* create segment for order-line table */

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

```

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

```

TPCCBCP .SQL

```

/* TPC-C Benchmark Kit
*/
/*
*/
/* TPCBCP.SQL
*/
/*
*/
/* 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

```

TPCCIRL .SQL

```

/* TPC-C Benchmark Kit
*/
/*
*/
/* TPCCIRL.SQL
*/
/*
*/
/* 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

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
                                @w_id
smallint,
                                @d_id
tinyint,
                                @c_id          int,
                                @o_ol_cnt
tinyint,
                                @o_all_local
tinyint,
@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,
                                @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
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

```

```

@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,

```

```

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 s_quantity - @li_qty
                                else s_quantity
                          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

```

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,
                           @datetime,
                           @h_amount,
                           @w_name + "
                           " + @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

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
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

```

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          tinyint,
                             @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 holdlock
            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
    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 holdlock
    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
```

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

```

Loader Source

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 100000
#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;
TPCCLDR_ARGS *aptr, args;

//=====
//
// Function name: main
//
//=====

int main(int argc, char **argv)
{
    DWORD      dwThreadID[MAX_MAIN_THREADS];
    HANDLE     hThread[MAX_MAIN_THREADS];
    FILE       *fLoader;
    char       buffer[255];
    int        main_threads_started;
    RETCODE    retcode;
    LOGINREC  *login;

    printf("\n*****");
    printf("\n*                               *");
    printf("\n* Microsoft SQL Server 6.5       *");
    printf("\n*                               *");
    printf("\n* TPC-C BENCHMARK KIT: Database loader *");
    printf("\n* Version %s                      *",
TPCKIT_VER);
    printf("\n*                               *");
    printf("\n*****\n\n");

    // process command line arguments

    aptr = &args;
    GetArgsLoader(argc, argv, aptr);

```

```

    if (aptr->build_index = 0)
        printf("data load only\n");
    if (aptr->build_index = 1)
        printf("data load and index creation\n");

    // install dblib error handlers

    dbmsghandle((DBMSGHANDLE_PROC)SQLMsgHandler);
    dberrhandle((DBERRHANDLE_PROC)SQLErrHandler);

    // open connections to SQL Server

    OpenConnections();

    // open file for loader results
    fLoader = fopen(aptr->loader_res_file, "a");

    if (fLoader == NULL)
    {
        printf("Error, loader result file open failed.");
        exit(-1);
    }

    // start loading data

    sprintf(buffer, "TPC-C load started for %ld warehouses: ", aptr->num_warehouses);
    if (aptr->build_index = 0)
        strcat(buffer, "data load only\n");
    if (aptr->build_index = 1)
        strcat(buffer, "data load and index creation\n");

    printf("%s", buffer);
    fprintf(fLoader, "%s", buffer);

    main_time_start = (TimeNow() / MILLI);

    // start parallel load threads

    main_threads_completed = 0;
    main_threads_started = 0;

    if ((aptr->table == NULL) || !(strcmp(aptr->table, "item")))
    {
        fprintf(fLoader, "\nStarting loader threads for: item\n");

        hThread[0] = CreateThread(NULL,
                                0,
                                (LPTHREAD_START_ROUTINE) LoadItem,
                                NULL,
                                0,
                                &dwThreadID[0]);

        if (hThread[0] == NULL)
        {
            printf("Error, failed in creating creating thread =
0.\n");
            exit(-1);
        }
    }

```

```

    }
    main_threads_started++;
}
if ((aptr->table == NULL) || !(strcmp(aptr->table,"warehouse")))
{
    fprintf(fLoader, "Starting loader threads for:
warehouse\n");
    hThread[1] = CreateThread(NULL,
                                0,
(LPTHREAD_START_ROUTINE) LoadWarehouse,
                                NULL,
                                0,
&dwThreadID[1]);
    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 loader 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);
    dbcclose(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(0L,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 = %ld\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, NULL, 0, 0, 4);
        bcp_bind(w_dbproc2, (BYTE *) d_street_2,    0,
ADDRESS_LEN, NULL, 0, 0, 5);
        bcp_bind(w_dbproc2, (BYTE *) d_city,        0,
ADDRESS_LEN, NULL, 0, 0, 6);
        bcp_bind(w_dbproc2, (BYTE *) d_state,      0, STATE_LEN,
NULL, 0, 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++)
    {
        for (s_w_id = aptr->starting_warehouse; s_w_id < aptr-
>num_warehouses+1; s_w_id++)
        {
            s_quantity = RandomNumber(10L,100L);
            len = MakeAlphaString(24,24,S_DIST_LEN, s_dist_01);
            len = MakeAlphaString(24,24,S_DIST_LEN, s_dist_02);
            len = MakeAlphaString(24,24,S_DIST_LEN, s_dist_03);
            len = MakeAlphaString(24,24,S_DIST_LEN, s_dist_04);
            len = MakeAlphaString(24,24,S_DIST_LEN, s_dist_05);
            len = MakeAlphaString(24,24,S_DIST_LEN, s_dist_06);
            len = MakeAlphaString(24,24,S_DIST_LEN, s_dist_07);
            len = MakeAlphaString(24,24,S_DIST_LEN, s_dist_08);
            len = MakeAlphaString(24,24,S_DIST_LEN, s_dist_09);
            len = MakeAlphaString(24,24,S_DIST_LEN, s_dist_10);

            len = MakeOriginalAlphaString(26,50, S_DATA_LEN,
s_data,10);

            if (!bcp_sendrow(w_dbproc2))
                printf("Error, Stock() failed calling
bcp_sendrow(). Check
error file.\n");
            stock_rows_loaded++;
            CheckForCommit(w_dbproc2, stock_rows_loaded,
"stock", &time_start);
        }
    }

    bcp_done(w_dbproc2);
    dbclose(w_dbproc2);

    printf("Finished loading stock table.\n");

    if (aptr->build_index == 1)

```

```

        BuildIndex("idxstkcl");
    }
    return;
}

//=====
//
// Function   : LoadCustomer
//
//=====

void LoadCustomer()
{
    LOADER_TIME_STRUCT    customer_time_start;
    LOADER_TIME_STRUCT    history_time_start;
    short                 w_id;
    short                 d_id;
    DWORD                 dwThreadID[MAX_CUSTOMER_THREADS];
    HANDLE                 hThread[MAX_CUSTOMER_THREADS];
    char                   name[20];
    char                   buf[250];

    printf("\nLoading customer and history tables...\n");

    // Seed with unique number
    seed(5);

    // Initialize bulk copy
    sprintf(name, "%s..%s", aptr->database, "customer");
    bcp_init(c_dbproc1, name, NULL, "logs\\customer.err", DB_IN);

    sprintf(name, "%s..%s", aptr->database, "history");
    bcp_init(c_dbproc2, name, NULL, "logs\\history.err", DB_IN);

    customer_rows_loaded = 0;
    history_rows_loaded  = 0;

    CustomerBufInit();

    customer_time_start.time_start = (TimeNow() / MILLI);
    history_time_start.time_start = (TimeNow() / MILLI);

    for (w_id = aptr->starting_warehouse; w_id <= aptr->
>num_warehouses; w_id++)
    {
        for (d_id = 1L; d_id <= DISTRICT_PER_WAREHOUSE; d_id++)
        {
            CustomerBufLoad(d_id, w_id);

            // Start parallel loading threads here...

            customer_threads_completed=0;

            // Start customer table thread

            printf("...Loading customer table for: d_id = %d,
w_id = %d\n", d_id, w_id);

            hThread[0] = CreateThread(NULL,

```

```

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 and c_d_id = 1", LOADER_NURAND_C);
    dbcmd(c_dbproc1, buf);
    dbsqlexec(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 buffer 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 = 10.0;
        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_buf[i].c_id;
        c_d_id = customer_buf[i].c_d_id;
        c_w_id = customer_buf[i].c_w_id;
        h_amount = customer_buf[i].h_amount;
        strcpy(h_data, customer_buf[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 = aptr->starting_warehouse; w_id <= aptr-
>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,
(LPPTHREAD_START_ROUTINE) LoadOrdersTable,
&orders_time_start,
&dwThreadID[0]);
        if (hThread[0] == NULL)
        {
            printf("Error, failed in creating creating
thread = 0.\n");
        }
        // start NewOrder table thread
        printf("...Loading New-Order Table for: d_id = %d,
w_id = %d\n", d_id, w_id);
        hThread[1] = CreateThread(NULL,
                                0,
(LPPTHREAD_START_ROUTINE) LoadNewOrderTable,
&new_order_time_start,
&dwThreadID[1]);
        if (hThread[1] == NULL)
        {
            printf("Error, failed in creating creating
thread = 1.\n");
        }
        // start Order-Line table thread
        printf("...Loading Order-Line Table for: d_id = %d,
w_id = %d\n", d_id, w_id);
        hThread[2] = CreateThread(NULL,
                                0,
(LPPTHREAD_START_ROUTINE) LoadOrderLineTable,

```

```

&order_line_time_start,
                                0,
&dwThreadID[2]);
        if (hThread[2] == NULL)
        {
            printf("Error, failed in creating creating
thread = 2.\n");
        }
        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[ol].ol_amount = 0;
            // Added to insure ol_delivery_d set
properly during load

            CurrentDate(&orders_buf[o_id].o_ol[ol].ol_delivery_d);
        }
        else
        {
            orders_buf[o_id].o_ol[ol].ol_amount =
RandomNumber(1,999999)/100.0;
            // Added to insure ol_delivery_d set
properly during load

            strcpy(orders_buf[o_id].o_ol[ol].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 error file.\n");
    orders_rows_loaded++;
    // CheckForCommit(o_dbproc1, orders_rows_loaded, "ORDERS",
&orders_time_start->time_start);
}

bcp_batch(o_dbproc1);

if ((o_w_id == aptr->num_warehouses) && (o_d_id == 10))
{
    bcp_done(o_dbproc1);
    dbclose(o_dbproc1);

    if (aptr->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);
    }

    bcp_batch(o_dbproc2);

    if ((o_w_id == aptr->num_warehouses) && (o_d_id == 10))
    {
        bcp_done(o_dbproc2);
        dbclose(o_dbproc2);

        if (aptr->build_index == 1)
            BuildIndex("idxnodcl");
    }

    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);
    }

    bcp_batch(o_dbproc3);

    if ((o_w_id == aptr->num_warehouses) && (o_d_id == 10))
    {
        bcp_done(o_dbproc3);
        dbcClose(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 % aptr->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
(%d rows)\n",
                aptr->batch,
                table_name,
                time_diff,
                rows_loaded,
                (float) aptr->batch / (time_diff ? time_diff
: 1L));

        *time_start = time_end;
    }

    return;
}

//=====
//
// Function   : OpenConnections
//
//=====
void OpenConnections()

```

```

{
    RETCODE retcode;
    LOGINREC *login;

    login = dblogin();

    retcode = DBSETLUSER(login, aptr->user);
    if (retcode == FAIL)
    {
        printf("DBSETLUSER failed.\n");
    }
    retcode = DBSETLPWD(login, aptr->password);
    if (retcode == FAIL)
    {
        printf("DBSETLPWD failed.\n");
    }

    retcode = DBSETLPACKET(login, (USHORT) aptr->pack_size);
    if (retcode == FAIL)
    {
        printf("DBSETLPACKET failed.\n");
    }

    printf("DB-Library packet size: %ld\n", aptr->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, aptr->server)) == NULL)
    {
        printf("Error on login 1 to server %s.\n", aptr->server);
        exit(-1);
    }

    if ((w_dbproc1 = dbopen(login, aptr->server)) == NULL)
    {
        printf("Error on login 2 to server %s.\n", aptr->server);
        exit(-1);
    }

    if ((w_dbproc2 = dbopen(login, aptr->server)) == NULL)
    {
        printf("Error on login 3 to server %s.\n", aptr->server);
        exit(-1);
    }

    if ((c_dbproc1 = dbopen(login, aptr->server)) == NULL)
    {
        printf("Error on login 4 to server %s.\n", aptr->server);
        exit(-1);
    }

    if ((c_dbproc2 = dbopen(login, aptr->server)) == NULL)
    {

```

```

        printf("Error on login 5 to server %s.\n", aptr->server);
        exit(-1);
    }

    if ((o_dbproc1 = dbopen(login, aptr->server)) == NULL)
    {
        printf("Error on login 6 to server %s.\n", aptr->server);
        exit(-1);
    }

    if ((o_dbproc2 = dbopen(login, aptr->server)) == NULL)
    {
        printf("Error on login 7 to server %s.\n", aptr->server);
        exit(-1);
    }

    if ((o_dbproc3 = dbopen(login, aptr->server)) == NULL)
    {
        printf("Error on login 8 to server %s.\n", aptr->server);
        exit(-1);
    }
}

//=====
//
// Function name: SQLExceptionHandler
//
//=====

int SQLExceptionHandler(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);
}

```

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     = DEFLDPACKSIZE;
    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 throught */
        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
tables \n");
        printf("    [item|warehouse|customer|orders]\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("[%ld]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_period   = 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("[%ld]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;

            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 =
                    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("[%ld]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 Connections
%d\n", (long) NUM_THREADS);
    printf("-p TDS Packet Size
%d\n", (long) DEFCLPACKSIZE);
    printf("-d Disable Delivery Result Files (no = 0, yes = 1)
%d\n", (long) DISABLE_DELIVERY_RESFILES);
    printf("-s Master Driver Servername
SYNCH_SERVERNAME);                %s\n",

```

```

    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;
    pargs->sqlstat_filename = SQLSTAT_FILENAME;
    pargs->run_id = UNDEF;

    /* check for zero command line args */
    if ( argc == 1 )
        GetArgsSQLStatUsage();

    for ( i = 1; i < argc; ++i)
    {
        if (argv[i][0] != '-' && argv[i][0] != '/')
        {
            printf("\nUnrecognized command");
            GetArgsSQLStatUsage();
            exit(1);
        }

        ptr = argv[i];

        switch (ptr[1])
        {
            case 'S':
                pargs->server = ptr+2;
                break;

            case 'U':
                pargs->user = ptr+2;
                break;

            case 'P':
                pargs->password = ptr+2;
                break;

            case 'A':
                pargs->admin_database = ptr+2;
                break;

            case 'i':
                pargs->run_id = atol(ptr+2);
                break;

            case 'f':
                pargs->sqlstat_filename = ptr+2;
                break;

            default:
                GetArgsSQLStatUsage();
                exit(-1);
        }
    }
}

```

```

        break;
    }
}

/* check for required args */
if (pargs->run_id == UNDEF )
{
    printf("Error, Run ID is required.\n");
    exit(-2);
}

return;
}

//=====
//
// Function name: GetArgsSQLStatUsage
//
//=====

void GetArgsSQLStatUsage()
{
    printf("SQLSTAT:\n\n");
    printf("Parameter
Default\n");
    printf("-----\n");
    printf("-S Server           %s\n",
SERVER);
    printf("-U Username           %s\n",
USER);
    printf("-P Password           %s\n",
PASSWORD);
    printf("-A Admin Database     %s\n",
ADMIN_DATABASE);
    printf("-i Run ID
(required)\n");
    printf("-f Statistics Result file
%s\n", SQLSTAT_FILENAME);

    printf("\nNote:  Command line switches are case sensitive.\n");

    exit(0);
}

RANDOM.C

/* FILE: RANDOM.C
* Microsoft TPC-C Kit Ver. 3.00.000
* Audited 08/23/96, By Francois Raab
*
* 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("[%ld]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("[%ld]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("[%ld]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("[%ld]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("[%ld]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("[%ld]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("[%ld]DBG: RandomNumber between %ld & %ld ==> %ld\n",
          (int) GetCurrentThreadId(), lower, upper,
rand_num);
#endif

    return rand_num;
}
#endif

//=====
// Function   : NURand
//
// Description:
//=====
long NURand(int iConst,
            long x,
            long y,
            long C)
{
    long rand_num;

#ifdef DEBUG
    printf("[%ld]DBG: Entering NURand()...\n", (int)
GetCurrentThreadId());
#endif

    rand_num = (((RandomNumber(0,iConst) | RandomNumber(x,y)) + C) % (y-
x+1))+x;

#ifdef DEBUG
    printf("[%ld]DBG: NURand: num = %d\n", (int) GetCurrentThreadId(),
rand_num);
#endif

    return rand_num;
}

```

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("[%ld]DBG: Entering UtilSleep()\n", (int)
GetCurrentThreadId());
#endif

#ifdef DEBUG
    printf("[%ld]DBG: Sleeping for %ld seconds...\n", (int)
GetCurrentThreadId(), delay);
#endif

    Sleep(delay * 1000);
}

//=====
//
// Function name: UtilSleep
//
//=====

void UtilSleepMs(long delay)
{
#ifdef DEBUG
    printf("[%ld]DBG: Entering UtilSleepMs()\n", (int)
GetCurrentThreadId());
#endif

#ifdef DEBUG
    printf("[%ld]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("[%ld]DBG: Entering UtilPrintNewOrder()\n", (int)
GetCurrentThreadId());
#endif

    EnterCriticalSection(&ConsoleCritSec);

    printf("\n[%04ld]\tNewOrder Transaction\n\n", (int)
GetCurrentThreadId());

    printf("Warehouse: %ld\n"
        "District: %ld\n"
        "Date: %02ld/%02ld/%04ld %02ld:%02ld:%02ld\n\n"
        "Customer Number: %ld\n"
        "Customer Name: %s\n"
        "Customer Credit: %s\n"
        "Cusotmer Discount: %02.2f%%\n\n"
        "Order Number: %ld\n"
        "Warehouse Tax: %02.2f%%\n"
        "District Tax: %02.2f%%\n"
        "Number of Order Lines: %ld\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");
    printf("-----\n");

    for (i=0;i < pNewOrder->o_ol_cnt;i++)
    {
        printf("%04ld %06ld %24s %02ld %03ld %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\n",
        (float) pNewOrder->total_amount);

    printf("Execution Status: %s\n\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\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\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\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: |%50s|\n", data_line_1);
printf("                |%50s|\n", data_line_2);
printf("                |%50s|\n", data_line_3);
printf("                |%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",

```

```

        (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-
>O1OrderStatusData[i].ol_supply_w_id,
          (int) pOrderStatus-
>O1OrderStatusData[i].ol_i_id,
          (int) pOrderStatus-
>O1OrderStatusData[i].ol_delivery_d.month,
          (int) pOrderStatus-
>O1OrderStatusData[i].ol_delivery_d.day,
          (int) pOrderStatus-
>O1OrderStatusData[i].ol_delivery_d.year,
          (int) pOrderStatus-
>O1OrderStatusData[i].ol_quantity,
          (double) pOrderStatus-
>O1OrderStatusData[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\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("[%ld]DBG: Entering UtilPrintStockLevel()\n", (int)
GetCurrentThreadId());
#endif

    EnterCriticalSection(&ConsoleCritSec);

    printf("\n[%04ld]\tStock-Level Transaction\n\n", (int)
GetCurrentThreadId());

    printf("Warehouse: %ld\nDistrict: %ld\n",
          (int) pStockLevel->w_id,
          (int) pStockLevel->d_id);

    printf("Stock Level Threshold: %ld\n\n", (int) pStockLevel-
>thresh_hold);

    printf("Low Stock Count: %ld\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("[%ld]DBG: Entering UtilError()\n", (int)
GetCurrentThreadId());
#endif

    printf("[%ld] %s: %s\n", (int) threadid, header, msg);
}

//=====
//
// Function name: UtilFatalError
//
//=====
void UtilFatalError(long threadid, char * header, char *msg)
{
#ifdef DEBUG
    printf("[%ld]DBG: Entering UtilFatalError()\n", (int)
GetCurrentThreadId());
#endif

    printf("[Thread: %ld]... %s: %s\n", (int) threadid, header, msg);
    exit(-1);
}

//=====
//
// Function name: UtilStrCpy
//
//=====
void UtilStrCpy(char * pDest, char * pSrc, int n)
{
#ifdef DEBUG
    printf("[%ld]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 pad)
{
    COORD    dwWriteCoord = {0, 0};
    DWORD    cCharsWritten;
    LPVOID    dummy;
    int      len, i;

#ifdef DEBUG
    printf("[%ld]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);
    }
#endif
}

```

```

        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
    {
        memcpy(node_to_get, delivery_head, sizeof (struct
delivery_node));

        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);
#endif
}

```

```

        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 - Tunable Parameters

Microsoft Windows NT 4.0 SP1 Configuration Parameters

There were no Windows NT Registry parameters that were changed from their default settings.

The following services were disabled in the Windows NT Control Panel/ Services on the Server:

- Computer Browser
- License Logging Service
- Messenger
- NT LM Security Support Provider
- Plug and Play
- Server
- Spooler
- TCP/ IP Netbios Helper

Microsoft SQL Server 6.50.247 Startup Parameters

```
c:\ mssql\ binn\ sqlservr -c -x -t3502 -t1081
```

Where:

- -c Start SQL Server independently of the Windows NT Service Control Manager
- -x Disables the keeping of CPU time and cache- hit ratio statistics
- -t1081 Allows the index pages a “second” trip through the cache

- -t3502 Prints a message to the SQL Server log at the start and end of each checkpoint

SQL Server Stack Size

The default stack size for Microsoft SQL Server 6.5.SP3 (6.50.247) was changed using the EDITBIN utility. The EDITBIN utility ships with Microsoft Visual C++ V4.0. The command used to change the stack size is:

```
editbin /S: 16384 sqlservr. Exe
```

This command is fully documented as an article in the Microsoft Knowledge Base on the Microsoft Web Site at www.microsoft.com/support.

DBCC GAMINIT

Prior to the execution of the benchmark, the following script was run to proactively populate the Global Allocation Map (GAM) rather than allowing it to be populated on an as- needed basis.

```
Use tpcc
go
dbcc gaminit
go
```

This command is fully documented as an article in the Microsoft Knowledge Base on the Microsoft Web Site at www.microsoft.com/support.

Internet Information Server Registry Parameters

Key Name: SYSTEM\CurrentControlSet\Services\InetInfo\Parameters

Class Name: <NO CLASS>
Last Write Time: 3/26/97 - 1:20 PM

Value 0
Name: BandwidthLevel
Type: REG_DWORD
Data: 0xffffffff

Value 1
Name: ListenBackLog
Type: REG_DWORD
Data: 0x6a4

Value 2
Name: PoolThreadLimit
Type: REG_DWORD
Data: 0x6a4

Value 3
Name: ThreadTimeout
Type: REG_DWORD
Data: 0x15180

Key Name:

SYSTEM\CurrentControlSet\Services\InetInfo\Parameters\Filter

Class Name: <NO CLASS>
Last Write Time: 3/19/97 - 3:50 AM

Value 0
Name: FilterType
Type: REG_DWORD
Data: 0

Value 1
Name: NumDenySites
Type: REG_DWORD
Data: 0

Value 2
Name: NumGrantSites
Type: REG_DWORD
Data: 0

Key Name:

SYSTEM\CurrentControlSet\Services\InetInfo\Parameters\MimeMap

Class Name: <NO CLASS>
Last Write Time: 3/19/97 - 3:50 AM

Value 0
Name: application/envoy, evy, , 5
Type: REG_SZ
Data:

Value 1
Name: application/mac-binhex40, hqx, , 4
Type: REG_SZ

Data:

Value 2
Name: application/msword, doc, , 5
Type: REG_SZ
Data:

Value 3
Name: application/msword, dot, , 5
Type: REG_SZ
Data:

Value 4
Name: application/octet-stream, *, , 5
Type: REG_SZ
Data:

Value 5
Name: application/octet-stream, bin, , 5
Type: REG_SZ
Data:

Value 6
Name: application/octet-stream, exe, , 5
Type: REG_SZ
Data:

Value 7
Name: application/oda, oda, , 5
Type: REG_SZ
Data:

Value 8
Name: application/pdf, pdf, , 5
Type: REG_SZ
Data:

Value 9
Name: application/postscript, ai, , 5
Type: REG_SZ
Data:

Value 10
Name: application/postscript, eps, , 5
Type: REG_SZ
Data:

Value 11
Name: application/postscript, ps, , 5
Type: REG_SZ
Data:

Value 12
Name: application/rtf, rtf, , 5
Type: REG_SZ
Data:

Value 13
Name: application/winhelp, hlp, , 5

Type:	REG_SZ	Name:	application/x-mscardfile,crd,,5
Data:		Type:	REG_SZ
		Data:	
Value 14		Value 26	
Name:	application/x-bcpio,bcpio,,5	Name:	application/x-msclip,clp,,5
Type:	REG_SZ	Type:	REG_SZ
Data:		Data:	
Value 15		Value 27	
Name:	application/x-cpio,cpio,,5	Name:	application/x-msexcel,xla,,5
Type:	REG_SZ	Type:	REG_SZ
Data:		Data:	
Value 16		Value 28	
Name:	application/x-csh,csh,,5	Name:	application/x-msexcel,xlc,,5
Type:	REG_SZ	Type:	REG_SZ
Data:		Data:	
Value 17		Value 29	
Name:	application/x-director,dcr,,5	Name:	application/x-msexcel,xlm,,5
Type:	REG_SZ	Type:	REG_SZ
Data:		Data:	
Value 18		Value 30	
Name:	application/x-director,dir,,5	Name:	application/x-msexcel,xls,,5
Type:	REG_SZ	Type:	REG_SZ
Data:		Data:	
Value 19		Value 31	
Name:	application/x-director,dxr,,5	Name:	application/x-msexcel,xlt,,5
Type:	REG_SZ	Type:	REG_SZ
Data:		Data:	
Value 20		Value 32	
Name:	application/x-dvi,dvi,,5	Name:	application/x-msexcel,xlw,,5
Type:	REG_SZ	Type:	REG_SZ
Data:		Data:	
Value 21		Value 33	
Name:	application/x-gtar,gtar,,9	Name:	application/x-msmediaview,m13,,5
Type:	REG_SZ	Type:	REG_SZ
Data:		Data:	
Value 22		Value 34	
Name:	application/x-hdf,hdf,,5	Name:	application/x-msmediaview,m14,,5
Type:	REG_SZ	Type:	REG_SZ
Data:		Data:	
Value 23		Value 35	
Name:	application/x-latex,latex,,5	Name:	application/x-msmetafile,wmf,,5
Type:	REG_SZ	Type:	REG_SZ
Data:		Data:	
Value 24		Value 36	
Name:	application/x-msaccess,mdb,,5	Name:	application/x-msmoney,mny,,5
Type:	REG_SZ	Type:	REG_SZ
Data:		Data:	
Value 25			

Value 37
 Name: application/x-mspowerpoint,ppt,,5
 Type: REG_SZ
 Data:

Value 38
 Name: application/x-msproject,mpp,,5
 Type: REG_SZ
 Data:

Value 39
 Name: application/x-mspublisher,pub,,5
 Type: REG_SZ
 Data:

Value 40
 Name: application/x-mstterminal,term,,5
 Type: REG_SZ
 Data:

Value 41
 Name: application/x-msworks,wks,,5
 Type: REG_SZ
 Data:

Value 42
 Name: application/x-mswrite,wri,,5
 Type: REG_SZ
 Data:

Value 43
 Name: application/x-netcdf,cdf,,5
 Type: REG_SZ
 Data:

Value 44
 Name: application/x-netcdf,nc,,5
 Type: REG_SZ
 Data:

Value 45
 Name: application/x-perfmon,pma,,5
 Type: REG_SZ
 Data:

Value 46
 Name: application/x-perfmon,pmc,,5
 Type: REG_SZ
 Data:

Value 47
 Name: application/x-perfmon,pml,,5
 Type: REG_SZ
 Data:

Value 48
 Name: application/x-perfmon,pmr,,5
 Type: REG_SZ
 Data:

Value 49
 Name: application/x-perfmon,pmw,,5
 Type: REG_SZ
 Data:

Value 50
 Name: application/x-sh,sh,,5
 Type: REG_SZ
 Data:

Value 51
 Name: application/x-shar,shar,,5
 Type: REG_SZ
 Data:

Value 52
 Name: application/x-sv4cpio,sv4cpio,,5
 Type: REG_SZ
 Data:

Value 53
 Name: application/x-sv4crc,sv4crc,,5
 Type: REG_SZ
 Data:

Value 54
 Name: application/x-tar,tar,,5
 Type: REG_SZ
 Data:

Value 55
 Name: application/x-tcl,tcl,,5
 Type: REG_SZ
 Data:

Value 56
 Name: application/x-tex,tex,,5
 Type: REG_SZ
 Data:

Value 57
 Name: application/x-texinfo,texi,,5
 Type: REG_SZ
 Data:

Value 58
 Name: application/x-texinfo,texinfo,,5
 Type: REG_SZ
 Data:

Value 59
 Name: application/x-troff,roff,,5
 Type: REG_SZ
 Data:

Value 60
 Name: application/x-troff,t,,5
 Type: REG_SZ

Data:

Value 61
 Name: application/x-troff,tr,,5
 Type: REG_SZ
 Data:

Value 62
 Name: application/x-troff-man,man,,5
 Type: REG_SZ
 Data:

Value 63
 Name: application/x-troff-me,me,,5
 Type: REG_SZ
 Data:

Value 64
 Name: application/x-troff-ms,ms,,5
 Type: REG_SZ
 Data:

Value 65
 Name: application/x-ustar,ustar,,5
 Type: REG_SZ
 Data:

Value 66
 Name: application/x-wais-source,src,,7
 Type: REG_SZ
 Data:

Value 67
 Name: application/zip,zip,,9
 Type: REG_SZ
 Data:

Value 68
 Name: audio/basic,au,,<
 Type: REG_SZ
 Data:

Value 69
 Name: audio/basic,snd,,<
 Type: REG_SZ
 Data:

Value 70
 Name: audio/x-aiff,aif,,<
 Type: REG_SZ
 Data:

Value 71
 Name: audio/x-aiff,aifc,,<
 Type: REG_SZ
 Data:

Value 72
 Name: audio/x-aiff,aiff,,<

Type: REG_SZ
 Data:

Value 73
 Name: audio/x-pn-realaudio,ram,,<
 Type: REG_SZ
 Data:

Value 74
 Name: audio/x-wav,wav,,<
 Type: REG_SZ
 Data:

Value 75
 Name: image/bmp,bmp,,:
 Type: REG_SZ
 Data:

Value 76
 Name: image/cis-cod,cod,,5
 Type: REG_SZ
 Data:

Value 77
 Name: image/gif,gif,,g
 Type: REG_SZ
 Data:

Value 78
 Name: image/ief,ief,,:
 Type: REG_SZ
 Data:

Value 79
 Name: image/jpeg,jpe,,:
 Type: REG_SZ
 Data:

Value 80
 Name: image/jpeg,jpeg,,:
 Type: REG_SZ
 Data:

Value 81
 Name: image/jpeg,jpg,,:
 Type: REG_SZ
 Data:

Value 82
 Name: image/tiff,tif,,:
 Type: REG_SZ
 Data:

Value 83
 Name: image/tiff,tiff,,:
 Type: REG_SZ
 Data:

Value 84

Name: image/x-cmu-raster,ras,,:
 Type: REG_SZ
 Data:

Value 85
 Name: image/x-cmx,cmx,,5
 Type: REG_SZ
 Data:

Value 86
 Name: image/x-portable-anymap,pnm,,:
 Type: REG_SZ
 Data:

Value 87
 Name: image/x-portable-bitmap,pbm,,:
 Type: REG_SZ
 Data:

Value 88
 Name: image/x-portable-graymap,pgm,,:
 Type: REG_SZ
 Data:

Value 89
 Name: image/x-portable-pixmap,ppm,,:
 Type: REG_SZ
 Data:

Value 90
 Name: image/x-rgb,rgb,,:
 Type: REG_SZ
 Data:

Value 91
 Name: image/x-xbitmap,xbm,,:
 Type: REG_SZ
 Data:

Value 92
 Name: image/x-ppixmap,xpm,,:
 Type: REG_SZ
 Data:

Value 93
 Name: image/x-xwindowdump,xwd,,:
 Type: REG_SZ
 Data:

Value 94
 Name: text/html,htm,,h
 Type: REG_SZ
 Data:

Value 95
 Name: text/html,html,,h
 Type: REG_SZ
 Data:

Value 96
 Name: text/html,stm,,h
 Type: REG_SZ
 Data:

Value 97
 Name: text/plain,bas,,0
 Type: REG_SZ
 Data:

Value 98
 Name: text/plain,c,,0
 Type: REG_SZ
 Data:

Value 99
 Name: text/plain,h,,0
 Type: REG_SZ
 Data:

Value 100
 Name: text/plain,txt,,0
 Type: REG_SZ
 Data:

Value 101
 Name: text/richtext,rtx,,0
 Type: REG_SZ
 Data:

Value 102
 Name: text/tab-separated-values,tsv,,0
 Type: REG_SZ
 Data:

Value 103
 Name: text/x-setext,etx,,0
 Type: REG_SZ
 Data:

Value 104
 Name: video/mpeg,mpe,,;
 Type: REG_SZ
 Data:

Value 105
 Name: video/mpeg,mpeg,,;
 Type: REG_SZ
 Data:

Value 106
 Name: video/mpeg,mpg,,;
 Type: REG_SZ
 Data:

Value 107
 Name: video/quicktime,mov,,;
 Type: REG_SZ
 Data:

Value 108
 Name: video/quicktime,qt,;,;
 Type: REG_SZ
 Data:

Value 109
 Name: video/x-msvideo,avi,;, <
 Type: REG_SZ
 Data:

Value 110
 Name: video/x-sgi-movie,movie,;, <
 Type: REG_SZ
 Data:

Value 111
 Name: x-world/x-vrml,flr,;, 5
 Type: REG_SZ
 Data:

Value 112
 Name: x-world/x-vrml,wrl,;, 5
 Type: REG_SZ
 Data:

Value 113
 Name: x-world/x-vrml,wrz,;, 5
 Type: REG_SZ
 Data:

Value 114
 Name: x-world/x-vrml,xaf,;, 5
 Type: REG_SZ
 Data:

Value 115
 Name: x-world/x-vrml,xof,;, 5
 Type: REG_SZ
 Data:

World Wide Web Server Registry Parameters

Key Name: SYSTEM\CurrentControlSet\Services\W3SVC
 Class Name: <NO CLASS>
 Last Write Time: 3/19/97 - 3:50 AM

Value 0
 Name: DependOnGroup
 Type: REG_MULTI_SZ
 Data:

Value 1
 Name: DependOnService
 Type: REG_MULTI_SZ
 Data: RPCSS
 NTLMSSP

Value 2
 Name: DisplayName
 Type: REG_SZ
 Data: World Wide Web Publishing Service

Value 3
 Name: ErrorControl
 Type: REG_DWORD
 Data: 0

Value 4
 Name: ImagePath
 Type: REG_EXPAND_SZ
 Data: D:\WINNT\System32\inet_srv\inetinfo.exe

Value 5
 Name: ObjectName
 Type: REG_SZ
 Data: LocalSystem

Value 6
 Name: Start
 Type: REG_DWORD
 Data: 0x2

Value 7
 Name: Type
 Type: REG_DWORD
 Data: 0x20

Key Name: SYSTEM\CurrentControlSet\Services\W3SVC\Enum
 Class Name: <NO CLASS>
 Last Write Time: 3/24/97 - 9:27 AM

Value 0
 Name: 0
 Type: REG_SZ
 Data: Root\LEGACY_W3SVC\0000

Value 1
 Name: Count
 Type: REG_DWORD
 Data: 0x1

Value 2
 Name: NextInstance
 Type: REG_DWORD
 Data: 0x1

Key Name: SYSTEM\CurrentControlSet\Services\W3SVC\Parameters
 Class Name: <NO CLASS>
 Last Write Time: 3/26/97 - 1:20 PM

Value 0
 Name: AcceptExOutstanding
 Type: REG_DWORD
 Data: 0x6a4

Value 1
 Name: AccessDeniedMessage
 Type: REG_SZ
 Data: Error: Access is Denied.

Value 2
 Name: AdminEmail
 Type: REG_SZ
 Data: Admin@corp.com

Value 3
 Name: AdminName
 Type: REG_SZ
 Data: Administrator

Value 4
 Name: AnonymousUserName
 Type: REG_SZ
 Data: IUSR_TPCWE1

Value 5
 Name: Authorization
 Type: REG_DWORD
 Data: 0x5

Value 6
 Name: CacheExtensions
 Type: REG_DWORD
 Data: 0x1

Value 7
 Name: CheckForWAISDB
 Type: REG_DWORD
 Data: 0

Value 8
 Name: ConnectionTimeout
 Type: REG_DWORD
 Data: 0x1c20

Value 9
 Name: DebugFlags
 Type: REG_DWORD
 Data: 0x8

Value 10
 Name: Default Load File
 Type: REG_SZ
 Data: Default.htm

Value 11
 Name: Dir Browse Control
 Type: REG_DWORD
 Data: 0x4000001e

Value 12
 Name: Filter DLLs
 Type: REG_SZ
 Data: D:\WINNT\System32\inet_srv\sspifilt.dll

Value 13
 Name: GlobalExpire
 Type: REG_DWORD
 Data: 0xffffffff

Value 14
 Name: InstallPath
 Type: REG_SZ
 Data: D:\WINNT\System32\inet_srv

Value 15
 Name: LogFileDirectory
 Type: REG_EXPAND_SZ
 Data: %SystemRoot%\System32\LogFiles

Value 16
 Name: LogFileFormat
 Type: REG_DWORD
 Data: 0

Value 17
 Name: LogFilePeriod
 Type: REG_DWORD
 Data: 0x1

Value 18
 Name: LogFileTruncateSize
 Type: REG_DWORD
 Data: 0x1388000

Value 19
 Name: LogSqlDataSource
 Type: REG_SZ
 Data: HTTPLOG

Value 20
 Name: LogSqlPassword
 Type: REG_SZ
 Data: sqllog

Value 21
 Name: LogSqlTableName
 Type: REG_SZ
 Data: Internetlog

Value 22
 Name: LogSqlUserName
 Type: REG_SZ
 Data: InternetAdmin

Value 23
 Name: LogType
 Type: REG_DWORD
 Data: 0

Value 24
 Name: MajorVersion
 Type: REG_DWORD


```

Data:          0x2
Value 25
Name:          MaxConnections
Type:          REG_DWORD
Data:          0x186a0
Value 26
Name:          MinorVersion
Type:          REG_DWORD
Data:          0
Value 27
Name:          NTAAuthenticationProviders
Type:          REG_SZ
Data:          NTLM
Value 28
Name:          ScriptTimeout
Type:          REG_DWORD
Data:          0x384
Value 29
Name:          SecurePort
Type:          REG_DWORD
Data:          0x1bb
Value 30
Name:          ServerComment
Type:          REG_SZ
Data:
Value 31
Name:          ServerSideIncludesEnabled
Type:          REG_DWORD
Data:          0x1
Value 32
Name:          ServerSideIncludesExtension
Type:          REG_SZ
Data:          .stm

Key Name:
SYSTEM\CurrentControlSet\Services\W3SVC\Parameters\
Class Name:    Script Map
Last Write Time: 3/19/97 - 3:50 AM
Value 0
Name:          .idc
Type:          REG_SZ
Data:          D:\WINNT\System32\inet_srv\httpodbc.dll

Key Name:
SYSTEM\CurrentControlSet\Services\W3SVC\Parameters\
Class Name:    Virtual Roots
Last Write Time: 3/19/97 - 1:51 PM

```

```

Value 0
Name:          /,
Type:          REG_SZ
Data:          D:\InetPub\wwwroot,,5
Value 1
Name:          /iisadmin,
Type:          REG_SZ
Data:          D:\WINNT\System32\inet_srv\iisadmin,,1
Value 2
Name:          /Scripts,
Type:          REG_SZ
Data:          D:\InetPub\scripts,,4

Key Name:
SYSTEM\CurrentControlSet\Services\W3SVC\Performance
Class Name:    <NO CLASS>
Last Write Time: 3/19/97 - 3:50 AM
Value 0
Name:          Close
Type:          REG_SZ
Data:          CloseW3PerformanceData
Value 1
Name:          Collect
Type:          REG_SZ
Data:          CollectW3PerformanceData
Value 2
Name:          First Counter
Type:          REG_DWORD
Data:          0x758
Value 3
Name:          First Help
Type:          REG_DWORD
Data:          0x759
Value 4
Name:          Last Counter
Type:          REG_DWORD
Data:          0x790
Value 5
Name:          Last Help
Type:          REG_DWORD
Data:          0x791
Value 6
Name:          Library
Type:          REG_SZ
Data:          w3ctrs.DLL
Value 7
Name:          Open
Type:          REG_SZ
Data:          OpenW3PerformanceData

```

Key Name: SYSTEM\CurrentControlSet\Services\W3SVC\Security
 Class Name: <NO CLASS>
 Last Write Time: 3/19/97 - 3:50 AM

Value 0
 Name: Security
 Type: REG_BINARY
 Data:

```

00000000 01 00 14 80 c0 00 00 00 - cc 00 00 00 14 00 00 00
.....
00000010 34 00 00 00 02 00 20 00 - 01 00 00 00 02 80 18 00
4.....
00000020 ff 01 0f 00 01 01 00 00 - 00 00 00 01 00 00 00 00
.....
00000030 20 02 00 00 02 00 8c 00 - 05 00 00 00 00 00 18 00
.....
00000040 8d 01 02 00 01 01 00 00 - 00 00 00 01 00 00 00 00
.....
00000050 00 00 00 00 00 00 1c 00 - fd 01 02 00 01 02 00 00
.....
00000060 00 00 00 05 20 00 00 00 - 23 02 00 00 00 00 00 00 ....
...
#.....
00000070 00 00 1c 00 ff 01 0f 00 - 01 02 00 00 00 00 00 05
.....
00000080 20 00 00 00 20 02 00 00 - 00 00 00 00 00 00 1c 00 ...
...
00000090 ff 01 0f 00 01 02 00 00 - 00 00 00 05 20 00 00 00
.....
000000a0 25 02 00 00 00 00 00 00 - 00 00 18 00 fd 01 02 00
%.....
000000b0 01 01 00 00 00 00 00 05 - 12 00 00 00 25 02 00 00
.....
....%...
000000c0 01 01 00 00 00 00 00 05 - 12 00 00 00 01 01 00 00
.....
000000d0 00 00 00 05 12 00 00 00 -
.....

```

Key Name: SYSTEM\CurrentControlSet\Services\W3SVC\W3SAMP
 Class Name: <NO CLASS>
 Last Write Time: 3/19/97 - 3:50 AM

Microsoft SQL Server Startup Parameters

Key Name: SOFTWARE\Microsoft\MSSQLServer\MSSQLServer\Parameters
 Class Name: <NO CLASS>
 Last Write Time: 3/27/97 - 8:28 AM

Value 0
 Name: SQLArg0
 Type: REG_SZ
 Data: -dC:\MSSQL\DATA\MASTER.DAT

Value 1
 Name: SQLArg1
 Type: REG_SZ
 Data: -eC:\MSSQL\LOG\ERRORLOG

Value 2
 Name: SQLArg2
 Type: REG_SZ
 Data: -c

Value 3
 Name: SQLArg3
 Type: REG_SZ
 Data: -t1081

Value 4
 Name: SQLArg4
 Type: REG_SZ
 Data: -t3502

Value 5
 Name: SQLArg5
 Type: REG_SZ
 Data: -x

Microsoft SQL Server Configuration Parameters

name	minimum	maximum	config_value	
run_value				
affinity mask	0	2147483647	0	0
allow updates	0	1	1	1
backup buffer size	1	32	1	1
backup threads	0	32	5	5
cursor threshold	-1	2147483647	-1	-
database size	2	10000	2	2
default language	0	9999	0	0
default sortorder id	0	255	50	
fill factor	0	100	0	0
free buffers	20	524288	5000	
hash buckets	4999	265003	265003	
language in cache	3	100	3	3

LE threshold maximum	2	500000	200
200			
LE threshold minimum	2	500000	20
20			
LE threshold percent	1	100	0
locks	5000	2147483647	5000
5000			
LogLRU buffers	0	2147483647	1200
1200			
logwrite sleep (ms)	-1	500	0
max async IO	1	1024	40
40			
max lazywrite IO	1	1024	50
50			
max text repl size	0	2147483647	65536
65536			
max worker threads	10	1024	92
92			
media retention	0	365	0
memory	2800	1048576	890000
890000			
nested triggers	0	1	1
network packet size	512	32767	4096
4096			
open databases	5	32767	10
10			
open objects	100	2147483647	500
500			
priority boost	0	1	0
procedure cache	1	99	2
Protection cache size	1	8192	5
RA cache hit limit	1	255	4
RA cache miss limit	1	255	3
RA delay	0	500	15
15			
RA pre-fetches	1	1000	3
RA slots per thread	1	255	5
RA worker threads	0	255	0
recovery flags	0	1	0
recovery interval	1	32767	32767
32767			
remote access	0	1	0
remote conn timeout	-1	32767	10
10			
remote login timeout	0	2147483647	5
remote proc trans	0	1	0
remote query timeout	0	2147483647	0
remote sites	0	256	0
resource timeout	5	2147483647	10
10			
set working set size	0	1	0
show advanced options	0	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	4

time slice	50	1000	100
100			
user connections	5	32767	6460
6460			
user options	0	4095	0
0			0

Server SCSI RAID Controller Parameters

MegaRAID Adapter-1

Number Of Logical Drives: 3

Logical Drive 1
State : Optimal
RAID TYPE : 1
Write Policy : Write Thru
Read Policy : No Read Ahead
Cache Policy : Direct I/O
Stripe Size : 8K Byte
No. of Stripes : 2
Size : 2047MB
Component Physical Drives :

RANK 0
CHANNEL : 3, TARGET : 0
CHANNEL : 3, TARGET : 1

Logical Drive 2
State : Optimal
RAID TYPE : 0
Write Policy : Write Thru
Read Policy : No Read Ahead
Cache Policy : Direct I/O
Stripe Size : 8K Byte
No. of Stripes : 2
Size : 4094MB
Component Physical Drives :

RANK 0
CHANNEL : 3, TARGET : 2
CHANNEL : 3, TARGET : 4

Logical Drive 3
State : Optimal
RAID TYPE : 0
Write Policy : Write Thru
Read Policy : No Read Ahead
Cache Policy : Direct I/O
Stripe Size : 32K Byte
No. of Stripes : 8
Size : 66384MB
Component Physical Drives :

RANK 0
CHANNEL : 1, TARGET : 0

CHANNEL : 2, TARGET : 0
CHANNEL : 1, TARGET : 1
CHANNEL : 2, TARGET : 1
CHANNEL : 1, TARGET : 2
CHANNEL : 2, TARGET : 2
CHANNEL : 1, TARGET : 3
CHANNEL : 2, TARGET : 3

RANK 1

CHANNEL : 2, TARGET : 8
CHANNEL : 1, TARGET : 8
CHANNEL : 2, TARGET : 9
CHANNEL : 1, TARGET : 9
CHANNEL : 2, TARGET : 10
CHANNEL : 1, TARGET : 10
CHANNEL : 2, TARGET : 11
CHANNEL : 1, TARGET : 11

MegaRAID Adapter-2

Number Of Logical Drives: 1

Logical Drive 1

State : Optimal
RAID TYPE : 0
Write Policy : Write Thru
Read Policy : No Read Ahead
Cache Policy : Direct I/O
Stripe Size : 32K Byte
No. of Stripes : 8
Size : 102648MB
Component Physical Drives :

RANK 0

CHANNEL : 1, TARGET : 0
CHANNEL : 2, TARGET : 0
CHANNEL : 1, TARGET : 1
CHANNEL : 2, TARGET : 1
CHANNEL : 1, TARGET : 2
CHANNEL : 2, TARGET : 2
CHANNEL : 1, TARGET : 3
CHANNEL : 2, TARGET : 3

RANK 1

CHANNEL : 2, TARGET : 8
CHANNEL : 1, TARGET : 8
CHANNEL : 2, TARGET : 9
CHANNEL : 1, TARGET : 9
CHANNEL : 2, TARGET : 10
CHANNEL : 1, TARGET : 10
CHANNEL : 2, TARGET : 11
CHANNEL : 1, TARGET : 11

MegaRAID Adapter-3

Number Of Logical Drives: 1

Logical Drive 1

State : Optimal
RAID TYPE : 0
Write Policy : Write Thru
Read Policy : No Read Ahead
Cache Policy : Direct I/O
Stripe Size : 32K Byte
No. of Stripes : 8
Size : 66384MB
Component Physical Drives :

RANK 0

CHANNEL : 1, TARGET : 0
CHANNEL : 2, TARGET : 0
CHANNEL : 1, TARGET : 1
CHANNEL : 2, TARGET : 1
CHANNEL : 1, TARGET : 2
CHANNEL : 2, TARGET : 2
CHANNEL : 1, TARGET : 3
CHANNEL : 2, TARGET : 3

RANK 1

CHANNEL : 2, TARGET : 8
CHANNEL : 1, TARGET : 8
CHANNEL : 2, TARGET : 9
CHANNEL : 1, TARGET : 9
CHANNEL : 2, TARGET : 10
CHANNEL : 1, TARGET : 10
CHANNEL : 2, TARGET : 11
CHANNEL : 1, TARGET : 11

MegaRAID Adapter-4

Number Of Logical Drives: 2

Logical Drive 1

State : Optimal
RAID TYPE : 0
Write Policy : Write Thru
Read Policy : No Read Ahead
Cache Policy : Direct I/O
Stripe Size : 32K Byte
No. of Stripes : 8
Size : 66384MB
Component Physical Drives :

RANK 0

CHANNEL : 1, TARGET : 0
CHANNEL : 2, TARGET : 0
CHANNEL : 1, TARGET : 1
CHANNEL : 2, TARGET : 1
CHANNEL : 1, TARGET : 2
CHANNEL : 2, TARGET : 2

CHANNEL : 1, TARGET : 3
CHANNEL : 2, TARGET : 3
RANK 1
CHANNEL : 2, TARGET : 8
CHANNEL : 1, TARGET : 8
CHANNEL : 2, TARGET : 9
CHANNEL : 1, TARGET : 9
CHANNEL : 2, TARGET : 10
CHANNEL : 1, TARGET : 10
CHANNEL : 2, TARGET : 11
CHANNEL : 1, TARGET : 11

Logical Drive 2
State : Optimal
RAID TYPE : 5
Write Policy : Write Thru
Read Policy : Read Ahead
Cache Policy : Caching I/O
Stripe Size : 64K Byte
No. of Stripes : 7
Size : 52092MB
Component Physical Drives :

RANK 0
CHANNEL : 1, TARGET : 4
CHANNEL : 2, TARGET : 4
CHANNEL : 1, TARGET : 6
CHANNEL : 2, TARGET : 6
CHANNEL : 1, TARGET : 12
CHANNEL : 2, TARGET : 12
CHANNEL : 1, TARGET : 13MegaRAID Adapter-4

Number Of Logical Drives: 2

Logical Drive 1
State : Optimal
RAID TYPE : 0
Write Policy : Write Thru
Read Policy : No Read Ahead
Cache Policy : Direct I/O
Stripe Size : 32K Byte
No. of Stripes : 8
Size : 66384MB
Component Physical Drives :

RANK 0
CHANNEL : 1, TARGET : 0
CHANNEL : 2, TARGET : 0
CHANNEL : 1, TARGET : 1
CHANNEL : 2, TARGET : 1
CHANNEL : 1, TARGET : 2
CHANNEL : 2, TARGET : 2
CHANNEL : 1, TARGET : 3
CHANNEL : 2, TARGET : 3
RANK 1
CHANNEL : 2, TARGET : 8
CHANNEL : 1, TARGET : 8
CHANNEL : 2, TARGET : 9
CHANNEL : 1, TARGET : 9

CHANNEL : 2, TARGET : 10
CHANNEL : 1, TARGET : 10
CHANNEL : 2, TARGET : 11
CHANNEL : 1, TARGET : 11

Logical Drive 2
State : Optimal
RAID TYPE : 5
Write Policy : Write Thru
Read Policy : Read Ahead
Cache Policy : Caching I/O
Stripe Size : 64K Byte
No. of Stripes : 7
Size : 52092MB
Component Physical Drives :

RANK 0
CHANNEL : 1, TARGET : 4
CHANNEL : 2, TARGET : 4
CHANNEL : 1, TARGET : 6
CHANNEL : 2, TARGET : 6
CHANNEL : 1, TARGET : 12
CHANNEL : 2, TARGET : 12
CHANNEL : 1, TARGET : 13

MegaRAID Adapter-5

Number Of Logical Drives: 1

Logical Drive 1
State : Optimal
RAID TYPE : 0
Write Policy : Write Thru
Read Policy : No Read Ahead
Cache Policy : Direct I/O
Stripe Size : 32K Byte
No. of Stripes : 8
Size : 66384MB
Component Physical Drives :

RANK 0
CHANNEL : 1, TARGET : 0
CHANNEL : 2, TARGET : 0
CHANNEL : 1, TARGET : 1
CHANNEL : 2, TARGET : 1
CHANNEL : 1, TARGET : 2
CHANNEL : 2, TARGET : 2
CHANNEL : 1, TARGET : 3
CHANNEL : 2, TARGET : 3

RANK 1
CHANNEL : 2, TARGET : 8
CHANNEL : 1, TARGET : 8
CHANNEL : 2, TARGET : 9
CHANNEL : 1, TARGET : 9
CHANNEL : 2, TARGET : 10
CHANNEL : 1, TARGET : 10

CHANNEL : 2, TARGET : 11
CHANNEL : 1, TARGET : 11

MegaRAID Adapter-6

Number Of Logical Drives: 2

Logical Drive 1

State : Optimal
RAID TYPE : 1
Write Policy : Write Thru
Read Policy : Read Ahead
Cache Policy : Caching I/O
Stripe Size : 64K Byte
No. of Stripes : 2
Size : 17364MB
Component Physical Drives :

RANK 0

CHANNEL : 1, TARGET : 0
CHANNEL : 2, TARGET : 0

RANK 1

CHANNEL : 2, TARGET : 1
CHANNEL : 1, TARGET : 1

Logical Drive 2

State : Optimal
RAID TYPE : 1
Write Policy : Write Thru
Read Policy : Read Ahead
Cache Policy : Caching I/O
Stripe Size : 64K Byte
No. of Stripes : 2
Size : 17364MB
Component Physical Drives :

RANK 0

CHANNEL : 1, TARGET : 2
CHANNEL : 2, TARGET : 2

RANK 1

CHANNEL : 2, TARGET : 3
CHANNEL : 1, TARGET : 3

Appendix D - RTE Code

Profile used for performance run

Profile: RBS640
File Path: D:\RTE\bc052\RBS640.pro
Version: 1.0.0

Number of Engines: 8

Name: DRIVER1
Description: DA014 to WEB1
Directory: c:\tpcclogs\tpcc1.log
Machine: DA014
Parameter Set: ~Default
Index: 0
Seed: 25738
Configured Users: 800
Pipe Name: DRIVER1234533593
Connect Rate: 0
Start Rate: 0
CLIENT_NURAND: 233

Name: DRIVER2
Description: DA014 to WEB2
Directory: c:\tpcclogs\tpcc2.log
Machine: DA014
Parameter Set: ~Default
Index: 100000
Seed: 25738
Configured Users: 800
Pipe Name: DRIVER2234589468
Connect Rate: 0
Start Rate: 0
CLIENT_NURAND: 233

Name: DRIVER3
Description: DA014 to WEB3
Directory: C:\tpcclogs\tpcc3.log
Machine: DA014
Parameter Set: ~Default
Index: 200000
Seed: 25738
Configured Users: 800
Pipe Name: DRIVER3238800265
Connect Rate: 0
Start Rate: 0
CLIENT_NURAND: 233

Name: DRIVER4
Description: DA014 to WEB4
Directory: c:\tpcclogs\tpcc4.log
Machine: DA014
Parameter Set: ~Default
Index: 300000
Seed: 25738
Configured Users: 800
Pipe Name: DRIVER4238853140
Connect Rate: 0
Start Rate: 0
CLIENT_NURAND: 233

Name: DRIVER5
Description: DA015 to WEB1
Directory: c:\tpcclogs\tpcc5.log
Machine: DA015
Parameter Set: ~Default
Index: 400000
Seed: 25738

Configured Users: 800
Pipe Name: DRIVER5238997343
Connect Rate: 0
Start Rate: 0
CLIENT_NURAND: 233

Name: DRIVER6
Description: DA015 to WEB2
Directory: c:\tpcclogs\tpcc6.log
Machine: DA015
Parameter Set: ~Default
Index: 500000
Seed: 25738
Configured Users: 800
Pipe Name: DRIVER6238930140
Connect Rate: 0
Start Rate: 0
CLIENT_NURAND: 233

Name: DRIVER7
Description: DA015 to WEB3
Directory: c:\tpcclogs\tpcc7.log
Machine: DA015
Parameter Set: ~Default
Index: 600000
Seed: 25738
Configured Users: 800
Pipe Name: DRIVER7238960546
Connect Rate: 0
Start Rate: 0
CLIENT_NURAND: 233

Name: DRIVER8
Description: DA015 to WEB4
Directory: c:\tpcclogs\tpcc8.log
Machine: DA015
Parameter Set: ~Default
Index: 700000
Seed: 25738
Configured Users: 800
Pipe Name: DRIVER8238998328
Connect Rate: 0
Start Rate: 0
CLIENT_NURAND: 233

Number of User groups: 8

Driver Engine: DRIVER1
IIS Server: TPCCWEB1
SQL Server: SFRP6
User: sa
Protocol: Html
w_id Range: 1 - 80
w_id Max Warehouse: 640
Scale: Normal
User Count: 800
District Id: 1
Scale Down: No

Driver Engine: DRIVER2
IIS Server: TPCCWEB2
SQL Server: SFRP6
User: sa
Protocol: Html
w_id Range: 81 - 160
w_id Max Warehouse: 640
Scale: Normal
User Count: 800
District Id: 1
Scale Down: No

Driver Engine: DRIVER3
IIS Server: TPCCWEB3
SQL Server: SFRP6
User: sa
Protocol: Html
w_id Range: 161 - 240
w_id Max Warehouse: 640
Scale: Normal
User Count: 800
District Id: 1
Scale Down: No

Driver Engine: DRIVER4

IIS Server: TPCCWEB4
 SQL Server: SFRP6
 User: sa
 Protocol: Html
 w_id Range: 241 - 320
 w_id Max Warehouse: 640
 Scale: Normal
 User Count: 800
 District id: 1
 Scale Down: No

Driver Engine: DRIVERS5
 IIS Server: TPCCWEB1
 SQL Server: SFRP6
 User: sa
 Protocol: Html
 w_id Range: 321 - 400
 w_id Max Warehouse: 640
 Scale: Normal
 User Count: 800
 District id: 1
 Scale Down: No

Driver Engine: DRIVER6
 IIS Server: TPCCWEB2
 SQL Server: SFRP6
 User: sa
 Protocol: Html
 w_id Range: 401 - 480
 w_id Max Warehouse: 640
 Scale: Normal
 User Count: 800
 District id: 1
 Scale Down: No

Driver Engine: DRIVER7
 IIS Server: TPCCWEB3
 SQL Server: SFRP6
 User: sa
 Protocol: Html
 w_id Range: 481 - 560
 w_id Max Warehouse: 640
 Scale: Normal
 User Count: 800
 District id: 1
 Scale Down: No

Driver Engine: DRIVER8
 IIS Server: TPCCWEB4
 SQL Server: SFRP6
 User: sa
 Protocol: Html
 w_id Range: 561 - 640
 w_id Max Warehouse: 640
 Scale: Normal
 User Count: 800
 District id: 1
 Scale Down: No

Number of Parameter Sets: 1

~Default

Default Parameter Set	Txn Weight	Think Time	Key Time	RT Delay	RT Fence	Menu Delay	
New Order	44.56		12.05	18.01	0.10	5.00	0.10
Payment	43.20		12.05	3.01	0.10	5.00	0.10
Delivery	4.08		5.05	2.01	0.10	5.00	0.10
Stock Level	4.08		5.05	2.01	0.10	20.00	0.10
Order Status	4.08		10.05	2.01	0.10	5.00	0.10

Profile used for reproducibility run

Profile: run2
 File Path: D:\RTE\bc052\run2.pro
 Version: 1.0.0

Number of Engines: 8

Name: DRIVER1
 Description: DA014 to WEB1

Directory: c:\tpcclogs\tpcc1.log
Machine: DA014
Parameter Set: ~Default
Index: 0
Seed: 25738
Configured Users: 800
Pipe Name: DRIVER1234533593
Connect Rate: 50
Start Rate: 50
CLIENT_NURAND: 233

Name: DRIVER2
Description: DA014 to WEB2
Directory: c:\tpcclogs\tpcc2.log
Machine: DA014
Parameter Set: ~Default
Index: 100000
Seed: 25738
Configured Users: 800
Pipe Name: DRIVER2234589468
Connect Rate: 50
Start Rate: 50
CLIENT_NURAND: 233

Name: DRIVER3
Description: DA014 to WEB3
Directory: c:\tpcclogs\tpcc3.log
Machine: DA014
Parameter Set: ~Default
Index: 200000
Seed: 25738
Configured Users: 800
Pipe Name: DRIVER3238800265
Connect Rate: 50
Start Rate: 50
CLIENT_NURAND: 233

Name: DRIVER4
Description: DA014 to WEB4
Directory: c:\tpcclogs\tpcc4.log
Machine: DA014
Parameter Set: ~Default
Index: 300000
Seed: 25738
Configured Users: 800
Pipe Name: DRIVER4238853140
Connect Rate: 50
Start Rate: 50
CLIENT_NURAND: 233

Name: DRIVER5
Description: DA015 to WEB1
Directory: c:\tpcclogs\tpcc5.log
Machine: DA015
Parameter Set: ~Default
Index: 400000
Seed: 25738
Configured Users: 800
Pipe Name: DRIVER5238897343
Connect Rate: 50
Start Rate: 50
CLIENT_NURAND: 233

Name: DRIVER6
Description: DA015 to WEB2
Directory: c:\tpcclogs\tpcc6.log
Machine: DA015
Parameter Set: ~Default
Index: 500000
Seed: 25738
Configured Users: 800
Pipe Name: DRIVER6238930140
Connect Rate: 50
Start Rate: 50
CLIENT_NURAND: 233

Name: DRIVER7
Description: DA015 to WEB3
Directory: c:\tpcclogs\tpcc7.log
Machine: DA015
Parameter Set: ~Default
Index: 600000
Seed: 25738
Configured Users: 800

Pipe Name: DRIVER7238960546
Connect Rate: 50
Start Rate: 50
CLIENT_NURAND: 233

Name: DRIVER8
Description: DA015 to WEB4
Directory: c:\tpcclogs\tpcc8.1og
Machine: DA015
Parameter Set: ~Default
Index: 700000
Seed: 25738
Configured Users: 800
Pipe Name: DRIVER8238998328
Connect Rate: 50
Start Rate: 50
CLIENT_NURAND: 233

Number of User groups: 8

Driver Engine: DRIVER1
IIS Server: TPCCWEB1
SQL Server: SFRP6
User: sa
Protocol: Html
w_id Range: 1 - 80
w_id Max Warehouse: 640
Scale: Normal
User Count: 800
District id: 1
Scale Down: No

Driver Engine: DRIVER2
IIS Server: TPCCWEB2
SQL Server: SFRP6
User: sa
Protocol: Html
w_id Range: 81 - 160
w_id Max Warehouse: 640
Scale: Normal
User Count: 800
District id: 1
Scale Down: No

Driver Engine: DRIVER3
IIS Server: TPCCWEB3
SQL Server: SFRP6
User: sa
Protocol: Html
w_id Range: 161 - 240
w_id Max Warehouse: 640
Scale: Normal
User Count: 800
District id: 1
Scale Down: No

Driver Engine: DRIVER4
IIS Server: TPCCWEB4
SQL Server: SFRP6
User: sa
Protocol: Html
w_id Range: 241 - 320
w_id Max Warehouse: 640
Scale: Normal
User Count: 800
District id: 1
Scale Down: No

Driver Engine: DRIVER5
IIS Server: TPCCWEB1
SQL Server: SFRP6
User: sa
Protocol: Html
w_id Range: 321 - 400
w_id Max Warehouse: 640
Scale: Normal
User Count: 800
District id: 1
Scale Down: No

Driver Engine: DRIVER6
IIS Server: TPCCWEB2
SQL Server: SFRP6
User: sa

Protocol: Html
 w_id Range: 401 - 480
 w_id Max Warehouse: 640
 Scale: Normal
 User Count: 800
 District id: 1
 Scale Down: NO

 Driver Engine: DRIVER7
 IIS Server: TPCCWEB3
 SQL Server: SFRP6
 User: sa
 Protocol: Html
 w_id Range: 481 - 560
 w_id Max Warehouse: 640
 Scale: Normal
 User Count: 800
 District id: 1
 Scale Down: NO

 Driver Engine: DRIVER8
 IIS Server: TPCCWEB4
 SQL Server: SFRP6
 User: sa
 Protocol: Html
 w_id Range: 561 - 640
 w_id Max Warehouse: 640
 Scale: Normal
 User Count: 800
 District id: 1
 Scale Down: NO

Number of Parameter Sets: 7

SCALE75
 Run at about 75 % of full load.

Txn	Think Time	Key Time	RT Delay	RT Fence	Menu Delay	Menu
New Order	44.56	16.00	24.00	0.10	5.00	0.10
Payment	43.20	16.00	4.00	0.10	5.00	0.10
Delivery	4.08	6.70	2.66	0.10	5.00	0.10
Stock Level	4.08	6.70	2.66	0.10	20.00	0.10
Order Status	4.08	13.30	2.66	0.10	5.00	0.10

~Default
 Default Parameter Set

Txn	Think Time	Key Time	RT Delay	RT Fence	Menu Delay	Menu
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
Stock Level	4.10	5.10	2.01	0.10	20.00	0.10
Order Status	4.10	10.10	2.01	0.10	5.00	0.10

SCALE66
 Run at about 66 % of full load.

Txn	Think Time	Key Time	RT Delay	RT Fence	Menu Delay	Menu
New Order	44.56	18.05	36.01	0.10	5.00	0.10
Payment	43.20	18.05	6.01	0.10	5.00	0.10
Delivery	4.08	10.05	4.01	0.10	5.00	0.10
Stock Level	4.08	10.05	4.01	0.10	20.00	0.10
Order Status	4.08	20.05	4.01	0.10	5.00	0.10

SCALE50
 Run at about 50 % of full load.

Txn	Think Time	Key Time	RT Delay	RT Fence	Menu Delay	Menu
New Order	44.56	24.05	36.01	0.10	5.00	0.10
Payment	43.20	24.05	6.01	0.10	5.00	0.10
Delivery	4.08	10.05	4.01	0.10	5.00	0.10
Stock Level	4.08	10.05	4.01	0.10	20.00	0.10
Order Status	4.08	20.05	4.01	0.10	5.00	0.10

SCALE90
 Run at about 90 % of full load.

Txn	Think Time	Key Time	RT Delay	RT Fence	Menu Delay	Menu
New Order	44.56	13.40	20.01	0.10	5.00	0.10
Payment	43.20	13.40	3.50	0.10	5.00	0.10
Delivery	4.08	5.50	2.20	0.10	5.00	0.10
Stock Level	4.08	5.50	2.20	0.10	20.00	0.10
Order Status	4.08	11.05	2.20	0.10	5.00	0.10

SCALE80

Run at about 80 % of full load

Txn	Think Time	Key Time	RT Delay	RT Fence	Menu Delay	0.10
New Order	44.56	15.00	22.50	0.10	5.00	0.10
Payment	43.20	15.00	3.75	0.10	5.00	0.10
Delivery	4.08	6.25	3.75	0.10	5.00	0.10
Stock Level	4.08	6.25	2.50	0.10	20.00	0.10
Order Status	4.08	12.50	2.50	0.10	5.00	0.10

SCALE25

Run at about 25 % of full load.

Txn	Think Time	Key Time	RT Delay	RT Fence	Menu Delay	0.10
New Order	44.56	48.00	72.00	0.10	5.00	0.10
Payment	43.20	48.00	12.00	0.10	5.00	0.10
Delivery	4.08	20.00	8.00	0.10	5.00	0.10
Stock Level	4.08	20.00	8.00	0.10	20.00	0.10
Order Status	4.08	40.00	8.00	0.10	5.00	0.10

Appendix E - Disk Storage

TPC-C 180-Day Disk Space Requirements

Warehouses		650	tpmC	7407.05				
Table	Initial Rows	Data KB	Index KB	Extra 5% KB	Total With 5% KB			
Warehouse	650	1,300	10	66	1,376			
District	6,500	13,000	56	653	13,709			
Customer	19,500,000	13,002,600	1,009,188	700,589	14,712,377			
History	19,500,000	975,002	0	0	975,002			
Order	19,500,000	507,000	3,060	3,270	510,060			
New-Order	5,850,000	65,000	398	3,270	68,668			
Order-Line	195,004,080	10,839,880	70,856	46	10,910,736			
Item	100,000	9,100	46	457	9,603			
Stock	65,000,000	21,671,000	119,736	1,089,537	22,880,273			
Totals KB		47,083,882	1,203,350	1,794,572	50,081,804			
Dbspaces	Count	Size MB	MB Allocated	MB Loaded	+5%			
master, model, tempdb & msdb	5	58	58					
tpc_misc	5	310	2,525	1,541				
tpc_cs	5	195			36,712			
tpc_ol	5	6,883	38,975		10,655			
	5	912						
	5	2668	16,450					
	5	622			48,908			
Total Allocated MB			58,008		48,908			
Dynamic Space MB	12,033	Sum of data for orders, order_line & history						
Static Space	36,875	Sum of data+index+5% - Dynamic Space						
Free Space	9,100	Total allocated space - (Dynamic & Static Spaces)						
Daily Growth	2,194	(Dynamic Space / (W * 62.5)) * tpmC						
Daily Spread	5,809	Free space - 1.5 * Daily growth (zero if negative)						
	0	SQL Server can be configured to eliminate Daily Spread						
180 Day Space MB	431,788	Static Space + 180 * (Daily Growth + Daily Spread)						
180 Day Space GB	421.67							
8 hr log GB	18.64	(need double for mirroring)						
Disk Capacity MB	2047	1,9990 GB	Capacity of 2GB disks					
	4149	4,0518 GB	Capacity of 4GB disks					
	8682	8,4785 GB	Capacity of 9GB disks					
Space Usage	GB Needed	Disks Priced	GB Priced	GB Priced	4GB drives	9GB drives		
180-day space DB	421.67 GB	72	291.73 GB	427.38 GB	4GB drives	9GB drives		
		16	135.66 GB		4GB drives	9GB drives		
8-hr log+mirror	37.28 GB	6	50.87 GB		9GB drives			
OS, file sys	2.00 GB	1	2.00 GB		2GB drives			
Swap file (paging)	2.00 GB	1	2.00 GB		2GB drives			
	462.95 GB	96	482.25 GB					

Appendix F - Third-Party Price Quotations

01/02/97 WED 21:13 FAX 9387329

MICROSOFT RECEP 10 OCT

0003

Microsoft Corporation
One Microsoft Way
Redmond, WA 98052-6399

TEL 206 882 8000
TOLL 184750
FAX 206 926 7225



April 2, 1997

Mr. Cameron Spears
Advanced Logic Research, Inc.
9401 Jeronimo Rd.
Irvine, CA 92718
via FAX: (714) 581-7951

Dear Cameron,

Here is the information you requested regarding pricing of certain Microsoft products:

Microsoft SQL Server 6.5 software, incl 5 CALs	\$1399
Microsoft SQL Server Internet Connector License	\$2999
Microsoft SQL Workstation (includes programmers toolset)	\$499
Windows NT Server 4.0 software, incl 5 CALs (5 copies @ \$899 each)	\$4045
Visual C++ 32-bit edition (subscription)	\$499
3-yr maintenance for above software @ \$309/5/yr	\$10475

This quote is valid for the next 60 days. Please let me know if I can be of any further assistance.

Best regards,

Sid Arora
Product Manager, Microsoft SQL Server
Personal and Business Systems Group

Microsoft Corporation is an equal opportunity employer.



NETLUX
14180 Live Oak Ave, Unit E
Baldwin Park, Ca. 91760
1-800-799-1780
Phone #818-851-5737
Fax #818-851-9837

March 31, 1997

ALR, Inc.
Cameron Spears
9401 Jerouimo Road
Irvine, CA 92618
714-581-6770 Ph
714-581-7951 Fax

Quotation

Quantity	Description	Unit Price	Total
300	24-Port 10Base-T Rack Mount Ethernet Hub	\$251.00	\$75,300.00
	Total		\$75,300.00
	Shipping		TBD

Terms and Conditions:

FOB Origin
5 Year Warranty
Prices good for 60 Days
24-Port will be scheduled for delivery over 1 month

Sincerely,
Martin Parry
NETLUX



NETLUX 1-800-789-1780
14180 Live Oak Ave., Unit E Phone #818-851-9797
Baldwin Park, Ca. 91760 Fax #818-851-9837

March 31, 1997

AI.R, Inc.
Canteron Spears
9401 Jeronimo Road
Irvine, CA 92618
714-581-6770 Ph.
714-581-7951 Fax

Quotation

Quantity	Description	Unit Price	Total
3	8-Port 100Mbps FAST Ethernet Hub	\$595.00	\$1,785.00
		Total	\$1,785.00
		Shipping	TBD

Terms and Conditions:

FOB Origin
5 Year Warranty
Prices good for 60 Days
Shipping From Stock

Sincerely,
Marta Pary
NETLUX

Univ. Corporation
 3199 Pina Road
 Eugene OR 97121

UNISYS

April 2, 1997

Mr. Cameron Speers
 Advanced Logic Research, Inc.
 3401 Jeronimo Road
 Irvine, CA 92718

Dear Mr. Speers:


Here is the pricing you requested for the Unisys Client Hardware listed below:

<u>Qty</u>	<u>Description</u>	<u>Qty</u>	<u>Price</u>	<u>TOTAL</u>
1	Unisys Aquanta EMV/811	Four	\$345	\$1,372
1	Peritho 200mbz Proc/256KB Ca.	Four	1,087	4,348
1	32MB EDO Memory	16	244	3,904
1	SYCA Video Controller	Four	176	704
1	1.5GB IDE Hard Drive	Four	808	3,224
1	12 Speed IDE CD-ROM Drive	Four	357	1,428
1	100Mbit/sec PCI 32 bit ETHERNET	Four	156	624
1	10Mbit/sec PCI 32 bit ETHERNET	8	153	1,224
1	14" Color Monitor	Four	341	1,364
1	104 Key PC Keyboard	Four	40	160
1	Two Button Mouse	Four	30	120
1	Power Cord for Monitor	Four	8	32
*** TOTAL ***				\$ 16,504

Five year, on-site, Monday through Friday maintenance for the systems listed above would be \$818.00 per system, for a total of \$3,272.00.

Thank you for the opportunity to present this quotation. These prices are valid until June 2, 1997.

Sincerely,


 William C. Kehren
 National Account Manager
 (612) 687-2815

WCS/mr
 cc: Glenn Weeks, Unisys, San Jose, CA

04/02/97 09:21:13 FAX 9167329

MICROSOFT RECEP 10 OCT

21002

Microsoft Corporation
One Microsoft Way
Redmond, WA 98052-6399

TEL:206-552-8090
TOLL:1-822-83
FAX:206-936-7339



April 2, 1997

Mr. Cameron Spears
Advanced Logic Research, Inc.
9401 Lexington Rd.
Irvine, CA 92718

via FAX: (714) 881-7951

Dear Cameron,

Microsoft has received your request for permission to disclose the results of TPC-C benchmark tests conducted by ALR with Microsoft SQL Server 6.5 on the following system:

ALR Revolution Quad6 SD512 server, 4 processors, Pentium Pro, 200 MHz

Microsoft hereby grants ALR permission to disclose these results to third parties and acknowledges that ALR has formally requested permission to do so in accordance with the license agreement for Microsoft SQL Server software.

Best regards,

Sid Arora
Product Manager, Microsoft SQL Server
Personal and Business Systems Group

Microsoft Corporation is an equal opportunity employer.