TPC Benchmark™ C
Full Disclosure Report
For
Acer Incorporated AcerAltos 19000Pro4
Using
Microsoft SQL Server 6.5 Enterprise Edition
And
Microsoft Windows NT 4.0 Enterprise Edition

First Edition
Submitted for Review
February 16, 1998

First Printing, February 16, 1998

Acer Inc. believes that the information included in this document is accurate as of the publication date. The information in this document is subject to change without notice. Furthermore, Acer Inc. is not responsible for any errors contained within this document.

The pricing information given in this FDR is accurate as of the publication date, February 16, 1998. However, Acer Inc. 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 for 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. Acer Inc. does 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.

AcerAltos is a trademark of the Acer Inc.

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

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

Pentium Pro is a registered trademark of Intel.

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

Abstract

Overview

This report documents the methodology and results of the TPC Benchmark™ C test conducted on the Acer Inc AcerAltos 19000Pro4. The tests were run in a client/server configuration using 5 AcerAltos 900Pro as clients. The operating system used for the benchmark was Microsoft NT Server 4.0 Enterprise Edition for the server and NT Server 4.0 on the clients. The database was the Microsoft SQL Server Enterprise Edition 6.5. Tuxedo 6.3 Core Functional Services (CFS) provided the database connection queues. All tests were done in compliance with Revision 3.3.2 of the Transaction Processing Council's TPC Benchmark™ C Standard Specification. Two standard TPC Benchmark tm C metrics, transactions per second (tpmC) and price per tpmC ($/tpmC) are reported and referred to in this document. The results from the tests are summarized below.

Hardware	Software	Total System Cost	tpmC	$/tpmC	Availability Date
Acer Inc. AcerAltos 19000Pro4	Microsoft Windows NT 4.0 Enterprise Edition. Microsoft SQL Server Enterprise Edition	**$301,663**	**11,072.07**	**$27.25**	HW: February 16, 1998 SW: February 16, 1998

AUDITOR

The results of the benchmark and test methodology used to produce the results were audited by Lorna Livingtree of Performance Metrics, Inc. and have fully met the TPC-C rev 3.3.2 specifications.

Additional copies of this Full Disclosure Report can be obtained from either the Transaction Processing Performance Council or Acer Inc. at the following address:

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

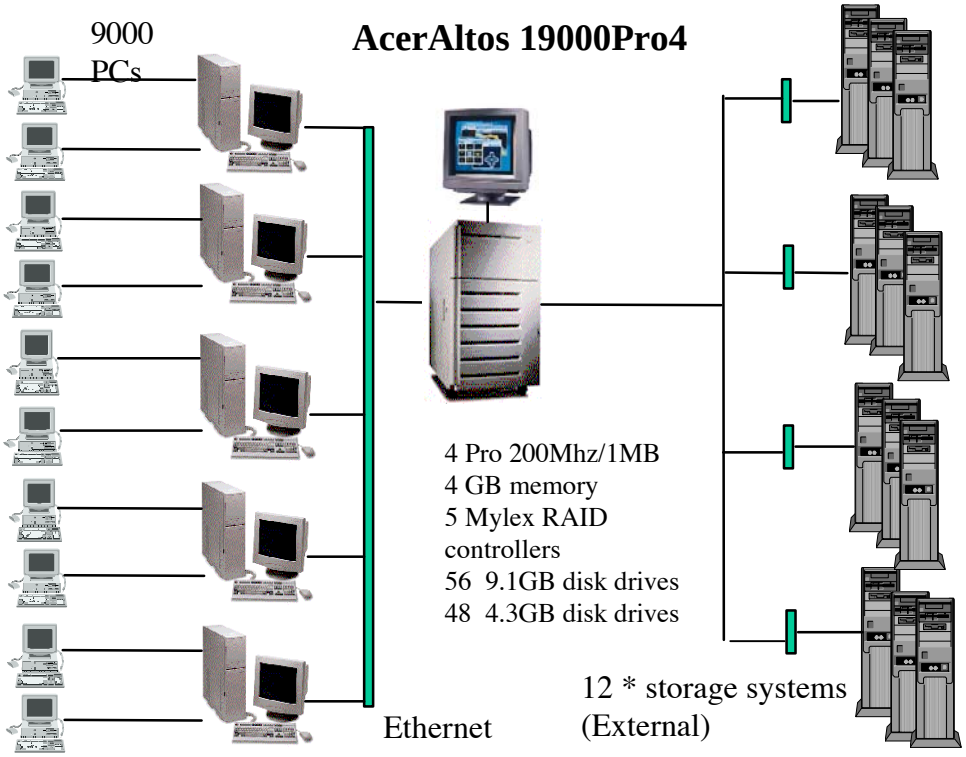
or

Acer Inc.
21F, 88, Sec. 1, Hsin Tai Wu Rd.,
Hsichih, Taipei Hsien 221,
Taiwan, R.O.C.
Phone: (02)8691-1530, fax 8691-2379

ACER	AcerAltos 19000Pro4 **Client/Server w/5 AcerAltos 900Pro Front Ends**	TPC-C Rev 3.3.2 Report Date Feb 16, 1998

Total System Cost	TPC-C Throughput	Price/Performance	Availability Date
$301,663	**11,072.07 tpmC**	**$27.25**	Feb 16, 1998

Processors	Database Manager	Operating System	Other Software	Number of Users
4 × Pentium Pro Processors 200 MHz 1MB L2 cache	Microsoft SQL Server 6.5 Enterprise Edition	Microsoft Windows NT Server4.0 Enterprise Edition	Microsoft Internet Information Server Microsoft Visual C++ Tuxedo 6.3 CFS	9,000



5 AcerAltos 900Pro

System Component	Server		Each Clients	
Processors Cache	4	Pentium Pro 200 MHz 1MB	1	Pentium Pro 200 MHz 256 KB
Memory	1	4096 MB	1	192 MB
Disk Controllers	5	Mylex DAC960PJ		Adaptec On-Board AIC-7880P
Disk Drives	56 48	9.1 GB 4.3 GB	1	2 GB
Total Storage		697.6 GB		2 GB
Other	1	CD-ROM	1	CD-ROM

AcerAltos 19000Pro4 Client/Server

TPC-C REV3.3.2 EXECUTIVE SUMMARY

Description	Part Number	Third Party	Unit Price	Qty	Extended Price	5 yr. Maint Price *
Server Hardware						
AcerAltos 19000Pro4 w/ CD-Rom16x, keyboard, mouse, UPS, 0 Proc , 0MB Mem, 14 hot-swap drive bays, Three 400W power supplies	91.AB555.001	1	$8,100	1	$8,100	$1,215
Pentium Pro 200MHz/1MB Cache	01.I00P6.K80	1	$3,050	4	$12,200	$1,830
256MB FPM DIMM	72.48178.00E	1	$872	16	$13,952	$2,093
Intel EtherExpress 10/100 PCI	54.AB023.001	1	$82	1	$82	$12
14"AcerView color monitor	91.75402.067	1	$163	1	$163	$24
9GB UW SCSI Disk Drive	91.AB032.001	1	$895	48	$42,960	$6,444
4GB UW SCSI Disk Drive	91.AB360.003	1	$370	32	$11,840	$1,776
4GB UW SCSI Disk Drive	91.AB360.205	1	$647	16	$10,352	$1,553
9GB UW SCSI Disk Drive -10000 RPM	91.AB037.002	1	$1,117	8	$8,936	$1,340
Mylex Disk Array Controller w/ 8MB EDO***	91.AB002.006	Mylex 5	$1,845	7	$12,915	$350
External SCSI Cable	90.AB600.001	1	$35	36	$1,260	$189
DAT Tape Drive	91.AB034.001	1	$1,065	1	$1,065	$160
SCSI Bus Extender Board		Symbios 3	$108	12	$1,296	$194
External Storage Cabinet	91.91720.004	1	$780	12	$9,360	$1,404
				Subtotal	$134,481	$18,585
Server Software						
MS SQL Server, Enterprise Edition 6.5, unlimited user license**		Microsoft 2	$28,999	1	$28,999	$10,475
MS Windows NT Server 4.0 Enterprise Edition, incl 25 CALs**		Microsoft 2	$3,999	1	$3,999	0
				Subtotal	$32,998	$10,475
Client Hardware						
AcerAltos900Pro w/ 1Ppro200MHz, 2GB Ultra Wide SCSI HD, Keyboard, Mouse, 200W power supply	91.AB244.206	1	$2,714	5	$13,570	$2,036
32MB ECC/EDO SIMM	91.11010.739		$105	30	$3,150	$473
Intel EtherExpress 10/100 PCI	54.AB023.001		$82	15	$1,230	$185
14"AcerView color monitor	91.75402.067		$163	5	$815	$122
				Subtotal	$18,765	$2,815
Client Software						
MS Windows NT Server 4.0, incl 5 CALs**		Microsoft 2	$809	5	$4,045	$0
MS Visual C++ 32-bit Edition(subscription)**		Microsoft 2	$499	1	$499	$0
Microsoft SQL Workstation(include programmers toolkit)**		Microsoft 2	$499	1	$499	$0
TUXEDO Core Functional Services 6.3 for NT		BEA 4	$3,000	5	$15,000	$11,250
				Subtotal	$20,043	$11,250
User Connectivity						
Netlux 8-port 100Mbps FAST Ethernet Hub ***	NX-H8TX	Netlux 6	$279	3	$837	$0
Netlux 16-port 10BASE-T Ethernet Hub ***	NX-H16ez	Netlux 6	$82	627	$51,414	$0
				Subtotal	$52,251	$0
				Total	$258,538	$43,125

Notes
* HW Maintenance - 1st 36 months included in Acer product cost.
** All Microsoft maintenance is covered by the maintenance cost of Microsoft SQL Server.
*** 10% or minimum 2 spares are added in place of onsite service (product have a five year return-to-vendor warranty)
Pricing: 1=Acer, 2=Microsoft, 3= Symbios, 4=BEA, 5=Mylex, 6=Netlux

Five-Year Cost of Ownership: $301,663
TpmC Rating: 11072.07
$/tpmC: $27.25

The benchmark results and test methodology were audited by Lorna Livingtree of Performance Metrics, Inc.

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

MQTh, computed Maximum Qualified Throughput	11,072.07tpmC
% throughput difference, reported & reproducibility runs	0.1%

Response Times (90th percentile | Average | Maximum) in seconds

- Neworder	0.97	0.65	113.71
- Payment	0.42	0.33	23.76
- Order Status	1.37	0.91	25.87
- Delivery (interactive portion)	0.28	0.28	0.82
- Delivery (deferred portion)	0.82	0.57	7.37
- Stock-Level	4.65	2.90	61.09
- Menu	0.31	0.20	1.10

Transaction Mix, in percent of total transactions

- New-Order	44.87 %
- Payment	43.03 %
- Order-Status	4.05 %
- Delivery	4.04 %
- Stock-Level	4.00 %

Keying/Think Times (in seconds),	Min		Average		Max	
- New-Order	18.02	0.00	18.03	12.06	18.10	121.00
- Payment	3.02	0.00	3.03	12.10	3.14	121.00
- Order-Status	2.02	0.00	2.03	10.16	2.05	100.99
- Delivery	2.02	0.00	2.03	5.14	2.16	51.00
- Stock-Level	2.02	0.00	2.03	5.02	2.05	50.26

Test Duration

- Ramp-up time	25 minutes
- Measurement interval	30 minutes
- Number of checkpoints in measured interval	1
- Checkpoint interval	30 minutes
- Number of transactions (all types) completed in measurement interval	740,234

Introduction

Document Structure

The contents of this report are determined by the TPC Benchmark C Standard Specification Revision 3.3.2, written and approved by the Transaction Processing Performance Council (TPC). The format of this report is based on this specification. Most sections of this report begins with the relevant specification requirements printed in italic type, immediately followed by the detail in plain type of how Acer Inc. complied with the specification. Where extensive listings are required (such as listing of code), a note is included which references an appendix containing the listing.

Benchmark Overview

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

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

The performance metric reported by TPC-C is a "business throughput" measuring the number of orders processed per minute. Multiple transactions are used to simulate the business activity of processing an order, and each transaction is subject to a response time constraint.

The performance metric for this benchmark is expressed in transactions-per-minute-C (tpmC). To be compliant with the TPC-C standard, all references to tpmC results must include the tpmC rate, the associated price-per-tpmC, and the availability date of the priced configuration.

Although these specifications express implementation in terms of a relational data model with conventional locking scheme, the database may be implemented using any commercially available database management system (DBMS), database server, file system, or other data repository that provides a functionally equivalent implementation. The terms "table", "row", and "column" are used in this document only as examples of logical data structures.

TPC-C uses terminology and metrics that are similar to other benchmarks, originated by the TPC or others. Such similarity in terminology does not in any way imply that TPC-C results are comparable to other benchmarks. The only benchmark results comparable to TPC-C are other TPC-C results conformant with the same revision.

Despite the fact that this benchmark offers a rich environment that emulates many OLTP applications, this benchmark does not reflect the entire range of OLTP requirements. In addition, the extent to which a customer can achieve the results reported by a vendor is highly dependent on how closely TPC-C approximates the customer application. The relative performance of systems derived from this benchmark

does not necessarily hold for other workloads or environments. Extrapolations to any other environment are not recommended.

Benchmark results are highly dependent upon workload, specific application requirements, and 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 benchmarking when critical capacity planning and/or product evaluation decisions are contemplated.

System Overview

The hardware configuration used in this TPC-C test is a AcerAltos 19000Pro4 Server driven by 5 AcerAltos 900Pro clients. The clients and server are networked together via a 10BaseT hub. Five remote terminal emulators (RTE), AcerAltos 9000Pro, emulated 9000 users executing the standard TPC-C workload. The RTEs were connected to the five clients.. Microsoft NT4.0 Enterprise Edition was the operating system used on the server and standard edition on the clients. Microsoft SQL server Enterprise Edition 6.5 was the database on the server machine.

Each of the clients has one 200MHz Pentium Pro processors with 256 KB of L2 cache, 192 MB of RAM, one 2 GB SCSI hard disk, and three Intel Ether Express PRO100/B ethernet adapters. On each client, one of the Intel ethernet adapters was connected to the server through a 10BaseT hub. The other two Intel ethernet adapters provided 2 network segments to the RTE machines, with 900 emulated users per segment.

General Items

Test Sponsor

A statement identifying the sponsor of the Benchmark and any other companies who have participated.

Acer Inc. was the test sponsor of this TPC Benchmark™ C.

Application Code and Definition Statements

The application program must be disclosed. This includes, but is not limited to, the code implementing the five transactions and the terminal input/output functions.

The Section 3.0 entitled Clause 3 Related Items contains a brief discussion of the database design and loading. The database definition statements, distribution across disk drives, loading scripts, and tables are provided in Appendix B-Database Design.

The program that implements the TPC Benchmark C translation and collects appropriate transaction statistics is referred to as the Remote Terminal Emulator (RTE) or Driver program. The Driver program is discussed in Section 7.0.

Parameter Settings

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

- *Database options*
- *Recover/commit options*
- *Consistency/locking options*
- *System parameter, application parameters, and configuration parameters.*

This requirement can be satisfied by providing a full listing of all parameters and options.

Appendix C contains all the database and operating system parameters used in this benchmark. Appendix C contains all the hardware configuration details.

Configuration Diagrams

Diagrams of both the measured priced system must be provided, accompanied by a description of the differences.

Figure 1 and 2 respectively show the measured and priced full client/server configurations. The SUT in the measured system and the priced system differed only in that 24-4GB drives from the measured system were converted to 24-9GB drives for the priced system.

Figure 1: Measured Configuration

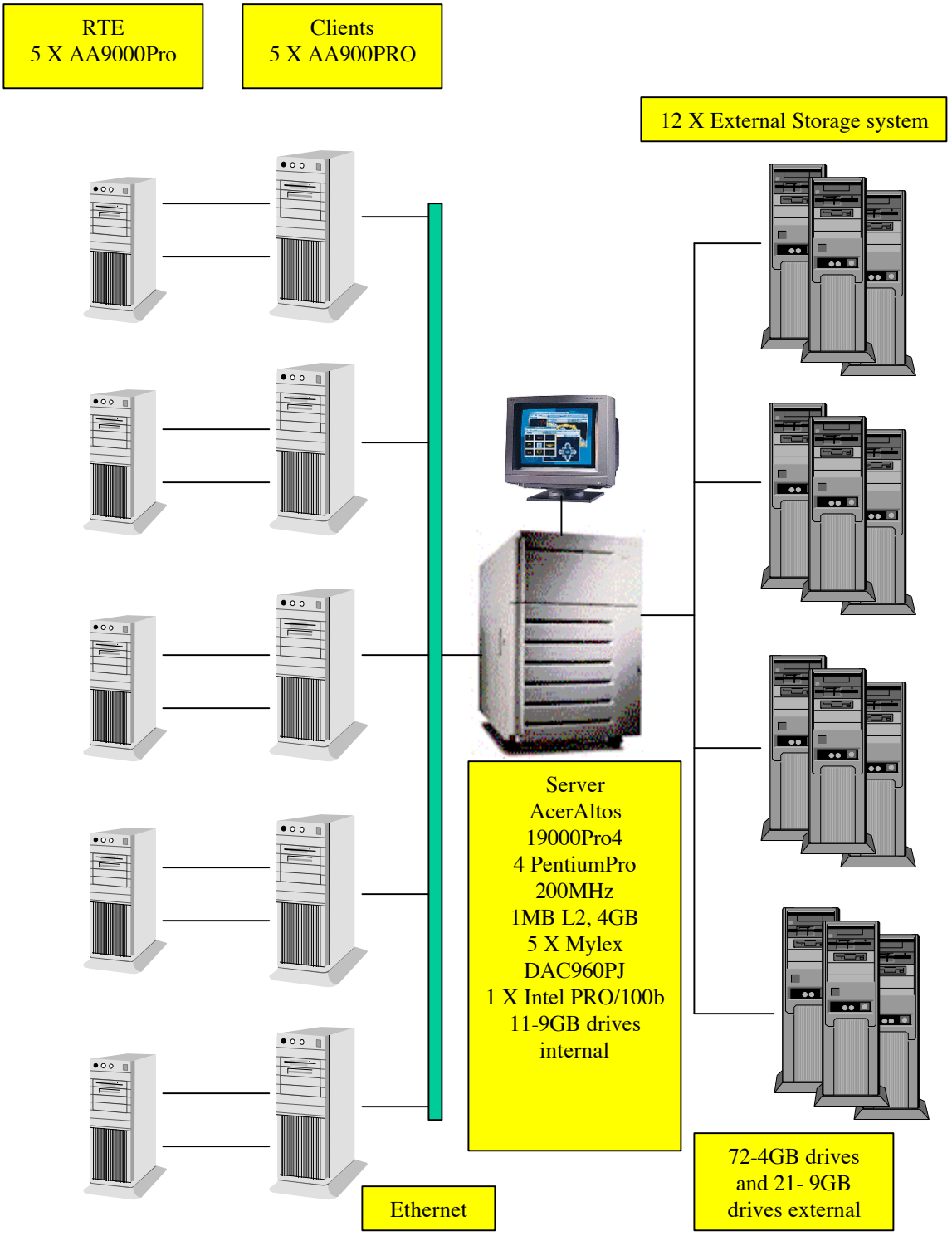
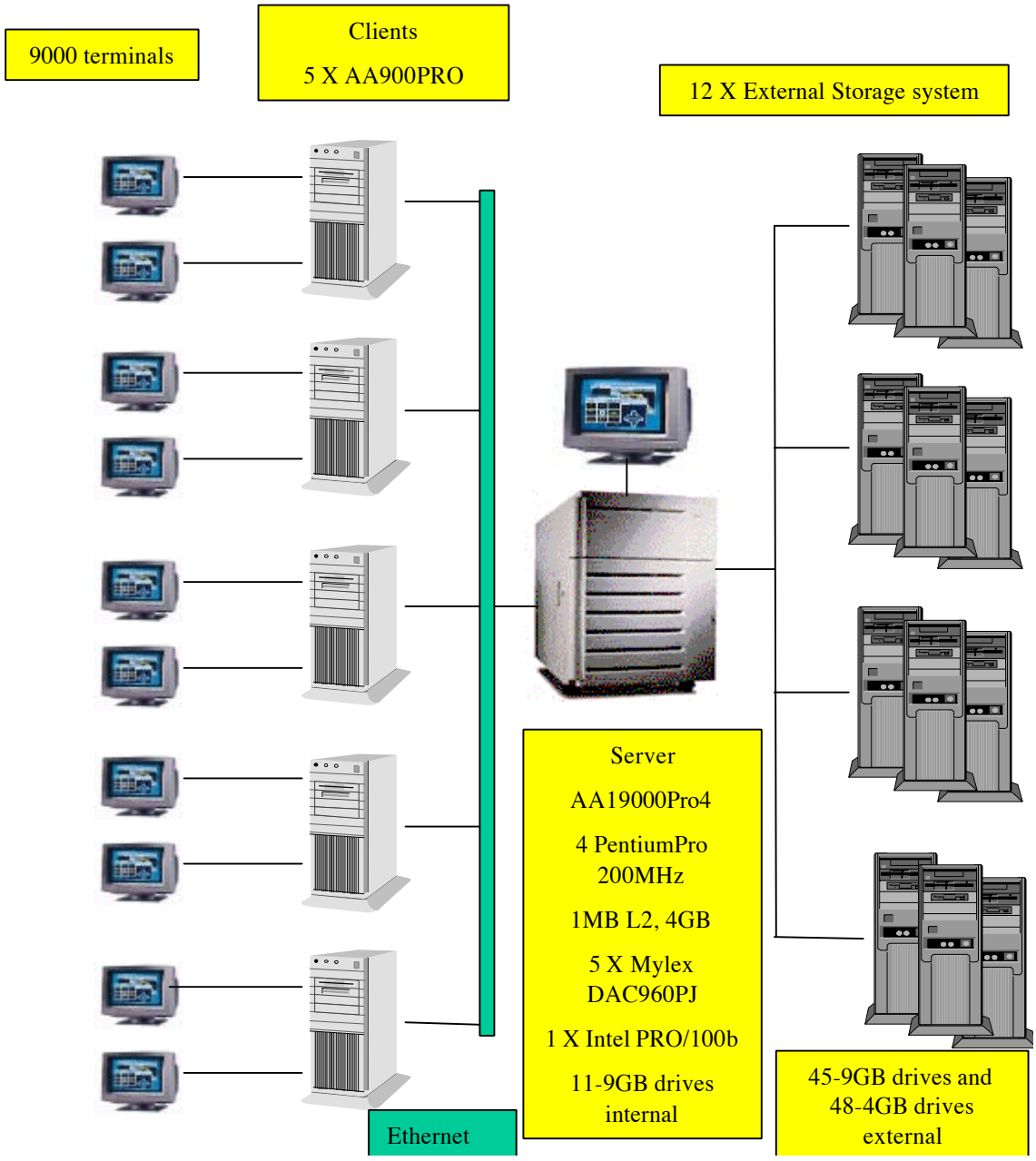


Figure 2: Priced Configuration



Clause 1 -- Logical Database Design Related Items

Table Definitions

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

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

Physical Organization of the Database

The physical organization of tables and indices, within the database, must be disclosed. (8.1.2.2)

The measured configuration used 104 disk drives, 32-9GB drives and 72-4GB drives. The organization is shown in Table 5: Data Distribution and in Appendix B.

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 restriction in the SUT database implementation that precludes inserts beyond the limits defined in Clause 1.4.11 must be disclosed. This includes the maximum number of rows that can be inserted and the maximum key value for these new rows. (8.1.2.3)

Insert and delete functionality was fully operational during the benchmark.

Horizontal and Vertical 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. (8.1.2.4)

Partitioning was not used in this benchmark.

Replication

Replication of tables, if used, must be disclosed (see Clause 1.4.6). (8.1.2.5)

Replication was not used in this benchmark.

Table Attributes

Additional and/or duplicated attributes in any table must be disclosed along with a statement on the impact on performance (see Clause 1.4.7). (8.1.2.6)

No 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. (8.1.3.1)

The seeds and offsets for the random number generator were collected and verified to be different for each driver. The auditor selected samples of the generated numbers from the database. The samples were verified to have no discernible patterns.

Screen Layout

The actual layouts of the terminal input/output screens must be disclosed. (8.1.3.2)

The screen layouts are based on those in Clauses 2.4.3, 2.5.3, 2.6.3, 2.7.3, and 2.8.3 of the TPC-C Standard Specification. There are some differences based on the fact that this is a WEB client implementation.

Terminal Verification

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

The terminal features were verified by allowing the auditor to manually execute each of the five transaction types, using Microsoft Internet Explorer version 3.0.

Intelligent Terminals

Any usage of presentation managers or intelligent terminals must be explained. (8.1.3.4)
***Comment 1**: 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.*
***Comment 2**: This disclosure also requires that all data manipulation functions performed by the local terminal to provide navigational aids for transaction(s) must also be described. Within this disclosure, the purpose of such additional function(s) must be explained.*

Application code involved in the manipulation of data was run on the client. Screen manipulation commands in the form of HTML were downloaded to the WEB browser which handled input and output presentation graphics. A listing of this code is included in Appendix A. Microsoft Internet Information Service assisted in the processing and presentation of this data.

Transaction Profiles

The percentage of home and remote order-lines in the New-Order transactions must be disclosed. (8.1.3.5)
The percentage of New-Order transactions that were rolled back as a result of an unused item number must be disclosed. (8.1.3.6)
The number of items per orders entered by New-Order transactions must be disclosed. (8.1.3.7)
The percentage of home and remote Payment transactions must be disclosed. (8.1.3.8)
The percentage of Payment and Order-Status transactions that used non-primary key (C_LAST) access to the database must be disclosed. (8.1.3.9)
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. 8.1.3.10)

Table 1: Transaction Statistics

Transaction	Function	Value
New Order	Home Warehouse Items	99.00%
	Remote Warehouse Items	1.00.%
	Rolled Back Transactions	1.03%
	Average Lines Per Order	9.99%
Payment	Home Warehouse	84.91%
	Remote Warehouse	15.09%
	Non-Primary Key Access	60.10%
Order Status	Non-Primary Key Access	59.95%
Delivery	Skipped Transactions	0

Transaction Mix

The mix (i.e., percentages) of transaction types seen by the SUT must be disclosed. (8.1.3.11)

Table 2: Transaction Mix

Transaction	Percentage
New Order	44.87%
Payment	43.03%
Order Status	4.05%
Delivery	4.04%
Stock Level	4.00%

Deferred Delivery Mechanism

The queuing mechanism used to defer the execution of the Delivery transaction must be disclosed. (8.1.3.12)

The client application submits delivery transactions to asynchronous Tuxedo servers, or queues, running on the client machines. There were multiple delivery servers with single execution threads running on each client machine. These delivery servers were responsible for processing deliveries queued to Tuxedo and submitting them to the database server.

The source code is listed in Appendix A.

Clause 3 -- Transaction and System Properties Related Items

ACID Tests

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. (8.1.4.1)

All ACID property tests were successful. The executions are described below.

Atomicity

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

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

Aborted Transactions

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

Consistency

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

Consistency conditions one through four were tested using a shell script to issue queries to the database. The results of the queries verified that the database was consistent for all four tests. A run was executed under full load lasting over ten (10) minutes and included a checkpoint. The shell script was executed again. 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 defined above (clause 3.4.1) 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 by the auditor to demonstrate the required isolation had been met.

For Isolation test seven, case A was followed.

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.

Durable Media Failure

Durability from media failure was demonstrated on a 10 warehouse database having similar characteristics to the fully scaled database. The standard driving mechanism was used to generate the transaction load of 100 users for the Loss of Data. The fully scaled database under full load would also have passed the following test.

Loss of Log and Loss of Data

The loss of data test was performed on a 10 Warehouse system. 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 medium containing TPC-C tables, the following steps were executed:

1. A 10 warehouse database was built in a manner similar to the full size database.
2. A backup was taken of the database using Microsoft SQL Server facilities.
3. A sum of d_next_o_id was taken and noted.
4. 100 users were logged into the database and proceeded to run transactions.
5. One mirrored log drive was removed from the system. The system continued running.
6. One disk drive was removed from a disk array containing database files.
7. SQL Server reported errors and transactions failed.
8. SQL Server was restarted and a dump of the transaction log was taken.
9. The tpcc database was dropped.
10. The system was shut down and the faulty disk drive was replaced.
11. The 10 warehouse database was recreated.
12. The 10 warehouse database was restored from backup.
13. The saved transaction log was loaded and transactions rolled forward.
14. A sum of d_next_o_id was taken and noted.
15. The number of transactions reported in the database and in the RTE log was compared.

Instantaneous Interruption and Loss of Memory

Because loss of power erases the contents of memory, the instantaneous interruption and the loss of memory tests were combined into a single test. This test was executed on a fully scaled database of 900 warehouses under a full load of 9000 users. The following steps were executed:

1. A sum of d_next_o_id was taken and noted.
2. 9000 users were logged into the database and proceeded to run transactions.
3. System power was terminated.
4. System power was reapplied and NT restarted.
5. SQL Server was started and recovery occured automatically.
6. A sum of d_next_o_id was taken and noted.
7. The number of transactions reported in the database and in the RTE log was compared.

Clause 4 -- Scaling and Database Population Related Items

Table Cardinality

The cardinality (e.g., the number of rows) of each table, as it existed at the start of the benchmark run (see Clause 4.2), must be disclosed. If the database was over-scaled and inactive rows of the WAREHOUSE table were deleted (see Clause 4.2.2), the cardinality of the WAREHOUSE table as initially configured and the number of rows deleted must be disclosed. (8.1.5.1)

Table 3: Table Cardinality

Table	Cardinality as Benchmarked
Warehouse	900
District	9000
Customer	27000000
History	27000000
Orders	27000000
New Order	8100000
Order Line	269999044
Stock	90000000
Item	100000

Constant Values

The following values were used as constant value inputs to the NURand function for this benchmark.

Table 4: Constant Values

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

Data Distribution

The distribution of tables and logs across all media must be explicitly depicted for the tested and priced systems. (8.1.5.2)

Table 5 : Data Distribution

Controller	Drives	Partition	Size	Use
Mylex Controller 1	3 – 9GB	C: & H:		OS, SQL and wdi_seg
	8 – 9GB	D:	30 GB	C_log_dev
Mylex Controller 2	24 – 4GB	I:	8192 MB	C_cust1_dev
		L:	12288 MB	C_stock1_dev
		O:	30720 MB	Tpcdmp1.dat
Mylex Controller 3	24 – 4GB	J:	8192 MB	C_cust2_dev
		M:	12288 MB	C_stock2_dev
		P:	30720 MB	Tpvdmp2.dat

Mylex Controller 4	24 – 4GB	K:	8192 MB	C_cust3_dev
		N:	12288 MB	C_stock3_dev
		Q:	30720 MB	Tpcdmp3.dat
Mylex Controller 5	21 – 9GB	E:	30720 MB	C_ol1_dev
		F:	30720 MB	C_ol2_dev
		G:	10240 MB	C_grow_dev

***Comment**: Detailed diagrams for layout of database files on disks can widely vary, and it is difficult to provide exact guideline suitable for all implementations. The intent is to provide sufficient detail to allow independent reconstruction of the test database. The two figures below are examples of database layout descriptions and are not intended to depict or imply any optimal layout for the TPC-C database.*
8.1.5.3 A statement must be provided that describes:

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

Microsoft SQL Server v6.5 is a relational DBMS.

The interface used was Microsoft SQL Server stored procedures accessed with Remote Procedure Calls embedded in C code using the Microsoft DBLIB interface.

Partition Mapping

The mapping of database partitions/replications must be explicitly described.
***Comment**: The intent is to provide sufficient detail about partitioning and replication to allow independent reconstruction of the test database. (8.1.5.4)*
An description of a database partitioning scheme is presented below as an example. The nomenclature of this example was outlined using the CUSTOMER table (in Clause 8.1.2.1), and has been extended to use the ORDER and ORDER_LINE tables as well.

The database was not replicated.

180 Day Space Calculation

Details of the 180 day space computations along with proof that the database is configured to sustain 8 hours of growth for the dynamic tables (Order, Order-Line, and History) must be disclosed (see Clause 4.2.3). (8.1.5.5)

1. To calculate the space required to sustain the database log for 8 hours of growth at steady state, the following steps were followed:
2. The free space on the log file was queried using *DBCC sqlperf(logspace).*
3. Transactions were run against the database with a full load of users.
4. The free space was again queried using *DBCC sqlperf(logspace).*
5. The space used was calculated as the difference between the first and second query.
6. The number of NEW-ORDERS was verified from an RTE report covering the entire run.
7. The space used was divided by the number of NEW-ORDERS giving a spaceused per NEW-ORDER transaction.

8. 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 28.78GB(including mirror) to sustain the log for 8 hours. Space available on the transaction log volume was 29.29GB (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.

The details of the 180-day space requirement is shown in Appendix D.

Clause 5 -- Performance Metrics and Response Time Related Items

Measured tpmC

Measured tpmC must be reported. (8.1.6.1)

Measured tpmC **11,072.07 tpmC**
Price per tpmC **$27.25 / 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. (8.1.6.2)

Table 5: Transaction Response Times

Transaction	Average	Maximum	90%
New Order	0.65	113.71	0.97
Payment	0.33	23.76	0.42
Order Status	0.91	25.87	1.37
Interactive Delivery	0.28	0.82	0.28
Deferred Delivery	0.57	7.37	0.82
Stock Level	2.90	61.09	4.65
Menu	0.20	1.10	0.31

Think Times & Key Times

The minimum, the average, and the maximum keying and think times must be reported for each transaction type. (8.1.6.3)

Table 6: Transaction Key Times

Transaction	Average	Max	Min
New Order	18.03	18.10	18.02
Payment	3.03	3.14	3.02
Order Status	2.03	2.05	2.02
Delivery	2.03	2.16	2.02
Stock Level	2.03	2.05	2.02

Table 7: Transaction Think Times

Transaction	Average	Max	Min
New Order	12.06	121.00	0.00
Payment	12.10	121.00	0.00
Order Status	10.16	100.99	0.00
Delivery	5.14	51.00	0.00
Stock Level	5.02	50.26	0.00

Response Time Distribution Curves

Response Time frequency distribution curves (see Clause 5.6.1) must be reported for each transaction type. (8.1.6.4)

Figure 3: New Order Response Time Distribution

Figure 4: Payment Response Time Distribution

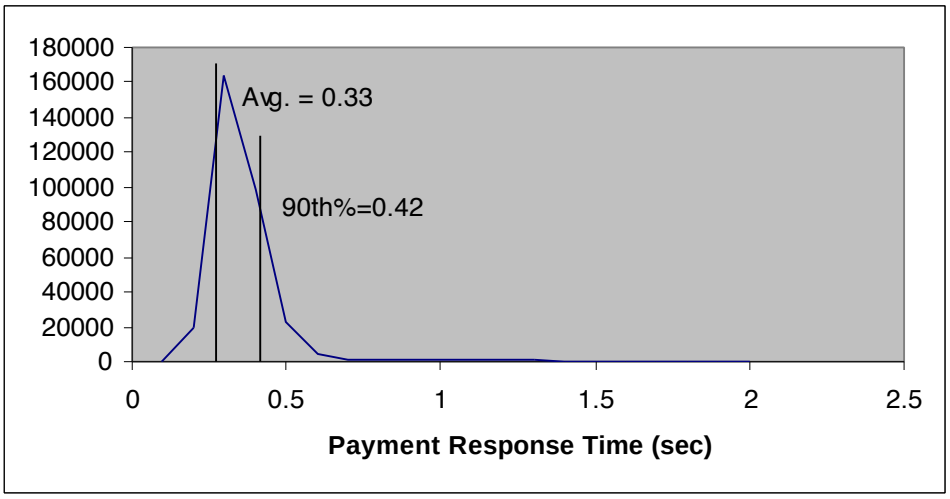


Figure 5: Order Status Response Time Distribution

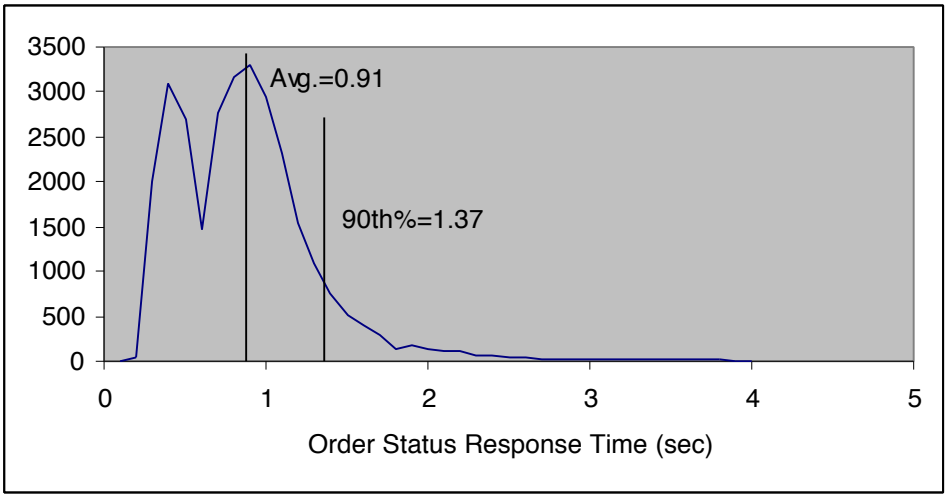


Figure 6: Delivery Response Time Distribution

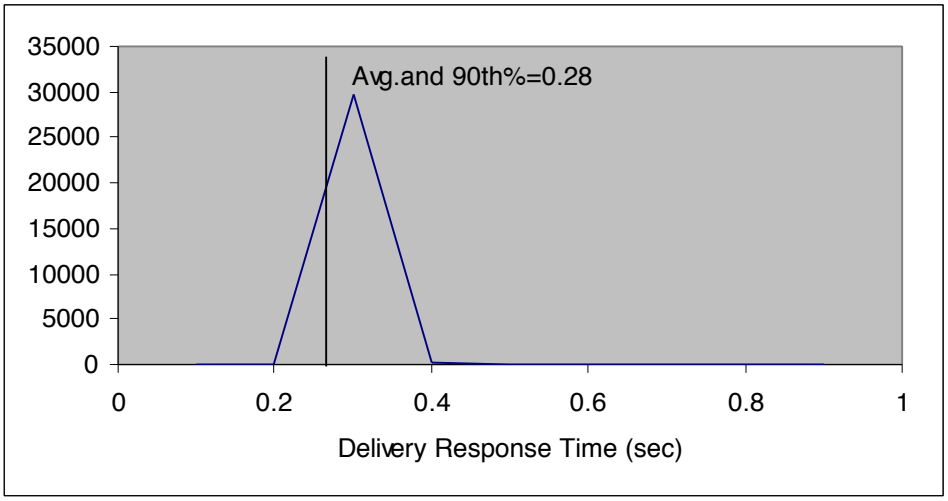
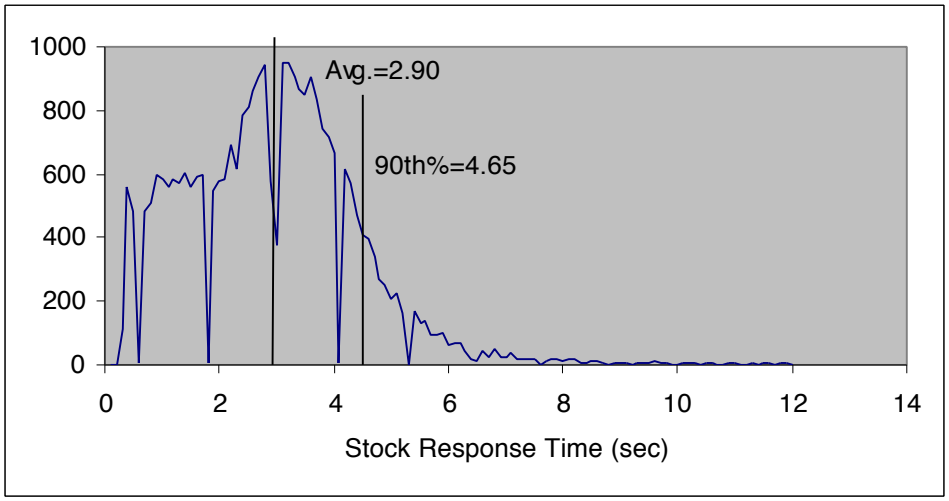


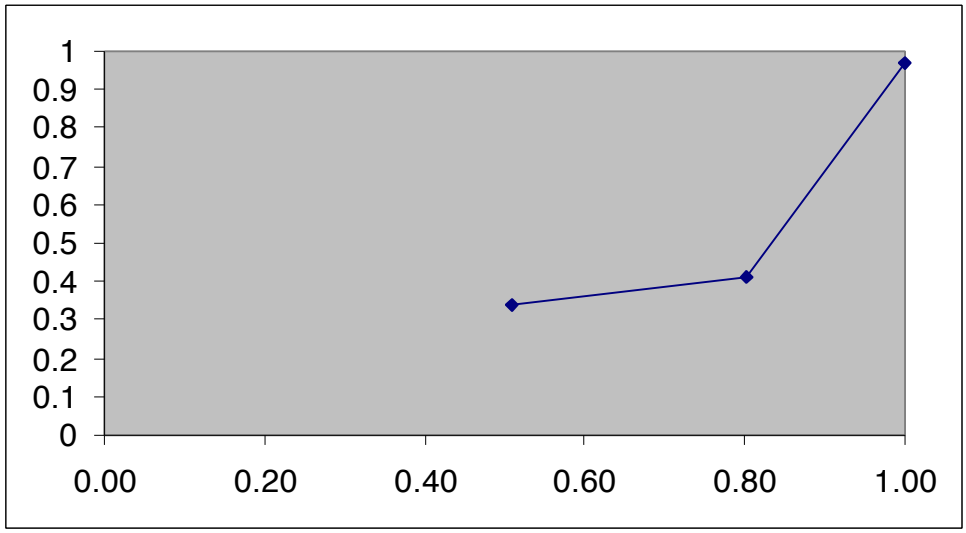
Figure 7: Stock Level Response Time Distribution



New-Order Response Time vs. Throughput Graph

The performance curve for response times versus throughput (see Clause 5.6.2) must be reported for the New-Order transaction. (8.1.6.5)

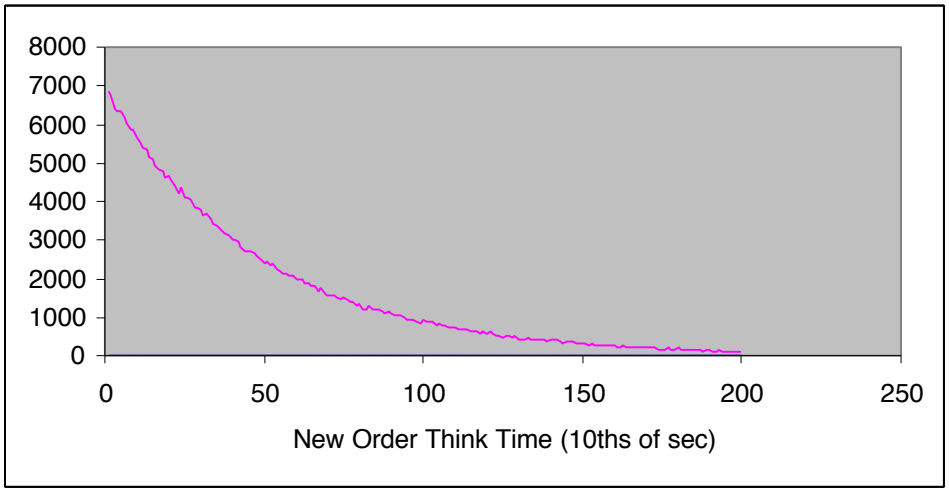
Figure 8: New Order Response Time vs. Throughput



New-Order Think Time Distribution Graph

Think Time frequency distribution curves (see Clause 5.6.3) must be reported for the New-Order transaction. (8.1.6.6)

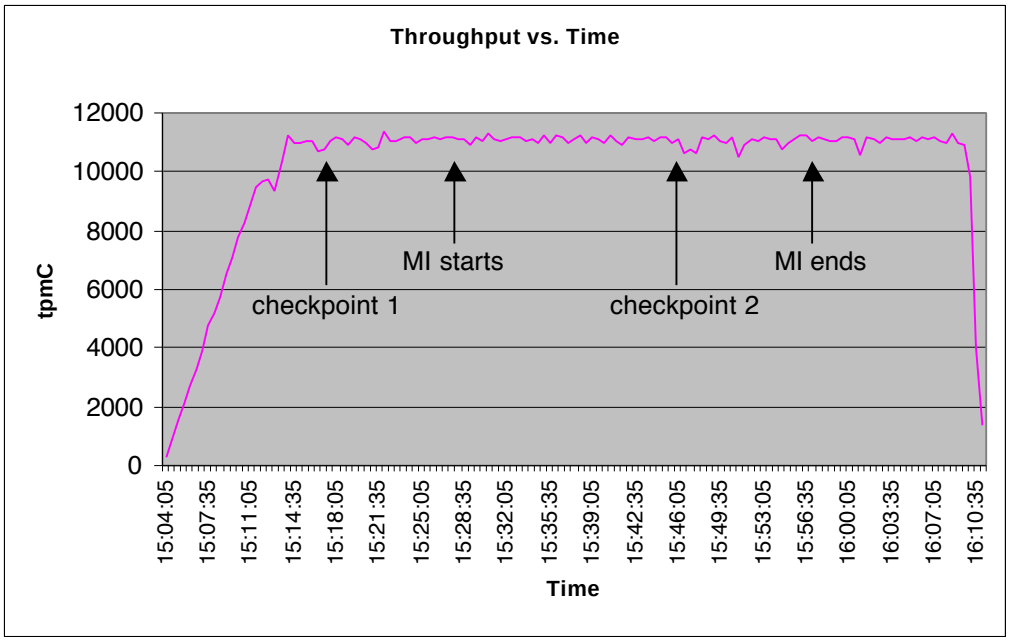
Figure 9: New Order Think Time Distribution



Steady-State Graph

A graph of throughput versus elapsed time (see Clause 5.6.5) must be reported for the New-Order transaction. (8.1.6.8)

Figure 10: New Order Throughput vs. Time



Steady-State Methodology

The method used to determine that the SUT had reached a steady state prior to commencing the measurement interval (see Clause 5.5) must be described. (8.1.6.9)

Steady state was determined using the RTE. Steady state was further confirmed by the throughput data collected during the run and graphed in Figure 10.

Work Performed During Steady State

A description of how the work normally performed during a sustained test (for example checkpoint ing, writing redo/undo log records, etc.), actually occurred during the measurement interval must be reported. (8.1.6.10)

The RTE generated the required input data to choose a transaction from the menu. This data was timestamped. The response for the requested transaction was verified and timestamped in the RTE log files.

The RTE generated the required input data for the chosen transaction. It waited to complete the minimum required key time before transmitting the input screen. The transmission was timestamped. The return of the screen with the required response data was timestamped. The difference between these two timestamps was the response time for that transaction and was logged in the RTE log.

The RTE then waited the required think time interval before repeating the process starting at selecting another transaction from the menu.

The RTE transmissions were sent to application processes running on the client machines through Ethernet LANs. These client application processes handled all screen I/O as well as all requests to the database on the server. The applications communicated with the database server over another Ethernet LAN using Microsoft SQL Server DBLIB library and RPC calls.

To perform checkpoints at specific intervals, we set SQL Server *recovery interval* to the maximum allowable value and wrote a script to schedule multiple checkpoints at specific intervals. By setting the TRACE FLAG #3502, SQL Server logged the checkpoint beginning and ending time in the ERRORLOG file. The script included a wait time between each checkpoint equal to the measurement interval, which was 30 minutes. The checkpoint script was started manually after the RTE had all users logged in and sending transactions.

At each checkpoint, Microsoft SQL Server wrote to disk all memory pages that had been updated but not yet physically written to disk. Upon completion of the checkpoint, Microsoft SQL Server wrote a special record to the recovery log to indicate that all disk operations had been satisfied to this point. The positioning of the checkpoint was verified to be clear of the guard zones and is depicted on the graph in Figure 10.

Reproducibility Methodology

A description of the method used to determine the reproducibility of the measurement results must be reported. (8.1.6.11)

Multiple performance runs were tested and verified. All compliant tests were within 1% of the reported measurement result.

Measurement Interval

A statement of the duration of the measurement interval for the reported Maximum Qualified Throughput (tpmC) must be included. (8.1.6.12)

The measurement interval was 30 minutes.

Transaction Mix

8.1.6.13 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. (8.1.6.13)

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

The percentage of the total mix for each transaction type must be disclosed. (8.1.6.14)

Table 8: Transaction Mix

Transaction	Percentage
New Order	44.87%
Payment	43.03%
Order Status	4.05%
Delivery	4.04%
Stock Level	4.00%

Other Metrics

The percentage of New-Order transactions rolled back as a result of invalid item number must be disclosed. (8.1.6.15)
The average number of order-lines entered per New-Order transaction must be disclosed. (8.1.6.16)
The percentage of remote order-lines entered per New-Order transaction must be disclosed. (8.1.6.17)
The percentage of remote Payment transactions must be disclosed. (8.1.6.18)
The percentage of customer selections by customer last name in the Payment and Order-Status transactions must be disclosed. (8.1.6.19)
The percentage of Delivery transactions skipped due to there being fewer than necessary orders in the New-Order table must be disclosed. (8.1.6.20)

Table 9: Transaction Statistics

Transaction	Function	Value
New Order	Home Warehouse Items	99.00%
	Remote Warehouse Items	1.00.%
	Rolled Back Transactions	1.03%
	Average Lines Per Order	9.99%
Payment	Home Warehouse	84.91%
	Remote Warehouse	15.09%
	Non-Primary Key Access	60.10%
Order Status	Non-Primary Key Access	59.95%
Delivery	Skipped Transactions	0

Checkpoints

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. (8.1.6.21)

The first checkpoint was started 12 minutes into the rampup. The second checkpoint was started 996 seconds into the measurement interval. The checkpoint in the measurement interval lasted 5 minutes 9 seconds. The measurement interval contains the second checkpoint, and is clear of the guard zones.

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

RTE Parameters

The RTE input parameters, code fragments, functions, etc. used to generate each transaction input field must be disclosed. (8.1.7.1)
***Comment**: The intent is to demonstrate the RTE was configured to generate transaction input data as specified in Clause 2.*

The RTE used was the Microsoft BenchCraft RTE System. The RTE parameters are 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 that of the priced system. The results of the test described in Clause 6.6.3.4 must be disclosed. (8.1.7.2)

No components were emulated.

Benchmarked and Targeted System Configuration Diagrams

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 software and hardware functionality being performed on the Driver System, and its interface to the SUT must be disclosed (see Clause 6.6.3.6). (8.1.7.3)

The driver system performed the data generation and communication to the client through the standard web browser (HTTP) protocol. It also captured and timestamped the SUT output data for post-processing of the reported metrics. No other functionality was included on the driver system.

Figures 1 & 2 of this report contain detailed diagrams of both the benchmark configuration and the priced configuration.

Network Configuration

The network configurations of both the tested services and the 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 (see Clause 6.6.4). (8.1.7.4)

The network configuration of the benchmarked and priced configurations differed as follows:
100 BaseT(100 Mbit/sec) between the Clients and Server was priced, and 10BaseT was actually run in the test.

Network Bandwidth

The bandwidth of the network(s) used in the tested/priced configuration must be disclosed. (8.1.7.5)

The bandwidth of the tested and priced networks were as follows:
- 10 BaseT (10 Mbit/sec) network segments between the RTE/Emulated Users and the Clients.
- 100 BaseT(100 Mbit/sec) between the Clients and Server was priced, and 10BaseT was actually run in the test.

Operator Intervention

If the configuration requires operator intervention (see Clause 6.6.6), the mechanism and the frequency of this intervention must be disclosed. (8.1.7.6)

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

Clause 7 -- Pricing Related Items

Hardware and Software List

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

Pricing source(s) and effective date(s) of price(s) must also be reported. (8.1.8.1)

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. (8.1.8.2)

The details of the hardware and software are reported in the front of this report as part of the executive summary. All third party quotations are included at the end of this report as Appendix E.

Availability Date

The committed delivery date for general availability (availability date) of products used in the price calculations must be reported. When the priced system includes products with different availability dates, the reported availability date for the priced system must be the date at which all components are committed to be available. (8.1.8.3)

Hardware Availability Date: February 16, 1998
Software Availability Date: February 16, 1998

Measured TpmC

A statement of the measured tpmC, as well as the respective calculations for 5-year pricing , price/performance (price/tpmC), and the availability date must be included. (8.1.8.4)

Maximum Qualified Throughput: **11,072.07 tpmC**
Price Performance Metric: **$27.25 per tpmC**

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. (8.1.8.5)

This system is priced for the United States of America.

Usage Pricing

For any usage pricing, the sponsor must disclose (8.1.8.6):

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

***Comment**: Usage pricing may include, but is not limited to, the operating system and database management software.*

The component pricing based on usage is shown below:

- 5 Microsoft Windows NT 4.0 Licenses
- 1 Microsoft Windows NT 4.0 Enterprise Edition license
- 1 Microsoft SQL Server, Enterprise Edition, v.6.50 License.
- 1 Microsoft Internet Connector License
- 1 Microsoft SQL Server Programmers Toolkit
- 1 Microsoft Visual C++ version 4.2
- 5 years support for all components.

System Pricing

System pricing should include subtotals for the following components: Server Hardware, Server Software, Client Hardware, Client Software, and Network Components used for terminal connection (see Clause 7.2.2.3). Clause 6.1 describes the Server and Client components. An example of the standard pricing sheet is shown in Appendix B. (8.1.8.7)
System pricing must include line item indication where non-sponsoring companies' brands are used. System pricing must also include line item indication of third party pricing. See example in Appendix B. (8.1.8.8)
***Comment:** By standardizing the pricing spreadsheet and adding subtotals the value of the FDR and executive summary will be enhanced. This will allow the reader to more easily compare results and determine pricing.*

The details of the hardware and software are reported in the front of this report as part of the executive summary. All third party quotations are included at the end of this report as Appendix E.

Clause 9 -- Audit Related Items

Auditor

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 . (8.1.9.1)
A review of the pricing model is required to ensure that all components required are priced (see Clause 9.2.8). The auditor is not required to review the final Full Disclosure Report or the final pricing prior to issuing the attestation letter. (8.1.9.2)

This TPC-C benchmark was audited by Lorna Livingtree of Performance Metrics, Inc.

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:

Acer Inc.
7F, 88, Sec. 1, Hsin Tai Wu Rd.,
Hsichih, Taipei Hsien 221,
Taiwan, R.O.C.

PERFORMANCE METRICS INC.
TPC Certified Auditors

February 9, 1998

Mr. Duncan Liu
Project Researcher
Acer Inc.
21F, 88, Sec. 1, Hsin Tai Wu Rd.,
Hsichih, Taipei Hsien 221,
Taiwan, ROC
劉建成
宏碁電腦股份有限公司
台北縣 221 汐止鎮新台五路一段 88 號 21 樓

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

Platform: AcerAltos 19000Pro4
Database Manager: Microsoft SQL Server 6.5 Enterprise Edition
Operating System: Miscrosoft NT Server 4.0 Enterprise Edition
Transaction Monitor: Tuxedo 6.3

Server: AcerAltos 19000Pro4				
CPU's	Memory	Disks	90% Response	tpmC
4 Pentium Pro @ 200 Mhz	Main: 4 GB Cache: 1MB each	72 @ 4.5GB 24 @ 9.1GB	0.97 sec	11,072.07
5 Clients: AcerAltos900Pro				
1 Pentium Pro @ 200 MHz	Main: 191 MB Cache: 512K	1 @ 2 GB	na	na

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

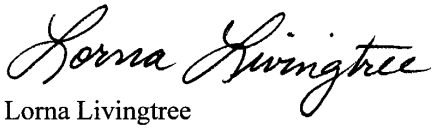
- The transactions were correctly implemented.

2229 Benita Dr. Suite 101, Rancho Cordova, CA 95670
(916) 635-2822 fax: (916) 858-0109 email: Lorna@PerfMetrics.com

- The ACID tests were performed on a database scaled to 10 warehouses.
- Input data was generated according to the specified percentages.
- Eight hours of mirrored log space was present on the tested system.
- Eight hours of growth space for the dynamic tables was present on the tested system.
- The data for the 180 day space calculation was verified.
- The steady state portion of the test was 30 minutes.
- One checkpoint was taken before the measured interval.
- One checkpoint was taken during the measured interval.
- The checkpoints were verified to be clear of the guard zone.
- The system pricing was checked for major components and maintenance.
- Third party quotes were verified for compliance.
- The priced disks met all requirements for disk substitution.

Auditor Notes: None.

Sincerely,



Lorna Livingtree
Auditor

2229 Benita Dr. Suite 101, Rancho Cordova, CA 95670
(916) 635-2822 fax: (916) 858-0109 email: Lorna@PerfMetrics.com

Appendix A – Source Code

Microsoft/PTC WEB Client

Makefile

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

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

!IF "$(TUXDIR)" == ""
TUXDIR    = C:\TUXEDO
!MESSAGE No TUXDIR specified.  Defaulting to E:\TUXEDO
!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
DBLIB           = $(SQL_LOC)
DBLIBINC        = $(DBLIB)\include
DBLIBDIR        = $(DBLIB)\lib

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


LINK32_LIBS     = user32.lib msacm32.lib advapi32.lib
$(DBLIBDIR)\ntwdblib.lib
TUX_LIBS  = $(TUXDIR)\lib\libtux.lib
$(TUXDIR)\lib\libbuft.lib $(TUXDIR)\lib\libtux2.lib  \
$(TUXDIR)\lib\libfml.lib $(TUXDIR)\lib\libfml32.lib
$(TUXDIR)\lib\libgp.lib
OTHER_LIBS          = wsock32.lib kernel32.lib
gdi32.lib comdlg32.lib winspool.lib
LINK32_OBJS     = "$(OBJDIR)\tpcc.obj"
"$(OBJDIR)\tpcc.res"
LINK32_DEF      = "$(SRCDIR)\tpcc.def"
LINK32_FLAGS    = /nologo /subsystem:windows /dll
/incremental:no $(LDEBUG) /def:"$(LINK32_DEF)"
/out:"$(OUTDIR)\tpcc.dll"

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

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

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

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

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

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

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

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

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

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

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

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

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

Delivery.c

```
/*        FILE:               DELIVERY.C
 *
 *        Based on: Microsoft TPC-C Kit Ver. 3.00.000
 *
 *                                      Copyright
Microsoft, 1996
 *                                      Copyright
Performance Tuning Corporation, 1997
 *
 *        PURPOSE:  New Order Tuxedo Server.
 *        Author:             Philip Durr
 *
          philipdu@Microsoft.com
 *
```

```
 *  MODIFIED        Changed for modularity and to allow
for the Tuxedo TM
 *
 *  Author:                     Edward Whalen
 *                                      Performance
Tuning Corporation
 *
         ewhalen@perftuning.com
 */

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

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

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

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

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

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

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

DBPROCESS *pdbproc;

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

#ifdef LOCAL_ALLOC
DELIVERY_DATA DeliveryData;
#else
TUX_DATA TuxData;
#endif
TERM Term;

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

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

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

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

#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.

FILE      *fpLog;

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

int tpsvrinit ( int argc, char *argv[] )
{

          if ( GetParameters(argc, argv) )
          {
                    PrintParameters();
                    return -1;
          }

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

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

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

          OpenLogFile();

          return 0;
}

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

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

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

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

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

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

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

          bError = FALSE;

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

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

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

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

          bError = FALSE;

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

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

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


/* FUNCTION: void CalculateElapsedTime(int *pElapsed,
LPSYSTEMTIME lpBegin, LPSYSTEMTIME lpEnd)
 *
 * PURPOSE:         This function calculates the
elapsed time a delivery transaction took.
 *
 * ARGUMENTS:       int
          *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(DBPROCESS *dbproc,
DELIVERY *pDelivery, short deadlock_retry )
 *
 * PURPOSE:         This function processes the
delivery transaction.
 *
 * ARGUMENTS:       DELIVERY            *pDelivery
          Pointer to delivery transaction
structure
 *
 * RETURNS:                  int
          ERR_DBGETDATA_FAILED
          Delivery get data operation failed.
 *
          ERR_SUCCESS
                    Delivery successfull, no error
 *
 *
 * COMMENTS:        None
 *
 */

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

          deadlock_count = 0;


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

                              if (dbrpcexec(dbproc) ==
SUCCEED)
                              {
                                        while (((rc =
dbresults(dbproc)) != NO_MORE_RESULTS) && (rc != FAIL))
                                        {
```

```
                                                  while
(((rc = dbnextrow(dbproc)) != NO_MORE_ROWS) && (rc !=
FAIL))
                                                  {

          for (i=0;i<10;i++)

          {

                    if(pData=dbdata(dbproc, i+1))

                              pDelivery->o_id[i] =
*((DBINT *)pData);

                    else

                              pDelivery->o_id[i] = 0;

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

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

          fprintf(fpLog,
"%2.2d/%2.2d/%2.2d,%2.2d:%2.2d:%2.2d:%3.3d,%2.2d:%2.2d:
%2.2d:%3.3d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d\r\n"
,
                    trans_end.wYear - 1900,
trans_end.wMonth, trans_end.wDay,
                    pDelivery->queue_time.wHour,
pDelivery->queue_time.wMinute,
                    pDelivery->queue_time.wSecond,
pDelivery->queue_time.wMilliseconds,
                    trans_end.wHour, trans_end.wMinute,
trans_end.wSecond, trans_end.wMilliseconds,
                    elapsed,
                    pDelivery->w_id, pDelivery-
>o_carrier_id,
                    pDelivery->o_id[0], pDelivery-
>o_id[1], pDelivery->o_id[2], pDelivery->o_id[3],
                    pDelivery->o_id[4], pDelivery-
>o_id[5], pDelivery->o_id[6], pDelivery->o_id[7],
                    pDelivery->o_id[8], pDelivery-
>o_id[9] );

          if ( bFlush )
                    fflush(fpLog);


          return ERR_SUCCESS;
}

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

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

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

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

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

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

          if ( bRc )
                    return ERR_REGISTRY_NOT_SETUP;

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

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

          fpLog = fopen(szLogPath, "ab");

          if ( !fpLog )
                    return
ERR_CANNOT_CREATE_RESULTS_FILE;

          return ERR_SUCCESS;
}

/*
 *        Common Code for all Servers
 */

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



```
 *
 */
BOOL SQLInit ()
{
          dbinit();

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

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

          return TRUE;
}


/* FUNCTION: BOOL
SQLOpenConnection(EXTENSION_CONTROL_BLOCK
          *pECB, 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, long *pack_size)
          {

                    RETCODE   rc;
                    char      buffer[30];

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

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


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

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

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

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

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

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

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

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

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


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

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

                    return FALSE;
          }

#else

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


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

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

                    gpECB = pECB;

                    login = dblogin();
```

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

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

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

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

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

                    //set pECB data into dbproc
                    pEcbInfo =
(PECBINFO)malloc(sizeof(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

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

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

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

          return;
}

/* FUNCTION: int err_handler(DBPROCESS *dbproc, int
severity, int dberr, int oserr, char *dberrstr, char
*oserrstr)
 *
 * PURPOSE:         This function handles DB-Library
errors
 *
 * ARGUMENTS:       DBPROCESS           *dbproc
                    DBPROCESS id pointer
 *                                  int
                    severity
          severity of error
```

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


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

          pEcbInfo = NULL;

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

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

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

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

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

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

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

                    fclose(fp);
          }
          return INT_CANCEL;
}


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

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

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


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

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

          if (msgno == 0)
                    return INT_CONTINUE;
```

```
          else
          {
                    TMLog("MsgHandler: SQL Error %s",
msgtext);

                    if ( pEcbInfo )
                              pEcbInfo->bFailed = TRUE;

                    bError = TRUE;

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

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


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

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

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

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

          strcpy(szServer, argv[i]+2);

          break;
                                        case 'V':
                                        case 'v':

          verbose = TRUE;

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

}
```

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
 */

//#define LOCAL_ALLOC 1

//VERSION RESOURCE DEFINES
#define _APS_NEXT_RESOURCE_VALUE
                    101
#define _APS_NEXT_COMMAND_VALUE
                    40001
#define _APS_NEXT_CONTROL_VALUE
                    1000
#define _APS_NEXT_SYMED_VALUE
                    101

#define TP_MAX_RETRIES
                                    50

#define ERR_BAD_ITEM_ID
                                    1
          //expected abort record in txnRecord
#define ERR_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_TYPE_TUXEDO
                                        9
          //tuxedo error

#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*".
#define ERR_TPINIT_BAD
                                        5001
          //"Bad TPINIT"
#define ERR_TPALLOC_BAD
                                        5002
          //"Bad TPALLOC"
#define ERR_TPCALL_BAD
                                        5003
          //"Bad TPCALL"

//note that the welcome form must be processed first as
terminal ids assigned here, once the
//terminal id is assigned then the forms can be
processed in any order.
#define WELCOME_FORM
                              1
          //beginning form no term id assigned, form id
#define MAIN_MENU_FORM
                              2
          //term id assigned main menu form id
#define NEW_ORDER_FORM
                              3                   //new
order form id
#define PAYMENT_FORM
                              4
          //payment form id
#define DELIVERY_FORM
                              5
          //delivery form id
#define ORDER_STATUS_FORM
                    6                   //order status
id
#define STOCK_LEVEL_FORM
                    7                   //stock level
form id

//This macro is used to prevent the compiler error
unused formal parameter
#define UNUSEDPARAM(x) (x = x)

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

//This structure is used for posting delivery
transactions
typedef struct _DELIVERY_TRANSACTION
{
          SYSTEMTIME          queue;
          //time delivery transaction queued
          short               w_id;
          //delivery warehouse
          short               o_carrier_id;
          //carrier id
} DELIVERY_TRANSACTION;

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

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

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

          PDBPROCESS          dbproc;
                              //dblib connection
pointer
          int                           spid;
                                        //spid assigned
from dblib
          int                           iSyncId;
                              //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

#ifndef LOCAL_ALLOC
          TUX_DATA                      *TuxDataPtr;
                    //Tuxedo Data Structure for all
transactions
#endif // LOCAL_ALLOC

} CLIENTDATA;

typedef CLIENTDATA *PCLIENTDATA;
          //pointer to client structure

//This structure is used to define the operational
interface for terminal id support
typedef struct _TERM
{
          int                           iAvailable;

          //total allocated terminal array entries
          int                           iNext;

          //next available terminal array element
```

```
          int                              iMasterSyncId;

          //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 LogTuxError(int TpRc, char *ErrMessage);
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);
static int SQLDelivery(DBPROCESS *dbproc, DELIVERY_DATA
*pDelivery, short deadlock_retry);
static void WriteLog(DELIVERY_DATA *pDelivery,
SYSTEMTIME *trans_end );
static void CalculateElapsedTime(int *pElapsed,
LPSYSTEMTIME lpBegin, LPSYSTEMTIME lpEnd);
BOOL SQLDetectDeadlock(DBPROCESS *dbproc);
static void FormatString(char *szDest, char *szPic,
char *szSrc);
static char *MakeStockLevelForm(int iTermId, int
iSyncId, BOOL bInput);
static char *MakeMainMenuForm(int iTermId, int
iSyncId);
static char *MakeWelcomeForm(void);
static char *MakeNewOrderForm(int iTermId, int iSyncId,
BOOL Rollback, 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);
static void         PrintParameters(void);
```

```
#ifdef USE_ODBC
          void dbsetuserdata(PDBPROCESS dbproc, void
*uPtr);
          void *dbgetuserdata(PDBPROCESS dbproc);
          void BindParameter(PDBPROCESS dbproc, UWORD
ipar, SWORD fCType, SWORD fSqlType, UDWORD cbColDef,
SWORD ibScale, PTR rgbValue, SDWORD cbValueMax);
          void ODBCError(PDBPROCESS dbproc);
          BOOL ExecuteStatement(PDBPROCESS dbproc, char
*szStatement);
          BOOL BindColumn(PDBPROCESS dbproc,
SQLUSMALLINT icol, SQLSMALLINT fCType, SQLPOINTER
rgbValue, SQLINTEGER cbValueMax);
          BOOL GetResults(PDBPROCESS dbproc);
          BOOL MoreResults(PDBPROCESS  dbproc);
          BOOL 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;
                    int
                    retval;
                    int
                    error;
                    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;
                int
                 retval;
                int
                 error;
                char
          execution_status[STATUS_LEN];
          } PAYMENT_DATA;

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

          typedef struct
          {
                short               w_id;
                short               d_id;
                long                c_id;
                char
c_first[FIRST_NAME_LEN+1];
                char
c_middle[MIDDLE_NAME_LEN+1];
                char
c_last[LAST_NAME_LEN+1];
                double              c_balance;
                long                o_id;
          #ifdef USE_ODBC
                TIMESTAMP_STRUCT    o_entry_d;
          #else
                DBDATEREC           o_entry_d;
          #endif
                short               o_carrier_id;
                OL_ORDER_STATUS_DATA
OlOrderStatusData[MAX_OL_ORDER_STATUS_ITEMS];
                short               o_ol_cnt;
                long                num_deadlocks;
                int
                 retval;
                int
                 error;
                char
execution_status[STATUS_LEN];
          } ORDER_STATUS_DATA;

          typedef struct
          {
                long
          o_id;
          } DEL_ITEM;

          typedef struct
          {
                short                         w_id;
                short
          o_carrier_id;
                SYSTEMTIME
          queue_time;
                long
          num_deadlocks;
                long
          o_id[10];
                int
                retval;
                int
                error;
                char
          execution_status[STATUS_LEN];
          } DELIVERY_DATA;

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

          typedef struct
          {
                NEW_ORDER_DATA
          NewOrderData;               //new order
form data
                PAYMENT_DATA
          PaymentData;                //payment form
data
                ORDER_STATUS_DATA
          OrderStatusData;    //order status form data
                DELIVERY_DATA
          DeliveryData;               //delivery form
data
                STOCK_LEVEL_DATA    StockLevelData;
                //stock level form data
          } TUX_DATA;

#endif
```

Tpcc.c

```
/*        FILE:               TPCC.C
 *
 *        Based on: Microsoft TPC-C Kit Ver. 3.00.000
 *
 *                                      Copyright
Microsoft, 1996
 *                                      Copyright
Performance Tuning Corporation, 1997
 *
 *        PURPOSE:  TPC-C main program.
 *        Author:             Philip Durr
 *
          philipdu@Microsoft.com
 *
```

```
 *  MODIFIED        Changed for modularity and to allow
for the Tuxedo TM
 *
 *  Author:                     Edward Whalen
 *                                      Performance
Tuning Corporation
 *
          ewhalen@perftuning.com
 *
*/


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

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

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

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

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

static TPINIT       *tpinf;
static DWORD        TLSIsTpInitedKey;
static int                  ThrTpInit();

char      szServer[32]        = "EDW";
          //global variables used with this DLL
char      szUser[32]                  = "sa";
char      szPassword[32]      = "";
char      szDatabase[32]      = "tpcc";

BOOL      bLog                        = FALSE;
BOOL      dLog                        = FALSE;

int                 iThreads            = 5;
int                 iMaxWareHouses      = 500;
int                 iQSlotts            = 3000;
int                 iDelayMs            = 100;
int                 iConnectDelay       = 500;
short     iDeadlockRetry      = (short)3;
short     iMaxConnections = (short)25;
int                 iErrVal                     = 0;

//char buffer[256];

//allowable client command strings i.e. CMD=command
char *szCmds[] =
{
          "..NewOrder..", "..Payment..",
"..Delivery..", "..Order-Status..", "..Stock-Level..",
"..Exit..",
          "Submit", "Begin", "Process", "Menu",
"Clear", "Users", ""
};

//defined command string functions, called via
CMD=command http string from html client.

void (*DoCmd[])(EXTENSION_CONTROL_BLOCK *pECB, int
iFormId, int iTermId, int 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=\"ERROR\" 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    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(&ErrorLogCriticalSe
ction);

                              (*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;

                              TLSIsTpInitedKey =
TlsAlloc(); // check for failure later
                              // assumption:value
inited to 0
                              break;
                    case DLL_THREAD_ATTACH:
                              break;
                    case DLL_THREAD_DETACH:
                              if ( dLog )
                              {
                                        SYSTEMTIME
          systemTime;
                                        FILE *fp;


          GetLocalTime(&systemTime);
                                        fp =
fopen(szErrorLogPath, "ab");
                                        fprintf(fp,
"\r\nError: %2.2d/%2.2d/%2.2d %2.2d:%2.2d:%2.2d\r\n",

          systemTime.wYear, systemTime.wMonth,
systemTime.wDay,

          systemTime.wHour, systemTime.wMinute,
systemTime.wSecond);
                                        fprintf(fp,
"DLL_THREAD_DETACH \r\n");
                                        fclose(fp);
                              }
                              break;
                    case DLL_PROCESS_DETACH:
                              if ( pSD )
                                        free( pSD );

                              bTpccExit = TRUE;

                              (*Term.Restore)();


          DeleteCriticalSection(&CriticalSection);

          DeleteCriticalSection(&ErrorLogCriticalSectio
n);


          TlsFree(TLSIsTpInitedKey);
                              break;
          }
          return TRUE;
}


/* FUNCTION: BOOL WINAPI
GetExtensionVersion(HSE_VERSION_INFO *pVer)
 *
 * PURPOSE:         This function is called by the inet
service when the DLL is first loaded.
 *
 * ARGUMENTS:       HSE_VERSION_INFO    *pVer
          passed in structure in which to place
expected version number.
 *
 * RETURNS:                    TRUE      inet service
expected return value.
 *
 * COMMENTS:        None
 *
 */

BOOL WINAPI GetExtensionVersion(HSE_VERSION_INFO *pVer)
{

          pVer->dwExtensionVersion =
MAKELONG(HSE_VERSION_MINOR, HSE_VERSION_MAJOR);
          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;

//        static BOOL bReadRegistry = FALSE;

          if ( iMaxConnections == -1 )
          {
                    ErrorMessage(pECB,
ERR_CAN_NOT_SET_MAX_CONNECTIONS, ERR_TYPE_WEBDLL, NULL,
-1, -1);
                    return HSE_STATUS_SUCCESS;
          }

          //if registry setting is for html logging
then show http string passed in.
          if ( bLog )
          {
                    SYSTEMTIME          systemTime;

                    fp = fopen(szTpccLogPath, "ab");

                    GetLocalTime(&systemTime);

                    fprintf(fp, "* QUERY *
%2.2d/%2.2d/%2.2d %2.2d:%2.2d:%2.2d\r\n\r\n%s\r\n\r\n",
                              systemTime.wYear,
systemTime.wMonth, systemTime.wDay,
                              systemTime.wHour,
systemTime.wMinute, systemTime.wSecond,
                              pECB->lpszQueryString);
                    fclose(fp);
```



```
        }


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

        if ( TermId != 0 )
        {
                if ( !IsValidTermId(TermId) )
                {
                        ErrorMessage(pECB,
ERR_INVALID_TERMID, ERR_TYPE_WEBDLL, NULL, TermId,
iSyncId);
                        return
HSE_STATUS_SUCCESS_AND_KEEP_CONN;
                }

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

        }

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

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

        //finish up and keep connection
        return HSE_STATUS_SUCCESS_AND_KEEP_CONN;
}

/* FUNCTION: static BOOL IsValidTermId(int TermId)
 *
 * PURPOSE:         This function checks to see of the
passed in terminal id is valid.
 *
 * ARGUMENTS:       int
        TermId
        client terminal id
 *
 * RETURNS:                 BOOL      FALSE

        Terminal ID Invalid
 *
        TRUE
                Terminal ID valid
 *
 * COMMENTS:        None
 *
 */

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

/* FUNCTION: BOOL
ProcessQueryString(EXTENSION_CONTROL_BLOCK *pECB, int
*pCmd, int *pFormId, int *pTermId, int *pSyncId)
 *
 * PURPOSE:         This function extracts the relevent
information out of the http command passed in from
 *                          the browser.
 *
 * ARGUMENTS:       EXTENSION_CONTROL_BLOCK       *pECB
                structure pointer to passed in
internet
 *
                                service information.
 *                                      int
                                *pCmd
                        returned command id
 *                                      int
                                *pFormId
        returned active form client browser is on
 *                                      int
                                *pTermId
        returned client terminal id
 *
 * RETURNS:                 BOOL      FALSE

        success
 *
        TRUE
                command passed in is invalid
 *
 * COMMENTS:        If this is the initial connection
i.e. client is at welcome screen then
 *                                      there will not
be a terminal id or current form id if 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, FALSE, 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) );
          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;

          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);


#ifdef PURE_PERFORMIX
          (*pECB->ServerSupportFunction)(pECB->ConnID,
HSE_REQ_DONE_WITH_SESSION, NULL, 0, 0);
#else
          (*pECB->ServerSupportFunction)(pECB->ConnID,
HSE_REQ_SEND_RESPONSE_HEADER, szHeader, &iSize,
(LPDWORD)szHeader1);
#endif

          (*pECB->WriteClient)(pECB->ConnID, szStr,
&lpbSize, 0);

          return;
}

/* FUNCTION: void h_printf(EXTENSION_CONTROL_BLOCK
*pECB, char *format, ...)
 *
 * PURPOSE:         This function forms a high level
printf for an HTML browser
 *
 * ARGUMENTS:       EXTENSION_CONTROL_BLOCK       *pECB
          passed in structure pointer from inetsrv.
 *                                  char
                    *format
          printf style format string
 *                                  ...

                    other arguments as required by
printf style format string.
 *
 * RETURNS:                  None
 *
 * COMMENTS:        This function is mainly used for
developmental support.
 */

static void h_printf(EXTENSION_CONTROL_BLOCK *pECB,
char *format, ...)
{
//        int lpbSize;
          char szBuff[512];
          char szTmp[512];

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

//        lpbSize = wsprintf(szBuff, "<html>%s</html>",
szTmp) + 1;
//
//        (*pECB->WriteClient)(pECB->ConnID, szBuff,
&lpbSize, 0);

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

          WriteZString(pECB, szBuff);

          return;
}
```

```
void LogTuxError( int TpErrno, char *ErrMessage )
{
          FILE *fp;
          SYSTEMTIME          systemTime;

          GetLocalTime(&systemTime);

          fp = fopen(szErrorLogPath, "ab");
          fprintf(fp, "\r\nError: %2.2d/%2.2d/%2.2d
%2.2d:%2.2d:%2.2d\r\n",
                    systemTime.wYear,
systemTime.wMonth, systemTime.wDay,
                    systemTime.wHour,
systemTime.wMinute, systemTime.wSecond);
          fprintf(fp, "Thread %d: TPCCWEB(%d): %s: %s",
                    GetCurrentThreadId(), TpErrno,
tpstrerror(TpErrno), ErrMessage);
          fclose(fp);
}

/* FUNCTION: void ErrorMessage(EXTENSION_CONTROL_BLOCK
*pECB, int iError, int iErrorType, char *szMsg)
 *
 * PURPOSE:          This function displays an error
message in the client browser.
 *
 * ARGUMENTS:        EXTENSION_CONTROL_BLOCK       *pECB
          passed in structure pointer from inetsrv.
 *                                   int
                    iError
          id of error message
 *                                   int
                    iErrorType
          error type, ERR_TYPE_SQL, ERR_TYPE_DBLIB, or
ERR_TYPE_WEBDLL
 *                                   int
                    iTermId
          terminal id from browser
 *                                   int
                    iSyncid
          sync id from browser
 *                                   char *
                    szMsg
          optional error message string used with
ERR_TYPE_SQL and
 *
                    ERR_TYPE_DBLIB
 *
 * RETURNS:                   None
 *
 * COMMENTS:         If the error type is
ERR_TYPE_WEBDLL the szmsg parameter may be NULL because
it
 *                                   is ignored. If
the error type is ERR_TYPE_SQL or ERR_TYPE_DBLIB then
the szMsg
 *                                   parameter
contains the text of the error message, so the szMsg
parameter cannot
 *                                   be NULL.
 *
 */


void ErrorMessage(EXTENSION_CONTROL_BLOCK *pECB, int
iError, int iErrorType, char *szMsg, int iTermId, int
iSyncId)
{
          int i;
          SYSTEMTIME          systemTime;

          static SERRORMSG errorMsgs[] =
          {
                    {         ERR_SUCCESS,

          "Success, no error."

                                                  },
                    {         ERR_COMMAND_UNDEFINED,

          "Command undefined."

                                                  },
                    {         ERR_NOT_IMPLEMENTED_YET,
                                        "Not
Implemented Yet."

                                        },
                    {         ERR_CANNOT_INIT_TERMINAL,
                                        "Cannot
initialize 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;

        GetLocalTime(&systemTime);

        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=\"ERROR\" VALUE=\"%d\">", iError);

        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=\"ERROR\" VALUE=\"%d\">", iError);

        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=\"ERROR\" VALUE=\"%d\">", iError);

        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=\"ERROR\" VALUE=\"%d\">", iError);

        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");
                        strcat(szForm,
"</FORM><BODY></HTML>");
                        WriteZString(pECB,
szForm);
                        break;
                case ERR_TYPE_SOCKET:
                        strcpy(szForm,
        "<HTML><HEAD><TITLE>Welcome To TPC-
C</TITLE></HEAD><BODY><FORM ACTION=\"tpcc.dll\"
METHOD=\"GET\">");
```



```
          wsprintf(szForm+strlen(szForm), "<INPUT
TYPE=\"hidden\" NAME=\"STATUSID\" VALUE=\"%d\">",
iErrorType);

          wsprintf(szForm+strlen(szForm), "<INPUT
TYPE=\"hidden\" NAME=\"ERROR\" VALUE=\"%d\">", iError);

          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:
SOCKET");
                              strcat(szForm,
"</FORM><BODY></HTML>");
                              WriteZString(pECB,
szForm);
                              break;
                    case ERR_TYPE_DEADLOCK:
                              strcpy(szForm,
          "<HTML><HEAD><TITLE>Welcome To TPC-
C</TITLE></HEAD><BODY><FORM ACTION=\"tpcc.dll\"
METHOD=\"GET\">");

          wsprintf(szForm+strlen(szForm), "<INPUT
TYPE=\"hidden\" NAME=\"STATUSID\" VALUE=\"%d\">",
iErrorType);

          wsprintf(szForm+strlen(szForm), "<INPUT
TYPE=\"hidden\" NAME=\"ERROR\" VALUE=\"%d\">", iError);

          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:
Deadlock");
                              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: 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;

#ifndef LOCAL_ALLOC
                    tpfree((char
*)Term.pClientData[id].TuxDataPtr);
#endif  // Not LOCAL_ALLOC

          }

          return;
}

/* FUNCTION: BOOL Init(EXTENSION_CONTROL_BLOCK *pECB,
int iTermId, int iSyncId, char *szServer, char *szUser,
char *szPassword, char *szDatabase)
 *
 * PURPOSE:         This function initializes the sql
connection for use.
 *
 * ARGUMENTS:       EXTENSION_CONTROL_BLOCK       *pECB
                    passed in structure pointer from
inetsrv.
 *                                     int

          iTermId             id of browser client that
this connection is for.
 *                                     int

          iSyncId             sync id for this client
session
```

```
 *                                      char
                                        *szServer sql
server name
 *                                      char
                                        *szUser
          user name
 *                                      char
                                        *szPassword
          user password
 *                                      char
                                        *szDatabase
          database to use
 *
 * RETURNS:                    BOOL      FALSE     if
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];
#ifndef LOCAL_ALLOC
          char      buf[64];
          int                 TpRc;
#endif // Not LOCAL_ALLOC

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

          Term.pClientData[iTermId].dbproc = NULL;

#ifndef LOCAL_ALLOC // Globally allocate tuxedo
structures

                    if ( dLog )
                    {
                              FILE *fp;
                              fp = fopen(szTpccLogPath,
"ab");
                              fprintf(fp,
"Sizeof(TUX_DATA) %d \r\n", sizeof(TUX_DATA));
                              fprintf(fp,
"Sizeof(NEW_ORDER_DATA) %d \r\n",
sizeof(NEW_ORDER_DATA));
                              fprintf(fp,
"Sizeof(PAYMENT_DATA) %d \r\n", sizeof(PAYMENT_DATA));
                              fprintf(fp,
"Sizeof(ORDER_STATUS_DATA) %d \r\n",
sizeof(ORDER_STATUS_DATA));
                              fprintf(fp,
"Sizeof(DELIVERY_DATA) %d \r\n",
sizeof(DELIVERY_DATA));
                              fprintf(fp,
"Sizeof(STOCK_LEVEL_DATA) %d \r\n",
sizeof(STOCK_LEVEL_DATA));
                              fclose(fp);
                    }

//        Add initialization of Tuxedo Structures

                    if
((Term.pClientData[iTermId].TuxDataPtr = (TUX_DATA
*)tpalloc("CARRAY", NULL, sizeof(TUX_DATA))) == NULL)
                    {
                              TpRc = tperrno;
                              sprintf(buf, "Tuxedo
tpalloc failed:");
                              LogTuxError(TpRc, buf);
                              ErrorMessage(pECB,
ERR_CANNOT_INIT_TERMINAL, ERR_TYPE_WEBDLL, NULL,
iTermId, iSyncId);
                              return TRUE;
                    }

                    if ( dLog )
                    {
                              FILE *fp;

                              fp = fopen(szTpccLogPath,
"ab");

                              fprintf(fp, "Thread %d
iTermId %d *  TuxDataPtr: %x \r\n",

          GetCurrentThreadId(), iTermId,
&Term.pClientData[iTermId].TuxDataPtr);
                              fclose(fp);
                    }
#endif // LOCAL_ALLOC

          return FALSE;
}

/* FUNCTION: BOOL Close(EXTENSION_CONTROL_BLOCK
          *pECB, int iTermId, 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].dbpro
c)) )
                    {
                              pEcbInfo->iTermId = -1;
                              pEcbInfo->iSyncId = -1;
                              free(pEcbInfo);
          //free up user info
                    }
//                  return SQLCloseConnection(pECB,
Term.pClientData[iTermId].dbproc);
                    return 1;
          }

          UNUSEDPARAM(iSyncId);
}


/* FUNCTION: void FormatString(char *szDest, char
*szPic, char *szSrc)
 *
 * PURPOSE:          This function formats a character
string for inclusion in the
 *                              HTML formatted page being
constructed.
 *
 * ARGUMENTS:        char                *szDest
          Destination buffer where formatted string is
to be placed
 *                                      char
          *szPic    picture string which describes how
character value is to be
 *
                                        formatted.
 *                                      char
          *szSrc    character string value.
 *
 * RETURNS:                    None
 *
 * COMMENTS:        This functions is used to format
TPC-C phone and zip value strings.
 *
 */

static void FormatString(char *szDest, char *szPic,
char *szSrc)
{
          while( *szPic )
          {
                    if ( *szPic == 'X' )
                    {
                              if ( *szSrc )
```

```
                                        *szDest++ =
*szSrc++;
                                else
                                        *szDest++ = '
';
                        }
                        else
                                        *szDest++ = *szPic;
                        szPic++;
                }
                *szDest = 0;

                return;
}

/* FUNCTION: char *MakeStockLevelForm(int iTermId, int
iSyncId, BOOL bInput)
 *
 * PURPOSE:             This function constructs the Stock
Level HTML page.
 *
 * ARGUMENTS:           int
                iTermId   client browser terminal id
 *                                      int
                                iSyncId   client browser
sync id
 *                                      BOOL
                        bInput    TRUE if form is being
constructed for input else FALSE
 *
 * RETURNS:                     char *
                A pointer to buffer inside client structure
where HTML form is built.
 *
 * COMMENTS:            The internal client buffer is
created when the terminal id is assigned and should not
 *                                      be freed except
when the client terminal id is no longer needed.
 */

static char *MakeStockLevelForm(int iTermId, int
iSyncId, BOOL bInput)
{
          char      *szForm;

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

          Term.pClientData[iTermId].StockLevelData.w_id
                                =
(short)Term.pClientData[iTermId].w_id;
          Term.pClientData[iTermId].StockLevelData.d_id
                                =
(short)Term.pClientData[iTermId].d_id;
          Term.pClientData[iTermId].StockLevelData.num_
deadlocks = 0;

          strcpy(szForm, "<HTML><HEAD><TITLE>TPC-C
Stock Level</TITLE></HEAD>");
          strcat(szForm, "<FORM ACTION=\"tpcc.dll\"
METHOD=\"GET\">");
          if ( bInput )
                    strcat(szForm, "<INPUT
TYPE=\"hidden\" NAME=\"PI*\" VALUE=\"\">");
          strcat(szForm, "<INPUT TYPE=\"hidden\"
NAME=\"STATUSID\" VALUE=\"0\">");
          strcat(szForm, "<INPUT TYPE=\"hidden\"
NAME=\"ERROR\" VALUE=\"0\">");
          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\">");
          strcat(szForm, "<INPUT TYPE=\"hidden\"
NAME=\"ERROR\" 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, int
SyncId, BOOL Rollback, BOOL bInput, BOOL bValid)
 *
 * PURPOSE:         This function
 *
 * ARGUMENTS:       int
          iTermId   client browser terminal id
 *                                    int
                            iSyncId   client browser
sync id
 *                                    BOOL
                    bInput    TRUE if form is being
constructed for input else FALSE
 *                                    BOOL
                    bValid    TRUE if NewOrderData
valid, ELSE FALSE effects output only
 *
 * RETURNS:                  char *
          A pointer to buffer inside client structure
where HTML form is built.
 *
 * COMMENTS:        The internal client buffer is
created when the terminal id is assigned and should not
 *                                    be freed except
when the client terminal id is no longer needed.
 */

static char *MakeNewOrderForm(int iTermId, int iSyncId,
BOOL Rollback, BOOL bInput, BOOL bValid)
{
          char      *szForm;
          char      szName[146];
          char      szCredit[14];
          int                 i;

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

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

          strcpy(szForm,      "<HTML>"

          "<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);
          }

          if (Rollback == FALSE)
                    strcat(szForm, "<INPUT
TYPE=\"hidden\" NAME=\"ERROR\" VALUE=\"0\">");
          else
                    strcat(szForm, "<INPUT
TYPE=\"hidden\" NAME=\"ERROR\" VALUE=\"1\">");

          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_entr
y_d.day,

          Term.pClientData[iTermId].NewOrderData.o_entr
y_d.month,

          Term.pClientData[iTermId].NewOrderData.o_entr
y_d.year,

          Term.pClientData[iTermId].NewOrderData.o_entr
y_d.hour,

          Term.pClientData[iTermId].NewOrderData.o_entr
y_d.minute,

          Term.pClientData[iTermId].NewOrderData.o_entr
y_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 *
100.0);

          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_c
nt,

          Term.pClientData[iTermId].NewOrderData.w_tax
* 100.0,

          Term.pClientData[iTermId].NewOrderData.d_tax
* 100.0);

                              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.execut
ion_status,

          Term.pClientData[iTermId].NewOrderData.total_
amount);
                    }
                    else
                    {

          sprintf(szForm+strlen(szForm), "Execution
Status: %24.24s                  Total:",

          Term.pClientData[iTermId].NewOrderData.execut
ion_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\">");
          strcat(szForm, "<INPUT TYPE=\"hidden\"
NAME=\"ERROR\" 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 *
100.0);

                    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_amoun
t,

          Term.pClientData[iTermId].PaymentData.c_balan
ce);

                    sprintf(szForm+strlen(szForm),
"Credit Limit:   $%13.2f<BR><BR>",

          Term.pClientData[iTermId].PaymentData.c_credi
t_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_i
d = 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\">");
          strcat(szForm, "<INPUT TYPE=\"hidden\"
NAME=\"ERROR\" 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_i
d, c_first, c_middle, c_last);

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

          Term.pClientData[iTermId].OrderStatusData.c_b
alance);

                    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_i
d,

          Term.pClientData[iTermId].OrderStatusData.o_e
ntry_d.day,

          Term.pClientData[iTermId].OrderStatusData.o_e
ntry_d.month,

          Term.pClientData[iTermId].OrderStatusData.o_e
ntry_d.year,

          Term.pClientData[iTermId].OrderStatusData.o_e
ntry_d.hour,

          Term.pClientData[iTermId].OrderStatusData.o_e
ntry_d.minute,

          Term.pClientData[iTermId].OrderStatusData.o_e
ntry_d.second,

          Term.pClientData[iTermId].OrderStatusData.o_c
arrier_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.OlO
rderStatusData[i].ol_supply_w_id,

          Term.pClientData[iTermId].OrderStatusData.OlO
rderStatusData[i].ol_i_id,

          Term.pClientData[iTermId].OrderStatusData.OlO
rderStatusData[i].ol_quantity,

          Term.pClientData[iTermId].OrderStatusData.OlO
rderStatusData[i].ol_amount,

          Term.pClientData[iTermId].OrderStatusData.OlO
rderStatusData[i].ol_delivery_d.day,

          Term.pClientData[iTermId].OrderStatusData.OlO
rderStatusData[i].ol_delivery_d.month,

          Term.pClientData[iTermId].OrderStatusData.OlO
rderStatusData[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);
                                  strcat(szForm, "<INPUT
TYPE=\"hidden\" NAME=\"ERROR\" VALUE=\"2\">");
                  }
                  else
                  {
                                  strcat(szForm, "<INPUT
TYPE=\"hidden\" NAME=\"STATUSID\" VALUE=\"0\">");
                                  strcat(szForm, "<INPUT
TYPE=\"hidden\" NAME=\"ERROR\" VALUE=\"0\">");
                  }
        }

        wsprintf(szForm+strlen(szForm), "<INPUT
TYPE=\"hidden\" NAME=\"FORMID\" VALUE=\"%d\">",
DELIVERY_FORM);
        wsprintf(szForm+strlen(szForm), "<INPUT
TYPE=\"hidden\" NAME=\"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_carr
ier_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.execut
ion_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
 *
 */

#ifdef LOCAL_ALLOC // Allocate the tpalloc structure
for each transaction
//        This saves on some memory at the expense of
some CPU cycles.
static void ProcessNewOrderForm(EXTENSION_CONTROL_BLOCK
          *pECB, int iTermId, int iSyncId)
{
           int      TpRc, iRc, iError;
           long     ilen, *olen;
           char     buf[128];

           NEW_ORDER_DATA
          *NewOrderDataPtr;   //New Order Tuxedo Buffer

           if((iRc = ThrTpInit()) <0)
           {
                     // This is bad
                    sprintf(buf, "ProcessNewOrder
Failed ThrTpInit: iRc = %d", iRc);
                    LogTuxError(0, buf);
                    ErrorMessage(pECB,
ERR_NEW_ORDER_NOT_PROCESSED, ERR_TYPE_WEBDLL, NULL,
iTermId, iSyncId);
                    return;
           }

          memset(&Term.pClientData[iTermId].NewOrderDat
a, 0, sizeof(NEW_ORDER_DATA));

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

          if ( (iError=GetNewOrderData(pECB-
>lpszQueryString,
&Term.pClientData[iTermId].NewOrderData)) !=
ERR_SUCCESS )
          {
                    ErrorMessage(pECB, iError,
ERR_TYPE_WEBDLL, NULL, iTermId, iSyncId);
                    return;
          }

          if ((NewOrderDataPtr = (NEW_ORDER_DATA
*)tpalloc("CARRAY", NULL, sizeof(NEW_ORDER_DATA))) ==
NULL)
          {
                     TpRc = tperrno;
                     sprintf(buf, "ProcessNewOrder
Tpcalloc Failed");
                    LogTuxError(TpRc, buf);
                    ErrorMessage(pECB,
ERR_NEW_ORDER_NOT_PROCESSED, ERR_TYPE_WEBDLL, NULL,
iTermId, iSyncId);
                     return;
          }

           *NewOrderDataPtr =
Term.pClientData[iTermId].NewOrderData;

     ilen = sizeof(NEW_ORDER_DATA);
     olen = &ilen;

           if ( dLog )
           {
                    FILE *fp;

                    fp = fopen(szTpccLogPath, "ab");
                    fprintf(fp, "* ProcessNewOrderL
Thread %d iTermId %d NewOrderDataPtr: %x size %d \r\n",
                              GetCurrentThreadId(),
iTermId, &NewOrderDataPtr, sizeof(*NewOrderDataPtr));
                    fclose(fp);
           }

     if ((iRc = tpcall("NEWORDER", (char
*)NewOrderDataPtr, ilen,
                     (char **)&NewOrderDataPtr, (long
*)olen, TPSIGRSTRT)) == -1)
           {
          TpRc = tperrno;
                     sprintf(buf, "Neworder tpcall
failed");
                     LogTuxError(TpRc, buf);
                     ErrorMessage(pECB,
ERR_NEW_ORDER_NOT_PROCESSED, ERR_TYPE_WEBDLL, NULL,
iTermId, iSyncId);
          return;
     }

           if ( dLog )
           {
                    FILE *fp;

                    fp = fopen(szTpccLogPath, "ab");
                    fprintf(fp, "** ProcessNewOrderL
Thread %d iTermId %d NewOrderDataPtr: %x size %d\r\n",
                              GetCurrentThreadId(),
iTermId, &NewOrderDataPtr, sizeof(*NewOrderDataPtr));
                    fclose(fp);
           }

           Term.pClientData[iTermId].NewOrderData =
*NewOrderDataPtr;

     iRc = NewOrderDataPtr->retval;
           iError = NewOrderDataPtr->error;

           tpfree((char *)NewOrderDataPtr);

          if ( iRc < 0 )
          {
                    if (iError == ERR_TYPE_DEADLOCK)
                              ErrorMessage(pECB,
ERR_NEW_ORDER_NOT_PROCESSED, ERR_TYPE_DEADLOCK, NULL,
iTermId, iSyncId);
                    else
                              ErrorMessage(pECB,
ERR_NEW_ORDER_NOT_PROCESSED, ERR_TYPE_WEBDLL, NULL,
iTermId, iSyncId);
          }
          else
                    if ( iError == ERR_BAD_ITEM_ID)
                              WriteZString(pECB,
MakeNewOrderForm(iTermId, iSyncId, TRUE, FALSE,
(BOOL)iRc) );
                    else
                              WriteZString(pECB,
MakeNewOrderForm(iTermId, iSyncId, FALSE, FALSE,
(BOOL)iRc) );

          return;
}
#else // Not LOCAL_ALLOC
static void ProcessNewOrderForm(EXTENSION_CONTROL_BLOCK
          *pECB, int iTermId, int iSyncId)
{
          int                 TpRc, iRc, iError;
          long      ilen, *olen;
          char      buf[128];

     if((iRc = ThrTpInit()) <0)
           {
                     // This is bad
                    sprintf(buf, "ProcessNewOrder
Failed ThrTpInit: iRc = %d", iRc);
                    LogTuxError(0, buf);
                    ErrorMessage(pECB,
ERR_NEW_ORDER_NOT_PROCESSED, ERR_TYPE_WEBDLL, NULL,
iTermId, iSyncId);
                    return;
           }

          memset(&Term.pClientData[iTermId].NewOrderDat
a, 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;
        }

        Term.pClientData[iTermId].TuxDataPtr-
>NewOrderData = Term.pClientData[iTermId].NewOrderData;

    ilen = sizeof(TUX_DATA);
    olen = &ilen;

         if ( dLog )
         {
                FILE *fp;

                fp = fopen(szTpccLogPath, "ab");
                fprintf(fp, "* ProcessNewOrder
Thread %d iTermId %d TuxDataPtr: %x size %d \r\n",
                        GetCurrentThreadId(),
iTermId, &Term.pClientData[iTermId].TuxDataPtr,

        sizeof(*Term.pClientData[iTermId].TuxDataPtr)
);
                fclose(fp);
         }

    if ((iRc = tpcall("NEWORDER", (char
*)Term.pClientData[iTermId].TuxDataPtr, ilen,
                 (char
**)&Term.pClientData[iTermId].TuxDataPtr, (long *)olen,
TPSIGRSTRT)) == -1)
         {
        TpRc = tperrno;
                 sprintf(buf, "Neworder tpcall
failed");
                 LogTuxError(TpRc, buf);
                 ErrorMessage(pECB,
ERR_NEW_ORDER_NOT_PROCESSED, ERR_TYPE_WEBDLL, NULL,
iTermId, iSyncId);
        return;
    }

         if ( dLog )
         {
                FILE *fp;

                fp = fopen(szTpccLogPath, "ab");
                fprintf(fp, "** ProcessNewOrder
Thread %d iTermId %d TuxDataPtr: %x size %d\r\n",
                        GetCurrentThreadId(),
iTermId, &Term.pClientData[iTermId].TuxDataPtr,

        sizeof(*Term.pClientData[iTermId].TuxDataPtr)
);
                fclose(fp);
         }

    Term.pClientData[iTermId].NewOrderData =
Term.pClientData[iTermId].TuxDataPtr->NewOrderData;

    iRc = Term.pClientData[iTermId].TuxDataPtr-
>NewOrderData.retval;
         iError =
Term.pClientData[iTermId].TuxDataPtr-
>NewOrderData.error;

        if ( iRc < 0 )
        {
                if (iError == ERR_TYPE_DEADLOCK)
                        ErrorMessage(pECB,
ERR_NEW_ORDER_NOT_PROCESSED, ERR_TYPE_DEADLOCK, NULL,
iTermId, iSyncId);
                else
                        ErrorMessage(pECB,
ERR_NEW_ORDER_NOT_PROCESSED, ERR_TYPE_WEBDLL, NULL,
iTermId, iSyncId);
        }
        else
                if ( iError == ERR_BAD_ITEM_ID)
                        WriteZString(pECB,
MakeNewOrderForm(iTermId, iSyncId, TRUE, FALSE,
(BOOL)iRc) );
                else
                        WriteZString(pECB,
MakeNewOrderForm(iTermId, iSyncId, FALSE, FALSE,
(BOOL)iRc) );

        return;
}
#endif // LOCAL_ALLOC

/* FUNCTION: void
ProcessPaymentForm(EXTENSION_CONTROL_BLOCK *pECB, int
iTermId, int iSyncId)
 *
 * PURPOSE:         This function gets and validates
the input data from the payment form
 *                          filling in the required
input variables. It then calls the SQLPayment
 *                          transaction, constructs
the output form and writes it back to client
 *                          browser.
 *
 * ARGUMENTS:       EXTENSION_CONTROL_BLOCK       *pECB
        passed in structure pointer from inetsrv.
 *                                  int

        iTermId   client browser terminal id
 *                                  int

        iSyncId client browser sync id
 *
 * RETURNS:                 None
 *
 * COMMENTS:        None
 *
 */

#ifdef LOCAL_ALLOC // Allocate the tpalloc structure
for each transaction
//        This saves on some memory at the expense of
some CPU cycles.
static void ProcessPaymentForm(EXTENSION_CONTROL_BLOCK
*pECB, int iTermId, int iSyncId)
{
          int               TpRc, iRc, iError;
          long     ilen, *olen;
          char     buf[128];

          PAYMENT_DATA
         *PaymentDataPtr;    //Payment Tuxedo Buffer

    if((iRc = ThrTpInit()) <0)
          {
                   // This is bad
                  sprintf(buf, "ProcessPayment
Failed ThrTpInit: iRc = %d", iRc);
                  LogTuxError(0, buf);
                  ErrorMessage(pECB,
ERR_PAYMENT_NOT_PROCESSED, ERR_TYPE_WEBDLL, NULL,
iTermId, iSyncId);
                  return;
          }


memset(&Term.pClientData[iTermId].PaymentData, 0,
sizeof(PAYMENT_DATA));

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

    if ( (iError=GetPaymentData(pECB->lpszQueryString,
&Term.pClientData[iTermId].PaymentData)) != ERR_SUCCESS
)
    {
         ErrorMessage(pECB, iError, ERR_TYPE_WEBDLL,
NULL, iTermId, iSyncId);
         return;
    }

          if ((PaymentDataPtr = (PAYMENT_DATA
*)tpalloc("CARRAY", NULL, sizeof(PAYMENT_DATA))) ==
NULL)
         {
                  TpRc = tperrno;
                  sprintf(buf, "ProcessPayment
Tpcalloc Failed");
                  LogTuxError(TpRc, buf);
                  ErrorMessage(pECB,
ERR_PAYMENT_NOT_PROCESSED, ERR_TYPE_WEBDLL, NULL,
iTermId, iSyncId);
                  return;
         }

    *PaymentDataPtr =
Term.pClientData[iTermId].PaymentData;

          ilen = sizeof(PAYMENT_DATA);
    olen = &ilen;

          if ( dLog )
          {
                  FILE *fp;

                  fp = fopen(szTpccLogPath, "ab");
                  fprintf(fp, "* ProcessPayment
Thread %d iTermId %d PaymentDataPtr: %x \r\n",
```

```
                                GetCurrentThreadId(),
iTermId, &PaymentDataPtr);
                    fclose(fp);
          }

     if (( iRc = tpcall("PAYMENT", (char
*)PaymentDataPtr, ilen,
                     (char **)&PaymentDataPtr, (long
*)olen, TPSIGRSTRT)) == -1)
          {
         TpRc = tperrno;
                     sprintf(buf, "ProcessPayment
tpcall failed");
                     LogTuxError(TpRc, buf);
                     ErrorMessage(pECB,
ERR_PAYMENT_NOT_PROCESSED, ERR_TYPE_WEBDLL, NULL,
iTermId, iSyncId);
          return;
     }

          if ( dLog )
          {
                   FILE *fp;

                   fp = fopen(szTpccLogPath, "ab");
                   fprintf(fp, "** ProcessPayment
Thread %d iTermId %d PaymentDataPtr: %x \r\n",
                                GetCurrentThreadId(),
iTermId, &PaymentDataPtr);
                   fclose(fp);
          }

     Term.pClientData[iTermId].PaymentData =
*PaymentDataPtr;

     iRc = PaymentDataPtr->retval;
          iError = PaymentDataPtr->error;

          tpfree((char *)PaymentDataPtr);

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

     return;

}
#else // Not LOCAL_ALLOC
static void ProcessPaymentForm(EXTENSION_CONTROL_BLOCK
*pECB, int iTermId, int iSyncId)
{
          int                 TpRc, iRc, iError;
          long      ilen, *olen;
          char      buf[128];

     if((iRc = ThrTpInit()) <0)
          {
                    // This is bad
                    sprintf(buf, "ProcessPayment
Failed ThrTpInit: iRc = %d", iRc);
                    LogTuxError(0, buf);
                    ErrorMessage(pECB,
ERR_PAYMENT_NOT_PROCESSED, ERR_TYPE_WEBDLL, NULL,
iTermId, iSyncId);
                    return;
          }


memset(&Term.pClientData[iTermId].PaymentData, 0,
sizeof(PAYMENT_DATA));

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

     if ( (iError=GetPaymentData(pECB->lpszQueryString,
&Term.pClientData[iTermId].PaymentData)) != ERR_SUCCESS
)
     {
          ErrorMessage(pECB, iError, ERR_TYPE_WEBDLL,
NULL, iTermId, iSyncId);
          return;
     }

     Term.pClientData[iTermId].TuxDataPtr->PaymentData
= Term.pClientData[iTermId].PaymentData;

          ilen = sizeof(TUX_DATA);
     olen = &ilen;

          if ( dLog )
          {
                   FILE *fp;

                   fp = fopen(szTpccLogPath, "ab");
                   fprintf(fp, "* ProcessPayment
Thread %d iTermId %d PaymentDataPtr: %x \r\n",
                                GetCurrentThreadId(),
iTermId, &Term.pClientData[iTermId].TuxDataPtr);
                   fclose(fp);
          }

     if (( iRc = tpcall("PAYMENT", (char
*)Term.pClientData[iTermId].TuxDataPtr, ilen,
                     (char
**)&Term.pClientData[iTermId].TuxDataPtr, (long *)olen,
TPSIGRSTRT)) == -1)
          {
         TpRc = tperrno;
                     sprintf(buf, "ProcessPayment
tpcall failed");
                     LogTuxError(TpRc, buf);
                     ErrorMessage(pECB,
ERR_PAYMENT_NOT_PROCESSED, ERR_TYPE_WEBDLL, NULL,
iTermId, iSyncId);
          return;
     }

          if ( dLog )
          {
                   FILE *fp;

                   fp = fopen(szTpccLogPath, "ab");
                   fprintf(fp, "** ProcessPayment
Thread %d iTermId %d TuxDataPtr: %x \r\n",
                                GetCurrentThreadId(),
iTermId, &Term.pClientData[iTermId].TuxDataPtr);
                   fclose(fp);
          }

     Term.pClientData[iTermId].PaymentData =
Term.pClientData[iTermId].TuxDataPtr->PaymentData;

     iRc = Term.pClientData[iTermId].TuxDataPtr-
>PaymentData.retval;
          iError =
Term.pClientData[iTermId].TuxDataPtr-
>PaymentData.error;

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

     return;
}
#endif // LOCAL_ALLOC

/* FUNCTION: void
ProcessOrderStatusForm(EXTENSION_CONTROL_BLOCK *pECB,
int iTermId, int iSyncId)
 *
 * PURPOSE:          This function gets and validates
the input data from the Order Status
 *                              form filling in the
required input variables. It then calls the
```

```
 *                           SQLOrderStatus
transaction, constructs the output form and writes it
 *                           back to client browser.
 *
 * ARGUMENTS:       EXTENSION_CONTROL_BLOCK       *pECB
         passed in structure pointer from inetsrv.
 *                                    int

         iTermId   client browser terminal id
 *                                    int

         iSyncId client browser sync id
 *
 * RETURNS:                  None
 *
 * COMMENTS:        None
 *
 */

#ifdef LOCAL_ALLOC // Allocate the tpalloc structure
for each transaction
//       This saves on some memory at the expense of
some CPU cycles.
static void
ProcessOrderStatusForm(EXTENSION_CONTROL_BLOCK *pECB,
int iTermId, int iSyncId)
{
          int                TpRc, iRc, iError;
          long      ilen, *olen;
          char      buf[128];

          ORDER_STATUS_DATA   *OrderStatusDataPtr;
          //Order Status Tuxedo Buffer

     if((iRc = ThrTpInit()) <0)
           {
                     // This is bad
                    sprintf(buf, "ProcessOrderStatus
Failed ThrTpInit: iRc = %d", iRc);
                    LogTuxError(0, buf);
                    ErrorMessage(pECB,
ERR_ORDER_STATUS_NOT_PROCESSED, ERR_TYPE_WEBDLL, NULL,
iTermId, iSyncId);
                    return;
           }

     memset(&Term.pClientData[iTermId].OrderStatusData,
0, sizeof(ORDER_STATUS_DATA));

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

     if ( (iError=GetOrderStatusData(pECB-
>lpszQueryString,
&Term.pClientData[iTermId].OrderStatusData)) !=
ERR_SUCCESS )
     {
          ErrorMessage(pECB, iError, ERR_TYPE_WEBDLL,
NULL, iTermId, iSyncId);
          return;
     }

          if ((OrderStatusDataPtr = (ORDER_STATUS_DATA
*)tpalloc("CARRAY", NULL, sizeof(ORDER_STATUS_DATA)))
== NULL)
          {
                    TpRc = tperrno;
                    sprintf(buf, "ProcessOrderStatus
Tpcalloc Failed");
                    LogTuxError(TpRc, buf);
                    ErrorMessage(pECB,
ERR_ORDER_STATUS_NOT_PROCESSED, ERR_TYPE_WEBDLL, NULL,
iTermId, iSyncId);
                    return;
          }

     *OrderStatusDataPtr =
Term.pClientData[iTermId].OrderStatusData;

           ilen = sizeof(ORDER_STATUS_DATA);
     olen = &ilen;

           if ( dLog )
           {
                    FILE *fp;

                    fp = fopen(szTpccLogPath, "ab");
                    fprintf(fp, "* ProcessOrderStatus
Thread %d iTermId %d OrderStatusDataPtr: %x \r\n",
                              GetCurrentThreadId(),
iTermId, &OrderStatusDataPtr);
                    fclose(fp);
           }

     if (( iRc = tpcall("ORDERSTATUS", (char
*)OrderStatusDataPtr, ilen,
                    (char **)&OrderStatusDataPtr,
(long *)olen, TPSIGRSTRT)) == -1)
           {
          TpRc = tperrno;
                    sprintf(buf, "ProcessOrderStatus
tpcall failed");
                    LogTuxError(TpRc, buf);
                    ErrorMessage(pECB,
ERR_ORDER_STATUS_NOT_PROCESSED, ERR_TYPE_WEBDLL, NULL,
iTermId, iSyncId);
          return;
     }

           if ( dLog )
           {
                    FILE *fp;

                    fp = fopen(szTpccLogPath, "ab");
                    fprintf(fp, "** ProcessOrderStatus
Thread %d iTermId %d OrderStatusDataPtr: %x \r\n",
                              GetCurrentThreadId(),
iTermId, &OrderStatusDataPtr);
                    fclose(fp);
           }

     Term.pClientData[iTermId].OrderStatusData =
*OrderStatusDataPtr;

     iRc = OrderStatusDataPtr->retval;
           iError = OrderStatusDataPtr->error;

           tpfree((char *)OrderStatusDataPtr);

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

     return;
}
#else // Not LOCAL_ALLOC
static void
ProcessOrderStatusForm(EXTENSION_CONTROL_BLOCK *pECB,
int iTermId, int iSyncId)
{
          int                TpRc, iRc, iError;
          long      ilen, *olen;
          char      buf[128];

     if((iRc = ThrTpInit()) <0)
           {
                     // This is bad
                    sprintf(buf, "ProcessOrderStatus
Failed ThrTpInit: iRc = %d", iRc);
                    LogTuxError(0, buf);
                    ErrorMessage(pECB,
ERR_ORDER_STATUS_NOT_PROCESSED, ERR_TYPE_WEBDLL, NULL,
iTermId, iSyncId);
                    return;
           }

     memset(&Term.pClientData[iTermId].OrderStatusData,
0, sizeof(ORDER_STATUS_DATA));

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

     if ( (iError=GetOrderStatusData(pECB-
>lpszQueryString,
&Term.pClientData[iTermId].OrderStatusData)) !=
ERR_SUCCESS )
     {
          ErrorMessage(pECB, iError, ERR_TYPE_WEBDLL,
NULL, iTermId, iSyncId);
          return;
```

```
     }

     Term.pClientData[iTermId].TuxDataPtr-
>OrderStatusData =
Term.pClientData[iTermId].OrderStatusData;

           ilen = sizeof(TUX_DATA);
     olen = &ilen;

           if ( dLog )
           {
                    FILE *fp;

                    fp = fopen(szTpccLogPath, "ab");
                    fprintf(fp, "* ProcessOrderStatus
Thread %d iTermId %d TuxDataPtr: %x \r\n",
                              GetCurrentThreadId(),
iTermId, &Term.pClientData[iTermId].TuxDataPtr);
                    fclose(fp);
           }

     if (( iRc = tpcall("ORDERSTATUS", (char
*)Term.pClientData[iTermId].TuxDataPtr, ilen,
                     (char
**)&Term.pClientData[iTermId].TuxDataPtr, (long *)olen,
TPSIGRSTRT)) == -1)
           {
          TpRc = tperrno;
                     sprintf(buf, "ProcessOrderStatus
tpcall failed");
                     LogTuxError(TpRc, buf);
                     ErrorMessage(pECB,
ERR_ORDER_STATUS_NOT_PROCESSED, ERR_TYPE_WEBDLL, NULL,
iTermId, iSyncId);
          return;
     }

           if ( dLog )
           {
                    FILE *fp;

                    fp = fopen(szTpccLogPath, "ab");
                    fprintf(fp, "** ProcessOrderStatus
Thread %d iTermId %d TuxDataPtr: %x \r\n",
                              GetCurrentThreadId(),
iTermId, &Term.pClientData[iTermId].TuxDataPtr);
                    fclose(fp);
           }

     Term.pClientData[iTermId].OrderStatusData =
Term.pClientData[iTermId].TuxDataPtr->OrderStatusData;

     iRc = Term.pClientData[iTermId].TuxDataPtr-
>OrderStatusData.retval;
           iError =
Term.pClientData[iTermId].TuxDataPtr-
>OrderStatusData.error;

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

     return;

}
#endif // LOCAL_ALLOC

/* FUNCTION: void
ProcessDeliveryForm(EXTENSION_CONTROL_BLOCK *pECB, int
iTermId, int iSyncId)
 *
 * PURPOSE:         This function gets and validates
the input data from the delivery form
 *                           filling in the required
input variables. It then calls the PostDeliveryInfo
 *                           Api, The client is then
informed that the transaction has been posted.
 *
 * ARGUMENTS:       EXTENSION_CONTROL_BLOCK       *pECB
        passed in structure pointer from inetsrv.
 *                                    int

        iTermId   client browser terminal id
 *                                    int

        iSyncId   clinet browser sync id
 *
 * RETURNS:                  None
 *
 * COMMENTS:        None
 *
 */

#ifdef LOCAL_ALLOC // Allocate the tpalloc structure
for each transaction
//        This saves on some memory at the expense of
some CPU cycles.
static void ProcessDeliveryForm(EXTENSION_CONTROL_BLOCK
*pECB, int iTermId, int iSyncId)
{
          int                TpRc, iRc;
          char      szTmp[26];
          BOOL      bSuccess;
          long      ilen, *olen;
          char      buf[128];

          DELIVERY_DATA
          *DeliveryDataPtr;   //Delivery Tuxedo Buffer

     if((iRc = ThrTpInit()) <0)
           {
                      // This is bad
                     sprintf(buf, "ProcessDelivery
Failed ThrTpInit: iRc = %d", iRc);
                      LogTuxError(0, buf);
                      ErrorMessage(pECB,
ERR_DELIVERY_MISSING_OCD_KEY, ERR_TYPE_WEBDLL, NULL,
iTermId, iSyncId);
                      return;
           }


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

          if ( !GetKeyValue(pECB->lpszQueryString,
"OCD*", szTmp, sizeof(szTmp)) )
          {
                    ErrorMessage(pECB,
ERR_DELIVERY_MISSING_OCD_KEY, ERR_TYPE_WEBDLL, NULL,
iTermId, iSyncId);
                    return;
          }

          if ( !IsNumeric(szTmp) )
          {
                    ErrorMessage(pECB,
ERR_DELIVERY_CARRIER_INVALID, ERR_TYPE_WEBDLL, NULL,
iTermId, iSyncId);
                    return;
          }

          Term.pClientData[iTermId].DeliveryData.o_carr
ier_id    = atoi(szTmp);

          if (
Term.pClientData[iTermId].DeliveryData.o_carrier_id >
10 ||
Term.pClientData[iTermId].DeliveryData.o_carrier_id < 1
)
          {
                    ErrorMessage(pECB,
ERR_DELIVERY_CARRIER_ID_RANGE, ERR_TYPE_WEBDLL, NULL,
iTermId, iSyncId);
                    return;
          }

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


GetLocalTime(&Term.pClientData[iTermId].DeliveryData.qu
eue_time);

          if ((DeliveryDataPtr = (DELIVERY_DATA
*)tpalloc("CARRAY", NULL, sizeof(DELIVERY_DATA))) ==
NULL)
          {
                    TpRc = tperrno;
```

```
                    sprintf(buf, "ProcessDelivery
Tpcalloc Failed");
                    LogTuxError(TpRc, buf);

          strcpy(Term.pClientData[iTermId].DeliveryData
.execution_status, "Delivery Post Failed");
                    bSuccess = FALSE;
                    WriteZString(pECB,
MakeDeliveryForm(iTermId, iSyncId, FALSE, bSuccess) );
                    return;
          }

     *DeliveryDataPtr =
Term.pClientData[iTermId].DeliveryData;

     ilen = sizeof(DELIVERY_DATA);
     olen = &ilen;

           if ( dLog )
           {
                    FILE *fp;

                    fp = fopen(szTpccLogPath, "ab");
                    fprintf(fp, "* ProcessDelivery
Thread %d iTermId %d DeliveryDataPtr: %x \r\n",
                              GetCurrentThreadId(),
iTermId, &DeliveryDataPtr);
                    fclose(fp);
           }

     if (( iRc = tpacall("DELIVERY", (char
*)DeliveryDataPtr, ilen, TPNOREPLY)) == -1)
           {
                    TpRc = tperrno;
                    sprintf(buf, "ProcessDelivery
Tpcalloc Failed");
                    LogTuxError(TpRc, buf);

          strcpy(Term.pClientData[iTermId].DeliveryData
.execution_status, "Delivery Post Failed");
                    bSuccess = FALSE;
                    WriteZString(pECB,
MakeDeliveryForm(iTermId, iSyncId, FALSE, bSuccess) );
                    return;
     }

           if ( dLog )
           {
                    FILE *fp;

                    fp = fopen(szTpccLogPath, "ab");
                    fprintf(fp, "** ProcessDelivery
Thread %d iTermId %d DeliveryDataPtr: %x TpRc = %d
\r\n",
                              GetCurrentThreadId(),
iTermId, &DeliveryDataPtr, TpRc);
                    fclose(fp);
           }

          tpfree((char *)DeliveryDataPtr);

          strcpy(Term.pClientData[iTermId].DeliveryData
.execution_status, "Delivery has been queued.");
          bSuccess = TRUE;

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

    return;

}
#else // Not LOCAL_ALLOC
static void ProcessDeliveryForm(EXTENSION_CONTROL_BLOCK
*pECB, int iTermId, int iSyncId)
{
          int                 TpRc, iRc;
          char      szTmp[26];
          BOOL      bSuccess;
          long      ilen, *olen;
          char      buf[128];

     if((iRc = ThrTpInit()) <0)
           {
                     // This is bad
                    sprintf(buf, "ProcessDelivery
Failed ThrTpInit: iRc = %d", iRc);
                    LogTuxError(0, buf);
                    ErrorMessage(pECB,
ERR_DELIVERY_MISSING_OCD_KEY, ERR_TYPE_WEBDLL, NULL,
iTermId, iSyncId);
                    return;
           }

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

          if ( !GetKeyValue(pECB->lpszQueryString,
"OCD*", szTmp, sizeof(szTmp)) )
          {
                    ErrorMessage(pECB,
ERR_DELIVERY_MISSING_OCD_KEY, ERR_TYPE_WEBDLL, NULL,
iTermId, iSyncId);
                    return;
          }

          if ( !IsNumeric(szTmp) )
          {
                    ErrorMessage(pECB,
ERR_DELIVERY_CARRIER_INVALID, ERR_TYPE_WEBDLL, NULL,
iTermId, iSyncId);
                    return;
          }

          Term.pClientData[iTermId].DeliveryData.o_carr
ier_id    = atoi(szTmp);

          if (
Term.pClientData[iTermId].DeliveryData.o_carrier_id >
10 ||
Term.pClientData[iTermId].DeliveryData.o_carrier_id < 1
)
          {
                    ErrorMessage(pECB,
ERR_DELIVERY_CARRIER_ID_RANGE, ERR_TYPE_WEBDLL, NULL,
iTermId, iSyncId);
                    return;
          }

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


GetLocalTime(&Term.pClientData[iTermId].DeliveryData.qu
eue_time);

     Term.pClientData[iTermId].TuxDataPtr->DeliveryData
= Term.pClientData[iTermId].DeliveryData;

     ilen = sizeof(TUX_DATA);
     olen = &ilen;

           if ( dLog )
           {
                    FILE *fp;

                    fp = fopen(szTpccLogPath, "ab");
                    fprintf(fp, "* ProcessDelivery
Thread %d iTermId %d TuxDataPtr: %x \r\n",
                              GetCurrentThreadId(),
iTermId, &Term.pClientData[iTermId].TuxDataPtr);
                    fclose(fp);
           }

     if (( iRc = tpacall("DELIVERY", (char
*)Term.pClientData[iTermId].TuxDataPtr, ilen,
TPNOREPLY)) == -1)
           {
                    TpRc = tperrno;
                    sprintf(buf, "ProcessDelivery
Tpcalloc Failed");
                    LogTuxError(TpRc, buf);

          strcpy(Term.pClientData[iTermId].DeliveryData
.execution_status, "Delivery Post Failed");
                    bSuccess = FALSE;
                    WriteZString(pECB,
MakeDeliveryForm(iTermId, iSyncId, FALSE, bSuccess) );
                    return;
     }

           if ( dLog )
           {
                    FILE *fp;

                    fp = fopen(szTpccLogPath, "ab");
                    fprintf(fp, "** ProcessDelivery
Thread %d iTermId %d TuxDataPtr: %x TpRc = %d \r\n",
                              GetCurrentThreadId(),
iTermId, &Term.pClientData[iTermId].TuxDataPtr, TpRc);
                    fclose(fp);
           }

          strcpy(Term.pClientData[iTermId].DeliveryData
.execution_status, "Delivery has been queued.");
          bSuccess = TRUE;
```

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

}
#endif // LOCAL_ALLOC

/* 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
 *
 */

#ifdef LOCAL_ALLOC // Allocate the tpalloc structure
for each transaction
//        This saves on some memory at the expense of
some CPU cycles.
static void
ProcessStockLevelForm(EXTENSION_CONTROL_BLOCK *pECB,
int iTermId, int iSyncId)
{
         int               TpRc, iRc, iError;
         char      szTmp[26];
         long      ilen, *olen;
         char      buf[128];

         STOCK_LEVEL_DATA    *StockLevelDataPtr;
         //Stock Level Tuxedo Buffer

          if((iRc = ThrTpInit()) <0)
          {
                    // This is bad
                   sprintf(buf, "ProcessStockLevel
Failed ThrTpInit: iRc = %d", iRc);
                   LogTuxError(0, buf);
                   ErrorMessage(pECB,
ERR_STOCKLEVEL_NOT_PROCESSED, ERR_TYPE_WEBDLL, NULL,
iTermId, iSyncId);
                   return;
          }

     memset(&Term.pClientData[iTermId].StockLevelData,
0, sizeof(STOCK_LEVEL_DATA));

     Term.pClientData[iTermId].StockLevelData.w_id =
Term.pClientData[iTermId].w_id;
     Term.pClientData[iTermId].StockLevelData.d_id =
Term.pClientData[iTermId].d_id;


     if ( !GetKeyValue(pECB->lpszQueryString, "TT*",
szTmp, sizeof(szTmp)) )
     {
         ErrorMessage(pECB,
ERR_STOCKLEVEL_MISSING_THRESHOLD_KEY, ERR_TYPE_WEBDLL,
NULL, iTermId, iSyncId);
         return;
     }

     if ( !IsNumeric(szTmp) )
     {
         ErrorMessage(pECB,
ERR_STOCKLEVEL_THRESHOLD_INVALID, ERR_TYPE_WEBDLL,
NULL, iTermId, iSyncId);
         return;
     }


Term.pClientData[iTermId].StockLevelData.thresh_hold =
atoi(szTmp);

     if (
Term.pClientData[iTermId].StockLevelData.thresh_hold >=
100 ||
Term.pClientData[iTermId].StockLevelData.thresh_hold <
0 )
     {
          ErrorMessage(pECB,
ERR_STOCKLEVEL_THRESHOLD_RANGE, ERR_TYPE_WEBDLL, NULL,
iTermId, iSyncId);
          return;
     }

          if ((StockLevelDataPtr = (STOCK_LEVEL_DATA
*)tpalloc("CARRAY", NULL, sizeof(STOCK_LEVEL_DATA))) ==
NULL)
          {
                    TpRc = tperrno;
                    sprintf(buf, "ProcessStockLevel
Tpcalloc Failed");
                    LogTuxError(TpRc, buf);
                    ErrorMessage(pECB,
ERR_STOCKLEVEL_NOT_PROCESSED, ERR_TYPE_WEBDLL, NULL,
iTermId, iSyncId);
                    return;
          }

    *StockLevelDataPtr =
Term.pClientData[iTermId].StockLevelData;

     ilen = sizeof(STOCK_LEVEL_DATA);
     olen = &ilen;

           if ( dLog )
           {
                    FILE *fp;

                    fp = fopen(szTpccLogPath, "ab");
                    fprintf(fp, "* ProcessStockLevel
Thread %d iTermId %d StockLevelDataPtr: %x \r\n",
                                   GetCurrentThreadId(),
iTermId, &StockLevelDataPtr);
                    fclose(fp);
           }

     if (( iRc = tpcall("STOCKLEVEL", (char
*)StockLevelDataPtr, ilen,
                    (char **)&StockLevelDataPtr, (long
*) olen, TPSIGRSTRT)) == -1)
          {
         TpRc = tperrno;
                    sprintf(buf, "ProcessStockLevel
tpcall failed");
                    LogTuxError(TpRc, buf);
                    ErrorMessage(pECB,
ERR_STOCKLEVEL_NOT_PROCESSED, ERR_TYPE_WEBDLL, NULL,
iTermId, iSyncId);
         return;
     }

           if ( dLog )
           {
                    FILE *fp;

                    fp = fopen(szTpccLogPath, "ab");
                    fprintf(fp, "** ProcessStockLevel
Thread %d iTermId %d StockLevelDataPtr: %x \r\n",
                                   GetCurrentThreadId(),
iTermId, &StockLevelDataPtr);
                    fclose(fp);
           }

     Term.pClientData[iTermId].StockLevelData =
*StockLevelDataPtr;

     iRc = StockLevelDataPtr->retval;
           iError = StockLevelDataPtr->error;

           tpfree((char *)StockLevelDataPtr);

           if ( iRc == 0 )
           {
                    if ( iError == ERR_TYPE_DEADLOCK )
                              ErrorMessage(pECB,
ERR_STOCKLEVEL_NOT_PROCESSED, ERR_TYPE_DEADLOCK, NULL,
iTermId, iSyncId);
                    else
                              ErrorMessage(pECB,
ERR_STOCKLEVEL_NOT_PROCESSED, ERR_TYPE_WEBDLL, NULL,
iTermId, iSyncId);
     }
     else
          WriteZString(pECB,
MakeStockLevelForm(iTermId, iSyncId, FALSE) );
```

```
     return;
}
#else // Not LOCAL_ALLOC
static void
ProcessStockLevelForm(EXTENSION_CONTROL_BLOCK *pECB,
int iTermId, int iSyncId)
{
          int                TpRc, iRc, iError;
          char      szTmp[26];
          long      ilen, *olen;
          char      buf[128];

     if((iRc = ThrTpInit()) <0)
           {
                     // This is bad
                     sprintf(buf, "ProcessStockLevel
Failed ThrTpInit: iRc = %d", iRc);
                     LogTuxError(0, buf);
                     ErrorMessage(pECB,
ERR_STOCKLEVEL_NOT_PROCESSED, ERR_TYPE_WEBDLL, NULL,
iTermId, iSyncId);
                     return;
           }

     memset(&Term.pClientData[iTermId].StockLevelData,
0, sizeof(STOCK_LEVEL_DATA));

     Term.pClientData[iTermId].StockLevelData.w_id =
Term.pClientData[iTermId].w_id;
     Term.pClientData[iTermId].StockLevelData.d_id =
Term.pClientData[iTermId].d_id;


     if ( !GetKeyValue(pECB->lpszQueryString, "TT*",
szTmp, sizeof(szTmp)) )
     {
          ErrorMessage(pECB,
ERR_STOCKLEVEL_MISSING_THRESHOLD_KEY, ERR_TYPE_WEBDLL,
NULL, iTermId, iSyncId);
          return;
     }

     if ( !IsNumeric(szTmp) )
     {
          ErrorMessage(pECB,
ERR_STOCKLEVEL_THRESHOLD_INVALID, ERR_TYPE_WEBDLL,
NULL, iTermId, iSyncId);
          return;
     }


Term.pClientData[iTermId].StockLevelData.thresh_hold =
atoi(szTmp);

     if (
Term.pClientData[iTermId].StockLevelData.thresh_hold >=
100 ||
Term.pClientData[iTermId].StockLevelData.thresh_hold <
0 )
     {
          ErrorMessage(pECB,
ERR_STOCKLEVEL_THRESHOLD_RANGE, ERR_TYPE_WEBDLL, NULL,
iTermId, iSyncId);
          return;
     }

     Term.pClientData[iTermId].TuxDataPtr-
>StockLevelData =
Term.pClientData[iTermId].StockLevelData;

     ilen = sizeof(TUX_DATA);
     olen = &ilen;

           if ( dLog )
           {
                     FILE *fp;

                     fp = fopen(szTpccLogPath, "ab");
                     fprintf(fp, "* ProcessStockLevel
Thread %d iTermId %d TuxDataPtr: %x \r\n",
                                GetCurrentThreadId(),
iTermId, &Term.pClientData[iTermId].TuxDataPtr);
                     fclose(fp);
           }

     if (( iRc = tpcall("STOCKLEVEL", (char
*)Term.pClientData[iTermId].TuxDataPtr, ilen,
                      (char
**)&Term.pClientData[iTermId].TuxDataPtr, (long *)
olen, TPSIGRSTRT)) == -1)
           {
          TpRc = tperrno;
                     sprintf(buf, "ProcessStockLevel
tpcall failed");
                     LogTuxError(TpRc, buf);
                     ErrorMessage(pECB,
ERR_STOCKLEVEL_NOT_PROCESSED, ERR_TYPE_WEBDLL, NULL,
iTermId, iSyncId);
          return;
     }

           if ( dLog )
           {
                     FILE *fp;

                     fp = fopen(szTpccLogPath, "ab");
                     fprintf(fp, "** ProcessStockLevel
Thread %d iTermId %d TuxDataPtr: %x \r\n",
                                GetCurrentThreadId(),
iTermId, &Term.pClientData[iTermId].TuxDataPtr);
                     fclose(fp);
           }

     Term.pClientData[iTermId].StockLevelData =
Term.pClientData[iTermId].TuxDataPtr->StockLevelData;

     iRc = Term.pClientData[iTermId].TuxDataPtr-
>StockLevelData.retval;
           iError =
Term.pClientData[iTermId].TuxDataPtr-
>StockLevelData.error;

           if ( iRc == 0 )
           {
                     if ( iError == ERR_TYPE_DEADLOCK )
                               ErrorMessage(pECB,
ERR_STOCKLEVEL_NOT_PROCESSED, ERR_TYPE_DEADLOCK, NULL,
iTermId, iSyncId);
                     else
                               ErrorMessage(pECB,
ERR_STOCKLEVEL_NOT_PROCESSED, ERR_TYPE_WEBDLL, NULL,
iTermId, iSyncId);
     }
     else
          WriteZString(pECB,
MakeStockLevelForm(iTermId, iSyncId, FALSE) );

           return;

}
#endif // LOCAL_ALLOC

/* FUNCTION: int GetNewOrderData(LPSTR lpszQueryString,
NEW_ORDER_DATA *pNewOrderData)
 *
 * PURPOSE:         This function extracts and
validates the new order form data from an 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;
          dLog                              = FALSE;
          iMaxWareHouses      = 500;
          iThreads            = 5;
          iQSlotts            = 3000;
          iDelayMs            = 100;
          iDeadlockRetry      = (short)3;
          strcpy(szTpccLogPath, "tpcclog.");
          strcpy(szErrorLogPath, "tpccerr.");


          if ( RegOpenKeyEx(HKEY_LOCAL_MACHINE,
"SOFTWARE\\Microsoft\\TPCC", 0, KEY_READ, &hKey) !=
ERROR_SUCCESS )
                    return TRUE;
          size = sizeof(szTmp);
```

```
          if ( RegQueryValueEx(hKey, "PATH", 0, &type,
szTmp, &size) == ERROR_SUCCESS )
          {
                    strcpy(szTpccLogPath, szTmp);
                    strcat(szTpccLogPath, "tpcclog.");
                    strcpy(szErrorLogPath, szTmp);
                    strcat(szErrorLogPath, "tpccerr.");
          }

          size = sizeof(szTmp);
          if ( RegQueryValueEx(hKey, "LOG", 0, &type,
szTmp, &size) == ERROR_SUCCESS )
          {
                    if ( !stricmp(szTmp, "ON") )
                              bLog = TRUE;
          }

          size = sizeof(szTmp);
          if ( RegQueryValueEx(hKey, "DEBUG", 0, &type,
szTmp, &size) == ERROR_SUCCESS )
          {
                    if ( !stricmp(szTmp, "ON") )
                              dLog = TRUE;
          }

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

          size = sizeof(szTmp);
          if ( RegQueryValueEx(hKey,
"NumberOfDeliveryThreads", 0, &type, szTmp, &size) ==
ERROR_SUCCESS )
                    iThreads = atoi(szTmp);
          if ( !iThreads )
                    iThreads = 5;

          size = sizeof(szTmp);
          if ( RegQueryValueEx(hKey, "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;

          RegCloseKey(hKey);

          return FALSE;

}



/* FUNCTION: BOOL IsNumeric(char *ptr)
 *
 * PURPOSE:         This function determines if a
string is numeric. It fails if any characters other
 *                          than numeric and null
terminator are present.
 *
 * ARGUMENTS:       char                          *ptr
          pointer to string to check.
 *
 * RETURNS:                  BOOL      FALSE     if
string is not all numeric
 *
          TRUE      if string contains only numeric
characters i.e. '0' - '9'
 *
 * COMMENTS:        None
 *
 */

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

          while( *ptr && isdigit(*ptr) )
                    ptr++;
          return ( !*ptr );
}


/* FUNCTION: void FormatHTMLString(char *szBuff, int
iLen, char *szStr)
 *
 * PURPOSE:         This function Handles translation
of HTML specific character field data
 *                          when an HTML output form
is generated.
 *
 * ARGUMENTS:       char      *szBuff   Returned string
information
 *                                    char
          *szStr    input string to be formatted.
 *                                    int
          iLen      Length of returned string
 *
 * RETURNS:                  none
 *
 * COMMENTS:        The length paramter is the absolute
length of the returned string in
 *                                    HTML
characters. For example the input string > would be
returned as
 *                                    &gt; which
would be counted as 1 character.If the number of input
 *                                    characters is
less than the iLen parameter spaces are appended to
 *                                    the end of the
string to ensure that at least iLen characters are
 *                                    returned in the
szBuff parameter.
 *
 */

static void FormatHTMLString(char *szBuff, char *szStr,
int iLen)
{
          while( iLen && *szStr )
          {
                    switch( *szStr )
                    {
                              case '>':
                                        *szBuff++ =
'&';
                                        *szBuff++ =
'g';
                                        *szBuff++ =
't';
                                        *szBuff++ =
';';
                                        szStr++;
                                        break;
                              case '<':
                                        *szBuff++ =
'&';
                                        *szBuff++ =
'l';
                                        *szBuff++ =
't';
                                        *szBuff++ =
';';
                                        szStr++;
                                        break;
                              case '&':
                                        *szBuff++ =
'&';
                                        *szBuff++ =
'a';
                                        *szBuff++ =
'm';
                                        *szBuff++ =
'p';
```

```
                                        *szBuff++ =
';';
                                        szStr++;
                                        break;
                                case '\"':
                                        *szBuff++ =
'&';
                                        *szBuff++ =
'q';
                                        *szBuff++ =
'u';
                                        *szBuff++ =
'o';
                                        *szBuff++ =
't';
                                        *szBuff++ =
';';
                                        szStr++;
                                        break;
                                default:
                                        *szBuff++ =
*szStr++;
                                        break;
                        }
                        iLen--;
                }
                while( iLen-- )
                        *szBuff++ = ' ';

                *szBuff = 0;

                return;
}


static int ThrTpInit()
{

     static int num_tpinits=0;
     static int x=1;
     static int once=0;
     static CRITICAL_SECTION     TpCriticalSection;
           int lasterr, iRc, TpRc;
           int retry = 0;
           BOOL Success = FALSE;

     if(!TlsGetValue(TLSIsTpInitedKey))
     {
                        if (!once)
                        {

                InitializeCriticalSection(&TpCriticalSection)
;
                                once=1;
                        }

                        if ( dLog )
                        {
                                FILE *fp;

                                fp = fopen(szTpccLogPath,
"ab");
                                fprintf(fp, "* In
ThrTpInit Thread %d * \r\n", GetCurrentThreadId());
                                fclose(fp);
                        }

                        while ( retry < TP_MAX_RETRIES )
                        {

                EnterCriticalSection(&TpCriticalSection);
                                if(tpinf == NULL)
                                {
                                        if ((tpinf = (
TPINIT *)tpalloc("TPINIT", NULL, sizeof(TPINIT))) ==
NULL)
                                   {

                LeaveCriticalSection(&TpCriticalSection);
                                                        TpRc
= tperrno;
                                                        {

                FILE *fp;


                fp = fopen(szErrorLogPath, "ab");

                fprintf(fp, ">>>> ThrTpInit:%d : tpalloc of
tpinit failed: %d : %s\r\n",

                        GetCurrentThreadId(), TpRc,
tpstrerror(TpRc));

                fclose(fp);
                                                        }

                retry++;

                continue;
                                        }
                                tpinf-
>flags|=TPMULTICONTEXTS;
                                }

                                if (retry == 0)

                itoa(++num_tpinits, tpinf->cltname, 10);
// Do the TPINIT
                                iRc = tpinit(tpinf);
                                TpRc = tperrno;

                                // check tpalloc() ?
                                if (iRc < 0)
                                {

                LeaveCriticalSection(&TpCriticalSection);
                                        retry++;
                                        lasterr =
GetLastError();
/*                                      TpRc = tperrno;
                                        {
                                                FILE
*fp;

                                                fp =
fopen(szErrorLogPath, "ab");

                fprintf(fp, ">>>> ThrTpInit:%d : tpinit
failed: %d %s :try # %d\r\n",

                GetCurrentThreadId(), iRc, tpstrerror(TpRc),
retry);

                fclose(fp);
                                        }
*/
                                }
                                else
                                {
                                        Success = TRUE;

                LeaveCriticalSection(&TpCriticalSection);
                                        break;
                                }

                                Sleep(50); // Relinquish
thread timeslice
                        } // retry the tpinit if it failed
the first time

                        if ( Success == FALSE )
                        {
                                {
                                        char ebuf[128];

                                        sprintf(ebuf,
">>>> ThrTpInit %d : Cannot tpinit after %d tries iRc =
%d LastErr = %d \r\n",

                GetCurrentThreadId(), TP_MAX_RETRIES, iRc,
lasterr);

                LogTuxError(TpRc, ebuf);
                                }
                                return -1;
                        }
                        if ( Success == TRUE )
                        {
                                if ( retry > 0 )
                                {
                                        char ebuf[128];

                                        sprintf(ebuf,
">>>> ThrTpInit %d : Cannot tpinit after %d tries iRc =
%d LastErr = %d \r\n",

                GetCurrentThreadId(), TP_MAX_RETRIES, iRc,
lasterr);

                                        sprintf(ebuf,
"* ThrTpInit Thread %d Success retry count %d with
LastErr = %d * \r\n",

                GetCurrentThreadId(), retry, lasterr);
```

```
        LogTuxError(TpRc, ebuf);
                        }

                        if ( (
iRc=TlsSetValue(TLSIsTpInitedKey,&x)) == 0)
                        {
                                {
                                        FILE
*fp;

                                        fp =
fopen(szErrorLogPath, "ab");

        fprintf(fp, ">>>> ThrTpInit %d : TlsSetValue
Failed iRc: %d \r\n",

        GetCurrentThreadId(), iRc);

        fclose(fp);
                                }
                        }
                }
        }
        else
        {
                if ( dLog )
                {
                        FILE *fp;

                        fp = fopen(szTpccLogPath,
"ab");
                        fprintf(fp, "* ThrTpInit
Thread %d already tpinited * \r\n",
GetCurrentThreadId());
                        fclose(fp);
                }
        }
    return 0;
}
```

Neworder.c

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

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

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

#include "trans.h"
                //tpckit transaction header
contains definations of structures specific to TPC-C
#include "httpext.h"
                //ISAPI DLL information header
#include "tpcc.h"
                //this dlls specific structure,
value e.t. header.

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

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

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

DBPROCESS *pdbproc;

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

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

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

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

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

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

int tpsvrinit ( int argc, char *argv[] )
{

        if ( GetParameters(argc, argv) )
        {
                PrintParameters();
                return -1;
        }

        if ( verbose )
                TMLog("TPSVRINIT: NewOrder: Server
%s, Database %s, User %s, Password %s, Flush %d.",
```

```
                              szServer, szDatabase,
szUser, szPassword, bFlush);


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

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

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

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

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

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

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

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

          bError = FALSE;

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

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

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

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

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

          bError = FALSE;

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

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

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

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

/* 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
 *
 */

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);
//                      dbrpcparam(dbproc, NULL,
0, SQLINT1, -1, -1, (BYTE *) &pNewOrder->o_all_local);

                        pNewOrder->o_all_local =
1;
                        for (i = 0; i <
pNewOrder->o_ol_cnt; i++)
                        {
                                if ( pNewOrder-
>o_all_local && pNewOrder->Ol[i].ol_supply_w_id !=
pNewOrder->w_id )

        pNewOrder->o_all_local = 0;
                        }
                        dbrpcparam(dbproc, NULL,
0, SQLINT1, -1, -1, (BYTE *) &pNewOrder->o_all_local);

                        for (i = 0; i <
pNewOrder->o_ol_cnt; i++)
                        {

        dbrpcparam(dbproc, NULL, 0, SQLINT4, -1, -1,
(BYTE *) &pNewOrder->Ol[i].ol_i_id);

        dbrpcparam(dbproc, NULL, 0, SQLINT2, -1, -1,
(BYTE *) &pNewOrder->Ol[i].ol_supply_w_id);

        dbrpcparam(dbproc, NULL, 0, SQLINT2, -1, -1,
(BYTE *) &pNewOrder->Ol[i].ol_quantity);
                        }

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

                                // Get 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))


        dbconvert(dbproc, SQLNUMERIC, pData,
dbdatlen(dbproc,4), SQLFLT8, (BYTE *)&pNewOrder-
>Ol[i].ol_i_price, 8);



                        if(pData=dbdata(dbproc,
5))



        dbconvert(dbproc, SQLNUMERIC, pData,
dbdatlen(dbproc,5), SQLFLT8, (BYTE *)&pNewOrder-
>Ol[i].ol_amount, 8);







                        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))


                        dbconvert(dbproc,
SQLNUMERIC, pData, dbdatlen(dbproc,1), SQLFLT8, (BYTE
*)&pNewOrder->w_tax, 8);



                if(pData=dbdata(dbproc, 2))


                        dbconvert(dbproc,
SQLNUMERIC, pData, dbdatlen(dbproc,2), SQLFLT8, (BYTE
*)&pNewOrder->d_tax, 8);



                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))
```

```
                                dbconvert(dbproc,
SQLNUMERIC, pData, dbdatlen(dbproc,5), SQLFLT8, (BYTE
*)&pNewOrder->c_discount, 8);









                    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.");
                                          pNewOrder-
>error=ERR_BAD_ITEM_ID;
                                          return FALSE;
                                }
                    }
          }

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

          return -1;          //        "deadlock max
retry reached!"
}

/*
 *        Common Code for all Servers
 */

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

          dbinit();

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

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

          return TRUE;
}


/* FUNCTION: BOOL
SQLOpenConnection(EXTENSION_CONTROL_BLOCK
          *pECB, 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, long *pack_size)
          {
```

```
                    RETCODE   rc;
                    char      buffer[30];

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

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


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

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

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

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

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

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

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

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

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


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

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

                    return FALSE;
          }

#else

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

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

                    gpECB = pECB;

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

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

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

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

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

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

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

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

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

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

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

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

                    return FALSE;
          }

#endif

/* FUNCTION: BOOL
SQLCloseConnection(EXTENSION_CONTROL_BLOCK *pECB,
DBPROCESS *dbproc)
 *
 * PURPOSE:         This function closes the sql
connection.
 *
```

```
 * ARGUMENTS:       EXTENSION_CONTROL_BLOCK       *pECB
         passed in structure pointer from inetsrv.
 *                                  DBPROCESS
         *dbproc   pointer to DBPROCESS
 *
 * RETURNS:                  BOOL      FALSE     if
successfull
 *
         TRUE      if an error occurs
 *
 * COMMENTS:        None
 *
 */

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

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

BOOL SQLDetectDeadlock(DBPROCESS *dbproc)
{
         PECBINFO  pEcbInfo;

         if ( (pEcbInfo =
(PECBINFO)dbgetuserdata(dbproc)) )
         {
                  if ( pEcbInfo->bDeadlock )
                  {
                           pEcbInfo->bDeadlock =
FALSE;
                           return TRUE;
                  }
         }
         return FALSE;
}

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

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

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

         return;
}

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


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

         pEcbInfo = NULL;

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

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

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

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

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

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

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

                fclose(fp);
        }

        return INT_CANCEL;
}


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

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

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


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

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

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

                if ( pEcbInfo )
                        pEcbInfo->bFailed = TRUE;

                bError = TRUE;

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

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


/* FUNCTION: BOOL GetParameters(int argc, char *argv[])
 *
 * PURPOSE:         This function parses the command
line passed in to the delivery executable, initializing
 *                          and filling in global
variable parameters.
 *
 * ARGUMENTS:       int                 argc
        number of command line arguments passed to
delivery
 *                                      char
        *argv[]   array of command line argument
pointers
 *
 * RETURNS:                 BOOL      FALSE
        parameter read successfull
 *
        TRUE      user has requested parameter
information screen be displayed.
 *
```

```
 * COMMENTS:        None
 *
 */

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

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

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

          strcpy(szServer, argv[i]+2);

          break;
                                        case 'V':
                                        case 'v':

          verbose = TRUE;

          break;
                                        case '?':

          return TRUE;
                              }
                    }
          }
          return FALSE;
}

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

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

}
```

Orderstatus.c

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

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

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

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

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

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

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

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

DBPROCESS *pdbproc;

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

#ifdef LOCAL_ALLOC
ORDER_STATUS_DATA OrderStatusData;
#else
TUX_DATA TuxData;
#endif
TERM Term;

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

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

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

/* FUNCTION: tpsvrinit ( int argc, char *argv[])
 *
 * PURPOSE:         Initialize the Server to Database
connection.
```

```
 *
 * RETURNS:                    int       0
        Success
 *                                                -1
        Failure
 *
 *
 * COMMENTS:        None
 *
 */

int tpsvrinit ( int argc, char *argv[] )
{

        if ( GetParameters(argc, argv) )
        {
                PrintParameters();
                return -1;
        }

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


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

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

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

/* FUNCTION: ORDERSTATUS ( TPSVCINFO *rqst )
 *
 * PURPOSE:         Process an Order Status request.
 *
 * RETURNS:                    int       0
        Success
 *                                                -1
        Failure
 *
 *
 * COMMENTS:        None
 *
 */

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

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

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

        bError = FALSE;

        OrderStatusData.retval = SQLOrderStatus(
NULL, 0, 0, pdbproc, &OrderStatusData, iDeadlockRetry);

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

        if ( verbose )
                TMLog(" ORDERSTATUS: Return Value
%d", OrderStatusData.retval);

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

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

        bError = FALSE;

        TuxData.OrderStatusData.retval =
SQLOrderStatus( NULL, 0, 0, pdbproc,
&TuxData.OrderStatusData, iDeadlockRetry);

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

        if ( verbose )
                TMLog(" ORDERSTATUS: Return Value
%d  error = %d",

        TuxData.OrderStatusData.retval,
TuxData.OrderStatusData.error);

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

/* FUNCTION: int SQLOrderStatus(EXTENSION_CONTROL_BLOCK
*pECB, int iTermId, int iSyncId, DBPROCESS *dbproc,
ORDER_STATUS_DATA *pOrderStatus, short deadlock_retry)
 *
 * PURPOSE:         This function processes the Order
Status transaction.
 *
 * ARGUMENTS:       EXTENSION_CONTROL_BLOCK       *pECB
                        passed in structure
pointer from inetsrv.
 *                                      int

        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
 *
 */
```

```
static int SQLOrderStatus(EXTENSION_CONTROL_BLOCK
*pECB, int iTermId, int iSyncId, DBPROCESS *dbproc,
ORDER_STATUS_DATA *pOrderStatus, short deadlock_retry)
{
          RETCODE             rc;
          int                           tryit;
          int                           i;
          char                printbuf[25];
//        BOOL                by_name;
          DBDATETIME          datetime;
          BYTE                *pData;
          PECBINFO  pEcbInfo;

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

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

          for (tryit=0; tryit < deadlock_retry;
tryit++)
          {
                    if (dbrpcinit(dbproc,
"tpcc_orderstatus", 0) == SUCCEED)
                    {
                              dbrpcparam(dbproc, NULL,
0, SQLINT2, -1, -1, (BYTE *) &pOrderStatus->w_id);
                              dbrpcparam(dbproc, NULL,
0, SQLINT1, -1, -1, (BYTE *) &pOrderStatus->d_id);
                              dbrpcparam(dbproc, NULL,
0, SQLINT4, -1, -1, (BYTE *) &pOrderStatus->c_id);
                              if (pOrderStatus->c_id ==
0)
                              {


          dbrpcparam(dbproc, NULL, 0, SQLCHAR, -1,
strlen(pOrderStatus->c_last), pOrderStatus->c_last);
                              }
                    }
                    if (dbrpcexec(dbproc) == SUCCEED)
                    {
                              while (((rc =
dbresults(dbproc)) != NO_MORE_RESULTS) && (rc != FAIL))

                              {
                                        if
(DBROWS(dbproc) && (dbnumcols(dbproc) == 5))
                                        {
                                                  i=0;
                                                  while
(((rc = dbnextrow(dbproc)) != NO_MORE_ROWS) && (rc !=
FAIL))
                                                  {

          if(pData=dbdata(dbproc, 1))

                    pOrderStatus-
>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))

                    dbconvert(dbproc, SQLNUMERIC,
pData, dbdatlen(dbproc,4), SQLFLT8, (BYTE
*)&pOrderStatus->OlOrderStatusData[i].ol_amount, 8);

          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))

                    dbconvert(dbproc, SQLNUMERIC,
pData, dbdatlen(dbproc,7), SQLFLT8, (BYTE
*)&pOrderStatus->c_balance, 8);

          if(pData=dbdata(dbproc, 8))

                    pOrderStatus->o_id = (*(DBINT *)
pData);
                                                  }
                                        }
                                        if (i==0)

          return 0; //"No orders found for customer"
                              }
                    }
                    if (SQLDetectDeadlock(dbproc))
                    {
                              pOrderStatus-
>num_deadlocks++;

          sprintf(printbuf,"deadlock: retry:
%d",pOrderStatus->num_deadlocks);

          Sleep(DEADLOCKWAIT*tryit);
                    }
                    else
                    {
                              if (pOrderStatus->c_id ==
0 && pOrderStatus->c_last[0] == 0)
                              {

          strcpy(pOrderStatus-
>execution_status,"Invalid Customer id,name.");
```

```
                                        pOrderStatus-
>error=ERR_NOSUCH_CUSTOMER;
                                        TMLog("
ORDERSTATUS: No such customer ");
                                }
                                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.  ");
          pOrderStatus->error=ERR_TYPE_DEADLOCK;
          return -1; //"deadlock max retry reached!"
}

/*
 *        Common Code for all Servers
 */

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

          dbinit();

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

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

          return TRUE;
}


/* FUNCTION: BOOL
SQLOpenConnection(EXTENSION_CONTROL_BLOCK
          *pECB, 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, long *pack_size)
          {

                    RETCODE   rc;
                    char      buffer[30];

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

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


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

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

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

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

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

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

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

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

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

```
                        return TRUE;


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

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

                return FALSE;
        }

#else

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

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

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

                gpECB = pECB;

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

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

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

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

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

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

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

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

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

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

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

                return FALSE;
        }

#endif

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

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

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

BOOL SQLDetectDeadlock(DBPROCESS *dbproc)
{
        PECBINFO  pEcbInfo;

        if ( (pEcbInfo =
(PECBINFO)dbgetuserdata(dbproc)) )
```

```
        {
                if ( pEcbInfo->bDeadlock )
                {
                        pEcbInfo->bDeadlock =
FALSE;
                        return TRUE;
                }
        }
        return FALSE;
}

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

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

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

        return;
}

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

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

        pEcbInfo = NULL;

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

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

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

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

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

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

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

                fclose(fp);
        }
        return INT_CANCEL;
}


/* FUNCTION: int msg_handler(DBPROCESS *dbproc, DBINT
msgno, int msgstate, int severity, char *msgtext)
 *
 * PURPOSE:         This function handles DB-Library
SQL Server error messages
 *
 * ARGUMENTS:       DBPROCESS           *dbproc
                  DBPROCESS id pointer
 *                                  DBINT
                  msgno
        message number
 *                                  int
                          msgstate
        message state
 *                                  int
                          severity
        message severity
```

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

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

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


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

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

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

                    if ( pEcbInfo )
                              pEcbInfo->bFailed = TRUE;

                    bError = TRUE;

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

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


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

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


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

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

          strcpy(szServer, argv[i]+2);

          break;
                                        case 'V':
                                        case 'v':

          verbose = TRUE;

          break;
                                        case '?':

          return TRUE;
                              }
                    }
          }
          return FALSE;
}

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

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

}
```

Appendix A-Application Code

Payment.c

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

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

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

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

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

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

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

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

DBPROCESS *pdbproc;

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

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

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

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

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

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

int tpsvrinit ( int argc, char *argv[] )
{

          if ( GetParameters(argc, argv) )
          {
                    PrintParameters();
                    return -1;
          }

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


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

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

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

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

/* FUNCTION: PAYMENT ( TPSVCINFO *rqst )
```

```
 *
 * PURPOSE:         Process a Payment request.
 *
 * RETURNS:                 int       0
         Success
 *                                          -1
         Failure
 *
 *
 * COMMENTS:        None
 *
 */

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

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

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

         bError = FALSE;

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

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

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

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

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

         bError = FALSE;

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

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

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

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

/* FUNCTION: int SQLPayment(EXTENSION_CONTROL_BLOCK
*pECB, int iTermId, int 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
 *
 */

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];
//       BOOL                by_name;
         DBDATETIME          datetime;
         BYTE                *pData;
         PECBINFO  pEcbInfo;

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

         pPayment->num_deadlocks = 0;

//       if (pPayment->c_id == 0)
//                by_name = TRUE;
//       else
//                by_name = FALSE;
//
         for (tryit=0; tryit < deadlock_retry;
tryit++)
         {
                  if (dbrpcinit(dbproc,
"tpcc_payment", 0) == SUCCEED)
                  {
                           dbrpcparam(dbproc, NULL,
0, SQLINT2, -1, -1, (BYTE *) &pPayment->w_id);
                           dbrpcparam(dbproc, NULL,
0, SQLINT2, -1, -1, (BYTE *) &pPayment->c_w_id);
                           dbrpcparam(dbproc, NULL,
0, SQLFLT8, -1, -1, (BYTE *) &pPayment->h_amount);
                           dbrpcparam(dbproc, NULL,
0, SQLINT1, -1, -1, (BYTE *) &pPayment->d_id);
                           dbrpcparam(dbproc, NULL,
0, SQLINT1, -1, -1, (BYTE *) &pPayment->c_d_id);
                           dbrpcparam(dbproc, NULL,
0, SQLINT4, -1, -1, (BYTE *) &pPayment->c_id);
                           if (pPayment->c_id == 0)
                           {

         dbrpcparam(dbproc, NULL, 0, SQLCHAR, -1,
strlen(pPayment->c_last), pPayment->c_last);
                           }
                  }
                  if (dbrpcexec(dbproc) == SUCCEED)
                  {
                           while (((rc =
dbresults(dbproc)) != NO_MORE_RESULTS) && (rc != FAIL))
                           {
                                    if
(DBROWS(dbproc) && (dbnumcols(dbproc) == 27))
                                    {
                                                       while
(((rc = dbnextrow(dbproc)) != NO_MORE_ROWS) && (rc !=
FAIL))
                                                       {

         if(pData=dbdata(dbproc, 1))

                  pPayment->c_id = *((DBINT *)
pData);

         if(pData=dbdata(dbproc, 2))

                  UtilStrCpy(pPayment->c_last, pData,
dbdatlen(dbproc, 2));
```

```
          if(pData=dbdata(dbproc, 3))

          {

                    datetime = *((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))

                    dbconvert(dbproc, SQLNUMERIC,
pData, dbdatlen(dbproc,24), SQLFLT8, (BYTE *)&pPayment-
>c_credit_lim, 8);

          if(pData=dbdata(dbproc, 25))

                    dbconvert(dbproc, SQLNUMERIC,
pData, dbdatlen(dbproc,25), SQLFLT8, (BYTE *)&pPayment-
>c_discount, 8);

          if(pData=dbdata(dbproc, 26))

                    dbconvert(dbproc, SQLNUMERIC,
pData, dbdatlen(dbproc,26), SQLFLT8, (BYTE *)&pPayment-
>c_balance, 8);

          if(pData=dbdata(dbproc, 27))

                    UtilStrCpy(pPayment->c_data, pData,
dbdatlen(dbproc, 27));
                                                  }
                                        }
                              }
                    }
                    if (SQLDetectDeadlock(dbproc))
                    {
                              pPayment-
>num_deadlocks++;

          sprintf(printbuf,"deadlock: retry:
%d",pPayment->num_deadlocks);

          Sleep(DEADLOCKWAIT*tryit);
                    }
                    else
                    {
                              if ( pPayment->c_id == 0
)
                              {

          strcpy(pPayment->execution_status,"Invalid
Customer id,name.");
                                        pPayment-
>error=ERR_NOSUCH_CUSTOMER;
                                        TMLog("
PAYMENT: No such customer ");
                                        return 0;
                              }
                              else

          strcpy(pPayment-
>execution_status,"Transaction commited.");
                              return TRUE;
                    }
          }
          // If we reached here, it means we quit after
MAX_RETRY deadlocks
          strcpy(pPayment->execution_status,"Hit
deadlock max.  ");
          pPayment->error=ERR_TYPE_DEADLOCK;
          return -1; //"deadlock max retry reached!"
}


/*
 *        Common Code for all Servers
```

```
 */

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

          dbinit();

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

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

          return TRUE;
}


/* FUNCTION: BOOL
SQLOpenConnection(EXTENSION_CONTROL_BLOCK
          *pECB, 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, long *pack_size)
          {

                    RETCODE   rc;
                    char      buffer[30];

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

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


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

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

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

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

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

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

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

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

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


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

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

                    return FALSE;
          }

#else

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

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

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

                    gpECB = pECB;

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

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

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

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

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


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


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


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


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

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

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

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

                    return FALSE;
          }

#endif
```

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

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

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

BOOL SQLDetectDeadlock(DBPROCESS *dbproc)
{
          PECBINFO  pEcbInfo;

          if ( (pEcbInfo =
(PECBINFO)dbgetuserdata(dbproc)) )
          {
                    if ( pEcbInfo->bDeadlock )
                    {
                              pEcbInfo->bDeadlock =
FALSE;
                              return TRUE;
                    }
          }
          return FALSE;
}

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

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

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

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

          return;
}

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


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

          pEcbInfo = NULL;

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

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

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

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

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

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

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

                    fclose(fp);
          }

          return INT_CANCEL;
}


/* FUNCTION: int msg_handler(DBPROCESS *dbproc, DBINT
msgno, int msgstate, int severity, char *msgtext)
 *
 * PURPOSE:         This function handles DB-Library
SQL Server error messages
 *
 * ARGUMENTS:       DBPROCESS           *dbproc
                    DBPROCESS id pointer
 *                                       DBINT
                              msgno
          message number
 *                                       int
                              msgstate
          message state
 *                                       int
                              severity
          message severity
 *                                       char
                    *msgtext            printable
message description
 *
 * RETURNS:                   int
                    INT_CONTINUE        continue if
error is SQLETIME else INT_CANCEL action
 *
                              INT_CANCEL
          cancel operation
 *
 * COMMENTS:        This function also sets the dead
lock dbproc variable if necessary.
 *
 */
```

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

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


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

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

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

                    if ( pEcbInfo )
                              pEcbInfo->bFailed = TRUE;

                    bError = TRUE;

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

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

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

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


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

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

          strcpy(szServer, argv[i]+2);

          break;
                                        case 'V':
                                        case 'v':

          verbose = TRUE;

          break;
                                        case '?':

          return TRUE;
                              }
                    }
          }
          return FALSE;
}

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

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

}
```

Stocklevel.c

```
/*        FILE:               STOCKLEVEL.C
 *
 *        Based on: Microsoft TPC-C Kit Ver. 3.00.000
 *
 *                                      Copyright
Microsoft, 1996
 *                                      Copyright
Performance Tuning Corporation, 1997
 *
 *        PURPOSE:  New Order Tuxedo Server.
 *        Author:             Philip Durr
 *
          philipdu@Microsoft.com
 *
```

```
 *  MODIFIED        Changed for modularity and to allow
for the Tuxedo TM
 *
 *  Author:                     Edward Whalen
 *                                      Performance
Tuning Corporation
 *
         ewhalen@perftuning.com
 */

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

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

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

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

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

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

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

DBPROCESS *pdbproc;

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

#ifdef LOCAL_ALLOC
STOCK_LEVEL_DATA StockLevelData;
#else
TUX_DATA TuxData;
#endif
TERM Term;

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

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

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

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

int tpsvrinit ( int argc, char *argv[] )
{

          if ( GetParameters(argc, argv) )
          {
                    PrintParameters();
                    return -1;
          }

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


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

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

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

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


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

void STOCKLEVEL ( TPSVCINFO *rqst )
{
```

```
        PECBINFO pECBInfo = dbgetuserdata(pdbproc);
        int size = rqst->len;

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

        if (verbose )
        {
                TMLog(" STOCKLEVEL: w_id %d ",
StockLevelData.w_id);
                TMLog(" STOCKLEVEL: d_id %d ",
StockLevelData.d_id);
                TMLog(" STOCKLEVEL: c_id %d ",
StockLevelData.thresh_hold);
        }

        bError = FALSE;

        StockLevelData.retval = SQLStockLevel( NULL,
0, 0, pdbproc, &StockLevelData, iDeadlockRetry);


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

        if ( verbose )
                TMLog(" STOCKLEVEL: Return Value
%d", StockLevelData.retval);

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

        if (verbose )
        {
                TMLog(" STOCKLEVEL: w_id %d ",
TuxData.StockLevelData.w_id);
                TMLog(" STOCKLEVEL: d_id %d ",
TuxData.StockLevelData.d_id);
                TMLog(" STOCKLEVEL: c_id %d ",
TuxData.StockLevelData.thresh_hold);
        }

        bError = FALSE;

        TuxData.StockLevelData.retval =
SQLStockLevel( NULL, 0, 0, pdbproc,
&TuxData.StockLevelData, iDeadlockRetry);


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

        if ( verbose )
                TMLog(" STOCKLEVEL: Return Value
%d", TuxData.StockLevelData.retval);

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

/* FUNCTION: SQLStockLevel(EXTENSION_CONTROL_BLOCK
        *pECB, int iTermId, 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
 *
 */

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 TRUE;
                        }
        }

        // If we reached here, it means we quit after
MAX_RETRY deadlocks
        strcpy(pStockLevel->execution_status, "Hit
deadlock max. ");
        pStockLevel->error=ERR_TYPE_DEADLOCK;
        return -1;
}
```

```
/*
 *        Common Code for all Servers
 */

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

          dbinit();

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

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

          return TRUE;
}


/* FUNCTION: BOOL
SQLOpenConnection(EXTENSION_CONTROL_BLOCK
          *pECB, 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, long *pack_size)
          {

                    RETCODE   rc;
                    char      buffer[30];

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

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


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

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

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

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

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

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

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

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

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


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

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

                    return FALSE;
          }

#else

          static BOOL
SQLOpenConnection(EXTENSION_CONTROL_BLOCK *pECB, int
iTermId, int iSyncId, DBPROCESS **dbproc, char *server,
char *database, char *user, char *password, char *app,
int *spid)
          {
```

```
                    LOGINREC  *login;
                    PECBINFO  pEcbInfo;

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

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

                    gpECB = pECB;

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

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

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

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

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


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


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


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

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

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

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

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

                    return FALSE;
          }

#endif

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

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

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

BOOL SQLDetectDeadlock(DBPROCESS *dbproc)
{
          PECBINFO  pEcbInfo;

          if ( (pEcbInfo =
(PECBINFO)dbgetuserdata(dbproc)) )
          {
                    if ( pEcbInfo->bDeadlock )
                    {
                              pEcbInfo->bDeadlock =
FALSE;
                              return TRUE;
                    }
          }
          return FALSE;
}

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

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

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

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

          return;
}

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


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

          pEcbInfo = NULL;

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

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

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

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

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

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

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

                    fclose(fp);
          }

          return INT_CANCEL;
}


/* FUNCTION: int msg_handler(DBPROCESS *dbproc, DBINT
msgno, int msgstate, int severity, char *msgtext)
 *
 * PURPOSE:         This function handles DB-Library
SQL Server error messages
 *
 * ARGUMENTS:       DBPROCESS           *dbproc
                    DBPROCESS id pointer
 *                                  DBINT
                    msgno
          message number
 *                                  int
                              msgstate
          message state
 *                                  int
                              severity
          message severity
 *                                  char
                    *msgtext            printable
message description
 *
 * RETURNS:                 int
                    INT_CONTINUE        continue if
error is SQLETIME else INT_CANCEL action
 *
                                  INT_CANCEL
          cancel operation
 *
 * COMMENTS:        This function also sets the dead
lock dbproc variable if necessary.
 *
 */
```

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

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


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

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

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

                    if ( pEcbInfo )
                              pEcbInfo->bFailed = TRUE;

                    bError = TRUE;

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

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

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

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

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

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

          strcpy(szServer, argv[i]+2);

          break;
                                        case 'V':
                                        case 'v':

          verbose = TRUE;

          break;
                                        case '?':

          return TRUE;
                              }
                    }
          }
          return FALSE;
}

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

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

}
```

Appendix B – Database Design

Build

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_ol1_dev     = 30000,
                  c_ol2_dev     = 30000,
                  c_grow_dev    = 10000,
                  c_wdi_dev     =   100,
                  c_cust1_dev   =  6470,
                  c_cust2_dev   =  6470,
                  c_cust3_dev   =  8000,
                  c_stock1_dev  = 10060,
                  c_stock2_dev  = 10060,
                  c_stock3_dev  = 12000
                  log on c_log_dev =  30000
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_log_dev",
          physname = "D:",
          vdevno   = 14,
          size     = 15728640
go

disk init name = "c_ol1_dev",
          physname = "E:",
          vdevno   = 15,
          size     = 15728640
go

disk init name = "c_ol2_dev",
          physname = "F:",
          vdevno   = 16,
          size     = 15728640
go

disk init name = "c_grow_dev",
          physname = "G:",
          vdevno   = 17,
          size     = 5242880
go

disk init name = "c_wdi_dev",
          physname = "H:",
          vdevno   = 18,
          size     = 51200
go

disk init name = "c_cust1_dev",
          physname = "I:",
          vdevno   = 19,
          size     = 4194304
go

disk init name = "c_cust2_dev",
          physname = "J:",
          vdevno   = 20,
          size     = 4194304
go

disk init name = "c_cust3_dev",
          physname = "K:",
          vdevno   = 21,
          size     = 4194304
go

disk init name = "c_stock1_dev",
          physname = "L:",
          vdevno   = 22,
          size     = 6291456
go

disk init name = "c_stock2_dev",
          physname = "M:",
          vdevno   = 23,
          size     = 6291456
go

disk init name = "c_stock3_dev",
          physname = "N:",
          vdevno   = 24,
          size     = 6291456
go
```

Segment.sql

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


use tpcc
go

exec sp_addsegment ol_seg, c_ol1_dev
exec sp_extendsegment ol_seg, c_ol2_dev
exec sp_addsegment grow_seg, c_grow_dev
exec sp_addsegment wdi_seg, c_wdi_dev
exec sp_addsegment cust_seg, c_cust1_dev
exec sp_extendsegment cust_seg, c_cust2_dev
exec sp_extendsegment cust_seg, c_cust3_dev
exec sp_addsegment stock_seg, c_stock1_dev
exec sp_extendsegment stock_seg, c_stock2_dev
exec sp_extendsegment stock_seg, c_stock3_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 wdi_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 wdi_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 cust_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 grow_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 grow_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 grow_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 wdi_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 stock_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 cust_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 cust_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 wdi_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 wdi_seg
go
select getdate()
go
```

ldxnodcl.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 grow_seg
go
select getdate()
go
```

ldxodlcl.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
```

ldxordcl.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 grow_seg
go
select getdate()
go
```

ldxstkcl.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 stock_seg
go
select  getdate()
go
```

ldxwarcl.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 wdi_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 Procedures

Neword.sql

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


use tpcc
go

/* new-order transaction stored procedure */

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

/*  Modified by rick vicik, 2/4/97  */
/*  Combined initialization of local variables into
district update statement  */
/*  Combined 3 huge case select statements into a
single one  */
```

```
create proc tpcc_neworder

                    @w_id           smallint,

                    @d_id           tinyint,

                    @c_id           int,

                    @o_ol_cnt       tinyint,

                    @o_all_local    tinyint,

                    @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   */
                                         "Dec 31,
1889",      /* constant             */
                                         @li_qty,
/* lineitem qty         */
                                         @i_price *
@li_qty, /* ol_amount            */
                                         @s_dist)
/* from stock           */


           /* send line-item data to client */

                      select @i_name,
                  @s_quantity,
                  b_g = case when (
(patindex("%ORIGINAL%",@i_data) > 0) and

(patindex("%ORIGINAL%",@s_data) > 0) )
                       then "B" else "G" end,
                  @i_price,
                  @i_price * @li_qty

       end
       else
                   begin

                       /*  no item found - triggers
rollback condition */

                            select "",0,"",0,0
                            select @commit_flag = 0

                   end
         end
```

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

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

    /* insert fresh row into orders table */

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

@o_entry_d,

0,

@o_ol_cnt,

@o_all_local)

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

          insert into new_order values (@o_id,

                                @d_id,

                                @w_id)

          /*  select warehouse tax */

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

          if (@commit_flag = 1)

                    commit transaction n
          else
                    /* all that work for nuthin!!! */
                    rollback transaction n

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

end

go
```

Payment.sql

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


use tpcc
go
```

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

create proc tpcc_payment @w_id           smallint,

           @c_w_id         smallint,

           @h_amount       numeric(6,2),

           @d_id           tinyint,

           @c_d_id         tinyint,

           @c_id           int,

           @c_last         char(16) = ""




as
declare  @w_street_1     char(20),
         @w_street_2     char(20),
         @w_city         char(20),
         @w_state        char(2),
         @w_zip          char(9),
         @w_name         char(10),
         @d_street_1     char(20),
         @d_street_2     char(20),
         @d_city         char(20),
         @d_state        char(2),
         @d_zip          char(9),
         @d_name         char(10),
         @c_first        char(16),
         @c_middle       char(2),
         @c_street_1     char(20),
         @c_street_2     char(20),
         @c_city         char(20),
         @c_state        char(2),
         @c_zip          char(9),
         @c_phone        char(16),
         @c_since        datetime,
         @c_credit       char(2),
         @c_credit_lim   numeric(12,2),
         @c_balance      numeric(12,2),
         @c_discount     numeric(4,4),
         @data1          char(250),
         @data2          char(250),
         @c_data_1       char(250),
         @c_data_2       char(250),
         @datetime       datetime,
         @w_ytd          numeric(12,2),
         @d_ytd          numeric(12,2),
         @cnt            smallint,
         @val            smallint,
         @screen_data          char(200),
                     @d_id_local         tinyint,
                     @w_id_local         smallint,
                     @c_id_local         int

select @screen_data = ""

begin tran p

          /* get payment date */

          select @datetime = getdate()

          if (@c_id = 0)
          begin
                    /* get customer id and info using
last name */

                    select @cnt = count(*)
                    from customer holdlock
                    where c_last = @c_last and
                          c_w_id = @c_w_id and
                          c_d_id = @c_d_id


                    select @val = (@cnt + 1) / 2
                    set rowcount @val

                    select @c_id = c_id
                    from customer holdlock
                    where c_last = @c_last and
                          c_w_id = @c_w_id and
                          c_d_id = @c_d_id
                    order by c_w_id, c_d_id, c_last,
c_first
                    set rowcount 0
          end

          /* get customer info and update balances */

          update customer set
                    @c_balance     = c_balance =
c_balance - @h_amount,
                    c_payment_cnt  = c_payment_cnt + 1,
                    c_ytd_payment  = c_ytd_payment +
@h_amount,
                    @c_first       = c_first,
                    @c_middle      = c_middle,
               @c_last        = c_last,
               @c_street_1    = c_street_1,
                    @c_street_2    = c_street_2,
                    @c_city        = c_city,
                    @c_state       = c_state,
                    @c_zip         = c_zip,
                    @c_phone       = c_phone,
                    @c_credit      = c_credit,
                    @c_credit_lim  = c_credit_lim,
                    @c_discount    = c_discount,
                    @c_since       = c_since,
                    @data1         = c_data_1,
                    @data2         = c_data_2,
                    @c_id_local    = c_id
          where c_id   = @c_id and
                c_w_id = @c_w_id and
                c_d_id = @c_d_id

          /* if customer has bad credit get some more
info */

          if (@c_credit = "BC")
          begin

                    /* compute new info */

                    select @c_data_2 =
substring(@data1,209,42) +

               substring(@data2, 1, 208)
                    select @c_data_1 =
convert(char(5),@c_id) +

               convert(char(4),@c_d_id) +

convert(char(5),@c_w_id) +

convert(char(4),@d_id) +

convert(char(5),@w_id) +

convert(char(19),@h_amount) +

substring(@data1, 1, 208)

                    /* update customer info */

                    update customer set
                           c_data_1 = @c_data_1,
                                  c_data_2 = @c_data_2
                    where c_id   = @c_id and
                                c_w_id = @c_w_id and
                          c_d_id = @c_d_id

                    select @screen_data = substring
(@c_data_1,1,200)
          end



          /* get district data and update year-to-date
*/

          update district
                    set d_ytd       = d_ytd +
@h_amount,
                              @d_street_1 = d_street_1,
                              @d_street_2 = d_street_2,
                              @d_city     = d_city,
                              @d_state    = d_state,
                              @d_zip      = d_zip,
                              @d_name     = d_name,
                              @d_id_local = d_id
                    where d_w_id = @w_id and
                          d_id   = @d_id

          /* get warehouse data and update year-to-date
*/
```

```
        update warehouse
        set w_ytd       = w_ytd + @h_amount,
                @w_street_1 = w_street_1,
            @w_street_2 = w_street_2,
            @w_city     = w_city,
            @w_state    = w_state,
            @w_zip      = w_zip,
            @w_name     = w_name,
                @w_id_local = w_id
        where w_id = @w_id

        /* create history record */

        insert into history values  (@c_id_local,

                        @c_d_id,

                              @c_w_id,

                              @d_id_local,

                              @w_id_local,

                              @datetime,

                              @h_amount,

                              @w_name + "    " +
@d_name)

commit tran p

/* return data to client */

select  @c_id,
                    @c_last,
                    @datetime,
                    @w_street_1,
                    @w_street_2,
                    @w_city,
                    @w_state,
                    @w_zip,
                    @d_street_1,
                    @d_street_2,
                    @d_city,
                    @d_state,
                    @d_zip,
                    @c_first,
                    @c_middle,
                    @c_street_1,
                    @c_street_2,
                    @c_city,
                    @c_state,
                    @c_zip,
                    @c_phone,
                    @c_since,
                    @c_credit,
                    @c_credit_lim,
                    @c_discount,
                    @c_balance,
                    @screen_data

go
```

Delivery.sql

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


use tpcc
go

/* delivery transaction */

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

create proc tpcc_delivery                          @w_id
smallint,


                                @o_carrier_id   smallint
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
```

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
```

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
```

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_comple
ted);

}


//=====================================================
==================
//
// 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_comple
ted);

}


//=====================================================
=======-===============
//
// 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_del
ivery_d);
                              }
                              else
                              {

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

          strcpy(orders_buf[o_id].o_ol[ol].ol_delivery_
d,"Dec 31, 1889");
                              }
                    }
          }
}


//=====================================================
==================
//
// Function   : LoadOrdersTable
//
//=====================================================
==================

void LoadOrdersTable(LOADER_TIME_STRUCT
*orders_time_start)
{
          int         i;
    long        o_id;
          short     o_d_id;
          short     o_w_id;
    long        o_c_id;
    short       o_carrier_id;
    short       o_ol_cnt;
    short       o_all_local;
          char                o_entry_d[50];

          // bind ORDER data
    bcp_bind(o_dbproc1, (BYTE *) &o_id,          0, -1,
NULL, 0, 0, 1);
    bcp_bind(o_dbproc1, (BYTE *) &o_d_id,        0, -1,
NULL, 0, 0, 2);
    bcp_bind(o_dbproc1, (BYTE *) &o_w_id,        0, -1,
NULL, 0, 0, 3);
    bcp_bind(o_dbproc1, (BYTE *) &o_c_id,        0, -1,
NULL, 0, 0, 4);
    bcp_bind(o_dbproc1, (BYTE *) o_entry_d,      0, 50,
NULL, 0, SQLCHAR, 5);
    bcp_bind(o_dbproc1, (BYTE *) &o_carrier_id,  0, -1,
NULL, 0, 0, 6);
    bcp_bind(o_dbproc1, (BYTE *) &o_ol_cnt,      0, -1,
NULL, 0, 0, 7);
    bcp_bind(o_dbproc1, (BYTE *) &o_all_local,   0, -1,
NULL, 0, 0, 8);

    for (i = 0; i < ORDERS_PER_DISTRICT; i++)
    {
                    o_id         = orders_buf[i].o_id;
                    o_d_id       =
orders_buf[i].o_d_id;
                    o_w_id       =
orders_buf[i].o_w_id;
                    o_c_id       =
orders_buf[i].o_c_id;
                    o_carrier_id =
orders_buf[i].o_carrier_id;
                    o_ol_cnt     =
orders_buf[i].o_ol_cnt;
                    o_all_local  =
orders_buf[i].o_all_local;
                    CurrentDate(&o_entry_d);

                    // send data to server
                    if (!bcp_sendrow(o_dbproc1))
                    printf("Error, LoadOrdersTable()
failed calling bcp_sendrow().  Check error file.\n");
                    orders_rows_loaded++;
                    // CheckForCommit(o_dbproc1,
orders_rows_loaded, "ORDERS", &orders_time_start-
>time_start);
    }

          bcp_batch(o_dbproc1);

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

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

          InterlockedIncrement(&order_threads_completed
);

}
```

```
//=====================================================
==================
//
// Function   : LoadNewOrderTable
//
//=====================================================
==================

void LoadNewOrderTable(LOADER_TIME_STRUCT
*new_order_time_start)
{
    int         i;
    long        o_id;
    short       o_d_id;
    short       o_w_id;

          // Bind NEW-ORDER data
    bcp_bind(o_dbproc2, (BYTE *) &o_id,          0, -1,
NULL, 0, 0, 1);
    bcp_bind(o_dbproc2, (BYTE *) &o_d_id,        0, -1,
NULL, 0, 0, 2);
    bcp_bind(o_dbproc2, (BYTE *) &o_w_id,        0, -1,
NULL, 0, 0, 3);

          for (i = 2100; i < 3000; i++)
          {
                    o_id    = orders_buf[i].o_id;
                    o_d_id  = orders_buf[i].o_d_id;
                    o_w_id  = orders_buf[i].o_w_id;

                    if (!bcp_sendrow(o_dbproc2))
                    printf("Error, LoadNewOrderTable()
failed calling bcp_sendrow().  Check error file.\n");
                    new_order_rows_loaded++;
                    // CheckForCommit(o_dbproc2,
new_order_rows_loaded, "NEW_ORDER",
&new_order_time_start->time_start);
          }

          bcp_batch(o_dbproc2);

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

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

          InterlockedIncrement(&order_threads_completed
);

}


//=====================================================
==================
//
// Function   : LoadOrderLineTable
//
//=====================================================
==================

void LoadOrderLineTable(LOADER_TIME_STRUCT
*order_line_time_start)
{
          int         i,j;
    long        o_id;
          short     o_d_id;
          short     o_w_id;
    long        ol;
          long      ol_i_id;
    short       ol_supply_w_id;
    short       ol_quantity;
    double      ol_amount;
    short       o_all_local;
    char        ol_dist_info[DIST_INFO_LEN+1];
          char                ol_delivery_d[50];

          // bind ORDER-LINE data
    bcp_bind(o_dbproc3, (BYTE *) &o_id,
0, -1, NULL, 0, 0, 1);
    bcp_bind(o_dbproc3, (BYTE *) &o_d_id,
0, -1, NULL, 0, 0, 2);
    bcp_bind(o_dbproc3, (BYTE *) &o_w_id,
0, -1, NULL, 0, 0, 3);
    bcp_bind(o_dbproc3, (BYTE *) &ol,
0, -1, NULL, 0, 0, 4);
    bcp_bind(o_dbproc3, (BYTE *) &ol_i_id,
0, -1, NULL, 0, 0, 5);
    bcp_bind(o_dbproc3, (BYTE *) &ol_supply_w_id,
0, -1, NULL, 0, 0, 6);
          bcp_bind(o_dbproc3, (BYTE *) ol_delivery_d,
                    0, 50, NULL, 0, SQLCHAR,
7);
    bcp_bind(o_dbproc3, (BYTE *) &ol_quantity,
0, -1, NULL, 0, 0, 8);
    bcp_bind(o_dbproc3, (BYTE *) &ol_amount,
0, -1, NULL, 0, SQLFLT8, 9);
    bcp_bind(o_dbproc3, (BYTE *) ol_dist_info,
0, DIST_INFO_LEN, NULL, 0, 0, 10);

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

                    for (j=0; j <
orders_buf[i].o_ol_cnt; j++)
                    {
                              ol             =
orders_buf[i].o_ol[j].ol;
                              ol_i_id        =
orders_buf[i].o_ol[j].ol_i_id;
                              ol_supply_w_id =
orders_buf[i].o_ol[j].ol_supply_w_id;
                              ol_quantity    =
orders_buf[i].o_ol[j].ol_quantity;
                              ol_amount      =
orders_buf[i].o_ol[j].ol_amount;
                              //  Changed to insure
ol_delivery_d set properly (now set in OrdersBufLoad)
                              //
CurrentDate(&ol_delivery_d);

          strcpy(ol_delivery_d,orders_buf[i].o_ol[j].ol
_delivery_d);


          strcpy(ol_dist_info,orders_buf[i].o_ol[j].ol_
dist_info);

                              if
(!bcp_sendrow(o_dbproc3))
                                        printf("Error,
LoadOrderLineTable() failed calling bcp_sendrow().
Check error file.\n");
                              order_line_rows_loaded++;
                              //
CheckForCommit(o_dbproc3, order_line_rows_loaded,
"ORDER_LINE", &order_line_time_start->time_start);
                    }

          }

          bcp_batch(o_dbproc3);

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

Tpcc.h

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

Appendix B-Database Design

```
 *
 *        PURPOSE:  Header file for Microsoft TPC-C
Benchmark Kit
 *        Author:             Damien Lindauer
 *
          damienl@Microsoft.com
 */

// Build number of TPC Benchmark Kit
#define TPCKIT_VER  "3.00.02"

// General headers
#include <windows.h>
#include <winbase.h>
#include <stdlib.h>
#include <stdio.h>
#include <process.h>
#include <stddef.h>
#include <stdarg.h>
#include <string.h>
#include <signal.h>
#include <time.h>
#include <timeb.h>
#include <types.h>
#include <wincon.h>

#ifdef USE_ODBC
// ODBC headers
#include <sql.h>
#include <sqlext.h>
HENV                                          henv;
#endif

// DB-Library headers
#include <sqlfront.h>
#include <sqldb.h>

#include "trans.h"                            //pgd
5-6-96 split transaction structs definations into own
header

                              //for tpcform.c i.e.
telnet application

// Critical section declarations
CRITICAL_SECTION              ConsoleCritSec;
CRITICAL_SECTION              QueuedDeliveryCritSec;
CRITICAL_SECTION              WriteDeliveryCritSec;
CRITICAL_SECTION
          DroppedConnectionsCritSec;
CRITICAL_SECTION              ClientErrorLogCritSec;

// General constants
#define SQLCONN DBPROCESS
#define DUMB_MESSAGE                      5701
#define ABORT_ERROR                       6104
#define INVALID_ITEM_ID                          0
#define MILLI                             1000
#define MAX_THREADS                       2510
#define STATS_MSG_LOW                             3600
#define STATS_MSG_HIGH                            3700
#define SHOWPLAN_MSG_LOW                  6200
#define SHOWPLAN_MSG_HIGH                 6300
#define FALSE                                    0
#define TRUE                                     1
#define UNDEF
             -1
#define MINPRINTASCII                               32
#define MAXPRINTASCII                               126

// Default environment constants
#define SERVER                                   ""
#define DATABASE
          "tpcc"
#define USER                                     "sa"
#define PASSWORD                                   ""
#define SYNCH_SERVERNAME                         ""

// Statistic constants
#define INTERVAL                           20    //
Total interval of buckets, in sec
#define UNIT                               .1    //
Time period of each bucket
#define HIST_MAX                          200    // Num
of histogram buckets = INTERVAL/UNIT
#define BUCKET                            100    //
Division factor for response time

// Default master arguments
#define ADMIN_DATABASE
          "tpcc_admin"
#define RAMP_UP                 600
#define STEADY_STATE           1200
#define RAMP_DOWN               120
#define NUM_USERS                10
#define NUM_WAREHOUSES                    1
#define THINK_TIMES
          0
#define DISPLAY_DATA                      0
#define DEFMSPACKSIZE                      4096
#define TRANSACTION                       0
#define CLIENT_MODE                       1
#define DEF_WW_T                                        120
#define DEF_WW_a
          1
#define DEADLOCK_RETRY                                 4
#define DELIVERY_BACKOFF                               2
#define DELIVERY_MODE
          0
#define NEWORDER_MODE
          0
#define DEF_LOAD_MULTIPLIER     1.0
#define DEF_CHECKPOINT_INTERVAL   960
#define DEF_FIRST_CHECKPOINT    240
#define DISABLE_90TH
          0
#define   RESFILENAME
                    "results.txt"
#define SQLSTAT_FILENAME
          "sqlstats.txt"
#define ENABLE_SQLSTAT
          0
#define SQLSTAT_PERIOD                                  100
#define SHUTDOWN_SERVER
          0
#define AUTO_RUN
          0
#define DISABLE_SQLPERF
          0

// Default client arguments
#define NUM_THREADS                          10
#define X_FLAG                        0
#define Y_FLAG                        1
#define NUM_DELIVERIES
          2
#define CLIENT_NURAND                           223
#define DISABLE_DELIVERY_RESFILES   1
#define ENABLE_QJ
          0

// Globals for queued delivery handling
typedef struct      delivery_node *DELIVERY_PTR;
DELIVERY_PTR        delivery_head, delivery_tail;
short                         queued_delivery_cnt;
HANDLE                        hDeliveryMonPipe;
struct delivery_node
{
          short                                       w_id;
          short
          o_carrier_id;
          SYSTEMTIME
          queue_time;
          long
          tran_start_time;
          struct
          delivery_node *next_delivery;
};

// Default loader arguments
#define BATCH                             10000
#define DEFLDPACKSIZE                                    4096
#define ORDERS_PER_DIST                    3000
#define LOADER_RES_FILE     "load.out"
#define LOADER_NURAND_C                                  123
#define DEF_STARTING_WAREHOUSE                         1
#define BUILD_INDEX
          1
#define INDEX_SCRIPT_PATH     "scripts"

// Transaction types
#define EMPTY                             0
#define NEW_ORDER_TRAN                    1
#define PAYMENT_TRAN                      2
#define ORDER_STATUS_TRAN                 3
#define DELIVERY_TRAN                     4
#define STOCK_LEVEL_TRAN                  5

// Statistic structures
typedef struct
{
    long
          tran_count;
    long
          total_time;
```

```
    long
          resp_time;
    long
          resp_min;
    long
          resp_max;
    long
          rolled_back;
    long
          tran_2sec;
    long
          tran_5sec;
    long
          tran_sqr;
          long
          num_deadlocks;
    long
          resp_hist[HIST_MAX];
} TRAN_STATS;

typedef struct
{
    TRAN_STATS                          NewOrderStats;
    TRAN_STATS                          PaymentStats;
    TRAN_STATS
          OrderStatusStats;
    TRAN_STATS
          QueuedDeliveryStats;
          TRAN_STATS
          DeliveryStats;
    TRAN_STATS
          StockLevelStats;
} CLIENT_STATS;

// driver structures
typedef struct
{
    char
          *server;
    char
          *database;
    char
          *user;
    char
          *password;
    char
          *table;
    long
          num_warehouses;
    long
          batch;
    long
          verbose;
          long
          pack_size;
          char
          *loader_res_file;
          char
          *synch_servername;
          long
          case_sensitivity;
          long
          starting_warehouse;
          long
          build_index;
          char
          *index_script_path;
} TPCCLDR_ARGS;

typedef struct
{
    char
          *server;
    char
          *user;
    char
          *password;
          char
          *admin_database;
    char
          *sqlstat_filename;
          long
          run_id;
} SQLSTAT_ARGS;

typedef struct
{
    SQLCONN                             *sqlconn;
    char                                *server;
    char                                *database;
          char
          *admin_database;
    char                                *user;
    char                                *password;
    long                                ramp_up;
    long                                steady_state;
    long                                ramp_down;
    long                                num_users;
    long                                num_warehouses;
    long                                think_times;
    long                                display_data;
          long                          client_mode;

          long                          tran;
          long
          deadlock_retry;
          long
          delivery_backoff;
          long
          num_deliveries;
          char
          *comment;
          double
          load_multiplier;
          long
          checkpoint_interval;
          long
          first_checkpoint;
          long
          disable_90th;
          char
          *resfilename;
          char
          *sqlstat_filename;
          long
          enable_sqlstat;
          long
          sqlstat_period;
          long
          shutdown_server;
          long
          auto_run;
          long
          dropped_connections;
    short                               spid;
          long
          disable_sqlperf;
} MASTER_DATA;

typedef struct
{
          long
          num_threads;
    char                                *server;
    char                                *database;
          char
          *admin_database;
    char                                *user;
    char                                *password;

          long
          pack_size;
          short
          x_flag;
          char
          *synch_servername;
          long
          disable_delivery_resfiles;
          long
          enable_qj;
#ifdef USE_CONMON
          HANDLE
          hConMon;
          short
          con_id;
          short
          con_x;
          short
          con_y;
#endif
} GLOBAL_CLIENT_DATA;

typedef struct
{
#ifdef USE_ODBC
          HDBC                                          hdbc;
          HSTMT
          hstmt;
#else
    SQLCONN                             *sqlconn;
#endif
          short
          threadid;
    char                                *server;
    char                                *database;
```



```
        char
        *admin_database;
    char                            *user;
    char                            *password;

        long
        ramp_up;
        long
        steady_state;
    long                            ramp_down;
    long                            num_warehouses;
        long                        client_mode;
        long                        tran;
        long
        deadlock_retry;
        long
        think_times;
        long
        pack_size;
    long
        tran_start_time;
    long                            tran_end_time;
    long                            display_data;
        long                        id;

        short                               w_id;
    short                           spid;
        long
        disable_90th;
        double
        load_multiplier;
        long
        num_deliveries;
        long
        enable_qj;
#ifdef USE_CONMON
        HANDLE
        hConMon;
        short
        con_id;
        short
        con_x;
        short
        con_y;
        short
        fTimerStat;
#endif
} CLIENT_DATA;

typedef struct
{
#ifdef USE_ODBC
        HDBC                                hdbc;
        HSTMT
        hstmt;
#else
        SQLCONN
        *sqlconn;
#endif
        SYSTEMTIME
        queue_time;
        SYSTEMTIME
        completion_time;
    long
        tran_start_time;
    long                            tran_end_time;
        short
        threadid;
        FILE
        *fDelivery;
    short                           spid;
    short                           w_id;
        short                               d_id;
        short
        o_carrier_id;
        DEL_ITEM
        DelItems[10];
        char
        *server;
    char                            *database;
        char
        *admin_database;
    char                            *user;
    char                            *password;

        long
        ramp_up;
    long                            steady_state;
    long                            ramp_down;

        long
        pack_size;
        long                        id;
        long
        disable_90th;
        long
        delivery_backoff;
        long
        disable_delivery_resfiles;
        long
        enable_qj;
} DELIVERY;

typedef struct
{
        long
        pipe_num;
} DELIVERY_ARGS;

// For client synchronization
#define LINE_LEN   80
#define NAME_SIZE  25
#define IN_BUF_SIZE        1000
#define OUT_BUF_SIZE       1000
#define TIME_OUT           0
#define PLEASE_READ        1000
#define PLEASE_WRITE       1000

typedef struct _WRTHANDLE
{       HANDLE              hPipe;
    DWORD           threadID;
        CHAR                Name[NAME_SIZE];
        struct              _WRTHANDLE *        next;
}WRTHANDLE;

// For client console monitor
#ifdef USE_CONMON
#define   CON_LINE_SIZE                 40
#define   DEADLOCK_X                                17
#define DEADLOCK_Y                      4
#define CUR_STATE_X                     15
#define CUR_STATE_Y                     3
#define   YELLOW                                    0
#define   RED
          1
#define   GREEN                                     2
int                 total_deadlocks;
#endif

// Functions in random.c
void      seed();
long      irand();
double    drand();
void      WUCreate();
short     WURand();

// Functions in getargs.c;
void      GetArgsLoader();
void      GetArgsLoaderUsage();
void      GetArgsMaster();
void      GetArgsMasterUsage();
void      GetArgsClient();
void      GetArgsClientUsage();
void      GetArgsDelivery();
void      GetArgsDeliveryUsage();
void      GetArgsSQLStat();
void      GetArgsSQLStatUsage();

// Functions in master.c
void      ReadClientDone();
BOOL    CtrlHandler();

// Functions in client.c
void      ClientMain();
void      DeliveryMain();
void      Delivery();
void      ClientEmulate();
short     ClientSelectTransaction();
void      ClientShuffleDeck();

//Functions in tran.c
BOOL      TranNewOrder();
BOOL      TranPayment();
BOOL      TranOrderStatus();
BOOL      TranDelivery();
BOOL      TranStockLevel();

// Functions in data.c
void      DataNewOrder();
void      DataPayment();
void      DataOrderStatus();
void      DataDelivery();
void      DataStockLevel();
short     DataRemoteWarehouse();

// Functions in time.c
```

```
long      TimeNow();
void      TimeInit();
void      TimeKeying();
void      TimeThink();

// Functions in stats.c
void      StatsInit();
void      StatsInitTran();
void      StatsGeneral();
void      StatsDelivery();

// Functions in sqlfuncs.c
BOOL      SQLExec();
BOOL      SQLExecCmd();
BOOL      SQLOpenConnection();
void      SQLClientInit();
int       SQLMasterInit();
void      SQLDeliveryInit();
int       SQLClientStats();
int       SQLDeliveryStats();
void      SQLTranStats();
void      SQLMasterStats();
void      SQLMasterTranStats();
void      SQLIOStats();
void      SQLCheckpointStats();
void      SQLInitResFile();
void      SQLGetRunId();
BOOL      SQLNewOrder();
BOOL      SQLPayment();
BOOL      SQLOrderStatus();
BOOL      SQLStockLevel();
void    SQLDelivery();
int       SQLGetCustId();
void      SQLExit();
void      SQLInit();
void      SQLInitPrivate();
void      SQLClientInitPrivate();
void      SQLDeliveryInitPrivate();
int       SQLMsgHandler();
int       SQLErrHandler();
int       SQLClientMsgHandler();
int       SQLClientErrHandler();
int       SQLDeliveryMsgHandler();
int       SQLDeliveryErrHandler();
void      SQLInitDate();
void      SQLShutdown();
#ifdef USE_ODBC
void      ODBCOpenConnection();
void      ODBCOpenDeliveryConnection();
BOOL      ODBCError();
void      ODBCExit();
#endif

// Functions in util.c
void      UtilSleep();
void      UtilPrintNewOrder();
void      UtilPrintPayment();
void      UtilPrintOrderStatus();
void      UtilPrintDelivery();
void      UtilPrintStockLevel();
void      UtilPrintOlTable();
void      UtilError();
void      UtilFatalError();
void      UtilStrCpy();
#ifdef USE_CONMON
void      WriteConsoleString();
#endif
void      WriteDeliveryString();
BOOL      AddDeliveryQueueNode();
BOOL      GetDeliveryQueueNode();


// Functions in strings.c
void      MakeAddress();
void      LastName();
int       MakeAlphaString();
int       MakeOriginalAlphaString();
int       MakeNumberString();
int       MakeZipNumberString();
void      InitString();
void      InitAddress();
void      PaddString();

// Functions in delivery.c
void      DeliveryHMain();
void      DeliveryH();
```

Trans.h

```
//FILE: TRANS.H
// TPC-C Benchmark Kit
//
// Module:  TRANS.H
// Author:  PhilipDu, from tpcc.h, Author DamienL
//
//                              Copyright Microsoft inc.
1996, All Rights Reserved


#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
```

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[] =
"0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqr
stuvwxyz";
        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
```

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;
}
```


Appendix C – Tunable Parameters

Microsoft Windows NT Server version 4.0 Tunable Parameters

No Windows NT registry parameters were modified for this benchmark.

Microsoft SQL Server version 6.5 Startup Parameters

Microsoft SQL Server was started with the following command line options

sqlservr -c -x -t1081 -t3502 –t812 –t1140 –Cd1440000 –Cp5000

where

-c	Start 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.
-t812	Disables checkpoint buffer sorting
-t1140	Optimizes free space allocation
-t3502	Prints a message to the log at the beginning and end of each checkpoint.
-Cd1440000	Defines number of 2K pages for data cache buffers
-Cp5000	Defines number of 2K pages for procedure cache buffers

Cache Column of Sysobjects Table

The following updates were made to the sysobjects table:
use tpcc
go
update sysobjects set cache=2 where name=”stock”
go
update sysobjects set cache=5 where name=”customer”
go

Microsoft SQL Server 6.5 Configuration Parameters

```
1> 2>  name                                 minimum     maximum     config_value
       run_value
 ---------------------------------- ----------- ----------- ------------
       -----------
 affinity mask                              0  2147483647           15
              15
 allow updates                              0           1            0
               0
 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         5000
```


Appendix C – Tunable Parameters

```
             5000
hash buckets                            4999      1000003      1000003
          1000003
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         2500
             2500
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          150
              150
media retention                            0          365            0
                0
memory                                  2800      1572864       950000
           950000
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            1
                1
```

```
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          250
           250
user options                                   0        4095            0
                0
```

Disk Array Configuration Parameters

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

************************************************************************

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

     3 Channel - 15 Target  DAC960PG   #1    Firmware version 4.00


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

     Number of Packs = 2

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

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

     Number of System Drives = 2

     Sys Drv #   Phy. Size   Raid Level   Eff. Size   Write Policy
     =========   =========   ==========   =========   ============
         0        26049  MB       0        26049  MB   Write Thru
         1        69464  MB       6        34732  MB   Write Thru

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

************************************************************************

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



```
     3 Channel - 15 Target  DAC960PG   #2    Firmware version 4.00


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

     Number of Packs = 3

     Pack  0 : [0:0]  [0:1]  [0:2]  [0:3]  [0:4]  [0:5]  [0:6]  [0:8]
     Pack  1 : [1:0]  [1:1]  [1:2]  [1:3]  [1:4]  [1:5]  [1:6]  [1:8]
     Pack  2 : [2:0]  [2:1]  [2:2]  [2:3]  [2:4]  [2:5]  [2:6]  [2:8]

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

     Number of System Drives = 1

     Sys Drv #   Phy. Size   Raid Level   Eff. Size   Write Policy
     =========   =========   ==========   =========   ============
         0        104160  MB       0        104160  MB   Write Thru



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

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

     3 Channel - 15 Target  DAC960PG   #3    Firmware version 4.00


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

     Number of Packs = 3

     Pack  0 : [0:0]  [0:1]  [0:2]  [0:3]  [0:4]  [0:5]  [0:6]  [0:8]
     Pack  1 : [1:0]  [1:1]  [1:2]  [1:3]  [1:4]  [1:5]  [1:6]  [1:8]
     Pack  2 : [2:0]  [2:1]  [2:2]  [2:3]  [2:4]  [2:5]  [2:6]  [2:8]

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

     Number of System Drives = 1

     Sys Drv #   Phy. Size   Raid Level   Eff. Size   Write Policy
     =========   =========   ==========   =========   ============
         0        104160  MB       0        104160  MB   Write Thru


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

```
     *                              Version 4.71
*

************************************************************************

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

     3 Channel - 15 Target  DAC960PG   #4    Firmware version 4.00


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

     Number of Packs = 3

     Pack  0 : [0:0]  [0:1]  [0:2]  [0:3]  [0:4]  [0:5]  [0:6]  [0:8]
     Pack  1 : [1:0]  [1:1]  [1:2]  [1:3]  [1:4]  [1:5]  [1:6]  [1:8]
     Pack  2 : [2:0]  [2:1]  [2:2]  [2:3]  [2:4]  [2:5]  [2:6]  [2:8]

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

     Number of System Drives = 1

     Sys Drv #   Phy. Size   Raid Level   Eff. Size   Write Policy
     =========   =========   ==========   =========   ============
         0        104160  MB       0        104160  MB   Write Thru


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

************************************************************************

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

     3 Channel - 15 Target  DAC960PG   #5    Firmware version 4.00


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

     Number of Packs = 3

     Pack  0 : [0:0]  [0:1]  [0:2]  [0:3]  [0:4]  [0:5]  [0:6]
     Pack  1 : [1:0]  [1:1]  [1:2]  [1:3]  [1:4]  [1:5]  [1:6]
     Pack  2 : [2:0]  [2:1]  [2:2]  [2:3]  [2:4]  [2:5]  [2:6]

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

     Number of System Drives = 1

     Sys Drv #   Phy. Size   Raid Level   Eff. Size   Write Policy
     =========   =========   ==========   =========   ============
```

```
         0        182343  MB       0        182343  MB   Write Thru
```

Server Hardware Configuration

```
Microsoft Diagnostics Report For
\\X3
------------------------------
------------------------------
----

OS Version Report
------------------------------
------------------------------
----
Microsoft (R) Windows NT (TM)
Server
Version 4.0 (Build 1381: Service
Pack 3) x86 Multiprocessor Free
Registered Owner: NSBU, Acer Inc.
Product Number: 70238-111-
1111111-49231
------------------------------
------------------------------
----

System Report
------------------------------
------------------------------
----
System: AT/AT COMPATIBLE
Hardware Abstraction Layer: MPS
1.4 - APIC platform
BIOS Date: 12/11/97
BIOS Version: <unavailable>

Processor list:
   0:  x86 Family 6 Model 1
Stepping 9 GenuineIntel ~199 Mhz
   1:  x86 Family 6 Model 1
Stepping 9 GenuineIntel ~199 Mhz
   2:  x86 Family 6 Model 1
Stepping 9 GenuineIntel ~199 Mhz
   3:  x86 Family 6 Model 1
Stepping 9 GenuineIntel ~199 Mhz
------------------------------
------------------------------
----

Video Display Report
------------------------------
------------------------------
----
BIOS Date: <unavailable>
Adapter:
   Setting: 1024 x 768 x 256
            60 Hz
   Type: ati compatible display
adapter
   String: ATI Graphics
Accelerator
   Memory: 2 MB
   Chip Type: Mach 64 VT-A
   DAC Type: DAC built into ASIC
Driver:
   Vendor: ATI Technologies Inc.
   File(s): ati.sys, ati.dll,
8514a.dll
   Version: 3.1.77, 4.0.0


Drives Report
------------------------------
------------------------------
----
C:\  (Local - NTFS)  Total: 0KB,
Free: 0KB
O:\  (Local - NTFS)  Total:
85,682,676KB, Free: 18,203,544KB
P:\  (Local - NTFS)  Total:
85,682,676KB, Free: 18,203,948KB
Q:\  (Local - NTFS)  Total:
85,682,676KB, Free: 63,203,684KB
Y:\  (Local - NTFS)  Total:
4,192,964KB, Free: 2,091,372KB

Memory Report
------------------------------
------------------------------
----
Handles: 1,136
Threads: 129
Processes: 17

Physical Memory (K)
   Total: 4,128,176
   Available: 899,200
   File Cache: 54,760



Services Report
------------------------------
------------------------------
----
EventLog (Event log)
Running   (Automatic)
Server
Running   (Automatic)
Workstation (NetworkProvider)
Running   (Automatic)
TCP/IP NetBIOS Helper
Running   (Automatic)
Messenger
Running   (Automatic)
```



```
Remote Procedure Call (RPC)
Service           Running
(Automatic)



Drivers Report
---------------------------------
---------------------------------
----
AFD Networking Support
Environment (TDI)      Running
(Automatic)
ati (Video)
Running   (System)
Beep (Base)
Running   (System)
dac960nt (SCSI miniport)
Running   (Boot)
Disk (SCSI Class)
Running   (Boot)
Intel 82557-based PRO Adapter
Driver (NDIS)   Running
(Automatic)
Fastfat (Boot file system)
Running   (Disabled)
Floppy (Primary disk)
Running   (System)
i8042 Keyboard and PS/2 Mouse
Port Driver (Keyboard Port)
Running   (System)
Keyboard Class Driver (Keyboard
Class)        Running   (System)
KSecDD (Base)
Running   (System)
Mouse Class Driver (Pointer
Class)            Running
(System)
Msfs (File system)
Running   (System)
Mup (Network)
Running   (Manual)
Microsoft NDIS System Driver
(NDIS)           Running
(System)
NetBIOS Interface (NetBIOSGroup)
Running   (Manual)
WINS Client(TCP/IP) (PNP_TDI)
Running   (Automatic)
Npfs (File system)
Running   (System)
Ntfs (File system)
Running   (Disabled)
Null (Base)
Running   (System)
Rdr (Network)
Running   (Manual)
Scsiscan (SCSI Class)
Running   (System)
Srv (Network)
Running   (Manual)
TCP/IP Service (PNP_TDI)
Running   (Automatic)
VgaSave (Video Save)
Running   (System)



IRQ and Port Report
---------------------------------
---------------------------------
----
Devices
Vector Level  Affinity
---------------------------------
---------------------------------
----
MPS 1.4 - APIC platform
8     8 0x0000000f
MPS 1.4 - APIC platform
0     0 0x0000000f
MPS 1.4 - APIC platform
1     1 0x0000000f
MPS 1.4 - APIC platform
2     2 0x0000000f
MPS 1.4 - APIC platform
3     3 0x0000000f
MPS 1.4 - APIC platform
4     4 0x0000000f
MPS 1.4 - APIC platform
5     5 0x0000000f
MPS 1.4 - APIC platform
6     6 0x0000000f
MPS 1.4 - APIC platform
7     7 0x0000000f
MPS 1.4 - APIC platform
8     8 0x0000000f
MPS 1.4 - APIC platform
9     9 0x0000000f
MPS 1.4 - APIC platform
10    10 0x0000000f
MPS 1.4 - APIC platform
11    11 0x0000000f
MPS 1.4 - APIC platform
12    12 0x0000000f
MPS 1.4 - APIC platform
13    13 0x0000000f
MPS 1.4 - APIC platform
14    14 0x0000000f
MPS 1.4 - APIC platform
15    15 0x0000000f
MPS 1.4 - APIC platform
16    16 0x0000000f
MPS 1.4 - APIC platform
17    17 0x0000000f
MPS 1.4 - APIC platform
18    18 0x0000000f
MPS 1.4 - APIC platform
19    19 0x0000000f
```

```
MPS 1.4 - APIC platform
20    20 0x0000000f
MPS 1.4 - APIC platform
21    21 0x0000000f
MPS 1.4 - APIC platform
22    22 0x0000000f
MPS 1.4 - APIC platform
23    23 0x0000000f
MPS 1.4 - APIC platform
24    24 0x0000000f
MPS 1.4 - APIC platform
25    25 0x0000000f
MPS 1.4 - APIC platform
26    26 0x0000000f
MPS 1.4 - APIC platform
27    27 0x0000000f
MPS 1.4 - APIC platform
28    28 0x0000000f
MPS 1.4 - APIC platform
29    29 0x0000000f
MPS 1.4 - APIC platform
30    30 0x0000000f
MPS 1.4 - APIC platform
31    31 0x0000000f
MPS 1.4 - APIC platform
32    32 0x0000000f
MPS 1.4 - APIC platform
33    33 0x0000000f
MPS 1.4 - APIC platform
34    34 0x0000000f
MPS 1.4 - APIC platform
35    35 0x0000000f
MPS 1.4 - APIC platform
36    36 0x0000000f
MPS 1.4 - APIC platform
37    37 0x0000000f
MPS 1.4 - APIC platform
38    38 0x0000000f
MPS 1.4 - APIC platform
39    39 0x0000000f
MPS 1.4 - APIC platform
40    40 0x0000000f
MPS 1.4 - APIC platform
41    41 0x0000000f
MPS 1.4 - APIC platform
42    42 0x0000000f
MPS 1.4 - APIC platform
43    43 0x0000000f
MPS 1.4 - APIC platform
44    44 0x0000000f
MPS 1.4 - APIC platform
45    45 0x0000000f
MPS 1.4 - APIC platform
46    46 0x0000000f
MPS 1.4 - APIC platform
47    47 0x0000000f
MPS 1.4 - APIC platform
61    61 0x0000000f
MPS 1.4 - APIC platform
65    65 0x0000000f
MPS 1.4 - APIC platform
80    80 0x0000000f
MPS 1.4 - APIC platform
193   193 0x0000000f
MPS 1.4 - APIC platform
225   225 0x0000000f
MPS 1.4 - APIC platform
253   253 0x0000000f
MPS 1.4 - APIC platform
254   254 0x0000000f
MPS 1.4 - APIC platform
255   255 0x0000000f
i8042prt
1     1 0xffffffff
i8042prt
12    12 0xffffffff
E100B
5     5 0xdd39f370
Floppy
6     6 0x00000000
dac960nt
7     7 0x00000000
dac960nt
15    15 0x00000000
dac960nt
11    11 0x00000000
dac960nt
10    10 0x00000000
dac960nt
9     9 0x00000000
---------------------------------
---------------------------------
----
Devices
Physical Address  Length
---------------------------------
---------------------------------
----
MPS 1.4 - APIC platform
0x00000000  0x0000000010
MPS 1.4 - APIC platform
0x00000020  0x0000000002
MPS 1.4 - APIC platform
0x00000040  0x0000000004
MPS 1.4 - APIC platform
0x00000048  0x0000000004
MPS 1.4 - APIC platform
0x00000061  0x0000000001
MPS 1.4 - APIC platform
0x00000070  0x0000000002
MPS 1.4 - APIC platform
0x00000080  0x0000000010
MPS 1.4 - APIC platform
0x00000092  0x0000000001
MPS 1.4 - APIC platform
0x000000a0  0x0000000002
MPS 1.4 - APIC platform
0x000000c0  0x0000000010
MPS 1.4 - APIC platform
0x000000d0  0x0000000010
```

Appendix C – Tunable Parameters

```
MPS 1.4 - APIC platform
0x000000f0  0x0000000010
MPS 1.4 - APIC platform
0x00000400  0x0000000010
MPS 1.4 - APIC platform
0x00000461  0x0000000002
MPS 1.4 - APIC platform
0x00000464  0x0000000002
MPS 1.4 - APIC platform
0x00000480  0x0000000010
MPS 1.4 - APIC platform
0x000004c2  0x000000000e
MPS 1.4 - APIC platform
0x000004d0  0x0000000002
MPS 1.4 - APIC platform
0x000004d4  0x000000002c
MPS 1.4 - APIC platform
0x00000c84  0x0000000001
i8042prt
0x00000060  0x0000000001
i8042prt
0x00000064  0x0000000001
E100B
0x00007000  0x0000000014
Floppy
0x000003f0  0x0000000006
Floppy
0x000003f7  0x0000000001
ati
0x000003b0  0x000000000c
ati
0x000003c0  0x0000000020
ati
0x000003c4  0x0000000002
ati
0x000003c5  0x0000000001
ati
0x000003ce  0x0000000002
ati
0x000003cf  0x0000000001
ati
0x000001ce  0x0000000002
ati
0x000001cf  0x0000000001
ati
0x00007400  0x0000000100
VgaSave
0x000003b0  0x000000000c
VgaSave
0x000003c0  0x0000000020
VgaSave
0x000001ce  0x0000000002



DMA and Memory Report
---------------------------------
---------------------------------
----
```

```
Devices
Channel    Port
---------------------------------
---------------------------------
----
Floppy
2     0
---------------------------------
---------------------------------
----
Devices
Physical Address  Length
---------------------------------
---------------------------------
----
MPS 1.4 - APIC platform
0xfec00000  0x00000400
MPS 1.4 - APIC platform
0xfee00000  0x00000400
E100B
0xfc362000  0x00000014
E100B
0xfc362000  0x00000014
dac960nt
0xfc360000  0x00002000
dac960nt
0xfc364000  0x00002000
dac960nt
0xfc366000  0x00002000
dac960nt
0xfe400000  0x00002000
dac960nt
0xfe402000  0x00002000
ati
0x000a0000  0x00020000
ati
0xfd000000  0x00800000
VgaSave
0x000a0000  0x00020000



Environment Report
---------------------------------
---------------------------------
----



System Environment Variables

ComSpec=C:\WINNT\system32\cmd.exe
   NUMBER_OF_PROCESSORS=4
   OS=Windows_NT

Os2LibPath=C:\WINNT\system32\os2\
dll;

Path=C:\WINNT\system32;C:\WINNT;C
:\MSSQL\BINN
```

```
   PROCESSOR_ARCHITECTURE=x86
   PROCESSOR_IDENTIFIER=x86
Family 6 Model 1 Stepping 9,
GenuineIntel
   PROCESSOR_LEVEL=6
   PROCESSOR_REVISION=0109
   windir=C:\WINNT



Environment Variables for Current
User
   DIRCMD=/o
   TEMP=C:\TEMP
   TMP=C:\TEMP



Network Report
---------------------------------
---------------------------------
----
Your Access Level: Admin & Local
Workgroup or Domain: WORKGROUP
Network Version: 4.0
LanRoot: WORKGROUP
Logged On Users: 1
Current User (1): Administrator
  Logon Domain: X3
  Logon Server: X3

Transport: NetBT_E100B1, 00-A0-
C9-6D-F3-A0, VC's: 1, Wan: Wan

Character Wait: 3,600
Collection Time: 250
Maximum Collection Count: 16
Keep Connection: 600
Maximum Commands: 5
Session Time Out: 45
Character Buffer Size: 512
Maximum Threads: 50
Lock Quota: 6,144
Lock Increment: 10
Maximum Locks: 500
Pipe Increment: 10
Maximum Pipes: 500
Cache Time Out: 40
Dormant File Limit: 45
Read Ahead Throughput:
4,294,967,295
Mailslot Buffers: 3
Server Announce Buffers: 20
Illegal Datagrams: 5
Datagram Reset Frequency: 60
Bytes Received: 8,107,473
SMB's Received: 3,052
Server File Opens: 14
```

Client Hardware Configuration

```
Microsoft Diagnostics Report For
\\CLIENT1
---------------------------------
---------------------------------
----

OS Version Report
---------------------------------
---------------------------------
----
Microsoft (R) Windows NT (TM)
Server
Version 4.0 (Build 1381: Service
Pack 3) x86 Uniprocessor Free
Registered Owner: Samuel, Acer
Product Number: 21296-OEM-
0123456-01235
---------------------------------
---------------------------------
----

System Report
---------------------------------
---------------------------------
----
System: AT/AT COMPATIBLE
Hardware Abstraction Layer: PC
Compatible Eisa/Isa HAL
BIOS Date: 03/28/97
BIOS Version: <unavailable>

Processor list:
   0:  x86 Family 6 Model 1
Stepping 6 GenuineIntel ~198 Mhz
---------------------------------
---------------------------------
----

Video Display Report
---------------------------------
---------------------------------
----
BIOS Date: 01/07/94
BIOS Version: CL-GD542X VGA BIOS
Version 1.41

Adapter:
   Setting: 800 x 600 x 256
            60 Hz
   Type: cirrus compatible
display adapter
   String: Cirrus Logic
Compatible
   Memory: 1 MB
   Chip Type: CL 5424
   DAC Type: Integrated RAMDAC
Driver:
```



```
   Vendor: Microsoft Corporation
   File(s): cirrus.sys, vga.dll,
cirrus.dll, vga256.dll,
vga64K.dll
   Version: 4.00, 4.0.0


Drives Report
--------------------------------
--------------------------------
----
C:\  (Local - NTFS)  Total: 0KB,
Free: 0KB

Memory Report
--------------------------------
--------------------------------
----
Handles: 22,348
Threads: 433
Processes: 58

Physical Memory (K)
   Total: 196,020
   Available: 51,000
   File Cache: 18,720



Services Report
--------------------------------
--------------------------------
----
Alerter
Running   (Automatic)
Computer Browser
Running   (Automatic)
EventLog (Event log)
Running   (Automatic)
Server
Running   (Automatic)
Workstation (NetworkProvider)
Running   (Automatic)
License Logging Service
Running   (Automatic)
TCP/IP NetBIOS Helper
Running   (Automatic)
Messenger
Running   (Automatic)
FTP Publishing Service
Running   (Manual)
NT LM Security Support Provider
Running   (Manual)
Plug and Play (PlugPlay)
Running   (Automatic)
Remote Procedure Call (RPC)
Service           Running
(Automatic)
Simple TCP/IP Services
Running   (Automatic)
Spooler (SpoolerGroup)
Running   (Automatic)
TUXEDO IPC Helper
Running   (Automatic)
TListen (Port: 3050)
Running   (Automatic)
World Wide Web Publishing Service
Running   (Automatic)



Drivers Report
--------------------------------
--------------------------------
----
AFD Networking Support
Environment (TDI)      Running
(Automatic)
aic78xx (SCSI miniport)
Running   (Boot)
atapi (SCSI miniport)
Running   (Boot)
Beep (Base)
Running   (System)
Cdfs (File system)
Running   (Disabled)
Cdrom (SCSI CDROM Class)
Running   (System)
cirrus (Video)
Running   (System)
Disk (SCSI Class)
Running   (Boot)
Intel 82557-based PRO Adapter
Driver (NDIS)   Running
(Automatic)
Fastfat (Boot file system)
Running   (Disabled)
Floppy (Primary disk)
Running   (System)
i8042 Keyboard and PS/2 Mouse
Port Driver (Keyboard Port)
Running   (System)
Keyboard Class Driver (Keyboard
Class)        Running   (System)
KSecDD (Base)
Running   (System)
Mouse Class Driver (Pointer
Class)            Running
(System)
Msfs (File system)
Running   (System)
Mup (Network)
Running   (Manual)
Microsoft NDIS System Driver
(NDIS)           Running
(System)
NetBIOS Interface (NetBIOSGroup)
Running   (Manual)
WINS Client(TCP/IP) (PNP_TDI)
Running   (Automatic)
```



```
Npfs (File system)
Running   (System)
Ntfs (File system)
Running   (Disabled)
Null (Base)
Running   (System)
Rdr (Network)
Running   (Manual)
Srv (Network)
Running   (Manual)
TCP/IP Service (PNP_TDI)
Running   (Automatic)
```

```
IRQ and Port Report
-----------------------------------
-----------------------------------
----
Devices
Vector Level  Affinity
-----------------------------------
-----------------------------------
----
i8042prt
1     1 0xffffffff
i8042prt
12    12 0xffffffff
E100B
7     7 0x00000000
E100B
4     4 0x00000000
E100B
10    10 0x00000000
Floppy
6     6 0x00000000
aic78xx
11    11 0x00000000
atapi
0    14 0x00000000
-----------------------------------
-----------------------------------
----
Devices
Physical Address  Length
-----------------------------------
-----------------------------------
----
i8042prt
0x00000060  0x0000000001
i8042prt
0x00000064  0x0000000001
E100B
0x00007040  0x0000000014
E100B
0x00007000  0x0000000014
E100B
0x00007080  0x0000000014
Floppy
0x000003f0  0x0000000006
Floppy
0x000003f7  0x0000000001
aic78xx
0x00007400  0x0000000100
atapi
0x000001f0  0x0000000008
atapi
0x000003f6  0x0000000001
cirrus
0x000003b0  0x000000000c
cirrus
0x000003c0  0x0000000020
```

```
DMA and Memory Report
-----------------------------------
-----------------------------------
----
Devices
Channel    Port
-----------------------------------
-----------------------------------
----
Floppy
2     0
-----------------------------------
-----------------------------------
----
Devices
Physical Address  Length
-----------------------------------
-----------------------------------
----
E100B
0x11601000  0x00000014
E100B
0x11601000  0x00000014
E100B
0x11600000  0x00000014
E100B
0x11600000  0x00000014
E100B
0x11602000  0x00000014
E100B
0x11602000  0x00000014
aic78xx
0x1d700000  0x00001000
cirrus
0x000a0000  0x00020000
```

```
Environment Report
-----------------------------------
-----------------------------------
----
```



```
System Environment Variables
   APPDIR=C:\InetPub\wwwroot

ComSpec=C:\WINNT\system32\cmd.exe
   NUMBER_OF_PROCESSORS=1
   OS=Windows_NT

Os2LibPath=C:\WINNT\system32\os2\
dll;

Path=C:\WINNT\system32;C:\WINNT;;
C:\MSSQL\BINN;C:\TUXEDO\bin
   PROCESSOR_ARCHITECTURE=x86
   PROCESSOR_IDENTIFIER=x86
Family 6 Model 1 Stepping 6,
GenuineIntel
   PROCESSOR_LEVEL=6
   PROCESSOR_REVISION=0106
   TMCONTEXTS=1

TUXCONFIG=C:\InetPub\wwwroot\tuxc
onfig
   windir=C:\WINNT



Environment Variables for Current
User
   include=c:\program
files\devstudio\vc\include;c:\pro
gram
files\devstudio\vc\atl\include;c:
\program
files\devstudio\vc\mfc\include;c:
\program
files\devstudio\vc\include;c:\pro
gram
files\devstudio\vc\atl\include;c:
\program
files\devstudio\vc\mfc\include;%i
nclude%
   lib=c:\program
files\devstudio\vc\lib;c:\program
files\devstudio\vc\mfc\lib;c:\pro
gram
files\devstudio\vc\lib;c:\program
files\devstudio\vc\mfc\lib;%lib%
   MSDevDir=C:\Program
Files\DevStudio\SharedIDE
   path=c:\program
files\devstudio\sharedide\bin\ide
;c:\program
files\devstudio\sharedide\bin;c:\
program files\devstudio\vc\bin
   TEMP=C:\TEMP
   TMP=C:\TEMP



Network Report
--------------------------------
--------------------------------
----
Your Access Level: Admin & Local
Workgroup or Domain: NSBU
Network Version: 4.0
LanRoot: NSBU
Logged On Users: 1
Current User (1): Administrator
  Logon Domain: CLIENT1
  Logon Server: CLIENT1

Transport: NetBT_E100B2, 00-A0-
C9-85-EB-85, VC's: 0, Wan: Wan
Transport: NetBT_E100B1, 00-A0-
C9-6D-F4-23, VC's: 0, Wan: Wan
Transport: NetBT_E100B3, 00-A0-
C9-6D-F3-A3, VC's: 0, Wan: Wan

Character Wait: 3,600
Collection Time: 250
Maximum Collection Count: 16
Keep Connection: 600
Maximum Commands: 5
Session Time Out: 45
Character Buffer Size: 512
Maximum Threads: 50
Lock Quota: 6,144
Lock Increment: 10
Maximum Locks: 500
Pipe Increment: 10
Maximum Pipes: 500
Cache Time Out: 40
Dormant File Limit: 45
Read Ahead Throughput:
4,294,967,295
Mailslot Buffers: 3
Server Announce Buffers: 20
Illegal Datagrams: 5
Datagram Reset Frequency: 60
Bytes Received: 543
SMB's Received: 4
Server File Opens: 132
```

Client NT Registry Parameters

```
Key Name:
SOFTWARE\Microsoft\TPCC
Class Name:        <NO CLASS>
Last Write Time:   1/6/97 - 3:01
PM
Value 0
  Name:            BackoffDelay
  Type:            REG_SZ
  Data:            500

Value 1
  Name:            DeadlockRetry
```

```
  Type:             REG_SZ
  Data:             3

Value 2
  Name:             DEBUG
  Type:             REG_SZ
  Data:             OFF

Value 3
  Name:             LOG
  Type:             REG_SZ
  Data:             OFF

Value 4
  Name:             MaxConnections
  Type:             REG_SZ
  Data:             2048

Value 5
  Name:
MaximumWarehouses
  Type:             REG_SZ
  Data:             900

Value 6
  Name:
NumberOfDeliveryThreads
  Type:             REG_SZ
  Data:             7

Value 7
  Name:             PATH
  Type:             REG_SZ
  Data:
C:\InetPub\wwwroot\

Value 8
  Name:             QueueSlotts
  Type:             REG_SZ
  Data:             3000


Key Name:
SYSTEM\CurrentControlSet\Services
\InetInfo\Parameters
Class Name:         <NO CLASS>
Last Write Time:    1/8/98 - 11:02
AM
Value 0
  Name:             BandwidthLevel
  Type:             REG_DWORD
  Data:             0xffffffff

Value 1
  Name:             ListenBackLog
  Type:             REG_DWORD
  Data:             0xc1c

Value 2
  Name:
PoolThreadsLimit
  Type:             REG_DWORD
  Data:             0x200

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


Key Name:
SYSTEM\CurrentControlSet\Services
\InetInfo\Parameters\Filter
Class Name:         <NO CLASS>
Last Write Time:    9/22/97 - 3:55
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:    9/22/97 - 3:55
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
\W3SVC\Parameters
Class Name:        <NO CLASS>
Last Write Time:   1/4/97 - 6:35
PM
Value 0
  Name:
AcceptExOutstanding
  Type:            REG_DWORD
  Data:            0xc1c

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_CLIENT1

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:             0x384

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\sspifil
t.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\Deny IP List
Class Name:         <NO CLASS>
Last Write Time:    9/22/97 - 3:59
PM

Key Name:
SYSTEM\CurrentControlSet\Services
\W3SVC\Parameters\Grant IP List
Class Name:         <NO CLASS>
Last Write Time:    9/22/97 - 3:59
PM

Key Name:
SYSTEM\CurrentControlSet\Services
\W3SVC\Parameters\Script Map
Class Name:         <NO CLASS>
Last Write Time:    9/22/97 - 3:55
PM
Value 0
  Name:             .idc
  Type:             REG_SZ
  Data:
C:\WINNT\System32\inetsrv\httpodb
c.dll


Key Name:
SYSTEM\CurrentControlSet\Services
\W3SVC\Parameters\Virtual Roots
Class Name:         <NO CLASS>
Last Write Time:    9/23/97 - 9:58
AM
Value 0
  Name:             /,
  Type:             REG_SZ
  Data:
C:\InetPub\wwwroot,,5

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

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

Tuxedo Servers Configuration File

```
*RESOURCES
IPCKEY 133133

MAXACCESSERS 1900
MAXSERVERS 100
MAXSERVICES 100
MODEL SHM
MASTER CLIENT1
LDBAL Y
SCANUNIT 15
BLOCKTIME 60
BBLQUERY 60

*MACHINES
DEFAULT:

"CLIENT1" LMID= CLIENT1
      TUXDIR="C:\tuxedo"
      APPDIR="C:\InetPub\wwwroot"
      TUXCONFIG="C:\InetPub\wwwro
ot\tuxconfig"
      ULOGPFX="C:\InetPub\wwwroot
\ULOG"
      TYPE="WinNT"
      UID= 0
      GID= 0

*GROUPS
GROUPNO
      LMID=CLIENT1 GRPNO=1
OPENINFO=NONE

GROUPPAY
```



```
      LMID=CLIENT1 GRPNO=2
OPENINFO=NONE

GROUPOS
      LMID=CLIENT1 GRPNO=3
OPENINFO=NONE

GROUPSL
      LMID=CLIENT1 GRPNO=4
OPENINFO=NONE

GROUPDEL
      LMID=CLIENT1 GRPNO=5
OPENINFO=NONE

*SERVERS
DEFAULT:

neworder SRVGRP=GROUPNO
      SRVID=100
      MIN=14 MAX=20
      CLOPT="-A -- -SX3"
      RQADDR=newq REPLYQ=Y

payment SRVGRP=GROUPPAY
      SRVID=200
      MIN=9 MAX=20
      CLOPT="-A -- -SX3"
      RQADDR=payq REPLYQ=Y

orderstatus SRVGRP=GROUPOS
      SRVID=300
      MIN=3 MAX=10
      CLOPT="-A -- -SX3"
      RQADDR=ordq REPLYQ=Y

stocklevel SRVGRP=GROUPSL
      SRVID=400
      MIN=9 MAX=10
      CLOPT="-A -- -SX3"
      RQADDR=stkq REPLYQ=Y

delivery SRVGRP=GROUPDEL
      SRVID=500
      MIN=3 MAX=10
      CLOPT="-A -- -SX3 -F"
      RQADDR=delq REPLYQ=N


*SERVICES
```

RTE Parameters

```
Profile:    X3100x4
File Path:  C:\Benchcrf\Profile\X3_900b.pro
Version:    1.0.1

      Number of Engines: 10

            Name: DRIVER1A
            Description: 900 users to client1a
            Directory: C:\Benchcrf\Log\driver1a
            Machine: DRIVER1
            Parameter Set: PARAM2
            Index: 100000000
            Seed: 1
            Configured Users: 900
            Pipe Name: DRIVER6243468
            Connect Rate: 100
            Start Rate: 0
            CLIENT_NURAND: 233
            CPU: 0

            Name: DRIVER1B
            Description: 900 users to client1b
            Directory: C:\benchcrf\log\driver1b
            Machine: DRIVER1
            Parameter Set: PARAM2
            Index: 200000000
            Seed: 1
```

```
Configured Users: 900
Pipe Name: DRIVER6320062
Connect Rate: 100
Start Rate: 0
CLIENT_NURAND: 233
CPU: 0

Name: DRIVER2A
Description: 900 users to client2a
Directory: C:\benchcrf\log\driver2a
Machine: DRIVER2
Parameter Set: PARAM2
Index: 300000000
Seed: 1
Configured Users: 900
Pipe Name: DRIVER7395984
Connect Rate: 100
Start Rate: 0
CLIENT_NURAND: 233
CPU: 0

Name: DRIVER2B
Description: 900 users to client2b
Directory: C:\benchcrf\log\driver2b
Machine: DRIVER2
Parameter Set: PARAM2
Index: 400000000
Seed: 1
Configured Users: 900
Pipe Name: DRIVER7438125
Connect Rate: 100
Start Rate: 0
CLIENT_NURAND: 233
CPU: 0

Name: DRIVER3A
Description: 900 users to client3a
Directory: C:\Benchcrf\log\driver3a
Machine: DRIVER3
Parameter Set: PARAM2
Index: 500000000
Seed: 1
Configured Users: 900
Pipe Name: DRIVER55343046
Connect Rate: 100
Start Rate: 0
CLIENT_NURAND: 233
CPU: 0

Name: DRIVER3B
Description: 900 users to client3b
Directory: C:\Benchcrf\log\driver3b
Machine: DRIVER3
Parameter Set: PARAM2
Index: 600000000
Seed: 1
Configured Users: 900
Pipe Name: DRIVER65402046
Connect Rate: 100
Start Rate: 0
```



```
        CLIENT_NURAND: 233
        CPU: 0

        Name: DRIVER4A
        Description: 900 users to client4a
        Directory: C:\Benchcrf\Log\driver4a
        Machine: DRIVER4
        Parameter Set: PARAM2
        Index: 700000000
        Seed: 1
        Configured Users: 900
        Pipe Name: DRIVER75442078
        Connect Rate: 100
        Start Rate: 0
        CLIENT_NURAND: 233
        CPU: 0

        Name: DRIVER4B
        Description: 900 users to client4b
        Directory: C:\Benchcrf\Log\driver4b
        Machine: DRIVER4
        Parameter Set: PARAM2
        Index: 800000000
        Seed: 1
        Configured Users: 900
        Pipe Name: DRIVER85481812
        Connect Rate: 100
        Start Rate: 0
        CLIENT_NURAND: 233
        CPU: 0

        Name: DRIVER5A
        Description: 900 users to client5a
        Directory: C:\Benchcrf\Log\driver5a
        Machine: DRIVER5
        Parameter Set: PARAM2
        Index: 900000000
        Seed: 1
        Configured Users: 900
        Pipe Name: DRIVER95525125
        Connect Rate: 100
        Start Rate: 0
        CLIENT_NURAND: 233
        CPU: 0

        Name: DRIVER5B
        Description: 900 users to client5b
        Directory: C:\Benchcrf\Log\driver5b
        Machine: DRIVER5
        Parameter Set: PARAM2
        Index: 1000000000
        Seed: 1
        Configured Users: 900
        Pipe Name: DRIVER105571421
        Connect Rate: 100
        Start Rate: 0
        CLIENT_NURAND: 233
        CPU: 0

    Number of User groups: 10
```

```
Driver Engine: DRIVER1A
IIS Server: client1a
SQL Server: TPC
User: sa
Protocol: Html
w_id Range: 1 - 90
w_id Max Warehouse: 900
Scale: Normal
User Count: 900
District id: 1
Scale Down: No

Driver Engine: DRIVER4A
IIS Server: client4a
SQL Server: X3
User: sa
Protocol: Html
w_id Range: 541 - 630
w_id Max Warehouse: 900
Scale: Normal
User Count: 900
District id: 1
Scale Down: No

Driver Engine: DRIVER4B
IIS Server: client4b
SQL Server: X3
User: sa
Protocol: Html
w_id Range: 631 - 720
w_id Max Warehouse: 900
Scale: Normal
User Count: 900
District id: 1
Scale Down: No

Driver Engine: DRIVER5A
IIS Server: client5a
SQL Server: X3
User: sa
Protocol: Html
w_id Range: 721 - 810
w_id Max Warehouse: 900
Scale: Normal
User Count: 900
District id: 1
Scale Down: No

Driver Engine: DRIVER5B
IIS Server: client5b
SQL Server: X3
User: sa
Protocol: Html
w_id Range: 811 - 900
w_id Max Warehouse: 900
Scale: Normal
User Count: 900
District id: 1
Scale Down: No
```



```
Driver Engine: DRIVER1B
IIS Server: client1b
SQL Server: TPC
User: sa
Protocol: Html
w_id Range: 91 - 180
w_id Max Warehouse: 900
Scale: Normal
User Count: 900
District id: 1
Scale Down: No

Driver Engine: DRIVER2A
IIS Server: client2a
SQL Server: TPC
User: sa
Protocol: Html
w_id Range: 181 - 270
w_id Max Warehouse: 900
Scale: Normal
User Count: 900
District id: 1
Scale Down: No

Driver Engine: DRIVER2B
IIS Server: client2b
SQL Server: TPC
User: sa
Protocol: Html
w_id Range: 271 - 360
w_id Max Warehouse: 900
Scale: Normal
User Count: 900
District id: 1
Scale Down: No

Driver Engine: DRIVER3A
IIS Server: client3a
SQL Server: X3
User: sa
Protocol: Html
w_id Range: 361 - 450
w_id Max Warehouse: 900
Scale: Normal
User Count: 900
District id: 1
Scale Down: No

Driver Engine: DRIVER3B
IIS Server: client3b
SQL Server: X3
User: sa
Protocol: Html
w_id Range: 451 - 540
w_id Max Warehouse: 900
Scale: Normal
User Count: 900
District id: 1
Scale Down: No
```



```
      Number of Parameter Sets: 2


            PARAM2

                                Txn      Think    Key      RT       RT       Menu
                                Weight   Time     Time     Delay    Fence
Delay
               New Order          44.81    12.10        18.02         0.10
5.00     0.10
               Payment            43.09    12.10         3.02         0.10
5.00     0.10
               Delivery            4.02     5.10         2.02         0.10
5.00     0.10
               Stock Level         4.04     5.10         2.02         0.10
20.00    0.10
               Order Status        4.04    10.10         2.02         0.10
5.00     0.10

               ~Default
               Default Parameter Set
                                Txn      Think    Key      RT       RT       Menu
                                Weight   Time     Time     Delay    Fence
Delay
               New Order          10.00    12.05        18.01         0.10
5.00     0.10
               Payment            10.00    12.05         3.01         0.10
5.00     0.10
               Delivery            1.00     5.05         2.01         0.10
5.00     0.10
               Stock Level         1.00     5.05         2.01         0.10
20.00    0.10
               Order Status        1.00    10.05         2.01         0.10
5.00     0.10
```


Appendix D – Disk Storage

Note : Numbers are in KBytes unless otherwise specified

Warehouses	900	tpmC	11072.07	tpmC/W	12.30	

Table	Rows	Data	Index	5% Space	8H Space	Total Space
Warehouse	900	1,800	12	91		1,903
District	9,000	18,000	76	904		18,980
Item	100,000	9,100	46	210		9,356
New-order	8,100,000	90,000	548		18,000	108,548
History	27,000,000	1,350,002	0		222,105	1,572,107
Orders	27,000,000	702,000	4,234		116,191	822,425
Customer	27,000,000	18,003,600	1,397,328	446,221		19,847,149
Order-line	269,999,044	15,008,724	98,104		2,485,409	17,592,237
Stock	90,000,000	30,006,000	165,786	693,951		30,865,737
Totals	449,208,944	65,189,226	1,666,134	1,141,377	2,841,705	70,838,442

Segment	LogDev Cnt.	Seg. Size	Needed	Overhead		Not Needed
c_wdi_dev	1	102,400	30,541	305	w,d,l	71,553
c_grow_dev	1	10,240,000	2,528,111	25,281	h,o,no	7,686,608
c_ol_dev	2	61,440,000	17,768,159	177,682	ol	43,494,159
c_stock_dev	3	32,890,880	31,174,394	311,744	s	1,404,742
c_cust_dev	3	21,442,560	20,045,621	200,456	c	1,196,4830
Totals		126,115,840	71,546,827	715,468		53,853,545

Dynamic space	16,600,086	Sum of Data for Order, Order-Line and History (excluding free extents)
Static space	52,112,119	Data + Index + 5% Space + Overhead - Dynamic space
Free space	3,550,090	Total Seg. Size - Dynamic Space - Static Space - Not Needed
Daily growth	3,267,508	(Dynamic space/W * 62.5)* tpmC
Daily spread	(1,351,172)	Free space - 1.5 * Daily growth (zero if negative)
180 day (KB)	640,263,539	Static space + 180 (daily growth + daily spread)
180 day (GB)	610.60	Excludes OS, Paging and RDBMS Logs
Log size (MB)	30000	Total size of log file
% Log used	19.8356	% of log file used during entire run
Total N-O Txn	1074827	Total count of N-O transactions during entire run
Log per N-O txn	2.8346394	Number of 2K blocks per New-Order transaction
8 Hour Log (GB)	28.73	57.468247 mirrored

Appendix E – Price Quotations

Microsoft

January 30, 1998

Mr. Duncan Liu
Acer Inc.
21F, 88, Sec. 1, Hsin Tai Wu Road
hsichih, Taipei Hsien 221
Taiwan

via FAX: 011-886-2-8691-2379

Dear Duncan,

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

Microsoft SQL Server, Enterprise Edition 6.5, unlimited user license	$28999
Microsoft Windows NT Server, Enterprise Edition 4.0, incl 25 CALs	$3999
Windows NT Server 4.0, incl 5 CALs	$809
Microsoft SQL Workstation (includes programmers toolkit)	$499
Visual C++ 32-bit edition (subscription)	$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 Group

Appendix E – Price Quotations

