NEC

TPC Benchmark™ C
Full Disclosure Report

NEC Express5800 MH4000

Using Microsoft SQL Server 6.5,
Microsoft Windows NT 4.0 Server

First Edition
Submitted for Review
August 14, 1997

NEC, the Sponsors of this benchmark test, believe that the information in this document is accurate as of the publication date. The information in this document is subject to change without notice. The Sponsor assumes no responsibility for any errors that may appear in this document. The pricing information in this document is believed to accurately reflect the current prices as of the publication date. However, The Sponsor provides no warranty of the pricing information in this document.

Benchmark results are highly dependent upon workload, specific application requirements, and system design and implementation. Relative system performance will vary as a result of these and other factors. Therefore, 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.

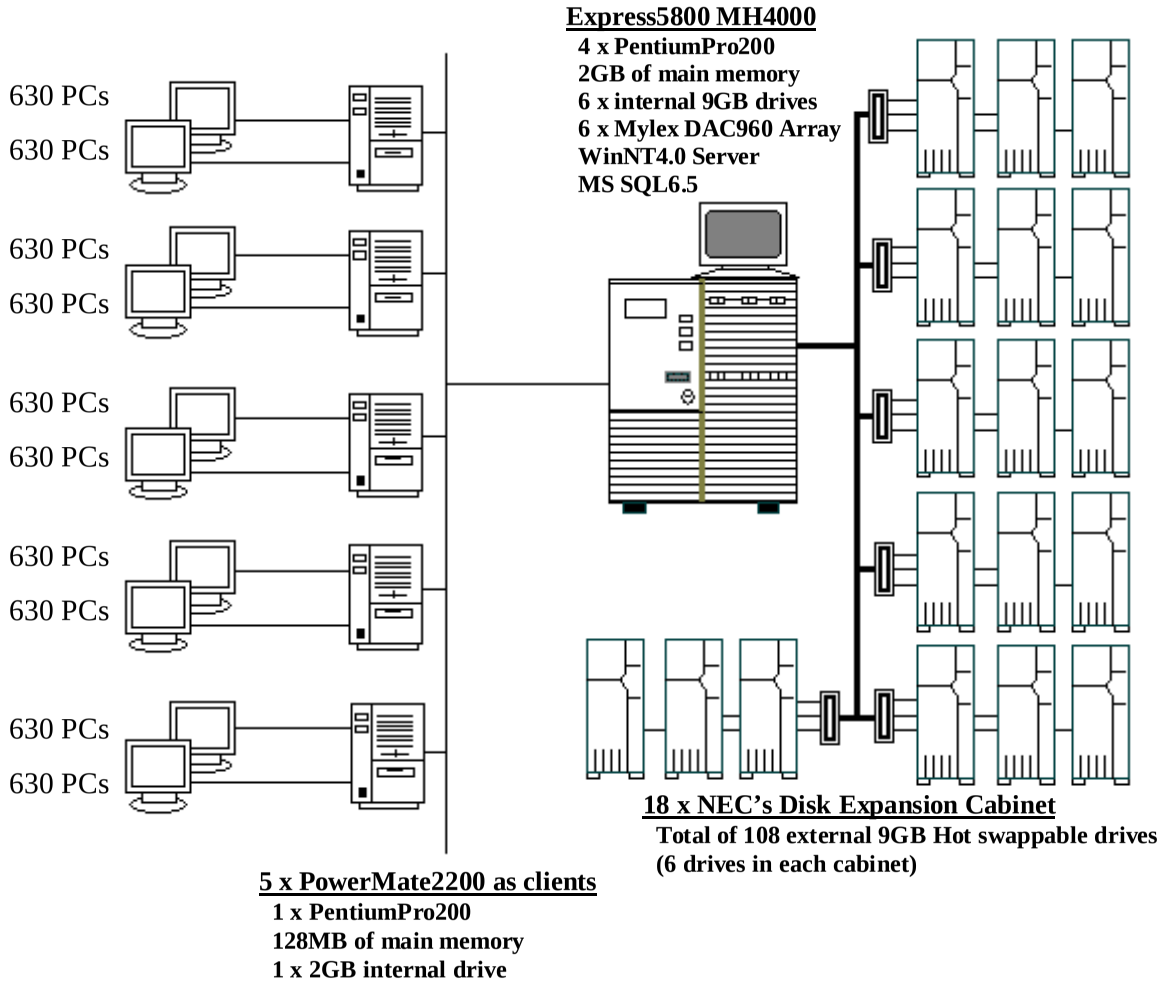
All performance data contained in this report were obtained in a rigorously controlled environment. Results obtained in other operating environments may vary significantly. NEC do not warrant or represent that a user can or will achieve similar performance expressed in transactions per minute (tpmC) or normalized price/performance ($/tpmC). No warranty of system performance or price/performance is expressed or implied in this report.

Printed in USA, 1997

NEC	Express5800 MH4000 C/S	TPC-C REV 3.3 Report Date: 14 August 1997

Total System Cost	TPC-C Throughput	Price/Performance	Availability Date
$ 526,240	7625.63 tpmC	$ 69.00 per tpmC	August 1997

Processors	Database Manager	Operating System	Other Software	Number of Users
4 x PentiumPro 200MHz 512KB cache	Microsoft SQL Server6.5 (SP3)	Microsoft Windows NT 4.0 Server	Microsoft Internet Information Server Microsoft Visual C++	6,300



System Components	Server		Clients	
Processors Cache	4	200MHz Intel Pentium Pro 512 KB	5	200MHz Intel Pentium Pro 256KB
Memory	1	2048MB	5	128MB
Disk Controllers	7	6)Mylex DAC960-PDU 1) Integrated Adaptec Ultra Wide SCSI	5	Integrated IDE
Disk Drives	114	9GB drive (8.47GB usable)	5	2GB drive
Total Storage		965.58GB		10GB
Other	1 1 1	EISA 100 base Ethernet adapter CD-ROM drive Tape Drive	15 5	PCI 100/10base Ethernet adapter CD-ROM drive

NEC	NEC Express 5800 MH4000 C/S				TPC-C REV 3.3 Report Date: 14 August 1997		
Description	**Part Number**	**Third Party**		**Unit Price**	**Quantity**	**Extended Price**	**5-year Maint. Price**
Server Hardware		**Brand**	**Pricing**				
Express 5800 MH4000 1 x PentiumPro/200Mhz/512Kb CPU 1024MB integrated 10/100 NIC CD-ROM, Keyboard, Mouse integrated Ultra-Wide SCSI controller	MH-3430-200GB		1	25,899	1	25,899	6,216
Pentium Pro 200/512 CPU Upgrade Kit	FI-430-34102		1	1,499	3	4,497	0
1024MB ECC (4X256MB DIMM)	FI-410-34305		1	17,599	1	17,599	0
3com FAST Ether Link NIC (+ 2 spares)	3C597-TX	3com	5	233	3	699	0
NEC 15" Multisync Monitor	JC15W1VMA		1	313	1	313	75
4/8GB SCSI Connor 4mm DAT Drive	FI-230-34001		1	749	1	749	0
Disk Subsystem							
Mylex DAC960-PDU 3 channel (+ 2 spares)	DACPU-3-8E-MY4	Mylex	3	2,020	8	16,160	3,878
Disk Expansion cabinet	OP-560-71101		1	5,263	18	94,734	0
9GB Hard Disk Drives	OP-220-76103		1	1,894	114	215,916	0
Internal SCSI cable	K210-16(00)		1	70	6	420	0
External SCSI cable	K208-38(01)		1	47	18	846	0
					Subtotal	377,832	10,169
Server Software							
Microsoft Windows NT Server V. 4.0 w/5 cal		Microsoft	2	809	1	809	0
Microsoft SQL Server6.5 unlimitted user license		Microsoft	2	24,999	1	24,999	10,475
					Subtotal	25,808	10,475
Client Hardware							
PowerMate Pro 2200 PentiumPro200Mhz, 16MB, 2GB HDD, CD-ROM, KB, Mouse	MT-1790-24853C		1	1,894	5	9,470	2,273
32MB EDO memory	OP-410-18005		1	328	20	6,560	0
EtherExpress Pro/100B NIC(20 pack)	PILA8465B	Intel	4	1,135	1	1,135	0
NEC 15" Multisync Monitor	JC15W1VMA		1	313	5	1,565	376
					Subtotal	18,730	2,648
Client Software							
Microsoft Windows NT Server V. 4.0 w/5 cal		Microsoft	2	809	5	4,045	0
Microsoft Visual C++ Professional Edition		Microsoft	2	499	1	499	0
Microsoft SQL Workstation(includes program's toolkit)		Microsoft	2	499	1	499	0
					Subtotal	5,043	0
User Connectivity							
NetLux 8-Port 100BaseT Hub (+ 2 spares)	NX-H8TX	NetLux	6	329	3	987	0
NetLux 24-Port 10Base-T Hub (+ 10% spares)	NX-H24	NetLux	6	251	297	74,547	0
					Subtotal	75,534	0
					TOTAL	502,947	23,293

Notes:

All Microsoft maintenance is covered by the maintenance costs of Microsoft SQL Server

Pricing: 1-NEC 2-Microsoft 3-Mylex 4-Intel 5-3com 6-NetLux

4,5 offer a lifetime warranty free of charge: 6 offers standard with 5-year warranty

Audited by Francois Raab of Information Paradigm, Inc

Five-Year Cost of Ownership: $526,240

tpmC Rating: 7625.63

$ / tpmC: 69.00

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 reflects 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

MQTh, Computed Maximum Qualified Throughput

% throughput difference ,reported & reproducibility runs 0.37

Response Times(in seconds)	**90%**	**Average**	**Maximum**
New-Order	2.02	0.97	7.95
Payment	1.80	0.75	4.59
Order-status	2.47	1.62	8.05
Delivery(interactive portion)	0.28	0.28	0.67
Delivery(deferred portion)	9.55	3.02	48.45
Stock-Level	4.88	3.21	9.68
Menu	0.31	0.20	0.79

Response time delay added for emulated components 0.1

Transaction Mix , in percent of total transaction

New-Order	44.79 %
Payment	43.07 %
Order-status	4.01 %
Delivery	4.01 %
Stock-level	4.11 %

Keying/Think Times (in seconds)	**Min.**		**Average**		**Max**	
New-Order	18.02	0.0	18.03	12.05	18.07	120.71
Payment	3.02	0.0	3.03	12.07	3.07	120.70
Order-Status	2.02	0.0	2.03	10.08	2.04	84.47
Delivery	2.02	0.0	2.03	5.01	2.05	50.70
Stock-Level	2.02	0.0	2.03	5.07	2.06	40.44

Test Duration

Ramp-up time	29 minutes
Measurement interval	30 minutes
Number of checkpoints	1
Checkpoint interval	29.5 minutes
Number of transactions (all types) completed in measurement interval	510,720

Abstract

This report documents the compliance of NEC Corporation's TPC Benchmark™ C tests on the NEC Express 5800/MH4000 client/server system with version 3.3 of the TPC Benchmark C Standard Specification. 5 Clients (NEC PowerMate Pro2200) were used as the front-end clients.

Two standard metrics, transaction-per-minute-C (tpmC) and price per tpmC ($/tpmC) are reported, in accordance with the TPC Benchmark™ C Standard. The independent auditor's report by Francois Raab appears at the end of this report.

TPC Benchmark™ C Metrics

The standard TPC Benchmark C metrics, tpmC (transactions per minute), price per tpmC (five year capital cost per measured tpmC), and the availability date are reported as:

7625.63 tpmC
$69.00 per tpmC
All hardware components are currently available.

Standard and Executive Summary Statements

The following pages contain executive summary of results for this benchmark.

Auditor

The benchmark configuration, environment and methodology were audited by Francois Raab of Information Paradigm,Inc. to verify compliance with the relevant TPC specifications.

Preface

The TPC Benchmark™ C was developed by the Transaction Processing Performance Council (TPC). The TPC was founded to define transaction-processing benchmarks and to disseminate objective, verifiable performance data to the industry. This full disclosure report is based on the TPC Benchmark™ C Standard Specifications Version 3.3 released April 8, 1997.

TPC Benchmark™ C Overview

The TPC describes this benchmark in Clause 0.1 of the specifications as follows:

TPC Benchmark™ C is an On Line Transaction Processing (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 of data access and update

The performance metric reported by TPC-C is a "business throughput" measurement that measures 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.

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-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.

Document Structure

This TPC Benchmark™ C Full Disclosure Report is organized as follows:

- The main body of the document lists each item in Clause 8 of the TPC-C Standard and explains how each requirement is satisfied.
- Appendix A contains the source code of the TPC-C application code used to implement the TPC-C transactions.
- Appendix B contains the database definition and population code used in the tests.
- Appendix C contains the Remote Terminal Emulator (RTE) parameters used to generate and record transactions.
- Appendix D contains the tunable parameters used in the TPC-C tests.
- Appendix E contains space calculation table.
- Appendix F contains the independent auditor's report on the compliance of this disclosure with the benchmark specifications.
- Appendix G contains third-party price quotations.

TPC Benchmark™ C Full Disclosure

The TPC Benchmark™ C Standard Specification requires test sponsors to publish, and make available to the public, a full disclosure report for the results to be considered compliant with the Standard. The required contents of the full disclosure report are specified in Clause 8. This report is intended to satisfy the Standard's requirement for full disclosure. It documents the compliance of the benchmark tests with each item listed in Clause 8 of the TPC Benchmark™ C Standard Specification.

In the Standard Specification, the main headings in Clause 8 are keyed to the other clauses. The headings in this report use the same sequence, so that they correspond to the titles or subjects referred to in Clause 8.

Each section in this report begins with the text of the corresponding item from Clause 8 of the Standard Specification, printed in italic type. The plain text that follows explains how the tests comply with the TPC Benchmark™ C requirement. In sections where Clause 8 requires extensive listings, the section refers to the appropriate appendix at the end of this report.

General Items

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 for TPC-C standard specification. 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 sections in this report correspond with that of the TPC-C standard specification.

Summary Statement

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

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

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,*
- *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, and percentage of transaction mix for each transaction type.*

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

Application Program

The application program (as defined in 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 application source codes used in the TPC-C benchmark.

Sponsor

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

NEC Corporation sponsored this benchmark test. Packard Bell NEC has authorized NEC Corp. to publish TPC-C performance and price/performance results for the NEC Epress5800 MH4000. Price quotations contained in Appendix G correspond to the NEC Express5800 MH4000 server.

Parameters and Options

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

- *Database tuning options*
- *Recovery/locking options*
- *Operating system and application configuration parameters*

Appendix D contains the tunable parameters used in the TPC-C tests.

Configuration Diagrams

Provide diagrams of both the measured and priced configurations, accompanied by a description of the differences. This includes, but not limited to:

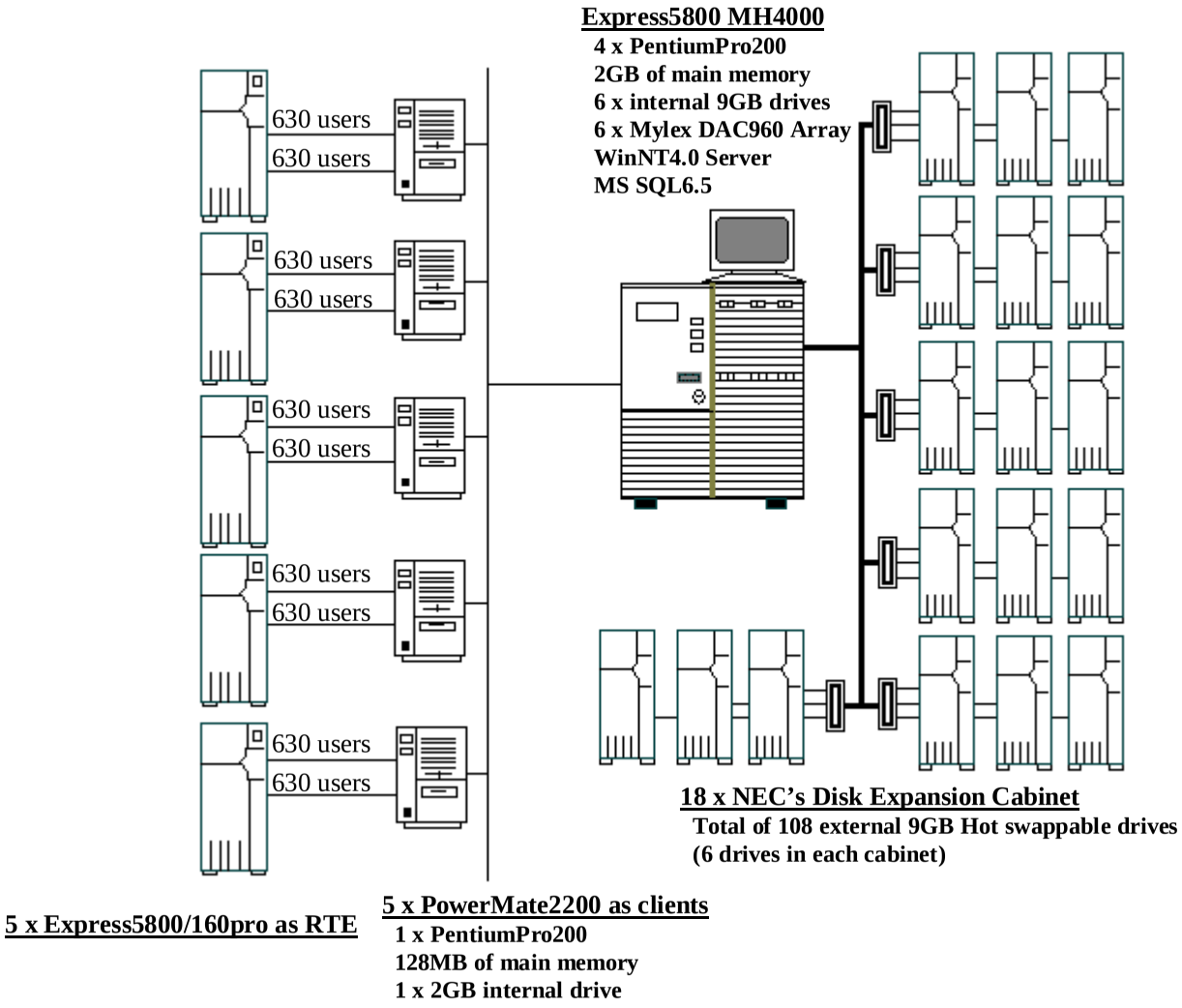
- *Number and type of processors*
- *Size of allocated memory, and any specific mapping/partitioning or memory unique to the test*
- *Number and type of disk drive units (and controllers, if applicable)*
- *Number of channels or bus connections to disk units, including their protocol type*
- *Number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

Figure 1.1 shows the measured configuration diagram.
Figure 1.2 shows the priced configuration diagram.

Measured Configuration

The following figure represents the measured configuration. The benchmark system used a remote terminal emulator(RTE) to initiate transactions and measure response times of transactions, as well as record various data for each transaction.

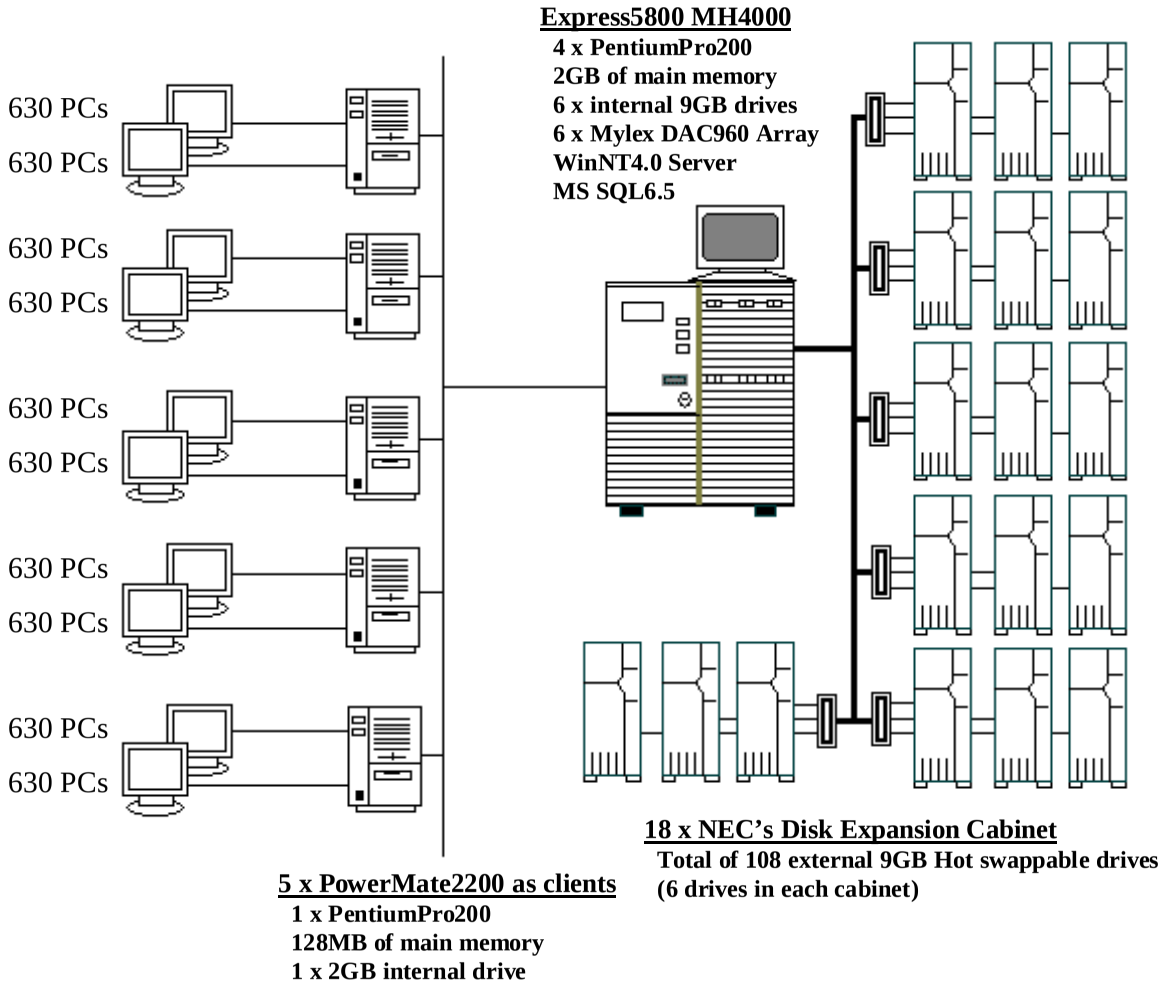
Figure 1.1 Express5800 MH4000, Measured Configuration Diagram



Priced System Configuration

The following figure depicts the priced system, whose cost determines the normalized price per tpmC reported for the test.

Figure1.2: Express5800 MH4000, Priced Configuration Diagram



Clause 1: Logical Database Design and Related Items

Table Definitions

Listing must be provided for all table definition statements and all other statements used to set up the database.

Appendix B contains the code used to define and load the database tables.

Table Organization

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

Appendix B contains the code used to define the physical organization of tables and indices.

Insert and 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 restrictions in the SUT database implementation that precludes inserts beyond the limits defined in Clause 1.4.11 must be disclosed. This includes the maximum number of rows that can be inserted and the minimum key value for these new rows.

All insert and delete functions were fully operational during the entire benchmark.

Disclosure of Partitioning

While there are a few restrictions placed upon horizontal or vertical partitioning of tables and rows in the TPC-C benchmark(see Clause 1.6), any such partitioning must be disclosed.

Partitioning was not used on any table in this benchmark.

Replication of Tables

Replication of tables, if used, must be disclosed.

No tables were replicated in this benchmark test.

Additional and/or Duplicated Attributes in any Table

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

No duplications or additional attributes were used in this benchmark.

Clause 2 : Transaction and Terminal profiles Related Items

Random Number Generation

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

Random numbers were generated internally by the Microsoft BenchCraft RTE program which was already audited independently.

Terminal Input/Output Screen Layout

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

All screen layouts followed the specifications exactly.

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 8.1.3.3.3 must be disclosed and commercially available (including supporting software and maintenance).

The auditor tested each of five transaction types. The auditor verified that all the features specified in Clause 2.2.2.4 were provided.

Presentation Manager or Intelligent Terminal

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

Comment1: *The intent of this clause is to describe any special manipulations performed by a local terminal or workstation to off-load work from the SUT. This includes, but is not limited to : screen presentations, message bundling, and local storage of TPC-C rows.*
Comment2: *This disclosure also requires that all data manipulation functions also be described. Within this disclosure, the purpose of such additional function(s) must be explained.*

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

Transaction Profiles

The percentage of home and remote order-lines in the New-Order transactions must be disclosed.
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 orders entered by New-Order transaction s must be disclosed.
The percentage of home and remote Payment transaction s must be disclosed.
The percentage of Payment and Order-Status transaction s 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.

Table 1 shows the numerical quantities required by Clause 8.1.3.5 through 8.1.3.10.

Transaction Mix

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

Table 1 shows the mix of transaction types seen by the SUT during the reported measurement interval.

Following table summarizes the data required for disclosure in section 3.5 through 3.11.

Table 1 Transaction Statistics

	Statistic	Value
New Order	Home warehouse order lines	**99.0%**
	Remote warehouse order lines	**1.0%**
	Rolled back transactions	**0.99%**
	Average items per order	**10.01%**
Payment	Home warehouse payments	**84.61%**
	Remote warehouse payments	**15.39%**
	Accessed by last name	**59.81%**
Order Status	Accessed by last name	**59.59%**
Delivery	Skipped deliveries	**0**
Transaction Mix	New Order	**44.79%**
	Payment	**43.07%**
	Order status	**4.01%**
	Delivery	**4.01%**
	Stock level	**4.11%**

Queuing Mechanism

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

The client application processes submitted delivery transactions to named pipe delivery server software running on the client machines. There was a single delivery server with multiple execution threads running on each client machine. These delivery servers were responsible for processing deliveries queued to the named pipe and submitting them to the database server. The source code is listed in Appendix A.

Clause 3 : Transaction and System Properties Related Items

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 quotes the specification definition of each of those properties and describes the tests done as specified and monitored by the auditor, to demonstrate compliance.

Atomicity Tests

The system under test must guarantee that the database 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.

Completed Transactions

Perform the Payment for randomly selected warehouse, district and customer (by customer number as specified in Clause 2.5.1.2) and verify that the records in the CUSTOMER, DISTRICT and WAREHOUSE tables have been changed appropriately.

The value of w_ytd, d_ytd, c_balance, c_ytd_payment and c_payment_cnt of a randomly selected warehouse, district, and customer were retrieved. The Payment transaction was executed on the same warehouse, district, and customer. The transaction was committed. The values w_ytd, d_ytd, c_balance, c_ytd_payment, and c_payment_cnt were retrieved again. It was verified that all values had been changed appropriately.

Aborted Transactions

Perform the Payment transaction for randomly selected warehouse, district and customer(by customer number as specified in Clause 2.5.1.2) and substitute a ROLLBACK of the transaction for the COMMIT of the transaction. Verify that records in CUSTOMER, DISTRICT and WAREHOUSE tables have Not been changed.

The value of w_ytd, d_ytd, c_balance, c_ytd_payment and c_payment_cnt of randomly selected warehouse , district, and customer were retrieved. The Payment transaction was executed on the same warehouse, district, and customer. The transaction was rolled back. The values of w_ytd, d_ytd, c_balance, c_ytd_payment, c_payment_cnt were retrieved again. It was verified that none of the values had changed.

Consistency Tests

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

Consistency conditions one through four were tested using a script to issue queries to the database. The results of the queries verified that the database was consistent for all four tests. A run was executed over 10 minutes and included a checkpoint under 1000 users (100 active warehouse) condition . The shell script was executed before and after the run. The result of the same queries verified that the database remained consistent after the run.

Isolation

Sufficient conditions must be enabled at either the system or application level to ensure the required isolation level is obtained.

Isolation tests one through nine were executed using shell scripts to issue queries to the database. Each script included timestamps to demonstrate the concurrency of operations. The results of the queries were captured to files. The captured files were verified to demonstrate the required isolation had been met.
Case A was followed for Isolation Test 7.

Durability

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

- *Permanent irrecoverable failure of any single durable medium containing database, ABTH files/tables, or recovery log data.*
- *Instantaneous interruption (system crash/system hang) in processing which requires system reboot to recover.*
- *Failure of all or part of memory(loss of contents)*

Loss of Memory and Loss of Log

Because the loss of power erases the contents of memory, both of instantaneous interruption and loss of memory were combined into a single test. Also loss of log was combined into the test.

The following steps were performed on a database of 630 warehouses under the full load of 6300 users.

1. A sum of D_NEXT_O_ID of all rows in the district table was taken.
2. 6300 users were logged in to the database and running transactions for 5 minutes.
3. A checkpoint was initiated.
4. One disk drive holding the transaction log was removed causing an NT alert message, but causing no failures.
5. The running continued 2 minutes.
6. the system was powered off.
7. The RTE was allowed to continue running and time out transactions.
8. The RTE was eventually shut down.
9. The system was powered back up, SQL Server was restarted and automatically recovered.
10. A new count of D_NEXT_O_ID was taken.
11. This number was compared with the number of new orders reported by the RTE.

Loss of Data

Loss of data was demonstrated on a 10 Warehouse database for convenience. The standard driving mechanism was used to generate the transaction load of 100 users for the test. To demonstrate recovery from a permanent failure of durable media containing TPC-C tables, the following steps were performed. A fully scaled database would also pass this test.

1. A 10 Warehouse database was built having similar characteristics to the large database.
2. The database was backed up using SQL Server backup facilities.
3. A sum of D_NEXT_O_ID was taken.
4. 100 users were logged in to the database and running transactions.
5. One disk drive in the array was removed causing SQL Server to shut down.
6. SQL Server was restarted and a dump of the transaction log was taken.
7. The 10 Warehouse database was restored from backup.
8. The transaction log was restored and transactions rolled forward.
9. A new count of D_NEXT_O_ID was taken.
10. This number was compared with the number of new orders reported by the RTE.

Clause 4 : Scaling and Database Population Related Items

Initial Cardinality of Tables

The cardinality (e.g. 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, the cardinality of the WAREHOUSE table as initially configured and the number of rows deleted must be disclosed.

The TPC-C database was originally built with 650 warehouses. 20 rows were deleted from the warehouse table prior to the run per Clause 4.2.2 of the TPC specification.

Table 2 Number of Rows for Server

Table	Cardinality as benchmarked
Warehouse	630
Distinct	6,500
Customer	19,500,000
History	19,500,000
Orders	19,500,000
New Order	5,850,000
Order Line	195,004,080
Stock	65,000,000
Item	10,000
Deleted Warehouse Rows	20

Distribution of Tables and Logs

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

Table 3 depicts the distribution of the database tables over the disks of the tested system. Figure 1.1, 1.2 shows the disk configuration for measured and priced system.

Table 3 : Disk Usage in tested System

Controller	Partition	Size(MB)	Use
DAC960-PDU #1	C:	4095	OS
	H:	7060	cs_seg
DAC960-PDU #2	I:	7060	cs_seg
DAC960-PDU #3	J:	7060	cs_seg
DAC960-PDU #4	L:	4385	cs_seg
	S:	384	misc_seg
	T:	384	misc_seg
	U:	384	misc_seg
	V:	384	misc_seg
	W:	384	misc_seg
DAC960-PDU #5	M:	5562	cs_seg
	N:	2651	ol_seg
	O:	2651	ol_seg
	P:	2651	ol_seg
	Q:	2651	ol_seg
	R:	4322	ol_seg
DAC960-PDU #6	K:	7060	cs_seg
	Y:	51199	Backup
Adaptec	E:	8096	log_seg
	E: (mirror)	8096	log_seg
	F:	8096	log_seg
	F: (mirror)	8096	log_seg
	G:	8096	log_seg
	G: (mirror)	8096	log_seg

Type of Database

A statement must be provided that describes:
1)The data model implemented by DBMS used (e.g. relational, network, hierarchical).
2)The database interface (e.g. embedded, call level) and access language (e.g. SQL, PL/1, COBOL read/write used to implement the TPC-C transaction. If more than one interface/access language is used to implement TPC-C, each interface/access language must be described and a list of which interface/access language is used with which transaction type must be disclosed.

Microsoft SQL Server 6.5, a relational database, was used in this benchmark. SQL Server stored procedures were used and invoked through DB-Library function calls embedded in C code.

Database Mapping

The mapping of database partitions/replications must be explicitly described.
No partitioning or replication was used.

180-Days Space

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

The detail of 180-day space calculation is shown in Appendix E.
To calculate the space required to sustain the database log for 8 hours of growth at steady state, the following steps were followed:

1. The free space on the log file was queried using *DBCC sqlperf(logspace).*
2. Transactions were run against the database with a full load of users.
3. The free space was again queried using *DBCC sqlperf(logspace).*
4. The space used was calculated as the difference between the first and second query.
5. The number of NEW-ORDERS was verified from an RTE report covering the entire run.
6. The space used was divided by the number of NEW-ORDERS giving a spaceused per NEW-ORDER transaction.
7. The space used per transaction was multiplied by the measured tpmC rate times 480 minutes.

The results of the above steps yielded a requirement of 19.64 GB to sustain the log for 8 hours.
Space available on the transaction log volume was 50.84 GB (including mirror), indicating that enough storage was configured to sustain 8 hours of growth.

The same methodology was used to compute growth requirements for dynamic tables Order, Order-Line and History.

Clause 5 : Performance Metrics and Response Time Related Items

Throughput

Measured tpmC must be reported

Table 4 : Measured tpmC

7625.63tpmC	$69.00 per tpmC

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: Response Times (in seconds)

Type	Average	Maximum	90th %
New-Order	0.97	7.95	2.02
Payment	0.75	4.59	1.80
Order-Status	1.62	8.05	2.47
Interactive Delivery	0.28	0.67	0.28
Deferred Delivery	3.02	48.45	9.55
Stock Level	3.21	9.68	4.88
Menu	0.20	0.79	0.31

Keying and Think Times

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

Table 6: Keying Times

Type	Minimum	Average	Maximum
New-Order	18.02	18.03	18.07
Payment	3.02	3.03	3.07
Order-Status	2.02	2.03	2.04
Interactive Delivery	2.02	2.03	2.05
Stock Level	2.02	2.03	2.06

Table 7: Think Times

Type	Minimum	Average	Maximum
New-Order	0.00	12.05	120.71
Payment	0.00	12.07	120.70
Order-Status	0.00	10.08	84.47
Interactive Delivery	0.00	5.01	50.70
Stock Level	0.00	5.07	40.44

Response Time Frequency Distribution Curves and Other Graphs

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

Figure 2.1: New Order Response Time Distribution

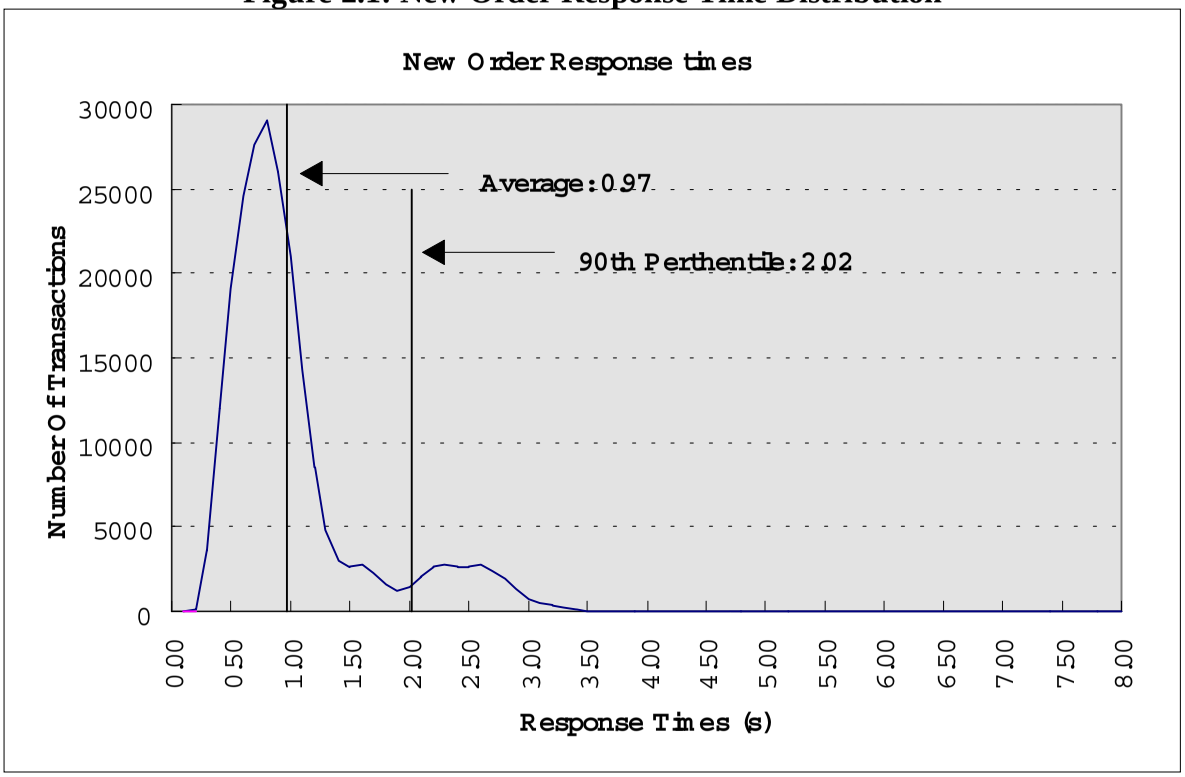


Figure 2.2: Payment Response Time Distribution

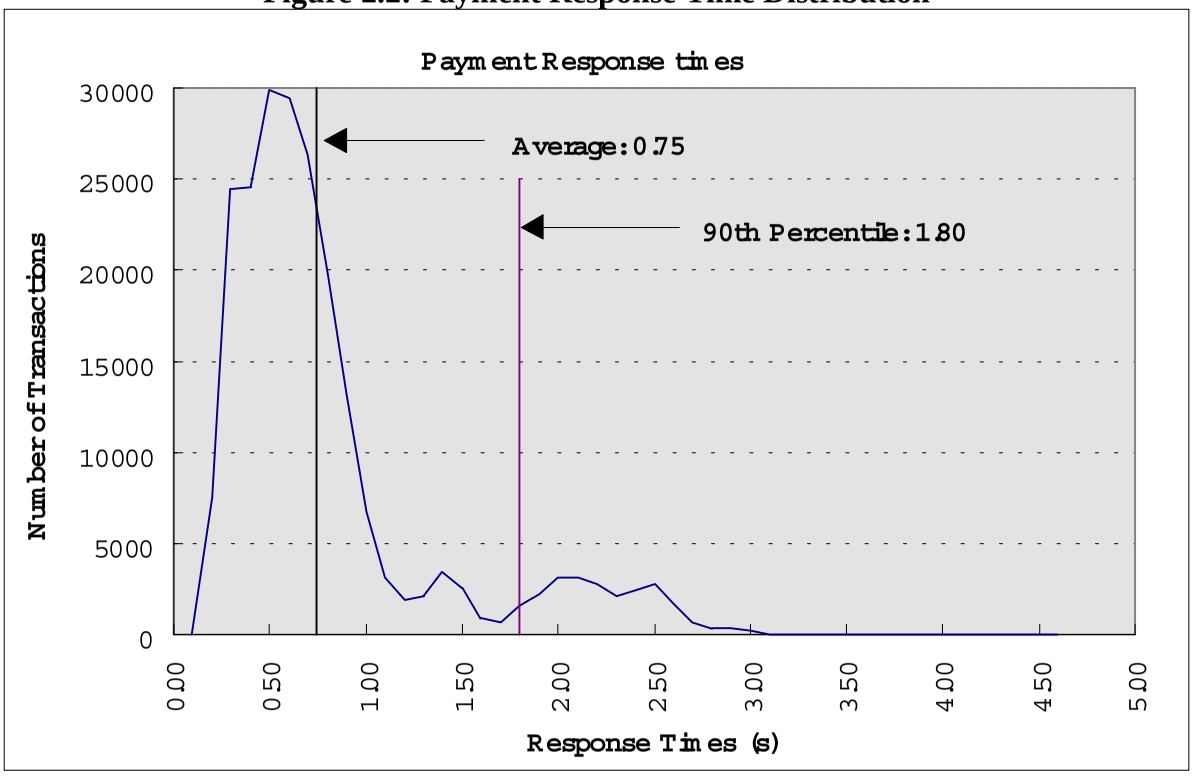


Figure 2.3: Order Status Response Time Distribution

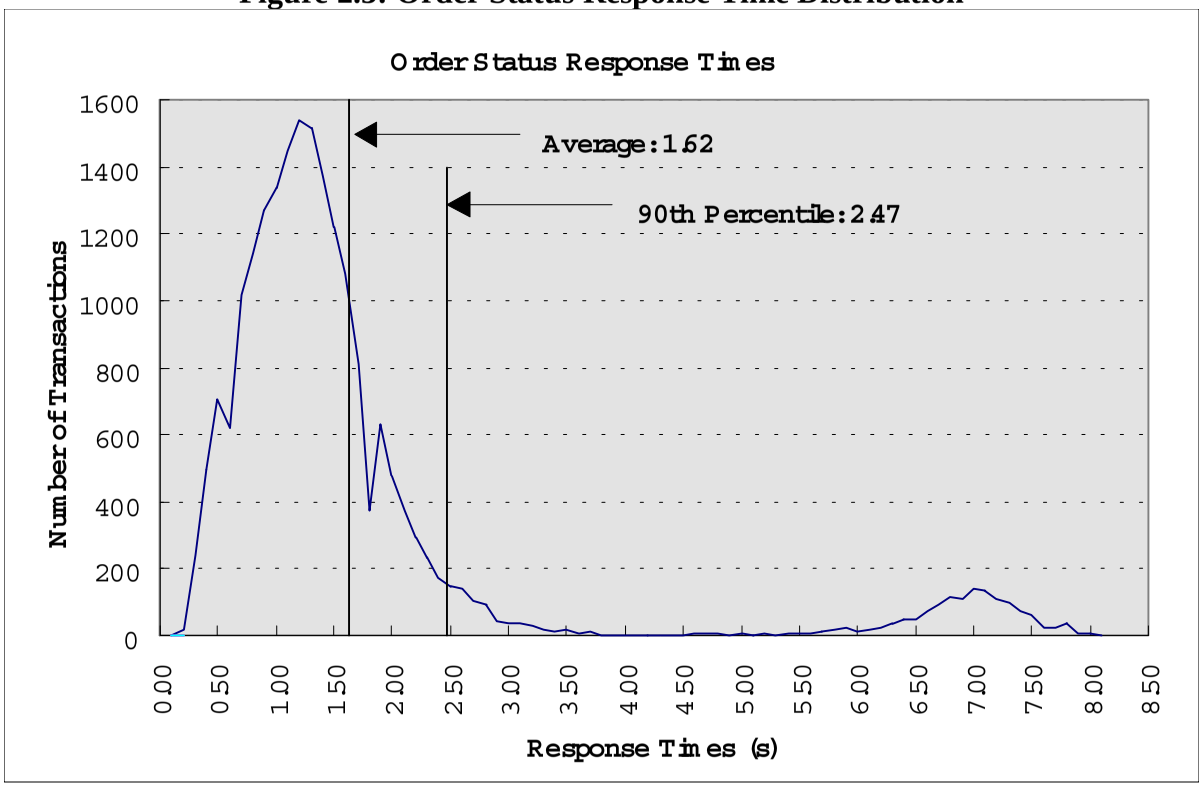


Figure 2.4: Delivery Response Time Distribution

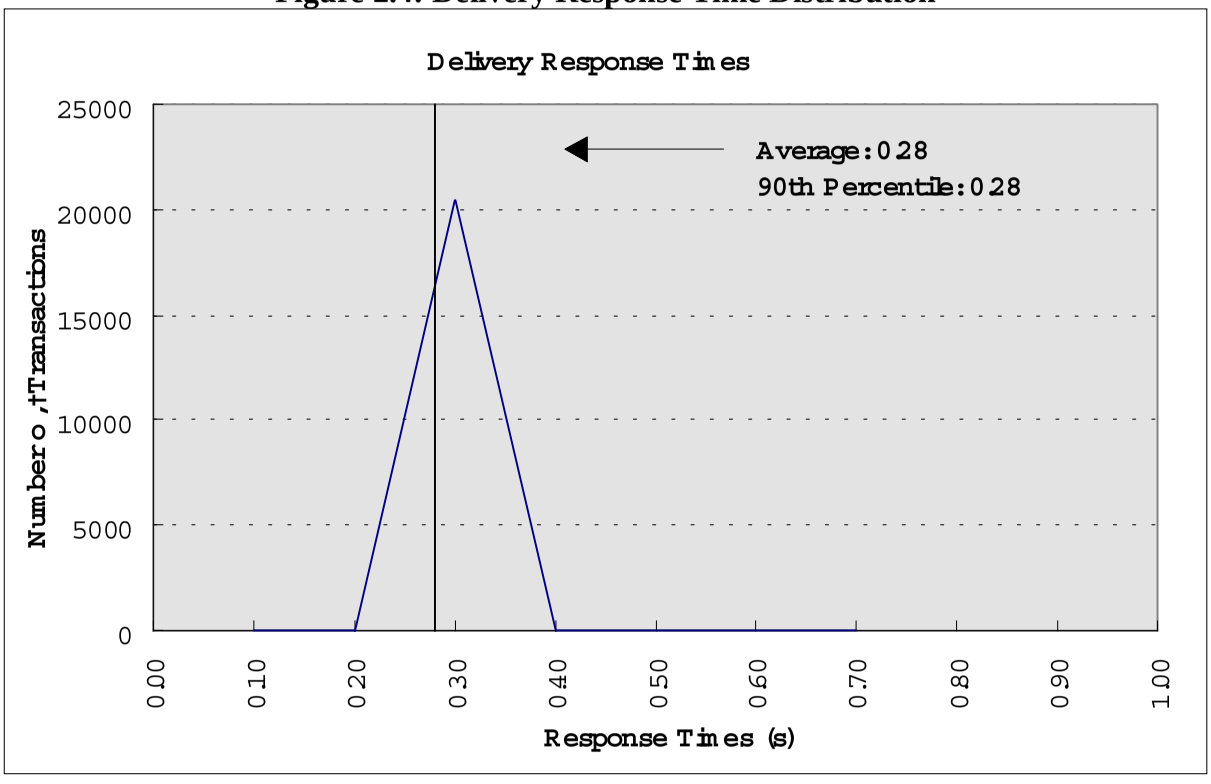
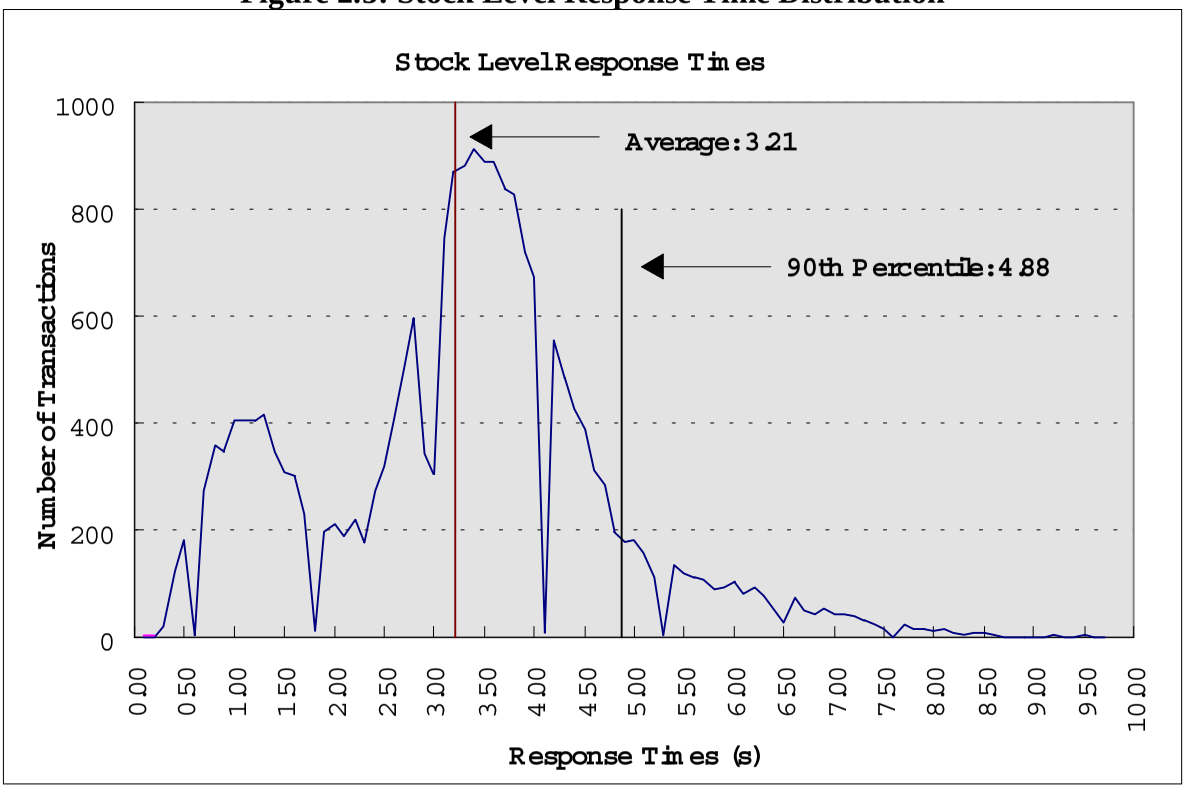


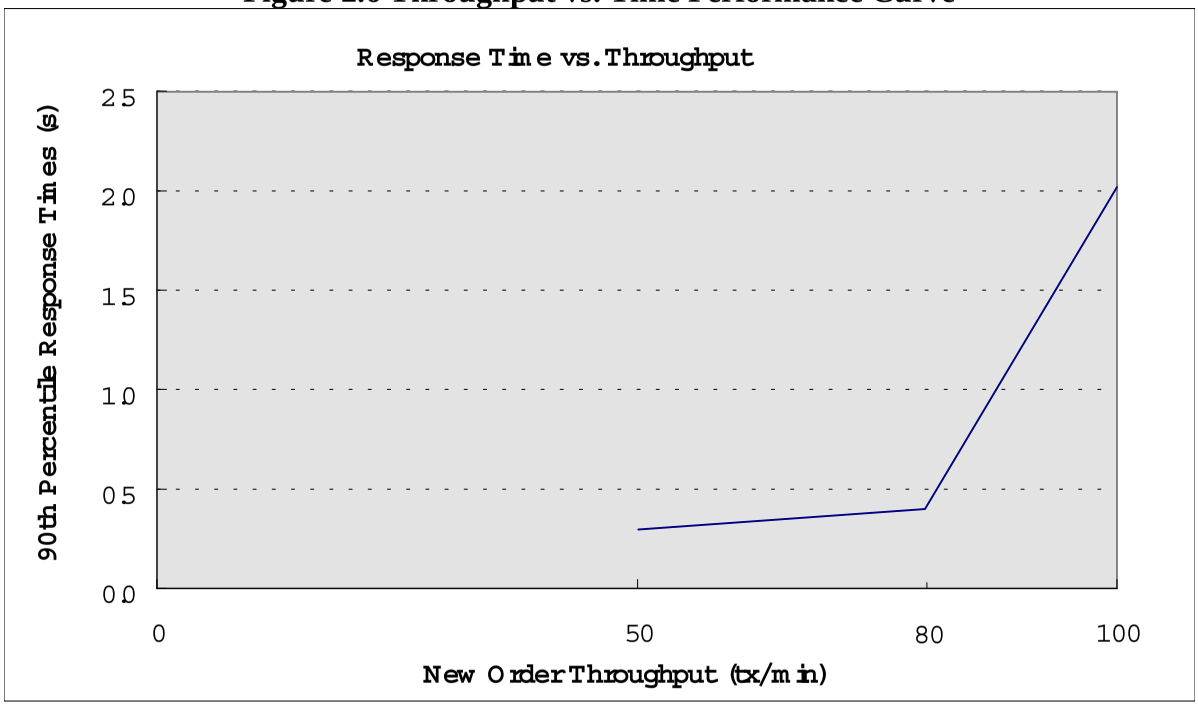
Figure 2.5: Stock Level Response Time Distribution



Response time versus Throughput Performance Curve

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

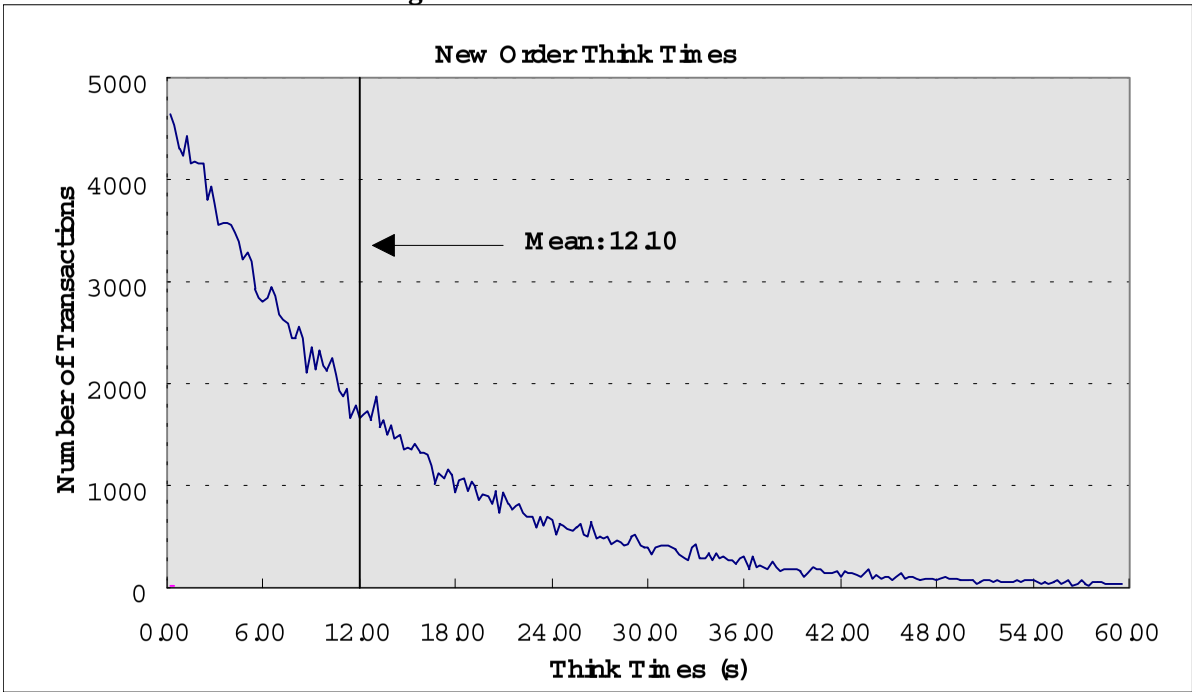
Figure 2.6 Throughput vs. Time Performance Curve



NEW-Order Think Time

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

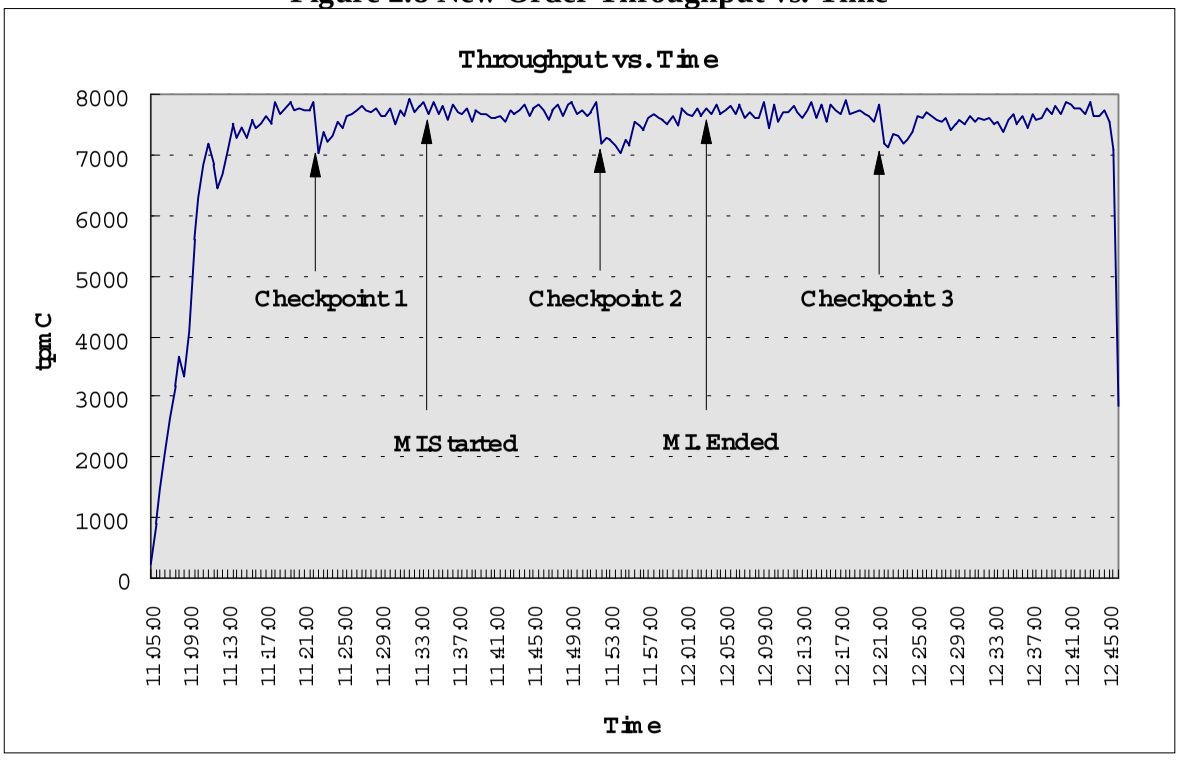
Figure 2.7 New-Order Think Time



New-Order Throughput vs. Elapsed Time

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

Figure 2.8 New Order Throughput vs. Time



Steady State

The method used to determine that the SUT had reached a steady state prior to commencing the measurement interval must be disclosed.

Steady state was confirmed by the throughput data collected during the run and graphed in Figure 2.8.

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.

A checkpoint in Microsoft SQL Server writes to disk all updated memory pages that have not been yet actually written to disk. SQL Server recovery interval parameter was set to the maximum allowable value to perform checkpoint at specific intervals. A checkpoint script, which issues specified number of checkpoint at specified (29.5 minutes) intervals, was started after all users logged in and sending transactions.

Reproducibility

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

The reproducibility test result is taken from another, non-overlapping, measurement interval of the same duration as the reported interval. The throughput difference measured over that interval was within 0.37% of reported interval result.

Measurement Period Duration

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

The reported measured interval was exactly 30 minutes long.

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 weighted random distribution that could not be adjusted during the run.

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 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.

The above statistics are disclosed in Table 1.

Checkpoint Count and Location

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.

Initial checkpoint was started 17.5 minutes after the start of ramp-up. Second checkpoint was started 29.5 minutes after the 1st checkpoint. The time from the start of the Measurement interval was 1080 seconds after. In accord with Clause 5.5.22, there is no checkpoint within the “guard zones” 1800/4=450 seconds from the beginning and end of the measurement interval.

Clause 6: SUT, Driver, and Communication Definition Related Items

Descriptions of RTE

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

The RTE used was the Microsoft BenchCraft RTE System. The RTE input parameters are listed in Appendix C.

Emulated Components

It must be demonstrated that the functionality and performance of the components being emulated in the Driver System are equivalent to the priced system. The results of the test described in Clause 6.6.3.4 must be disclosed..

AS configured for this test, the driver software emulates the traffic that would be observed from the users' PCs connected by Ethernet to the front-end clients using HTTP (HyperText Transfer Protocol) over TCP/IP. One tenth of a second (100 milli seconds) was added to each transaction time to compensate for the overhead of the Web browser.

Functional Diagrams and Detail of Driver System

A complete functional diagram of both the benchmark configuration 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.

The diagrams in figure 1.1 and 1.2 show the tested and priced benchmark configurations.

Network configurations and Driver system

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

Figure 1.1 and 1.2 in this report has the network configurations of both the tested system and the priced system.

The front-end clients were connected over one 100Mbps 100Base-T Ethernet segments to the back-end. The front-end clients were connected to the RTE over two 10Mbps 10Base-T Ethernet segments.

The priced PCs are also connected using 10Mbps Ethernet to the front-end clients.

Network Bandwidth

The bandwidth of the networks used in the tested/priced configuration must be disclosed.

The Ethernet used in the local area network (LAN) between the emulated terminals and the front-end system compiles with the IEEE 802.3 standard and has a bandwidth of 10Mbps.

Operator Intervention

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

This configuration does not require any operator intervention to sustain eight hours of the reported throughput.

Clause 7: Pricing Related Items

Hardware and Software Components

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 data. 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 and effective date(s) of price(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.

The detailed list of all hardware and software for the priced configuration is listed in the system-pricing summary.

Availability

The committed delivery date for general availability (availability date) of products used in the price calculation must be reported. When the priced system included 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.

All software and hardware components used in the tested and priced system are available now.

Throughput, and Price Performance

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.

- Maximum Qualified Throughput 7625.63 tpmC
- Price per tpmC $69.00 per tpmC
- NEC Express5800 MH4000 is available since May 21, 1997. All other hardware and Software are currently available.
- Total 5-yesr cost of ownership

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

This system is being priced for the United States of America.

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.*

None

Clause 9: Audit Related Items

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.

Appendix F contains the complete independent auditor's report by Francois Raab of Information Paradigm for the test described in this report.

Availability of the Full Disclosure Report

The Full Disclosure Report must be readily available to the public at a reasonable charge, similar to the charges for similar documents by the test sponsor. The report must be made available when results are made public. In order to use the phrase "TPC Benchmark™ C", the Full Disclosure Report must have been submitted to the TPC Administrator as well as written permission obtained to distribute same.

Requests for this TPC Benchmark™ C Full Disclosure Report should be sent to:
Transaction Processing Performance Council
c/o Shanley Public Relations
777 North First Street, Suite 6000
San Jose, CA 95112-6311
or your local NEC / Packard Bell -NEC office.

Appendix A : Application Source Code

Microsoft WEB Client

Makefile

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

!IF "$(SQL_LOC)" == ""
SQL_LOC=C:\MSSQL\DBLIB
!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                            = \odbcsdk

DBLIB           = $(SQL_LOC)
DBLIBINC        = $(DBLIB)\INCLUDE
ODBCINCDIR                      = $(ODBC)\INCLUDE
DBLIBDIR        = $(DBLIB)\LIB
ODBCLIBDIR                      = $(ODBC)\LIB32

!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 DEFCLPACKSIZE
                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 queued
            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 mesage 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 );

#ifndef 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, SQLUSMALLINT 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_HTTPEXTENSIONPROC )(
EXTENSION_CONTROL_BLOCK *pECB );

#ifdef __cplusplus
}
#endif

#endif  // end definition _HTTPEXT_H_
```

Resource.h

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

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE        101
#define _APS_NEXT_COMMAND_VALUE         40001
#define _APS_NEXT_CONTROL_VALUE         1000
#define _APS_NEXT_SYMED_VALUE           101
#endif
#endif
```

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_WEBDLL
                                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    //"Cannot process order status form.
#define     ERR_W_ID_INVALID
                        1009    //"Invalid Warehouse ID.
#define     ERR_CAN_NOT_SET_MAX_CONNECTIONS
                        1010    //"Insufficient memory to allocate # connections.
#define     ERR_NOSUCH_CUSTOMER
                                1011    //"No such customer.
#define     ERR_D_ID_INVALID
                        1012    //"Invalid District ID Must be 1 to 10.
#define     ERR_MAX_CONNECT_PARAM
                        1013    //"Max client connections exceeded, run
install to increase.
#define     ERR_INVALID_SYNC_CONNECTION
                        1014    //"Invalid Terminal Sync ID.
#define     ERR_INVALID_TERMID
                        1015    //"Invalid Terminal ID.
#define     ERR_PAYMENT_INVALID_CUSTOMER
                        1016    //"Payment Form, No such Customer.
#define     ERR_SQL_OPEN_CONNECTION
                        1017    //"SQLOpenConnection API Failed.
#define     ERR_STOCKLEVEL_MISSING_THRESHOLD_KEY   1018
                        //"Stock Level missing Threshold key "TT*".
#define     ERR_STOCKLEVEL_THRESHOLD_INVALID
                        1019    //"Stock Level Threshold invalid data type range = 1 -
99.
#define     ERR_STOCKLEVEL_THRESHOLD_RANGE
                        1020    //"Stock Level Threshold out of range, range must be
1 - 99.
#define     ERR_STOCKLEVEL_NOT_PROCESSED
                        1021    //"Stock Level not processed.
#define     ERR_NEWORDER_FORM_MISSING_DID
                        1022    //"New Order missing District key "DID*".
#define     ERR_NEWORDER_DISTRICT_INVALID
                        1023    //"New Order District ID Invalid range 1 - 10.
#define     ERR_NEWORDER_DISTRICT_RANGE
                        1024    //"New Order District ID out of Range. Range = 1 - 10.
#define     ERR_NEWORDER_CUSTOMER_KEY
                        1025    //"New Order missing Customer key "CID*".
```

```
#define     ERR_NEWORDER_CUSTOMER_INVALID
            1026        //"New Order customer id invalid data type, range = 1
to 3000.
#define     ERR_NEWORDER_CUSTOMER_RANGE
                        1027        //"New Order customer id out of range,
range = 1 to 3000.
#define     ERR_NEWORDER_MISSING_IID_KEY
            1028        //"New Order missing Item Id key "IID*".
#define     ERR_NEWORDER_ITEM_BLANK_LINES
            1029        //"New Order blank order lines all orders must be
continuous.
#define     ERR_NEWORDER_ITEMID_INVALID
            1030        //"New Order Item Id is wrong data type, must be
numeric.
#define     ERR_NEWORDER_MISSING_SUPPW_KEY
            1031        //"New Order missing Supp_W key "SP##*".
#define     ERR_NEWORDER_SUPPW_INVALID
            1032        //"New Order Supp_W invalid data type must be
numeric.
#define     ERR_NEWORDER_MISSING_QTY_KEY
            1033        //"New Order Missing Qty key "Qty##*".
#define     ERR_NEWORDER_QTY_INVALID
            1034        //"New Order Qty invalid must be numeric range 1 -
99.
#define     ERR_NEWORDER_SUPPW_RANGE
            1035        //"New Order Supp_W value out of range range = 1 -
Max Warehouses.
#define     ERR_NEWORDER_ITEMID_RANGE
            1036        //"New Order Item Id is out of range. Range = 1 to
999999.
#define     ERR_NEWORDER_QTY_RANGE
                        1037        //"New Order Qty is out of range. Range
= 1 to 99.
#define     ERR_PAYMENT_DISTRICT_INVALID
            1038        //"Payment District ID is invalid must be 1 - 10.
#define     ERR_NEWORDER_SUPPW_WITHOUT_ITEMID
            1039        //"New Order Supp_W field entered without a
corrisponding Item_Id.
#define     ERR_NEWORDER_QTY_WITHOUT_ITEMID
            1040        //"New Order Qty entered without a corrisponding
Item_Id.
#define     ERR_NEWORDER_NOITEMS_ENTERED
            1041        //"New Order Blank Items between items, items must
be continuous.
#define     ERR_PAYMENT_MISSING_DID_KEY
            1042        //"Payment missing District Key "DID*".
#define     ERR_PAYMENT_DISTRICT_RANGE
            1043        //"Payment District Out of range, range = 1 - 10.
#define     ERR_PAYMENT_MISSING_CID_KEY
            1044        //"Payment missing Customer Key "CID*".
#define     ERR_PAYMENT_CUSTOMER_INVALID
            1045        //"Payment Customer data type invalid, must be
numeric.
#define     ERR_PAYMENT_MISSING_CLT
                        1046        //"Payment missing Customer Last
Name Key "CLT*".
#define     ERR_PAYMENT_LAST_NAME_TO_LONG
            1047        //"Payment Customer last name longer than 16
characters.
#define     ERR_PAYMENT_CUSTOMER_RANGE
            1048        //"Payment Customer ID out of range, must be 1 to
3000.
#define     ERR_PAYMENT_CID_AND_CLT
                        1049        //"Payment Customer ID and Last Name
entered must be one or other.
#define     ERR_PAYMENT_MISSING_CDI_KEY
            1050        //"Payment missing Customer district key "CDI*".

#define     ERR_PAYMENT_CDI_INVALID
                        1051        //"Payment Customer district invalid
must be numeric.
#define     ERR_PAYMENT_CDI_RANGE
                        1052        //"Payment Customer district out of
range must be 1 - 10.
#define     ERR_PAYMENT_MISSING_CWI_KEY
            1053        //"Payment missing Customer Warehouse key
"CWI*".
#define     ERR_PAYMENT_CWI_INVALID
                        1054        //"Payment Customer Warehouse
invalid must be numeric.
#define     ERR_PAYMENT_CWI_RANGE
                        1055        //"Payment Customer Warehouse out of
range, 1 to Max Warehouses.
#define     ERR_PAYMENT_MISSING_HAM_KEY
            1056        //"Payment missing Amount key "HAM*".
#define     ERR_PAYMENT_HAM_INVALID
                        1057        //"Payment Amount invalid data type
must be numeric.
#define     ERR_PAYMENT_HAM_RANGE
                        1058        //"Payment Amount out of range, 0 -
9999.99.
#define     ERR_ORDERSTATUS_MISSING_DID_KEY
            1059        //"Order Status missing District key "DID*".
#define     ERR_ORDERSTATUS_DID_INVALID
            1060        //"Order Status District invalid, value must be numeric
1 - 10.
#define     ERR_ORDERSTATUS_DID_RANGE
            1061        //"Order Status District out of range must be 1 - 10.
#define     ERR_ORDERSTATUS_MISSING_CID_KEY
            1062        //"Order Status missing Customer key "CID*".
#define     ERR_ORDERSTATUS_MISSING_CLT_KEY
            1063        //"Order Status missing Customer Last Name key
"CLT*".
#define     ERR_ORDERSTATUS_CLT_RANGE
            1064        //"Order Status Customer last name longer than 16
characters.
#define     ERR_ORDERSTATUS_CID_INVALID
            1065        //"Order Status Customer ID invalid, range must be
numeric 1 - 3000.
#define     ERR_ORDERSTATUS_CID_RANGE
            1066        //"Order Status Customer ID out of range must be 1 -
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 queued
            short           w_id;
            //delivery warehouse
            short           o_carrier_id; //carrier id
} DELIVERY_TRANSACTION;

#ifdef USE_ODBC
            typedef struct _DBPROCESS
            {
                        HDBC        hdbc;
                        HSTMT       hstmt;
                        int                 spid;
                        void        *uPtr;
            } DBPROCESS, *PDBPROCESS;

            //dblib error message return values
            #define INT_EXIT      0
            #define INT_CONTINUE  1
            #define INT_CANCEL    2
#endif

//This structure defines the data necessary to keep distinct for each terminal or
client connection.
typedef struct _CLIENTDATA
{
            int                             inUse;
                                    //in use flag allows client entries to be
reused
            int                             w_id;
                                    //warehouse id assigned at welcome
form
            int                             d_id;
                                    //district id assigned at welcome form

            PDBPROCESS      dbproc;
                        //dblib connection pointer
            int                             spid;
                                    //spid assigned from dblib
            int                             iSyncId;
                        //syncronization 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;
				//syncronization 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;	//terminal id
	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, UDWORD cbColDef, SWORD ibScale,
PTR rgbValue, SDWORD cbValueMax);
	void ODBCError(PDBPROCESS dbproc);
	BOOL ExecuteStatement(PDBPROCESS dbproc, char
*szStatement);
	BOOL BindColumn(PDBPROCESS dbproc, SQLUSMALLINT icol,
SQLSMALLINT fCType, SQLPOINTER rgbValue, SQLINTEGER cbValueMax);
	BOOL GetResults(PDBPROCESS dbproc);
	BOOL MoreResults(PDBPROCESS  dbproc);
	BOOL 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
 */

#ifndef _INC_TRANS

          #define _INC_TRANS

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

          #ifndef DBINT
                    typedef long DBINT;
          #endif

          #define   DEFCLPACKSIZE
          4096
          #define   DEADLOCKWAIT
          10

          // String length constants
          #define SERVER_NAME_LEN          20
          #define DATABASE_NAME_LEN        20
          #define USER_NAME_LEN            20
          #define PASSWORD_LEN             20
          #define TABLE_NAME_LEN           20
          #define I_DATA_LEN               50
          #define I_NAME_LEN               24
          #define BRAND_LEN                 1
          #define LAST_NAME_LEN            16
          #define W_NAME_LEN               10
          #define ADDRESS_LEN              20
          #define STATE_LEN                 2
          #define ZIP_LEN                   9
          #define S_DIST_LEN               24
          #define S_DATA_LEN               50
          #define D_NAME_LEN               10
          #define FIRST_NAME_LEN           16
          #define MIDDLE_NAME_LEN           2
          #define PHONE_LEN                16
          #define DATETIME_LEN             30
          #define CREDIT_LEN                2
          #define C_DATA_LEN              250
          #define H_DATA_LEN               24
          #define DIST_INFO_LEN            24
          #define MAX_OL_NEW_ORDER_ITEMS    15
          #define MAX_OL_ORDER_STATUS_ITEMS 15
          #define STATUS_LEN               25
          #define OL_DIST_INFO_LEN                                  24

          // transaction structures

          typedef struct
          {
                    short
          ol_supply_w_id;
                    long
          ol_i_id;
                    char
          ol_i_name[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;
          #ifdef USE_ODBC
                    TIMESTAMP_STRUCT     o_entry_d;
          #else
                    DBDATEREC            o_entry_d;
          #endif
                    short          o_all_local;
                    double                    total_amount;
                    long           num_deadlocks;
                    char
          execution_status[STATUS_LEN];
                    OL_NEW_ORDER_DATA
Ol[MAX_OL_NEW_ORDER_ITEMS];
          } NEW_ORDER_DATA;

          typedef struct
          {
                    short                                       w_id;
                    short                                       d_id;
                    long                                        c_id;
                    short                                       c_d_id;
                    short                                       c_w_id;
                    double                                      h_amount;
          #ifdef USE_ODBC
                    TIMESTAMP_STRUCT     h_date;
          #else
                    DBDATEREC                 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
                    DBDATEREC                                   c_since;
          #endif
                    char
c_credit[CREDIT_LEN+1];
                    double                                      c_credit_lim;
                    double                                      c_discount;
                    double                                      c_balance;
                    char
c_data[200+1];
                    long
num_deadlocks;
                    char
execution_status[STATUS_LEN];
          } PAYMENT_DATA;

          typedef struct
          {
                    long                                        ol_i_id;
                    short
          ol_supply_w_id;
                    short                                       ol_quantity;
                    double                                      ol_amount;
          #ifdef USE_ODBC
                    TIMESTAMP_STRUCT     ol_delivery_d;
          #else
                    DBDATEREC
          ol_delivery_d;
          #endif
          } OL_ORDER_STATUS_DATA;
```

```
        typedef struct
        {
                short           w_id;
                short           d_id;
                long            c_id;
                char            c_first[FIRST_NAME_LEN+1];
                char            c_middle[MIDDLE_NAME_LEN+1];
                char            c_last[LAST_NAME_LEN+1];
                double           c_balance;
                long            o_id;
        #ifdef USE_ODBC
                TIMESTAMP_STRUCT      o_entry_d;
        #else
                DBDATEREC          o_entry_d;
        #endif
                short           o_carrier_id;
                OL_ORDER_STATUS_DATA
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;
                DEL_ITEM                        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
```

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

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

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

BOOL                                            bDone;

        //delivery executable termination request flag
BOOL                                            bFlush;

        //Flush delivery log info when written.

LPDELIVERY_PACKET               pDeliveryCache;

int
        versionMS = 4;
        //delivery executable version number.
int
        versionMM = 0;
        //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) *
iQSlotts)) )
		return ERR_INSUFFICIENT_MEMORY;

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

	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 successfull
 *
		TRUE	user has requested parameter information screen be
displayed.
 *
 * COMMENTS:	None
 *
 */
```

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


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

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

	strcpy(szServer, argv[i]+2);
					break;
				case '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 flush when written.
					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.%2.2d.%3.3d                         *\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
 *

	to setup registry.
 *
	ERR_SUCCESS
	Reqistry 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<iQSlotts; i++)
		{
			if ( !pDeliveryCache[i].bInUse )
				break;
		}
		if ( i < iQSlotts )
		{

	EnterCriticalSection(&DeliveryCriticalSection);
			pDeliveryCache[i].bInUse = TRUE;

	LeaveCriticalSection(&DeliveryCriticalSection);
		}
		else
		{

	EnterCriticalSection(&DeliveryCriticalSection);
			if ( !(pDeliveryCache =
(LPDELIVERY_PACKET)realloc(pDeliveryCache, sizeof(DELIVERY_PACKET) *
(iQSlotts+512))) )
			{

	ErrorMessage(ERR_INSUFFICIENT_MEMORY);

	LeaveCriticalSection(&DeliveryCriticalSection);
				return;
			}
			for(i=iQSlotts; i<iQSlotts+512; i++)
				pDeliveryCache[i].bInUse
= FALSE;
			i = iQSlotts;
			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
 *
 */
```

```
#ifndef 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
 *                                      DBINT
        msgno                           message number
 *                                      int
                        msgstate        message state
 *                                      int
                        severity        message severity
 *                                      char
        *msgtext                printable message description
 *
 * RETURNS:                     int
        INT_CONTINUE            continue if error is SQLETIME else
INT_CANCEL action
 *
                        INT_CANCEL                      cancel
operation
 *
 * COMMENTS:            This function also sets the dead lock dbproc variable
if necessary.
 *
 */
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 successfull
 *
        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");

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

                        dbsqlexec(*dbproc);
                        while (dbresults(*dbproc) != NO_MORE_RESULTS)
```

```
                        while (dbnextrow(*dbproc) != NO_MORE_ROWS);

                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;
}

#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;

                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            Pointer to storage for the data. If
rgbValue is a null pointer, the
 *
                                          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:		DBRPOCESS		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:		DBRPOCESS		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 defination 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=\"TERMID\"
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 baised on the
 *					fact that DLL_PROCESS_ATTACH is
only called from the inet service once. Connections
 *					are sent to this function as thread
attachments.
 *
 * ARGUMENTS:		HANDLE	hModule
		module handle
 *						DWORD
		ul_reason_for_call		reason for call
 *						LPVOID	lpReserved
			reserved for future use
 *
 * RETURNS:				BOOL	FALSE
			errors occured in initialization
 *
		TRUE							DLL
successfully initialized
 *
 * COMMENTS:		None
 *
 */

BOOL APIENTRY 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_TYPE_BYTE |
PIPE_READMODE_BYTE | PIPE_NOWAIT,
					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);
	lstrcpyn(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.wYear,
systemTime.wMonth, systemTime.wDay,
			systemTime.wHour,
systemTime.wMinute, systemTime.wSecond,
			pECB->lpszQueryString);
		fclose(fp);
	}


	//process http query
	if ( !ProcessQueryString(pECB, &iCmd, &FormId, &TermId,
&iSyncId) )
	{
		if ( TermId < 0 )
			ErrorMessage(pECB,
ERR_INVALID_TERMID, ERR_TYPE_WEBDLL, NULL, TermId, iSyncId);
		else
			ErrorMessage(pECB,
ERR_COMMAND_UNDEFINED, ERR_TYPE_WEBDLL, NULL, TermId,
iSyncId);
		return
HSE_STATUS_SUCCESS_AND_KEEP_CONN;
	}

	if ( TermId != 0 )
	{
		if ( !IsValidTermId(TermId) )
		{
```

```
                                        ErrorMessage(pECB,
ERR_INVALID_TERMID, ERR_TYPE_WEBDLL, NULL, TermId, iSyncId);
                                        return
HSE_STATUS_SUCCESS_AND_KEEP_CONN;
                                }

                                //must have a valid syncid here since termid is valid
                                if ( iSyncId < 1 || iSyncId !=
Term.pClientData[TermId].iSyncId )
                                {
                                        ErrorMessage(pECB,
ERR_INVALID_SYNC_CONNECTION, ERR_TYPE_WEBDLL, NULL, TermId,
iSyncId);
                                        return
HSE_STATUS_SUCCESS_AND_KEEP_CONN;
                                }

        }

        //set use time
        Term.pClientData[TermId].iTickCount = GetTickCount();

        //go execute http: command
        (*DoCmd[iCmd])(pECB, FormId, TermId, iSyncId);

        //finish up and keep connection
        return HSE_STATUS_SUCCESS_AND_KEEP_CONN;
}

/* FUNCTION: static BOOL IsValidTermId(int TermId)
 *
 * PURPOSE:             This function checks to see of the passed in terminal
id is valid.
 *
 * ARGUMENTS:           int
        TermId                                                  client
terminal id
 *
 * RETURNS:                             BOOL    FALSE
                                                Terminal ID Invalid
 *
        TRUE
        Terminal ID valid
 *
 * COMMENTS:            None
 *
 */

static BOOL IsValidTermId(int TermId)
{
        return (BOOL) ( TermId > 0 && TermId <= Term.iAvailable &&
Term.pClientData[TermId].inUse );
}

/* FUNCTION: BOOL ProcessQueryString(EXTENSION_CONTROL_BLOCK
*pECB, int *pCmd, int *pFormId, int *pTermId, int *pSyncId)
 *
 * PURPOSE:             This function extracts the relevent information out of
the http command passed in from
 *                              the browser.
 *
 * ARGUMENTS:           EXTENSION_CONTROL_BLOCK         *pECB
        structure pointer to passed in internet
 *
        service information.
 *                                              int
                *pCmd
        returned command id
 *                                              int
                *pFormId                                        returned
active form client browser is on
 *                                              int
                *pTermId                                        returned
client terminal id
 *
 * RETURNS:                     BOOL    FALSE
                                        success
 *
        TRUE
        command passed in is invalid
 *
 * COMMENTS:            If this is the initial connection i.e. client is at welcome
screen then
 *                                              there will not be a terminal
id or current form id if 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 ( !stricmp(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 borwser
 *
        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
 *

        service information.
 * 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
 *

                service information.
 * RETURNS:                         None
 *
 * COMMENTS:            A terminal id can be allocated but still be invalid if the
requested warehouse number
 *                                                  is outside the range
specified in the registry. This then will force the client id
 *                                                  to be invalid and an error
message sent to the users browser.
 */

void SubmitCmd(EXTENSION_CONTROL_BLOCK *pECB, int iFormId, int
iTermId, int iSyncId)
{
                int             iCurrent;

                if ( (iCurrent = (*Term.Add)(pECB, pECB->lpszQueryString)) < 0 )
                {
                                ErrorMessage(pECB,
ERR_CANNOT_INIT_TERMINAL, ERR_TYPE_WEBDLL, NULL, iCurrent,
iSyncId);
                                return;
                }

                if ( Term.pClientData[iCurrent].w_id > iMaxWareHouses ||
Term.pClientData[iCurrent].w_id < 1 )
                {
                                ErrorMessage(pECB, ERR_W_ID_INVALID,
ERR_TYPE_WEBDLL, NULL, iCurrent, iSyncId);
                                (*Term.Delete)(pECB, iCurrent);
                                return;
                }
                if ( Term.pClientData[iCurrent].d_id < 1 ||
Term.pClientData[iCurrent].d_id > 10 )
                {
                                ErrorMessage(pECB, ERR_D_ID_INVALID,
ERR_TYPE_WEBDLL, NULL, iCurrent, iSyncId);
                                (*Term.Delete)(pECB, iCurrent);
                                return;
                }

                WriteZString(pECB, MakeMainMenuForm(iCurrent,
Term.pClientData[iCurrent].iSyncId) );

                return;
}

/* FUNCTION: void BeginCmd(EXTENSION_CONTROL_BLOCK *pECB, int
iFormId, int iTermId, int iSyncId)
 *
 * PURPOSE:             This function is the first command executed. It is
executed with the command
 *                                  CMD=Begin?Server=xxx from the http
command line.
 *
 * ARGUMENTS:           int
                                iFormId                 unused
```

```
 *                                                  int
                                                    iTermId
                id of calling browser, i.e. 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
 *
 * 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_TERMID,
                                    "Invalid Terminal ID."

                                                },
                        {
            ERR_PAYMENT_INVALID_CUSTOMER,
            "Payment Form, No such Customer."

            },
                        {           ERR_SQL_OPEN_CONNECTION,
                                    "SQLOpenConnection API Failed."

                                    },
                        {
            ERR_STOCKLEVEL_MISSING_THRESHOLD_KEY,  "Stock Level
missing Threshold key \"TT*\"."
                        },
                        {
            ERR_STOCKLEVEL_THRESHOLD_INVALID,
            "Stock Level Threshold invalid data type range = 1 - 99."
                        },
                        {
            ERR_STOCKLEVEL_THRESHOLD_RANGE,
            "Stock Level Threshold out of range, range must be 1 - 99."
                        },
                        {
            ERR_STOCKLEVEL_NOT_PROCESSED,
            "Stock Level not processed."

            },
                        {
            ERR_NEWORDER_FORM_MISSING_DID,
            "New Order missing District key \"DID*\"."
                                                },
                        {
            ERR_NEWORDER_DISTRICT_INVALID,
            "New Order District ID Invalid range 1 - 10."
                                                },
                        {
            ERR_NEWORDER_DISTRICT_RANGE,
            "New Order District ID out of Range. Range = 1 - 10."
                        },
                        {           ERR_NEWORDER_CUSTOMER_KEY,
                                    "New Order missing Customer key
\"CID*\"."
            },
                        {
            ERR_NEWORDER_CUSTOMER_INVALID,
            "New Order customer id invalid data type, range = 1 to 3000."
            },
                        {
            ERR_NEWORDER_CUSTOMER_RANGE,
            "New Order customer id out of range, range = 1 to 3000."
                        },
                        {           ERR_NEWORDER_MISSING_IID_KEY,
                                    "New Order missing Item Id key \"IID*\"."

            },
                        {
            ERR_NEWORDER_ITEM_BLANK_LINES,
            "New Order blank order lines all orders must be continuous."
            },
                        {           ERR_NEWORDER_ITEMID_INVALID,
                        "New Order Item Id is wrong data type, must be
numeric."                           },
```

```
                {
        ERR_NEWORDER_MISSING_SUPPW_KEY,
        "New Order missing Supp_W key \"SP##*\"."
                                        },
                {       ERR_NEWORDER_SUPPW_INVALID,
                        "New Order Supp_W invalid data type
must be numeric."                       },
                {
        ERR_NEWORDER_MISSING_QTY_KEY,
        "New Order Missing Qty key \"Qty##*\"."
                                        },
                {       ERR_NEWORDER_QTY_INVALID,
                        "New Order Qty invalid must be numeric
range 1 - 99."                          },
                {       ERR_NEWORDER_SUPPW_RANGE,
                        "New Order Supp_W value out of range
range = 1 - Max Warehouses."    },
                {       ERR_NEWORDER_ITEMID_RANGE,
                        "New Order Item Id is out of range.
Range = 1 to 999999."                   },
                {       ERR_NEWORDER_QTY_RANGE,
                                "New Order Qty is out of
range. Range = 1 to 99."
        },
                {       ERR_PAYMENT_DISTRICT_INVALID,
                "Payment District ID is invalid must be 1 - 10."
                                        },
                {
        ERR_NEWORDER_SUPPW_WITHOUT_ITEMID,
        "New Order Supp_W field entered without a corrisponding
Item_Id."       },
                {
        ERR_NEWORDER_QTY_WITHOUT_ITEMID,
        "New Order Qty entered without a corrisponding Item_Id."
                },
                {
        ERR_NEWORDER_NOITEMS_ENTERED,
        "New Order Blank Items between items, items must be
continuous." },
                {       ERR_PAYMENT_MISSING_DID_KEY,
                "Payment missing District Key \"DID*\"."
                                        },
                {       ERR_PAYMENT_DISTRICT_RANGE,
                        "Payment District Out of range, range =
1 - 10."                                },
                {       ERR_PAYMENT_MISSING_CID_KEY,
                "Payment missing Customer Key \"CID*\"."


        },
                {
        ERR_PAYMENT_CUSTOMER_INVALID,
        "Payment Customer data type invalid, must be numeric."
                                },
                {       ERR_PAYMENT_MISSING_CLT,
                        "Payment missing Customer Last Name
Key \"CLT*\"."
        },
                {
        ERR_PAYMENT_LAST_NAME_TO_LONG,
        "Payment Customer last name longer than 16 characters."
                                },
                {
        ERR_PAYMENT_CUSTOMER_RANGE,
                "Payment Customer ID out of range, must be 1 to
3000."                                  },
                {       ERR_PAYMENT_CID_AND_CLT,
                        "Payment Customer ID and Last Name
entered must be one or other."  },
                {       ERR_PAYMENT_MISSING_CDI_KEY,
                "Payment missing Customer district key \"CDI*\"."
                                        },
                {       ERR_PAYMENT_CDI_INVALID,
                        "Payment Customer district invalid must
be numeric."                            },
                {       ERR_PAYMENT_CDI_RANGE,
                                "Payment 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=\"TERMID\" 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=\"TERMID\" 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=\"TERMID\" 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=\"TERMID\" 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 valud found
 *
 *
 * COMMENTS:            http keys are formatted either KEY=value& or
KEY=value\0. This DLL formats
 *                                              TPC-C input fields in such
a manner that the keys can be extracted in the
 *                                              above manner.
 */
```

```
static BOOL GetKeyValue(char *pQueryString, char *pKey, char *pValue, int
iMax)
{
                char *ptr;


                if ( !(ptr=strstr(pQueryString, pKey)) )
                        return FALSE;
                if ( !(ptr=strchr(ptr, '=')) )
                        return FALSE;
                ptr++;
                iMax--;
                while( *ptr && *ptr != '&' && iMax)
                {
                        *pValue++ = *ptr++;
                        iMax--;
                }
                *pValue = 0;
                return TRUE;
}

/* FUNCTION: void TermInit(void)
 *
 * PURPOSE:             This function initializes the client ternimal 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
 *
 */

#ifndef 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        message state
 *                                      int
                        severity        message severity
 *                                      char
                *msgtext                printable message description
 *
 * RETURNS:                             int
                INT_CONTINUE            continue if error is SQLETIME else
INT_CANCEL action
 *
                        INT_CANCEL                      cancel
operation
 *
 * COMMENTS:            This function also sets the dead lock dbproc variable
if necessary.
 *
 */

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

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


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

                // deadlock message
                if (msgno == 1205)
                {
                        // set the deadlock indicator
                        if ( pEcbInfo )
                                pEcbInfo->bDeadlock = TRUE;
                        else
                                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_WEBDLL, 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.
 *				int
		iTermId			terminal id of browser
 *				int
		iSyncId			sync id of browser
 *				DBPROCESS
	**dbproc	pointer to returned DBPROCESS
 *				char
	*server			SQL server name
 *				char
	*database	SQL server database
 *				char
	*user			user name
 *				char
	*password	user password
 *				char
	*app			pointer to returned application array
 *				int
		*spid			pointer to returned spid
 *				long
	*pack_size	pointer to returned default pack size
 *
 * RETURNS:			BOOL		FALSE		if successfull
 *
	TRUE		if an error occurs
 *
 * COMMENTS:		None
 *
 */


#ifdef USE_ODBC
	static BOOL
SQLOpenConnection(EXTENSION_CONTROL_BLOCK *pECB, int iTermId, int
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(ECBINFO));

		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");

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

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

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

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

                        return FALSE;
        }

#endif

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

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

/* FUNCTION: 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.
 *                                              int
                                                iTermId
        terminal id of browser
 *                                              int
                                                iSyncId
        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;
                }

#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 commited.");
                                        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 commited.");
                                        return FALSE;
                                }
                        }

                        // If we reached here, it means we quit after
MAX_RETRY deadlocks
                        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.
 *                                                      int
                                                        iTermId
                terminal id of browser
 *                                                      int
                                                        iSyncId
                sync id of browser
 *                                                      DBPROCESS
                                        *dbproc
                connection db process id
 *                                                      NEW_ORDER_DATA
                        *pNewOrder                      pointer to new order
structure for input/output data
 *                                                      short
                                        deadlock_retry          retry count if
deadlocked
 *
 * RETURNS:                             int             TRUE            transaction
committed
 *                                                                      FALSE
                item number not valid
 *                                                                      -1
                deadlock max retry reached
 *
 *
 * COMMENTS:            None
 *
 */

#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 commited.");
                                                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 resutls
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 commited.");
                                                        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 deadlocks
                        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.
 *                                                      int
                iTermId
                terminal id of browser
 *                                                      int
                iSyncId
                sync id of browser
 *                                                      DBPROCESS
                *dbproc
                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
                max deadlocked reached
 *
 * 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,      0, &pPayment->w_id,     0);
                                BindParameter(dbproc, 2,
SQL_C_SSHORT,   SQL_SMALLINT, 0,      0, &pPayment->c_w_id,   0);
                                BindParameter(dbproc, 3,
SQL_C_DOUBLE,   SQL_NUMERIC,  6,      2, &pPayment->h_amount, 0);
                                BindParameter(dbproc, 4,
SQL_C_STINYINT, SQL_TINYINT,  0,      0, &pPayment->d_id,     0);
                                BindParameter(dbproc, 5,
SQL_C_STINYINT, SQL_TINYINT,  0,      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 commited.");

                                                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 commited.");
                                                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.
 *                                                      int
                                                        iTermId
        terminal id of browser
 *                                                      int
                                                        iSyncId
        sync id of browser
 *                                                      DBPROCESS
                                        *dbproc
        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 successfull
 *
 * COMMENTS:          None
 *
 */

#ifdef 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 commited.");
                                                return 1;
                                        }
                        }

                        // If we reached here, it means we quit after MAX_RETRY deadlocks
                        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 commited.");
                                                return 1;
                                        }
                        }
```

```
                        // If we reached here, it means we quit after
MAX_RETRY deadlocks
                        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\">");
                wsprintf(szForm+strlen(szForm), "<INPUT TYPE=\"hidden\"
NAME=\"FORMID\" VALUE=\"%d\">", STOCK_LEVEL_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>                   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=\"TERMID\" 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>"

	"<HEAD><TITLE>TPC-C New Order</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 ( bValid )
			strcat(szForm, "<INPUT
TYPE=\"hidden\" NAME=\"STATUSID\" VALUE=\"0\">");
		else
			wsprintf(szForm+strlen(szForm),
"<INPUT TYPE=\"hidden\" NAME=\"STATUSID\" VALUE=\"%d\">",
ERR_BAD_ITEM_ID);
	}

	wsprintf(szForm+strlen(szForm), "<INPUT TYPE=\"hidden\"
NAME=\"FORMID\" VALUE=\"%d\">", NEW_ORDER_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>			New Order<BR>");

	if ( bInput )
	{
		wsprintf(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		Date: %2.2d-%2.2d-%4.4d
%2.2d:%2.2d:%2.2d <BR>",

	Term.pClientData[iTermId].newOrderData.w_id,

	Term.pClientData[iTermId].newOrderData.d_id,
```

```
                Term.pClientData[iTermId].newOrderData.o_entry_d.day,

                Term.pClientData[iTermId].newOrderData.o_entry_d.month,

                Term.pClientData[iTermId].newOrderData.o_entry_d.year,

                Term.pClientData[iTermId].newOrderData.o_entry_d.hour,

                Term.pClientData[iTermId].newOrderData.o_entry_d.minute,

                Term.pClientData[iTermId].newOrderData.o_entry_d.second);
                        }
                        else
                        {
                                wsprintf(szForm+strlen(szForm),
"Warehouse: %4.4d   District: %2.2d                Date:<BR>",

                Term.pClientData[iTermId].newOrderData.w_id,

                Term.pClientData[iTermId].newOrderData.d_id);
                        }

                        FormatHTMLString(szName,
Term.pClientData[iTermId].newOrderData.c_last, 16),
                        FormatHTMLString(szCredit,
Term.pClientData[iTermId].newOrderData.c_credit, 2);

                        wsprintf(szForm+strlen(szForm), "Customer:  %4.4d
Name: %s   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..\">"

                "<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 *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\">");
                wsprintf(szForm+strlen(szForm), "<INPUT TYPE=\"hidden\"
NAME=\"FORMID\" VALUE=\"%d\">", PAYMENT_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>
Payment<BR>");

                if ( bInput )
                        strcat(szForm,          "Date:<BR><BR>" );
                else
```

```
        {
                wsprintf(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);
        }
        wsprintf(szForm+strlen(szForm), "Warehouse: %4.4d",
Term.pClientData[iTermId].paymentData.w_id);

        if ( bInput )
        {
                strcat(szForm,          "          District:
<INPUT NAME=\"DID*\" SIZE=1><BR><BR><BR><BR><BR>"

        "Customer: <INPUT NAME=\"CID*\" SIZE=4>"

        "Cust-Warehouse: <INPUT NAME=\"CWI*\" SIZE=4>  "

        "Cust-District: <INPUT NAME=\"CDI*\" SIZE=1><BR>"

        "Name:          <INPUT NAME=\"CLT*\" SIZE=16>
Since:<BR>"

        "                       Credit:<BR>"

        "                       Disc:<BR>"

        "                       Phone:<BR><BR>"

        "Amount Paid:   $<INPUT NAME=\"HAM*\" SIZE=7>   New
Cust Balance:<BR>"

        "Credit Limit:<BR><BR>Cust-Data:
<BR><BR><BR><BR></PRE><HR>"

        "<INPUT TYPE=\"submit\" NAME=\"CMD\"
VALUE=\"Process\"><INPUT TYPE=\"submit\" NAME=\"CMD\"
VALUE=\"Menu\">"

        "</BODY></FORM></HTML>" );
        }
        else
        {
                sprintf(szForm+strlen(szForm),          "
District: %2.2d<BR>",

        Term.pClientData[iTermId].paymentData.d_id);

                FormatHTMLString(szTmpStr1,
Term.pClientData[iTermId].paymentData.w_street_1, 20);
                FormatHTMLString(szTmpStr2,
Term.pClientData[iTermId].paymentData.d_street_1, 20);

                sprintf(szForm+strlen(szForm),"%s
%s<BR>", szTmpStr1, szTmpStr2);

                FormatHTMLString(szTmpStr1,
Term.pClientData[iTermId].paymentData.w_street_2, 20);
                FormatHTMLString(szTmpStr2,
Term.pClientData[iTermId].paymentData.d_street_2, 20);

                sprintf(szForm+strlen(szForm),"%s
%s<BR>", szTmpStr1, szTmpStr2);

                FormatString(szW_Zip, szZipPic,
Term.pClientData[iTermId].paymentData.w_zip);
                FormatString(szD_Zip, szZipPic,
Term.pClientData[iTermId].paymentData.d_zip);

                FormatHTMLString(szTmpStr1,
Term.pClientData[iTermId].paymentData.w_city, 20);
                FormatHTMLString(szTmpStr2,
Term.pClientData[iTermId].paymentData.w_state, 2);
                FormatHTMLString(szTmpStr3,
Term.pClientData[iTermId].paymentData.d_city, 20);
                FormatHTMLString(szTmpStr4,
Term.pClientData[iTermId].paymentData.d_state, 2);

                wsprintf(szForm+strlen(szForm), "%s %s %10.10s
%s %s %10.10s<BR><BR>",
                        szTmpStr1, szTmpStr2, szW_Zip,
szTmpStr3, szTmpStr4, szD_Zip );

                wsprintf(szForm+strlen(szForm), "Customer: %4.4d
Cust-Warehouse: %4.4d  Cust-District: %2.2d<BR>",

        Term.pClientData[iTermId].paymentData.c_id,

        Term.pClientData[iTermId].paymentData.c_w_id,

        Term.pClientData[iTermId].paymentData.c_d_id);

                FormatHTMLString(szTmpStr1,
Term.pClientData[iTermId].paymentData.c_first, 16);
                FormatHTMLString(szTmpStr2,
Term.pClientData[iTermId].paymentData.c_middle, 2);
                FormatHTMLString(szTmpStr3,
Term.pClientData[iTermId].paymentData.c_last, 16);

                wsprintf(szForm+strlen(szForm), "Name:   %s %s %s
Since:  %2.2d-%2.2d-%4.4d<BR>",
                        szTmpStr1, szTmpStr2, szTmpStr3,

        Term.pClientData[iTermId].paymentData.c_since.day,

        Term.pClientData[iTermId].paymentData.c_since.month,

        Term.pClientData[iTermId].paymentData.c_since.year);

                FormatHTMLString(szTmpStr1,
Term.pClientData[iTermId].paymentData.c_street_1, 20);
                FormatHTMLString(szTmpStr2,
Term.pClientData[iTermId].paymentData.c_credit, 2);

        wsprintf(szForm+strlen(szForm), "       %s                  Credit: %s<BR>",
szTmpStr1, szTmpStr2);

                FormatHTMLString(szTmpStr1,
Term.pClientData[iTermId].paymentData.d_street_2, 20);
        sprintf(szForm+strlen(szForm), "       %s                  %%Disc:
%5.2f<BR>",
                        szTmpStr1,
Term.pClientData[iTermId].paymentData.c_discount);

                FormatString(szC_Zip, szZipPic,
Term.pClientData[iTermId].paymentData.c_zip);
                FormatString(szC_Phone, "XXXXXX-XXX-XXX-
XXXX", Term.pClientData[iTermId].paymentData.c_phone);

                FormatHTMLString(szTmpStr1,
Term.pClientData[iTermId].paymentData.c_city, 20);
                FormatHTMLString(szTmpStr2,
Term.pClientData[iTermId].paymentData.c_state, 2);

                wsprintf(szForm+strlen(szForm), "       %s %s
%10.10s       Phone:  %-19.19s<BR><BR>",
                        szTmpStr1, szTmpStr2, szC_Zip,
szC_Phone );

                sprintf(szForm+strlen(szForm), "Amount Paid:
$%7.2f      New 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 <= l )

        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
 *							int
		iSyncId		client browser sync id
 *							BOOL
		bInput		TRUE if form is being constructed for input else
FALSE
 *
 * RETURNS:				char *
		A pointer to buffer inside client structure where HTML form is built.
 *
 * COMMENTS:		The internal client buffer is created when the terminal
id is assigned and should not
 *							be freed except when the
client terminal id is no longer needed.
 */

static char *MakeOrderStatusForm(int iTermId, int iSyncId, BOOL bInput)
{
		char		*szForm;
		char		c_first[98];
		char		c_middle[14];
		char		c_last[98];
		int			i;

		szForm = (char *)Term.pClientData[iTermId].szBuffer;

		Term.pClientData[iTermId].orderStatusData.w_id =
Term.pClientData[iTermId].w_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=\"SYNCID\" 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:<BR>"

			"Supply-W	Item-Id	Qty	Amount	Delivery-
Date<BR></PRE>"

			"<HR><INPUT TYPE=\"submit\" NAME=\"CMD\"
VALUE=\"Process\"><INPUT TYPE=\"submit\" NAME=\"CMD\"
VALUE=\"Menu\">"

			"</BODY></FORM></HTML>" );
		}
		else
		{
			wsprintf(szForm+strlen(szForm), "District:
%2.2d<BR>", Term.pClientData[iTermId].orderStatusData.d_id);

			FormatHTMLString(c_first,
Term.pClientData[iTermId].orderStatusData.c_first, 16);
			FormatHTMLString(c_middle,
Term.pClientData[iTermId].orderStatusData.c_middle, 2);
			FormatHTMLString(c_last,
Term.pClientData[iTermId].orderStatusData.c_last, 16);

			wsprintf(szForm+strlen(szForm), "Customer: %4.4d
Name: %s %s %s<BR>",

			Term.pClientData[iTermId].orderStatusData.c_id, c_first, c_middle,
c_last);

			sprintf(szForm+strlen(szForm), "Cust-Balance:
$%9.2f<BR><BR>",

			Term.pClientData[iTermId].orderStatusData.c_balance);

			wsprintf(szForm+strlen(szForm), "Order-Number:
%8.8d  Entry-Date: %2.2d-%2.2d-%4.4d %2.2d:%2.2d:%2.2d  Carrier-Number:
%2.2d<BR>",

			Term.pClientData[iTermId].orderStatusData.o_id,

			Term.pClientData[iTermId].orderStatusData.o_entry_d.day,

			Term.pClientData[iTermId].orderStatusData.o_entry_d.month,

			Term.pClientData[iTermId].orderStatusData.o_entry_d.year,

			Term.pClientData[iTermId].orderStatusData.o_entry_d.hour,

			Term.pClientData[iTermId].orderStatusData.o_entry_d.minute,

			Term.pClientData[iTermId].orderStatusData.o_entry_d.second,

			Term.pClientData[iTermId].orderStatusData.o_carrier_id);
				strcat(szForm+strlen(szForm), "Supply-W	Item-Id
Qty	Amount	Delivery-Date<BR>");

				for(i=0;
i<Term.pClientData[iTermId].orderStatusData.o_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..\">"

			"<INPUT TYPE=\"submit\" NAME=\"CMD\" VALUE=\"..Order-
Status..\">"

			"<INPUT TYPE=\"submit\" NAME=\"CMD\" VALUE=\"..Stock-
Level..\">"

			"<INPUT TYPE=\"submit\" NAME=\"CMD\" VALUE=\"..Exit..\">"

			"</BODY></FORM></HTML>" );
		}

		return szForm;
}

/* FUNCTION: char *MakeDeliveryForm(int iTermId, int iSyncId, BOOL bInput,
BOOL bSuccess)
 *
 * 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
                bSuccess TRUE if Delivery succeeded else FALSE
 *
 * RETURNS:                       char *
                A pointer to buffer inside client structure where HTML form is built.
 *
 * COMMENTS:              The internal client buffer is created when the terminal
id is assigned and should not
 *                                                  be freed except when the
client terminal id is no longer needed.
 */

static char *MakeDeliveryForm(int iTermId, int 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
pDst and places a
 *                                  null character at the end of the
destination string.
 *
 * ARGUMENTS:             char                                    *pDest
                destination string pointer
 *                                                  char
                *pSrc           source string pointer
 *                                                  int
                                n                       number of characters to
copy
 *
 * RETURNS:                       None
 *
 * COMMENTS:              Unlike strncpy this function ensures that the result
string is
 *                                                  always null terminated.
 *
 */

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

                return;
}

/* FUNCTION: void ProcessNewOrderForm(EXTENSION_CONTROL_BLOCK
                *pECB, int iTermId, int iSyncId)
 *
 * PURPOSE:               This function gets and validates the input data from
the new order form
 *                                  filling in the required input variables. it
then calls the SQLNewOrder
 *                                  transaction, constructs the output form
and writes it back to client
 *                                  browser.
 *
 * ARGUMENTS:             EXTENSION_CONTROL_BLOCK         *pECB
                passed in structure pointer from inetsrv.
 *                                                  int
                                                        iTermId         client
browser terminal id
 *                                                  int
                                                        iSyncId client browser
sync id
 *
 * RETURNS:                       None
 *
 * COMMENTS:              None
 *
 */

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.
 *                                              int
                                                iTermId         client
browser terminal id
 *                                              int
                                                iSyncId client browser
sync id
 *
 * 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.
 *                                                      int
                                                        iTermId         client
browser terminal id
 *                                                      int
                                                        iSyncId         clinet
browser sync id
 *
 * 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;
                        _strupr( szTmp );
                        strcpy(pOrderStatusData->c_last, szTmp);
                        if ( strlen(pOrderStatusData->c_last) > 16 )
                                        return
ERR_ORDERSTATUS_CLT_RANGE;
        }
        else
        {
                        if ( !IsNumeric(szTmp) )
                                        return
ERR_ORDERSTATUS_CID_INVALID;
                        pOrderStatusData->c_id = atoi(szTmp);
                        if ( !GetKeyValue(lpszQueryString, "CLT*", szTmp,
sizeof(szTmp)) )
                                        return
ERR_ORDERSTATUS_MISSING_CLT_KEY;
                        if ( szTmp[0] )
                                        return
ERR_ORDERSTATUS_CID_AND_CLT;
        }

        return ERR_SUCCESS;
}

/* FUNCTION: BOOL ReadRegistrySettings(void)
 *
 * PURPOSE:             This function reads the NT registry for startup
parameters. There parameters are
 *                                      under the TPCC key.
 *
 * ARGUMENTS:   None
 *
 * RETURNS:                     None
 *
 * COMMENTS:            This function also sets up required operation
variables to 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 ( !stricmp(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 ( !stricmp(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	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			Pointer to storage for the data. If
rgbValue is a null pointer, the
 *
					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                     main(int argc, char *argv[]);
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                     iTotalConnections;
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", iTotalConnections);
                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<iTotalConnections; 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", iTotalConnections);
                paktc();

                for(i=0; i<iTotalConnections; 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()
 *
		*pTermId		terminal id array
 *
		*pSyncId		sync id array
 *
 * RETURNS:			0 if successfull or error code if an error
occurs.
 *
 * COMMENTS:		The TermID and SyncID returned are provieded to the
logout function
 *
 *
 */

int Login(HINTERNET	hInternet, int *pTermId, int *pSyncId)
{
		int
		int
		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()
 *
		iTermId		terminal id array, provided from Login
API.
 *
		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()
 *
		*szUrl		URL string passed to the internet
service
 *
		**ppBuffer	pointer to dynamically allocated buffer containing
 *
the result HTML page read
from the URL.
 *
		*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;
        iTotalConnections = 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 (
!(iTotalConnections = atoi(argv[i]+2)) )

        iTotalConnections = 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.                     local  \n");
        printf("-I Internet server for test run.(Required)          none
\n");
        printf("-C Total connections for test run.                  100
\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.%2.2d.%3.3d                       *\n",
versionMS, versionMM, versionLS);
        printf("*                                               *\n");
        printf("*********************************************************\n\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

create database tpcc on
                                c_cs1_dev    =  6744,
                                c_cs2_dev    =  6744,
                                c_cs3_dev    =  6744,
                                c_cs4_dev    =  6744,
                                c_cs5_dev    =  4002,
                                c_cs6_dev    =  3000,
                                c_cs1_dev    =  380,
                                c_cs2_dev    =  380,
                                c_cs3_dev    =  380,
                                c_cs4_dev    =  380,
                                c_cs5_dev    =  380,
                                c_cs6_dev    =  2560,

                                c_ol1_dev    =  2647,
                                c_ol2_dev    =  2647,
                                c_ol3_dev    =  2647,
                                c_ol4_dev    =  2647,
                                c_ol5_dev    =  4320,

                                c_misc1_dev  =  380,
                                c_misc2_dev  =  380,
                                c_misc3_dev  =  380,
                                c_misc4_dev  =  380,
                                c_misc5_dev  =  1024

                log on c_log1_dev =  8000
go
```

DISKINIT.SQL

```
//*  TPC-C Benchmark Kit                                 */
/*                                                      */
/*  DISKINIT.SQL                                        */
/*                                                      */
/*  This script is used create the database devices for a 500     */
/*  warehouse database.                                 */
/*  NOTE! This version of DISKINIT.SQL assumes that you are using */
/*  some form of NT partitioning. If you wish to use raw          */
/*  partitions, YOU MUST SPECIFY A DRIVE LETTER ONLY for the      */
/*  physname parm! Raw partitions will not accept a file name.    */
/*  Also note that use of a drive letter only for the physname    */
/*  parm will result in corruption of any normal NT partition!    */

use master
go

disk init name = "c_log1_dev",
                physname = "E:",
                vdevno   = 14,
                size     = 4096000
go

/* disk init name = "c_log2_dev", */
/*              physname = "F:",            */
/*              vdevno   = 15, */
/*              size     = 4096000 */
/* go */
/* */
/* disk init name = "c_log3_dev", */
/*              physname = "G:",            */
/*              vdevno   = 16,              */
/*              size     = 4096000          */
/* go*/

disk init name = "c_cs1_dev",
                physname = "H:",
                vdevno   = 17,
                size     = 3611648
go

disk init name = "c_cs2_dev",
                physname = "I:",
                vdevno   = 18,
                size     = 3611648
go

disk init name = "c_cs3_dev",
                physname = "J:",
                vdevno   = 19,
                size     = 3611648

disk init name = "c_cs4_dev",
                physname = "K:",
                vdevno   = 20,
                size     = 3611648
go

disk init name = "c_cs5_dev",
                physname = "L:",
                vdevno   = 21,
                size     = 2243584
go

disk init name = "c_cs6_dev",
                physname = "M:",
                vdevno   = 22,
                size     = 2846720
go

disk init name = "c_ol1_dev",
                physname = "N:",
                vdevno   = 23,
                size     = 1355264
go

disk init name = "c_ol2_dev",
                physname = "O:",
                vdevno   = 24,
                size     = 1355264
go

disk init name = "c_ol3_dev",
                physname = "P:",
                vdevno   = 25,
                size     = 1355264
go

disk init name = "c_ol4_dev",
                physname = "Q:",
                vdevno   = 26,
                size     = 1355264
go

disk init name = "c_ol5_dev",
                physname = "R:",
                vdevno   = 27,
                size     = 2211840
go

disk init name = "c_misc1_dev",
                physname = "S:",
                vdevno   = 28,
                size     = 194560
go

disk init name = "c_misc2_dev",
                physname = "T:",
                vdevno   = 29,
                size     = 194560
go

disk init name = "c_misc3_dev",
                physname = "U:",
                vdevno   = 30,
                size     = 194560
go

disk init name = "c_misc4_dev",
                physname = "V:",
                vdevno   = 31,
                size     = 194560
go

disk init name = "c_misc5_dev",
                physname = "W:",
                vdevno   = 32,
                size     = 524288
go
```

SEGMENT.SQL

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


use tpcc
go

exec sp_addsegment cs_seg, c_cs1_dev
```

```
exec sp_extendsegment cs_seg, c_cs2_dev
exec sp_extendsegment cs_seg, c_cs3_dev
exec sp_extendsegment cs_seg, c_cs4_dev
exec sp_extendsegment cs_seg, c_cs5_dev
exec sp_extendsegment cs_seg, c_cs6_dev

exec sp_addsegment ol_seg, c_ol1_dev
exec sp_extendsegment ol_seg, c_ol2_dev
exec sp_extendsegment ol_seg, c_ol3_dev
exec sp_extendsegment ol_seg, c_ol4_dev
exec sp_extendsegment ol_seg, c_ol5_dev

exec sp_addsegment misc_seg, c_misc1_dev
exec sp_extendsegment misc_seg, c_misc2_dev
exec sp_extendsegment misc_seg, c_misc3_dev
exec sp_extendsegment misc_seg, c_misc4_dev
exec sp_extendsegment misc_seg, c_misc5_dev
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
```

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_c1' )
        drop index orders.orders_c1
go

select getdate()
go
create unique clustered index orders_c1 on orders(o_w_id, o_d_id, o_id)
        with sorted_data on misc_seg
go
select getdate()
go
```

IDXSTKCL.SQL

```
/*  TPC-C Benchmark Kit                                 */
/*                                                      */
/*  IDXSTKCL.SQL                                        */
/*                                                      */
/*  Creates clustered index on stock (seg)              */


use tpcc
go


if exists ( select name from sysindexes where  name = 'stock_c1' )
        drop index stock.stock_c1
go

select getdate()
go
create unique clustered index stock_c1 on stock(s_i_id, s_w_id)
        with sorted_data on 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_c1' )
        drop index warehouse.warehouse_c1
go

select getdate()
go
create unique clustered index warehouse_c1 on warehouse(w_id)
        with fillfactor=1 on misc_seg
go
select getdate()
go
```

DBOPT1.SQL

```
/*  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
```

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
```

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
```

TPCCBCP.SQL

```
/*  TPC-C Benchmark Kit                                 */
/*                                                      */
/*  TPCCBCP.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 Procedure

DELIVERY.SQL

```
/*  File:       DELIVERY.SQL                            */
/*              Microsoft TPC-C Kit Ver. 3.00.000       */
/*              Audited 08/23/96, By Francois Raab      */
/*                                                      */
/*              Copyright Microsoft, 1996               */
/*                                                      */
/*  Purpose:    Delivery transaction for Microsoft TPC-C Benchmark Kit  */
/*  Author:     Damien Lindauer                         */
/*              damienl@Microsoft.com                   */


use tpcc
go

/* delivery transaction */

if exists (select name from sysobjects where name = "tpcc_delivery" )
        drop procedure tpcc_delivery
go

create proc tpcc_delivery                       @w_id           smallint,

                                @o_carrier_id   smallint
```

```
as


declare @d_id tinyint,
        @o_id int,
        @c_id int,
        @total numeric(12,2),
        @oid1 int,
        @oid2 int,
        @oid3 int,
        @oid4 int,
        @oid5 int,
        @oid6 int,
        @oid7 int,
        @oid8 int,
        @oid9 int,
        @oid10 int

select @d_id = 0

begin tran d

    while (@d_id < 10)
    begin

        select @d_id  = @d_id + 1,
               @total = 0,
               @o_id  = 0

        select @o_id = min(no_o_id)
        from new_order holdlock
        where no_w_id = @w_id and
                                    no_d_id = @d_id

        if (@@rowcount <> 0)
        begin

            /* claim the order for this district */

            delete new_order
            where no_w_id = @w_id and
                                        no_d_id = @d_id and
                                                  no_o_id = @o_id

            /* set carrier_id on this order (and get customer id) */

            update orders
                                    set o_carrier_id = @o_carrier_id,
                @c_id       = o_c_id
            where o_w_id = @w_id and
                                        o_d_id = @d_id and
                                                 o_id   = @o_id

            /* set date in all lineitems for this order (and sum amounts) */

            update order_line
                                    set ol_delivery_d = getdate(),
                           @total       = @total + ol_amount
            where ol_w_id = @w_id and
                  ol_d_id = @d_id and
                                                 ol_o_id = @o_id

            /* accummulate 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
```

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,

                @i_id1  int = 0, @s_w_id1 smallint = 0, @ol_qty1 smallint = 0,

                @i_id2  int = 0, @s_w_id2 smallint = 0, @ol_qty2 smallint = 0,

                @i_id3  int = 0, @s_w_id3 smallint = 0, @ol_qty3 smallint = 0,

                @i_id4  int = 0, @s_w_id4 smallint = 0, @ol_qty4 smallint = 0,

                @i_id5  int = 0, @s_w_id5 smallint = 0, @ol_qty5 smallint = 0,

                @i_id6  int = 0, @s_w_id6 smallint = 0, @ol_qty6 smallint = 0,

                @i_id7  int = 0, @s_w_id7 smallint = 0, @ol_qty7 smallint = 0,

                @i_id8  int = 0, @s_w_id8 smallint = 0, @ol_qty8 smallint = 0,

                @i_id9  int = 0, @s_w_id9 smallint = 0, @ol_qty9 smallint = 0,

                @i_id10 int = 0, @s_w_id10 smallint = 0, @ol_qty10 smallint = 0,

                @i_id11 int = 0, @s_w_id11 smallint = 0, @ol_qty11 smallint = 0,

                @i_id12 int = 0, @s_w_id12 smallint = 0, @ol_qty12 smallint = 0,

                @i_id13 int = 0, @s_w_id13 smallint = 0, @ol_qty13 smallint = 0,

                @i_id14 int = 0, @s_w_id14 smallint = 0, @ol_qty14 smallint = 0,

                @i_id15 int = 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 availible order id and update */
/*  plus initialize local variables  */

    update district
                set @d_tax      = d_tax,
```

```
            @o_id       = d_next_o_id,
                        d_next_o_id = d_next_o_id + 1,
    @o_entry_d = getdate(),
    @li_no=0,
    @commit_flag = 1
            where d_w_id = @w_id and
                d_id   = @d_id

        /*  process orderlines */
while (@li_no < @o_ol_cnt)
        begin

    select @li_no = @li_no + 1

        /*  Set i_id, s_w_id, and qty for this lineitem */

        select @li_id = case @li_no
            when 1 then @i_id1
                        when 2 then @i_id2
                        when 3 then @i_id3
                        when 4 then @i_id4
                        when 5 then @i_id5
                        when 6 then @i_id6
                        when 7 then @i_id7
                        when 8 then @i_id8
                        when 9 then @i_id9
                        when 10 then @i_id10
                        when 11 then @i_id11
                        when 12 then @i_id12
                        when 13 then @i_id13
                        when 14 then @i_id14
                        when 15 then @i_id15
                        end,

                        @li_s_w_id = case @li_no
                        when 1 then @s_w_id1
                        when 2 then @s_w_id2
                        when 3 then @s_w_id3
                        when 4 then @s_w_id4
                        when 5 then @s_w_id5
                        when 6 then @s_w_id6
                        when 7 then @s_w_id7
                        when 8 then @s_w_id8
                        when 9 then @s_w_id9
                        when 10 then @s_w_id10
                        when 11 then @s_w_id11
                        when 12 then @s_w_id12
                        when 13 then @s_w_id13
                        when 14 then @s_w_id14
                        when 15 then @s_w_id15
                        end,

                        @li_qty = case  @li_no
                        when 1 then @ol_qty1
                        when 2 then @ol_qty2
                        when 3 then @ol_qty3
                        when 4 then @ol_qty4
                        when 5 then @ol_qty5
                        when 6 then @ol_qty6
                        when 7 then @ol_qty7
                        when 8 then @ol_qty8
                        when 9 then @ol_qty9
                        when 10 then @ol_qty10
                        when 11 then @ol_qty11
                        when 12 then @ol_qty12
                        when 13 then @ol_qty13
                        when 14 then @ol_qty14
                        when 15 then @ol_qty15
                                    end

    /* get item data (no one updates item) */

    select @i_price = i_price,
          @i_name  = i_name,
          @i_data  = i_data
    from item (tablock holdlock)
                where i_id = @li_id

                /* if there actually is an item with this id, go to work */

                            if (@@rowcount > 0)
                            begin
        update stock set s_ytd       = s_ytd + @li_qty,
                    @s_quantity  = s_quantity,
                    s_quantity   = s_quantity - @li_qty +
                       case when (s_quantity - @li_qty < 10) then 91 else 0 end,
                    s_order_cnt  = s_order_cnt + 1,
                    s_remote_cnt = s_remote_cnt + case
                       when (@li_s_w_id = @w_id) then 0 else 1 end,
                    @s_data      = s_data,
                    @s_dist      = case @d_id
                                                    when 1  then s_dist_01
                                                    when 2  then s_dist_02
                                                    when 3  then s_dist_03
                                                    when 4  then s_dist_04
                                                    when 5  then s_dist_05
                                                    when 6  then s_dist_06
                                                    when 7  then s_dist_07
                                                    when 8  then s_dist_08
                                                    when 9  then s_dist_09
                                                    when 10 then s_dist_10
                                        end
        where s_i_id = @li_id and
                                            s_w_id = @li_s_w_id

                                        /* insert order_line data (using data from
item and stock) */

            insert into order_line values(@o_id,            /* from district update */
                            @d_id,                          /* input param          */
                            @w_id,          /* input param          */
                            @li_no,         /* orderline number     */
                            @li_id,         /* lineitem id          */
                            @li_s_w_id,     /* lineitem warehouse   */
                            "jan 1, 1900",  /* constant             */
                            @li_qty,        /* lineitem qty         */
                            @i_price * @li_qty, /* ol_amount        */
                            @s_dist)        /* from stock           */

        /* send line-item data to client */

                                select @i_name,
            @s_quantity,
            b_g = case when ( (patindex("%ORIGINAL%",@i_data) > 0) and
                                (patindex("%ORIGINAL%",@s_data) > 0) )
                then "B" else "G" end,
            @i_price,
            @i_price * @li_qty

        end
        else
                            begin

                                /*  no item found - triggers rollback condition */
                                        select "",0,"",0,0
                                        select @commit_flag = 0

                            end
                end

    /* get customer last name, discount, and credit rating */

    select @c_last     = c_last,
          @c_discount = c_discount,
          @c_credit   = c_credit,
                                @c_id_local = c_id
    from customer holdlock
    where c_id   = @c_id and
          c_w_id = @w_id and
          c_d_id = @d_id

    /* insert fresh row into orders table */

    insert into orders values (@o_id,
                                        @d_id,
                                            @w_id,

@c_id_local,

@o_entry_d,
                                                                                0,

@o_ol_cnt,

@o_all_local)

                /* insert corresponding row into new-order table */

                insert into new_order values (@o_id,

                            @d_id,

                            @w_id)

                /*  select warehouse tax */

                select @w_tax = w_tax
                from warehouse holdlock
                where w_id = @w_id

                if (@commit_flag = 1)
                            commit transaction n
                else
                            /* all that work for nuthin!!! */
                            rollback transaction n

                /* return order data to client */
                select @w_tax,
                      @d_tax,
                                @o_id,
                                @c_last,
                                @c_discount,
                                @c_credit,
                                @o_entry_d,
                                @commit_flag

end

go
```

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

                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
```

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 = 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
```

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,

                @threshhold                 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 <  @threshhold
go
```

Loader Source Code

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);
                dbclose(i_dbproc1);

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

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

                InterlockedIncrement(&main_threads_completed);
}


//============================================================
============
//
// Function   : LoadWarehouse
//
// Loads WAREHOUSE table and loads Stock and District as Warehouses are
created
//
//============================================================
============

void LoadWarehouse()
{
    short  w_id;
    char   w_name[W_NAME_LEN+1];
    char   w_street_1[ADDRESS_LEN+1];
    char   w_street_2[ADDRESS_LEN+1];
    char   w_city[ADDRESS_LEN+1];
    char   w_state[STATE_LEN+1];
    char   w_zip[ZIP_LEN+1];
    double w_tax;
    double w_ytd;
                char   name[20];
                long   time_start;


                printf("\nLoading warehouse table...\n");

                // Seed with unique number
                seed(2);

                InitString(w_name, W_NAME_LEN+1);
                InitAddress(w_street_1, w_street_2, w_city, w_state, w_zip);

                sprintf(name, "%s..%s", aptr->database, "warehouse");
                bcp_init(w_dbproc1, name, NULL, "logs\\whouse.err", DB_IN);

                bcp_bind(w_dbproc1, (BYTE *) &w_id,        0, -1,          NULL, 0,
0, 1);
                bcp_bind(w_dbproc1, (BYTE *) w_name,       0, W_NAME_LEN,
NULL, 0, 0, 2);
                bcp_bind(w_dbproc1, (BYTE *) w_street_1,   0, ADDRESS_LEN,
NULL, 0, 0, 3);
                bcp_bind(w_dbproc1, (BYTE *) w_street_2,   0, ADDRESS_LEN,
NULL, 0, 0, 4);
                bcp_bind(w_dbproc1, (BYTE *) w_city,       0, ADDRESS_LEN,
NULL, 0, 0, 5);
                bcp_bind(w_dbproc1, (BYTE *) w_state,      0, STATE_LEN,
NULL, 0, 0, 6);
                bcp_bind(w_dbproc1, (BYTE *) w_zip,        0, ZIP_LEN,     NULL,
0, 0, 7);
                bcp_bind(w_dbproc1, (BYTE *) &w_tax,       0, -1,          NULL, 0,
SQLFLT8, 8);
                bcp_bind(w_dbproc1, (BYTE *) &w_ytd,       0, -1,          NULL, 0,
SQLFLT8, 9);

                time_start = (TimeNow() / MILLI);

                warehouse_rows_loaded = 0;

                for (w_id = aptr->starting_warehouse; w_id < aptr-
>num_warehouses+1; w_id++)
                {

                        MakeAlphaString(6,10, W_NAME_LEN, w_name);

                        MakeAddress(w_street_1, w_street_2, w_city,
w_state, w_zip);

                        w_tax = ((float)
RandomNumber(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);
                dbclose(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,

                          (LPTHREAD_START_ROUTINE)
LoadOrdersTable,

                          &orders_time_start,

                          0,

                          &dwThreadID[0]);

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

                        // start NewOrder table thread

                        printf("...Loading New-Order Table for:
d_id = %d, w_id = %d\n", d_id, w_id);

                        hThread[1] = CreateThread(NULL,

                          0,

                          (LPTHREAD_START_ROUTINE)
LoadNewOrderTable,

                          &new_order_time_start,

                          0,

                          &dwThreadID[1]);

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

                        // 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,

                          (LPTHREAD_START_ROUTINE)
LoadOrderLineTable,

                          &order_line_time_start,

                          0,

                          &dwThreadID[2]);

                        if (hThread[2] == NULL)
                        {
                                printf("Error, failed in
creating creating thread = 2.\n");
                                exit(-1);
                        }

                        while (order_threads_completed != 3)
                                Sleep(1000L);

                }
        }

        printf("Finished loading orders.\n");

        InterlockedIncrement(&main_threads_completed);

    return;
}


//=============================================================
============
//
// Function   : OrdersBufInit
//
// Clears shared buffer for ORDERS, NEWORDER, and ORDERLINE
//
//=============================================================
============

void OrdersBufInit()
{
        int     i;
        int                     j;

        for (i=0;i<ORDERS_PER_DISTRICT;i++)
        {
                        orders_buf[i].o_id = 0;
                        orders_buf[i].o_d_id = 0;
                        orders_buf[i].o_w_id = 0;
                        orders_buf[i].o_c_id = 0;
```

```
                        orders_buf[i].o_carrier_id = 0;
                        orders_buf[i].o_ol_cnt = 0;
                        orders_buf[i].o_all_local = 0;

                        for (j=0;j<=14;j++)
                        {
                                orders_buf[i].o_ol[j].ol = 0;
                                orders_buf[i].o_ol[j].ol_i_id = 0;
                                orders_buf[i].o_ol[j].ol_supply_w_id = 0;
                                orders_buf[i].o_ol[j].ol_quantity = 0;
                                orders_buf[i].o_ol[j].ol_amount = 0;

        strcpy(orders_buf[i].o_ol[j].ol_dist_info,"");
                        }
        }



}


//=============================================================
============
//
// Function   : OrdersBufLoad
//
// Fills shared buffer for ORDERS, NEWORDER, and ORDERLINE
//
//=============================================================
============

void OrdersBufLoad(int d_id, int w_id)
{

        int     cust[ORDERS_PER_DIST+1];
        long    o_id;
        short   ol;

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

        GetPermutation(cust, ORDERS_PER_DIST);

        for (o_id=0;o_id<ORDERS_PER_DISTRICT;o_id++)
        {

                // Generate ORDER and NEW-ORDER data

                orders_buf[o_id].o_d_id = d_id;
                orders_buf[o_id].o_w_id = w_id;
                orders_buf[o_id].o_id = o_id+1;
                orders_buf[o_id].o_c_id = cust[o_id+1];
                orders_buf[o_id].o_ol_cnt = RandomNumber(5L,
15L);

                if (o_id < 2100)
                {
                        orders_buf[o_id].o_carrier_id =
RandomNumber(1L, 10L);
                        orders_buf[o_id].o_all_local  = 1;
                }
                else
                {
                        orders_buf[o_id].o_carrier_id = 0;
                        orders_buf[o_id].o_all_local  = 1;
                }

                for (ol=0;ol<orders_buf[o_id].o_ol_cnt;ol++)
                        {

                                orders_buf[o_id].o_ol[ol].ol = ol+1;
                                orders_buf[o_id].o_ol[ol].ol_i_id =
RandomNumber(1L, MAXITEMS);
                                orders_buf[o_id].o_ol[ol].ol_supply_w_id
= w_id;
                                orders_buf[o_id].o_ol[ol].ol_quantity = 5;
                                MakeAlphaString(24, 24,
OL_DIST_INFO_LEN, &orders_buf[o_id].o_ol[ol].ol_dist_info);

                                // Generate ORDER-LINE data
                                if (o_id < 2100)
                                {

                orders_buf[o_id].o_ol[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);
   }

        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);

        }
```

```
		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);
			}

		}

		if ((o_w_id == aptr->num_warehouses) && (o_d_id == 10))
		{
			bcp_done(o_dbproc3);
			dbclose(o_dbproc3);

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

		InterlockedIncrement(&order_threads_completed);

}


//=============================================================
============
//
// Function   : GetPermutation
//
//=============================================================
============

void GetPermutation(int perm[], int n)
{
		int i, r, t;

		for (i=1;i<=n;i++)
			perm[i] = i;

		for (i=1;i<=n;i++)
		{
			r = RandomNumber(i,n);
			t = perm[i];
			perm[i] = perm[r];
			perm[r] = t;
		}

}


//=============================================================
============
//
// Function   : CheckForCommit
//
//=============================================================
============

void CheckForCommit(DBPROCESS *dbproc,
					int rows_loaded,
					char
*table_name,
					long *time_start)
{

   long		time_end, time_diff;

   // commit every "batch" rows

   if ( !(rows_loaded % 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 (%.2f rps)\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: SQLErrHandler
//
//============================================================
============

int SQLErrHandler(SQLCONN *dbproc,
                  int    severity,
                  int    err,
                  int    oserr,
                  char   *dberrstr,
                  char   *oserrstr)
{

        char msg[256];
        FILE *fp1;
        char timebuf[128];
        char datebuf[128];

        _strtime(timebuf);
        _strdate(datebuf);

        sprintf(msg, "%s %s : DBLibrary (%ld) %s\n", datebuf, timebuf, err,
dberrstr);
   printf("%s",msg);

        fp1 = fopen("logs\tpccldr.err","a");
        if (fp1 == NULL)
        {
                printf("Error in opening errorlog file.\n");
        }
        else
        {
                fprintf(fp1, msg);
                fclose(fp1);
        }

   if (oserr != DBNOERR)
   {
                sprintf(msg, "%s %s : OSError (%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                                 all 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 throught */
                                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 = atol(ptr+2);
                                                                break;
```

```
                                        case 's':
                                                pClient-
>synch_servername = ptr+2;
                                                break;

                                        case 'q':
                                                pClient-
>enable_qj = atol(ptr+2);
                                                break;

                                        default:

                GetArgsClientUsage();
                                                exit(-1);
                                                break;
                        }
                        }
                }

    return;
}


//============================================================
============
//
// Function name: GetArgsClientUsage
//
//============================================================
============

void GetArgsClientUsage()
{

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

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

    exit(0);
}


//============================================================
============
//
// Function name: GetArgsDelivery
//
//============================================================
============
void GetArgsDelivery(int argc, char **argv, DELIVERY_ARGS *pDelivery)
{
    int                 i;
    char        *ptr;

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

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\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");

   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\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-
>OlOrderStatusData[i].ol_supply_w_id,
                                (int) pOrderStatus-
>OlOrderStatusData[i].ol_i_id,
                                (int) pOrderStatus-
>OlOrderStatusData[i].ol_delivery_d.month,
                                (int) pOrderStatus-
>OlOrderStatusData[i].ol_delivery_d.day,
                                (int) pOrderStatus-
>OlOrderStatusData[i].ol_delivery_d.year,
                                (int) pOrderStatus-
>OlOrderStatusData[i].ol_quantity,
                                (double) pOrderStatus-
>OlOrderStatusData[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);

			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);
			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);

}
```

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

//Orginal 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;
}
```

STRINGS.C

```
/*              FILE:                   STRINGS.C
 *                                                      Microsoft TPC-C Kit Ver.
3.00.000
```

```
 *                                                      Audited 08/23/96, By
Francois Raab
 *
 *                                                      Copyright Microsoft, 1996
 *
 *              PURPOSE:   String generation functions for Microsoft TPC-C
Benchmark Kit
 *              Author:                 Damien Lindauer
 *                                                      damienl@Microsoft.com
 */

// Includes
#include "tpcc.h"
#include <string.h>
#include <ctype.h>

//============================================================
============
//
// Function name: MakeAddress
//
//============================================================
============

void MakeAddress(char *street_1,
                                                      char *street_2,
                                                      char *city,
                                                      char *state,
                                                      char *zip)
{

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

   MakeAlphaString (10, 20, ADDRESS_LEN, street_1);
   MakeAlphaString (10, 20, ADDRESS_LEN, street_2);
   MakeAlphaString (10, 20, ADDRESS_LEN, city);
   MakeAlphaString ( 2,  2, STATE_LEN, state);
   MakeZipNumberString( 9,  9, ZIP_LEN, zip);

#ifdef DEBUG
   printf("[%ld]DBG: MakeAddress: street_1: %s, street_2: %s, city: %s, state:
%s, zip: %s\n",
                                      (int) GetCurrentThreadId(), street_1,
street_2, city, state, zip);
#endif

   return;

}


//============================================================
============
//
// Function name: LastName
//
//============================================================
============

void LastName(int  num,
                                char *name)
{
   int                                  i;
            int                                 len;
   static char *n[] =
   {
                                        "BAR" , "OUGHT", "ABLE" , "PRI"  , "PRES",
                                        "ESE" , "ANTI" , "CALLY", "ATION", "EING"
   };

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

   if ((num >= 0) && (num < 1000))
   {
                                        strcpy(name, n[(num/100)%10]);
                                        strcat(name, n[(num/10)%10]);
                                        strcat(name, n[(num/1)%10]);

                                        if (strlen(name) < LAST_NAME_LEN)
                                        {
                                                PaddString(LAST_NAME_LEN, name);
                                        }

   }
   else
   {
                                        printf("\nError in LastName()... num <%ld> out of
range (0,999)\n", num);
                                        exit(-1);
   }


#ifdef DEBUG
   printf("[%ld]DBG: LastName: num = [%d] ==> [%d][%d][%d]\n",
                                                (int) GetCurrentThreadId(), num,
num/100, (num/10)%10, num%10);
            printf("[%ld]DBG: LastName: String = %s\n", (int)
GetCurrentThreadId(), name);
#endif

   return;

}


//============================================================
============
//
// Function name: MakeAlphaString
//
//============================================================
============

//philipdu 08/13/96 Changed MakeAlphaString to use A-Z, a-z, and 0-9 in
//accordance with spec see below:
//The spec says:
//4.3.2.2       The notation random a-string [x .. y]
//(respectively, n-string [x .. y]) represents a string of random alphanumeric
//(respectively, numeric) characters of a random length of minimum x, maximum
y,
//and mean (y+x)/2. Alphanumerics are A..Z, a..z, and 0..9.  The only other
//requirement is that the character set used "must be able to represent a
minimum
//of 128 different characters".  We are using 8-bit chars, so this is a non issue.
//It is completely unreasonable to stuff non-printing chars into the text fields.
//-CLevine 08/13/96

int MakeAlphaString( int  x, int  y, int  z, char *str)
{
            int                         len;
            int                         i;
            static          char chArray[] =
"0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxy
z";
            static          int             chArrayMax = 61;

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

            len= RandomNumber(x, y);

            for (i=0; i<len; i++)
                        str[i] = chArray[RandomNumber(0, chArrayMax)];
            if ( len < z )
                        memset(str+len, ' ', z - len);
            str[len] = 0;

   return len;
}

#if 0
//philipdu 08/13/96 Orginal MakeAlphaString

int MakeAlphaString( int  x,
                                                                  int  y,
                                                                        int  z,
                                                                  char *str)
{

            int                         len;
            int                         i;

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

            len= RandomNumber(x, y);

            for (i=0; i<len; i++)
            {
                        str[i] = RandomNumber(MINPRINTASCII,
MAXPRINTASCII);
            }

            str[len] = '\0';

            if (len < z)
            {
                        PaddString(z, str);
            }

   return (len);
}
#endif


//============================================================
============
//
// Function name: MakeOriginalAlphaString
//
//============================================================
============

int MakeOriginalAlphaString(int  x,

int  y,
```

```
                int  z,
char *str,
                int  percent)
{
   int                  len;
   int                  val;
   int                  start;

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

   // verify prercentage is valid
   if ((percent < 0) || (percent > 100))
   {
                printf("MakeOrigianlAlphaString: Invalid percentage:
%d\n", percent);
                exit(-1);
   }

   // verify string is at least 8 chars in length
   if ((x + y) <= 8)
   {
                printf("MakeOriginalAlphaString: string length must
be >= 8\n");
                exit(-1);
   }

   // Make Alpha String
   len = MakeAlphaString(x,y, z, str);

   val = RandomNumber(1,100);
   if (val <= percent)
   {
                start = RandomNumber(0, len - 8);
                strncpy(str + start, "ORIGINAL", 8);
   }


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

   return strlen(str);
}


//=============================================================
============
//
// Function name: MakeNumberString
//
//=============================================================
============
int MakeNumberString(int  x, int  y, int  z, char *str)
{
                char tmp[16];

                //MakeNumberString is always called MakeZipNumberString(16,
16, 16, string)

                memset(str, '0', 16);
                itoa(RandomNumber(0, 99999999), tmp, 10);
                memcpy(str, tmp, strlen(tmp));

                itoa(RandomNumber(0, 99999999), tmp, 10);
                memcpy(str+8, tmp, strlen(tmp));

                str[16] = 0;

   return 16;
}

#if 0
int MakeNumberString(int  x,
                                                int  y,
                                                        int  z,
                                                char *str)
{
   int                  len;
   int                  i;

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

   len = RandomNumber(x,y);


   for (i=0; i < len; i++)
   {
                str[i] = (char) (RandomNumber(48,57));
   }

   str[len] = '\0';

                PaddString(z, str);

   return strlen(str);
}

#endif

//=============================================================
============
//
// Function name: MakeZipNumberString
//
//=============================================================
============
int MakeZipNumberString(int  x, int  y, int  z, char *str)
{
                char tmp[16];

                //MakeZipNumberString is always called MakeZipNumberString(9,
9, 9, string)

                strcpy(str, "000011111");

                itoa(RandomNumber(0, 9999), tmp, 10);
                memcpy(str, tmp, strlen(tmp));

   return 9;
}

#if 0
//pgd 08/14/96 Orginal Code Below
int MakeZipNumberString(int  x,
                                                        int  y,
                int  z,
                                                                char *str)
{
   int                  len;
   int                  i;

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

   len = RandomNumber(x-5,y-5);

   for (i=0; i < len; i++)
   {
                str[i] = (char) (RandomNumber(48,57));
   }

   str[len] = '\0';

                strcat(str, "11111");

                PaddString(z, str);

   return strlen(str);
}
#endif

//=============================================================
============
//
// Function name: InitString
//
//=============================================================
============
void InitString(char *str, int len)
{
   int i;

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

                memset(str, ' ', len);
   str[len] = 0;
}

#if 0
//Orginal pgd 08/14/96
void InitString(char *str, int len)
{
   int i;

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


   for (i=0; i< len; i++)
                str[i] = ' ';
   str[len] = '\0';
}

#endif

//=============================================================
============
```

```
// Function name: InitAddress
//
// Description:
//
//============================================================
============

void InitAddress(char *street_1, char *street_2, char *city, char *state, char *zip)
{
        int i;


        memset(street_1, ' ', ADDRESS_LEN+1);
        memset(street_2, ' ', ADDRESS_LEN+1);
        memset(city, ' ', ADDRESS_LEN+1);

    street_1[ADDRESS_LEN+1] = 0;
    street_2[ADDRESS_LEN+1] = 0;
    city[ADDRESS_LEN+1]     = 0;

        memset(state, ' ', STATE_LEN+1);
    state[STATE_LEN+1] = 0;

        memset(zip, ' ', ZIP_LEN+1);
    zip[ZIP_LEN+1] = 0;
}

#if 0
//Orginal pgd 08/14/96
void InitAddress(char *street_1,
                                                        char *street_2,
                                                        char *city,
                                                        char *state,
                                                        char *zip)
{
        int i;

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

    for (i=0; i< ADDRESS_LEN+1; i++)
    {
                                street_1[i] = ' ';
                                street_2[i] = ' ';
                                city[i]     = ' ';
    }

    street_1[ADDRESS_LEN+1] = '\0';
    street_2[ADDRESS_LEN+1] = '\0';
    city[ADDRESS_LEN+1]     = '\0';

    for (i=0; i< STATE_LEN+1; i++)
                                state[i] = ' ';
    state[STATE_LEN+1] = '\0';

    for (i=0; i< ZIP_LEN+1; i++)
                                zip[i] = ' ';
    zip[ZIP_LEN+1] = '\0';
}

#endif
//============================================================
============
//
// Function name: PaddString
//
//============================================================
============

void PaddString(int max, char *name)
{
        int                     i;
        int                     len;

        len = strlen(name);
        if ( len < max )
                memset(name+len, ' ', max - len);
        name[max] = 0;

        return;
}

#if 0
        //pgd 08/14/96 Orginal code below
        void PaddString(int                     max,
                                                        char
        *name)
        {

                int                     i;
                int                     len;

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

                len = strlen(name);

                for (i=1;i<=(max - len);i++)
                {
                        strcat(name, " ");
                }

        }
#endif
```

Appendix C : RTE Input Parameters

The following parameters were used with Microsoft BenchCraft RTE..

Profile: 6300
File Path: E:\benchcrf\6300.pro
Version: 1.0.1

Number of Engines: 5

Name: DRIVER1
Description:
Directory: \RTE001.OUT
Machine: RTE001
Parameter Set: ~Default
Index: 0
Seed: 81242
Configured Users: 1260
Pipe Name: DRIVER111063125
Connect Rate: 200
Start Rate: 0
CLIENT_NURAND: 233
CPU: 0

Name: DRIVER2
Description:
Directory: \RTE002.OUT
Machine: RTE002
Parameter Set: ~Default
Index: 100000
Seed: 81242
Configured Users: 1260
Pipe Name: DRIVER211089578
Connect Rate: 200
Start Rate: 0
CLIENT_NURAND: 233
CPU: 0

Name: DRIVER3
Description:
Directory: \RTE003.OUT
Machine: RTE003
Parameter Set: ~Default
Index: 200000
Seed: 81242
Configured Users: 1260
Pipe Name: DRIVER311107937
Connect Rate: 200
Start Rate: 0
CLIENT_NURAND: 233
CPU: 0

Name: DRIVER4
Description:
Directory: \RTE004.OUT
Machine: RTE004
Parameter Set: ~Default
Index: 300000
Seed: 81242
Configured Users: 1260
Pipe Name: DRIVER411123546
Connect Rate: 200
Start Rate: 0
CLIENT_NURAND: 233
CPU: 0

Name: DRIVER5
Description:
Directory: \RTE005.OUT
Machine: RTE005
Parameter Set: ~Default
Index: 400000
Seed: 81242
Configured Users: 1260
Pipe Name: DRIVER511140390
Connect Rate: 200
Start Rate: 0
CLIENT_NURAND: 233
CPU: 0

Number of User groups: 10

Driver Engine: DRIVER1
IIS Server: IIS0011
SQL Server: U99C
User: sa
Protocol: Html
w_id Range: 1 - 63
w_id Max Warehouse: 630
Scale: Normal
User Count: 630
District id: 1
Scale Down: No

Driver Engine: DRIVER5
IIS Server: IIS0051
SQL Server: U99C
User: sa
Protocol: Html
w_id Range: 505 - 567
w_id Max Warehouse: 630
Scale: Normal
User Count: 630
District id: 1
Scale Down: No

Driver Engine: DRIVER5
IIS Server: IIS0052
SQL Server: U99C
User: sa
Protocol: Html
w_id Range: 568 - 630
w_id Max Warehouse: 630
Scale: Normal
User Count: 630
District id: 1
Scale Down: No

Driver Engine: DRIVER1
IIS Server: IIS0012
SQL Server: U99C
User: sa
Protocol: Html
w_id Range: 64 - 126
w_id Max Warehouse: 630
Scale: Normal
User Count: 630
District id: 1
Scale Down: No

Driver Engine: DRIVER2
IIS Server: IIS0021
SQL Server: U99C
User: sa
Protocol: Html
w_id Range: 127 - 189
w_id Max Warehouse: 630
Scale: Normal
User Count: 630
District id: 1
Scale Down: No

Driver Engine: DRIVER2
IIS Server: IIS0022
SQL Server: U99C
User: sa
Protocol: Html
w_id Range: 190 - 252
w_id Max Warehouse: 630
Scale: Normal
User Count: 630
District id: 1

Scale Down: No

Driver Engine: DRIVER3
IIS Server: IIS0031
SQL Server: U99C
User: sa
Protocol: Html
w_id Range: 253 - 315
w_id Max Warehouse: 630
Scale: Normal
User Count: 630
District id: 1
Scale Down: No

Driver Engine: DRIVER3
IIS Server: IIS0032
SQL Server: U99C
User: sa
Protocol: Html
w_id Range: 316 - 378
w_id Max Warehouse: 630
Scale: Normal
User Count: 630
District id: 1
Scale Down: No

Driver Engine: DRIVER4
IIS Server: IIS0041
SQL Server: U99C
User: sa
Protocol: Html
w_id Range: 379 - 441
w_id Max Warehouse: 630
Scale: Normal
User Count: 630
District id: 1
Scale Down: No

Driver Engine: DRIVER4
IIS Server: IIS0042
SQL Server: U99C
User: sa
Protocol: Html
w_id Range: 442 - 504
w_id Max Warehouse: 630
Scale: Normal
User Count: 630
District id: 1
Scale Down: No

Number of Parameter Sets: 2

PARAM2
New Parameter Set

	Txn Weight	Think Time	Key Time	RT Delay	RT Menu	Fence	Delay
New Order	44.60	12.07	18.02	0.10	0.10	5.00	
Payment	43.10	12.07	3.02	0.10	0.10	5.00	
Delivery	4.10	5.07	2.02	0.10	0.10	5.00	
Stock Level	4.10	5.07	2.02	0.10	0.10	20.00	
Order Status	4.10	10.07	2.02	0.10	0.10	5.00	

~Default
Default Parameter Set

	Txn Weight	Think Time	Key Time	RT Delay	RT Menu	Fence	Delay
New Order	44.80	12.07	18.02	0.10	0.10	5.00	
Payment	43.05	12.07	3.02	0.10	0.10	5.00	
Delivery	4.05	5.07	2.02	0.10	0.10	5.00	
Stock Level	4.05	5.07	2.02	0.10	0.10	20.00	
Order Status	4.05	10.07	2.02	0.10	0.10	5.00	

Appendix D : Tunable Parameters

<Server Configuration>

Microsoft Windows NT Server version 4.0 Configuration Parameters

The following services were disabled in the Windows NT Control Panel/Service:

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

NT Registry

No Windows NT registry parameters were modified for this benckmark.

<Client Configuration>

IIS Registry

Key Name: SYSTEM\CurrentControlSet\Services\InetInfo
Class Name: <NO CLASS>
Last Write Time: 4/11/97 - 6:15 PM

Key Name:
SYSTEM\CurrentControlSet\Services\InetInfo\Parameters
Class Name: <NO CLASS>
Last Write Time: 7/16/97 - 11:27 AM
Value 0
Name: BandwidthLevel
Type: REG_DWORD
Data: 0xffffffff

Value 1
Name: ListenBackLog
Type: REG_DWORD
Data: 0x800

Value 2
Name: PoolThreadLimit
Type: REG_DWORD
Data: 0x400

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

Key Name:
SYSTEM\CurrentControlSet\Services\InetInfo\Parameters\Filter
Class Name: <NO CLASS>
Last Write Time: 4/11/97 - 6:15 PM
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: 4/11/97 - 6:15 PM
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/winhlp,hlp,,5
Type: REG_SZ
Data:

Value 14
Name: application/x-bcpio,bcpio,,5
Type: REG_SZ
Data:

Value 15
Name: application/x-cpio,cpio,,5
Type: REG_SZ
Data:

Value 16
Name: application/x-csh,csh,,5
Type: REG_SZ
Data:

Value 17
Name: application/x-director,dcr,,5
Type: REG_SZ
Data:

Value 18
Name: application/x-director,dir,,5
Type: REG_SZ
Data:

Value 19
Name: application/x-director,dxr,,5
Type: REG_SZ
Data:

Value 20
Name: application/x-dvi,dvi,,5
Type: REG_SZ
Data:

Value 21
Name: application/x-gtar,gtar,,9
Type: REG_SZ
Data:

Value 22
Name: application/x-hdf,hdf,,5
Type: REG_SZ
Data:

Value 23
Name: application/x-latex,latex,,5
Type: REG_SZ
Data:

Value 24
Name: application/x-msaccess,mdb,,5
Type: REG_SZ
Data:

Value 25
Name: application/x-mscardfile,crd,,5
Type: REG_SZ
Data:

Value 26
Name: application/x-msclip,clp,,5
Type: REG_SZ
Data:

Value 27
Name: application/x-msexcel,xla,,5
Type: REG_SZ
Data:

Value 28
Name: application/x-msexcel,xlc,,5
Type: REG_SZ
Data:

Value 29
Name: application/x-msexcel,xlm,,5
Type: REG_SZ
Data:

Value 30
Name: application/x-msexcel,xls,,5
Type: REG_SZ
Data:

Value 31
Name: application/x-msexcel,xlt,,5
Type: REG_SZ
Data:

Value 32
Name: application/x-msexcel,xlw,,5
Type: REG_SZ
Data:

Value 33
Name: application/x-msmediaview,m13,,5
Type: REG_SZ
Data:

Value 34
Name: application/x-msmediaview,m14,,5
Type: REG_SZ
Data:

Value 35
Name: application/x-msmetafile,wmf,,5
Type: REG_SZ
Data:

Value 36
Name: application/x-msmoney,mny,,5
Type: REG_SZ
Data:

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-msterminal,trm,,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-xpixmap,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:

Key Name:
SYSTEM\CurrentControlSet\Services\InetInfo\Performance
Class Name: <NO CLASS>
Last Write Time: 4/11/97 - 6:15 PM
Value 0
Name: Close
Type: REG_SZ
Data: CloseINFOPerformanceData

Value 1
Name: Collect
Type: REG_SZ
Data: CollectINFOPerformanceData

Value 2
Name: First Counter
Type: REG_DWORD
Data: 0x738

Value 3
Name: First Help
Type: REG_DWORD
Data: 0x739

Value 4
Name: Last Counter
Type: REG_DWORD
Data: 0x756

Value 5
Name: Last Help
Type: REG_DWORD
Data: 0x757

Value 6
Name: Library
Type: REG_SZ
Data: infoctrs.DLL

Value 7
Name: Open
Type: REG_SZ
Data: OpenINFOPerformanceData

TPC-C application registry

Key Name: SOFTWARE\Microsoft\TPCC
Class Name: <NO CLASS>
Last Write Time: 7/14/97 - 5:52 PM
Value 0
Name: BackoffDelay
Type: REG_SZ
Data: 500

Value 1
Name: ConnectionPooling
Type: REG_SZ
Data: OFF

Value 2
Name: ConnectionPoolRetryTime
Type: REG_SZ
Data: 500

Value 3
Name: DeadlockRetry
Type: REG_SZ
Data: 3

Value 4
Name: LastInstalledVersion
Type: REG_SZ
Data: DBLIB

Value 5
Name: LOG
Type: REG_SZ
Data: OFF

Value 6
Name: MaxConnections
Type: REG_SZ
Data: 1600

Value 7
Name: MaximumWarehouses
Type: REG_SZ
Data: 640

Value 8
Name: NumberOfDeliveryThreads
Type: REG_SZ
Data: 5

Value 9
Name: PATH
Type: REG_SZ
Data: C:\InetPub\wwwroot\

Value 10
Name: QueueSlotts
Type: REG_SZ
Data: 3000

WWW Service Registry

Key Name: SYSTEM\CurrentControlSet\Services\W3SVC
Class Name: <NO CLASS>
Last Write Time: 4/11/97 - 6:15 PM
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: C:\WINNT\System32\inetsrv\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: 7/27/97 - 10:21 PM
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\HTMLA
Class Name: <NO CLASS>
Last Write Time: 4/11/97 - 6:16 PM

Key Name:
SYSTEM\CurrentControlSet\Services\W3SVC\Parameters
Class Name: <NO CLASS>
Last Write Time: 6/26/97 - 5:03 PM
Value 0
Name: AcceptExOutstanding
Type: REG_DWORD
Data: 0x7d0

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_IIS003

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: C:\WINNT\System32\inetsrv\sspifilt.dll

Value 13
Name: GlobalExpire
Type: REG_DWORD
Data: 0xffffffff

Value 14
Name: InstallPath
Type: REG_SZ
Data: C:\WINNT\System32\inetsrv

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: NTAuthenticationProviders
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\Script Map
Class Name: <NO CLASS>
Last Write Time: 4/11/97 - 6:15 PM
Value 0
Name: .idc
Type: REG_SZ
Data: C:\WINNT\System32\inetsrv\httpodbc.dll

Key Name:
SYSTEM\CurrentControlSet\Services\W3SVC\Parameters\Virtual Roots
Class Name: <NO CLASS>
Last Write Time: 4/14/97 - 10:24 AM
Value 0
Name: /,
Type: REG_SZ
Data: C:\InetPub\wwwroot,,5

Value 1
Name: /iisadmin,
Type: REG_SZ
Data: C:\WINNT\System32\inetsrv\iisadmin,,1

Value 2
Name: /Scripts,
Type: REG_SZ
Data: C:\InetPub\scripts,,4

Key Name:
SYSTEM\CurrentControlSet\Services\W3SVC\Performance
Class Name: <NO CLASS>
Last Write Time: 4/11/97 - 6:15 PM
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

<Microsoft SQL Server version 6.5 Startup Parameters>

sqlservr -c -x -t1081 -t3502

-c Sart SQL Server independently of the Microsoft Windows NT Service Control Manager.
-x Disable the keeping of CPU time and cache-hit ration statistics.
-t1081 Allow the index pages a second trip through cache before those pages are released.
-t3502 Prints a message to the log at the beginning and end of each checkpoint.

Microsoft SQL Server Stack Size

The default stack size of Microsoft SQL Server was changed using the EDITBIN utility. The EDITBIN utility ships with Microsoft Visual C++ 4.2.
The command used was editbin /stack:65536 sqlservr.exe.

DBCC GAMINIT

Prior to the execution of the benchmark the command "dbcc gaminit" was run in order to populate the Global Allocation Map (GAM) rather than populating it on a dynamic basis.

```
use tpcc
go
dbcc gaminit
go
```

<Microsoft SQL Server 6.5 Configuration Parameters>

SQL SERVER CONFIGURATION PARAMETER

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	-1
database size	2	10000	2	2
default language	0	9999	0	0
default sortorder id	0	255	50	50
fill factor	0	100	0	0
free buffers	20	524288	2000	2000
hash buckets	4999	265003	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	0
locks	5000	2147483647	5000	5000
LogLRU buffers	0	2147483647	2000	2000
logwrite sleep (ms)	-1	500	-1	-1
max async IO	1	1024	32	32
max lazywrite IO	1	1024	48	48
max text repl size	0	2147483647	65536	65536
max worker threads	10	1024	144	144
media retention	0	365	0	0
memory	2800	1048576	890000	890000
nested triggers	0	1	1	1
network packet size	512	32767	4096	4096
open databases	5	32767	20	20
open objects	100	2147483647	500	500
priority boost	0	1	0	0
procedure cache	1	99	2	2
Protection cache size	1	8192	15	15
RA cache hit limit	1	255	4	4
RA cache miss limit	1	255	3	3
RA delay	0	500	15	15
RA pre-fetches	1	1000	3	3
RA slots per thread	1	255	5	5
RA worker threads	0	255	0	0
recovery flags	0	1	0	0
recovery interval	1	32767	32767	32767
remote access	0	1	1	1
remote conn timeout	-1	32767	10	10
remote login timeout	0	2147483647	5	5
remote proc trans	0	1	0	0
remote query timeout	0	2147483647	0	0
remote sites	0	256	10	10
resource timeout	5	2147483647	10	10
set working set size	0	1	0	0
show advanced options	0	1	1	1
SMP concurrency	-1	64	-1	-1
sort pages	64	511	64	64
spin counter	1	2147483647	10000	10000
tempdb in ram (MB)	0	2044	5	5
time slice	50	1000	100	100
user connections	5	32767	6350	6350
user options	0	4095	0	0

Disk Array Configuration

```
*******************************************************************************
*       MYLEX Disk Array Controller - Configuration Utility                   *
*                       Version 4.71                                          *
*******************************************************************************

CONFIGURATION INFORMATION OF :
==============================

3 Channel - 15 Target  DAC960PDU  #1    Firmware version 3.50


PHYSICAL PACK INFORMATION :
===========================

Number of Packs = 3

Pack  0 : [0:0]  [0:1]  [0:2]  [0:3]  [0:4]  [0:5]
Pack  1 : [1:0]  [1:1]  [1:2]  [1:3]  [1:4]  [1:5]
Pack  2 : [2:0]  [2:1]  [2:2]  [2:3]  [2:4]  [2:5]

SYSTEM DRIVE INFORMATION :
==========================

Number of System Drives = 1

Sys Drv #   Phy. Size   Raid Level   Eff. Size   Write Policy
=========  =========  ==========  =========  ============
   0       156294  MB      0       156294  MB   Write Thru

*******************************************************************************
*       MYLEX Disk Array Controller - Configuration Utility                   *
*                       Version 4.71                                          *
*******************************************************************************

CONFIGURATION INFORMATION OF :
==============================

3 Channel - 15 Target  DAC960PDU  #2    Firmware version 3.50


PHYSICAL PACK INFORMATION :
===========================

Number of Packs = 3

Pack  0 : [0:0]  [0:1]  [0:2]  [0:3]  [0:4]  [0:5]
Pack  1 : [1:0]  [1:1]  [1:2]  [1:3]  [1:4]  [1:5]
Pack  2 : [2:0]  [2:1]  [2:2]  [2:3]  [2:4]  [2:5]

SYSTEM DRIVE INFORMATION :
==========================

Number of System Drives = 1

Sys Drv #   Phy. Size   Raid Level   Eff. Size   Write Policy
=========  =========  ==========  =========  ============
   0       156294  MB      0       156294  MB   Write Thru

*******************************************************************************
*       MYLEX Disk Array Controller - Configuration Utility                   *
*                       Version 4.71                                          *
*******************************************************************************

CONFIGURATION INFORMATION OF :
==============================

3 Channel - 15 Target  DAC960PDU  #3    Firmware version 3.50


PHYSICAL PACK INFORMATION :
===========================

Number of Packs = 3

Pack  0 : [0:0]  [0:1]  [0:2]  [0:3]  [0:4]  [0:5]
Pack  1 : [1:0]  [1:1]  [1:2]  [1:3]  [1:4]  [1:5]
Pack  2 : [2:0]  [2:1]  [2:2]  [2:3]  [2:4]  [2:5]

SYSTEM DRIVE INFORMATION :
==========================

Number of System Drives = 1

Sys Drv #   Phy. Size   Raid Level   Eff. Size   Write Policy
=========  =========  ==========  =========  ============
   0       156294  MB      0       156294  MB   Write Thru

*******************************************************************************
*       MYLEX Disk Array Controller - Configuration Utility                   *
*                       Version 4.71                                          *
*******************************************************************************

CONFIGURATION INFORMATION OF :
==============================

3 Channel - 15 Target  DAC960PDU  #4    Firmware version 3.50
```

```
PHYSICAL PACK INFORMATION :
==========================

Number of Packs = 3

Pack  0 : [0:0]  [0:1]  [0:2]  [0:3]  [0:4]  [0:5]
Pack  1 : [1:0]  [1:1]  [1:2]  [1:3]  [1:4]  [1:5]
Pack  2 : [2:0]  [2:1]  [2:2]  [2:3]  [2:4]  [2:5]

SYSTEM DRIVE INFORMATION :
==========================

Number of System Drives = 1

Sys Drv #   Phy. Size   Raid Level   Eff. Size   Write Policy
=========  =========  ==========  =========  ============
   0       156294  MB      0       156294  MB   Write Thru

*************************************************************************
*       MYLEX Disk Array Controller - Configuration Utility             *
*                       Version 4.71                                    *
*************************************************************************

CONFIGURATION INFORMATION OF :
==============================

3 Channel - 15 Target  DAC960PDU  #5    Firmware version 3.50


PHYSICAL PACK INFORMATION :
==========================

Number of Packs = 3

Pack  0 : [0:0]  [0:1]  [0:2]  [0:3]  [0:4]  [0:5]
Pack  1 : [1:0]  [1:1]  [1:2]  [1:3]  [1:4]  [1:5]
Pack  2 : [2:0]  [2:1]  [2:2]  [2:3]  [2:4]  [2:5]

SYSTEM DRIVE INFORMATION :
==========================

Number of System Drives = 1

Sys Drv #   Phy. Size   Raid Level   Eff. Size   Write Policy
=========  =========  ==========  =========  ============
   0       156294  MB      0       156294  MB   Write Thru

*************************************************************************
*       MYLEX Disk Array Controller - Configuration Utility             *
*                       Version 4.71                                    *
*************************************************************************

CONFIGURATION INFORMATION OF :
==============================

3 Channel - 15 Target  DAC960PDU  #6    Firmware version 3.50


PHYSICAL PACK INFORMATION :
==========================

Number of Packs = 3

Pack  0 : [0:0]  [0:1]  [0:2]  [0:3]  [0:4]  [0:5]
Pack  1 : [1:0]  [1:1]  [1:2]  [1:3]  [1:4]  [1:5]
Pack  2 : [2:0]  [2:1]  [2:2]  [2:3]  [2:4]  [2:5]

SYSTEM DRIVE INFORMATION :
==========================

Number of System Drives = 1

Sys Drv #   Phy. Size   Raid Level   Eff. Size   Write Policy
=========  =========  ==========  =========  ============
   0       156294  MB      0       156294  MB   Write Thru
```

Appendix E: Space Calculation

180 Day Space

Note : Numbers are in KBytes unless otherwise specified						
Warehouses	630	**tpmC**	762563	**tpmC/W**	12.10	
Table	**Rows**	**Data**	**Index**	**5% Space**	**8H Space**	**Total Space**
Warehouse	650	1,300	10	66		1,376
District	6,500	13,000	56	653		13,709
Item	100,000	9,100	46	210		9,356
New-order	5,850,000	64,350	394		13,000	77,744
History	19,500,000	975,002	0		151,749	1,126,751
Orders	19,500,000	507,000	3,060		79,386	589,446
Customer	19,500,000	13,002,600	1,009,188	322,271		14,334,059
Order-line	195,004,080	10,842,228	70,872		1,698,512	12,611,612
Stock	65,000,000	21,671,000	119,736	501,187		22,291,923
Totals		47,085,580	1,203,362	824,387	1,942,646	51,055,975
Segment	**LogDev Cnt.**	**Seg. Size**	**Needed**	**Overhead**		**Not Needed**
misc_seg	5	2,605,056	1,836,565	18,366		750,125
cs_seg	12	39,073,792	36,992,242	369,922		1,711,628
ol_seg	5	15,265,792	12,737,728	127,377		2,400,687
Totals		56,944,640	51,566,535	515,665		4,862,440
Dynamic space	11,991,476	Sum of Data for Order, Order-Line and History (excluding free extents)				
Static space	37,637,518	Data + Index + 5% Space + Overhead - Dynamic space				
Free space	2,453,206	Total Seg. Size - Dynamic Space - Static Space - Not Needed				
Daily growth	2,250,894	(Dynamic space/W * 62.5) * tpmC				
Daily spread	(923,135)	Free space - 1.5 * Daily growth (zero if negative)				
180 day (KB)	442,798,389	Static space + 180 (daily growth + daily spread)				
180 day (GB)	422.29	Excludes OS, Paging and RDBMS Logs				
Log size (MB)	16000	Total size of log file				
% Log used	25.2782	% of log file used during entire run				
Total N-O Txn	736192	Total count of N-O transactions during entire run				
Log per N-O txn	2.8128	Number of 2K blocks per New-Order transaction				
8 Hour Log (GB)	19.64					
os, file sys, swap (GB)	4.00					
	Disks	**Qty**	**Total**	**Needed**	**Extra**	
Database, Sys (GB)	8.47	108	915.08	426.29	488.80	
Mirrored Log (GB)	8.47	6	50.84	39.28	11.56	

Appendix F: Auditor's letter

8/7/97 21:53 Macintosh P.001

Information Paradigm

TPC TRANSACTION PROCESSING PERFORMANCE COUNCIL

Certified Auditor

Test Sponsor: Eiichi Kenai
NEC
3rd Development Dept
3rd Computers Software Dept
Fuchu City Tokyo 183, Japan

Aug 7, 1997

I verified the TPC Benchmark™ C performance of the following configuration:

Platform: Express 5800 MH4000 C/S
DataBase Manager: Microsoft SQL Server 6.5 (SP3)
Operating System: Microsoft Windows NT 4.0 Server

The results were:

CPU's	Memory	Disks	NewOrder 90% Response Time	tpmC
Server: Express 5800 MH4000 C/S				
4 x Intel Pentium Pro (200 MHz)	2048 MB	114 x 9 GB	2.02 Seconds	7625.63
Five Clients: PowerMate 2200 (Specification for each)				
1 x Intel Pentium Pro (200 MHz)	128 MB	1 x 2 GB	n/a	n/a

In my opinion, these performance results were produced in compliance with the TPC 3.3 requirements for the benchmark. The following verification items were given special attention:

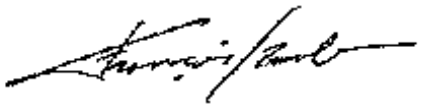
- The transactions were correctly implemented
- The database records were the proper size
- The database was properly scaled and populated
- The ACID properties were met
- Input data was generated according to the specified percentages
- The transaction cycle times included the required keying and think times

1378 North Franklin Street • Colorado Springs, CO 80903-3527 • Office: 719/473-7555 • Fax: 719/473-7554

- The reported response times were correctly measured.
- At least 90% of all delivery transactions met the 80 Second completion time limit
- All 90% response times were under the specified maximums
- The measurement interval was representative of steady state conditions
- The reported measurement interval was 30 minutes
- One checkpoint was taken during the measurement interval
- Measurement repeatability was verified
- The 180 day storage requirement was correctly computed
- The system pricing was verified for major components and maintenance

Additional Audit Notes:

Respectfully Yours,



François Raab
President

Express 5800 MH4000

1373 North Franklin Street • Colorado Springs, CO 80903-2527 • Office: 719/473-7555 • Fax: 719/473-7554

Appendix G: Price Quotation

08/07/97 08:34 P.02/04

Insight®

America's discount source for computers, hardware and software

6820 S. Harl Ave
Tempe, Az. 85283

08/07/97

KEVIN GRAY
NEC MICROMASS
Customer Number: 1732163
Quote Number: G85320

Thank you for your interest in Insight. Please find below the price quote you requested.

PRODUCT	DESCRIPTION	QUANTITY	PRICE/EA	TOTAL
MY399178	DACPUM-3-8E-MY4 PCI TO UWSCSI3 PC RAID 3 CHNL, 8MB EDRAM, GAM MFG Part#: DACPUM-3-8E-MY4	1	1999.99	1999.99
	Subtotal Without Shipping & Handling			1999.99
	Shipping Via 2-DAY STD AIR	+		19.46
	GRAND Total			2019.45

" LEASE IT FOR 79.77 PER MONTH, PLUS APPLICABLE TAX "
*** Lease term based on 36 month lease, A F.M.V. purchase option, 2 pmts in advance. O.A.C. ***
*Please contact your sales professional to add a 4 year warranty to your order.

* All products carry full manufacturer warranty

If you have any questions, or require additional information, please do not hesitate to give me a call. Thank you again for calling Insight.

Sincerely,

Thomas J Levy

Insight 800-INSIGHT
467-4448
www.insight.com

COMPUSA PROPOSAL

TO.. CUSTOMER: NEC
CONTACT: KEN GRAY
PHONE: (508)635-8077
FAX #: ( ) -

0006857 Order# N STORE#827

FRAMINGHAM, MA 01701

DATE SUBMITTED: 08/07/97 QUOTE NO.: 11779 QUOTE VALID FOR 30 Days

PRODUCT PRICING AND INFORMATION

PRODUCT CODE	PRODUCT DESCRIPTION	REQ. QTY	UNIT PRICE	EXTENDED PRICE
114878	INTE ETHXPS PCI *B*-20PACK	1	1,134.09	1,134.09
118542	3COM EISA ETH 10/100 TP	3	232.61	697.83

If you have any questions regarding this quote please contact:
MARTHA TAYLOR (FRAMINGHAM)
Phone:(508)935-3306 Fax:(508)935-3398
COMPUSA OFFERS CLASSROOM AND ONSITE TRAINING
FOR INFORMATION PLEASE CALL: (508)935-3810

QUOTE TOTAL: $ 1,831.92

Freight Charges will be added at the time of shipment based on the weight of products shipped. Sales Taxes will be added where applicable. The above line item component prices are valid for this quote only.

THANK YOU FOR THE OPPORTUNITY TO DO BUSINESS WITH YOU!

NETLUX

14180 Live Oak Ave., Unit E
Baldwin Park, Ca. 91760

1-800-789-1780
Phone #818-851-9737
Fax #818-851-9837

August 7, 1997

NEC Corp
Kevin Grey
PH# 508-635-6077
FAX# 508-635-6888

Quotation

Quantity	Description	Unit Price	Total
3	NX-H8TX 8 Port 100BASE-TX FAST Ethernet Hub	$329.00	$987.00
297	NX-H24 24Port 10BASE-T Ethernet Hub	$251.00	$74,547.00

Terms and Conditions:

FOB Origin
5 Year Warranty
Prices good for 60 Days

Sincerely,
Martin Parry
NETLUX

Redmond, WA 98052-6399 Fax 206 936 7329

Microsoft

August 5, 1997

Mr. Mikio Kanemaki
Packard Bell NEC
1414 Massachusetts Avenue
Boxborough, MA 01719-2298

Via FAX: 508-635-6888

Dear Kanemaki-san,

Microsoft has received your request for permission to disclose the results of TPC-C benchmark tests conducted by NEC with Microsoft SQL Server 6.5 on the following system:

NEC Express 5800 MH400, 4-processors, Pentium Pro, 200 MHz
Test results: 7625.63 tpmC @ $65/tpmC approx.

Microsoft hereby grants NEC permission to disclose these results to third parties and acknowledges that NEC has formally requested permission to do so in accordance with the license agreement for Microsoft SQL Server software.

Best regards,

Sid Arora

Sid Arora
Product Manager, Microsoft SQL Server
Personal and Business Systems Division

Redmond, WA 98052-6399 Fax 206 936 7329

Microsoft

August 5, 1997

Mr. Mikio Kanemaki
Packard Bell NEC
1414 Massachusetts Avenue
Boxborough, MA 01719-2298

Via FAX: 508-635-6888

Dear Kanemaki-san,

Here is the information you requested regarding US pricing of certain Microsoft products:

Microsoft SQL Server 6.5 software, unlimited user license	$24999
Windows NT Server 4.0 software, incl 5 CALs	$809
Microsoft SQL Workstation (includes programmers toolkit)	$499
Visual C++ 5.0 Professional Edition	$499
5-yr maintenance for above software @ $2095/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 Division