PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 :
G06F 11/00, 12/00

A1

(11) International Publication Number: WO 98/21654

(43) International Publication Date: 22 May 1998 (22.05.98)

(21) International Application Number: PCT/US97/20561

(22) International Filing Date: 11 November 1997 (11.11.97)

(30) Priority Data:
60/030,905 14 November 1996 (14.11.96) US

(81) Designated States: AU, CA, CN, IL, JP, KR, MX, NO, NZ, SG, European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Published
With international search report.
Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

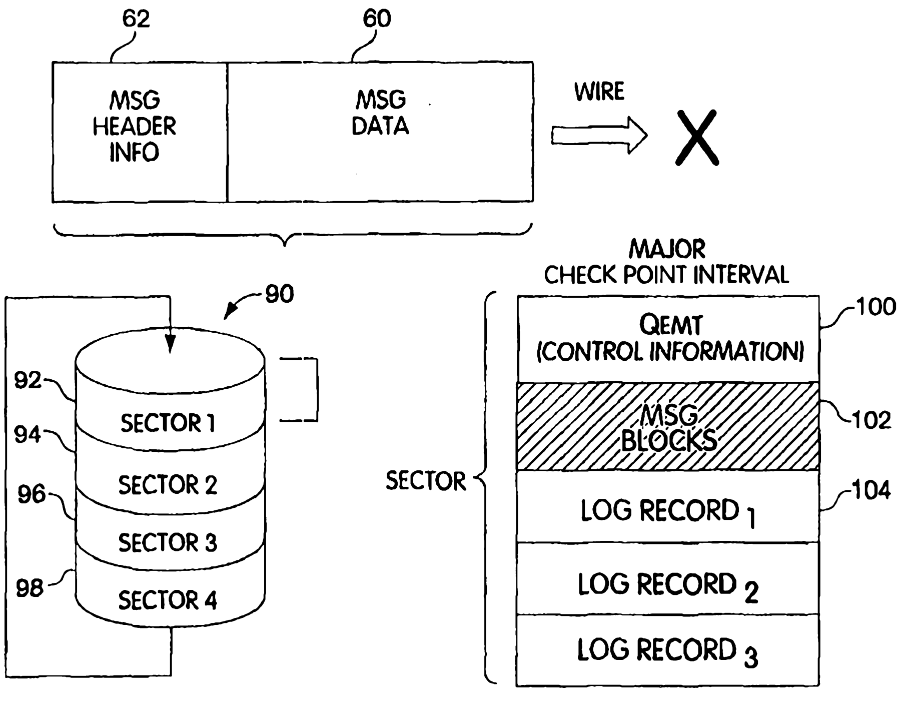
(71) Applicant: MITSUBISHI ELECTRIC INFORMATION TECHNOLOGY CENTER AMERICA, INC. [US/US]; 201 Broadway, Cambridge, MA 02139 (US).

(72) Inventors: WONG, David, W., H.; 162 Guggins Lane, Boxborough, MA 01719 (US). SCHWENKE, Derek, L.; 95 Rice Street, Marlboro, MA 01752 (US).

(74) Agent: TENDLER, Robert, K.; 65 Atlantic Avenue, Boston, MA 02110 (US).

(54) Title: LOG BASED DATA ARCHITECTURE FOR A TRANSACTIONAL MESSAGE QUEUING SYSTEM

(57) Abstract



A message queuing system is provided that saves and stores messages and their state in an efficient single file on a single disk to enable rapid recovery from server failures. The single disk, single file storage system into which messages and their states are stored eliminates writes to three different disks, the data disk, the index structure disk and the log disk. The single disk, single file storage is made possible by clustering all information together in a contiguous space on the same disk. The result is that all writes are contained in one sweeping motion of the write head in which the write head moves only in one direction and only once to find the area where it needs to start writing messages and their states are stored. In order to keep track of the clustered information, a unique Queue Entry Map Table (100) is used which includes control information (100), message blocks (102) and log records (104) in conjunction with single file disk storage that allows the write head never to have to back–up to traverse saved data when writing new records. The system also permits locating damaged files without the requirement of scanning entire log files.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	**ES**	Spain	**LS**	Lesotho	**SI**	Slovenia
AM	Armenia	**FI**	Finland	**LT**	Lithuania	**SK**	Slovakia
AT	Austria	**FR**	France	**LU**	Luxembourg	**SN**	Senegal
AU	Australia	**GA**	Gabon	**LV**	Latvia	**SZ**	Swaziland
AZ	Azerbaijan	**GB**	United Kingdom	**MC**	Monaco	**TD**	Chad
BA	Bosnia and Herzegovina	**GE**	Georgia	**MD**	Republic of Moldova	**TG**	Togo
BB	Barbados	**GH**	Ghana	**MG**	Madagascar	**TJ**	Tajikistan
BE	Belgium	**GN**	Guinea	**MK**	The former Yugoslav	**TM**	Turkmenistan
BF	Burkina Faso	**GR**	Greece		Republic of Macedonia	**TR**	Turkey
BG	Bulgaria	**HU**	Hungary	**ML**	Mali	**TT**	Trinidad and Tobago
BJ	Benin	**IE**	Ireland	**MN**	Mongolia	**UA**	Ukraine
BR	Brazil	**IL**	Israel	**MR**	Mauritania	**UG**	Uganda
BY	Belarus	**IS**	Iceland	**MW**	Malawi	**US**	United States of America
CA	Canada	**IT**	Italy	**MX**	Mexico	**UZ**	Uzbekistan
CF	Central African Republic	**JP**	Japan	**NE**	Niger	**VN**	Viet Nam
CG	Congo	**KE**	Kenya	**NL**	Netherlands	**YU**	Yugoslavia
CH	Switzerland	**KG**	Kyrgyzstan	**NO**	Norway	**ZW**	Zimbabwe
CI	Côte d'Ivoire	**KP**	Democratic People's	**NZ**	New Zealand		
CM	Cameroon		Republic of Korea	**PL**	Poland		
CN	China	**KR**	Republic of Korea	**PT**	Portugal		
CU	Cuba	**KZ**	Kazakstan	**RO**	Romania		
CZ	Czech Republic	**LC**	Saint Lucia	**RU**	Russian Federation		
DE	Germany	**LI**	Liechtenstein	**SD**	Sudan		
DK	Denmark	**LK**	Sri Lanka	**SE**	Sweden		
EE	Estonia	**LR**	Liberia	**SG**	Singapore		

LOG BASED DATA ARCHITECTURE FOR A TRANSACTIONAL MESSAGE QUEUING SYSTEM.

FIELD OF INVENTION

This invention relates to message queuing, and more particularly to a fast, reliable message queuing system for both client-server and mobile agent applications.

CROSS-REFERENCE TO RELATED APPLICATIONS

A claim of priority is made to U.S. Provisional Patent Application Serial No. 60/030,905, filed November 14, 1996, entitled LOG BASED DATA ARCHITECTURE FOR A TRANSACTIONAL MESSAGE QUEUING SYSTEM.

BACKGROUND OF THE INVENTION

Message queuing is the most fundamental paradigm for communication between applications on different computer systems due to its inherent flexibility in allowing both synchronous and asynchronous processing. The message queuing middleware infrastructure is a very flexible framework for a number of application domains in both general client-server as well as mobile agent computing arenas, to wit work flow computing, object messaging, transactional messaging and data replication services.

It will be appreciated that in many transactional messaging scenarios data is oftentimes lost during the transmission. This is no more catastrophic than in the banking industry in which banking records transmitted from one location to another can be lost due to server failures, transmission line failures or other artifacts. It is incumbent upon the system managers to be able to quickly locate the fact that an error has occurred and to be able to reconstruct the data from a known point where the data was valid.

Establishing that point at which an error has occurred has in the past been accomplished by systems which scan an entire so-called log file to reconstruct the up-to-date state of the system before the crash. Log files are routinely utilized with their associated time stamps to identify messages and the data they contain. However, the scanning of entire log files to ascertain the up-to-date state can require scanning as many as 1,000 log records.

Not only is the scanning of the overall log record an inefficient way to ascertain where an error occurred and to be able to reconstruct files from that point, systems in the past have required two disk files, one serving as a data file, and the other serving as a log file.

Moreover, the correlation between the log entries and the data files or sectors is complicated by the fact that in the past, sectors were stored in some indiscriminant order, leaving the mapping between the log file and the sectors a somewhat time consuming process.

By way of further background, it will be appreciated that message queuing is used in general to be able to provide a fail-safe storage for data records which are transmitted from one point to another. If, for instance, an error occurs and data is lost at one location, it can be reconstructed at a second location due to the storage inherent in message queuing.

As an example, it is desirable, especially in stock market trades, that any interruption in trading to be minimized to minutes as opposed to hours. On occasion, however, when system servers go down, recovery can take from two to eight hours depending on the number of trades in the system at that time. There is thus a need to minimize down time and expense of locating and reconstructing damaged files.

Note that as used herein, the term queue file refers to the physical storage of messages that are in transmission. Queue files may also be viewed as holding cells for uncompleted operations. Basically, what this means is that if the receiver is not there to receive a given message, the message is held in the queue file and is deliverable at a later time. As a result, the queue files offer reliability in the retention of information that is transmitted.

Moreover, in traditional systems, the recovery data is not provided by the queue file itself. Thus, queue files have not been utilized to identify the state of the file when an error or lost data has occurred, and have thus not been used to reconstruct the data file from data which is previously uncorrupted. In a traditional system, the recovery data is not provided by the queue file itself.

Another example of how message queuing is applied to a real-world application involves how a message queuing infrastructure may support real-time on-line transaction processing using mobile agents. In this example, the customer, for instance, is a bank with geographically dispersed branches. Customer accounts are created and kept at the local branches where the account was opened. For illustrative purposes, this is called the home branch of the account. A copy of each account is also kept at the main office. A read operation on an account can be made from either the local branch or the main office. An update to an account, however, will require that both the home branch copy and the main office copy be updated in a coordinated fashion.

If the update request occurred at the home branch, the local copy must then be updated. This update can trigger an agent which then automatically submits an enqueue request to the queue manager or queue server. This queue manager in turn dequeues the request across a wide area network to another queue manager, which in turn, dequeues the update request to the database server for the mirror office accounts.

A message queue in this example provides asynchronous and reliable processing. Asynchronous processing begins with the agent that is triggered by the database update at one location. The agent submits the update request to the message queue manager in an asynchronous manner, and need not wait around for a response. The message queue manager serves as holding cell for the request so that the requester can continue processing without the need to wait for a response. The message queue manager also provides reliability in this example in that it maintains a copy of the update request in its queue until the recipient of this update request has acknowledged its receipt via a well-known handshaking protocol called the Two Phase Commit protocol, known in the industry as transactional message queuing.

While these types of message queuing systems have operated reliably in the past, they have relied on a data architecture that uses separate queue data and log record files to store the messages that are appended to a message queue. This architecture prevents rapid repair at the time of a serve crash and requires two storage disks, one for data and one for the log records. Moreover, traditional message queuing architectures are generally not optimized for write operations without requiring extra hardware to work efficiently, and are not appropriate for high throughput systems with low message residence times. The separate queue data and log files mentioned above also introduce an extra level of unreliability since there exists two points of potential file corruption and media failure. Additionally, there is usually no means for the message queuing systems administrator to predefine the amount of work needed to do recovery a priori.

Note, the above systems are commercially available as Digital Equipment Corporation's DECmessageQ, IBM's MQ Series, and Transarc's Encina RQS.

SUMMARY OF INVENTION

In order to solve the above noted problems with traditional message queuing, a message queuing system is provided that saves and stores messages and their state in an efficient single file on a single disk to enable rapid recovery from server failures. The single disk, single file storage system into which messages and their states are stored eliminates writes to three different disks, the data disk, the index structure disk and the log disk. The single disk, single file storage is made possible by clustering all information together in a contiguous space on the same disk. The result is that all writes are contained in one sweeping motion of the write head in which the write head moves only in one direction and only once to find the area where it needs to start writing messages and their states are stored. In order to keep track of the clustered information, a unique Queue Entry Map Table is used which includes control information, message blocks and log records in conjunction with single file disk storage that allows the write head never to have to back-up to traverse saved data when writing new records. The system also permits locating damaged files without the requirement of scanning entire log files.

In order to find the most recent valid data, a control check point interval system is utilized to find the most recent uncorrupted data. Scanning to find the most recent check point interval permits rapid identification of the last queue. Subsequent scanning of log records after the checkpoint establishes the most up-to-date state of all messages. The above system permits data recovery in an order of magnitude less time than previous systems, while at the same time establishing an efficient forward writing mechanism to prevent the need for searching through unordered sectors.

In one embodiment, a circular wrap around buffering system is used in which a modification of a previous sector is made by appending a new record at the last sector to indicate that the state of a file has changed, thus to reuse previous blocks that have been freed and no longer hold valid messages and/or log records.

The present invention thus provides a log-based data architecture for transactional message queuing systems which utilizes a combined on-disk file structure for the message queue data and log records. It is the combined queue data/log record file, in one embodiment, on a single disk, which improves write operation performance and reliability, while at the same time reducing the number of disks used. As mentioned above, system crash recovery is accelerated through the use of a Queue Entry Map Table which does not require searching though all of the log records to ascertain where the error occurred. The use of the Queue Entry Map Table also permits a priori assigning the number of requirements on a queue data file that results in extensibility and flexibility to system administrators.

Also as mentioned above, the subject system utilizes a circular queue that implies that there is potential wrap around of the queue data file for storage reuse. This requires that a reservation table or free space heap be maintained to ensure that when the queue wraps around, subsequent write operations do not overwrite queue data and/or log records that might still be valid.

In one embodiment, the queue data storage architecture consists of a single flat file that is created when a queue manager is first initialized based on a fixed size for the queue. The initial queue creation is based on the system administrator's feel for the peak load on the message queuing system, e.g., the maximum number of expected entries in message queue at any given point in time. Each message in the queue data file contains a Message Header and a Message Body. The Message Body, which contains the message content, is stored on disk in subsequent contiguous blocks that follow the message header.

In the above embodiment, the queue data file is partitioned into a predefined number of logical segments or sectors which can be extended at run time. Each segment contains a copy of the Queue Entry Map Table or QEMT for short, which is stored at the beginning of each segment. The QEMT contains control information for the queue entries and log record information stored in the entire queue file. Message headers, message bodies, and log records are stored after the QEMT with potential mixing of message data and log record blocks.

As will be appreciated, the QEMT size depends on some expected maximum number of queue entries defined by the user at queue creation time. Since the log record takes up some deterministic number of bytes, the queue data file will consist of mixed data types of log records, message headers, message bodies, and QEMTs.

When a new segment is reached in the queue data file, a new QEM Table is written to disk at the beginning of the new segment, with the message and log records following the QEM Table. Since the smallest on-disk data type is the log record, a segment in the queue data file is defined to consist of blocks, where one block is the size of the log record. This implementation enhancement simplifies development of search algorithms.

The state of a transactional message queuing system is captured by the control information contained in a QEMT. The QEMT is defined as a static data structure that multiple threads can operate on, rather than each thread maintaining its own copy.

As a result of the log-based data architecture, the subject invention provides a number of improvements over existing transactional message queuing data architectures. It improves on the performance of the write operation over existing message queuing architectures, which makes message queuing systems based on this invention highly appropriate for high throughput systems with low message residence times such as high speed banking applications. The subject system is also applicable to the underlying reliable messaging infrastructure for the transport of agents over unreliable networks and/or networks with different bandwidths.

Moreover, message data and log record write operations always proceed in the forward direction and both can be stored on the same disk file.

This system also improves the reliability of transactional message queuing systems. In this log-based data architecture, there exists a single place where file corruption can occur versus two potential file corruption scenarios with separate queue data and log record files. Reliability is also improved since fewer disk files are used. A combined queue data/log record file adheres to the Atomicity, Consistency, and Isolation properties of the well-known ACID properties. Also, as will be seen, one can utilize existing RAID technology to do transparent duplicate writes.

The subject system allows the resulting message queuing system to support any method of message data access including First In First Out, Last In First Out or priority-based message data access, while at the same time reducing the amount of time needed for recovery from system crashes. Instead of scanning all data in an entire file for log records in traditional approaches, the subject system only requires that one test a few Queue Entry Map Tables first to determine the most recent checkpoint, and then proceed to scan the log records within that segment.

Moreover, the subject system provides extensibility and flexibility to message queuing systems administration since the invention allows the administrator to control how much work they want to do on system recovery by a priori predefining the number of segments on a queue data file, and subsequently the number of checkpoint intervals, again determined a priori. System administrators can thus pay the overhead cost of writing the checkpoints up front to avoid paying the heavier cost of doing extensive log record scans upon recovery. This tradeoff can be adjusted and fine-tuned to suit the application requirements and domains.

The above advantages flow from the use of a pre-allocated on-disk queue buffer containing queue control information, message data, and transactional log records of message operations. The on-disk queue buffer consists of a number of segments or sectors. Each segment consists of the same predefined number of blocks. At the beginning of each segment is the aforementioned Queue Entry Map Table, which contains control information data regarding the state of the individual queue entries, and pointer offsets to where on disk the messages are physically stored. The Queue Entry Map Table serves as a fixed checkpoint interval for the entire message queuing system. Messages and transactional log records of message operations are stored on the blocks in the segment such that message blocks and log record blocks can be intertwined. Moreover, there is no requirement that the log record for a particular message be stored contiguously to the message.

As a feature of the subject invention, a message data write operation always proceeds in a forward manner for the disk head. Additionally, a message is stored contiguously on disk with no need for pointer traversal. Further, a log record write operation always proceeds in a forward manner for the disk head. Log records are written for change of state in a message operation that follows the Two Phase Commit protocol. Therefore, log records can be written for Prepare, Prepared, Commit, Abort, Acknowledge messages from a remote queue manager.

As an another unique feature, the entire queue can be scanned in a single pass. Moreover, on-disk garbage collection is always a linear process. Additionally, there exists a number of Queue Entry Map Tables on the same file, with the unique sequence number of the most recent table being stored on disk on a graceful shutdown of the queue manager.

Importantly, the read operation can follow the First In First Out, Last In First Out, or Priority-based policy such that no special provision is needed to implement any of the three policies.

Moreover, the recovery procedure is accelerated by searching only the Queue Entry Map Tables timestamp. This is because, the most recent Queue Entry Map Table serves as the starting state for the recovery process. Log records following this table are then read sequentially and changes are then made to the in-memory copy of this most recent Queue Entry Map Table to reflect changes made after the
last known checkpoint.

BRIEF DESCRIPTION OF THE DRAWINGS

These and other features of the Subject Invention will be better understood with reference to the Detailed Description taken in conjunction with the Drawings, of which:

Figure 1 is a block diagram of a typical banking application utilizing the subject system in which messages flow from the main office to subsidiary branches;

Figure 2 is a diagrammatic representation of a two file system in which data is recorded at one file, whereas logs are recorded on a separate file, with the data stored at non-consecutive sectors and with the requirement that the entire log file be scanned in order to reconstruct an up-to-date state, the recovery process involving both the data file and log file to obtain the complete state of all messages in the system;

Figure 3 is a diagrammatic representation of the subject system in which a single file is utilized to store the data and QEMT mapping table to permit rapid recovery of lost data with a minimum amount of hardware and with reduced scanning time required for data recovery;

Figure 4 is a diagrammatic illustration of the storage of blocks of data within the file of Figure 3, indicating a circular file with a single write direction;

Figure 5 is a diagrammatic illustration of the possible QEMT control blocks at various well known positions or offsets within the file indicating that through the utilization of these QEMT control blocks, the position and/or location of valid data can be easily ascertained;

Figure 6 is a diagrammatic illustration showing the interdispersion of state change log records with the message data blocks to enable the forward writing of the file;

Figure 7 is a table illustrating the QEMT structure, including the QEMT sequence number which serves as a time stamp and which contains the incremental check point information required to restore the system;

Figure 8 is a table providing information to permit the restoration of individual message states;

Figure 9 is a diagrammatic illustration of the forward directional flow of data in a wrap around system in which a circular queue is implemented;

Figure 10 is a table illustrating the information stored in the incremental log record with the log entries of Figure 6;

Figure 11 is a flow chart illustrating the procedure for fetching a message from the queue;

Figure 12 is a flow chart illustrating a procedure for writing a message in the queue; and,

Figure 13 is a flow chart illustrating the recovery process in which the most recent QEMT is identified by an initial scan, with subsequent reading of the log records following the identification of the most recent QEMT resulting in a completely restored state.

DETAILED DESCRIPTION

Referring now to Figure 1, a message queuing system 10 is provided between branch offices of banks 12 and a main office 14 for the purpose of transmitting updated account information from the branches to the main office. In order to accomplish this, data is entered at terminals 16, 18 and 20 respectively at different branch offices of the bank. This data is stored in local database servers 22, 24 and 26 of the respective branches, with each database server having its own local storage, here designated by reference character 28.

The output the database server is coupled to a series of message queuing servers 30, 32 and 34 respectively, each having their own storage units, here labeled by reference character 36.

The outputs of the message queuing servers are applied to a wide area network 40 which couples the outputs to a message queuing server 42 at the main office, with this server having associated respective storage units 44 as illustrated. The message queuing servers 30, 32, and 34 communicate with a wide to a database server 50 having its associated units 52 as illustrated. The output of the message queuing server 42 is coupled to a database server 50 having its associated units 52 as illustrated. The information in this database is viewable at terminals 54 at the main office.

It is the purpose of the message queuing system to be able to reliably transmit updated account information from the branches so that it will reside at the main office. It is also important that the transaction at the branches can proceed without regard to direct connection to the central office.

Referring now to Figure 2, in the past messages and headers such as illustrated at 60 and 62 were stored on data disks 64 in sectors 66, 68, 70 and 72, with the message and accompanying header being randomly placed within the sectors.

At the same time, message state information was stored on a log disk 80 which included records about each message stored in the data disk, including the order of arrival and its location on the data disk. Moreover, the state of the transaction was logged into log disk 80 for each of the messages and corresponding headers.

In the case of an interrupted transmission as indicated by "X" 82, in the past was a requirement that the entire log file, here illustrated at 84, be scanned to be able to reconstruct the up-to-date state of the data disk file just prior to the interruption of the transmission. As mentioned hereinbefore, this is a time-consuming process in which the entire log file must be scanned in order to be able to reconstruct the state of the system just prior to the crash. The situation is made even more complicated due to the storage of the message and header information at nonsequential sectors on the data disk, requiring the interaction of the log file and the data file in order to locate those messages which are uncorrupted at the time of the interruption of the transmission.

Referring now to Figure 3, in the subject system message data 60 and message header information 62 are stored on a single disk storage 90 in sequential sectors, here illustrated at 92, 94, 96 and 98. It is a feature of the subject invention that the message and header information is stored in an order which is accessible through the utilization of a queue entry management table, which locates message data through a checkpoint system to be described.

It will be appreciated that the message data is not stored across all of the sectors, but rather is stored in the above-mentioned sequential manner.

In order to be able to access the data stored in file 90, the queue entry management table, or QEMT, contains sector information which includes entries for control information 100, message blocks 102 and log records 104 all of which are designed to uniquely specify the sector in which relevant data and headers can be found. The QEMT therefore specifies the state of the system in so doing.

As will be seen in connection with Figures 4, 5 and 6 the Queue Entry Management Table is stored in file 90 interspersed between message data and header information.

Referring now to Figure 4, in one embodiment, file 90 is arranged such that contiguous sectors have blocks of information, here illustrated at 106, with the blocks of information entering from the left as illustrated by arrow 108 and traversing the file from left to right as illustrated by block number 1 entering from the left and block number 13 exiting from the right. It will be understood that the contiguous of blocks and the flow through the file creates a so-called write direction which does not change.

Referring now to Figure 5, it will be seen that the aforementioned QEMT control blocks 100 can be interspersed between other contiguous blocks 106 so that the position of the QEMT control information blocks 100 specify check points at well-known offsets throughout file 90.

The purpose of interspersing the QEMT control blocks at regular intervals is to be able to quickly locate a complete system state containing specific message data and header information by merely specifying the checkpoint number or checkpoint interval, as the case may be. The result is that it is possible to have message data and log record blocks to either side of a control QEMT control block, such that upon identification of a check point interval as being the last to have valid information, the contiguous blocks written after the QEMT block specifies where valid data may be found as well as its identity and location.

As an alternative explanation, the QEMT control blocks provide the recovery process with well-known locations to examine the state of the system.

Referring now to Figure 6, it will be seen that blocks 106 can be utilized as message data blocks as illustrated at 110 or incremental log blocks as illustrated at 112, with blocks 112 corresponding to log record 104 of Figure 3. These log records record state changes to messages in contiguous downstream blocks. Note, the control block provides only some known point for the beginning of the examination of the file, whereas the log records provide information concerning individual messages in the file.

Referring back to Figure 3, it will be appreciated that log record 104 is but one of a number of sequential log records relating to the data having its start point indicated by the QEMT control block. These log records record changes to information in the preceding message block so that a complete history of changes to that particular message block are annotated.

Referring back to Figure 6, it is noted that a given number of message blocks are bounded by QEMT control blocks which specify additional message data blocks that have occurred after the check point. Within this sector are transactional log records 112. It will be seen that log record T_1 can describe a change in any one of the message blocks. As can be seen from arrow 114, the information flow is from left to right. This being the case, transactional log record T_1 can describe the state change for any message in the system, which could be an acknowledgment that the message has been received and is no longer needed to be kept, or that a message has been sent and has not been received or acknowledged, the above reflecting the two pass handshaking technique for the transmission of the secure messages in this type of system.

For instance, transactional log record T_1 could indicate that a new message has been added to the file at that particular point. It will be appreciated that the position of the log record is determined by the write head when the log record is created. Thus, when the log record is created at a time T_1 the write head is at a particular point in the file. However, the log record can refer to transactions and messages anywhere within the whole file structure.

Likewise, transactional log records T_2, T_3 and T_4 reflect that these messages have changed state, with these log records being posted sequentially in time.

It will be appreciated that since the QEMT blocks and the log record blocks are insertable into the single file structure and since the single file structure in one embodiment has a information flow in one direction, it is possible to completely eliminate the two-file structure of the prior art. Moreover, the utilization of the QEMT blocks and the transactional log record blocks permits rapid diagnose of the effect of information interruption, with a way of specifying uniquely those messages which are uncorrupted, while thereafter permitting rapid recovery of the state of the system after failure.

Referring now to Figure 7, the organization of the Queue Entry Management Table header is illustrated at 120. As can be seen, in one embodiment, the header includes the number of segments in a queue file 122, the segment size 124, the QEMT sequence number or timestamp 126, the sequence number of the last log record in the previous segment 128, the current segment number 130, the queue head pointer 132, the queue tail pointer 134, the next available block in the current segment 136, the list of QEMT entries 138, the reservation table of disk blocks 140, the pending transaction list acting as coordinator 142 and the pending transaction list acting as participant 144.

It will be appreciated that the information contained in the header is supporting information for the recovery process.

Referring now to Figure 8, each QEMT entry 138 includes a sequence number 146, a message ID 148, a message operational mode 150, which is either Q_{put} or Q_{get}, the message recipient's node name 152, the message recipient's server name 154, the transaction state 156, which is either "active", "pending", "abort" or "commit", the participant 2 PC vote 158 which is the last known response that was received by the receiver, a set of additional flags 160 and a pointer on-disk location of message 162.

Thus the Queue Entry Management Table provides exact information as to the state of the file and more particularly any queue entry.

Referring now to Figure 9, what will be appreciated is that since a single message is stored in contiguous blocks, the reprocess involves reading contiguous blocks back. As a result, this cuts down on the head movement during a read operation.

In summary, in the prior art doing a read might require the read head to traverse noncontiguous blocks, and therefore take a considerable amount of time. In the subject system since the message are stored in contiguous blocks, only traversing these contiguous blocks is necessary in the read operation. Likewise, for a sequential write operation, the head traverses only a limited amount of the file.

In short, because there is a forward directional flow and wrap around on subsequent writes, the data is arranged in contiguous blocks and the above advantages flow therefrom.

Referring now to Figure 10, the transactional log record 112 of Figure 6 includes a special log record marker 162 in one embodiment. In this embodiment, a sequence number 164 is provided along with a message operational mode 166 which refers to either a Q_{get} or Q_{put} operation. Also included is a message ID 168, a set of operational flags 170, the transactional state 172 which includes "active", "pending", "abort" or "commit" states, the participant 2 PC vote 174 mentioned above and a pointer 176 to on-disk location of message in queue file.

Referring now to Figure 11, what is shown is a flow chart for the write or Q_{put} operation. In this flow chart, upon having started as illustrated at 180, a block queue head pointer 182 effectively puts a lock on the head of the list so that no other user can access the head entry. Thereafter, the system increments the queue head pointer and sets the transaction state to "active read". This indicates the beginning of the handshaking process.

As illustrated at 186, the system then unlocks the queue head pointer and then, as illustrated at 188, reads the messages from the on-disk queue file. Thereafter, the QEM Table is locked as illustrated at 190, whereafter the log record is written as illustrated at 192 and the QEM Table is unlocked as illustrated at 194. The output of the unlock QEM Table step is referred to a decision block 196 which ascertains if the message transmission is transactional. If so, as illustrated at 198, the system runs a two-phase "commit" protocol to permit handshaking. This completes the Q_{put} or write operation.

Referring now to Figure 12, a Q_{get} or read operation is described. As can be seen, upon starting as illustrated at 200, the queue tail pointer is locked as illustrated at 202 and a new QEM entry is created with the queue tail pointer being incremented as illustrated at 204. Thereafter, as illustrated at 206, the system fills in the QEM entry control information and sets the transaction state to "active control". Thereafter, as illustrated at 208, the queue tail pointer is unlocked and the QEM table is locked as illustrated at 210. Subsequently, as illustrated at 212, the system allocates on-disk blocks from the reservation table, with a block crossing a segment boundary being indicated at decision block 214. If the blocks cross segment boundaries, then as illustrated at 216, the system forces the QEMT check point write to disk. This refers to the fact of writing the in-memory copy to disk. It will be appreciated that block 206 updates the in-memory copy of the state of the QEM Table and thus the QEM entry.

After having forced the QEMT check point write to disk as illustrated at 218, the system writes the message data to disk and unlocks the QEM Table. Decision block 220 establishes whether or not the messages is a transactional one and if so, runs a two phase commit protocol as illustrated at 221 to facilitate the handshaking. The end of the write sequence is illustrated at 222. It will be appreciated that block 220 refers to the receiver end running the handshaking protocol.

Referring now to Figure 13, a recovery sequence is illustrated in which, upon starting as illustrated at 230, the queue table pointers are locked as illustrated at 232 and the system thereafter restores global data structure as illustrated at 234. This initializes the state of the system as a whole. Thereafter, as illustrated at 236 the system scans each QEMT in the queue file for the most recent QEMT. This establishes the most recent check point before communications interruption. Thereafter, as illustrated at 238, the system scans the log records in this segment for the log record with the latest QEMT. This means that the log records of the segment are applied to the messages referred to by the entries in the QEMT.

As illustrated at decision block 240, the system ascertains if there are more log records to scan. It will be appreciated that the QEMT specifies the most recent log record subsequent to the pointer associated with the QEMT in question. However, there may indeed be subsequent log records thereafter which need to be scanned. If this is the case, then the system contacts the participant about the transaction state of the message as illustrated at 242. In one instance, the receiver is queried as to whether it has received a message or not. Thereafter, the system invokes a two-phase "commit" protocol to resolve the transaction as illustrated at 244. This refers to the fact that the handshaking process is a two pass process. Thus, whatever state that one receives back from the receiver is used to restart the handshaking process at the point at which the system had failed.

As can be seen at 246, the system updates the state of the reservation table and determines a new file pointer position. Thus, the entire section is scanned to update the state of reservation table 140, with the determination of the new file pointer position being established by the current segment number 130 and the next available block in the current segment 136.

As illustrated at 248, the system then writes out the new QEMT state to the disk at which point the recovery is complete as illustrated at 250.

As described hereinafter, the programming listing for one embodiment of the subject invention written in C follows:

```
/*
 * Copyright(C)1995 MITSUBISHI ELECTRIC ITA.  ALL RIGHTS RESERVED.
 * UNPUBLISHED -- RIGHTS RESERVED UNDER THE COPYRIGHT
 * LAWS OF THE UNITED STATES.  USE OF A COPYRIGHT NOTICE
 * IS PRECAUTIONARY ONLY AND DOES NOT IMPLY PUBLICATION
 * OR DISCLOSURE.
 *
 * THIS SOFTWARE CONTAINS CONFIDENTIAL INFORMATION AND
 * TRADE SECRETS OF MITSUBISHI ELECTRIC ITA.  USE, DISCLOSURE,
 * OR REPRODUCTION IS PROHIBITED WITHOUT THE PRIOR
 * EXPRESS WRITTEN PERMISSION OF MITSUBISHI ELECTRIC ITA.
 *
 * OpenMQ
 *   Module: qmain.c
 *   Author: David Wong 9/8/95
 *
 * ==============================================================
 *
 * Command Line Syntax:
 *
 *  C> qserv [-c] [-lq <name>] [-pq <name>] [-p <pathname>] \
 *           [-t <threads>] [-n <#qentries>] [-e <#qextents>] \
 *           [-s <#segments>]
 *
 *  where []  = Optional flag.
 *        -c  = Create new queue file mode.  Will overwrite
 *              existing queue data and state files.  If not
 *              set, then it is a restart caes.
 *        -lq = Logical queue name.
 *        -pq = Physical queue server name.
 *        -p  = Path of queue data and state files.
 *        -t  = Number of queue server threads to allocate.
 *        -n  = Number of queue entries to allocate for new queue.
 *        -e  = Max number of additional extries that the queue
 *              can be extended to; not supported currently.
 *        -s  = Number of segments to allocate for new queue.
 *
 * ==============================================================
 */


// Include Files
//#include <memory.h>
#include "qlib.h"
#include "qserv.h"
#include "qadmin.h"


lpQEMT    MQEMT;
lpQSTR    MQstate;
lpOPSTATS MQops;
lpMTLIST Pending_TXNs = NULL;
int       BLOCKS_PER_SEG;
int       SEGMENT_SIZE;
int       TOT_MSG_BLOCKS;
int       QEMT_Size;
int       QEMT_Seq_No = 0;
int       LREC_Seq_No = 0;
int       NUM_SEGS;
int       MAX_ELMS;
int      shdn_flag=0;
int      holey_entries=0;
unsigned long QEME_TS = 1;
lpLPG_TS_STR Last_Pending_Get;
lpLRCLST     Active_LREC_List;


#define DefaultPath      "C:\\Q\\QSERV"
#define DefaultLogQue    "Q1"
#define DefaultPhyQue    "QS1"
#define Op_State         "_Op_State"
#define DefaultElms 100
#define DefaultExt   0
#define DefaultSegs  10
#define DefaultThrs  1

#define Format "C> qserv [-c] [-lq <name>] [-pq <name>] [-p <pathname>] [-t <thr


void Process_Msg(
   lpMSTR    arg)
{
   HANDLE    hQSHDN_Event;
   lpQHANDLE qhandle;
   lpTSTR    thr_str;
   SMBUF     ENQ;

   Diag("Queue Server Thread No. %d starting up...",arg->thr_no);

   // Open the queue for getting
   if (!(qhandle = Qopen(arg->physical,GET_MODE,0,0,0,-1,0)))
     Fail("QServ_main: could not open queue server %s",arg->physical);

   thr_str = (TSTR *)malloc(sizeof(TSTR));

   while ((thr_str->Que_File_Handle=Open_Queue_File(arg->qname))
                  == INVALID_HANDLE_VALUE);

   while (!shdn_flag && (QSUCCESS ==
         QlistenBeforeReply(qhandle,&ENQ.msgh,ENQ.mdata,MAXMSGDATA)))
    {
      strcpy(thr_str->logical,arg->logical);
      strcpy(thr_str->physical,arg->physical);
      strcpy(thr_str->qname,arg->qname);
      strcpy(thr_str->qstate,arg->qstate);
      thr_str->qhandle = qhandle;
      thr_str->lpsmbuf = &ENQ;

      Diag("");
      Diag("Request Serviced by QS Thread No. %d",arg->thr_no);

      switch(ENQ.msgh.mode)
       {
        case PUT_MODE:
          Diag("DiskQ(%s) Queue Server in PUT_MODE",arg->physical);
          QS_QPut(thr_str);
          break;

        case REQUEST_MODE:
          Diag("DiskQ(%s) Queue Server in GET_MODE",arg->physical);
          QS_QGet(thr_str);
          break;

        case ABORT_MODE:
        case COMMIT_MODE:
          if (ENQ.msgh.mode == ABORT_MODE)
            Diag("DiskQ(%s) Queue Server in ABORT_MODE",arg->physical);
            else Diag("DiskQ(%s) Queue Server in COMMIT_MODE",arg->physical);
          QS_QCommit(thr_str);
          break;

        case ADMINREQ_MODE:
          Diag("DiskQ(%s) Queue Server in ADMINREQ_MODE",arg->physical);
          QS_QAdmin(thr_str);
          if (shdn_flag)
           {
            QreplyAfterListen(qhandle,ADMINREP_MODE,SUB_MODE_OK,0,0,0);

            // Qclose(NULL,arg->physical);
            Qclose(&qhandle,0);

            hQSHDN_Event = OpenEvent(EVENT_MODIFY_STATE,TRUE,QSHDN_EVENT);
            if (!hQSHDN_Event)
              Diag("QS_Admin: Can't OpenEvent for QS Shutdown");

            if (SetEvent(hQSHDN_Event) == FALSE)
              Diag("QS_Admin: Can't SetEvent for QS Shutdown");
            return;
           }
          break;

        default:
          Diag("DiskQ(%s): Unexpected Mode=%d",arg->physical,ENQ.msgh.mode);
          QreplyAfterListen(qhandle,ACK_MODE,SUB_MODE_BAD_REQ,0,0,0);
       }

/*
Diag("head = %d, tail = %d, pgets = %d, pputs = %d, segment = %d, block = %d",
     MQEMT->que_hd_ptr, MQEMT->que_tl_ptr,
     MQops->pending_gets, MQops->pending_puts,
     MQEMT->next_avail_block.segment, MQEMT->next_avail_block.block);

print_RST();
print_Active_Log_List();
print_QEME_txn_states();
*/

      // Sleep(3000);
     }

}



void main(
   int    argc,
   CHAR **argv)
{
```

```
HANDLE    Que_File_Handle;
HANDLE    Que_State_Handle;
HANDLE    hQHD,hQTL,hQEMT;
HANDLE    hQEME,hLPG;
HANDLE    hMQstate,hMQops;
HANDLE    hQSHDN,hQSHDN_Event;
HANDLE    hTSM[MAX_QSERV_THREADS];
DWORD     idTSM[MAX_QSERV_THREADS];
DWORD     dwPointer;
DWORD     dwBytesRead,dwBytesWritten;
lpMSTR    thr_arg[MAX_QSERV_THREADS];
BOOL      Return_Status;
long      temp_time;
int       seg_no;
int       i,status;
int       newq=0;
CHAR      logical[NAMESIZE]=DefaultLogQue;
CHAR      physical[NAMESIZE]=DefaultPhyQue;
CHAR      path[QUE_FILE_SIZE]=DefaultPath;
CHAR      quefile[QUE_FILE_SIZE];
CHAR      qstate[QUE_FILE_SIZE];
CHAR      buf[255];
int       max_elms=DefaultElms;
int       ext_elms=DefaultExt;
int       num_segs=DefaultSegs;
int       num_threads=DefaultThrs;
int       max_txns_per_seg;


i = 1;
if ((argc > 1) && (!strcmp(argv[i],"-c")))
 {
  newq = 1;
  i++;
 }


while (i < argc)
 {
  if (!strcmp(argv[i],"-lq"))
   {
    i++;
    if (argv[i][0] != '-')
      strcpy(logical,argv[i]);
      else Diag("%s",Format);
   }
   else if (!strcmp(argv[i],"-pq"))
   {
    i++;
    if (argv[i][0] != '-')
      strcpy(physical,argv[i]);
      else Diag("%s",Format);
   }
   else if (!strcmp(argv[i],"-p"))
   {
    i++;
    if (argv[i][0] != '-')
      strcpy(path,argv[i]);
      else Diag("%s",Format);
   }
```

```
    else if (!strcmp(argv[i],"-t"))
    {
     i++;
     if (argv[i][0] != '-')
       num_threads = atoi(argv[i]);
       else Diag("%s",Format);
    }
    else if (!strcmp(argv[i],"-n"))
    {
     i++;
     if (argv[i][0] != '-')
       max_elms = atoi(argv[i]);
       else Diag("%s",Format);
    }
    else if (!strcmp(argv[i],"-e"))
    {
     i++;
     if (argv[i][0] != '-')
       ext_elms = atoi(argv[i]);
       else Diag("%s",Format);
    }
    else if (!strcmp(argv[i],"-s"))
    {
     i++;
     if (argv[i][0] != '-')
       num_segs = atoi(argv[i]);
       else Diag("%s",Format);
    }

   i++;
  }

sprintf(quefile,"%s\\%s.dat",path,physical);
sprintf(qstate,"%s\\%s.sta",path,physical);


// Check to see if QNETD is running.
if (!AttachSharedMemory())
  Fail("Error: QNETD is not running.");


// Create mutexes for protected data structures.
hQHD = CreateMutex(NULL,FALSE,QUE_HD_PTR_LOCK);

if (!hQHD)
  Diag("CreateMutex for Que Head Pointer lock failed.");

hQTL = CreateMutex(NULL,FALSE,QUE_TL_PTR_LOCK);

if (!hQTL)
  Diag("CreateMutex for Que Head Pointer lock failed.");

hQEME = CreateMutex(NULL,FALSE,QEME_TS_GEN_LOCK);

if (!hQEME)
  Diag("CreateMutex for QEME_TS Lock failed.");

hLPG = CreateMutex(NULL,FALSE,LPG_TS_GEN_LOCK);
```

```
if (!hLPG)
  Diag("CreateMutex for LPG_TS Lock failed.");

hQEMT = CreateMutex(NULL,FALSE,QEMT_LOCK);

if (!hQEMT)
  Diag("CreateMutex for QEM Table lock failed.");


hMQstate = CreateMutex(NULL,FALSE,MQstate_LOCK);

if (!hMQstate)
  Diag("CreateMutex for MQstate file lock failed.");

hMQops = CreateMutex(NULL,FALSE,MQops_LOCK);

if (!hMQops)
  Diag("CreateMutex for MQops stats lock failed.");

hQSHDN = CreateMutex(NULL,FALSE,QSHDN_LOCK);

if (!hQSHDN)
  Diag("CreateMutex for QShutdown lock failed.");

hQSHDN_Event = CreateEvent(NULL,TRUE,FALSE,QSHDN_EVENT);

if (!hQSHDN_Event)
  Diag("CreateEvent for QShutdown event failed.");


// Allocate structure for queue shutdown
// state and recovery statistics.
MQstate = (QSTR *)malloc(sizeof(QSTR));


// If new queue file get max_elms and num_segs
// from command line, else find most recent
// QEM table from disk.  Policy is to delete
// existing quefile and qstate file when the
// -c flag is used.
if (newq)
 {
  Que_File_Handle = Create_Queue_File(quefile,qstate,
                                      max_elms,num_segs);

  while (Que_File_Handle == INVALID_HANDLE_VALUE)
   {
    sprintf(buf,"del %s",quefile);
    system(buf);
    sprintf(buf,"del %s",qstate);
    system(buf);

    Que_File_Handle = Create_Queue_File(quefile,qstate,
                                        max_elms,num_segs);
   }

  MAX_ELMS = max_elms;
  NUM_SEGS = num_segs;
 }
 else
```

```
      {
       Que_File_Handle = Open_Queue_File(quefile);

       status = Find_Latest_QEM(Que_File_Handle,&MQEMT,&seg_no);

//  print_QEMT(MQEMT,1);

       Return_Status = CloseHandle(Que_File_Handle);

       MAX_ELMS = MQEMT->max_entries;
       NUM_SEGS = MQEMT->num_segs;

       Update_QEME_TS();
      }


    Update_Globals();

    Last_Pending_Get = (LPG_TS_STR *)malloc(sizeof(LPG_TS_STR));

    Last_Pending_Get->qeme_no = NIL;
    Last_Pending_Get->timestamp = 1;

    // Theoretically possible to have MAX_ELMS
    // GET operations and txn termination log
    // records in same segment?

    max_txns_per_seg = MAX_ELMS;
    // max_txns_per_seg = 2*MAX_ELMS;
    // max_txns_per_seg = (int)ceil((double)(MAX_ELMS/NUM_SEGS));

    Active_LREC_List = (LRCLST *)malloc(sizeof(LRCLST));

    Active_LREC_List->max_txns_per_seg = max_txns_per_seg;

    Active_LREC_List->address =
                      (int *)malloc(max_txns_per_seg*sizeof(int));

    // Reinitialize the active log record list.
    Init_Active_LREC_List();

    Que_State_Handle = Open_Queue_File(qstate);


    // If new queue file, or if a restart clear,
    // case, initialize global data structures
    // and create new initial QEM Table.
    if (newq)
      {
       status = Create_QEMT(&MQEMT,MAX_ELMS);
       status = Init_QEMT(MQEMT,MAX_ELMS,ext_elms,NUM_SEGS);
       status = Write_QEMT(Que_File_Handle,0,MQEMT);

       MQstate->svr_state = QUEUE_ACTIVE;
       MQstate->num_restarts = 0;
       MQstate->num_recov_tries = 0;
       temp_time = time(&MQstate->first_start_time);
       temp_time = time(&MQstate->last_restart_time);
       MQstate->last_recov_time = 0;
      }
```

```
else       // Restart case.
{
 // see if queue server shutdown cleanly last time

 dwPointer = SetFilePointer(Que_State_Handle,
                            0,NULL,FILE_BEGIN);

 Return_Status = ReadFile(Que_State_Handle,
                          MQstate,sizeof(QSTR),
                          &dwBytesRead,NULL);

 if ((Return_Status == FALSE) || (dwBytesRead != sizeof(QSTR)))
  {
   if (Return_Status == FALSE)
     Diag("status = FALSE");
   Fail("Main: Problem reading MQstate; BytesRead = %d",dwBytesRead);
  }

 MQstate->num_restarts++;
 temp_time = time(&MQstate->last_restart_time);

 // If queue server did not shutdown cleanly
 // last time, then run the recovery module.
 if (MQstate->svr_state == QUEUE_ACTIVE)
  {
   MQstate->num_recov_tries++;
   MQstate->last_recov_time = temp_time;

   dwPointer = SetFilePointer(Que_State_Handle,
                              0,NULL,FILE_BEGIN);
   Return_Status = WriteFile(Que_State_Handle,
                             MQstate,sizeof(QSTR),
                             &dwBytesWritten,NULL);
   if ((Return_Status == FALSE) ||
       (dwBytesWritten != sizeof(QSTR)))
    {
     if (Return_Status == FALSE)
       Diag("status = FALSE");
     Fail("Main: Problem writing MQstate; BytesWrtten = %d",dwBytesWritten);
    }

   QRecov(Que_File_Handle,seg_no);

   MQstate->num_recov_tries = 0;
   MQstate->last_recov_time = 0;
  }
  else  // Clean shutdown last time; simple restart case.
  {
   MQstate->svr_state = QUEUE_ACTIVE;
   MQEMT->qget_state = ENABLED;
   MQEMT->qput_state = ENABLED;

   if (!Reconstruct_RST())
     Fail("Restart procedure failed: cannot reconstruct RST");
  }
}


// update the queue server state file
dwPointer = SetFilePointer(Que_State_Handle,
                                 0,NULL,FILE_BEGIN);

  Return_Status = WriteFile(Que_State_Handle,
                            MQstate,sizeof(QSTR),
                            &dwBytesWritten,NULL);

  if ((Return_Status == FALSE) || (dwBytesWritten != sizeof(QSTR)))
    {
     if (Return_Status == FALSE)
       Diag("status = FALSE");
     Fail("Main: Problem writing MQstate; BytesWrtten = %d",dwBytesWritten);
    }

  Return_Status = CloseHandle(Que_File_Handle);
  Return_Status = CloseHandle(Que_State_Handle);


  // initialize RT statistical counters
  MQops = (OPSTATS *)malloc(sizeof(OPSTATS));
  MQops->num_gets = 0;
  MQops->num_puts = 0;
  MQops->num_aborts = 0;
  MQops->num_commits = 0;
  MQops->pending_gets = 0;
  MQops->pending_puts = 0;


  // do initial close to clear stale buffers
  Qclose(NULL,physical);

  // spawn off worker threads
  i = 0;
  while(i < num_threads)
   {
    thr_arg[i] = (MSTR *)malloc(sizeof(MSTR));

    strcpy(thr_arg[i]->logical,logical);
    strcpy(thr_arg[i]->physical,physical);
    strcpy(thr_arg[i]->qname,quefile);
    strcpy(thr_arg[i]->qstate,qstate);
    thr_arg[i]->thr_no = i;

    hTSM[i] = CreateThread(NULL,0,
                           (LPTHREAD_START_ROUTINE) Process_Msg,
                           thr_arg[i],0,&idTSM[i]);
    i++;
   }


  // Wait for shutdown to be signalled by a worker thread.
  // Add code later to check on status of worker threads.

  hQSHDN_Event = OpenEvent(SYNCHRONIZE,TRUE,QSHDN_EVENT);
  if ((status = WaitForSingleObject(hQSHDN_Event,INFINITE)) !=
                                              WAIT_OBJECT_0)
    Diag("QS_Main: Synch wait for QS Shutdown Event failed");

/*
```

```
   // Clear memory buffers held by worker threads.
   // Need to wait for remaining replies to be sent
   // back before issuing Qclose.

   Sleep(3000);
   Qclose(NULL,physical);
*/


   // Fetch all locks first.

   hQHD = OpenMutex(SYNCHRONIZE,FALSE,QUE_HD_PTR_LOCK);
   if (!hQHD)
     Diag("QS_Main: Can't OpenMutex for que head pointer lock");
   if ((status = WaitForSingleObject(hQHD,QUE_LOCK_TIMEOUT)) !=
                                          WAIT_OBJECT_0)
     Diag("QS_Main: Synch wait for que head pointer lock failed");

   hQTL = OpenMutex(SYNCHRONIZE,FALSE,QUE_TL_PTR_LOCK);
   if (!hQTL)
     Diag("QS_Main: Can't OpenMutex for que tail pointer lock");
   if ((status = WaitForSingleObject(hQTL,QUE_LOCK_TIMEOUT)) !=
                                          WAIT_OBJECT_0)
     Diag("QS_Main: Synch wait for que tail pointer lock failed");

   hQEME = OpenMutex(SYNCHRONIZE,FALSE,QEME_TS_GEN_LOCK);
   if (!hQEME)
     Diag("QS_Main: Can't OpenMutex for QEME_TS Lock");
   if ((status = WaitForSingleObject(hQEME,QUE_LOCK_TIMEOUT)) !=
                                          WAIT_OBJECT_0)
     Diag("QS_Main: Synch wait for QEME_TS Lock failed");

   hLPG = OpenMutex(SYNCHRONIZE,FALSE,LPG_TS_GEN_LOCK);
   if (!hLPG)
     Diag("QS_Main: Can't OpenMutex for LPG_TS Lock");
   if ((status = WaitForSingleObject(hLPG,QUE_LOCK_TIMEOUT)) !=
                                          WAIT_OBJECT_0)
     Diag("QS_Main: Synch wait for LPG_TS Lock failed");

   hQEMT = OpenMutex(SYNCHRONIZE,FALSE,QEMT_LOCK);
   if (!hQEMT)
     Diag("QS_Main: Can't OpenMutex for QEM Table lock");
   if ((status = WaitForSingleObject(hQEMT,QUE_LOCK_TIMEOUT)) !=
                                          WAIT_OBJECT_0)
     Diag("QS_Main: Synch wait for QEM Table lock failed");

   hQSHDN = OpenMutex(SYNCHRONIZE,FALSE,QSHDN_LOCK);
   if (!hQSHDN)
     Diag("QS_Main: Can't OpenMutex for QSHDN lock");
   if ((status = WaitForSingleObject(hQSHDN,QUE_LOCK_TIMEOUT)) !=
                                           WAIT_OBJECT_0)
     Diag("QS_Main: Synch wait for QSHDN lock failed");

   hMQstate = OpenMutex(SYNCHRONIZE,FALSE,MQstate_LOCK);
   if (!hMQstate)
     Diag("QS_Main: Can't OpenMutex for MQstate lock");
   if ((status = WaitForSingleObject(hMQstate,QUE_LOCK_TIMEOUT)) !=
                                              WAIT_OBJECT_0)
     Diag("QS_Main: Synch wait for MQstate lock failed");
   // write out last QEM Table before shutting down
   Que_File_Handle = Open_Queue_File(quefile);

   status = Gen_QEM_Seq_No(&MQEMT->qem_sn);

   MQEMT->next_avail_block.segment =
            (MQEMT->next_avail_block.segment+1)%MQEMT->num_segs;
   MQEMT->next_avail_block.block = 0;

   status = Write_QEMT(Que_File_Handle,
                       MQEMT->next_avail_block.segment,
                       MQEMT);

   Return_Status = CloseHandle(Que_File_Handle);


   // Write out graceful shutdown state
   MQstate->svr_state = QUEUE_SHUTDOWN;

   Que_State_Handle = Open_Queue_File(qstate);

   dwPointer = SetFilePointer(Que_State_Handle,
                              0,NULL,FILE_BEGIN);

   Return_Status = WriteFile(Que_State_Handle,
                             MQstate,sizeof(QSTR),
                             &dwBytesWritten,NULL);

   if ((Return_Status == FALSE) || (dwBytesWritten != sizeof(QSTR)))
    {
     if (Return_Status == FALSE)
       Diag("status = FALSE");
     Fail("QS_Main: Problem writing MQstate; BytesWritten = %d",
                    dwBytesWritten);
    }

   Return_Status = CloseHandle(Que_State_Handle);

   Say("Queue Server %s shuts down gracefully",physical);

   Sleep(3000);

// print_QEMT(MQEMT,1);

   // keep the locks to avoid worker
   // threads from writing to queue

/*
   ReleaseMutex(hMQstate);
   ReleaseMutex(hQSHDN);
   ReleaseMutex(hQEMT);
   ReleaseMutex(hLPG);
   ReleaseMutex(hQEME);
   ReleaseMutex(hQTL);
   ReleaseMutex(hQHD);
*/
}
```

```
User: root
Host: bunny
Class: bunny
Job: stdin
```

```
/*
 * Copyright(C)1995 MITSUBISHI ELECTRIC ITA.  ALL RIGHTS RESERVED.
 * UNPUBLISHED -- RIGHTS RESERVED UNDER THE COPYRIGHT
 * LAWS OF THE UNITED STATES.  USE OF A COPYRIGHT NOTICE
 * IS PRECAUTIONARY ONLY AND DOES NOT IMPLY PUBLICATION
 * OR DISCLOSURE.
 *
 * THIS SOFTWARE CONTAINS CONFIDENTIAL INFORMATION AND
 * TRADE SECRETS OF MITSUBISHI ELECTRIC ITA.  USE, DISCLOSURE,
 * OR REPRODUCTION IS PROHIBITED WITHOUT THE PRIOR
 * EXPRESS WRITTEN PERMISSION OF MITSUBISHI ELECTRIC ITA.
 *
 * OpenMQ
 *   Module: qrecov.c
 *   Author: David Wong 10/15/95
 */


// Include Files
#include "qlib.h"
#include "qserv.h"
#include "qadmin.h"


extern lpQEMT    MQEMT;
extern lpQSTR    MQstate;
extern lpMTLIST  Pending_TXNs;
extern int       BLOCKS_PER_SEG;
extern int       SEGMENT_SIZE;
extern int       TOT_MSG_BLOCKS;
extern int       QEMT_Size;
extern int       QEMT_Seq_No;
extern int       LREC_Seq_No;
extern int       NUM_SEGS;
extern int       MAX_ELMS;
extern unsigned long QEME_TS;



//
// Recovery module basic algorithm:
//   - lock queue table and head/tail ptrs
//   - restore global data structures
//   - fetch log records
//   - determine state
//   - contact client
//   - invoke (partial) 2PC protocol
//   - rollback bad cases
//   - commit good cases
//   - fix up reservation table
//   - determine file ptr position
//   - write new QEM Table out to disk
//



void QRecov(
   HANDLE  Que_File_Handle,
   int      seq_no)
{
   HANDLE   hQHD,hQTL,hQEMT;
   lpMTLIST This_TXN;
   LREC     log_rec;
   BOOL     Return_Status;
   DWORD    dwPointer;
   DWORD    dwBytesRead;
   SN       lrec_no;
   int      i,status;


   hQHD = OpenMutex(SYNCHRONIZE,FALSE,QUE_HD_PTR_LOCK);
   if (!hQHD)
     Diag("QS_QRecov: Can't OpenMutex for queue head pointer lock");
   if ((status = WaitForSingleObject(hQHD,QUE_LOCK_TIMEOUT)) != WAIT_OBJECT_0)
     Diag("QS_QRecov: Synch wait for queue head pointer lock failed");


   hQTL = OpenMutex(SYNCHRONIZE,FALSE,QUE_TL_PTR_LOCK);
   if (!hQTL)
     Diag("QS_QRecov: Can't OpenMutex for queue tail pointer lock");
   if ((status = WaitForSingleObject(hQTL,QUE_LOCK_TIMEOUT)) != WAIT_OBJECT_0)
     Diag("QS_QRecov: Synch wait for queue tail pointer lock failed");


   hQEMT = OpenMutex(SYNCHRONIZE,FALSE,QUE_TL_PTR_LOCK);
   if (!hQEMT)
     Diag("QS_QRecov: Can't OpenMutex for QEM table lock");
   if ((status = WaitForSingleObject(hQEMT,QUE_LOCK_TIMEOUT)) != WAIT_OBJECT_0
     Diag("QS_QRecov: Synch wait for QEM table lock failed");


   // set file pointer to start of log records
   dwPointer = SetFilePointer(Que_File_Handle,
                              seq_no*SEGMENT_SIZE,
                              NULL,FILE_BEGIN);

   dwPointer = SetFilePointer(Que_File_Handle,
                              QEMT_Size,
                              NULL,FILE_CURRENT);


   // Initialize log record timestamp, which is recorded
   // in QEMT.  Next, start reading the log records and
   // construct extended pending txn list based on one
   // fetched from QEMT.
   i = 0;
   lrec_no.timestamp = MQEMT->lrec_sn.timestamp;
   lrec_no.counter = MQEMT->lrec_sn.counter;

   while(i<BLOCKS_PER_SEG)
    {
     Return_Status = ReadFile(Que_File_Handle,
                              &log_rec,sizeof(LREC),
                              &dwBytesRead,NULL);

     i += LOG_REC_BLOCKS;

     if (log_rec.marker != LRMARK)
    {
     dwPointer = SetFilePointer(Que_File_Handle,
                               (MSG_BODY_BLOCKS+LOG_REC_BLOCKS)*BLOCK,
                               NULL,FILE_CURRENT);
     i += (MSG_BODY_BLOCKS+LOG_REC_BLOCKS);
    }
    else if (Bigger_Seq_No(&log_rec.seq_no,&lrec_no))
    {
     // only need to be concern about log records
     // with increasing timestamps.
     lrec_no.timestamp = log_rec.seq_no.timestamp;
     lrec_no.counter = log_rec.seq_no.counter;

     // find txn in pending list
     Return_Status = Find_MTlist(Pending_TXNs,&This_TXN,log_rec.mid);

     switch (log_rec.txn_state)
      {
       case PENDING:
         if (Return_Status == TRUE)
          {
           Return_Status = Find_Tlist(This_TXN->ops,log_rec.qeme_no);
           if (Return_Status == FALSE)
             Add_Tlist(&This_TXN->ops,log_rec.qeme_no);
          }
          else
          {
           Add_MTlist(&Pending_TXNs,&This_TXN,log_rec.mid);
           Add_Tlist(&This_TXN->ops,log_rec.qeme_no);
          }
         break;

       case ABORT:
       case COMMIT:
         if (Return_Status == TRUE)
          {
           Return_Status = Del_MTlist(&Pending_TXNs,log_rec.mid);
           if (Return_Status == FALSE)
             Diag("QS_Recovery: Error deleting TID %d from Host %d",
                       log_rec.mid.tid,log_rec.mid.host);
          }
         break;

       case EMPTY:
         break;

       default:
           Diag("QRecov: No such txn state");
         break;
      }
    }
  }

// now, resolve pending txns
QR_Resolve_PTL(Que_File_Handle);


// Flush out reconstructed QEMT to disk.
// No need to find last known QEME offset.
```

```
   MQEMT->next_avail_block.segment =
           (MQEMT->next_avail_block.segment+1)%MQEMT->num_segs;

   MQEMT->next_avail_block.block = 0;

   status = Gen_QEM_Seq_No(&MQEMT->qem_sn);
   status = Write_QEMT(Que_File_Handle,
                       MQEMT->next_avail_block.segment,
                       MQEMT);


   // Reconstruct reservation table.
   if (!Reconstruct_RST())
    {
     Diag("Recovery procedure failed: cannot reconstruct RST");
     ReleaseMutex(hQEMT);
     ReleaseMutex(hQHD);
     ReleaseMutex(hQTL);
     return;
    }


   MQEMT->qget_state = ENABLED;
   MQEMT->qput_state = ENABLED;


   ReleaseMutex(hQEMT);
   ReleaseMutex(hQHD);
   ReleaseMutex(hQTL);
}



void QR_Resolve_PTL(
   HANDLE  Que_File_Handle)
{
   lpQHANDLE qhandle;
   lpMTLIST This_TXN;
   lpTLIST This_TXN_ops;
   lpQEME  lpqeme;
   MSGH    msgh,msgh2;
   int     status;

   This_TXN = Pending_TXNs;

   while(This_TXN != NULL)
    {
     This_TXN_ops = This_TXN->ops;

     status = Cycle_QEME(This_TXN_ops->qeme_no,&lpqeme);

     status = Retrieve_Msg_Hdr(Que_File_Handle,lpqeme->offset,&msgh);

     // get buffer for communication with QNETD
     if (!(qhandle = QopenReply(0,&msgh,0,"QNETD",&status)))
       Fail("QS_QRecov: could not open connection to QNETD");

     // check with Derek on proper use of inquiry msg
```

```
      if (QSUCCESS == QsendAndReceive(qhandle,ADMINREQ_MODE,QNETD_TRAN_INQ,
                                      0,0,0,0,0,0,&msgh2))
       {
        while(This_TXN_ops != NULL)
          {
           if (msgh2.sub_mode == Q_COMMIT)
             QR_Resolve_TXN_Op(This_TXN_ops->qeme_no,COMMIT);
             else QR_Resolve_TXN_Op(This_TXN_ops->qeme_no,ABORT);

           This_TXN_ops = This_TXN_ops->next;
          }
       }
       else
       {
        Fail("QS_QRecov: no response from QNETD");
        Fail("QS_QRecov: Queue Server ABORTs Transaction %d",This_TXN->mid.tid);

        while(This_TXN_ops != NULL)
          {
           QR_Resolve_TXN_Op(This_TXN_ops->qeme_no,ABORT);
           This_TXN_ops = This_TXN_ops->next;
          }

       }

      This_TXN = This_TXN->next;

     }
}



void QR_Resolve_TXN_Op(
   int      qeme_no,
   short    mode)
{
   lpQEME   lpqeme,lpqeme2;
   lpQEME   hd_qeme,tl_qeme;
   int      qeme_no2;
   int      status;

   status = Cycle_QEME(qeme_no,&lpqeme);
   lpqeme->txn_state = mode;
   lpqeme->vote = mode;

   // if txn is ABORTed, we need to rollback either
   // the queue head or tail pointer depending on
   // whether it's a GET/PUT op

   if (mode == COMMIT)
    {
     if (lpqeme->mode == GET_MODE)
       {
        lpqeme->txn_state = EMPTY;
        lpqeme->vote = EMPTY;
       }
    }
    else     // mode == ABORT
```

```
{
 if (lpqeme->mode == PUT_MODE)
  {
   lpqeme->txn_state = EMPTY;
   lpqeme->vote = EMPTY;

   if (qeme_no == MQEMT->que_hd_ptr)
    {
     if (MQEMT->que_hd_ptr == MQEMT->que_tl_ptr)
      {
       MQEMT->que_hd_ptr = NIL;
       MQEMT->que_tl_ptr = NIL;
      }
       else MQEMT->que_hd_ptr = (MQEMT->que_hd_ptr+1)%MQEMT->max_entries;
    }
    else if (qeme_no == MQEMT->que_tl_ptr)
    {
     qeme_no2 = qeme_no;
     lpqeme2 = lpqeme;
     while ((lpqeme2->txn_state == EMPTY) &&
            (qeme_no2 != MQEMT->que_hd_ptr))
      {
       qeme_no2 = (MQEMT->max_entries+qeme_no2-1)%MQEMT->max_entries;
       status = Cycle_QEME(qeme_no2,&lpqeme2);
      }

     if ((qeme_no2 == MQEMT->que_hd_ptr) &&
         (lpqeme2->txn_state == EMPTY))
      {
       MQEMT->que_hd_ptr = NIL;
       MQEMT->que_tl_ptr = NIL;
      }
       else MQEMT->que_tl_ptr = qeme_no2;
    }
  }
  else     // failed GET operation
  {
   // reset the op flag to a PUT
   lpqeme->mode = PUT_MODE;

   lpqeme->txn_state = COMMIT;
   lpqeme->vote = COMMIT;

   if (MQEMT->que_hd_ptr == NIL)
    {
     MQEMT->que_hd_ptr = qeme_no;
     MQEMT->que_tl_ptr = MQEMT->que_hd_ptr;
    }
    else
    {
     status = Cycle_QEME(MQEMT->que_hd_ptr,&hd_qeme);
     status = Cycle_QEME(MQEMT->que_tl_ptr,&tl_qeme);

     if (lpqeme->timestamp < hd_qeme->timestamp)
       MQEMT->que_hd_ptr = qeme_no;
       else if (lpqeme->timestamp > tl_qeme->timestamp)
         MQEMT->que_tl_ptr = qeme_no;
    }
  }
}
```

```
}


int Reconstruct_RST()
{
   lpRSTSEG lprstseg;
   lpQEME   lpqeme;
   int     *msg_block;
   int      qeme_no;
   int      i,j,status;

   // Initialize the reservation table first.
   lprstseg = MQEMT->rst_ptr->seg_ptr;
   for (i=0; i<NUM_SEGS; i++)
    {
     msg_block = lprstseg->msg_block;
     for (j=0; j<MQEMT->rst_ptr->msgs_per_seg; j++)
      {
       *msg_block = NIL;
        msg_block++;
      }
     lprstseg++;
    }

   // Then, reconstruct it based on current QEMT state.
   qeme_no = MQEMT->que_hd_ptr;
   while (qeme_no != MQEMT->que_tl_ptr)
    {
     status = Cycle_QEME(qeme_no,&lpqeme);
     if (!Add_RST_Entry(lpqeme->offset.segment,lpqeme->offset.block))
      {
       Diag("RST reconstruction phase failed: too many msgs in a segment");
       return(0);
      }

     qeme_no = (qeme_no+1)%MQEMT->max_entries;
    }

   status = Cycle_QEME(qeme_no,&lpqeme);
   if (!Add_RST_Entry(lpqeme->offset.segment,lpqeme->offset.block))
    {
     Diag("RST reconstruction phase failed: too many msgs in a segment");
     return(0);
    }


   for (i=0; i<MQEMT->num_segs; i++)
     Sort_RST_Entries(i);

   return(1);
}
```

```
User: root
Host: bunny
Class: bunny
Job: stdin
```

```
/*
 * Copyright(C)1995 MITSUBISHI ELECTRIC ITA.  ALL RIGHTS RESERVED.
 * UNPUBLISHED -- RIGHTS RESERVED UNDER THE COPYRIGHT
 * LAWS OF THE UNITED STATES.  USE OF A COPYRIGHT NOTICE
 * IS PRECAUTIONARY ONLY AND DOES NOT IMPLY PUBLICATION
 * OR DISCLOSURE.
 *
 * THIS SOFTWARE CONTAINS CONFIDENTIAL INFORMATION AND
 * TRADE SECRETS OF MITSUBISHI ELECTRIC ITA.  USE, DISCLOSURE,
 * OR REPRODUCTION IS PROHIBITED WITHOUT THE PRIOR
 * EXPRESS WRITTEN PERMISSION OF MITSUBISHI ELECTRIC ITA.
 *
 * OpenMQ
 *   Module: qsadmin.c
 *   Author: David Wong 10/15/95
 */


// Include Files
#include "qlib.h"
#include "qserv.h"
#include "qadmin.h"


extern lpQEMT    MQEMT;
extern lpQSTR    MQstate;
extern lpOPSTATS MQops;
extern lpMTLIST Pending_TXNs;
extern int       BLOCKS_PER_SEG;
extern int       SEGMENT_SIZE;
extern int       TOT_MSG_BLOCKS;
extern int       QEMT_Size;
extern int       QEMT_Seq_No;
extern int       LREC_Seq_No;
extern int       MAX_ELMS;
extern int       NUM_SEGS;
extern int       shdn_flag;
extern int       holey_entries;
extern unsigned long QEME_TS;



void QS_QAdmin(
    lpTSTR      thr_arg)
{
    HANDLE  Que_File_Handle;
    HANDLE  Que_State_Handle;
    HANDLE  hQHD,hQTL,hQEMT;
    HANDLE  hMQstate,hMQops,hQSHDN;
    DWORD   dwPointer;
    DWORD   dwBytesRead;
    DWORD   dwBytesWritten;
    MSGH    msgh;
    lpQEME  lpqeme;
    BLAD    tblad;
    CHAR    buffer[MAXMSGDATA];
    BOOL    Return_Status;
    int     done,found;
    int     i,cnt,status;
int       qeme_no;
int       entry_num;
time_t    temp_time;
CHAR      *t0,*t1,*t2,*t3;
int       *int0,*int1,*int2;
short     *sh0,*sh1,*sh2;
LONG      Dist_to_Move;
QMSG      qmsg2;
lpMID           mid;
lpQMSG          qmsg;
lpQADMSEL       seldat;
lpQADMSTATS     lpstats;
lpQADMCTLS      ctls;
lpCNTSTR        cnt_ptr=NULL;
lpMLIST         m1=NULL,m2=NULL,m3=NULL;
lpMID           midstr1=NULL,midstr2=NULL;


Que_File_Handle = thr_arg->Que_File_Handle;

// Might have to fetched all locks if
// we do not assume a quiescent state.

switch(thr_arg->lpsmbuf->msgh.sub_mode)
 {
  case QADM_REQ_STATS:
    lpstats = (QADMSTATS *)malloc(sizeof(QADMSTATS));

    strcpy(lpstats->logical_qname,thr_arg->logical);
    strcpy(lpstats->physical_qname,thr_arg->physical);

    if (thr_arg->qhandle != QUEUE_TEST_VALUE)
     {
      strcpy(lpstats->node_name,SHAREDATA(hostname));
      lpstats->node_address = SHAREDATA(hostip);
     }

    lpstats->max_entries_limit = MQEMT->max_entries_limit;
    lpstats->max_entries = MQEMT->max_entries;

    lpstats->pending_puts = MQops->pending_puts;
    lpstats->pending_gets = MQops->pending_gets;

    if (Check_Queue_Empty(MQEMT) == TRUE)
     {
      lpstats->committed_entries = 0;
      lpstats->holey_entries = 0;
      lpstats->num_free_entries = MQEMT->max_entries;
      lpstats->amt_free_dspace = BLOCKS_PER_SEG*NUM_SEGS*BLOCK;
     }
     else
     {
      status = Find_Num_Entries(&cnt_ptr);
      lpstats->committed_entries = cnt_ptr->committed;

      if (holey_entries == 0)
        lpstats->holey_entries = cnt_ptr->holey;
        else lpstats->holey_entries = holey_entries;

      lpstats->num_free_entries = MQEMT->max_entries-
```

```
         (lpstats->committed_entries+lpstats->pending_gets+
          lpstats->pending_puts+lpstats->holey_entries);

      lpstats->amt_free_dspace =
       (int)(((float)lpstats->num_free_entries/(float)MQEMT->max_entries)*
                 TOT_MSG_BLOCKS*BLOCK);
     }


    lpstats->qget_state = MQEMT->qget_state;
    lpstats->qput_state = MQEMT->qput_state;

    lpstats->num_puts     =  MQops->num_puts;
    lpstats->num_gets     =  MQops->num_gets;
    lpstats->num_aborts   =  MQops->num_aborts;
    lpstats->num_commits  =  MQops->num_commits;

    lpstats->num_restarts = MQstate->num_restarts;
    lpstats->first_start_time = MQstate->first_start_time;
    lpstats->last_restart_time = MQstate->last_restart_time;

    if (thr_arg->qhandle != QUEUE_TEST_VALUE)
      QreplyAfterListen(thr_arg->qhandle,ADMINREP_MODE,SUB_MODE_OK,
                        (char *)lpstats,sizeof(QADMSTATS),0);


    free(lpstats);
    if (cnt_ptr != NULL)
      free(cnt_ptr);
    break;


  case QADM_SET_CONTROLS:
    ctls = (lpQADMCTLS)thr_arg->lpsmbuf->mdata;

    // Enable/Disable QGETs and QPUTs
    hQEMT = OpenMutex(SYNCHRONIZE,FALSE,QEMT_LOCK);
    if (!hQEMT)
      Diag("QS_Admin: Can't OpenMutex for QEM Table lock");
    if ((status = WaitForSingleObject(hQEMT,QUE_LOCK_TIMEOUT)) !=
                                        WAIT_OBJECT_0)
      Diag("QS_Admin: Synch wait for QEM Table lock failed");

    if (ctls->enable_qputs_flag)
      MQEMT->qput_state = ENABLED;
      else MQEMT->qput_state = DISABLED;

    if (ctls->enable_qgets_flag)
      MQEMT->qget_state = ENABLED;
      else MQEMT->qget_state = DISABLED;

    ReleaseMutex(hQEMT);


    if (ctls->stats_reset_flag)
     {
      hMQops = OpenMutex(SYNCHRONIZE,FALSE,MQops_LOCK);
      if (!hMQops)
        Diag("QS_Admin: Can't OpenMutex for MQops stats lock");
      if ((status = WaitForSingleObject(hMQops,QUE_LOCK_TIMEOUT)) !=
                                                 WAIT_OBJECT_0)
        Diag("QS_Admin: Synch wait for MQops stats lock failed");

      MQops->num_puts = 0;
      MQops->num_gets = 0;
      MQops->num_aborts = 0;
      MQops->num_commits = 0;

      ReleaseMutex(hMQops);
     }


  // full reset: clear stat counters and queue entries
  if (ctls->full_reset_flag)
   {
    // Fetch all locks first.
    hQHD = OpenMutex(SYNCHRONIZE,FALSE,QUE_HD_PTR_LOCK);
    if (!hQHD)
      Diag("QS_Admin: Can't OpenMutex for que head pointer lock");
    if ((status = WaitForSingleObject(hQHD,QUE_LOCK_TIMEOUT)) !=
                                                WAIT_OBJECT_0)
      Diag("QS_Admin: Synch wait for que head pointer lock failed");

    hQTL = OpenMutex(SYNCHRONIZE,FALSE,QUE_TL_PTR_LOCK);
    if (!hQTL)
      Diag("QS_Admin: Can't OpenMutex for que tail pointer lock");
    if ((status = WaitForSingleObject(hQTL,QUE_LOCK_TIMEOUT)) !=
                                                WAIT_OBJECT_0)
      Diag("QS_Admin: Synch wait for que tail pointer lock failed");

    hQEMT = OpenMutex(SYNCHRONIZE,FALSE,QEMT_LOCK);
    if (!hQEMT)
      Diag("QS_Admin: Can't OpenMutex for QEM Table lock");
    if ((status = WaitForSingleObject(hQEMT,QUE_LOCK_TIMEOUT)) !=
                                                WAIT_OBJECT_0)
      Diag("QS_Admin: Synch wait for QEM Table lock failed");

    hQSHDN = OpenMutex(SYNCHRONIZE,FALSE,QSHDN_LOCK);
    if (!hQSHDN)
      Diag("QS_Admin: Can't OpenMutex for QSHDN lock");
    if ((status = WaitForSingleObject(hQSHDN,QUE_LOCK_TIMEOUT)) !=
                                                WAIT_OBJECT_0)
      Diag("QS_Admin: Synch wait for QSHDN lock failed");

    hMQstate = OpenMutex(SYNCHRONIZE,FALSE,MQstate_LOCK);
    if (!hMQstate)
      Diag("QS_Admin: Can't OpenMutex for MQstate lock");
    if ((status = WaitForSingleObject(hMQstate,QUE_LOCK_TIMEOUT)) !=
                                                WAIT_OBJECT_0)
      Diag("QS_Admin: Synch wait for MQstate lock failed");

    hMQops = OpenMutex(SYNCHRONIZE,FALSE,MQops_LOCK);
    if (!hMQops)
      Diag("QS_Admin: Can't OpenMutex for MQops stats lock");
    if ((status = WaitForSingleObject(hMQops,QUE_LOCK_TIMEOUT)) !=
                                                WAIT_OBJECT_0)
      Diag("QS_Admin: Synch wait for MQops stats lock failed");


    // Clear queue and pending txn list and
  // write out initialized QEM table.
  Del_MTlist_All(&Pending_TXNs);

  Init_Active_LREC_List();

  status = Init_QEMT(MQEMT,MQEMT->max_entries,
                     MQEMT->max_entries_limit,
                     MQEMT->num_segs);


  status = Write_QEMT(Que_File_Handle,0,MQEMT);

  // Reset all stat counters.
  MQops->num_puts = 0;
  MQops->num_gets = 0;
  MQops->num_aborts = 0;
  MQops->num_commits = 0;
  MQops->pending_puts = 0;
  MQops->pending_gets = 0;

  // Dump out queue state info.
  MQstate->num_restarts = 0;
  MQstate->num_recov_tries = 0;

  temp_time = time(&MQstate->last_restart_time);
  MQstate->last_recov_time = 0;


  Que_State_Handle = Open_Queue_File(thr_arg->qstate);

  dwPointer = SetFilePointer(Que_State_Handle,
                             0,NULL,FILE_BEGIN);

  Return_Status = WriteFile(Que_State_Handle,
                            MQstate,sizeof(QSTR),
                            &dwBytesWritten,NULL);

  if ((Return_Status == FALSE) || (dwBytesWritten != sizeof(QSTR)))
   {
    printf("QS_Admin: Problem writing out MQstate.\n");
    printf("BytesWritten = %d\n",dwBytesWritten);
    if (Return_Status == FALSE)
      printf("status = FALSE\n");
   }

  Return_Status = CloseHandle(Que_State_Handle);

  QreplyAfterListen(thr_arg->qhandle,ADMINREP_MODE,
                    SUB_MODE_OK,0,0,0);

  ReleaseMutex(hMQops);
  ReleaseMutex(hMQstate);
  ReleaseMutex(hQSHDN);
  ReleaseMutex(hQEMT);
  ReleaseMutex(hQTL);
  ReleaseMutex(hQHD);
 }

if (ctls->shutdown_flag)
```

```
   {
    // Fetch all locks first.
    hQHD = OpenMutex(SYNCHRONIZE,FALSE,QUE_HD_PTR_LOCK);
    if (!hQHD)
      Diag("QS_Admin: Can't OpenMutex for que head pointer lock");
    if ((status = WaitForSingleObject(hQHD,QUE_LOCK_TIMEOUT)) !=
                                                    WAIT_OBJECT_0)
      Diag("QS_Admin: Synch wait for que head pointer lock failed");

    hQTL = OpenMutex(SYNCHRONIZE,FALSE,QUE_TL_PTR_LOCK);
    if (!hQTL)
      Diag("QS_Admin: Can't OpenMutex for que tail pointer lock");
    if ((status = WaitForSingleObject(hQTL,QUE_LOCK_TIMEOUT)) !=
                                                    WAIT_OBJECT_0)
      Diag("QS_Admin: Synch wait for que tail pointer lock failed");

    hQEMT = OpenMutex(SYNCHRONIZE,FALSE,QEMT_LOCK);
    if (!hQEMT)
      Diag("QS_Admin: Can't OpenMutex for QEM Table lock");
    if ((status = WaitForSingleObject(hQEMT,QUE_LOCK_TIMEOUT)) !=
                                                    WAIT_OBJECT_0)
      Diag("QS_Admin: Synch wait for QEM Table lock failed");

    hQSHDN = OpenMutex(SYNCHRONIZE,FALSE,QSHDN_LOCK);
    if (!hQSHDN)
      Diag("QS_Admin: Can't OpenMutex for QSHDN lock");
    if ((status = WaitForSingleObject(hQSHDN,QUE_LOCK_TIMEOUT)) !=
                                                    WAIT_OBJECT_0)
      Diag("QS_Admin: Synch wait for QSHDN lock failed");


    // Disable GETs/PUTs and set shutdown flag.
    MQEMT->qget_state = DISABLED;
    MQEMT->qput_state = DISABLED;

    shdn_flag = 1;

    ReleaseMutex(hQSHDN);
    ReleaseMutex(hQEMT);
    ReleaseMutex(hQTL);
    ReleaseMutex(hQHD);
   }

  break;


case QADM_REQ_COM_DATA:    // retrieve msg based on QEM entry#
case QADM_REQ_UNCOM_DATA:

  entry_num = (int)*thr_arg->lpsmbuf->mdata;

  found = 0;
  if (Check_Queue_Empty(MQEMT) == FALSE)
   {
    i = 0;
    qeme_no = MQEMT->que_hd_ptr;
    status = Cycle_QEME(qeme_no,&lpqeme);

    done = 0;
    while (!done)
    {
     if (((thr_arg->lpsmbuf->msgh.sub_mode == QADM_REQ_COM_DATA)
         && (lpqeme->txn_state == COMMIT)) ||
         ((thr_arg->lpsmbuf->msgh.sub_mode == QADM_REQ_UNCOM_DATA)
         && ((lpqeme->txn_state == ACTIVE) ||
             (lpqeme->txn_state == PENDING))))
       {
        if (i == entry_num)
          {
           done = 1;
           found = 1;
          }
         else if (i > entry_num)
           done = 1;
           else if (i < entry_num)
             i++;
       }

     if (!done)
       {
        if (qeme_no == MQEMT->que_tl_ptr)
           done = 1;
           else
           {
            qeme_no = (qeme_no+1)%MQEMT->max_entries;
            status = Cycle_QEME(qeme_no,&lpqeme);
           }
       }
    }      // end while

  }  // end if Check_Queue_Empty()

 if (found)
   {
    status = Retrieve_Msg_Hdr(Que_File_Handle,lpqeme->offset,&msgh);

    tblad.segment = lpqeme->offset.segment;
    tblad.block = lpqeme->offset.block+MSG_HDR_BLOCKS;

    status = Retrieve_Msg_Body(Que_File_Handle,tblad,msgh.size,buffer);
    QreplyAfterListen(thr_arg->qhandle,ADMINREP_MODE,SUB_MODE_OK,
                      buffer,msgh.size,&msgh);
   }
  else  // bad queue entry number
   {
    Diag("QS_Admin: Queue Entry %d is Invalid",entry_num);
    QreplyAfterListen(thr_arg->qhandle,ADMINREP_MODE,0,0,0,0);
   }
 break;


case QADM_REQ_SEL_DATA:    // retrieve list of MIDs based on key search

 seldat = (QADMSEL *)thr_arg->lpsmbuf->mdata;

 t0 = (CHAR *)malloc(sizeof(QMSG));

 if (Check_Queue_Empty(MQEMT) == FALSE)
   {
```

```
qeme_no = MQEMT->que_hd_ptr;
status = Cycle_QEME(qeme_no,&lpqeme);

ml = (MLIST *)malloc(sizeof(MLIST));
ml->next = NULL;
ml->mid.host = 0;
ml->mid.tid  = 0;
ml->mid.uid  = 0;
m2 = ml;

done = 0;
while (!done)
 {
  found = 0;

  if ((seldat->search_type == SEARCH_ALL_ENT) ||
      ((seldat->search_type == SEARCH_COM_ENT) &&
       (lpqeme->txn_state == COMMIT)) ||
       ((seldat->search_type == SEARCH_UNCOM_ENT) &&
        ((lpqeme->txn_state == ACTIVE) ||
        (lpqeme->txn_state == PENDING))))
   {
  Conv_Addr(&Dist_to_Move,&lpqeme->offset);

  dwPointer = SetFilePointer(Que_File_Handle,
                             Dist_to_Move,
                             NULL,FILE_BEGIN);

  Return_Status = ReadFile(Que_File_Handle,
                           t0,sizeof(QMSG),
                           &dwBytesRead,NULL);

  t1 = t0;
  t1 += seldat->preds[0].offset;
  if (seldat->preds[0].pred_type == INT_SEARCH_TYPE)
    int0 = (int *)t1;
    else if (seldat->preds[0].pred_type == SHORT_SEARCH_TYPE)
      sh0 = (short *)t1;

  if (seldat->num_preds > 1)
   {
    t2 = t0;
    t2 += seldat->preds[1].offset;
    if (seldat->preds[1].pred_type == INT_SEARCH_TYPE)
      int1 = (int *)t2;
      else if (seldat->preds[1].pred_type == SHORT_SEARCH_TYPE)
        sh1 = (short *)t2;

    if (seldat->num_preds > 2)
     {
      t3 = t0;
      t3 += seldat->preds[2].offset;
      if (seldat->preds[2].pred_type == INT_SEARCH_TYPE)
        int2 = (int *)t3;
        else if (seldat->preds[2].pred_type == SHORT_SEARCH_TYPE)
          sh2 = (short *)t3;
     }
   }
```

```
switch(seldat->num_preds)
 {
  case 1:
    if ((!seldat->preds[0].min_switch ||
         (((seldat->preds[0].pred_type == INT_SEARCH_TYPE) &&
           (*int0 >= seldat->preds[0].min_int_val)) ||
          ((seldat->preds[0].pred_type == SHORT_SEARCH_TYPE) &&
           (*sh0 >= seldat->preds[0].min_sh_val)) ||
          ((seldat->preds[0].pred_type == STR_SEARCH_TYPE) &&
           (strncmp(t1,seldat->preds[0].min_str_val,
                         seldat->preds[0].min_str_len) >= 0)))) &&

         (!seldat->preds[0].max_switch ||
         (((seldat->preds[0].pred_type == INT_SEARCH_TYPE) &&
           (*int0 <= seldat->preds[0].max_int_val)) ||
          ((seldat->preds[0].pred_type == SHORT_SEARCH_TYPE) &&
           (*sh0 <= seldat->preds[0].max_sh_val)) ||
          ((seldat->preds[0].pred_type == STR_SEARCH_TYPE) &&
           (strncmp(t1,seldat->preds[0].max_str_val,
                         seldat->preds[0].max_str_len) <= 0)))))
      found = 1;
    break;


  case 2:
    if ((!seldat->preds[0].min_switch ||
         (((seldat->preds[0].pred_type == INT_SEARCH_TYPE) &&
           (*int0 >= seldat->preds[0].min_int_val)) ||
          ((seldat->preds[0].pred_type == SHORT_SEARCH_TYPE) &&
           (*sh0 >= seldat->preds[0].min_sh_val)) ||
          ((seldat->preds[0].pred_type == STR_SEARCH_TYPE) &&
           (strncmp(t1,seldat->preds[0].min_str_val,
                     seldat->preds[0].min_str_len) >= 0)))) &&

         (!seldat->preds[0].max_switch ||
         (((seldat->preds[0].pred_type == INT_SEARCH_TYPE) &&
           (*int0 <= seldat->preds[0].max_int_val)) ||
          ((seldat->preds[0].pred_type == SHORT_SEARCH_TYPE) &&
           (*sh0 <= seldat->preds[0].max_sh_val)) ||
          ((seldat->preds[0].pred_type == STR_SEARCH_TYPE) &&
           (strncmp(t1,seldat->preds[0].max_str_val,
                     seldat->preds[0].max_str_len) <= 0)))) &&

         (!seldat->preds[1].min_switch ||
         (((seldat->preds[1].pred_type == INT_SEARCH_TYPE) &&
           (*int1 >= seldat->preds[1].min_int_val)) ||
          ((seldat->preds[1].pred_type == SHORT_SEARCH_TYPE) &&
           (*sh1 >= seldat->preds[1].min_sh_val)) ||
          ((seldat->preds[1].pred_type == STR_SEARCH_TYPE) &&
           (strncmp(t2,seldat->preds[1].min_str_val,
                     seldat->preds[1].min_str_len) >= 0)))) &&

         (!seldat->preds[1].max_switch ||
         (((seldat->preds[1].pred_type == INT_SEARCH_TYPE) &&
           (*int1 <= seldat->preds[1].max_int_val)) ||
          ((seldat->preds[1].pred_type == SHORT_SEARCH_TYPE) &&
           (*sh1 <= seldat->preds[1].max_sh_val)) ||
          ((seldat->preds[1].pred_type == STR_SEARCH_TYPE) &&
           (strncmp(t2,seldat->preds[1].max_str_val,
                     seldat->preds[1].max_str_len) <= 0)))))
```

```
        found = 1;
      break;


    case 3:
      if ((!seldat->preds[0].min_switch ||
          (((seldat->preds[0].pred_type == INT_SEARCH_TYPE) &&
            (*int0 >= seldat->preds[0].min_int_val)) ||
           ((seldat->preds[0].pred_type == SHORT_SEARCH_TYPE) &&
            (*sh0 >= seldat->preds[0].min_sh_val)) ||
           ((seldat->preds[0].pred_type == STR_SEARCH_TYPE) &&
            (strncmp(t1,seldat->preds[0].min_str_val,
                        seldat->preds[0].min_str_len) >= 0)))) &&

          (!seldat->preds[0].max_switch ||
          (((seldat->preds[0].pred_type == INT_SEARCH_TYPE) &&
            (*int0 <= seldat->preds[0].max_int_val)) ||
           ((seldat->preds[0].pred_type == SHORT_SEARCH_TYPE) &&
            (*sh0 <= seldat->preds[0].max_sh_val)) ||
           ((seldat->preds[0].pred_type == STR_SEARCH_TYPE) &&
            (strncmp(t1,seldat->preds[0].max_str_val,
                        seldat->preds[0].max_str_len) <= 0)))) &&

          (!seldat->preds[1].min_switch ||
          (((seldat->preds[1].pred_type == INT_SEARCH_TYPE) &&
            (*int1 >= seldat->preds[1].min_int_val)) ||
           ((seldat->preds[1].pred_type == SHORT_SEARCH_TYPE) &&
            (*sh1 >= seldat->preds[1].min_sh_val)) ||
           ((seldat->preds[1].pred_type == STR_SEARCH_TYPE) &&
            (strncmp(t2,seldat->preds[1].min_str_val,
                        seldat->preds[1].min_str_len) >= 0)))) &&

          (!seldat->preds[1].max_switch ||
          (((seldat->preds[1].pred_type == INT_SEARCH_TYPE) &&
            (*int1 <= seldat->preds[1].max_int_val)) ||
           ((seldat->preds[1].pred_type == SHORT_SEARCH_TYPE) &&
            (*sh1 <= seldat->preds[1].max_sh_val)) ||
           ((seldat->preds[1].pred_type == STR_SEARCH_TYPE) &&
            (strncmp(t2,seldat->preds[1].max_str_val,
                        seldat->preds[1].max_str_len) <= 0)))) &&

          (!seldat->preds[2].min_switch ||
          (((seldat->preds[2].pred_type == INT_SEARCH_TYPE) &&
            (*int2 >= seldat->preds[2].min_int_val)) ||
           ((seldat->preds[2].pred_type == SHORT_SEARCH_TYPE) &&
            (*sh2 >= seldat->preds[2].min_sh_val)) ||
           ((seldat->preds[2].pred_type == STR_SEARCH_TYPE) &&
            (strncmp(t3,seldat->preds[2].min_str_val,
                        seldat->preds[2].min_str_len) >= 0)))) &&

          (!seldat->preds[2].max_switch ||
          (((seldat->preds[2].pred_type == INT_SEARCH_TYPE) &&
            (*int2 <= seldat->preds[2].max_int_val)) ||
           ((seldat->preds[2].pred_type == SHORT_SEARCH_TYPE) &&
            (*sh2 <= seldat->preds[2].max_sh_val)) ||
           ((seldat->preds[2].pred_type == STR_SEARCH_TYPE) &&
            (strncmp(t3,seldat->preds[2].max_str_val,
                        seldat->preds[2].max_str_len) <= 0)))))
        found = 1;
      break;
```

```
          default:
            break;
        }          // end switch

        }          // endif


      if (found)
       {
        m3 = (MLIST *)malloc(sizeof(MLIST));
        m3->next = NULL;
        m2->next = m3;
        m2 = m3;

        qmsg = (QMSG *)t0;

        m3->mid.host = qmsg->Msg_Hdr.mid.host;
        m3->mid.tid  = qmsg->Msg_Hdr.mid.tid;
        m3->mid.uid  = qmsg->Msg_Hdr.mid.uid;
       }

      if (qeme_no == MQEMT->que_tl_ptr)
        done = 1;
        else
        {
         qeme_no = (qeme_no+1)%MQEMT->max_entries;
         status = Cycle_QEME(qeme_no,&lpqeme);
        }
     }     // end while

   }  // end if Check_Queue_Empty()


  if (m1 != NULL)
   {
    m2 = m1;
    m1 = m1->next;
    m2->next = NULL;
    free(m2);
   }

  if (m1 == NULL)
    {
     Diag("QS_Admin: No match on predicate");
     QreplyAfterListen(thr_arg->qhandle,ACK_MODE,SUB_MODE_EMPTY,0,0,0);
    }
    else
    {
     cnt = 0;
     m2 = m1;

     while (m2 != NULL)
      {
       cnt++;
       m2 = m2->next;
      }

     midstr1 = (MID *)malloc(cnt*sizeof(MID));
```

```
      m2 = m1;
      midstr2 = midstr1;

      while(m2 != NULL)
       {
        midstr2->host = m2->mid.host;
        midstr2->tid  = m2->mid.tid;
        midstr2->uid  = m2->mid.uid;
        m2 = m2->next;
        midstr2++;
       }

      QreplyAfterListen(thr_arg->qhandle,ADMINREP_MODE,SUB_MODE_OK,
                        (char *)midstr1,cnt*sizeof(MID),0);
     }

   // free up memory to clean up memory leaks
   free(t0);

   if (midstr1 != NULL)
     free(midstr1);

   while (m1 != NULL)
    {
     m2 = m1;
     m1 = m1->next;
     m2->next = NULL;
     free(m2);
    }

   break;


case QADM_REQ_MSG:    // retrieve msg based on MID

   mid = (MID *)thr_arg->lpsmbuf->mdata;

   status = Find_QEME(mid,&qeme_no,&lpqeme);

   if (status == QSUCCESS)
    {
     Conv_Addr(&Dist_to_Move,&lpqeme->offset);

     dwPointer = SetFilePointer(Que_File_Handle,
                                Dist_to_Move,
                                NULL,FILE_BEGIN);

     Return_Status = ReadFile(Que_File_Handle,
                              &qmsg2,sizeof(QMSG),
                              &dwBytesRead,NULL);

     QreplyAfterListen(thr_arg->qhandle,ADMINREP_MODE,SUB_MODE_OK,
                       qmsg2.Msg_Body.text,qmsg2.Msg_Hdr.size,
                       &qmsg2.Msg_Hdr);
    }
    else
    {
     Diag("QS_Admin: No msg with such MID");
     QreplyAfterListen(thr_arg->qhandle,ACK_MODE,SUB_MODE_EMPTY,0,0,0);
    }
```

```
          break;

        default:
          Diag("QS_Admin: Option %d Not Valid",thr_arg->lpsmbuf->msgh.sub_mode);
      }

}
```

```
/*
 * Copyright(C)1995 MITSUBISHI ELECTRIC ITA.  ALL RIGHTS RESERVED.
 * UNPUBLISHED -- RIGHTS RESERVED UNDER THE COPYRIGHT
 * LAWS OF THE UNITED STATES.  USE OF A COPYRIGHT NOTICE
 * IS PRECAUTIONARY ONLY AND DOES NOT IMPLY PUBLICATION
 * OR DISCLOSURE.
 *
 * THIS SOFTWARE CONTAINS CONFIDENTIAL INFORMATION AND
 * TRADE SECRETS OF MITSUBISHI ELECTRIC ITA.  USE, DISCLOSURE,
 * OR REPRODUCTION IS PROHIBITED WITHOUT THE PRIOR
 * EXPRESS WRITTEN PERMISSION OF MITSUBISHI ELECTRIC ITA.
 *
 * OpenMQ
 *   Module: qscommit.c
 *   Author: David Wong 9/8/95
 */


// Include Files
#include "qlib.h"
#include "qserv.h"
#include "qadmin.h"


extern lpQEMT    MQEMT;
extern lpQSTR    MQstate;
extern lpOPSTATS MQops;
extern lpMTLIST Pending_TXNs;
extern int       BLOCKS_PER_SEG;
extern int       SEGMENT_SIZE;
extern int       TOT_MSG_BLOCKS;
extern int       QEMT_Size;
extern int       QEMT_Seq_No;
extern int       LREC_Seq_No;
extern int       NUM_SEGS;
extern int       MAX_ELMS;




void QS_QCommit(
   lpTSTR  thr_arg)
{
   HANDLE  Que_File_Handle;
   HANDLE  hQEMT,hMQops;
   BLAD    tblad;
   BOOL    Return_Status;
   lpMTLIST This_TXN;
   lpTLIST This_TXN_ops;
   lpSMBUF lpsmbuf;
   LREC    log_rec;
   short   mode;
   int     status;
   int     tgets=0,tputs=0;
   int     count=0;

   Diag("DiskQ(%s): Termination for TID: %d from Host: %x",thr_arg->qname,
        thr_arg->lpsmbuf->msgh.mid.tid,thr_arg->lpsmbuf->msgh.mid.host);
```

```
/*
Diag("");
Diag("MID.host = %x",thr_arg->lpsmbuf->msgh.mid.host);
Diag("MID.tid  = %d",thr_arg->lpsmbuf->msgh.mid.tid);
Diag("MID.uid  = %d",thr_arg->lpsmbuf->msgh.mid.uid);
Diag("");
print_PTList();
Diag("");
*/

   lpsmbuf = thr_arg->lpsmbuf;

   if (lpsmbuf->msgh.mode == COMMIT_MODE)
     mode = COMMIT;
     else mode = ABORT;

   Return_Status = Find_MTlist(Pending_TXNs,&This_TXN,lpsmbuf->msgh.mid);

   if (Return_Status == FALSE)
     {
      Diag("QS_Commit: TXN %d from Host %x does not exist",
              lpsmbuf->msgh.mid.tid,lpsmbuf->msgh.mid.host);
      if (thr_arg->qhandle != QUEUE_TEST_VALUE)
        QreplyAfterListen(thr_arg->qhandle,ACK_MODE,SUB_MODE_INV_TID,0,0,0);
     }
     else
     {
     // first, update the entries in the
     // QEM Table for each op in txn
      This_TXN_ops = This_TXN->ops;

      while (This_TXN_ops != NULL)
       {
        QS_Resolve_TXN_Op(This_TXN_ops->qeme_no,
                          &tgets,&tputs,mode);

        This_TXN_ops = This_TXN_ops->next;
        count++;       // increment txn ops counter
       }

      // delete TID op list from pending txn list
      Return_Status = Del_MTlist(&Pending_TXNs,lpsmbuf->msgh.mid);
      if (Return_Status == FALSE)
        Diag("QS_Commit: error deleting TID %d from Host %d",
                log_rec.mid.tid,log_rec.mid.host);


      // then, write ABORT/COMMIT log record
      Que_File_Handle = thr_arg->Que_File_Handle;

      hQEMT = OpenMutex(SYNCHRONIZE,FALSE,QEMT_LOCK);
      if (!hQEMT)
        Diag("QS_QCommit: Can't OpenMutex for QEM Table lock");

      if ((status = WaitForSingleObject(hQEMT,QUE_LOCK_TIMEOUT)) !=
                                                     WAIT_OBJECT_0)
        Diag("QS_QCommit: Synch wait for QEM Table lock failed");

      log_rec.marker          = LRMARK;
      status                  = Gen_LREC_Seq_No(&log_rec.seq_no);
      log_rec.mode           = mode;
      log_rec.flags          = lpsmbuf->msgh.flags;
      log_rec.qeme_no        = NIL;
      log_rec.txn_state      = mode;
      log_rec.vote           = mode;
      log_rec.mid.host       = lpsmbuf->msgh.mid.host;
      log_rec.mid.tid        = lpsmbuf->msgh.mid.tid;
      log_rec.mid.uid        = lpsmbuf->msgh.mid.uid;
      log_rec.offset.segment = NIL;
      log_rec.offset.block   = NIL;

      // Fetch next available on-disk slot for log write.
      if (!Get_Next_BLAD(Que_File_Handle,LOG_WRITE,&tblad))
       {
        // queue data file is full or too fragmented.
        Diag("DiskQ(%s) is either full or too fragmented.",thr_arg->qname);
        if (thr_arg->qhandle != QUEUE_TEST_VALUE)
          QreplyAfterListen(thr_arg->qhandle,ACK_MODE,
                            SUB_MODE_FULL,0,0,0);

        ReleaseMutex(hQEMT);
        return;
       }

      status = Write_Log_Rec(Que_File_Handle,tblad,&log_rec);

      ReleaseMutex(hQEMT);


      // send ACK msg back to client
      if (thr_arg->qhandle != QUEUE_TEST_VALUE)
        QreplyAfterListen(thr_arg->qhandle,ACK_MODE,SUB_MODE_OK,0,0,0);


      // update RT stastical counters
      hMQops = OpenMutex(SYNCHRONIZE,FALSE,MQops_LOCK);
      if (!hMQops)
        Diag("QS_QCommit: Can't OpenMutex for MQops stats lock");

      if ((status = WaitForSingleObject(hMQops,QUE_LOCK_TIMEOUT)) !=
                                                       WAIT_OBJECT_0)
        Diag("QS_QCommit: Synch wait for MQops stats lock failed");

      if (mode == ABORT)
        MQops->num_aborts += count;
        else MQops->num_commits += count;

      MQops->pending_gets = MQops->pending_gets-tgets;
      MQops->pending_puts = MQops->pending_puts-tputs;

      ReleaseMutex(hMQops);
     }

}
```

```
void QS_Resolve_TXN_Op(
   int      qeme_no,
   int     *tgets,
   int     *tputs,
   short    mode)
{
   lpQEME  lpqeme;
   int     status;

   status = Cycle_QEME(qeme_no,&lpqeme);

   // increment number of pending GET/PUT
   // to decrement from MQops counter.

   if (lpqeme->mode == GET_MODE)
     (*tgets)++;
     else if (lpqeme->mode == PUT_MODE)
       (*tputs)++;

   lpqeme->txn_state = mode;
   lpqeme->vote = mode;

   if (mode == COMMIT)
    {
     if (lpqeme->mode == GET_MODE)
      {
       lpqeme->txn_state = EMPTY;
       lpqeme->vote = EMPTY;

       if (!Del_RST_Entry(lpqeme->offset.segment,lpqeme->offset.block))
         Diag("QS_QCommit: Problem with deleting entry from reservation table");

       Sort_RST_Entries(lpqeme->offset.segment);
      }
    }
    else    // ABORTed txn.
    {
     if (lpqeme->mode == PUT_MODE)    // aborted PUT operation
      {
       // set QEM entry to be a hole
       lpqeme->txn_state = EMPTY;
       lpqeme->vote = EMPTY;

       if (!Del_RST_Entry(lpqeme->offset.segment,lpqeme->offset.block))
         Diag("QS_QCommit: Problem with deleting entry from reservation table");

       Sort_RST_Entries(lpqeme->offset.segment);

       Fix_Que_Ptrs_on_Aborted_Put(lpqeme,qeme_no);
      }
      else                            // aborted GET operation
      {
       // reset the op flag to a PUT
       lpqeme->mode = PUT_MODE;

       lpqeme->txn_state = COMMIT;
       lpqeme->vote = COMMIT;

       Fix_Que_Ptrs_on_Aborted_Get(qeme_no);
```

```
            }
        }
}
```

```
User: root
Host: bunny
Class: bunny
Job: stdin
```

```
//////////////////////////////////////////////////////////////////////////////
//
// Common definitions used by OpenMQ Qservers
//
//////////////////////////////////////////////////////////////////////////////

// Synch objects for Queue Pointers
#define QEME_TS_GEN_LOCK  "Q/QEME_TS_Gen"
#define LPG_TS_GEN_LOCK   "Q/LPG_TS_Gen"
#define QUE_HD_PTR_LOCK   "Q/Que_Head_Ptr"
#define QUE_TL_PTR_LOCK   "Q/Que_Tail_Ptr"
#define QEMT_LOCK         "Q/QEM_Table"
#define MQstate_LOCK      "Q/MQstate"
#define MQops_LOCK        "Q/MQops"
#define QSHDN_LOCK        "Q/QS_Shutdown"
#define QSHDN_EVENT       "Q/QS_Shutdown_Event"

// Queue server states
#define QUEUE_ACTIVE       0L
#define QUEUE_SHUTDOWN     1L

// Other queue server attributes
#define QUE_LOCK_TIMEOUT  10000
#define MAX_QSERV_THREADS 20
#define NIL              -1L
#define FIFO              0L
#define QUEUE_TEST_VALUE  (HANDLE)-111

// Log record attributes
#define LOG_WRITE         0L
#define MSG_WRITE         1L
#define LRMARK           -999
#define QEMTMARK         -111

// Queue element size
// - disk block size is the larger of
// - log record or half a msg header
#define QUE_FILE_SIZE     50
#define QUE_EXTENT        0.0          // queue file extent percentage
#define BLOCK             46           // size of a block on disk
#define MSG_HDR_BLOCKS    2            // number of blocks for msg header
#define MSG_BODY_BLOCKS   177          // ceil((8192-sizeof(MSGH))/BLOCK)
                                       // blocks for msg body
#define LOG_REC_BLOCKS    1            // 1 block required for log record
#define MSG_HDR_SIZE      2*BLOCK      // msg header size
#define MSG_BODY_SIZE     177*BLOCK    // msg body size on disk
#define MSG_ENTRY_BLOCKS  (MSG_HDR_BLOCKS+MSG_BODY_BLOCKS+LOG_REC_BLOCKS)
#define GET_LOG_REC_BLOCKS   LOG_REC_BLOCKS;   // log record for GETs
#define TERM_LOG_REC_BLOCKS  LOG_REC_BLOCKS;   // ABORT/COMMOT log record

// Queue entry transactional states
#define INACTIVE          0L
#define ACTIVE            1L
#define INITIAL           2L
#define PENDING           3L
#define PREPARED          4L
#define ABORT             5L
#define COMMIT            6L
#define EMPTY             7L
```

```
// Queue server include files
#include <math.h>
#include <string.h>


// Queue server type defs
typedef struct msgb {          // Message body
   char  text[MAXMSGDATA];
} MSGB, *lpMSGB;

typedef struct blad {          // msg block address
   int   segment;              // segment number
   int   block;                // block offset in segment
} BLAD, *lpBLAD;

typedef struct qeme {              // QEM Table entry
   int            index;           // QEM entry number
   unsigned long  timestamp;       // entry creation timestamp
                                   // needed to preserve order
   MID            mid;             // unique msg ID
   short          mode;            // QGET/QPUT type
   short          priority;        // currently unused
   short          txn_state;       // TXN state
   short          vote;            // participant 2PC vote
   int            flags;           // flags for TXN/NOTXN
   BLAD           offset;          // msg block offset on disk
} QEME, *lpQEME;

typedef struct ptl {               // pending TXN list on disk
   MID   mid;                      // unique msg ID
   int   qeme_no;                  // entry index in QEM Table
} PTLIST, *lpPTLIST;

typedef struct sn {                // sequence number structure
   time_t   timestamp;             // timestamp based on time() call
   int      counter;               // counter that regenerates after each restart
} SN, *lpSN;

typedef struct lrclst {          // list of active log records in segment
  int       max_txns_per_seg;  // max number of active txns per segment
  int      *address;           // pointer to log record block values
} LRCLST, *lpLRCLST;

typedef struct rstseg {          // list msg block entry values in segment
  int       seg_no;              // segment number
  int      *msg_block;           // pointer to msg block entry values
} RSTSEG, *lpRSTSEG;

typedef struct rst {             // reservation table of msgs on disk
  int       num_segs;            // number of segments in queue data file
  int       msgs_per_seg;        // number of msgs in the segment
  lpRSTSEG  seg_ptr;             // pointer to buffer of segment entries
} RST, *lpRST;

typedef struct qemt {             // QEM Table
   int      marker;               // QEM Table marker
   SN       qem_sn;               // QEM sequence number
   SN       lrec_sn;              // beginning lrec timestamp in segment
   int      num_segs;          // number of segments in datafile
   int      max_entries;       // max number of entries on queue
   int      max_entries_limit; // limit on max entries value
   int      que_hd_ptr;        // QEM Table head pointer
   int      que_tl_ptr;        // QEM Table tail pointer
   short    qget_state;        // ENABLED/DISABLED switch for GETs
   short    qput_state;        // ENABLED/DISABLED switch for PUTs
   int      num_pts;           // number of pending txns
   BLAD     next_avail_block;  // next available block byte offset
   lpQEME   qeme_ptr;          // pointer to QEM Table entries
   lpRST    rst_ptr;           // pointer to reservation table
   lpPTLIST ptl_ptr;           // pointer to list of pending txns
} QEMT, *lpQEMT;

typedef struct opstats {      // RT operational statistics
   int      pending_gets;     // pending gets only
   int      pending_puts;     // pending puts only
   int      num_gets;         // committed gets only
   int      num_puts;         // committed puts only
   int      num_aborts;
   int      num_commits;
} OPSTATS, *lpOPSTATS;

typedef struct qstr {          // used to record queue state
   short    svr_state;         // designate clean shutdown
   int      num_restarts;      // number of restarts
   int      num_recov_tries;   // number of recovery attempts
   time_t   first_start_time;  // time of fresh startup
   time_t   last_restart_time; // time of last restart
   time_t   last_recov_time;   // time of last recovery attempt
} QSTR, *lpQSTR;

typedef struct msg {         // Queue Msg Block on disk
   MSGH   Msg_Hdr;           // Msg Header
   MSGB   Msg_Body;          // Msg Body
} QMSG, *lpQMSG;

typedef struct lrec {        // Log Record
   int    marker;            // log record marker
   SN     seq_no;            // increasing sequence number
   short  mode;              // QGET/QPUT flag
   short  txn_state;         // TXN state
   short  vote;              // participant 2PC vote
   short  dummy_1;           // not used at this time
   int    flags;             // Q_LOG|Q_TRAN flags
   int    qeme_no;           // QEM Table entry number
   MID    mid;               // unique msg ID
   BLAD   offset;            // msg block offset on disk
} LREC, *lpLREC;

typedef struct drec {        // dummy structure for retrieving
   int    field[11];         // records from disk at recovery
} DREC, *lpDREC;

typedef  struct tstr {
   CHAR      logical[NAMESIZE];       // logical queue name
   CHAR      physical[NAMESIZE];      // physical queue server name
   CHAR      qname[QUE_FILE_SIZE];    // queue data file name
   CHAR      qstate[QUE_FILE_SIZE];   // queue state file name
   HANDLE    Que_File_Handle;         // queue datafile handle
   lpQHANDLE qhandle;               // handle to SHM buffers
   lpSMBUF   lpsmbuf;               // pointer to SMBUF
} TSTR, *lpTSTR;

typedef  struct mstr {              // Structure for passing thread data
   CHAR       logical[NAMESIZE];    // logical queue name
   CHAR       physical[NAMESIZE];   // physical queue server name
   CHAR       qname[QUE_FILE_SIZE]; // queue data file name
   CHAR       qstate[QUE_FILE_SIZE]; // queue state file name
   int        thr_no;               // thread number
} MSTR, *lpMSTR;

typedef  struct tlist {             // list of ops for a TID
   struct tlist  *next;             // chain to next entry index
   int           qeme_no;           // entry index into QEM Table
} TLIST, *lpTLIST;

typedef  struct mtlist {            // list of pending TIDs
   struct mtlist  *next;            // chain to next TID
   lpTLIST        ops;              // list of ops for txn
   MID            mid;              // MID for each txn
} MTLIST, *lpMTLIST;

typedef  struct cntstr {
   int        committed;
   int        holey;
} CNTSTR, *lpCNTSTR;

typedef  struct lpg_ts {
   int            qeme_no;
   unsigned long timestamp;
} LPG_TS_STR, *lpLPG_TS_STR;


// Function prototypes.

void QS_QGet(
   lpTSTR  thr_arg);

void QS_QPut(
   lpTSTR  thr_arg);

void QS_QCommit(
   lpTSTR  thr_arg);

void QS_QAdmin(
   lpTSTR     thr_arg);

void QRecov(
   HANDLE  Que_File_Handle,
   int     seg_no);

void QR_Resolve_PTL(
   HANDLE  Que_File_Handle);

void QR_Resolve_TXN_Op(
   int     qeme_no,
   short   mode);

void QS_Resolve_TXN_Op(
   int     qeme_no,
   int    *tgets,
   int    *tputs,
   short   mode);

int Reconstruct_RST();

HANDLE Create_Queue_File(
   CHAR    *FileName,
   CHAR    *qstate,
   int      Max_Elms,
   int      Num_Segs);

HANDLE Open_Queue_File(
   CHAR *FileName);

BOOL Delete_Queue_File(
   CHAR   *FileName);

int Find_Num_Entries(
   lpCNTSTR  *cnt_ptr);

int Find_QEME(
   lpMID    mid,
   int     *qeme_no2,
   lpQEME  *lpqeme2);

int Cycle_QEME(
   int      qeme_no,
   lpQEME *lpqeme2);


void Find_Last_Pending_Get(
   int     *qeme_no);

void Fix_Que_Ptrs_on_Aborted_Put(
   lpQEME  lpqeme,
   int     qeme_no);

void Fix_Que_Ptrs_on_Aborted_Get(
   int     qeme_no);

void Update_QEME_TS();

int Find_Latest_QEM(
   HANDLE  Que_File_Handle,
   lpQEMT  *lpqemt2,
   int     *seg_num);

int Create_QEMT(
   lpQEMT  *lpqemt,
   int      max_elms);

int Init_QEMT(
   lpQEMT  lpqemt,
   int     max_elms,
   int     ext_elms,
   int     num_segs);

int Write_QEMT(
   HANDLE  Que_File_Handle,
   int     seg_no,
   lpQEMT  lpqemt);

int Retrieve_QEMT(
   HANDLE  Que_File_Handle,
   int     seg_no,
   lpQEMT *lpqemt2);

int Retrieve_Msg_Hdr(
   HANDLE  Que_File_Handle,
   BLAD    Msg_Addr,
   lpMSGH  Msg_Hdr);

int Write_Msg_Hdr(
   HANDLE  Que_File_Handle,
   BLAD    Msg_Addr,
   lpMSGH  Msg_Hdr);

int Retrieve_Msg_Body(
   HANDLE  Que_File_Handle,
   BLAD    Msg_Addr,
   DWORD   Msg_Length,
   CHAR   *Msg_Body);

int Write_Msg_Body(
   HANDLE  Que_File_Handle,
   BLAD    Msg_Addr,
   DWORD   Msg_Length,
   CHAR   *Msg_Body);

int Retrieve_Msg(
   HANDLE  Que_File_Handle,
   BLAD    Msg_Addr,
   lpQMSG  Msg);

int Retrieve_Log_Rec(
   HANDLE  Que_File_Handle,
   BLAD    LREC_Addr,
   lpLREC  LREC_ptr);

int Write_Log_Rec(
   HANDLE  Que_File_Handle,
   BLAD    LREC_Addr,
   lpLREC  LREC_ptr);

void Init_Active_LREC_List();

int msg_entry_comp(
   const void *arg1,
   const void *arg2);

void Sort_RST_Entries(
   int      seg_no);

int Add_RST_Entry(
   int      seg_no,
   int      entry_value);

int Del_RST_Entry(
   int     seg_no,
   int     entry_value);

int Entry_in_RST(
   int     seg_no,
   int     entry_value);

int log_entry_comp(
   const void *arg1,
   const void *arg2);

void Sort_Log_Entries();

int Add_Log_Entry(
   int     entry_value);

int Del_Log_Entry(
   int     entry_value);

int Del_Log_Entry(
   int     entry_value);

int In_Log_Rec_List(
   int     start,
   int     end,
   int    *hit);

int Cycle_RST_Seg(
   int         seg_no,
   lpRSTSEG *lprstseg);


int Get_Next_BLAD(
   HANDLE  Que_File_Handle,
   int     type,
   BLAD   *offset);

BOOL Check_Queue_Full(
   lpQEMT  lpqemt);

BOOL Check_Queue_Empty(
   lpQEMT  lpqemt);

void Update_Globals();

void Conv_from_MTlist();

void Conv_to_MTlist();

void Add_Tlist(
   lpTLIST *head,
   int      qeme_no);

void Add_MTlist(
   lpMTLIST *head,
   lpMTLIST *tail,
   MID       mid);

BOOL Find_MTlist(
   lpMTLIST  head,
   lpMTLIST *pres,
   MID       mid);

BOOL Find_Tlist(
   lpTLIST  head,
   int      qeme_no);

void Del_MTlist_All(
   lpMTLIST  *head);

BOOL Del_MTlist(
   lpMTLIST  *head,
   MID        mid);

void Del_Tlist(
   lpTLIST  *head);

void Sub_Addr (
   LONG    *diff,
   BLAD    *x,
   BLAD    *y);

void Conv_Addr (
   LONG    *diff,
   BLAD    *x);

int Gen_QEM_Seq_No(
   lpSN   tSeq_No);

int Gen_LREC_Seq_No(
   lpSN   tSeq_No);


int Gen_QEME_TS(
   unsigned long *tSeq_No);

void test_qemt(
   CHAR *filename);

void test_lrec(
   CHAR *filename);

void test_msgh(
   CHAR *filename);

void test_msgb(
   CHAR *filename);

void test_QS_funcs(
   CHAR   *logical,
   CHAR   *physical,
   CHAR   *qname);

void test_QRecov(
   CHAR   *qname);

void print_QEMT(
   lpQEMT    lpqemt,
   int       mode);
```

```
void print_QEME_txn_states();

void print_LREC(
   lpLREC     lplrec);

void print_SMBUF(
   lpSMBUF    lpsmbuf);

void print_MSGH(
   lpMSGH     lpmsgh);

void print_MSGB(
   CHAR       *lpmsgb);

void print_RST();

void print_Active_Log_List();

void print_PTList();

void print_QAdm_Stats();

BOOL Bigger_Seq_No(
   lpSN       seq_no1,
   lpSN       seq_no2);
```

```
/*
 * Copyright(C)1995 MITSUBISHI ELECTRIC ITA.  ALL RIGHTS RESERVED.
 * UNPUBLISHED -- RIGHTS RESERVED UNDER THE COPYRIGHT
 * LAWS OF THE UNITED STATES.  USE OF A COPYRIGHT NOTICE
 * IS PRECAUTIONARY ONLY AND DOES NOT IMPLY PUBLICATION
 * OR DISCLOSURE.
 *
 * THIS SOFTWARE CONTAINS CONFIDENTIAL INFORMATION AND
 * TRADE SECRETS OF MITSUBISHI ELECTRIC ITA.  USE, DISCLOSURE,
 * OR REPRODUCTION IS PROHIBITED WITHOUT THE PRIOR
 * EXPRESS WRITTEN PERMISSION OF MITSUBISHI ELECTRIC ITA.
 *
 * OpenMQ
 *   Module: qsget.c
 *   Author: David Wong 9/8/95
 */


// Include Files
#include "qlib.h"
#include "qserv.h"
#include "qadmin.h"


extern lpQEMT    MQEMT;
extern lpQSTR    MQstate;
extern lpOPSTATS MQops;
extern lpMTLIST Pending_TXNs;
extern int       BLOCKS_PER_SEG;
extern int       SEGMENT_SIZE;
extern int       TOT_MSG_BLOCKS;
extern int       QEMT_Size;
extern int       QEMT_Seq_No;
extern int       LREC_Seq_No;
extern int       NUM_SEGS;
extern int       MAX_ELMS;
extern unsigned long QEME_TS;
extern lpLPG_TS_STR Last_Pending_Get;



void QS_QGet(
   lpTSTR  thr_arg)
{
   HANDLE  Que_File_Handle;
   HANDLE  hQHD,hQTL,hLPG;
   HANDLE  hQEMT,hMQops;
   BOOL    Return_Status;
   lpMTLIST This_TXN;
   lpSMBUF lpsmbuf;
   lpQEME  lpqeme;
   LREC    log_rec;
   BLAD    tblad;
   int     status;
   int     qeme_no;
   int     new_hd_ptr;
   int     no_new_hd=0;
   int     done=0;
   int     found=0;
```

```
lpsmbuf = (SMBUF *)malloc(sizeof(SMBUF));
Que_File_Handle = thr_arg->Que_File_Handle;

// Check if GETs are disabled.
if (MQEMT->qget_state == DISABLED)
 {
  Diag("GET Operations for DiskQ(%s) is DISABLED",thr_arg->qname);
  if (thr_arg->qhandle != QUEUE_TEST_VALUE)
    QreplyAfterListen(thr_arg->qhandle,ACK_MODE,
                      SUB_MODE_DISABLED,0,0,0);
  return;
 }

// Obtain locks on queue pointers.
// Protocol is to obtain hQHD first,
// then hQTL.
hQHD = OpenMutex(SYNCHRONIZE,FALSE,QUE_HD_PTR_LOCK);

if (!hQHD)
  Diag("QS_QGet: Can't OpenMutex for queue head pointer lock");

if ((status = WaitForSingleObject(hQHD,QUE_LOCK_TIMEOUT)) != WAIT_OBJECT_0)
  Diag("QS_QGet: Synch wait for queue head pointer lock failed");

hQTL = OpenMutex(SYNCHRONIZE,FALSE,QUE_TL_PTR_LOCK);

if (!hQTL)
  Diag("QS_QGet: Can't OpenMutex for queue tail pointer lock");

if ((status = WaitForSingleObject(hQTL,QUE_LOCK_TIMEOUT)) !=
                                      WAIT_OBJECT_0)
  Diag("QS_QGet: Synch wait for queue tail pointer lock failed");


// Check if queue is EMPTY
if (Check_Queue_Empty(MQEMT) == TRUE)
 {
  Diag("DiskQ(%s): EMPTY",thr_arg->qname);
  if (thr_arg->qhandle != QUEUE_TEST_VALUE)
    QreplyAfterListen(thr_arg->qhandle,ACK_MODE,SUB_MODE_EMPTY,0,0,0);
  ReleaseMutex(hQHD);
  ReleaseMutex(hQTL);
  return;
 }


qeme_no = MQEMT->que_hd_ptr;
status = Cycle_QEME(qeme_no,&lpqeme);

if (MQEMT->que_hd_ptr == MQEMT->que_tl_ptr)
 {
  if ((lpqeme->mode == PUT_MODE) &&
      (lpqeme->txn_state == COMMIT))
   {
     MQEMT->que_hd_ptr = NIL;
     MQEMT->que_tl_ptr = NIL;
   }
  else
```

```
   {
    if (lpqeme->txn_state == EMPTY)
     {
      MQEMT->que_hd_ptr = NIL;
      MQEMT->que_tl_ptr = NIL;
     }

    Diag("DiskQ(%s): EMPTY",thr_arg->qname);
    if (thr_arg->qhandle != QUEUE_TEST_VALUE)
      QreplyAfterListen(thr_arg->qhandle,ACK_MODE,
                        SUB_MODE_EMPTY,0,0,0);
    ReleaseMutex(hQTL);
    ReleaseMutex(hQHD);
    return;
   }
}
else  // head != tail
{
 new_hd_ptr = MQEMT->que_hd_ptr;

 if ((lpqeme->mode == PUT_MODE) &&
     (lpqeme->txn_state == COMMIT))
   {
    MQEMT->que_hd_ptr = (MQEMT->que_hd_ptr+1)%MQEMT->max_entries;
   }
   else
   {
    // head entry is not yet committed;
    // find next one that is committed.

    while (!done && !found)
     {
      if ((lpqeme->mode == PUT_MODE) &&
          (lpqeme->txn_state == COMMIT))
       {
        done = 1;
        found = 1;
       }
       else
       {
        if ((lpqeme->mode == PUT_MODE) &&
            (lpqeme->txn_state == PENDING) &&
            (!no_new_hd))
         {
          new_hd_ptr = qeme_no;
          no_new_hd = 1;
         }

        if (qeme_no == MQEMT->que_tl_ptr)
          done = 1;
          else
          {
           qeme_no = (qeme_no+1)%MQEMT->max_entries;
           status = Cycle_QEME(qeme_no,&lpqeme);
          }
       }
     }

    if (found)
     {
        if ((qeme_no == MQEMT->que_tl_ptr) && (!no_new_hd))
         {
          MQEMT->que_hd_ptr = NIL;
          MQEMT->que_tl_ptr = NIL;
         }
          else MQEMT->que_hd_ptr = new_hd_ptr;
       }
       else
       {
        Diag("DiskQ(%s): EMPTY",thr_arg->qname);
        if (thr_arg->qhandle != QUEUE_TEST_VALUE)
          QreplyAfterListen(thr_arg->qhandle,ACK_MODE,
                            SUB_MODE_EMPTY,0,0,0);
        ReleaseMutex(hQTL);
        ReleaseMutex(hQHD);
        return;
       }
    }
 }

// Change state of QEM entry before
// releasing queue pointer locks.
lpqeme->txn_state = ACTIVE;
lpqeme->vote = INITIAL;

ReleaseMutex(hQTL);
ReleaseMutex(hQHD);

status = Retrieve_Msg_Hdr(Que_File_Handle,
                          lpqeme->offset,
                          &lpsmbuf->msgh);

lpqeme->mid.host = thr_arg->lpsmbuf->msgh.mid.host;
lpqeme->mid.tid  = thr_arg->lpsmbuf->msgh.mid.tid;
lpqeme->mid.uid  = thr_arg->lpsmbuf->msgh.mid.uid;
lpqeme->mode = GET_MODE;
lpqeme->flags = thr_arg->lpsmbuf->msgh.flags;
lpqeme->priority = FIFO;

tblad.segment = lpqeme->offset.segment;
tblad.block = lpqeme->offset.block+MSG_HDR_BLOCKS;

status = Retrieve_Msg_Body(Que_File_Handle,tblad,
                           lpsmbuf->msgh.size,lpsmbuf->mdata);

// Diag("DiskQ(%s): DEQUEs message - %s",thr_arg->qname,lpsmbuf->mdata);
Diag("DiskQ(%s): DEQUEs message",thr_arg->qname);


// increment get ops counter
hMQops = OpenMutex(SYNCHRONIZE,FALSE,MQops_LOCK);

if (!hMQops)
  Diag("QS_QGet: Can't OpenMutex for MQops stats lock");

if ((status = WaitForSingleObject(hMQops,QUE_LOCK_TIMEOUT)) !=
                                     WAIT_OBJECT_0)
  Diag("QS_QGet: Synch wait for MQops stats lock failed");
```

```
MQops->num_gets++;

hQEMT = OpenMutex(SYNCHRONIZE,FALSE,QEMT_LOCK);

if (!hQEMT)
  Diag("QS_QGet: Can't OpenMutex for QEM Table lock");

if ((status = WaitForSingleObject(hQEMT,QUE_LOCK_TIMEOUT)) !=
                                      WAIT_OBJECT_0)
  Diag("QS_QGet: Synch wait for QEM Table lock failed");


// if transactional, then write PREPARED log record
if (BITSET(Q_TRAN,thr_arg->lpsmbuf->msgh.flags))
 {
  log_rec.marker = LRMARK;
  status = Gen_LREC_Seq_No(&log_rec.seq_no);
  log_rec.mode = GET_MODE;
  log_rec.flags = thr_arg->lpsmbuf->msgh.flags;
  log_rec.txn_state = PENDING;
  log_rec.vote = PREPARED;
  log_rec.qeme_no = qeme_no;
  log_rec.mid.host = thr_arg->lpsmbuf->msgh.mid.host;
  log_rec.mid.tid  = thr_arg->lpsmbuf->msgh.mid.tid;
  log_rec.mid.uid  = thr_arg->lpsmbuf->msgh.mid.uid;
  log_rec.offset.segment = lpqeme->offset.segment;
  log_rec.offset.block   = lpqeme->offset.block;

  // Fetch next available on-disk slot for log write.
  if (!Get_Next_BLAD(Que_File_Handle,LOG_WRITE,&tblad))
   {
    // queue data file is full or too fragmented.
    Diag("DiskQ(%s) is either full or too fragmented.",thr_arg->qname);
    if (thr_arg->qhandle != QUEUE_TEST_VALUE)
      QreplyAfterListen(thr_arg->qhandle,ACK_MODE,
                        SUB_MODE_FULL,0,0,0);
    ReleaseMutex(hQEMT);
    ReleaseMutex(hMQops);
    return;
   }

  status = Write_Log_Rec(Que_File_Handle,tblad,&log_rec);

  // return msg queue data and ACK msg to client
  if (thr_arg->qhandle != QUEUE_TEST_VALUE)
    QreplyAfterListen(thr_arg->qhandle,ACK_MODE,SUB_MODE_OK,
                      lpsmbuf->mdata,lpsmbuf->msgh.size,
                      &lpsmbuf->msgh);

  // update QEM entry
  lpqeme->mode = GET_MODE;
  lpqeme->txn_state = PENDING;
  lpqeme->vote = PREPARED;

  // add txn and/or op to pending txn list
  Return_Status = Find_MTlist(Pending_TXNs,&This_TXN,lpqeme->mid);

  if (Return_Status == FALSE)
    Add_MTlist(&Pending_TXNs,&This_TXN,lpqeme->mid);
```

```
      Add_Tlist(&This_TXN->ops,qeme_no);

      MQops->pending_gets++;

      // update last pending GET timestamp
      hLPG = OpenMutex(SYNCHRONIZE,FALSE,LPG_TS_GEN_LOCK);

      if (!hLPG)
        Diag("QS_QGet: Can't OpenMutex for LPG_TS_GEN Lock");

      if ((status = WaitForSingleObject(hLPG,QUE_LOCK_TIMEOUT)) !=
                                                  WAIT_OBJECT_0)
        Diag("QS_QGet: Synch wait for LPG_TS_GEN Lock failed");

      if (lpqeme->timestamp > Last_Pending_Get->timestamp)
       {
        Last_Pending_Get->qeme_no = qeme_no;
        Last_Pending_Get->timestamp = lpqeme->timestamp;
       }

      ReleaseMutex(hLPG);
     }
     else
     {
      // return msg queue data and ACK msg to client
      if (thr_arg->qhandle != QUEUE_TEST_VALUE)
        QreplyAfterListen(thr_arg->qhandle,ACK_MODE,SUB_MODE_OK,
                          lpsmbuf->mdata,lpsmbuf->msgh.size,
                          &lpsmbuf->msgh);

      // update QEM entry
      lpqeme->mode = GET_MODE;
      lpqeme->txn_state = EMPTY;
      lpqeme->vote = EMPTY;

      // increment commit counter
      MQops->num_commits++;

      // delete on-disk entry slot from reservation table
      if (!Del_RST_Entry(lpqeme->offset.segment,lpqeme->offset.block))
        Diag("QS_QGet: Problem with deleting entry from reservation table");
        else Sort_RST_Entries(lpqeme->offset.segment);
     }

    ReleaseMutex(hMQops);
    ReleaseMutex(hQEMT);

    free(lpsmbuf);

Diag("TID = %d, UID = %d, index= %d, offset.segment = %d, offset.block = %d",
     lpqeme->mid.tid,lpqeme->mid.uid,lpqeme->index,
     lpqeme->offset.segment,lpqeme->offset.block);
}
```

```
/*
 * Copyright(C)1995 MITSUBISHI ELECTRIC ITA.  ALL RIGHTS RESERVED.
 * UNPUBLISHED -- RIGHTS RESERVED UNDER THE COPYRIGHT
 * LAWS OF THE UNITED STATES.  USE OF A COPYRIGHT NOTICE
 * IS PRECAUTIONARY ONLY AND DOES NOT IMPLY PUBLICATION
 * OR DISCLOSURE.
 *
 * THIS SOFTWARE CONTAINS CONFIDENTIAL INFORMATION AND
 * TRADE SECRETS OF MITSUBISHI ELECTRIC ITA.  USE, DISCLOSURE,
 * OR REPRODUCTION IS PROHIBITED WITHOUT THE PRIOR
 * EXPRESS WRITTEN PERMISSION OF MITSUBISHI ELECTRIC ITA.
 *
 * OpenMQ
 *   Module: qsput.c
 *   Author: David Wong 9/8/95
 */


// Include Files
#include "qlib.h"
#include "qserv.h"
#include "qadmin.h"


extern lpQEMT    MQEMT;
extern lpQSTR    MQstate;
extern lpOPSTATS MQops;
extern lpMTLIST Pending_TXNs;
extern int       BLOCKS_PER_SEG;
extern int       SEGMENT_SIZE;
extern int       TOT_MSG_BLOCKS;
extern int       QEMT_Size;
extern int       QEMT_Seq_No;
extern int       LREC_Seq_No;
extern int       NUM_SEGS;
extern int       MAX_ELMS;
extern unsigned long QEME_TS;
extern lpLPG_TS_STR Last_Pending_Get;
extern lpLRCLST     Active_LREC_List;


//
// Issue - create separate lock for obtaining
//         space in reservation table.
//


void QS_QPut(
   lpTSTR  thr_arg)
{
   HANDLE  Que_File_Handle;
   HANDLE  hQHD,hQTL;
   HANDLE  hQEME,hQEMT,hMQops;
   BOOL    Return_Status;
   lpMTLIST This_TXN;
   lpSMBUF lpsmbuf;
   lpQEME  lpqeme;
   LREC    log_rec;
   BLAD    temp_offset;
   int     status;
int      qeme_no;
int      qeme_no2;
int      done=0;
int      found=0;


lpsmbuf = thr_arg->lpsmbuf;
Que_File_Handle = thr_arg->Que_File_Handle;


// Check if PUTs are disabled.
if (MQEMT->qput_state == DISABLED)
 {
  Diag("PUT Operations for DiskQ(%s) is DISABLED",thr_arg->qname);
  if (thr_arg->qhandle != QUEUE_TEST_VALUE)
    QreplyAfterListen(thr_arg->qhandle,ACK_MODE,
                      SUB_MODE_DISABLED,0,0,0);
  return;
 }


// Obtain locks on queue pointers.
// Protocol is to obtain hQHD first,
// then hQTL.
hQHD = OpenMutex(SYNCHRONIZE,FALSE,QUE_HD_PTR_LOCK);

if (!hQHD)
  Diag("QS_QPUT: Can't OpenMutex for queue head pointer lock");

if ((status = WaitForSingleObject(hQHD,QUE_LOCK_TIMEOUT)) !=
                                      WAIT_OBJECT_0)
  Diag("QS_QPut: Synch wait for queue head pointer lock failed");

hQTL = OpenMutex(SYNCHRONIZE,FALSE,QUE_TL_PTR_LOCK);

if (!hQTL)
  Diag("QS_QPUT: Can't OpenMutex for queue tail pointer lock");

if ((status = WaitForSingleObject(hQTL,QUE_LOCK_TIMEOUT)) !=
                                       WAIT_OBJECT_0)
  Diag("QS_QPut: Synch wait for queue tail pointer lock failed");


if (Check_Queue_Full(MQEMT) == TRUE)
 {
  Diag("DiskQ(%s) is FULL",thr_arg->qname);
  if (thr_arg->qhandle != QUEUE_TEST_VALUE)
    QreplyAfterListen(thr_arg->qhandle,ACK_MODE,
                      SUB_MODE_FULL,0,0,0);
  ReleaseMutex(hQTL);
  ReleaseMutex(hQHD);
  return;
 }


// Obtain lock on QEMT to fetch next available
// entry in reservation table and disk writes.

hQEMT = OpenMutex(SYNCHRONIZE,FALSE,QEMT_LOCK);
```

```
if (!hQEMT)
  Diag("QS_QPUT: Can't OpenMutex for QEM Table lock");

if ((status = WaitForSingleObject(hQEMT,QUE_LOCK_TIMEOUT)) !=
                                WAIT_OBJECT_0)
  Diag("QS_QPut: Synch wait for QEM Table lock failed");


if (MQEMT->que_hd_ptr == NIL)
 {
  hMQops = OpenMutex(SYNCHRONIZE,FALSE,MQops_LOCK);
  if (!hMQops)
    Diag("QS_QCommit: Can't OpenMutex for MQops stats lock");

  if ((status = WaitForSingleObject(hMQops,QUE_LOCK_TIMEOUT)) !=
                                     WAIT_OBJECT_0)
    Diag("QS_QCommit: Synch wait for MQops stats lock failed");


  if (MQops->pending_gets == 0)
   {
    // Fetch next available on-disk slot for msg.
    if (!Get_Next_BLAD(Que_File_Handle,MSG_WRITE,&temp_offset))
     {
      // queue data file is full or too fragmented.
      Diag("DiskQ(%s) is too fragmented - PUT failed.",thr_arg->qname);
      if (thr_arg->qhandle != QUEUE_TEST_VALUE)
        QreplyAfterListen(thr_arg->qhandle,ACK_MODE,SUB_MODE_FULL,0,0,0);
      ReleaseMutex(hQEMT);
      ReleaseMutex(hQTL);
      ReleaseMutex(hQHD);
      return;
     }

    MQEMT->que_hd_ptr = 0;
    MQEMT->que_tl_ptr = 0;
    status = Cycle_QEME(MQEMT->que_tl_ptr,&lpqeme);
   }
   else
   {
    qeme_no = Last_Pending_Get->qeme_no;

    if (qeme_no == NIL)
     {
      Diag("DiskQ(%s) has faulty RT counters",thr_arg->qname);
      if (thr_arg->qhandle != QUEUE_TEST_VALUE)
        QreplyAfterListen(thr_arg->qhandle,ACK_MODE,
                          SUB_MODE_FULL,0,0,0);
      ReleaseMutex(hMQops);
      ReleaseMutex(hQTL);
      ReleaseMutex(hQHD);
      return;
     }

    qeme_no2 = qeme_no;
    qeme_no = (qeme_no+1)%MQEMT->max_entries;
    status = Cycle_QEME(qeme_no,&lpqeme);

    while (!done && !found)
     {
      if (lpqeme->txn_state == EMPTY)
       {
        done = 1;
        found = 1;
       }
       else
       {
        qeme_no = (qeme_no+1)%MQEMT->max_entries;
        if (qeme_no == qeme_no2)
          done = 1;
          else status = Cycle_QEME(qeme_no,&lpqeme);
       }
     }

    if (found)
     {
      // Fetch next available on-disk slot for msg.
      if (!Get_Next_BLAD(Que_File_Handle,MSG_WRITE,&temp_offset))
       {
        // queue data file is full or too fragmented.
        Diag("DiskQ(%s) is too fragmented - PUT failed.",thr_arg->qname);
        if (thr_arg->qhandle != QUEUE_TEST_VALUE)
          QreplyAfterListen(thr_arg->qhandle,ACK_MODE,SUB_MODE_FULL,0,0,0);
        ReleaseMutex(hQEMT);
        ReleaseMutex(hQTL);
        ReleaseMutex(hQHD);
        return;
       }

      MQEMT->que_hd_ptr = qeme_no;
      MQEMT->que_tl_ptr = qeme_no;
     }
     else
     {
      Diag("DiskQ(%s) has faulty que pointers",thr_arg->qname);
      if (thr_arg->qhandle != QUEUE_TEST_VALUE)
        QreplyAfterListen(thr_arg->qhandle,ACK_MODE,
                          SUB_MODE_FULL,0,0,0);
      ReleaseMutex(hMQops);
      ReleaseMutex(hQTL);
      ReleaseMutex(hQHD);
      return;
     }
   }

  ReleaseMutex(hMQops);
 }
 else
 {
  if (!Get_Next_BLAD(Que_File_Handle,MSG_WRITE,&temp_offset))
   {
    // queue data file is full or too fragmented.
    Diag("DiskQ(%s) is too fragmented - PUT failed.",thr_arg->qname);
    if (thr_arg->qhandle != QUEUE_TEST_VALUE)
      QreplyAfterListen(thr_arg->qhandle,ACK_MODE,SUB_MODE_FULL,0,0,0);
    ReleaseMutex(hQEMT);
    ReleaseMutex(hQTL);
    ReleaseMutex(hQHD);
    return;
   }
   // This case is true only iff que head is EMPTY.
   // Roll over both que head and tail pointers.

   if (((MQEMT->que_tl_ptr+1)%MQEMT->max_entries) == MQEMT->que_hd_ptr)
     MQEMT->que_hd_ptr = (MQEMT->que_hd_ptr+1)%MQEMT->max_entries;

   MQEMT->que_tl_ptr = (MQEMT->que_tl_ptr+1)%MQEMT->max_entries;
   status = Cycle_QEME(MQEMT->que_tl_ptr,&lpqeme);
  }

// Change state of QEM entry before
// releasing queue pointer locks.

lpqeme->txn_state = ACTIVE;
lpqeme->vote = INITIAL;

ReleaseMutex(hQTL);
ReleaseMutex(hQHD);


// Now assign storage location to QEM entry.

lpqeme->offset.segment = temp_offset.segment;
lpqeme->offset.block   = temp_offset.block;


// Fetch QEM entry timestamp generator lock.
hQEME = OpenMutex(SYNCHRONIZE,FALSE,QEME_TS_GEN_LOCK);
if (!hQEME)
  Diag("QS_QPUT: Can't OpenMutex for QEME_TS Lock");
if ((status = WaitForSingleObject(hQEME,QUE_LOCK_TIMEOUT)) !=
                                         WAIT_OBJECT_0)
  Diag("QS_QPut: Synch wait for QEME_TS Lock failed");

status = Gen_QEME_TS(&lpqeme->timestamp);

ReleaseMutex(hQEME);

lpqeme->mid.host = lpsmbuf->msgh.mid.host;
lpqeme->mid.tid  = lpsmbuf->msgh.mid.tid;
lpqeme->mid.uid  = lpsmbuf->msgh.mid.uid;
lpqeme->mode = PUT_MODE;
lpqeme->flags = lpsmbuf->msgh.flags;
lpqeme->priority = FIFO;


// NoWait case on disk write of msg
// send ACK msg to client immediately
if ((!BITSET(Q_TRAN,lpsmbuf->msgh.flags))
    && (!BITSET(Q_LOG,lpsmbuf->msgh.flags)))
  {
   if (thr_arg->qhandle != QUEUE_TEST_VALUE)
     QreplyAfterListen(thr_arg->qhandle,ACK_MODE,SUB_MODE_OK,0,0,0);
  }

status = Write_Msg_Hdr(Que_File_Handle,
                       lpqeme->offset,
                       &lpsmbuf->msgh);
```

```
temp_offset.segment = lpqeme->offset.segment;
temp_offset.block   = lpqeme->offset.block+(MSG_HDR_BLOCKS);

status = Write_Msg_Body(Que_File_Handle,
                        temp_offset,
                        lpsmbuf->msgh.size,
                        lpsmbuf->mdata);

// Diag("DiskQ(%s): ENQUEs message - %s",thr_arg->qname,lpsmbuf->mdata);
Diag("DiskQ(%s): ENQUEs messages",thr_arg->qname);


// increment put ops counter
hMQops = OpenMutex(SYNCHRONIZE,FALSE,MQops_LOCK);

if (!hMQops)
  Diag("QS_QPUT: Can't OpenMutex for MQops stats lock");

if ((status = WaitForSingleObject(hMQops,QUE_LOCK_TIMEOUT)) !=
                                          WAIT_OBJECT_0)
  Diag("QS_QPut: Synch wait for MQops stats lock failed");

MQops->num_puts++;


// if transactional, then write log record
if (BITSET(Q_TRAN,lpsmbuf->msgh.flags))
 {
  log_rec.marker = LRMARK;
  status = Gen_LREC_Seq_No(&log_rec.seq_no);
  log_rec.mode = PUT_MODE;
  log_rec.flags = lpsmbuf->msgh.flags;
  log_rec.txn_state = PENDING;
  log_rec.vote = PREPARED;
  log_rec.qeme_no = MQEMT->que_tl_ptr;
  log_rec.mid.host = lpqeme->mid.host;
  log_rec.mid.tid  = lpqeme->mid.tid;
  log_rec.mid.uid  = lpqeme->mid.uid;
  log_rec.offset.segment = lpqeme->offset.segment;
  log_rec.offset.block   = lpqeme->offset.block;

  temp_offset.segment = lpqeme->offset.segment;
  temp_offset.block   = lpqeme->offset.block +
                            (MSG_HDR_BLOCKS+MSG_BODY_BLOCKS);

  status = Write_Log_Rec(Que_File_Handle,temp_offset,&log_rec);

  // send ACK msg back to client
  if (thr_arg->qhandle != QUEUE_TEST_VALUE)
    QreplyAfterListen(thr_arg->qhandle,ACK_MODE,SUB_MODE_OK,0,0,0);

  // update QEM entry
  lpqeme->txn_state = PENDING;
  lpqeme->vote = PREPARED;

  // add txn and/or op to pending txn list
  Return_Status = Find_MTlist(Pending_TXNs,&This_TXN,lpsmbuf->msgh.mid);

  if (Return_Status == FALSE)
    Add_MTlist(&Pending_TXNs,&This_TXN,lpsmbuf->msgh.mid);
```

```
      Add_Tlist(&This_TXN->ops,MQEMT->que_tl_ptr);

      MQops->pending_puts++;

      ReleaseMutex(hQEMT);
     }
     else
     {
      ReleaseMutex(hQEMT);

      // send ACK msg back to client if Q_LOG flag
      if ((BITSET(Q_LOG,lpsmbuf->msgh.flags) == Q_LOG)
          && (thr_arg->qhandle != QUEUE_TEST_VALUE))
        QreplyAfterListen(thr_arg->qhandle,ACK_MODE,SUB_MODE_OK,0,0,0);

      // update QEM entry
      lpqeme->txn_state = COMMIT;
      lpqeme->vote = COMMIT;

      // increment commit counter
      MQops->num_commits++;
     }

    ReleaseMutex(hMQops);


Diag("TID = %d, UID = %d, index = %d, offset.segment = %d, offset.block = %d",
     lpqeme->mid.tid,lpqeme->mid.uid,lpqeme->index,
     lpqeme->offset.segment,lpqeme->offset.block);
}
```

```
/*
 * Copyright(C)1995 MITSUBISHI ELECTRIC ITA.  ALL RIGHTS RESERVED.
 * UNPUBLISHED -- RIGHTS RESERVED UNDER THE COPYRIGHT
 * LAWS OF THE UNITED STATES.  USE OF A COPYRIGHT NOTICE
 * IS PRECAUTIONARY ONLY AND DOES NOT IMPLY PUBLICATION
 * OR DISCLOSURE.
 *
 * THIS SOFTWARE CONTAINS CONFIDENTIAL INFORMATION AND
 * TRADE SECRETS OF MITSUBISHI ELECTRIC ITA.  USE, DISCLOSURE,
 * OR REPRODUCTION IS PROHIBITED WITHOUT THE PRIOR
 * EXPRESS WRITTEN PERMISSION OF MITSUBISHI ELECTRIC ITA.
 *
 * OpenMQ
 *   Module: qtest.c
 *   Author: David Wong 9/8/95
 */


// Include Files
#include "qlib.h"
#include "qserv.h"
#include "qadmin.h"


extern lpQEMT   MQEMT;
extern lpQSTR   MQstate;
extern lpMTLIST Pending_TXNs;
extern int      BLOCKS_PER_SEG;
extern int      SEGMENT_SIZE;
extern int      TOT_MSG_BLOCKS;
extern int      QEMT_Size;
extern int      QEMT_Seq_No;
extern int      LREC_Seq_No;
extern int      NUM_SEGS;
extern int      MAX_ELMS;
extern unsigned long QEME_TS;
extern lpLRCLST Active_LREC_List;



void test_qemt(
   CHAR *filename)
{
   HANDLE  Que_File_Handle;
   HANDLE  hQHD,hQTL,hQEMT;
   BOOL    Return_Status;
   lpQEMT  lpqemt,lpqemt2;
   lpQEME  lpqeme;
   int     i,status;

   status = Create_QEMT(&lpqemt,MAX_ELMS);
   status = Init_QEMT(lpqemt,MAX_ELMS,0,NUM_SEGS);

   hQHD = CreateMutex(NULL,                 // No security
                      TRUE,                 // Get ownership of mutex
                      QUE_HD_PTR_LOCK);     // Name

   if (!hQHD)
     Diag("CreateMutex for Que Head Pointer lock failed.");
   hQTL = CreateMutex(NULL,                   // No security
                      TRUE,                   // Get ownership of mutex
                      QUE_TL_PTR_LOCK);       // Name

   if (!hQTL)
     Diag("CreateMutex for Que Head Pointer lock failed.");

   hQEMT = CreateMutex(NULL,                  // No security
                       TRUE,                  // Get ownership of mutex
                       QUE_TL_PTR_LOCK);      // Name

   if (!hQEMT)
     Diag("CreateMutex for Que Head Pointer lock failed.");

   lpqeme = lpqemt->qeme_ptr;
   for(i=1; i<=10; i++)
    {
     lpqeme->mid.host = 11111;
     lpqeme->mid.tid  = 22222;
     lpqeme->mode = PUT_MODE;
     lpqeme->flags = 0;
     lpqeme->priority = 0;
     lpqeme->txn_state = ACTIVE;
     lpqeme->vote = INACTIVE;
     lpqeme->offset.segment = 44444;
     lpqeme->offset.block = 55550+i;
     lpqeme++;
    }

   Que_File_Handle = Open_Queue_File(filename);

   if (Que_File_Handle == INVALID_HANDLE_VALUE)
     Diag("Problem opening queue file ...");

   status = Gen_QEM_Seq_No(&MQEMT->qem_sn);
   status = Write_QEMT(Que_File_Handle,0,lpqemt);

   if (status == QFAIL)
     Diag("QEMT Write Failed ...");

   status = Create_QEMT(&lpqemt2,MAX_ELMS);

   status = Retrieve_QEMT(Que_File_Handle,1,&lpqemt2);

   if (status == QFAIL)
     Diag("QEMT Read Failed ...");

   Return_Status = CloseHandle(Que_File_Handle);

   if (Return_Status == FALSE)
     Diag("Could not close queue file handle");

   print_QEMT(lpqemt2,1);
}


void test_lrec(
   CHAR *filename)
{
   HANDLE  Que_File_Handle;
   BOOL    Return_Status;
   lpLREC  lplrec;
   lpLREC  lplrec2;
   BLAD    LREC_Addr;
   int     status;

   Que_File_Handle = Open_Queue_File(filename);

   LREC_Addr.segment = 1;
   LREC_Addr.block = 1;

   lplrec = (LREC *)malloc(sizeof(LREC));

   lplrec->marker = LRMARK;
   status = Gen_LREC_Seq_No(&lplrec->seq_no);
   lplrec->mode = PUT_MODE;
   lplrec->qeme_no = 999;
   lplrec->txn_state = ACTIVE;
   lplrec->vote = INACTIVE;
   lplrec->mid.host = 11111;
   lplrec->mid.tid  = 22222;
   lplrec->offset.segment = 1;
   lplrec->offset.block = 1;

   print_LREC(lplrec);

   status = Write_Log_Rec(Que_File_Handle,LREC_Addr,lplrec);

   if (status == QFAIL)
     Diag("Log Write Failure ... ");

   lplrec2 = (LREC *)malloc(sizeof(LREC));

   status = Retrieve_Log_Rec(Que_File_Handle,LREC_Addr,lplrec2);

   if (status == QFAIL)
     Diag("Log Read Failure ... ");

   print_LREC(lplrec2);

   Return_Status = CloseHandle(Que_File_Handle);

   if (Return_Status == FALSE)
     Diag("Could not close queue file handle");
}


void test_msgh(
   CHAR *filename)
{
   HANDLE  Que_File_Handle;
   BOOL    Return_Status;
   lpMSGH  lpmsgh;
   lpMSGH  lpmsgh2;
   BLAD    MSGH_Addr;
   int     status;

   Que_File_Handle = Open_Queue_File(filename);
   MSGH_Addr.segment = 1;
   MSGH_Addr.block = 1;

   lpmsgh = (MSGH *)malloc(sizeof(MSGH));

   lpmsgh->type_coding = 999999;
   lpmsgh->mode = 1;
   lpmsgh->sub_mode = 2;
   lpmsgh->flags = Q_TRAN;
   lpmsgh->mid.host = 11111;
   lpmsgh->mid.tid  = 22222;
   lpmsgh->time = 33333;
   lpmsgh->to_node = 12345;
   lpmsgh->to_port = 23456;
   lpmsgh->to_smbuf = 44444;
   strcpy(lpmsgh->to_server,"Que_Server_1");
   strcpy(lpmsgh->to_logical,"Que_Service_1");
   lpmsgh->from_smbuf = 45677;
   lpmsgh->reply_smbuf = 34567;
   lpmsgh->size = 888;


   print_MSGH(lpmsgh);

   status = Write_Msg_Hdr(Que_File_Handle,MSGH_Addr,lpmsgh);

   if (status == QFAIL)
     Diag("Msg Header Write Failure ... ");

   lpmsgh2 = (MSGH *)malloc(sizeof(MSGH));


   status = Retrieve_Msg_Hdr(Que_File_Handle,MSGH_Addr,lpmsgh2);

   if (status == QFAIL)
     Diag("Msg Header Read Failure ... ");

   print_MSGH(lpmsgh2);

   Return_Status = CloseHandle(Que_File_Handle);

   if (Return_Status == FALSE)
     Diag("Could not close queue file handle");
}


void test_msgb(
   CHAR *filename)
{
   HANDLE  Que_File_Handle;
   BOOL    Return_Status;
   BLAD    MSGB_Addr;
   CHAR    string1[41];
   CHAR    string2[41];
   int     status;

   Que_File_Handle = Open_Queue_File(filename);
   MSGB_Addr.segment = 1;
   MSGB_Addr.block = 1;

   strcpy(string1,"The quick brown fox jumps over the fence.");

   print_MSGB(string1);

   status = Write_Msg_Body(Que_File_Handle,MSGB_Addr,41,string1);

   if (status == QFAIL)
     Diag("Msg Body Write Failure ... ");

   status = Retrieve_Msg_Body(Que_File_Handle,MSGB_Addr,41,string2);

   if (status == QFAIL)
     Diag("Msg Body Read Failure ... ");

   print_MSGB(string2);

   Return_Status = CloseHandle(Que_File_Handle);

   if (Return_Status == FALSE)
     Diag("Could not close queue file handle");
}


void test_QS_funcs(
   CHAR    *logical,
   CHAR    *physical,
   CHAR    *qname)
{
   lpSMBUF lpsmbuf;
   lpTSTR  thr_arg;

   lpsmbuf = (SMBUF *)malloc(sizeof(SMBUF));

   strcpy(lpsmbuf->mdata,"This is a very small message");

   strcpy(lpsmbuf->name,"Tem_Buffer_1");
   lpsmbuf->status = 0;
   lpsmbuf->sub_status = 0;

   lpsmbuf->msgh.type_coding  = 1;
   lpsmbuf->msgh.mode         = PUT_MODE;
   lpsmbuf->msgh.sub_mode     = 0;
   // lpsmbuf->msgh.flags        = Q_TRAN;

   lpsmbuf->msgh.time         = 33333;

   lpsmbuf->msgh.to_node      = 1234567;
   lpsmbuf->msgh.to_port      = 2345678;
   lpsmbuf->msgh.to_smbuf     = 2345678;

   strcpy(lpsmbuf->msgh.to_server,"barny");
   strcpy(lpsmbuf->msgh.to_logical,"barny/Q1");

   lpsmbuf->msgh.from_smbuf   = 4567890;
   lpsmbuf->msgh.reply_smbuf  = 3456789;
   lpsmbuf->msgh.size         = strlen(lpsmbuf->mdata);
```

```
    thr_arg = (TSTR *)malloc(sizeof(TSTR));

    strcpy(thr_arg->logical,logical);
    strcpy(thr_arg->physical,physical);
    strcpy(thr_arg->qname,qname);
    thr_arg->qhandle = QUEUE_TEST_VALUE;
    thr_arg->lpsmbuf = lpsmbuf;

    lpsmbuf->msgh.flags          = Q_LOG;
    lpsmbuf->msgh.mid.host       = 33333;
    lpsmbuf->msgh.mid.tid        = 1;
    QS_QPut(thr_arg);

    lpsmbuf->msgh.mode           = ADMINREQ_MODE;
    lpsmbuf->msgh.sub_mode       = QADM_REQ_STATS;
    QS_QAdmin(thr_arg);

    lpsmbuf->msgh.flags          = Q_TRAN;
    lpsmbuf->msgh.mid.host       = 214;
    lpsmbuf->msgh.mid.tid        = 2;
    QS_QPut(thr_arg);

    lpsmbuf->msgh.flags          = Q_TRAN;
    lpsmbuf->msgh.mid.host       = 214;
    lpsmbuf->msgh.mid.tid        = 2;
    QS_QGet(thr_arg);

    lpsmbuf->msgh.mode           = ADMINREQ_MODE;
    lpsmbuf->msgh.sub_mode       = QADM_REQ_STATS;
    QS_QAdmin(thr_arg);

    lpsmbuf->msgh.mode           = ABORT_MODE;
    lpsmbuf->msgh.mid.host       = 214;
    lpsmbuf->msgh.mid.tid        = 2;
    QS_QCommit(thr_arg);

    lpsmbuf->msgh.mode           = ADMINREQ_MODE;
    lpsmbuf->msgh.sub_mode       = QADM_REQ_STATS;
    QS_QAdmin(thr_arg);

/*
    lpsmbuf->msgh.mid.host       = 216;
    lpsmbuf->msgh.mid.tid        = 3;
    QS_QPut(thr_arg);

    lpsmbuf->msgh.mid.host       = 217;
    lpsmbuf->msgh.mid.tid        = 4;
    QS_QPut(thr_arg);

    print_QEMT(MQEMT,1);

    lpsmbuf->msgh.mid.host       = 218;
    lpsmbuf->msgh.mid.tid        = 5;
    QS_QPut(thr_arg);

    lpsmbuf->msgh.mid.host       = 219;
    lpsmbuf->msgh.mid.tid        = 6;
    QS_QPut(thr_arg);
    lpsmbuf->msgh.mid.host      = 219;
    lpsmbuf->msgh.mid.tid       = 6;
    QS_QPut(thr_arg);

    lpsmbuf->msgh.mid.host      = 12345;
    lpsmbuf->msgh.mid.tid       = 678;
    QS_QPut(thr_arg);

    lpsmbuf->msgh.mid.host      = 12345;
    lpsmbuf->msgh.mid.tid       = 678;
    QS_QPut(thr_arg);

    print_QEMT(MQEMT,1);

    lpsmbuf->msgh.mode          = COMMIT_MODE;
    lpsmbuf->msgh.mid.host      = 214;
    lpsmbuf->msgh.mid.tid       = 2;
    QS_QCommit(thr_arg);

    lpsmbuf->msgh.mid.host      = 33333;
    lpsmbuf->msgh.mid.tid       = 111;
    QS_QGet(thr_arg);
    print_PTList();

    lpsmbuf->msgh.mode          = COMMIT_MODE;
    lpsmbuf->msgh.mid.host      = 33333;
    lpsmbuf->msgh.mid.tid       = 111;
    QS_QCommit(thr_arg);
    print_PTList();

    lpsmbuf->msgh.mode          = ABORT_MODE;
    lpsmbuf->msgh.mode          = COMMIT_MODE;

    QS_QGet(thr_arg);

    lpsmbuf->msgh.mode          = ADMINREQ_MODE;
    lpsmbuf->msgh.sub_mode      = QADM_REQ_STATS;
    QS_QAdmin(thr_arg);

    print_QEMT(MQEMT,1);
*/

}


void test_QRecov(
    CHAR    *qname)
{
    HANDLE  Que_File_Handle;
    BOOL    Return_Status;
    lpQEMT  lpqemt;
    int     seg_no;
    int     status;

    Que_File_Handle = Open_Queue_File(qname);

    status = Find_Latest_QEM(Que_File_Handle,&lpqemt,&seg_no);
```

```
   print_QEMT(lpqemt,1);

   Return_Status = CloseHandle(Que_File_Handle);
}


void print_QEMT(
   lpQEMT    lpqemt,
   int       mode)
{
   lpQEME    lpqeme;
   lpPTLIST  ptl_ptr;
   int       i;

   Diag("");
   Diag("QEMT->marker = %d",lpqemt->marker);
   Diag("QEMT->qem_sn.timestamp = \t%s",ctime(&lpqemt->qem_sn.timestamp));
   Diag("QEMT->qem_sn.counter = %d",lpqemt->qem_sn.counter);
   Diag("QEMT->num_segs = %d",lpqemt->num_segs);
   Diag("QEMT->max_entries = %d",lpqemt->max_entries);
   Diag("QEMT->max_entries_limit = %d",lpqemt->max_entries_limit);
   Diag("QEMT->que_hd_ptr = %d",lpqemt->que_hd_ptr);
   Diag("QEMT->que_tl_ptr = %d",lpqemt->que_tl_ptr);
   Diag("QEMT->qget_state = %d",lpqemt->qget_state);
   Diag("QEMT->qput_state = %d",lpqemt->qput_state);
   Diag("QEMT->num_pts = %d",lpqemt->num_pts);
   Diag("QEMT->next_avail_block.segment = %d",
           lpqemt->next_avail_block.segment);
   Diag("QEMT->next_avail_block.block = %d",
           lpqemt->next_avail_block.block);

  if (mode == 1)
    {
    lpqeme = lpqemt->qeme_ptr;
    for(i=0;i<lpqemt->max_entries;i++)
     {
      Diag("");
      Diag("index = %d",lpqeme->index);
      Diag("timestamp = %d",lpqeme->timestamp);
      Diag("MID.host = %d",lpqeme->mid.host);
      Diag("MID.tid = %d",lpqeme->mid.tid);
      Diag("mode = %d",lpqeme->mode);
      Diag("flags = %d",lpqeme->flags);
      Diag("priority = %d",lpqeme->priority);
      Diag("txn_state = %d",lpqeme->txn_state);
      Diag("vote = %d",lpqeme->vote);
      Diag("offset.segment = %d",lpqeme->offset.segment);
      Diag("offset.block = %d",lpqeme->offset.block);
      lpqeme++;
     }

    ptl_ptr = lpqemt->ptl_ptr;
    for(i=0;i<lpqemt->num_pts;i++)
     {
      Diag("");
      Diag("MID.host = %d",ptl_ptr->mid.host);
      Diag("MID.tid = %d",ptl_ptr->mid.tid);
      Diag("qeme_no = %d",ptl_ptr->qeme_no);
```

```
        ptl_ptr++;
       }
     }
}




void print_QEME_txn_states()
{
   lpQEME     lpqeme;
   char       buf[10];
   int        i;

   Diag("");
   Diag("qeme_no\t txn_state");
   Diag("-------\t ---------");

   lpqeme = MQEMT->qeme_ptr;
   for(i=0;i<MQEMT->max_entries;i++)
     {
      switch(lpqeme->txn_state)
        {
         case INACTIVE: strcpy(buf,"INACTIVE");
                break;
         case ACTIVE:   strcpy(buf,"ACTIVE");
                break;
         case INITIAL:  strcpy(buf,"INITIAL");
                break;
         case PENDING:  strcpy(buf,"PENDING");
                break;
         case PREPARED: strcpy(buf,"PREPARED");
                break;
         case ABORT:    strcpy(buf,"ABORT");
                break;
         case COMMIT:   strcpy(buf,"COMMIT");
                break;
         case EMPTY:    strcpy(buf,"EMPTY");
                break;
         default:       strcpy(buf,"GARBAGE");
        }

      Diag("  %d\t %s",i,buf);
      lpqeme++;
     }

   Diag("");
}




void print_LREC(
   lpLREC      lplrec)
{
   Diag("LREC->marker = %d",lplrec->marker);
   Diag("LREC->seq_no.timestamp = \t%s",ctime(&lplrec->seq_no.timestamp));
   Diag("LREC->seq_no.counter = %d",lplrec->seq_no.counter);
```

```
    Diag("LREC->mode = %d",lplrec->mode);
    Diag("LREC->qeme_no = %d",lplrec->qeme_no);
    Diag("LREC->txn_state = %d",lplrec->txn_state);
    Diag("LREC->vote = %d",lplrec->vote);
    Diag("LREC->mid.host = %d",lplrec->mid.host);
    Diag("LREC->mid.tid = %d",lplrec->mid.tid);
    Diag("LREC->offset.segment = %d",lplrec->offset.segment);
    Diag("LREC->offset.block = %d",lplrec->offset.block);
}


void print_SMBUF(
    lpSMBUF    lpsmbuf)
{
    Diag("SMBUF->name = %s",lpsmbuf->name);
    Diag("SMBUF->status = %d",lpsmbuf->status);
    Diag("SMBUF->sub_status = %d",lpsmbuf->sub_status);
    Diag("SMBUF->MSGH.type_coding = %d",lpsmbuf->msgh.type_coding);
    Diag("SMBUF->MSGH.mode = %d",lpsmbuf->msgh.mode);
    Diag("SMBUF->MSGH.sub_mode = %d",lpsmbuf->msgh.sub_mode);
    Diag("SMBUF->MSGH.flags = %d",lpsmbuf->msgh.flags);

    Diag("SMBUF->MSGH.mid.host = %d",lpsmbuf->msgh.mid.host);
    Diag("SMBUF->MSGH.mid.tid = %d",lpsmbuf->msgh.mid.tid);
    Diag("SMBUF->MSGH.time = %d",lpsmbuf->msgh.time);

    Diag("SMBUF->MSGH.to_node = %d",lpsmbuf->msgh.to_node);
    Diag("SMBUF->MSGH.to_port = %d",lpsmbuf->msgh.to_port);
    Diag("SMBUF->MSGH.to_smbuf = %d",lpsmbuf->msgh.to_smbuf);

    Diag("SMBUF->MSGH.to_server = %s",lpsmbuf->msgh.to_server);
    Diag("SMBUF->MSGH.to_logical = %s",lpsmbuf->msgh.to_logical);

    Diag("SMBUF->MSGH.from_smbuf = %d",lpsmbuf->msgh.from_smbuf);
    Diag("SMBUF->MSGH.reply_smbuf = %d",lpsmbuf->msgh.reply_smbuf);
    Diag("SMBUF->MSGH.size = %d",lpsmbuf->msgh.size);

    Diag("SMBUF->mdata:");
    Diag("%s",lpsmbuf->mdata);
}


void print_MSGH(
    lpMSGH     lpmsgh)
{
    Diag("MSGH->type_coding = %d",lpmsgh->type_coding);
    Diag("MSGH->mode = %d",lpmsgh->mode);
    Diag("MSGH->sub_mode = %d",lpmsgh->sub_mode);
    Diag("MSGH->flags = %d",lpmsgh->flags);
    Diag("MSGH->mid.host = %d",lpmsgh->mid.host);
    Diag("MSGH->mid.tid = %d",lpmsgh->mid.tid);
    Diag("MSGH->time = %d",lpmsgh->time);
    Diag("MSGH->to_node = %d",lpmsgh->to_node);
    Diag("MSGH->to_port = %d",lpmsgh->to_port);
    Diag("MSGH->to_smbuf = %d",lpmsgh->to_smbuf);
    Diag("MSGH->to_server = %s",lpmsgh->to_server);
    Diag("MSGH->to_logical = %s",lpmsgh->to_logical);
    Diag("MSGH->from_smbuf = %d",lpmsgh->from_smbuf);
```

```
   Diag("MSGH->reply_smbuf = %d",lpmsgh->reply_smbuf);
   Diag("MSGH->size = %d",lpmsgh->size);
}


void print_MSGB(
   CHAR       *lpmsgb)
{
   Diag("MSGB text:");
   Diag("%s",lpmsgb);
}


void print_RST()
{
   lpRSTSEG  lprstseg;
   CHAR       seg_str[255];
   int       *msg_block;
   int        i,j;

   lprstseg = MQEMT->rst_ptr->seg_ptr;
   Diag("Reservation Table");

   for (i=0; i<MQEMT->rst_ptr->num_segs; i++)
    {
     sprintf(seg_str,"Segment %d: ",lprstseg->seg_no);

     msg_block = lprstseg->msg_block;
     for (j=0; j<MQEMT->rst_ptr->msgs_per_seg; j++)
      {
       sprintf(seg_str,"%s %d",seg_str,*msg_block);
       msg_block++;
      }
     lprstseg++;
     Diag("%s",seg_str);
    }

   Diag("");
}


void print_Active_Log_List()
{
   CHAR       seg_str[255];
   int       *address;
   int        i;

   Diag("Active Log List:");
   sprintf(seg_str,"");
   address = Active_LREC_List->address;

   for (i=0; i<Active_LREC_List->max_txns_per_seg; i++)
    {
     if ((i != 0) && ((i%10) == 0))
      {
       Diag("%s",seg_str);
```

```
        sprintf(seg_str,"");
       }
      sprintf(seg_str,"%s %d",seg_str,*address);
      address++;
     }

   Diag("%s",seg_str);
   Diag("");
}


void print_Globals()
{
   Diag("NUM_SEGS = %d", NUM_SEGS);
   Diag("MAX_ELMS = %d", MAX_ELMS);
   Diag("BLOCKS_PER_SEG = %d", BLOCKS_PER_SEG);
   Diag("SEGMENT_SIZE = %d", SEGMENT_SIZE);
   Diag("QEMT_Size = %d", QEMT_Size);
}


void print_PTList()
{
   lpMTLIST  ptl_ptr;
   lpTLIST   tlist;

   ptl_ptr = Pending_TXNs;

   Diag("");
   while (ptl_ptr != NULL)
    {
      tlist = ptl_ptr->ops;
      Diag("");
      Diag("mid.host = %x",ptl_ptr->mid.host);
      Diag("mid.tid = %d",ptl_ptr->mid.tid);

      while (tlist != NULL)
       {
        Diag("qeme_no = %d",tlist->qeme_no);
        tlist = tlist->next;
       }

      ptl_ptr = ptl_ptr->next;
     }


}


void print_QAdm_Stats(
   lpQADMSTATS     lpstats)
{
   Diag("");
   Diag("logical_qname = %s",lpstats->logical_qname);
   Diag("max_entries_limit = %d",lpstats->max_entries_limit);
   Diag("max_entries = %d",lpstats->max_entries);
```

```
    Diag("committed_entries = %d",lpstats->committed_entries);
    Diag("pending_gets = %d",lpstats->pending_gets);
    Diag("pending_puts = %d",lpstats->pending_puts);
    Diag("holey_entries = %d",lpstats->holey_entries);
    Diag("num_free_entries = %d",lpstats->num_free_entries);
    Diag("amt_free_dspace = %d",lpstats->amt_free_dspace);
    Diag("qget_state = %d",lpstats->qget_state);
    Diag("qput_state = %d",lpstats->qput_state);
    Diag("num_puts = %d",lpstats->num_puts);
    Diag("num_gets = %d",lpstats->num_gets);
    Diag("num_commits = %d",lpstats->num_commits);
    Diag("num_aborts = %d",lpstats->num_aborts);
    Diag("num_restarts = %d",lpstats->num_restarts);
    Diag("first_start_time = \t%s",ctime(&lpstats->first_start_time));
    Diag("last_restart_time = \t%s",ctime(&lpstats->last_restart_time));
    Diag("");
}
```

```
/*
 * Copyright(C)1995 MITSUBISHI ELECTRIC ITA.  ALL RIGHTS RESERVED.
 * UNPUBLISHED -- RIGHTS RESERVED UNDER THE COPYRIGHT
 * LAWS OF THE UNITED STATES.  USE OF A COPYRIGHT NOTICE
 * IS PRECAUTIONARY ONLY AND DOES NOT IMPLY PUBLICATION
 * OR DISCLOSURE.
 *
 * THIS SOFTWARE CONTAINS CONFIDENTIAL INFORMATION AND
 * TRADE SECRETS OF MITSUBISHI ELECTRIC ITA.  USE, DISCLOSURE,
 * OR REPRODUCTION IS PROHIBITED WITHOUT THE PRIOR
 * EXPRESS WRITTEN PERMISSION OF MITSUBISHI ELECTRIC ITA.
 *
 * OpenMQ
 *   Module: qutil.c
 *   Author: David Wong 9/8/95
 */


// Include Files
#include "qlib.h"
#include "qserv.h"
#include "qadmin.h"


extern lpQEMT    MQEMT;
extern lpQSTR    MQstate;
extern lpMTLIST Pending_TXNs;
extern int       BLOCKS_PER_SEG;
extern int       SEGMENT_SIZE;
extern int       TOT_MSG_BLOCKS;
extern int       QEMT_Size;
extern int       QEMT_Seq_No;
extern int       LREC_Seq_No;
extern int       NUM_SEGS;
extern int       MAX_ELMS;
extern int       holey_entries;
extern unsigned long QEME_TS;
extern lpLRCLST Active_LREC_List;



// Routine for creating queue file
HANDLE Create_Queue_File(
   CHAR     *FileName,
   CHAR     *qstate,
   int       Max_Elms,
   int       Num_Segs)
{
   HANDLE     Que_File_Handle;
   HANDLE     Que_State_Handle;
   BOOL       Return_Status;
   DWORD      dwBytesWritten;
   DWORD      dwPointer;
   QSTR       que_state;

   // number of segments should divide
   // evenly among total block size
   NUM_SEGS = Num_Segs;
```

```
    MAX_ELMS = Max_Elms;

    Update_Globals();

    if (INVALID_HANDLE_VALUE == (Que_File_Handle = CreateFile(FileName,
                                 GENERIC_READ|GENERIC_WRITE,
                                 FILE_SHARE_READ|FILE_SHARE_WRITE,
                                 NULL,CREATE_NEW,
                                 FILE_ATTRIBUTE_NORMAL,
                                 NULL)))
      return(Que_File_Handle);

    dwPointer = SetFilePointer(Que_File_Handle,
                               SEGMENT_SIZE*NUM_SEGS,
                               NULL,FILE_BEGIN);

    SetEndOfFile(Que_File_Handle);

    que_state.svr_state = QUEUE_SHUTDOWN;
    que_state.num_restarts = 0;
    que_state.num_recov_tries = 0;
    time(&que_state.first_start_time);
    que_state.last_restart_time = 0;
    que_state.last_recov_time = 0;

    if (INVALID_HANDLE_VALUE == (Que_State_Handle = CreateFile(qstate,
                                  GENERIC_READ|GENERIC_WRITE,
                                  FILE_SHARE_READ|FILE_SHARE_WRITE,
                                  NULL,CREATE_NEW,
                                  FILE_ATTRIBUTE_NORMAL,
                                  NULL)))
      return(Que_File_Handle);

    Return_Status = WriteFile(Que_State_Handle,
                              &que_state,
                              sizeof(QSTR),
                              &dwBytesWritten,
                              NULL);

    Return_Status = CloseHandle(Que_State_Handle);

    return(Que_File_Handle);
}



// Routine to update global data structures.
void Update_Globals()
{
    int       elems_per_seg;
    int       temp;

    elems_per_seg = (int)ceil((double)(MAX_ELMS/NUM_SEGS));


    // Size of QEMT; refer to qutil.h
    // Subject to change.

    QEMT_Size = sizeof(QEMT) + (MAX_ELMS*sizeof(QEME)) +
                sizeof(RST)  + (NUM_SEGS*sizeof(RSTSEG)) +
                (NUM_SEGS*elems_per_seg*sizeof(int)) +
                (MAX_ELMS*sizeof(PTLIST));

   // Total number of msg blocks in queue data file.

   TOT_MSG_BLOCKS = (MAX_ELMS*sizeof(QMSG))/BLOCK;


   // Total msg blocks consists of msg header, msg
   // body, and 1 log record for recording PENDING
   // state for each operation in a segment.

   BLOCKS_PER_SEG = elems_per_seg*MSG_ENTRY_BLOCKS;


   // Include enough extra blocks to record PENDING state
   // of transactional GETs and transaction ABORT/COMMIT
   // records.  Set this to theorectical max of MAX_ELMS
   // for each log record type within a single segment.

   temp = GET_LOG_REC_BLOCKS+TERM_LOG_REC_BLOCKS;
   BLOCKS_PER_SEG += (MAX_ELMS*temp);


   // Total number of bytes in each segment.
   // Currently does not include an extent factor
   // for over-capacity.  This will be handled
   // by sysadmin interface later.

   SEGMENT_SIZE = (BLOCKS_PER_SEG*BLOCK)+QEMT_Size;

   return;
}



// Routine for opening an existing queue file
HANDLE Open_Queue_File(
   CHAR *FileName)
{
   HANDLE    Que_File_Handle;

   if (INVALID_HANDLE_VALUE == (Que_File_Handle = CreateFile(FileName,
                                GENERIC_READ|GENERIC_WRITE,
                                FILE_SHARE_READ|FILE_SHARE_WRITE,
                                NULL,OPEN_EXISTING,
                                FILE_ATTRIBUTE_NORMAL,
                                NULL)))
      Fail("Open_Queue_File: could not open file %s - Error = %d",
                             FileName,GetLastError());

   return(Que_File_Handle);
}


// Routine for deleting an existing queue file
```

```
BOOL Delete_Queue_File(
   CHAR    *FileName)
{
   return(DeleteFile(FileName));
}



int Find_Num_Entries(
   lpCNTSTR   *cnt_ptr)
{
   lpCNTSTR  cptr;
   lpQEME    lpqeme;
   int       qeme_no;
   int       status;

   cptr = (CNTSTR *)malloc(sizeof(CNTSTR));
   cptr->committed = 0;
   cptr->holey = 0;

   qeme_no = MQEMT->que_hd_ptr;

   while (qeme_no != MQEMT->que_tl_ptr)
    {
     status = Cycle_QEME(qeme_no,&lpqeme);

     switch(lpqeme->txn_state)
      {
       case COMMIT:
         cptr->committed++;
         break;
       case EMPTY:
         cptr->holey++;
         break;
       case ACTIVE:
       case PENDING:
         break;
      }

     qeme_no = (qeme_no+1)%MQEMT->max_entries;
    }

   status = Cycle_QEME(qeme_no,&lpqeme);

   switch(lpqeme->txn_state)
    {
     case COMMIT:
       cptr->committed++;
       break;
     case EMPTY:
       cptr->holey++;
       break;
     case ACTIVE:
     case PENDING:
       break;
    }

   *cnt_ptr = cptr;
   return(QSUCCESS);
```

```
}



// Find QEM entry based on MID
int Find_QEME(
   lpMID     mid,
   int      *qeme_no2,
   lpQEME   *lpqeme2)
{
   lpQEME    lpqeme;
   int       qeme_no;
   int       valid = 1;
   int       found = 0;
   int       status;

   if (Check_Queue_Empty(MQEMT) == FALSE)
    {
     qeme_no = MQEMT->que_hd_ptr;
     status = Cycle_QEME(qeme_no,&lpqeme);

     while (valid && !found)
      {
       if ((lpqeme->mid.host == mid->host) &&
           (lpqeme->mid.tid  == mid->tid) &&
           (lpqeme->mid.uid  == mid->uid))
        {
         *qeme_no2 = qeme_no;
         *lpqeme2 = lpqeme;
         found = 1;
        }
        else
        {
         if (qeme_no == MQEMT->que_tl_ptr)
           valid = 0;
           else
           {
            qeme_no = (qeme_no+1)%MQEMT->max_entries;
            status = Cycle_QEME(qeme_no,&lpqeme);
           }
        }
      }
    }

   if (found)
     return(QSUCCESS);
     else return(QFAIL);
}



int Cycle_QEME(
   int      qeme_no,
   lpQEME *lpqeme2)

{
   lpQEME    lpqeme;
   int      i;

   lpqeme = MQEMT->qeme_ptr;
   for (i=0; i<qeme_no; i++)
      lpqeme++;
   *lpqeme2 = lpqeme;

   return (QSUCCESS);
}


// Routine for determining next valid
// QEM entry for a PUT given that some
// entries have pending GETs.
//
// Not currently used since it is
// expensive.  Currently keeping
// track of the timestamp of last
// pending get operation.
//
void Find_Last_Pending_Get(
   int      *qeme_no)
{
   unsigned long latest_ts=0;
   lpQEME   lpqeme;
   int      latest_qeme=NIL;
   int      qeme_no2;
   int      status;


   for (qeme_no2=0; qeme_no2<MQEMT->max_entries; qeme_no2++)
     {
     status = Cycle_QEME(qeme_no2,&lpqeme);

     if ((lpqeme->txn_state == PENDING) &&
         (lpqeme->timestamp > latest_ts))
       {
       latest_ts = lpqeme->timestamp;
       latest_qeme = qeme_no2;
       }
     }

   *qeme_no = latest_qeme;
}



// Routine fixes queue ptrs after aborts.
//
// Issues:
//  - might need to handle deadlock
//
void Fix_Que_Ptrs_on_Aborted_Put(
   lpQEME  lpqeme,
   int     qeme_no)
{
   HANDLE  hQHD,hQTL;
   lpQEME  lpqeme2;
```

```
    int     qeme_no2;
    int     status;

    // Protocol is to obtain hQHD first,
    // then hQTL.
    hQHD = OpenMutex(SYNCHRONIZE,FALSE,QUE_HD_PTR_LOCK);
    if (!hQHD)
      Diag("QS_QCommit: Can't OpenMutex for queue head pointer lock");
    if ((status = WaitForSingleObject(hQHD,QUE_LOCK_TIMEOUT)) !=
                                         WAIT_OBJECT_0)
      Diag("QS_QCommit: Synch wait for queue head pointer lock failed");

    hQTL = OpenMutex(SYNCHRONIZE,FALSE,QUE_TL_PTR_LOCK);
    if (!hQTL)
      Diag("QS_QCommit: Can't OpenMutex for queue tail pointer lock");
    if ((status = WaitForSingleObject(hQTL,QUE_LOCK_TIMEOUT)) !=
                                         WAIT_OBJECT_0)
      Diag("QS_QCommit: Synch wait for queue tail pointer lock failed");

    if (qeme_no == MQEMT->que_hd_ptr)
     {
      if (MQEMT->que_hd_ptr == MQEMT->que_tl_ptr)
       {
        MQEMT->que_hd_ptr = NIL;
        MQEMT->que_tl_ptr = NIL;
       }
        else MQEMT->que_hd_ptr = (MQEMT->que_hd_ptr+1)%MQEMT->max_entries;
     }
     else if (qeme_no == MQEMT->que_tl_ptr)
     {
      qeme_no2 = qeme_no;
      lpqeme2 = lpqeme;
      while ((lpqeme2->txn_state == EMPTY) &&
             (qeme_no2 != MQEMT->que_hd_ptr))
       {
        qeme_no2 = (MQEMT->max_entries+qeme_no2-1)%MQEMT->max_entries;
        status = Cycle_QEME(qeme_no2,&lpqeme2);
       }

      if ((qeme_no2 == MQEMT->que_hd_ptr) &&
          (lpqeme2->txn_state == EMPTY))
       {
        MQEMT->que_hd_ptr = NIL;
        MQEMT->que_tl_ptr = NIL;
       }
        else MQEMT->que_tl_ptr = qeme_no2;
     }

   ReleaseMutex(hQTL);
   ReleaseMutex(hQHD);


/*
*/
Diag("Fix_on_Aborted_Put: qeme_no = %d, head = %d, tail = %d",
     qeme_no,MQEMT->que_hd_ptr,MQEMT->que_tl_ptr);
}
```

```
// Routine fixes queue ptrs after aborts.
// Use QEME timestamps for comparison.
//
// Issues:
//  - might need to handle deadlock
//
void Fix_Que_Ptrs_on_Aborted_Get(
   int      qeme_no)
{
   HANDLE   hQHD,hQTL;
   lpQEME   lpqeme,hd_qeme,tl_qeme;
   int      status;

   hQHD = OpenMutex(SYNCHRONIZE,FALSE,QUE_HD_PTR_LOCK);
   if (!hQHD)
     Diag("QS_QCommit: Can't OpenMutex for queue head pointer lock");
   if ((status = WaitForSingleObject(hQHD,QUE_LOCK_TIMEOUT)) !=
                                              WAIT_OBJECT_0)
     Diag("QS_QCommit: Synch wait for queue head pointer lock failed");

   hQTL = OpenMutex(SYNCHRONIZE,FALSE,QUE_TL_PTR_LOCK);
   if (!hQTL)
     Diag("QS_QCommit: Can't OpenMutex for queue tail pointer lock");
   if ((status = WaitForSingleObject(hQTL,QUE_LOCK_TIMEOUT)) !=
                                              WAIT_OBJECT_0)
     Diag("QS_QCommit: Synch wait for queue tail pointer lock failed");

   if (MQEMT->que_hd_ptr == NIL)
    {
     MQEMT->que_hd_ptr = qeme_no;
     MQEMT->que_tl_ptr = qeme_no;
    }
    else
    {
     status = Cycle_QEME(qeme_no,&lpqeme);
     status = Cycle_QEME(MQEMT->que_hd_ptr,&hd_qeme);
     status = Cycle_QEME(MQEMT->que_tl_ptr,&tl_qeme);

     if (lpqeme->timestamp < hd_qeme->timestamp)
       MQEMT->que_hd_ptr = qeme_no;
       else if (lpqeme->timestamp > tl_qeme->timestamp)
         MQEMT->que_tl_ptr = qeme_no;
    }

   ReleaseMutex(hQTL);
   ReleaseMutex(hQHD);

/*
*/
Diag("Fix_on_Aborted_Get: qeme_no = %d, head = %d, tail = %d",
     qeme_no,MQEMT->que_hd_ptr,MQEMT->que_tl_ptr);

}
```

```
// Routine for fetching next
// valid GET QEM entry slot.
// Not currently used.
int Fetch_Next_Get_Slot(
   int      *qeme_no)
{
   lpQEME   lpqeme;
   int      done=0;
   int      found=0;
   int      qeme_no2;
   int      status;

   qeme_no2 = MQEMT->que_hd_ptr;
   while (!done && !found)
    {
     status = Cycle_QEME(qeme_no2,&lpqeme);
     if ((lpqeme->mode == PUT_MODE) &&
         (lpqeme->txn_state == COMMIT))
      {
       *qeme_no = qeme_no2;
        found = 1;
      }
      else
      {
       if (qeme_no2 == MQEMT->que_tl_ptr)
         done = 1;
         else qeme_no2 = (qeme_no2+1)%MQEMT->max_entries;
      }
    }

   if (found)
     return(QSUCCESS);
     else return(QFAIL);

}



// Routine for fetching next
// valid PUT QEM entry slot.
// Not currently used.
int Fetch_Next_Put_Slot(
   int      *qeme_no)
{
   lpQEME   lpqeme;
   int      done=0;
   int      found=0;
   int      qeme_no2;
   int      status;

   qeme_no2 = MQEMT->que_tl_ptr;
   while (!done)
    {
     status = Cycle_QEME(qeme_no2,&lpqeme);
     if (lpqeme->txn_state == EMPTY)
      {
       *qeme_no = qeme_no2;
        found = 1;
      }
      else
      {
       if (qeme_no2 == MQEMT->que_hd_ptr)
         done = 1;
         else qeme_no2 = (qeme_no2+1)%MQEMT->max_entries;
      }
    }

   if (found)
     return(QSUCCESS);
     else return(QFAIL);
}



// Routine for resetting QEME timestamps
// starting from 1 after after recovery or
// at queue server shutdown.  Point is to
// guide against timestamp ever exceeding
// the max of an unsigned long.

void Update_QEME_TS()
{
   HANDLE   hQEME;
   lpQEME   lpqeme;
   int      status;
   int      qeme_no;
   int      done=0;

   hQEME = OpenMutex(SYNCHRONIZE,FALSE,QEME_TS_GEN_LOCK);
   if (!hQEME)
     Diag("Update_QEME_TS: Can't OpenMutex for QEME_TS Lock");
   if ((status = WaitForSingleObject(hQEME,QUE_LOCK_TIMEOUT)) !=
                                           WAIT_OBJECT_0)
     Diag("Update_QEME_TS: Synch wait for QEME_TS Lock failed");

   QEME_TS = 1;
   qeme_no = MQEMT->que_hd_ptr;

   while (!done)
    {
     status = Cycle_QEME(qeme_no,&lpqeme);
     lpqeme->timestamp = QEME_TS++;
     if (qeme_no == MQEMT->que_tl_ptr)
       done = 1;
       else qeme_no = (qeme_no+1)%MQEMT->max_entries;
    }

   ReleaseMutex(hQEME);
}
```

```
// Routine for finding most current
// QEM Table in queue data file.

int Find_Latest_QEM(
   HANDLE  Que_File_Handle,
   lpQEMT  *lpqemt2,
   int     *seg_num)
{
   lpQEMT   lpqemt;
   lpQEME   lpqeme;
   lpRST    lprst;
   lpRSTSEG lprstseg;
   lpPTLIST ptl_ptr;
   BOOL     Return_Status;
   DWORD    dwPointer;
   DWORD    dwBytesRead;
   SN       curr_qem_sn;
   SN       prev_qem_sn;
   int      tseq_no=0;
   int      done=0,i;
   int      elems_per_seg;

   curr_qem_sn.timestamp = 0;
   curr_qem_sn.counter = 0;
   prev_qem_sn.timestamp = 0;
   prev_qem_sn.counter = 0;

   lpqemt  = (QEMT *)malloc(sizeof(QEMT));

   dwPointer = SetFilePointer(Que_File_Handle,
                              0,NULL,FILE_BEGIN);

   if (!(Return_Status = ReadFile(Que_File_Handle,lpqemt,
                                  sizeof(QEMT),&dwBytesRead,NULL)))
     Fail("Find_Latest_QEM: could not read QEMT buffer - Error = %d",
                              GetLastError());

   curr_qem_sn.timestamp = lpqemt->qem_sn.timestamp;
   curr_qem_sn.counter = lpqemt->qem_sn.counter;

   NUM_SEGS = lpqemt->num_segs;
   MAX_ELMS = lpqemt->max_entries;
   tseq_no++;

   elems_per_seg = (int)ceil((double)(MAX_ELMS/NUM_SEGS));

   Update_Globals();

   lpqeme  = (QEME *)malloc(MAX_ELMS*sizeof(QEME));

   lprst = (RST *)malloc(sizeof(RST));

   lprstseg = (RSTSEG *)malloc(NUM_SEGS*sizeof(RSTSEG));

   lprst->num_segs = NUM_SEGS;
   lprst->msgs_per_seg = elems_per_seg;
   lprst->seg_ptr = lprstseg;
```

```
for (i=0; i<NUM_SEGS; i++)
 {
  lprstseg->seg_no = i;
  lprstseg->msg_block =
          (int *)malloc(elems_per_seg*sizeof(int));
  lprstseg++;
 }

ptl_ptr = (PTLIST *)malloc(MAX_ELMS*sizeof(PTLIST));

while (!done)
 {
  dwPointer = SetFilePointer(Que_File_Handle,
                             tseq_no*SEGMENT_SIZE,
                             NULL,FILE_BEGIN);

  if (!(Return_Status = ReadFile(Que_File_Handle,lpqemt,
                                 sizeof(QEMT),&dwBytesRead,NULL)))
    Fail("Find_Latest_QEM: could not read QEMT buffer - Error = %d",
                          GetLastError());

  if ((lpqemt->marker == QEMTMARK) &&
       Bigger_Seq_No(&lpqemt->qem_sn,&curr_qem_sn))
   {
    prev_qem_sn.timestamp = curr_qem_sn.timestamp;
    prev_qem_sn.counter = curr_qem_sn.counter;
    curr_qem_sn.timestamp = lpqemt->qem_sn.timestamp;
    curr_qem_sn.counter = lpqemt->qem_sn.counter;
    tseq_no = (tseq_no+1)%NUM_SEGS;
   }
    else done = 1;
 }

tseq_no = (NUM_SEGS+tseq_no-1)%NUM_SEGS;

dwPointer = SetFilePointer(Que_File_Handle,
                           tseq_no*SEGMENT_SIZE,
                           NULL,FILE_BEGIN);

if (!(Return_Status = ReadFile(Que_File_Handle,lpqemt,
                               sizeof(QEMT),&dwBytesRead,NULL)))
  Fail("Find_Latest_QEM: could not read QEMT buffer - Error = %d",
                        GetLastError());

if (!(Return_Status = ReadFile(Que_File_Handle,lpqeme,
                               MAX_ELMS*sizeof(QEME),
                               &dwBytesRead,NULL)))
  Fail("Find_Latest_QEM: could not read QEME buffer - Error = %d",
                        GetLastError());

if (!(Return_Status = ReadFile(Que_File_Handle,lprst,
                               sizeof(RST),&dwBytesRead,NULL)))
  Fail("Find_Latest_QEM: could not read in RST structure - Error = %d",
                   GetLastError());

lprstseg = lprst->seg_ptr;
for (i=0; i<lprst->num_segs; i++)
 {
  if (!(Return_Status = ReadFile(Que_File_Handle,
                                     lprstseg,sizeof(RSTSEG),
                                     &dwBytesRead,NULL)))
          Fail("Find_Latest_QEM: could not read in RSTSEG structure - Error = %d",
                          GetLastError());

        if (!(Return_Status = ReadFile(Que_File_Handle,
                                     lprstseg->msg_block,
                                     lprst->msgs_per_seg*sizeof(int),
                                     &dwBytesRead,NULL)))
          Fail("Find_Latest_QEM: could not read in msg_block entries - Error = %d",
                          GetLastError());

        lprstseg++;
       }

      if (!(Return_Status = ReadFile(Que_File_Handle,ptl_ptr,
                                   MAX_ELMS*sizeof(PTLIST),
                                   &dwBytesRead,NULL)))
        Fail("Find_Latest_QEM: could not read PTLIST buffer - Error = %d",
                            GetLastError());

      lpqemt->qeme_ptr = lpqeme;
      lpqemt->rst_ptr  = lprst;
      lpqemt->ptl_ptr  = ptl_ptr;

      *lpqemt2 = lpqemt;
      *seg_num  = tseq_no;

      return(QSUCCESS);
   }




   int Create_QEMT(
      lpQEMT  *lpqemt,
      int      max_elms)
   {
      lpQEMT   lpqemt2;
      lpRSTSEG lprstseg;
      int      i,elems_per_seg;

      elems_per_seg = (int)ceil((double)(MAX_ELMS/NUM_SEGS));


      lpqemt2 = (QEMT *)malloc(sizeof(QEMT));

      lpqemt2->qeme_ptr = (QEME *)malloc(max_elms*sizeof(QEME));

      lpqemt2->rst_ptr = (RST *)malloc(sizeof(RST));
      lpqemt2->rst_ptr->seg_ptr =
                        (RSTSEG *)malloc(NUM_SEGS*sizeof(RSTSEG));

      lpqemt2->rst_ptr->num_segs = NUM_SEGS;
      lpqemt2->rst_ptr->msgs_per_seg = elems_per_seg;

      lprstseg = lpqemt2->rst_ptr->seg_ptr;
      for (i=0; i<NUM_SEGS; i++)
       {
```

```
      lprstseg->seg_no = i;
      lprstseg->msg_block =
              (int *)malloc(elems_per_seg*sizeof(int));
      lprstseg++;
     }

   lpqemt2->ptl_ptr = (PTLIST *)malloc(max_elms*sizeof(PTLIST));

   *lpqemt = lpqemt2;

   return(QSUCCESS);
}



int Init_QEMT(
   lpQEMT  lpqemt,
   int     max_elms,
   int     ext_elms,
   int     num_segs)
{
   lpQEMT    lpqemt2;
   lpQEME    lpqeme;
   lpPTLIST  ptl_ptr;
   lpRSTSEG  lprstseg;
   int      *msg_block;
   int       i,j;

   lpqemt2 = lpqemt;

   Gen_QEM_Seq_No(&lpqemt2->qem_sn);

   lpqemt2->marker = QEMTMARK;
   lpqemt2->num_segs = num_segs;
   lpqemt2->max_entries = max_elms;
   lpqemt2->max_entries_limit = max_elms+ext_elms;
   lpqemt2->next_avail_block.segment = 0;
   lpqemt2->next_avail_block.block   = 0;

   lpqemt2->que_hd_ptr = NIL;
   lpqemt2->que_tl_ptr = NIL;

   lpqemt2->qget_state = ENABLED;
   lpqemt2->qput_state = ENABLED;

   lpqeme = lpqemt2->qeme_ptr;

   for (i=0; i<lpqemt2->max_entries; i++)
     {
      lpqeme->index          = i;
      lpqeme->timestamp      = 0;
      lpqeme->mid.host       = 0;
      lpqeme->mid.tid        = 0;
      lpqeme->mid.uid        = 0;
      lpqeme->mode           = 0;
      lpqeme->flags          = 0;
      lpqeme->priority       = 0;
      lpqeme->txn_state      = EMPTY;
      lpqeme->vote           = 0;
      lpqeme->offset.segment = 0;
      lpqeme->offset.block   = 0;
      lpqeme++;
     }

    lprstseg = lpqemt2->rst_ptr->seg_ptr;
    for (i=0; i<NUM_SEGS; i++)
     {
      msg_block = lprstseg->msg_block;
      for (j=0; j<lpqemt2->rst_ptr->msgs_per_seg; j++)
       {
        *msg_block = NIL;
        msg_block++;
       }
      lprstseg++;
     }

    lpqemt2->num_pts = 0;
    ptl_ptr = lpqemt2->ptl_ptr;

    for (i=0; i<lpqemt2->max_entries; i++)
     {
      ptl_ptr->mid.host = 0;
      ptl_ptr->mid.tid  = 0;
      ptl_ptr->qeme_no  = NIL;
      ptl_ptr++;
     }

    return(QSUCCESS);
}


int Write_QEMT(
    HANDLE  Que_File_Handle,
    int     seg_no,
    lpQEMT  lpqemt)
{
    lpRSTSEG lprstseg;
    BOOL     Return_Status;
    DWORD    dwPointer;
    DWORD    dwBytesWritten;
    int      status;
    int      i;

    status = Gen_LREC_Seq_No(&lpqemt->lrec_sn);

    Conv_from_MTlist();

    dwPointer = SetFilePointer(Que_File_Handle,
                               (LONG)(seg_no*SEGMENT_SIZE),
                               NULL,FILE_BEGIN);

    if (!(Return_Status = WriteFile(Que_File_Handle,lpqemt,sizeof(QEMT),
                               &dwBytesWritten,NULL)))
      Fail("Write_QEMT: could not write out QEM Table - Error = %d",
                        GetLastError());

    if (!(Return_Status = WriteFile(Que_File_Handle,lpqemt->qeme_ptr,
                               lpqemt->max_entries*sizeof(QEME),
                          &dwBytesWritten,NULL)))
      Fail("Write_QEMT: could not write out QEM entries - Error = %d",
                        GetLastError());

    if (!(Return_Status = WriteFile(Que_File_Handle,
                                    lpqemt->rst_ptr,
                                    sizeof(RST),
                                    &dwBytesWritten,NULL)))
      Fail("Write_QEMT: could not write out RST structure - Error = %d",
                        GetLastError());

    lprstseg = lpqemt->rst_ptr->seg_ptr;
    for (i=0; i<lpqemt->rst_ptr->num_segs; i++)
     {
      if (!(Return_Status = WriteFile(Que_File_Handle,
                                      lprstseg,
                                      sizeof(RSTSEG),
                                      &dwBytesWritten,NULL)))
        Fail("Write_QEMT: could not write out RSTSEG structure - Error = %d",
                          GetLastError());

      if (!(Return_Status = WriteFile(Que_File_Handle,
                                      lprstseg->msg_block,
                                      lpqemt->rst_ptr->msgs_per_seg*sizeof(int),
                                      &dwBytesWritten,NULL)))
        Fail("Write_QEMT: could not write out msg_block entries - Error = %d",
                          GetLastError());

      lprstseg++;
     }

    if (!(Return_Status = WriteFile(Que_File_Handle,
                          lpqemt->ptl_ptr,lpqemt->num_pts*sizeof(PTLIST),
                          &dwBytesWritten,NULL)))
      Fail("Write_QEMT: could not write out PT list - Error = %d",
                        GetLastError());

    // Now, write out the most current log record timestamp.

    return(QSUCCESS);
}



int Retrieve_QEMT(
    HANDLE   Que_File_Handle,
    int      seg_no,
    lpQEMT   *lpqemt2)
{
    lpQEMT   lpqemt;
    lpQEME   lpqeme;
    lpRST    lprst;
    lpRSTSEG lprstseg;
    lpPTLIST ptl_ptr;
    BOOL     Return_Status;
    DWORD    dwPointer;
    DWORD    dwBytesRead;
    int      elems_per_seg;
    int      i;
```

```
elems_per_seg = (int)ceil((double)(MAX_ELMS/NUM_SEGS));

lpqemt  = (QEMT *)malloc(sizeof(QEMT));

lpqeme  = (QEME *)malloc(MAX_ELMS*sizeof(QEME));

lprst = (RST *)malloc(sizeof(RST));

lprstseg = (RSTSEG *)malloc(NUM_SEGS*sizeof(RSTSEG));

lprst->num_segs = NUM_SEGS;
lprst->msgs_per_seg = elems_per_seg;

lprstseg = lprst->seg_ptr;
for (i=0; i<NUM_SEGS; i++)
 {
  lprstseg->seg_no = i;
  lprstseg->msg_block =
          (int *)malloc(elems_per_seg*sizeof(int));
  lprstseg++;
 }

ptl_ptr = (PTLIST *)malloc(MAX_ELMS*sizeof(PTLIST));

dwPointer = SetFilePointer(Que_File_Handle,
                           (LONG)(seg_no*SEGMENT_SIZE),
                           NULL,FILE_BEGIN);


if (!(Return_Status = ReadFile(Que_File_Handle,lpqemt,
                               sizeof(QEMT),&dwBytesRead,NULL)))
  Fail("Retrieve_QEMT: could not read QEMT Table - Error = %d",
                       GetLastError());

if (!(Return_Status = ReadFile(Que_File_Handle,lpqeme,
                               lpqemt->max_entries*sizeof(QEME),
                               &dwBytesRead,NULL)))
  Fail("Retrieve_QEMT: could not read QEME entries - Error = %d",
                       GetLastError());

if (!(Return_Status = ReadFile(Que_File_Handle,lprst,
                               sizeof(RST),&dwBytesRead,NULL)))
  Fail("Write_QEMT: could not write out RST structure - Error = %d",
                    GetLastError());

lprstseg = lprst->seg_ptr;
for (i=0; i<lprst->num_segs; i++)
 {
  if (!(Return_Status = ReadFile(Que_File_Handle,
                                 lprstseg,sizeof(RSTSEG),
                                 &dwBytesRead,NULL)))
    Fail("Write_QEMT: could not write out RSTSEG structure - Error = %d",
                      GetLastError());

  if (!(Return_Status = ReadFile(Que_File_Handle,
                                 lprstseg->msg_block,
                                 lprst->msgs_per_seg*sizeof(int),
                                 &dwBytesRead,NULL)))
    Fail("Write_QEMT: could not write out msg_block entries - Error = %d",
                            GetLastError());

      lprstseg++;
     }

   if (!(Return_Status = ReadFile(Que_File_Handle,ptl_ptr,
                                  lpqemt->num_pts*sizeof(PTLIST),
                                  &dwBytesRead,NULL)))
     Fail("Retrieve_QEMT: could not read PT list - Error = %d",
                          GetLastError());

   lpqemt->qeme_ptr = lpqeme;
   lpqemt->rst_ptr  = lprst;
   lpqemt->ptl_ptr  = ptl_ptr;

   *lpqemt2 = lpqemt;

   // Conv_to_MTlist();

   return(QSUCCESS);
}


int Retrieve_Msg_Hdr(
   HANDLE  Que_File_Handle,
   BLAD    Msg_Addr,
   lpMSGH  Msg_Hdr)
{
   BOOL     Return_Status;
   DWORD    dwBytesRead;
   DWORD    dwPointer;
   LONG     Dist_to_Move;

   Conv_Addr(&Dist_to_Move,&Msg_Addr);

   dwPointer = SetFilePointer(Que_File_Handle,
                              Dist_to_Move,
                              NULL,
                              FILE_BEGIN);

   Return_Status = ReadFile(Que_File_Handle,
                            Msg_Hdr,
                            MSG_HDR_SIZE,
                            &dwBytesRead,
                            NULL);

   if ((Return_Status == TRUE) && (dwBytesRead == MSG_HDR_SIZE))
     return(QSUCCESS);
     else return(QFAIL);
}


int Write_Msg_Hdr(
   HANDLE  Que_File_Handle,
   BLAD    Msg_Addr,
   lpMSGH  Msg_Hdr)
{
   BOOL      Return_Status;
   DWORD     dwBytesWritten;
   DWORD     dwPointer;
   LONG      Dist_to_Move;

   Conv_Addr(&Dist_to_Move,&Msg_Addr);

   dwPointer = SetFilePointer(Que_File_Handle,
                              Dist_to_Move,
                              NULL,
                              FILE_BEGIN);

   Return_Status = WriteFile(Que_File_Handle,
                             Msg_Hdr,
                             MSG_HDR_SIZE,
                             &dwBytesWritten,
                             NULL);

   if ((Return_Status == TRUE) && (dwBytesWritten == MSG_HDR_SIZE))
     return(QSUCCESS);
     else return(QFAIL);
}


int Retrieve_Msg_Body(
   HANDLE  Que_File_Handle,
   BLAD    Msg_Addr,
   DWORD   Msg_Length,
   CHAR   *Msg_Body)
{
   BOOL      Return_Status;
   LONG      Dist_to_Move;
   DWORD     dwBytesRead;
   DWORD     dwPointer;


   Conv_Addr(&Dist_to_Move,&Msg_Addr);

   dwPointer = SetFilePointer(Que_File_Handle,
                              Dist_to_Move,
                              NULL,
                              FILE_BEGIN);

   Return_Status = ReadFile(Que_File_Handle,
                            Msg_Body,
                            Msg_Length,
                            &dwBytesRead,
                            NULL);

   if ((Return_Status == TRUE) && (dwBytesRead == Msg_Length))
     return(QSUCCESS);
     else return(QFAIL);
}


int Write_Msg_Body(
   HANDLE  Que_File_Handle,
   BLAD    Msg_Addr,
   DWORD   Msg_Length,
   CHAR   *Msg_Body)
{
   BOOL      Return_Status;
   LONG      Dist_to_Move;
   DWORD     dwBytesWritten;
   DWORD     dwPointer;


   Conv_Addr(&Dist_to_Move,&Msg_Addr);

   dwPointer = SetFilePointer(Que_File_Handle,
                              Dist_to_Move,
                              NULL,
                              FILE_BEGIN);

   Return_Status = WriteFile(Que_File_Handle,
                             Msg_Body,
                             Msg_Length,
                             &dwBytesWritten,
                             NULL);

   if ((Return_Status == TRUE) && (dwBytesWritten == Msg_Length))
     return(QSUCCESS);
     else return(QFAIL);
}


int Retrieve_Msg(
   HANDLE  Que_File_Handle,
   BLAD    Msg_Addr,
   lpQMSG  qmsg)
{
   BOOL      Return_Status;
   LONG      Dist_to_Move;
   DWORD     dwBytesRead;
   DWORD     dwPointer;


   Conv_Addr(&Dist_to_Move,&Msg_Addr);

   dwPointer = SetFilePointer(Que_File_Handle,
                              Dist_to_Move,
                              NULL,FILE_BEGIN);

   Return_Status = ReadFile(Que_File_Handle,
                            qmsg,sizeof(QMSG),
                            &dwBytesRead,NULL);

   if ((Return_Status == TRUE) && (dwBytesRead == sizeof(QMSG)))
     return(QSUCCESS);
     else return(QFAIL);
}


int Retrieve_Log_Rec(
   HANDLE  Que_File_Handle,
   BLAD    LREC_Addr,
   lpLREC  LREC_ptr)
{
   BOOL     Return_Status;
   DWORD    dwBytesRead;
   LONG     Dist_to_Move;
   DWORD    dwPointer;

   Conv_Addr(&Dist_to_Move,&LREC_Addr);

   dwPointer = SetFilePointer(Que_File_Handle,
                              Dist_to_Move,
                              NULL,
                              FILE_BEGIN);

   Return_Status = ReadFile(Que_File_Handle,
                            LREC_ptr,
                            LOG_REC_BLOCKS*BLOCK,
                            &dwBytesRead,
                            NULL);

   if ((Return_Status == TRUE) &&
      (dwBytesRead == (LOG_REC_BLOCKS*BLOCK)))
     return(QSUCCESS);
     else return(QFAIL);
}


int Write_Log_Rec(
   HANDLE  Que_File_Handle,
   BLAD    LREC_Addr,
   lpLREC  LREC_ptr)
{
   BOOL     Return_Status;
   DWORD    dwBytesWritten;
   DWORD    dwPointer;
   LONG     Dist_to_Move;

   Conv_Addr(&Dist_to_Move,&LREC_Addr);

   dwPointer = SetFilePointer(Que_File_Handle,
                              Dist_to_Move,
                              NULL,
                              FILE_BEGIN);

   Return_Status = WriteFile(Que_File_Handle,
                             LREC_ptr,
                             LOG_REC_BLOCKS*BLOCK,
                             &dwBytesWritten,
                             NULL);

   if (Return_Status == FALSE)
     Diag("Error Code = %d",GetLastError());


   if ((Return_Status == TRUE) &&
      (dwBytesWritten == (LOG_REC_BLOCKS*BLOCK)))
     return(QSUCCESS);
     else return(QFAIL);
}
```

```
void Init_Active_LREC_List()
{
   int      i;
   int     *address;

   address = Active_LREC_List->address;
   for (i=0; i<Active_LREC_List->max_txns_per_seg; i++)

     {
      *address = NIL;
       address++;
     }
}



int msg_entry_comp(
   const void *arg1,
   const void *arg2)
{
   if (*(int*)arg1 < *(int*)arg2)
     return(-1);
     else if (*(int*)arg1 == *(int*)arg2)
       return(0);
       else if (*(int*)arg1 > *(int*)arg2)
         return(1);
}



void Sort_RST_Entries(
   int      seg_no)
{
   lpRSTSEG lprstseg;
   int      i;

   lprstseg = MQEMT->rst_ptr->seg_ptr;
   for (i=0; i<seg_no; i++)
     lprstseg++;

   qsort((void *)lprstseg->msg_block,
         (size_t)MQEMT->rst_ptr->msgs_per_seg,
         sizeof(int),msg_entry_comp);
}



int Add_RST_Entry(
   int      seg_no,
   int      entry_value)
{
   lpRSTSEG lprstseg;
   int      found=0;
   int       i=0;
   int      *msg_block;

   lprstseg = MQEMT->rst_ptr->seg_ptr;
   for (i=0; i<seg_no; i++)
     lprstseg++;

   msg_block = lprstseg->msg_block;
   while (!found && (i < MQEMT->rst_ptr->msgs_per_seg))
    {
     if (*msg_block == NIL)
       found = 1;
       else
       {
        i++;
        msg_block++;
       }
    }

   if (found)
     *msg_block = entry_value;

   return(found);
}



int Del_RST_Entry(
   int      seg_no,
   int      entry_value)
{
   lpRSTSEG lprstseg;
   int       found=0;
   int       i;
   int      *msg_block;

   lprstseg = MQEMT->rst_ptr->seg_ptr;
   for (i=0; i<seg_no; i++)
     lprstseg++;

   i = 0;
   msg_block = lprstseg->msg_block;
   while (!found && (i < MQEMT->rst_ptr->msgs_per_seg))
    {
     if (*msg_block == entry_value)
       found = 1;
       else
       {
        i++;
        msg_block++;
       }
    }

   if (found)
     *msg_block = NIL;
   return(found);
}
```

```
int Entry_in_RST(
   int      seg_no,
   int      entry_value)
{
   lpRSTSEG lprstseg;
   int       found=0;
   int       i=0;
   int      *msg_block;

   lprstseg = MQEMT->rst_ptr->seg_ptr;
   for (i=0; i<seg_no; i++)
     lprstseg++;

   msg_block = lprstseg->msg_block;
   while (!found && (i < MQEMT->rst_ptr->msgs_per_seg))
    {
     if (*msg_block == entry_value)
       found = 1;
       else
       {
        i++;
        msg_block++;
       }
    }

   return(found);
}



int log_entry_comp(
   const void *arg1,
   const void *arg2)
{
   if (*(int*)arg1 < *(int*)arg2)
     return(1);
     else if (*(int*)arg1 == *(int*)arg2)
       return(0);
       else if (*(int*)arg1 > *(int*)arg2)
         return(-1);
}



void Sort_Log_Entries()
{
   qsort((void *)Active_LREC_List->address,
         (size_t)Active_LREC_List->max_txns_per_seg,
         sizeof(int),log_entry_comp);
}



int Add_Log_Entry(
   int      entry_value)
```

```
{
   int       found=0;
   int       i=0;
   int      *address;

   address = Active_LREC_List->address;

   while (!found && (i<Active_LREC_List->max_txns_per_seg))
    {
     if (*address == NIL)
       found = 1;
       else
       {
        i++;
        address++;
       }
    }

   if (found)
     *address = entry_value;

   return(found);
}



int Del_Log_Entry(
   int      entry_value)
{
   int       found=0;
   int       i=0;
   int      *address;

   address = Active_LREC_List->address;

   while (!found && (i<Active_LREC_List->max_txns_per_seg))
    {
     if (*address == entry_value)
       found = 1;
       else
       {
        i++;
        address++;
       }
    }

   if (found)
     *address = NIL;

   return(found);
}



int In_Log_Rec_List(
   int      start,
   int      end,
   int     *hit)
```

```
{
   int      found=0;
   int      i=0;
   int     *address;

   address = Active_LREC_List->address;
   while (!found &&
         (i<Active_LREC_List->max_txns_per_seg) &&
         (*address != NIL))
    {
     if ((*address >= start) && (*address <= end))
       {
        *hit = *address;
         found = 1;
       }
       else
       {
        i++;
        address++;
       }
    }

   return(found);
}



int Cycle_RST_Seg(
   int        seg_no,
   lpRSTSEG *lprstseg2)
{
   lpRSTSEG lprstseg;
   int       i;

   lprstseg = MQEMT->rst_ptr->seg_ptr;
   for (i=0; i<seg_no; i++)
      lprstseg++;
   *lprstseg2 = lprstseg;

   return (QSUCCESS);
}



int Get_Next_BLAD(
   HANDLE  Que_File_Handle,
   int      type,
   BLAD    *offset)
{
   lpRSTSEG lprstseg;
   int       end=0;
   int       found=0;
   int       notfull=1;
   int       moved=0;
   int       cycled=0;
   int       start=1;
   int       holes=0;
   int       status;
```

```
    int      conflict;
    int     *msg_block,*msg_block2;
    int      msg_num;
    int      curr_block;
    int      orig_seg,curr_seg;
    int      blocks_needed;

    if (type == MSG_WRITE)
      blocks_needed = MSG_ENTRY_BLOCKS;
      else blocks_needed = LOG_REC_BLOCKS;

    orig_seg = MQEMT->next_avail_block.segment;

    curr_seg = MQEMT->next_avail_block.segment;
    curr_block = MQEMT->next_avail_block.block;

    status = Cycle_RST_Seg(curr_seg,&lprstseg);

    msg_num = 0;
    msg_block = lprstseg->msg_block;
    msg_block2 = msg_block;
    msg_block2++;


    while (notfull && !found)
     {
      while ((msg_num < MQEMT->rst_ptr->msgs_per_seg) &&
             (*msg_block == NIL))
       {
        msg_num++;
        msg_block++;
        holes++;
       }


/*
      Diag("segment = %d, curr_block = %d, msg_block = %d, msg_block2 = %d, msg_n
*/


      if (msg_num == MQEMT->rst_ptr->msgs_per_seg)
       {
        if (moved)
          curr_block = 0;

        if ((curr_block+blocks_needed) > BLOCKS_PER_SEG)
          end = 1;
          else
          {
           while (!found && ((curr_block+blocks_needed) <= BLOCKS_PER_SEG))
            {
             if (!In_Log_Rec_List(curr_block,
                                  curr_block+blocks_needed,
                                  &conflict))
               found = 1;
               else curr_block = conflict+1;
            }

           if (!found)
             end = 1;
      }
  }


 if (msg_num == (MQEMT->rst_ptr->msgs_per_seg-1))
  {
   if (moved)
     curr_block = 0;

   while (!found && ((curr_block+blocks_needed) <= *msg_block))
    {
     if (!In_Log_Rec_List(curr_block,
                          curr_block+blocks_needed,
                          &conflict))
       found = 1;
       else curr_block = conflict+1;
    }

   if (!found)
     curr_block = *msg_block+MSG_ENTRY_BLOCKS;

   while (!found && ((curr_block+blocks_needed) <= BLOCKS_PER_SEG))
    {
     if (!In_Log_Rec_List(curr_block,
                          curr_block+blocks_needed,
                          &conflict))
       found = 1;
       else curr_block = conflict+1;
    }

   if (!found)
     end = 1;
  }


 if (msg_num < (MQEMT->rst_ptr->msgs_per_seg-1))
  {
   if (moved)
     curr_block = 0;

   msg_block2 = msg_block;
   msg_block2++;

   while (!found && (msg_num < (MQEMT->rst_ptr->msgs_per_seg-1)))
    {
     if (moved && start)
      {
       while (!found && ((curr_block+blocks_needed) <= *msg_block))
        {
         if (!In_Log_Rec_List(curr_block,
                              curr_block+blocks_needed,
                              &conflict))
           found = 1;
           else curr_block = conflict+1;
        }

       start = 0;
      }

     if (!found)
```

```
      curr_block = *msg_block+MSG_ENTRY_BLOCKS;

    while (!found && ((curr_block+blocks_needed) <= *msg_block2))
     {
      if (!In_Log_Rec_List(curr_block,
                           curr_block+blocks_needed,
                           &conflict))
        found = 1;
        else curr_block = conflict+1;
     }

    if (!found)
     {
      msg_num++;
      msg_block++;

      if (msg_num < (MQEMT->rst_ptr->msgs_per_seg-1))
       {
        msg_block2 = msg_block;
        msg_block2++;
       }
       else
       {
        curr_block = *msg_block+MSG_ENTRY_BLOCKS;

        while (!found && ((curr_block+blocks_needed) <= BLOCKS_PER_SEG))
         {
          if (!In_Log_Rec_List(curr_block,
                               curr_block+blocks_needed,
                               &conflict))
            found = 1;
            else curr_block = conflict+1;
         }
       }
     }
   }        // end while

  if (!found)
    end = 1;
 }


 if (end)
  {
   moved = 1;
   start = 1;

   if (cycled)
     notfull = 0;
     else curr_seg = (curr_seg+1)%MQEMT->num_segs;

   if (!cycled)
    {
     if (curr_seg == orig_seg)
       cycled = 1;

     // Flush out QEMT and zero out
     // active log record list.

     MQEMT->next_avail_block.segment =
              (MQEMT->next_avail_block.segment+1)%MQEMT->num_segs;

          curr_block = 0;
          MQEMT->next_avail_block.block = 0;

          status = Gen_QEM_Seq_No(&(MQEMT->qem_sn));

          status = Write_QEMT(Que_File_Handle,
                              MQEMT->next_avail_block.segment,
                              MQEMT);

          Init_Active_LREC_List();

          status = Cycle_RST_Seg(curr_seg,&lprstseg);

          msg_num = 0;
          msg_block = lprstseg->msg_block;
          msg_block2 = msg_block;
          msg_block2++;
          end = 0;
         }
      }      // end if (end)

    }        // while (notfull && !found)

   if (found)
    {
/*
    Diag("Allocated Space: segment = %d, block = %d", curr_seg, curr_block);
*/

    offset->segment = MQEMT->next_avail_block.segment;
    offset->block = curr_block;
    MQEMT->next_avail_block.block = curr_block + blocks_needed;
    holey_entries = 0;

    if (type == MSG_WRITE)
     {
      if (!Add_RST_Entry(offset->segment,offset->block))
        notfull = 0;
        else Sort_RST_Entries(offset->segment);
     }
     else if (type == LOG_WRITE)
     {
      if (!Add_Log_Entry(offset->block))
        notfull = 0;
        else Sort_Log_Entries(offset->segment);
     }
    }
    else
    {
     holey_entries = holes;
     offset->segment = NIL;
     offset->block = NIL;
    }

   return(notfull);
 }
```

```
BOOL Check_Queue_Full(
   lpQEMT  lpqemt)
{
   lpQEME   lpqeme;
   BOOL     Full=TRUE;
   int      status;
   int      cnt;

   if (lpqemt->que_hd_ptr == NIL)
     Full = FALSE;
     else if (((lpqemt->que_tl_ptr+1)%lpqemt->max_entries)
                 == lpqemt->que_hd_ptr)
     {
      status = Cycle_QEME(lpqemt->que_hd_ptr,&lpqeme);
      if (lpqeme->txn_state == EMPTY)
        Full = FALSE;
     }
     else
     {
      lpqeme = lpqemt->qeme_ptr;
      cnt = 0;
      while (Full && (cnt < lpqemt->max_entries))
        {
        if (lpqeme->txn_state == EMPTY)
          Full = FALSE;
          else
          {
           cnt++;
           lpqeme++;
          }
        }
     }

   return(Full);
}


BOOL Check_Queue_Empty(
   lpQEMT  lpqemt)
{
 if (lpqemt->que_hd_ptr == NIL)
   return(TRUE);
   else return(FALSE);
}


void Conv_from_MTlist()
{
   lpPTLIST p1,p2;
   lpTLIST  t1;
   lpMTLIST m1;
   int      cnt = 0;

   p1 = (PTLIST *)malloc(MQEMT->max_entries*sizeof(PTLIST));
   ml = Pending_TXNs;
   p2 = pl;

   while (ml != NULL)
    {
     tl = ml->ops;
     while (tl != NULL)
      {
       p2->mid = ml->mid;
       p2->qeme_no = tl->qeme_no;
       cnt++;
       p2++;
       tl = tl->next;
      }
     ml = ml->next;
    }

   MQEMT->ptl_ptr = pl;
   MQEMT->num_pts = cnt;
}


void Conv_to_MTlist()
{
   lpPTLIST pl;
   lpMTLIST ml;
   lpTLIST  tl;
   MID      tmid;
   int      cnt;

   Pending_TXNs = NULL;
   pl  = MQEMT->ptl_ptr;

   cnt = 1;
   while (cnt <= MQEMT->num_pts)
    {
     tmid = pl->mid;
     Add_MTlist(&Pending_TXNs,&ml,pl->mid);
     tl = ml->ops;

     while ((cnt <= MQEMT->num_pts) &&
            (pl->mid.host == tmid.host) &&
            (pl->mid.tid == tmid.tid))
      {
       Add_Tlist(&tl,pl->qeme_no);
       pl++;
       cnt++;
      }
     ml->ops = tl;
    }

   Pending_TXNs = ml;
}


void Add_Tlist(
   lpTLIST *head,
   int      qeme_no)
{
   lpTLIST curr,prev;
   lpTLIST   new;

   new = (TLIST *)malloc(sizeof(TLIST));

   new->qeme_no = qeme_no;
   new->next = NULL;

   curr = *head;
   prev = curr;

   while (curr != NULL)
    {
     prev = curr;
     curr = curr->next;
    }

   if (prev == NULL)
     *head = new;
     else prev->next = new;
}


void Add_MTlist(
   lpMTLIST *head,
   lpMTLIST *tail,
   MID       mid)
{
   lpMTLIST curr,prev;
   lpMTLIST   new;

   new = (MTLIST *)malloc(sizeof(MTLIST));

   new->mid  = mid;
   new->ops  = NULL;
   new->next = NULL;
   *tail = new;

   curr = *head;
   prev = curr;

   while (curr != NULL)
    {
     prev = curr;
     curr = curr->next;
    }

   if (prev == NULL)
     *head = new;
     else prev->next = new;
}


BOOL Find_MTlist(
   lpMTLIST  head,
   lpMTLIST *pres,
   MID       mid)
{
   lpMTLIST curr;
   int      found = 0;

   curr = head;
   while ((curr != NULL) && !found)
    {
     if ((curr->mid.host == mid.host) &&
         (curr->mid.tid == mid.tid))
       found = 1;
       else curr = curr->next;
    }

   if (found)
    {
     *pres = curr;
     return(TRUE);
    }
    else
    {
     *pres = NULL;
     return(FALSE);
    }
}


BOOL Find_Tlist(
   lpTLIST  head,
   int      qeme_no)
{
   lpTLIST curr;
   int      found = 0;

   curr = head;
   while ((curr != NULL) && !found)
    {
     if (curr->qeme_no == qeme_no)
       found = 1;
       else curr = curr->next;
    }

   if (found)
     return(TRUE);
     else return(FALSE);
}



void Del_MTlist_All(
   lpMTLIST  *head)
{
   lpMTLIST curr,prev;
   BOOL     Return_Status;

   curr = *head;
   *head = NULL;
   while (curr != NULL)
    {
     prev = curr;
     curr = curr->next;
     prev->next = NULL;

     Del_Tlist(&prev->ops);
     free(prev);
     if (Return_Status == FALSE)
       Diag("Del_MTlist_All: Problem with freeing memory.");
    }
}



BOOL Del_MTlist(
   lpMTLIST  *head,
   MID        mid)
{
   lpMTLIST curr,prev;
   int      found = 0;
   BOOL     Return_Status;

   curr = *head;
   prev = curr;

   while ((curr != NULL) && !found)
    {
     if ((curr->mid.host == mid.host) &&
         (curr->mid.tid == mid.tid))
       found = 1;
       else
       {
        prev = curr;
        curr = curr->next;
       }
    }

   if (found)
    {
     Del_Tlist(&curr->ops);

     if (curr == prev)
       *head = curr->next;
       else if (curr->next == NULL)
         prev->next = NULL;
         else prev->next = curr->next;

     curr->next = NULL;

     free(curr);
     if (Return_Status == FALSE)
       Diag("Del_MTlist: Problem with freeing memory...");

     return(TRUE);
    }
     else return(FALSE);
}
```

```
void Del_Tlist(
   lpTLIST  *head)
{
   lpTLIST curr,prev;
   BOOL     Return_Status;

   curr = *head;
   while (curr != NULL)
    {
     prev = curr;
     curr = curr->next;
     prev->next = NULL;

     free(prev);
     if (Return_Status == FALSE)
       Diag("Del_Tlist: Problem with freeing memory...");
    }

   *head = NULL;
}


void Sub_Addr (
   LONG    *diff,
   BLAD    *x,
   BLAD    *y)
{
  *diff = (LONG) (((y->segment-x->segment)*SEGMENT_SIZE)+
                   QEMT_Size+(y->block-x->block)*BLOCK);
}


void Conv_Addr (
   LONG    *diff,
   BLAD    *x)
{
  *diff = (LONG) ( (x->segment*SEGMENT_SIZE)+QEMT_Size+(x->block*BLOCK));
}


int Gen_QEM_Seq_No(
   lpSN   tSeq_No)
{
  long     temp;

  // Generate unique QEMT sequence
  // number based on system time
  // and unique counter.

  temp = time(&tSeq_No->timestamp);
  tSeq_No->counter = QEMT_Seq_No++;

  return(QSUCCESS);
}
```

```
int Gen_LREC_Seq_No(
   lpSN   tSeq_No)
{
  long     temp;

  // Generate unique LREC sequence
  // number based on date and time.

  temp = time(&tSeq_No->timestamp);
  tSeq_No->counter = LREC_Seq_No++;


  return(QSUCCESS);
}


int Gen_QEME_TS(
   unsigned long *tSeq_No)
{
  // Generate unique QEMT sequence
  // number based on date and time.

  *tSeq_No = QEME_TS++;

  return(QSUCCESS);
}


BOOL Bigger_Seq_No(
   lpSN     seq_no1,
   lpSN     seq_no2)
{
   if ((seq_no1->timestamp > seq_no2->timestamp) ||
       ((seq_no1->timestamp == seq_no2->timestamp) &&
        (seq_no1->counter > seq_no2->counter)))
     return (TRUE);
     else return(FALSE);
}
```

```
/*
**
** Copyright(C)1995 MITSUBISHI ELECTRIC ITA.  ALL RIGHTS RESERVED.
** UNPUBLISHED -- RIGHTS RESERVED UNDER THE COPYRIGHT
** LAWS OF THE UNITED STATES.  USE OF A COPYRIGHT NOTICE
** IS PRECAUTIONARY ONLY AND DOES NOT IMPLY PUBLICATION
** OR DISCLOSURE.
**
** THIS SOFTWARE CONTAINS CONFIDENTIAL INFORMATION AND
** TRADE SECRETS OF MITSUBISHI ELECTRIC ITA.  USE, DISCLOSURE,
** OR REPRODUCTION IS PROHIBITED WITHOUT THE PRIOR
** EXPRESS WRITTEN PERMISSION OF MITSUBISHI ELECTRIC ITA.
**
** OpenMQ
**   Module: qnetd.c
**   Author: Derek Schwenke 9/8/95
**
*/

#include "qlib.h"
#include "qnetd.h"

// This thread listens on each Buffer that QNETD owns.
// Resurve a buffer
// Wait For READY
// Read the message

void QnetdAdminRequest(lpSMBUF buf) {
   char *p = buf->mdata; int i;

   switch(buf->msgh.sub_mode){
   case QNETD_TRAN_INQ:
      buf->msgh.sub_mode = ResolveTran(&buf->msgh.mid);
   break;

   case QNETD_RT_BROADCAST:
      Warn("QnetdAdminRequest: RT Broadcast not implemented");
   break;

   case QNETD_BUF_STATUS:
      sprintf(p,"QNETD status for %p\n",SHAREDATA(hostip)); p = p + strlen(p);
      for(i=0;i<SHAREDATA(nsbuf);i++) {
         lpSMBUF b = SMBUFADDR(i);
         sprintf(p,"%d: %d %s\n",i,b->status,b->name); p = p + strlen(p);
      }
   break;

   default: Warn("QnetdAdminRequest: undefined sub_type of request (%d)",
            buf->msgh.sub_mode);
   }

}


DWORD ListenSMBuff() {
   lpSMBUF  buf;
   int      stat,pass,bnum,rtn;
   SMOBJS   sync;
#define MAXPASS 1  /* 1 Attempt to send on the socket */
```

```
   Sleep(1000);  // Wait for others to init, no real reason

// Reserve a buffer
   Diag("ListenSMBuff[-]: Calling ReserveSharedBuffer");
   bnum = ReserveSharedBuffer(0,"QNETD",SMBUF_EMPTY,&sync);
   buf = SMBUFADDR(bnum);


   while(1) {
      stat = 0;
// Wait for a message
      Diag("ListenSMBuff[%d]: Waiting for buffer ready",bnum);
      if ((rtn = WaitForSingleObject(sync.readyh,INFINITE)) == WAIT_FAILED )
         Fail("ListenSMBuff[%d]: Failed while waiting for the buffer ready %d #%

// Sanity check the buffer
      if (buf->status != SMBUF_SEND_MESS) {
         Warn("ListenSMBuff[%d]: Buffer status was wrong! %d",bnum,buf->status);
      }

// Read the message
      Diag("ListenSMBuff[%d]: got:%s", bnum,buf->mdata);

      pass = 0;
      if (SHAREDATA(hostip) != buf->msgh.to_node){ // Not local
         while(pass++ < MAXPASS ) { // Make N trys to send the message
            if (SendRemoteMess(buf,0)) {
               if (buf->msgh.mode != ADMINREQ_MODE) // Admin req's may fail
               Warn("ListenSMBuff[%d]: SendRemoteMess failed",bnum);
            } else {
               // send was ok now wait for the ACK/NACK
               if ((rtn = WaitForSingleObject(sync.ackh,22000)) == WAIT_OBJECT_0
                  if (buf->msgh.mode == NACK_MODE) pass = 100;

                  Diag("ListenSMBuff[%d]: got an ack/nack",bnum);
                  break;
               } else if (rtn == WAIT_TIMEOUT) {
                  Warn("ListenSMBuff[%d]: No ack Timed out after 22 sec",bnum,rt
               } else {
                  Warn("ListenSMBuff[%d]: No ack WaitForSingleObject=%d",bnum,rt
               }
            }
         }
      } else { // Local message for QNETD
         Say("ListenSMBuff[%d]: GOT LOCAL MESSAGE",bnum);
         if (buf->msgh.mode == ADMINREQ_MODE){
            QnetdAdminRequest(buf);
         } else {
            Warn("ListenSMBuff[%d]: Qnetd drops non-admin request",bnum);
            pass = MAXPASS + 2;
         }
      }

      if (pass < MAXPASS + 1) buf->status = SMBUF_RETURN_MESS;
      else                    buf->status = SMBUF_RETURN_FAIL;


// Notify the sender  "Your message was read"
      Diag("ListenSMBuff[%d]: releases DONE status = %d",bnum,buf->status);
      if (!ReleaseSemaphore(sync.doneh,1,0))
         Warn("ListenSMBuff[%d]: ReleaseSemaphore done #%d",bnum,GetLastError())
   }  //loop
   Say("ListenSMBuff[%d] Aborts",bnum);
   return(0);
}
```

```
/////////////////////////////////////////////////////////////////////////////
//
// Common routines and definitions used by OpenMQ
//
///////////////////////////////////////////////////////////////////////////
#include    "qlib.h"
#include    "qnetd.h"
// Routines /////////////////////////////////////////////////////////////////


void ReadParms(int *nsbuf) {  // *nsbuf is Optional
   char line[LINESIZE];
   char parm[LINESIZE];
   char value[LINESIZE];
   FILE *fp = fopen(PARMNAME,"r");


   if (sm_base) { // If we have shared memory, then set defaults.
      SHAREDATA(diag)       = 1;
      SHAREDATA(time_out)   = 20000;
   }

   if ( ! fp )
          Fail("Cant open the parameters file %s",PARMNAME);

   while (fgets(line,LINESIZE,fp)) {
      *parm = *value = 0;
      if ( 2 != sscanf(line,"%s = %s",parm,value)) {
             *parm = 0;

      } else if (!strcmp(parm,"nsbuf") ) {
         if (nsbuf){
            sscanf(value,"%d",nsbuf);
                   if (*nsbuf > MAXNSMBUF){
                      Warn("parameter: nsbuf %d exceeds %s maximum",*nsbuf,MAXNS
               *nsbuf = MAXNSMBUF;
            }
         }
      } else if (sm_base && !strcmp(parm,"diag")) {
         sscanf(value,"%hd",&SHAREDATA(diag));

      } else if (sm_base && !strcmp(parm,"time_out")) {
         sscanf(value,"%d",&SHAREDATA(time_out));

      } else if (sm_base) {
         Say("ReadParms: Ignoring: %s",line);
      }
   }
   if (ferror(fp)){
      Fail("read error in parameters file %s",PARMNAME);
      clearerr(fp);
   }
   fclose(fp);
}
```

```
/*
**
** Copyright(C)1995 MITSUBISHI ELECTRIC ITA.  ALL RIGHTS RESERVED.
** UNPUBLISHED -- RIGHTS RESERVED UNDER THE COPYRIGHT
** LAWS OF THE UNITED STATES.  USE OF A COPYRIGHT NOTICE
** IS PRECAUTIONARY ONLY AND DOES NOT IMPLY PUBLICATION
** OR DISCLOSURE.
**
** THIS SOFTWARE CONTAINS CONFIDENTIAL INFORMATION AND
** TRADE SECRETS OF MITSUBISHI ELECTRIC ITA.  USE, DISCLOSURE,
** OR REPRODUCTION IS PROHIBITED WITHOUT THE PRIOR
** EXPRESS WRITTEN PERMISSION OF MITSUBISHI ELECTRIC ITA.
**
** OpenMQ
**   Module: qnetd.c
**   Author: Derek Schwenke 9/8/95
**
*/

#include "qlib.h"
#include "qnetd.h"
#include "rt.h"

main(int argc, char **argv) {
   HANDLE   hSMT, hTCPT, hDOG;
   DWORD    idSMT,idTCPT, idDOG;
   WSADATA  WSAData;
   int      nsbuf = 0;

// DLL Init
   if (WSAStartup(0x0101, &WSAData))
      Fail("WSAStartup() failed");

// Read only nsbuf.
   ReadParms(&nsbuf);

   SharedMemInit(nsbuf);

// Read and set all other shared memory parameters
   ReadParms(0);

// Load routing table into shared memory
   ReadRT();


// Create Threads - listen for connections from other qnetd's //AfxBeginThread??
   hSMT  = CreateThread(NULL,0,(LPTHREAD_START_ROUTINE) ListenConn,  0,0,&idTCPT
// Create Threads - listen for shared memory buffer requests
   hTCPT = CreateThread(NULL,0,(LPTHREAD_START_ROUTINE) ListenSMBuff,0,0,&idSMT)
// Create Background watchdog thread
   hDOG = CreateThread(NULL,0,(LPTHREAD_START_ROUTINE) WatchDog,0,0,&idDOG);



   Sleep(2000); Ask("QNETD All Done?");
   return(0);

}
```

```
/*
**
** Copyright(C)1995 MITSUBISHI ELECTRIC ITA.  ALL RIGHTS RESERVED.
** UNPUBLISHED -- RIGHTS RESERVED UNDER THE COPYRIGHT
** LAWS OF THE UNITED STATES.  USE OF A COPYRIGHT NOTICE
** IS PRECAUTIONARY ONLY AND DOES NOT IMPLY PUBLICATION
** OR DISCLOSURE.
**
** THIS SOFTWARE CONTAINS CONFIDENTIAL INFORMATION AND
** TRADE SECRETS OF MITSUBISHI ELECTRIC ITA.  USE, DISCLOSURE,
** OR REPRODUCTION IS PROHIBITED WITHOUT THE PRIOR
** EXPRESS WRITTEN PERMISSION OF MITSUBISHI ELECTRIC ITA.
**
** OpenMQ
**   Module: routab.c
**   Author: Derek Schwenke 9/8/95
**
** Routing table funtions
*/


#include "qlib.h"
#include "qnetd.h"
#include "rt.h"

#include <string.h>
#include <ctype.h>



  /*
int Str2Port(char *str){
   int port=0;
   if (sscanf(str,"%*[^\[][%d",&port))
      Say("Str2Port %s returns %d",str,port);
   else
      Say("Str2Port %s FAILED",str);
   return(port);
}
   */

// Routing table entry,,,, to be deleted
typedef struct rte {          //
   int   ip;                  //
   char  node[NAMESIZE];      //
   char  ntype[NAMESIZE];     //
   char  cs[NAMESIZE];        //
   int   qnetd_port;          //
   char  serv_class[NAMESIZE];//
   char  servers[4*NAMESIZE]; //
   char  physical[NAMESIZE];  //
   char  logicals[4*NAMESIZE];//
   char  serv_path[NAMESIZE]; //
   char  serv_opts[NAMESIZE]; //
} RTE, *lpRTE;



static int is_ip(char *s) {
   return( isdigit(s[0]) );
}

static int is_stype(char *s) {
   return( strlen(s) > 3 && strstr(&CLASSNAMES,s) );
}

// The table looks like this:
//137.203.1.1   mars            NT3.5           Server[100]
// class0               router          c:/q/route      -master
// class1               QS1[Q1,Q2,Q3]   c:/q/qserv      -threads 66 -more_opts
//
// The node line is:
// IP# node_name node_type Client_or_server[QNETD address] Preferred_servers,,,
//
// The Services lines is:
// type physical_name[logical_names,,,] path_to_executable options

int parseRTnode(  char  *line,
                  int   line_no,
                  int   *ip,
                  char  *node,
                  char  *ntype,
                  char  *cs,
                  int   *port,
                  char  *servers) {
   char ip_str[NAMESIZE];
   char cs_str[LINESIZE];

   strcpy(servers,",");
   if ( 4 > sscanf(line,"%s %s %s %s %s",ip_str,node,ntype,cs_str,&servers[1]
      Fail("Routing Table Line %d is not complete: %s",line_no,line);

   if (strlen(servers) > 1)
      strcat(servers,",");
   else
      strcpy(servers,"");

   if (!(*ip = Str2IP(ip_str)))
      Fail("Routing Table Line %d ip does not scan: %s",line_no,line);

   if (2 != sscanf(cs_str,"%[^\[][%d]",cs,port))
      Fail("Routing Table Line %d QNETD client/server does not scan: %s",line_no

   return(1);
}

int parseRTservices(char  *line,
                  int   line_no,
                  char  *clas,
                  char  *physical_name,
                  char  *logical_names,
                  char  *path,
                  char  *opts) {
   char services[LINESIZE];
   int len;
   strcpy(opts,"");
   if ( 3 > sscanf(line,"%s %s %s %[^]",clas,services,path,opts))
      Fail("Routing Table: line %d is not complete: %s",line_no,line);
   strcpy(logical_names,",");
   sscanf(services,"%[^\[][%s]",physical_name,&logical_names[1]);
   if((len = strlen(logical_names)) < 2) // No logical name
      sprintf(logical_names,",%s,",physical_name);
   else
      logical_names[--len] = ',';

   return(1);
}

/*
**
**char * NewStr(char * s) {
**   char * n = malloc(strlen(s)+1);
**   if (!n) Fail("Malloc error in NewStr");
**   return(strcpy(n,s));
**}
**
**
**void FreeNcopy(char ** o, char * s) {
**   char * n = malloc(strlen(s)+1);
**   if (!n) Fail("Malloc error in FreeNcopy");
**   strcpy(n,s);
**   free(*o);
**   *o = n;
**}
**
*/


void SwapRT(lpRT RT1, lpRT RT2) {
   lpRT P,R,PRT1=0,PRT2=0;

   Diag("SwapRT(%s,%s)",RT_NODE(RT1),RT_NODE(RT2));

   if (RT1 == RT2) return;

   // Scan for the pre1 and pre2 nodes
   P = RTROOT;
   while (R = NextRT(P)) { // Find both the previous nodes
      if (R == RT1) PRT1 = P;
      if (R == RT2) PRT2 = P;
      P = R;
   }
   if (!(PRT1 && PRT2)) return;
   // Swap pointers
   PRT1->next_offset = RT2C(RT2) - RT2C(PRT1);
   if (RT2->next_offset) PRT2->next_offset = RT2C(RT_NEXT(RT2)) - RT2C(PRT2);
   else                  PRT2->next_offset = 0;
   RT2->next_offset = RT2C(RT1) - RT2C(RT2);

}

int fgetsh(char *line,int max,HANDLE rth) {  // Aproximate fgets()
                                             // Read 1 char at a time until \n
   int rd=1,rdn=0;
   while ((rdn < max-1) && ReadFile(rth,line,1,&rd,0)) {
      if (rd != 1) break;  // Did not get one byte
      if (*line == '\n') break;
      line++; rdn++;
   }
   if (*line = '\n') line--;
   if (rd) line++;
   *line = 0;
   return(rdn);
}


HANDLE MakeRoutingTable() {    // Only be called by QNETD.
   int    i,line_no = 0;
   PIN_ADDR pia;
   char   line[LINESIZE];


   HANDLE rth = CreateFile( ROUTNAME ,
                GENERIC_WRITE,           // |GENERIC_READ
                FILE_SHARE_WRITE, 0,     // |FILE_SHARE_READ
                OPEN_ALWAYS,
                FILE_ATTRIBUTE_NORMAL, 0);


   sprintf(line,"RT_VERSION 1\r\n");
   WriteFile(rth,line,lstrlen(line), &i,0);

   sprintf(line,"#### Servers\r\n");
   WriteFile(rth,line,lstrlen(line), &i,0);


   i =   htonl( (u_long) SHAREDATA(hostip) );  // Back to network format
   pia = (PIN_ADDR) (&i);
   sprintf(line,"%s   %-16s Win          Server[4011]\r\n", inet_ntoa(*pia
   WriteFile(rth,line,lstrlen(line), &i,0);

   sprintf(line,"# class1        QS1[QS1]          c:\\q\\bin\\ramq   -options\r\
   WriteFile(rth,line,strlen(line), &i,0);

   sprintf(line,"#### Clients\r\n");
   WriteFile(rth,line,lstrlen(line), &i,0);

   CloseHandle(rth);

   rth = CreateFile( ROUTNAME ,
         GENERIC_READ,
         FILE_SHARE_READ, 0,
         OPEN_EXISTING,
         FILE_ATTRIBUTE_NORMAL, 0);

   if (rth == INVALID_HANDLE_VALUE )
      Fail("Cant create the default routing table file %s",&ROUTNAME);

   return(rth);
}



void ReadRT() {   // Only be called by QNETD.
   int    line_no = 0;
   char   line[LINESIZE];
   char   first[LINESIZE];
char  preferred[LINESIZE];

// must be read|write so that on second open we dont fail.
HANDLE rth = CreateFile( ROUTNAME ,
             GENERIC_WRITE|GENERIC_READ,    //      not
             FILE_SHARE_WRITE|FILE_SHARE_READ, 0, // not
             OPEN_EXISTING, //      not OPEN_ALWAYS
             FILE_ATTRIBUTE_NORMAL, 0);
RTE   rte;   // Current RTE
lpRT  rt,rts;
char *cp;

SHAREDATA(rt_ver) = 0;

if (SHAREDATA(rt_pingpong)) // Not the active RT
   rts = rt = (lpRT) sm_base->RT[0];
else
   rts = rt = (lpRT) sm_base->RT[1];


memset(rt,0,MAXRTSIZE);                 // Clear routing table (not required)
cp = rt->s + 1;                         // 1st node is always nop
rt->ntype_index = cp++ - (char *) rt->s; // 1st node is always nop
rt->apps_index  = cp++ - (char *) rt->s; // 1st node is always nop

strcpy(preferred,"");

if (rth == INVALID_HANDLE_VALUE ) {      // No routing table existed so makeon
   rth = MakeRoutingTable();
}

while (fgetsh(line,LINESIZE-1,rth)) {
   line_no++;
   sscanf(line,"%s",first);
   if (*first == '#') {
      line_no = line_no;
   } else if (is_ip(first)) {     // New node //
      // Perhaps we can filter other clinets out here to save space //
      parseRTnode(line,line_no,&rte.ip,rte.node,
                  rte.ntype,rte.cs,
                  &rte.qnetd_port,rte.servers);

      // Start a new node in the RT
      RT_NOFFSET(rt) = ++cp - RT2C(rt);
      rt = NextRT(rt);
      // Build RT in shared memory

      RT_IP(rt) = rte.ip;               // IP number
      RT_PORT(rt) = rte.qnetd_port;     // Port number
      RT_NOFFSET(rt) = 0;               // End of the chain
      strcpy(RT_NODE(rt),rte.node);     // Node string
      rt->ntype_index = strlen(RT_NODE(rt)) + 1;
      strcpy(RT_NTYPE(rt),rte.ntype);
      rt->apps_index = strlen(RT_NTYPE(rt)) + 1 + rt->ntype_index;
      cp = RT_APPS(rt);
      *cp = 0;


      if (!strcmp(SHAREDATA(hostname),RT_NODE(rt))) {
         strcpy(preferred,rte.servers);
```

```
            SHAREDATA(hostip) = RT_IP(rt);
            Diag("ParseRT this node is %s ip=%p",SHAREDATA(hostname),SHAREDATA(h
         }

      } else if (is_stype(first)) { // New service //
         parseRTservices(line,line_no,rte.serv_class,
                         rte.physical,rte.logicals,
                         rte.serv_path,rte.serv_opts);
      // format: [QS1],Q1,Q2,[QS2],Q4,
         sprintf(cp,"[%s]%s",rte.physical,rte.logicals);
         cp = strchr(cp,0);               // Advance the pointer

      } else if (!strcmp(first,"RT_VERSION")) {
         sscanf(line,"%*s %d",&SHAREDATA(rt_ver));
      }
   }
//   if (ferror(rth)){
//      Fail("ParseRT read error on %s.\n",&ROUTNAME);
//      clearerr(rth);
//   }
   CloseHandle(rth);

   // Sort this list according to the preferred servers
   PrintRT(rts,"Before sorting");

   if (strlen(preferred) > 2) {
      lpRT rtf,rtp = NextRT(rts);
      char *e,*p = preferred + 1;
      Diag("ReadRT: Sorting preferred servers:%s",preferred);
      // Swap entries to achive the right order
      while (rtp && ((strlen(p) > 2))) {
         if (e = strchr(p,',')) *e = 0;
         rtf = rts;
         while (rtf = NextRT(rtf)) if (!strcmp(RT_NODE(rtf),p)) break;
         if (!rtf) {
            Say("ReadRT: Cant find preferred server %s",p);
            break;
         }
         SwapRT(rtp,rtf);
         rtp = NextRT(rtp);

         if (e)   p = e + 1;
         else     break;

      }
      PrintRT(rts,"After sorting");
   }

   // Ping Pong the RT tables, make this one active
   if (SHAREDATA(rt_pingpong))
      SHAREDATA(rt_pingpong) = 0;
   else
      SHAREDATA(rt_pingpong) = 1;
   SHAREDATA(rt_rev)++;  // Every handle will re-load from new RT.

   // We must find some way to set the IP addr, and the RT is the only way to
   if(!SHAREDATA(hostip))
      Warn("ReadRT: could not find host %s in the file %s",SHAREDATA(hostname),&
}
```

```
//{{NO_DEPENDENCIES}}
// Microsoft Visual C++ generated include file.
// Used by qnetd.rc
//
#define ICON1                           2

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE        102
#define _APS_NEXT_COMMAND_VALUE         40001
#define _APS_NEXT_CONTROL_VALUE         1000
#define _APS_NEXT_SYMED_VALUE           101
#endif
#endif
```

```
/*
**
** Copyright(C)1995 MITSUBISHI ELECTRIC ITA.  ALL RIGHTS RESERVED.
** UNPUBLISHED -- RIGHTS RESERVED UNDER THE COPYRIGHT
** LAWS OF THE UNITED STATES.  USE OF A COPYRIGHT NOTICE
** IS PRECAUTIONARY ONLY AND DOES NOT IMPLY PUBLICATION
** OR DISCLOSURE.
**
** THIS SOFTWARE CONTAINS CONFIDENTIAL INFORMATION AND
** TRADE SECRETS OF MITSUBISHI ELECTRIC ITA.  USE, DISCLOSURE,
** OR REPRODUCTION IS PROHIBITED WITHOUT THE PRIOR
** EXPRESS WRITTEN PERMISSION OF MITSUBISHI ELECTRIC ITA.
**
** OpenMQ
**   Module: shm_init.c
**   Author: Derek Schwenke 9/8/95
**
*/

#include "qlib.h"

// Global data - for qnetd only
// QNETD keeps an open handle for each sync object so that they are
// never deleted by the OS, until QNETD exits, then all bets are off.
// QNETD will not call closeHandle() so they will stay active.

// static int sharmeminit_done = 0;  // sharmeminit done flag.

static HANDLE  SMT;                 // Handle for access to SM table.
       SMOBJS  SYNC[MAXNSMBUF];     // Handles for all sync objects

// ShareMemInit should be called once only once by QNETD.

LPVOID SharedMemInit(int nsbuf) {
   HANDLE    hMap;       // Shared memory mapping
   lpSMBUFH  lpBUFH;     // Pointer to base of shared Memory
   PHOSTENT  lpent;
   int i,sm_size,CFM_error;
   char * c;


   //Say("SharedMemInit: starts %d buffers.",nsbuf);

   if ( 0 ) {  // must think of a better test... see if shared mem aleady exists
   // a lock would just say if another qnet had the shared mem. not anything abo
      Warn("ShareMemInit was already done!!!!");
   } else {
          if (!nsbuf) Fail("nsbuf can not be '0' ");


      // Make Qopen() lock
      if (!CreateMutex( NULL, FALSE, MUTQOPEN ))
         Fail("ShareMemInit cant create mutex %s",&MUTQOPEN);

      // Make shared memory update lock
      if(!(SMT = CreateMutex(NULL, TRUE, SMEMTABLE)))
         Fail("ShareMemInit cant create mutex %s",&SMEMTABLE);

        // Calculate the size of shared memory
        sm_size = MAXSMSIZE(nsbuf);
```

```
    //Say("Shared memory will be %d bytes\n",sm_size);

    // Create the shared memory
hMap = CreateFileMapping(
    (HANDLE)0xFFFFFFFF, // Do not use a real disk file
    NULL,               // No security
    PAGE_READWRITE,     // Page protection
    0, sm_size,         // File size
    SMEMNAME);          // Name

CFM_error = GetLastError();

if ( ! hMap )
   Fail("CreateFileMapping returned null");

   // Map the shared memory into my address space
lpBUFH = sm_base = MapViewOfFile(
      hMap,            // object HANDLE
      FILE_MAP_WRITE,  // Access
      0,0,             // Address into the file
      0 );             // Size (Full file)

if ( ! sm_base )
   Fail("MapViewOfFile returned null");


if (CFM_error == ERROR_ALREADY_EXISTS) {
   Fail("Shared memory already in use.");

           // Code should also check clcok to see if QNET is running

           ForcedBufferReset(i,0,0);
   for (i = 0 ; i < nsbuf ; i++ ) {
                   ForcedBufferReset(i,0,0);
   }
} else {
// Initialize the buffers data
   for (i = 0 ; i < nsbuf ; i++ ) {
      lpSMBUF b = SMBUFADDR(i);
      strcpy(b->name,"empty");
      b->status = SMBUF_EMPTY;
      b->sub_status = 0;
   }
}

// Create all syncronization objects free,ready,done.
for (i = 0 ; i < nsbuf ; i++ ) {
   char freename[NAMESIZE], readyname[NAMESIZE], donename[NAMESIZE], ackna
   sprintf(freename,MUTFREFMT,i);
   sprintf(readyname,SEMRDYFMT,i);
   sprintf(donename,SEMDONFMT,i);
   sprintf(ackname,SEMACKFMT,i); // sync QNETD tcp lister with buffer list
   if ( ! (SYNC[i].freeh  = CreateMutex(NULL,FALSE,freename) ) )
      Fail("Cant create mutex %s",freename);
   if ( ! (SYNC[i].readyh = CreateSemaphore(NULL,0,1,readyname) ) )
      Fail("Cant create semaphore %s",readyname);
   if ( ! (SYNC[i].doneh  = CreateSemaphore(NULL,0,1,donename) ) )
      Fail("Cant create semaphore %s",donename);
   if ( ! (SYNC[i].ackh  = CreateSemaphore(NULL,0,1,ackname) ) )
      Fail("Cant create semaphore %s",ackname);
      }

      // Get this host name
      if (gethostname(SHAREDATA(hostname), sizeof (SHAREDATA(hostname))))
         Warn("ReadParms: gethostname failed WSAGetLastError()=%d",WSAGetLast


      // Get host ip from the hostent table
      if (!(lpent = gethostbyname(SHAREDATA(hostname))) )
         Warn("ReadParms: gethostbyname failed WSAGetLastError()=%d",WSAGetLastE

      i =  *((int*) lpent->h_addr);
      SHAREDATA(hostip) = htonl(i);


      // Get rid of the domain extention "bob.meitca.com"
      if (strchr(SHAREDATA(hostname),'.')) {
         Say("Truncating host name %s",SHAREDATA(hostname));
         *strchr(SHAREDATA(hostname),'.') = 0;
      }

      SHAREDATA(nsbuf) = nsbuf;
      ReleaseMutex(SMT);

      //Say("ShareMemInit returns %d bytes at %p",sm_size,sm_base);
      return(sm_base);
   }
}
```

```
User: root
Host: bunny
Class: bunny
Job: stdin
```

```
/*
**
** Copyright(C)1995 MITSUBISHI ELECTRIC ITA.  ALL RIGHTS RESERVED.
** UNPUBLISHED -- RIGHTS RESERVED UNDER THE COPYRIGHT
** LAWS OF THE UNITED STATES.  USE OF A COPYRIGHT NOTICE
** IS PRECAUTIONARY ONLY AND DOES NOT IMPLY PUBLICATION
** OR DISCLOSURE.
**
** THIS SOFTWARE CONTAINS CONFIDENTIAL INFORMATION AND
** TRADE SECRETS OF MITSUBISHI ELECTRIC ITA.  USE, DISCLOSURE,
** OR REPRODUCTION IS PROHIBITED WITHOUT THE PRIOR
** EXPRESS WRITTEN PERMISSION OF MITSUBISHI ELECTRIC ITA.
**
** OpenMQ
**    Module: tcp.c
**    Author: Derek Schwenke 9/8/95
**
*/

#include "qlib.h"
#include "qnetd.h"
#include "rt.h"
#include "netadmin.h"

extern SMOBJS SYNC[MAXNSMBUF];


int SAY_WSAERROR_TEXT(){
   int rc = WSAGetLastError();
 #define ERROR_BUF_LEN (1000)
   char pszError[ERROR_BUF_LEN];
  wsprintf(pszError, "WinSock error %d: ", rc);
  switch (rc)
  {
    case WSAEINTR:          lstrcat(pszError, "Interrupted system call"); break;
    case WSAEBADF:          lstrcat(pszError, "Bad file number"); break;
    case WSAEACCES:         lstrcat(pszError, "Permission denied"); break;
    case WSAEFAULT:         lstrcat(pszError, "Bad address"); break;
    case WSAEINVAL:         lstrcat(pszError, "Invalid argument"); break;
    case WSAEMFILE:         lstrcat(pszError, "Too many open files"); break;
    case WSAEWOULDBLOCK:    lstrcat(pszError, "Operation would block"); break;
    case WSAEINPROGRESS:    lstrcat(pszError, "Operation now in progress"); break
    case WSAEALREADY:       lstrcat(pszError, "Operation already in progress"); b
    case WSAENOTSOCK:       lstrcat(pszError, "Socket operation on non-socket");
    case WSAEDESTADDRREQ:   lstrcat(pszError, "Destination address required"); br
    case WSAEMSGSIZE:       lstrcat(pszError, "Message too long"); break;
    case WSAEPROTOTYPE:     lstrcat(pszError, "Protocol wrong type for socket");
    case WSAENOPROTOOPT:    lstrcat(pszError, "Protocol not available"); break;
    case WSAEPROTONOSUPPORT: lstrcat(pszError, "Protocol not supported"); break;
    case WSAESOCKTNOSUPPORT: lstrcat(pszError, "Socket type not supported"); bre
    case WSAEOPNOTSUPP:     lstrcat(pszError, "Operation not supported on socket"
    case WSAEPFNOSUPPORT:   lstrcat(pszError, "Protocol family not supported"); b
    case WSAEAFNOSUPPORT:   lstrcat(pszError, "Address family not supported by pr
    case WSAEADDRINUSE:     lstrcat(pszError, "Address already in use"); break;
    case WSAEADDRNOTAVAIL:  lstrcat(pszError, "Can't assign requested address");
    case WSAENETDOWN:       lstrcat(pszError, "Network is down"); break;
    case WSAENETUNREACH:    lstrcat(pszError, "Network is unreachable"); break;
    case WSAENETRESET:      lstrcat(pszError, "Network dropped connection on rese
    case WSAECONNABORTED:   lstrcat(pszError, "Software caused connection abort")
    case WSAECONNRESET:     lstrcat(pszError, "Connection reset by peer"); break;
    case WSAENOBUFS:         lstrcat(pszError, "No buffer space available"); break
    case WSAEISCONN:         lstrcat(pszError, "Socket is already connected"); bre
    case WSAENOTCONN:        lstrcat(pszError, "Socket is not connected"); break;
    case WSAESHUTDOWN:       lstrcat(pszError, "Can't send after socket shutdown")
    case WSAETOOMANYREFS:    lstrcat(pszError, "Too many references: can't spli
    case WSAETIMEDOUT:       lstrcat(pszError, "Connection timed out"); break;
    case WSAECONNREFUSED:    lstrcat(pszError, "Connection refused"); break;
    case WSAELOOP:           lstrcat(pszError, "Too many levels of symbolic links"
    case WSAENAMETOOLONG:    lstrcat(pszError, "File name too long"); break;
    case WSAEHOSTDOWN:       lstrcat(pszError, "Host is down"); break;
    case WSAEHOSTUNREACH:    lstrcat(pszError, "No route to host"); break;
    case WSAENOTEMPTY:       lstrcat(pszError, "Directory not empty"); break;
    case WSAEPROCLIM:        lstrcat(pszError, "Too many processes"); break;
    case WSAEUSERS:          lstrcat(pszError, "Too many users"); break;
    case WSAEDQUOT:          lstrcat(pszError, "Disc quota exceeded"); break;
    case WSAESTALE:          lstrcat(pszError, "Stale NFS file handle"); break;
    case WSAEREMOTE:         lstrcat(pszError, "Too many levels of remote in path"
    case WSASYSNOTREADY:     lstrcat(pszError, "Network sub-system is unusable");
    case WSAVERNOTSUPPORTED: lstrcat(pszError, "WinSock DLL cannot support this
    case WSANOTINITIALISED: lstrcat(pszError, "WinSock not initialized"); break;
    case WSAHOST_NOT_FOUND: lstrcat(pszError, "Host not found"); break;
    case WSATRY_AGAIN:       lstrcat(pszError, "Non-authoritative host not found")
    case WSANO_RECOVERY:     lstrcat(pszError, "Non-recoverable error"); break;
    case WSANO_DATA:         lstrcat(pszError, "Valid name, no data record of requ
#ifdef _WIN32
    case WSAEDISCON:         lstrcat(pszError, "Disconnect"); break;
#endif
    default:                 lstrcpy(pszError, "Unknown WinSock error"); break;
  }
  Say(pszError);
  return(rc);
}


lpST STroot = NULL;

lpST FindSocket (int ip){
   lpST stp = STroot;

   while(stp) {
      if (stp->ip == ip) break;
      stp = stp->next;
   }

   if (stp == NULL)
      Say("FindSocket: ip=%p not found.",ip);
   else
      Diag("FindSocket: ip=%p found.",ip);

   return(stp);
}


int RemoveSocket (SOCKET s){       // This needs to be thread safe
   lpST stp = STroot;              // Needs to kill the thread listening on this s
   lpST stpp = NULL;

   while(stp) {
      if (stp->s == s) break;
      stpp = stp;
      stp = stp->next;
   }

   if (stp) {
      Diag("RemoveSocket: closesocket() ip=%p",stp->ip);
      closesocket(s);

      if (stpp)
         stpp->next = stp->next;
      else
         STroot = stp->next;
      free(stp);
       return(1);
   } else if ((int) s  )
      Warn("RemoveSocket: s=%p NOT FOUND!!",s);
   return(0);
}


DWORD  ListenOnConn(lpST stab){
   SOCKET   s = stab->s;
   int      ip = stab->ip;
   int      port = stab->port;
   SMBUF    buf;  // Buffer where messages will be held
   lpSMBUF  abuf;  // Buffer where ack data will be copied
   int      rc;      // recv code
   int      i,bnum;    // Buffer number to send ack
   QHANDLE  q;   // Psudo handle used to forward messages
   memset(&q,0,sizeof(QHANDLE));

   Diag("ListenOnConn: PARAMETERS socket=%d ip=%s port=%d",s,IP2Name(ip),port


   Diag("ListenOnConn(%s:%d): Waiting for some data",IP2Name(ip),s);

   while((rc = recv(s,(char *) &buf.msgh,MAXTXSIZE,0)) > 0) {

      SHAREDATA(stat.rx)++; // Any communication received statistics



      if (rc == SOCKET_ERROR) {
         Warn("ListenOnConn(%s:%d): ends with SOCKET_ERROR=%d",IP2Name(ip),s,SAY
                     break;
              }
              if (rc < sizeof(MSGH)) { // Safty check
         Warn("ListenOnConn(%s:%d): Dropping too small message",IP2Name(ip),s);
         continue;
      }
              if (buf.msgh.size > MAXMSGDATA) {
         Warn("ListenOnConn(%s:%d): message %d:%d with too big size %d bytes",
            IP2Name(ip),s,buf.msgh.mode,buf.msgh.sub_mode,buf.msgh.size);
         // continue;
      }
      Diag("ListenOnConn(%x:%d): Mode=%d:%d flag=%x %d bytes:%s",IP2Name(ip),s,
         buf.msgh.mode,buf.msgh.sub_mode,buf.msgh.flags,rc,buf.mdata);

      switch(buf.msgh.mode){
      case PUT_MODE:
      case REQUEST_MODE:
```

```
case COMMIT_MODE:
case ABORT_MODE:
case ADMINREQ_MODE:
      // This could be optimized but for now:
      // (1) Fake a que handle
      // (2) Put data to the fake que

   // If the previous q handle was not for this server, fake the queue han
   if (0 == strcmp(buf.msgh.to_server,"")){ // No server name (see the rep
      lpSMBUF b = SMBUFADDR(buf.msgh.to_smbuf);
      if (strcmp(b->name,"empty"))
         strcpy(buf.msgh.to_server,b->name);
      else
         Warn("ListenOnConn(%s:%d): msg to 'empty' buffer %d",IP2Name(ip),
   }

   if (strcmp(q.msgh.to_server, buf.msgh.to_server)) {
       //Construct que handle from messge header
      memcpy(&q.msgh,&buf.msgh,sizeof(MSGH));

      q.base_flags  = 0;
      q.time_out    = SHAREDATA(time_out);

      if(!FindSMBuffers(&q,buf.msgh.to_node,buf.msgh.to_server,-1)) {
         Warn("ListenOnConn(%s:%d): ERROR No local buffers found for %s",I

         strcpy(q.msgh.to_server,""); // So next time you remake the que
         buf.msgh.to_node = ip;   // send back to orign
         buf.msgh.to_port = port; // send back to orign (port not used)
         buf.msgh.mode = NACK_MODE; // Perhaps Qsar() will set buf.msgh it
         Diag("ListenOnConn(%s:%d): Sending NACK to %p [%s]",IP2Name(ip
        if (SendRemoteMess(&buf,0))
            Warn("ListenOnConn(%s:%d): Could not SendRemoteMess(NACK)",IP2
         continue;
      }
   } else { // re-use the existing handle, but update the message header
      Diag("ListenOnConn(%s:%d): Reusing handle for %s",IP2Name(ip),s,buf.
      memcpy(&q.msgh,&buf.msgh,sizeof(MSGH));
      //q.msgh.mode = buf.msgh.mode; // Qsar() uses q.msgh.mode. Set recei
   }

   if (buf.msgh.flags & Q_FAILOVER) // No Failover please
      buf.msgh.flags = buf.msgh.flags & (!Q_FAILOVER);
 // Put data into the local que and get any answer
   Diag("ListenOnConn(%s:%d): Before Qsar() MODE=%d ",IP2Name(ip),s,q.msgh
// Last change: added last parameter of Qsar() the gotten mess header with
   if (QSUCCESS != QsendAndReceive(&q,0,0, /*buf.msgh.mode,buf.msgh.sub_mo
                    MAXMSGDATA,buf.mdata,&buf.msgh.size,&buf.msgh)) {
      Warn("ListenOnConn(%s:%d): Qsar error. Will make NACK_MODE",IP2Name(
      buf.msgh.mode = NACK_MODE; // Perhaps Qsar() will set buf.msgh itsel
   }
   else buf.msgh.mode = ACK_MODE; // Added 1/20/96 for recipts

 // Generate an ACK to the sending QNETD buffer
   Diag("ListenOnConn(%s:%d): After Qsar() MODE=%d ",IP2Name(ip),s,buf.msg

   buf.msgh.to_node = ip;   // send back to orign
   buf.msgh.to_port = port; // send back to orign (port not used)
   Diag("ListenOnConn(%s:%d): Sending n/ACK_MODE to %p [%s]",IP2Name(ip),s
   if (SendRemoteMess(&buf,0))
```

```
        Warn("ListenOnConn(%s:%d): Could not SendRemoteMess(ACK)",IP2Name(i

break;
case GET_MODE:
   Warn("ListenOnConn(%s:%d): GET_MODE not supported",IP2Name(ip),s);
break;
case ACK_MODE:
case ADMINREP_MODE:
case QNETDREP_MODE:
   abuf = SMBUFADDR(bnum = buf.msgh.from_smbuf); // Target of ACK
   Diag("ListenOnConn(%s:%d): Relaying ACK to buffer %d",IP2Name(ip),s,bnu
   memcpy(&abuf->msgh,&buf.msgh,rc); // Copy all bytes received to ACKs ta

   // Notify the sender
   if (!ReleaseSemaphore(SYNC[bnum].ackh,1,0))
      Warn("ListenOnConn(%s:%d): ReleaseSemaphore ack %d #%d",IP2Name(ip),

break;
case NACK_MODE:
   abuf = SMBUFADDR(bnum = buf.msgh.from_smbuf); // Target of ACK
   Diag("ListenOnConn(%s:%d): Relaying NACK to buffer %d",IP2Name(ip),s,bn
   memcpy(&abuf->msgh,&buf.msgh,rc); // Copy all bytes received to ACKs ta

   // Notify the sender
   if (!ReleaseSemaphore(SYNC[bnum].ackh,1,0))
      Warn("ListenOnConn(%s:%d): ReleaseSemaphore ack %d #%d",IP2Name(ip),

break;
case QNETDREQ_MODE:
   // Set up the reply message's data
   buf.msgh.mode = QNETDREP_MODE;   // send back to orign
   buf.msgh.to_node = ip;   // send back to orign
   buf.msgh.to_port = port; // send back to orign (port not used)


   if (buf.msgh.sub_mode == NETMAN_SMBUFH) {
      Diag("ListenOnConn(%s:%d): QNETDREQ_MODE sending SMBUFH=%d bytes",IP
      memcpy(&buf.mdata,sm_base,(sizeof(SMBUFH) - (2 * MAXRTSIZE) ) );
      buf.msgh.size = (sizeof(SMBUFH) - (2 * MAXRTSIZE));

   } else if (buf.msgh.sub_mode == NETMAN_SMBUFS) {
      lpBSA b = (lpBSA) &buf.mdata;
      b->nsbuf = SHAREDATA(nsbuf);
      for (i=0; i < SHAREDATA(nsbuf); i++)
         memcpy(&b->bs[i], SMBUFADDR(i) , sizeof(BS) );
      buf.msgh.size = sizeof(BSA);
   } else if (buf.msgh.sub_mode == NETMAN_SOCKETS) {
      lpSSA b = (lpSSA) &buf.mdata;
      lpST stp = STroot;
      b->sockets = 0;

      while (stp) {
         b->ss[b->sockets].ip = stp->ip;
         b->ss[b->sockets].port = stp->port;
         stp = stp->next; b->sockets++;
      }
      b->ss[b->sockets].ip = 0;
      b->ss[b->sockets].port = 0;
      buf.msgh.size = sizeof(SSA);
   } else if (buf.msgh.sub_mode == NETMAN_RT_READ) {
```

```
            lpRTA b = (lpRTA) &buf.mdata;
            if(SHAREDATA(rt_pingpong))
               memcpy(&b->RT, &SHAREDATA(RT[1]) , MAXRTSIZE );
            else
               memcpy(&b->RT, &SHAREDATA(RT[0]) , MAXRTSIZE );

            buf.msgh.size = MAXRTSIZE;
         } else if (buf.msgh.sub_mode == NETMAN_CLR_FOL) {
            SHAREDATA(failed_servers) = 0;
            buf.msgh.size = 0;
         } else if (buf.msgh.sub_mode == NETMAN_RT_NEW) { /* BROADCAST */
            FILE *RTFP = fopen(ROUTNAME,"w");   // Routing Table File
             Say("ListenOnConn(%s:%d): QNETDREQ_MODE NETMAN_RT_NEW! %s %d bytes"
            if (RTFP) {
               if ( buf.msgh.size != (int) fwrite(&buf.mdata,sizeof( char ), buf
                  Warn("ListenOnConn(%s:%d): QNETDREQ_MODE NETMAN_RT_NEW trouble
               fclose(RTFP);
               // ReadRT() is called by qnetd watchdog.c when file date changes
             } else
              Warn("ListenOnConn(%s:%d): QNETDREQ_MODE NETMAN_RT_NEW can't open
                      buf.msgh.size = 0;
         } else if (buf.msgh.sub_mode == NETMAN_RT_GET) {
            FILE *RTFP = fopen(ROUTNAME,"r");   // Routing Table File
            Say("ListenOnConn(%s:%d): QNETDREQ_MODE NETMAN_RT_GET %s",IP2Name(ip
            if (RTFP) {
               buf.msgh.size = fread(&buf.mdata,sizeof( char ), MAXMSGDATA,RTFP)
               if (!buf.msgh.size)
                 Warn("ListenOnConn(%s:%d): QNETDREQ_MODE NETMAN_RT_GET can't re
               fclose(RTFP);
            } else
               Warn("ListenOnConn(%s:%d): QNETDREQ_MODE NETMAN_RT_GET can't c
         } else if (buf.msgh.sub_mode == NETMAN_TRAN) {
         ;
         } else {
            Warn("ListenOnConn(%s:%d): QNETDREQ_MODE unknown submode %d",IP2Name
            buf.msgh.size = 0;
         }

         if (SendRemoteMess(&buf,0))
             Warn("ListenOnConn(%s:%d): Could not SendRemoteMess(QNETDREP_MODE)"

      break;
      default:
         Warn("ListenOnConn(%s:%d): Got a message I dont understand type=%d",IP2
      }

      Diag("ListenOnConn(%s:%d): waits for another message",IP2Name(ip),s);
   }

      Say("ListenOnConn(%s:%d): Done Reading socket Good Bye",IP2Name(ip),s);
      RemoveSocket(s);
      ExitThread(0);
      return(0);
}


DWORD ListenConn(){
   SOCKET      sClient,sServer = INVALID_SOCKET;
//    SOCKADDR     saClient;
```

```
    SOCKADDR_IN sas_in,sac_in;
    lpRT        this_node;
    int         sizeofsocket = sizeof(SOCKADDR);
    //char hostname[NAMESIZE];

    Diag("ListenConn Thread");


    //hostname now set by shm_init()
    //if (gethostname(hostname, sizeof (hostname)))
    //   Fail("QNETDlistenConn: gethostname failed WSAGetLastError()=%d",WSAGetLa

    if (!(this_node = RTByName(SHAREDATA(hostname))))
       Fail("QNETDlistenConn: Could not find local node %s in %s",SHAREDATA(hostn

    Diag("QNETDlistenConn: host=%s listens_on=%d",SHAREDATA(hostname),this_node->

// (2) Establish a Socket
    if ((sServer = socket(AF_INET, SOCK_STREAM, 0)) == INVALID_SOCKET)
       Fail("QNETDlistenConn: Unable to open socket, WSAGetLastError() = %d",
          WSAGetLastError());


// (3) Bind the socket
  //   _WNetGetHostAddress("bob", "wnetbnch", "tcp", &sa);

    sas_in.sin_family = AF_INET;
    sas_in.sin_port = htons( (short) this_node->qnetd_port);
    sas_in.sin_addr.s_addr = htonl(INADDR_ANY);

    if (bind(sServer, (LPSOCKADDR) &sas_in, sizeof (sas_in)) != 0)
       Fail("QNETDlisten: Unable to bind socket, WSAGetLastError() = %d",
          SAY_WSAERROR_TEXT());

    if (listen(sServer, 5)) // 5 is max number of mess to buffer
       Fail("QNETDlisten: Listen failed WSAGetLastError() = %d",
          SAY_WSAERROR_TEXT());


    while (TRUE) {
       HANDLE   thread_h;
       DWORD    thread_id;
       lpST     stab; // Socket table entry pointer

       Diag("QNETDlisten: Waiting for client connections...");
       sClient = accept(sServer,(LPSOCKADDR) &sac_in, &sizeofsocket);
       Diag("QNETDlisten: Got connected to a clinet");
       Diag("QNETDlisten: Clinets ID in network format ip=%p port=%d",
          sac_in.sin_addr.s_addr,sac_in.sin_port);
       Diag("QNETDlisten: Clinets ID in regular format ip=%p port=%d",
          ntohl(sac_in.sin_addr.s_addr),ntohs(sac_in.sin_port));

       stab = AddSocketToTable(sClient,&sac_in);

       Diag("QNETDlisten: Creating thread");
       thread_h = CreateThread(NULL,0,(LPTHREAD_START_ROUTINE)
                   ListenOnConn,stab,0,&thread_id);

    }
    return(0);// This cant happen
```

```
}

typedef struct lpar {   // Message header
   SOCKET      skt;     // socket number
   SOCKADDR_IN sac;     // inet address of client
} LPAR, *lpLPAR;


lpST FindOrMakeSocket (lpSMBUF buf) {
   int         trouble = 0;
   lpST        stp;
   lpST        stp_new;
   SOCKADDR_IN sas_in;
   HANDLE      thread_h;
   DWORD       thread_id;


  if ((stp = FindSocket(buf->msgh.to_node)) == NULL) {

      Diag("FindOrMakeSocket(%p): Making new socket.",buf->msgh.to_node);
      stp_new = (lpST) malloc(sizeof(ST));
      stp_new->next  = NULL;
      stp_new->ip    = buf->msgh.to_node;
      stp_new->port  = buf->msgh.to_port; // Not used

      if ((stp_new->s = socket(AF_INET, SOCK_STREAM, 0)) == INVALID_SOCKET) {
         Warn("FindOrMakeSocket(%p): Unable to open socket, WSAGetLastError() =
            buf->msgh.to_node,SAY_WSAERROR_TEXT());
         free(stp_new);
      } else {
         // setsockopt(stp->s, IPPROTO_TCP, TCP_NODELAY,
         //    (LPSTR) &bNoDelay, sizeof (BOOL));
         sas_in.sin_family = AF_INET;
         sas_in.sin_port = htons( (short) buf->msgh.to_port);
         sas_in.sin_addr.s_addr = htonl(buf->msgh.to_node);

         if (connect(stp_new->s, (LPSOCKADDR) &sas_in, sizeof (sas_in)) != 0) {
            Diag("FindOrMakeSocket(%p): connect() failed, WSAGetLastError() = %d
               buf->msgh.to_node,SAY_WSAERROR_TEXT());
            free(stp_new);
         } else {

            stp_new->port = ntohs(sas_in.sin_port);
            Diag("FindOrMakeSocket(%p): Connected to server on port %d",buf->msg
            Diag("FindOrMakeSocket: Starting new listening thread");
            thread_h = CreateThread(NULL,0,(LPTHREAD_START_ROUTINE)
                       ListenOnConn,stp_new,0,&thread_id);

            // Add to top of the chain
            stp_new->next = STroot;
            STroot = stp_new;
            stp = stp_new;
            // Sleep(500); // make sure args are copied before exit this routine
         }
      }
   }
   return(stp);
}

lpST  AddSocketToTable(SOCKET sock,SOCKADDR_IN *sa) {
   int ip = ntohl(sa->sin_addr.s_addr);
   int port = ntohs(sa->sin_port);
   lpST stp;

   Diag("AddSocketToTable: s=%d ip=%p",sock,ip);

   if (stp=FindSocket(ip)) {
      stp->s = sock;
   } else {
      stp = (lpST) malloc(sizeof(ST));
      stp->s = sock;
      stp->ip = ip;
      stp->port = port;

      stp->next = STroot;  // add to front of the list
      STroot = stp;
   }
   return(stp);
}

/* FAILOVER:
First try the existing IP connection  2 times
Then we will find a NEW IP and try that connection.
*/


int SendRemoteMess(lpSMBUF buf, int pass) {
// Called from QNETD, TCP/ListenOnCon
//
// Pass tells how many times we have been called.
//

   lpST  skt;
   int sentsize;
   int msgsize = sizeof(MSGH) + buf->msgh.size;


   Diag("SendRemoteMess: About to FindOrMakeSocket mode=%d",buf->msgh.mode);

   if (!(skt = FindOrMakeSocket(buf))) {
      Warn("SendRemoteMess: Could not FindOrMakeSocket for ip=%d",buf->msgh.to_n
   } else {
      Diag("SendRemoteMess: About to send mode=%d data:%s",buf->msgh.mode,buf->m

      sentsize = send(skt->s,(char *) &buf->msgh, msgsize, 0);


      if (sentsize == msgsize) {     // Send was good
         Diag("SendRemoteMess: send completed");
         SHAREDATA(stat.tx)++; // Mess sent statistics
         return(0); // Good
      } else if (sentsize == -1) {  // Send Error
         Warn("SendRemoteMess: send() error=%d", SAY_WSAERROR_TEXT());
         RemoveSocket(skt->s);    // Try to ReOpen the TCP Conn
         if (pass++ < 2) {
            Warn("SendRemoteMess: Remaking the connection");
            return(SendRemoteMess(buf,pass));
         }

      } else {  // sentsize != msgsize ; No error but not completed
```

```
            Warn("SendRemoteMess: did not send full message. sent %d of %d bytes",
            sentsize,buf->mdata);
        }
    }
    return(1); // Bad
}
```

```
/*
**
** Copyright(C)1995 MITSUBISHI ELECTRIC ITA.  ALL RIGHTS RESERVED.
** UNPUBLISHED -- RIGHTS RESERVED UNDER THE COPYRIGHT
** LAWS OF THE UNITED STATES.  USE OF A COPYRIGHT NOTICE
** IS PRECAUTIONARY ONLY AND DOES NOT IMPLY PUBLICATION
** OR DISCLOSURE.
**
** THIS SOFTWARE CONTAINS CONFIDENTIAL INFORMATION AND
** TRADE SECRETS OF MITSUBISHI ELECTRIC ITA.  USE, DISCLOSURE,
** OR REPRODUCTION IS PROHIBITED WITHOUT THE PRIOR
** EXPRESS WRITTEN PERMISSION OF MITSUBISHI ELECTRIC ITA.
**
** OpenMQ
**   Module: qnetd.c
**   Author: Derek Schwenke 9/8/95
**
*/

#include "qlib.h"
#include "qnetd.h"

DWORD WatchDog() {
   int sec=0;
   BY_HANDLE_FILE_INFORMATION Info;
   FILETIME  LastTime;
   HANDLE RTF = CreateFile( ROUTNAME ,
                            GENERIC_WRITE|GENERIC_READ,
                            FILE_SHARE_WRITE|FILE_SHARE_READ, 0,
                            OPEN_ALWAYS,
                            FILE_ATTRIBUTE_NORMAL, 0);

   while(1) {
      Sleep(1000);  // One second
      time(&SHAREDATA(time));

      if (sec++ >= 15) {
         sec = 0;
         if (RTF && GetFileInformationByHandle( RTF, &Info )) {
            //Diag("WatchDog: RT file time =  %d %d", Info.ftLastWriteTime.dwHig
            if (memcmp(&LastTime,&Info.ftLastWriteTime,sizeof(FILETIME))) {
                memcpy(&LastTime,&Info.ftLastWriteTime,sizeof(FILETIME));
               Say("WatchDog: calls ReadRT");
               ReadRT();
            } // file time differs
         }  // GetFileInformationByHandle
      } // 15 seconds
   } // forever

   CloseHandle(RTF);

   return(0);
}
```

```
/*
**
** Copyright(C)1995 MITSUBISHI ELECTRIC ITA.  ALL RIGHTS RESERVED.
** UNPUBLISHED -- RIGHTS RESERVED UNDER THE COPYRIGHT
** LAWS OF THE UNITED STATES.  USE OF A COPYRIGHT NOTICE
** IS PRECAUTIONARY ONLY AND DOES NOT IMPLY PUBLICATION
** OR DISCLOSURE.
**
** THIS SOFTWARE CONTAINS CONFIDENTIAL INFORMATION AND
** TRADE SECRETS OF MITSUBISHI ELECTRIC ITA.  USE, DISCLOSURE,
** OR REPRODUCTION IS PROHIBITED WITHOUT THE PRIOR
** EXPRESS WRITTEN PERMISSION OF MITSUBISHI ELECTRIC ITA.
**
** OpenMQ
**   Module: ramq.c
**   Author: Derek Schwenke 9/8/95
**
** This is a single threaded ram queue
*/

// Each message could be an enq or a deq.

#include "qlib.h"

#define  QSIZE 8
#define  ENQ ramq[ramq_head]
#define  DEQ ramq[ramq_tail]
#define  ENTRIES  ((( ramq_head + QSIZE )  - ramq_tail) % QSIZE)

SMBUF ramq[QSIZE];
int   ramq_head = 0; // points to empty space
int   ramq_tail = 0; // points to filled entry


int main(int argc, char **argv) {
   lpQHANDLE    q;
   int          status;
   char         qname[NAMESIZE];

   if(argc> 1) strcpy(qname,argv[1]);
   else        strcpy(qname,"Q1");


   Say("RAMQ(%s): About to Qopen",qname);
   if (!(q = Qopen(qname,GETTING)))
      Say("RAMQ(%s) Could not open the queue",qname);



   while (QSUCCESS == Qgnp_get(q,0,0,&ENQ.msgh,ENQ.mdata,MAXMSGDATA)){
      Say("RAMQ(%s) Server Got :%s",qname,ENQ.mdata);

      if (ENQ.msgh.mode == PUTTING) {
         Say("RAMQ(%s): ENQUEs message %s",qname,ENQ.mdata);
         ramq_head = ++ramq_head % QSIZE;
         if (ramq_head == ramq_tail) Say("RAMQ(%s): OVERFLOW!",qname);
         Qgnp_put(q, ACK,0,0);
```

```
    } else if (ENQ.msgh.mode == REQUESTING) {
       if (ramq_head == ramq_tail) {
          Say("RAMQ(%s): EMPTY",qname);
          Qgnp_put(q, EMPTY, 0,0);
       } else {
          Say("RAMQ(%s): DEQUEs message %s",qname,DEQ.mdata);
          Qgnp_put(q, RETURN_DATA, DEQ.mdata, DEQ.msgh.size);
          ramq_tail = ++ramq_tail % QSIZE;
       }

    } else {
       Say("RAMQ(%s): Unexpected Mode=%d",qname,ENQ.msgh.mode);
       Qgnp_put(q,NACK, 0, 0);
    }

    Say("RAMQ(%s) has %d entries.",qname,ENTRIES);
 }

 Ask("RAMQ(%s): all done",qname);
 return(0);
}
```

```
User: root
Host: bunny
Class: bunny
Job: stdin
```

```
/*
**
** Copyright(C)1995 MITSUBISHI ELECTRIC ITA.  ALL RIGHTS RESERVED.
** UNPUBLISHED -- RIGHTS RESERVED UNDER THE COPYRIGHT
** LAWS OF THE UNITED STATES.  USE OF A COPYRIGHT NOTICE
** IS PRECAUTIONARY ONLY AND DOES NOT IMPLY PUBLICATION
** OR DISCLOSURE.
**
** THIS SOFTWARE CONTAINS CONFIDENTIAL INFORMATION AND
** TRADE SECRETS OF MITSUBISHI ELECTRIC ITA.  USE, DISCLOSURE,
** OR REPRODUCTION IS PROHIBITED WITHOUT THE PRIOR
** EXPRESS WRITTEN PERMISSION OF MITSUBISHI ELECTRIC ITA.
**
** OpenMQ
**   Module: ramq.c
**   Author: Derek Schwenke 9/8/95
**
** This is a single threaded ram queue
*/

// Each message could be an enq or a deq.

#include "qlib.h"
#include "qadmin.h"
// Que entry states:
#define  INVALID     0
#define  VALID       1
#define  PENDING_PUT 2
#define  PENDING_GET 3

#define  MAXQSIZE    1001
#define  MAXRAMQDATA 100
#define  ENQ ramq[ramq_head]
#define  DEQ ramq[ramq_tail]
#define  MES ramq[ramq_mess]
#define  MODE ENQ.msgh.mode
#define  FLAGIS(X) BITSET(X,ENQ.msgh.flags)
#define  QINC(X)    X = ++X % QSIZE
#define  QDEC(X)    X = --X % QSIZE
#define  USEDSPACE ((( ramq_head + QSIZE )  - ramq_tail) % QSIZE)
#define  ALLENTRIES  (st.committed_entries + st.pending_gets + st.pending_puts)

typedef  struct rqbuf {     //Ramq
   int   status;
   MSGH  msgh;
   char  mdata[MAXRAMQDATA];
} RQBUF, *lpRQBUF;


QADMSTATS st; // Statistics
RQBUF     ramq[MAXQSIZE];
int       QSIZE = MAXQSIZE;
int       ramq_head = 0; // points to empty space
int       ramq_tail = 0; // points to filled entry
int       ramq_mess = 0; // points to entry that was just filled
char      qname[NAMESIZE];


void PrintQ() {
   char n[80]; int i;
   for ( i = 0 ; i < QSIZE ; i++ ){

      if (ramq[i].status == INVALID)  // Clear entry just for display
         *ramq[i].mdata = ramq[i].msgh.mid.host = ramq[i].msgh.mid.tid = 0;

      strcpy(n,"");
      if (i == ramq_head) strcat(n," <- head");
      if (i == ramq_tail) strcat(n," <- tail");
      Say("  %d mid=%p:%d status=%d : %s%s",i,
         ramq[i].msgh.mid.host, ramq[i].msgh.mid.tid, ramq[i].status,
         ramq[i].mdata,n);

   }
}


void StatReset(char *qname){
   st.amt_free_dspace = -1;

   st.num_puts = 0;
   st.num_gets = 0;
   st.num_commits = 0;
   st.num_aborts = 0;
   if (qname) strcpy(st.physical_qname,qname);
   if (qname) strcpy(st.logical_qname,"");
   st.max_entries_limit = MAXQSIZE - 1;
}

void QueReset(char *qname){
   int i;
   for (i = 0;  i < QSIZE ; i++)  ramq[i].status = 0;
   StatReset(qname);
   st.committed_entries = 0;
   st.pending_gets = 0;
   st.pending_puts = 0;
   ramq_head = ramq_tail = 0;
   st.num_restarts++;
   st.last_restart_time = SHAREDATA(time);
   st.node_address = SHAREDATA(hostip);
   strcpy(st.node_name,SHAREDATA(hostname));
}



void SetSize(int size){ // Resize the que
   size++; // Internaly it one lager
   QSIZE = size;
   QueReset(NULL);
}

int FindNthMess(int nth, int mstat1, int mstat2) { // Finds adm req data
   int p = ramq_tail;
   nth++;

   while ((nth) && (p != ramq_head)) {
      if (ramq[p].status == mstat1 || ramq[p].status == mstat2 ) nth--;
      QINC(p);
   }

   if (nth) return(-1);
   else     return(QDEC(p));
}


void tran(action) { // COMMIT or ABORT all parts of a transaction
   int p_gets = st.pending_gets, i;
   int p_puts = st.pending_puts;
   Diag("RAMQ-TRAN: %d mid=%p:%d",
      action,MES.msgh.mid.host,MES.msgh.mid.tid);
   for (i = 0 ; i < QSIZE ; i++)
      if (ramq[i].msgh.mid.tid == MES.msgh.mid.tid)
         if (ramq[i].msgh.mid.host == MES.msgh.mid.host)
            if (ramq[i].status == PENDING_PUT){      // Uncommitted PUT
               if (action) {
                  ramq[i].status = VALID;   // Committed put
                  st.num_puts++;
                  st.committed_entries++;
               } else
                  ramq[i].status = INVALID; // Aborted put
               st.pending_puts--;
            } else if (ramq[i].status == PENDING_GET){ // Uncommitted GET
               if (action) {
                  ramq[i].status = INVALID; // Committed get
                  st.num_gets++;
               } else {
                  ramq[i].status = VALID;   // Aborted get
                  st.committed_entries++;
               }
               st.pending_gets--;
            }

   if ((p_gets == st.pending_gets) && (p_puts == st.pending_puts))
      Warn("tran(%d) did not commit or abort anything",action);
   //if (p_gets = p_gets - st.pending_gets)  Say("tran(%d) clears %d gets",actio
   //if (p_puts = p_puts - st.pending_puts)  Say("tran(%d) clears %d puts",actio

   if(DEQ.status == 0 && ramq_head != ramq_tail) Diag("RAMQ-TRAN: Adjusting Tail
   while(DEQ.status == 0 && ramq_head != ramq_tail) //Adjust the tail
      QINC(ramq_tail);

   if(action) st.num_commits++;
   else       st.num_aborts++;

}


DWORD RQ(){
   lpQHANDLE     q;
   int           x;
   char          keys[MAXMSGDATA];
   lpQADMCTLS    pCTL; // Cast to ramq[].data
   lpQADMSEL     pKEY;
   lpMID         pMID;


   Say("RAMQ(%s): %d Entires",qname,QSIZE - 1);
   if (!(q = Qopen(qname,GET_MODE,0,0,0,0,0)))
      Say("RAMQ(%s) Could not open the logical queue",qname);
```

```
st.num_restarts = -1;
st.first_start_time = SHAREDATA(time);
st.qget_state = 1; // enable puts
st.qput_state = 1; // enable gets

QueReset(qname);

while (QSUCCESS == QlistenBeforeReply(q,&ENQ.msgh,ENQ.mdata,MAXMSGDATA)) {
   Diag("RAMQ(%s) Server Got md=%d mid:%p:%d :%s",qname,MODE,ENQ.msgh.mid.hos
   ramq_mess = ramq_head; // mess is where the last message went.

   if (MODE == PUT_MODE) {
      if (!st.qput_state) {
         Say("RAMQ(%s) put is disabled!",qname);
         QreplyAfterListen(q,ACK_MODE,SUB_MODE_DISABLED, 0,0,0);
      } else if (USEDSPACE >= QSIZE - 1) {
         Say("RAMQ(%s) is FULL!",qname);
         QreplyAfterListen(q,ACK_MODE,SUB_MODE_FULL, 0,0,0);
      } else {
         Diag("RAMQ(%s): ENQUEs message %s",qname,ENQ.mdata);

         if (FLAGIS(Q_TRAN)) {
            ENQ.status = PENDING_PUT;
            st.pending_puts++;
         } else {
            ENQ.status = VALID;
            st.num_puts++;
            st.committed_entries++;
         }

         QINC(ramq_head);
         QreplyAfterListen(q, ACK_MODE,SUB_MODE_OK, 0,0,0);
     }


   } else if (MODE == REQUEST_MODE) {
      // Find next valid message
      int g = ramq_tail;
      while (ramq[g].status != VALID && g != ramq_head) QINC(g);
      if (!st.qget_state) {
         Say("RAMQ(%s) get is disabled!",qname);
         QreplyAfterListen(q,ACK_MODE,SUB_MODE_DISABLED, 0,0,0);
      } else if (g == ramq_head) {
         Diag("RAMQ(%s): is EMPTY!",qname);
         QreplyAfterListen(q, ACK_MODE,SUB_MODE_EMPTY, 0,0,0);
      } else {
         Diag("RAMQ(%s): DEQUEs message %s",qname,ramq[g].mdata);
         QreplyAfterListen(q, ACK_MODE,SUB_MODE_OK, ramq[g].mdata, ramq[g].ms

         if (FLAGIS(Q_TRAN)) {
            ramq[g].status = PENDING_GET; // Uncommtted get
            memcpy(&ramq[g].msgh.mid,&ENQ.msgh.mid,sizeof(MID)); // This MSG
            st.pending_gets++;
         } else {
            ramq[g].status = INVALID; // Committed get
            st.num_gets++;
         }
         st.committed_entries--;

         while(ramq[ramq_tail].status == INVALID && ramq_head != ramq_tail)
```

```
            QINC(ramq_tail); //Adjust the tail
        }

    } else if (MODE == ADMINREQ_MODE) { // Administrative messages

        switch(ENQ.msgh.sub_mode) {
        case QADM_REQ_STATS:
            st.max_entries = QSIZE - 1;
            st.holey_entries = USEDSPACE - ALLENTRIES ;
            st.num_free_entries = QSIZE - 1 - USEDSPACE;
            QreplyAfterListen(q,ADMINREP_MODE,SUB_MODE_OK,(char *) &st, sizeof(s


        break;
        case QADM_REQ_COM_DATA:

            if (-1 != (x = FindNthMess((int) *ENQ.mdata,VALID,VALID)))
                QreplyAfterListen(q,ADMINREP_MODE,SUB_MODE_OK, ramq[x].mdata, 0,&
            else
                QreplyAfterListen(q,ADMINREP_MODE,0, 0, 0,0);

        break;
        case QADM_REQ_UNCOM_DATA:

            if (-1 != (x = FindNthMess((int) *ENQ.mdata,PENDING_PUT,PENDING_GET)
                QreplyAfterListen(q,ADMINREP_MODE,SUB_MODE_OK, ramq[x].mdata, 0,&
            else
                QreplyAfterListen(q,ADMINREP_MODE,0, 0, 0,0);
        break;

        case QADM_SET_CONTROLS:
            pCTL = (lpQADMCTLS) &ramq[ramq_mess].mdata;
            st.qput_state = pCTL->enable_qputs_flag;
            st.qget_state = pCTL->enable_qgets_flag;

            Say("%d qput_state",st.qput_state);
            Say("%d qget_state",st.qget_state);

            if ( pCTL->stats_reset_flag ) StatReset(qname);
            if ( pCTL->full_reset_flag ) QueReset(qname);
            if ((pCTL->max_entries_value != QSIZE - 1) && (pCTL->max_entries_val
                SetSize(pCTL->max_entries_value);
            QreplyAfterListen(q,ADMINREP_MODE,SUB_MODE_OK, 0, 0,0);

            if ( pCTL->shutdown_flag ) { Qclose(&q,0); exit(0);}

            // PrintQ();

        break;
        case QADM_REQ_MSG:
            {   int e = ramq_tail;
                pMID = (lpMID) &ramq[ramq_mess].mdata;
                Diag("Finding mid  %p %d %d",pMID->host,pMID->tid,pMID->uid);
                while(e != ramq_head) {
                    if(0 == memcmp(&ramq[e].msgh.mid,&ramq[ramq_mess].mdata,sizeof
                    e = QINC(e);
                }
                if ((e == ramq_head) || (ramq[e].status != VALID))
                    x = 0;
                else
            x = ramq[e].msgh.size;
         QreplyAfterListen(q,ADMINREP_MODE,SUB_MODE_OK,   ramq[e].mdata, x
      }
   break;
    case QADM_REQ_SEL_DATA:  // KEY SEARCH
      pKEY = (lpQADMSEL) &ramq[ramq_mess].mdata;
      pMID = (lpMID) &keys;
      if (pKEY->num_preds) {
         int gt,lt, e = ramq_tail, p, p_matches = 0;

         if      (pKEY->search_type == SEARCH_UNCOM_ENT) {gt = PENDING_PUT
         else if (pKEY->search_type == SEARCH_ALL_ENT )  {gt = VALID; lt =
         else                                            {gt = lt = VALID;

         while (e != ramq_head){  // Scan the que looking for a match
            if ((ramq[e].status >= gt) && (ramq[e].status <= lt) ) {
               p_matches = 0;
               for (p = 0; p < pKEY->num_preds; p++) {
                  if ((UINT) pKEY->preds[p].offset < (sizeof(MSGH) + ramq[

                     if (pKEY->preds[p].pred_type == INT_SEARCH_TYPE) {
                        //int iv = *((int*)((char *)&ramq[e].msgh + pKEY->
                        if ((pKEY->preds[p].min_switch == 0)
                           || (pKEY->preds[p].min_int_val <=
                           *((int*)((char *)&ramq[e].msgh + pKEY->preds[p]
                              p_matches++;
                        if ((pKEY->preds[p].max_switch == 0)
                           || (pKEY->preds[p].max_int_val >=
                           *((int*)((char *)&ramq[e].msgh + pKEY->preds[p]
                              p_matches++;

                     } else if (pKEY->preds[p].pred_type == SHORT_SEARCH_T
                        if ((pKEY->preds[p].min_switch == 0)
                           || (pKEY->preds[p].min_sh_val <=
                           *((short*)((char *)&ramq[e].msgh + pKEY->preds[
                              p_matches++;
                        if ((pKEY->preds[p].max_switch == 0)
                           || (pKEY->preds[p].max_sh_val >=
                           *((short*)((char *)&ramq[e].msgh + pKEY->preds[
                              p_matches++;

                     } else if (pKEY->preds[p].pred_type == STR_SEARCH_TYP
                        if ((pKEY->preds[p].min_switch == 0)
                           || (0 >= strncmp(pKEY->preds[p].min_str_val,
                           ((char *)&ramq[e].msgh + pKEY->preds[p].offset)
                              p_matches++;
                        if ((pKEY->preds[p].max_switch == 0)
                           || (0 <= strncmp(pKEY->preds[p].max_str_val,
                           ((char *)&ramq[e].msgh + pKEY->preds[p].offset)
                              p_matches++;
                     }
                  } // offset within size
               } // for 3 predicates
               if (p_matches == 2 * pKEY->num_preds) {
                  memcpy(pMID,&ramq[e].msgh.mid,sizeof(MID));
                  pMID = (lpMID)((char *)pMID + sizeof(MID));
               }
            } // if Valid
            e = QINC(e);
         } // Scan que
```

```
            } // Predicates

            Diag("Matches = %d",(x = (((char *)pMID - keys) / sizeof(MID)) ));
            QreplyAfterListen(q,ADMINREP_MODE,SUB_MODE_OK,  keys, ((char *)pMID
         break;
         default: Say("What?");
         }

      } else if (MODE == COMMIT_MODE) {
         tran(1);
         QreplyAfterListen(q,ACK_MODE,SUB_MODE_OK, 0, 0,0);

      } else if (MODE == ABORT_MODE) {
         tran(0);
         QreplyAfterListen(q,ACK_MODE,SUB_MODE_OK, 0, 0,0);

      } else {
         Warn("RAMQ(%s): Unexpected Mode=%d",qname,ENQ.msgh.mode);
         QreplyAfterListen(q,ACK_MODE,SUB_MODE_BAD_REQ, 0, 0,0); // Unexpected m
      }

      if (FLAGIS(Q_TRAN_END)) tran(1);

      Diag("RAMQ(%s) has %d entries %d not commited.",qname,
         st.committed_entries,st.pending_puts + st.pending_gets);

      if (MODE != ADMINREQ_MODE) if (SHAREDATA(diag)) PrintQ(); // Print the que
   }

   Say("Bye",qname);
   return(0);

}



int main(int argc, char **argv) {
   HANDLE      hRQ;
   DWORD       idRQ;

   if(argc> 1) strcpy(qname,argv[1]);
   else        strcpy(qname,"QS1");

   if(argc> 2) {
      sscanf(argv[2],"%d",&QSIZE);
      QSIZE++;
      QSIZE = __min(MAXQSIZE,QSIZE);
      QSIZE = __max(2,QSIZE);
   }

   hRQ  = CreateThread(NULL,0,(LPTHREAD_START_ROUTINE) RQ,  0,0,&idRQ);


   Ask("All done");

   Qclose(NULL,qname); // Removes shared buffer so messages dont need to time-ou
   return(0);
```

```
}
```

```
/*
**
** Copyright(C)1995 MITSUBISHI ELECTRIC ITA.  ALL RIGHTS RESERVED.
** UNPUBLISHED -- RIGHTS RESERVED UNDER THE COPYRIGHT
** LAWS OF THE UNITED STATES.  USE OF A COPYRIGHT NOTICE
** IS PRECAUTIONARY ONLY AND DOES NOT IMPLY PUBLICATION
** OR DISCLOSURE.
**
** THIS SOFTWARE CONTAINS CONFIDENTIAL INFORMATION AND
** TRADE SECRETS OF MITSUBISHI ELECTRIC ITA.  USE, DISCLOSURE,
** OR REPRODUCTION IS PROHIBITED WITHOUT THE PRIOR
** EXPRESS WRITTEN PERMISSION OF MITSUBISHI ELECTRIC ITA.
**
** OpenMQ
**   Module: ramqcs.c
**   Author: Derek Schwenke 9/8/95
**
** ramq's client and server
**
*/

#include "qlib.h"
#include "rt.h"
#include "qadmin.h"

void Help(){
   Say("O <name> - Open");
   Say("P <data> - Put");
   Say("G        - Get");
   Say("B        - Begin tran");
   Say("C        - Commit tran");
   Say("A        - Abort tran");
   Say("E        - End tran on next op");
   Say("S        - Single tran op");
   Say("N        - No tran");
   Say("L        - List status");
   Say("D        - Diagnostics on/off");
   Say("R        - Reset all");
   Say("T        - Reset statistics");
   Say("K        - Shutdown");
   Say("Q <put/get> - Enable/disable ops");
   Say("H        - Help");
   Say("Z <numb> - Set size of que");
}

main(int argc, char **argv) {
   int          ret,i,flags=0,next_flag=0,sz;
   char         qname[NAMESIZE];
   MSGH         mh;
   lpQHANDLE    Q;
   lpQADMSTATS  pSTA = (lpQADMSTATS) AskAnswer;
 //  lpQADMCTLS   pCTL = (lpQADMCTLS) AskAnswer;
   QADMCTLS     ctl;

   memset(&ctl,0,sizeof(ctl));
   ctl.enable_qputs_flag++;
   ctl.enable_qgets_flag++;
   if(argc==2) strcpy(qname,argv[1]);
   else        strcpy(qname,"Q1"); // appl
   Say("RAMQCS: Server=%s",qname);
```

```
// Open the queue for putting
  if (!(Q = Qopen(qname,PUT_MODE,0,0,0,0,0)))
     Warn("RAMQCS: could not open the que %s",qname);


// Put data into the que
  Say("Open Put Get Begin End Commit Abort Single List Kshutdown sTat siZe Rese
  while( strlen(Ask("ramqcs"))) {

#define SAR_WAS_OK(A1,A2,A3,A4,A5,A6,A7,A8,A9,A10)  QSUCCESS >= (ret = QsendAndR
     if (strchr("Pp",*AskAnswer) && AskAnswer[1] == ' ' && strlen(AskAnswer) >

        if (SAR_WAS_OK(Q,PUT_MODE,0, flags,strlen(AskAnswer),&AskAnswer[2],
           LINESIZE,AskAnswer,&sz,&mh)) {

           if (ret == QSUCCESS_FAILOVER) Say("RAMQCS: Failover successfull");
           if (mh.mode == ACK_MODE && mh.sub_mode == SUB_MODE_OK)
              Say("RAMQCS: Succceded: (mode=%d) %s",mh.mode,AskAnswer);
           else
              if (mh.sub_mode == SUB_MODE_FULL)
                 Say("RAMQCS: que was FULL");
              else
                 Say("RAMQCS: Put failed %d",mh.sub_mode);
        } else
           Say("RAMQCS: Qput comm error");
        flags = next_flag;
     }

     else if (strchr("Gg",*AskAnswer)) { // Get

        if (SAR_WAS_OK(Q,REQUEST_MODE,0, flags,strlen(AskAnswer)+1,AskAnswer,
           LINESIZE,AskAnswer,&sz,&mh)){

           if (ret == QSUCCESS_FAILOVER) Say("RAMQCS: Failover successfull");
           if (mh.mode == ACK_MODE && mh.sub_mode == SUB_MODE_OK)
              Say("RAMQCS: Succceded: (mode=%d) %d bytes %s",mh.mode,mh.size,As
           else
              if (mh.sub_mode == SUB_MODE_EMPTY)
                 Say("RAMQCS: que was EMPTY");
              else
                 Say("RAMQCS: Qget failed %d",mh.sub_mode);
        }
        flags = next_flag;
     }

     else if (strchr("Ll",*AskAnswer)) { // List

        if (QSUCCESS >= QsendAndReceive(Q,ADMINREQ_MODE,QADM_REQ_STATS, 0,0,0,
           sizeof(QADMSTATS),AskAnswer,&sz,&mh)) {
           int pg,pp,ce,he,fe,me;
           Say("\n\n%s:%s %s %x",pSTA->physical_qname,pSTA->logical_qname,pSTA-
           Say("%7d committed_entries",ce = pSTA->committed_entries);
           Say("%7d pending_gets",pg = pSTA->pending_gets);
           Say("%7d pending_puts",pp = pSTA->pending_puts);
           Say("%7d holey_entries",he = pSTA->holey_entries);
           Say("%7d num_free_entries",fe = pSTA->num_free_entries);
           if (pSTA->amt_free_dspace != -1)
           Say("%7d amt_free_dspace",pSTA->amt_free_dspace);
           if (pSTA->max_entries_limit >= pSTA->max_entries )
        Say("%7d max      %4d limit",me = pSTA->max_entries,pSTA->max_entrie
        else
        Say("%7d max      ",me = pSTA->max_entries);
        Say("%7d putstate %4d getstate",pSTA->qput_state,pSTA->qget_state);
        Say("%7d puts     %4d gets",pSTA->num_puts,pSTA->num_gets);
        Say("%7d committs %4d aborts %d restarts",pSTA->num_commits,pSTA-
        Say("\n\tFirst start time  %s\tLast restart time %s",
           ctime(&pSTA->first_start_time),ctime(&pSTA->last_restart_time));
        if (me - (ce + pg + pp + he + fe))
           Warn("REPORTED ENTRIES OFF BY %d",me - (ce + pg + pp + he + fe));

        *AskAnswer = i = 0; sz = 1;
        while(sz){
           *AskAnswer = i++;
           if (QSUCCESS >= QsendAndReceive(Q,ADMINREQ_MODE,QADM_REQ_COM_DATA
               0,sizeof(int),AskAnswer, LINESIZE,AskAnswer,&sz,&mh))
           if (sz) Say("%d : %s",i,AskAnswer);
        }


        *AskAnswer = i = 0; sz = 1; Say("");
        while(sz){
           *AskAnswer = i++;
           if (QSUCCESS >= QsendAndReceive(Q,ADMINREQ_MODE,QADM_REQ_UNCOM_DA
             0,sizeof(int),AskAnswer, LINESIZE,AskAnswer,&sz,&mh))
           if (sz) Say("%d submode=%d : %s",i,mh.sub_mode,AskAnswer);
        }
     }

}

else if (strchr("Qq",*AskAnswer)) { // Shutdown
   if ('0' == AskAnswer[4]) ctl.enable_qputs_flag = 0;
   if ('1' == AskAnswer[4]) ctl.enable_qputs_flag = 1;
   if ('0' == AskAnswer[8]) ctl.enable_qgets_flag = 0;
   if ('1' == AskAnswer[8]) ctl.enable_qgets_flag = 1;
     if (QSUCCESS >= QsendAndReceive(Q,ADMINREQ_MODE,QADM_SET_CONTROLS,
        0,sizeof(QADMCTLS),(char*)&ctl,  0,0,&sz,&mh))
        Say("puts %d gets %d",ctl.enable_qputs_flag,ctl.enable_qgets_flag
}

else if (strchr("Kk",*AskAnswer)) { // Shutdown
   ctl.shutdown_flag++;
   if (QSUCCESS >= QsendAndReceive(Q,ADMINREQ_MODE,QADM_SET_CONTROLS,
        0,sizeof(QADMCTLS),(char*)&ctl,  0,0,&sz,&mh)) Say("Shutdown");
   ctl.shutdown_flag--;
}

else if (strchr("Rr",*AskAnswer)) { // Reset
   ctl.full_reset_flag++;
   if (QSUCCESS >= QsendAndReceive(Q,ADMINREQ_MODE,QADM_SET_CONTROLS,
        0,sizeof(QADMCTLS),(char*)&ctl,  0,0,&sz,&mh)) Say("full Reset");
   ctl.full_reset_flag--;
}

else if (strchr("Zz",*AskAnswer) && (sscanf(AskAnswer,"%*s %d",&i))) {
   if ((i) < 2)  Say("Bad Size");
   else { ctl.max_entries_value = i;
         if (QSUCCESS >= QsendAndReceive(Q,ADMINREQ_MODE,QADM_SET_CONTROLS
         0,sizeof(QADMSTATS),(char*)&ctl,  0,0,&sz,&mh)) Say("Size=%d",i);
```

```
      } ctl.max_entries_value = 0;
    }

    else if (strchr("Tt",*AskAnswer)) { // Kleer
       ctl.stats_reset_flag++;
       if (QSUCCESS >= QsendAndReceive(Q,ADMINREQ_MODE,QADM_SET_CONTROLS,
             0,sizeof(QADMCTLS),(char*)&ctl,  0,0,&sz,&mh)) Say("Stats reset")
       ctl.stats_reset_flag--;
    }
    /*
    else if (strchr("x",*AskAnswer)) { // Broadcast
       BroadcastRT();
    }
    else if (strchr("X",*AskAnswer)) { // Get
       CopyRT(RT_IP(RTByName(&AskAnswer[2])));
    }
    */

    else if (strchr("Hh",*AskAnswer)) Help();
    else if (strchr("Bb",*AskAnswer)&&!(flags & Q_TRAN)) {flags = Q_TRAN_BEGIN
    else if (strchr("Cc",*AskAnswer)&&flags) {Qcommit(COMMIT_MODE);flags = nex
    else if (strchr("Aa",*AskAnswer)&&flags) {Qcommit(ABORT_MODE); flags = nex
    else if (strchr("Ee",*AskAnswer)) {flags = Q_TRAN_END; next_flag = 0;}
    else if (strchr("Ss",*AskAnswer)) {flags = Q_TRAN_SINGLE; next_flag = 0;}
    else if (strchr("Nn",*AskAnswer)) {flags = next_flag = 0;}
    else if (strchr("Dd",*AskAnswer)) {SHAREDATA(diag) = (!SHAREDATA(diag));}
    else if (strchr("Oo",*AskAnswer) && AskAnswer[1] == ' ' && (strlen(AskAnsw
       int open_flags = 0;
       sscanf(AskAnswer,"%*s %s %lx",qname,&open_flags);
       Say("RAMQCS: Open %s flags=%x",qname,open_flags);
       if (!(Q = Qopen(qname,PUT_MODE,0,open_flags,0,0,0)))
          Fail("RAMQCS: could not open the que %s",qname);
    } else
       Say("Try again!");


    if (flags) Diag("RAMQCS: flags %p",flags);
  }

  Say("RAMQCS: ALL DONE");
  return(0);

}
```

```
/*
**
** Copyright(C)1995 MITSUBISHI ELECTRIC ITA.  ALL RIGHTS RESERVED.
** UNPUBLISHED -- RIGHTS RESERVED UNDER THE COPYRIGHT
** LAWS OF THE UNITED STATES.  USE OF A COPYRIGHT NOTICE
** IS PRECAUTIONARY ONLY AND DOES NOT IMPLY PUBLICATION
** OR DISCLOSURE.
**
** THIS SOFTWARE CONTAINS CONFIDENTIAL INFORMATION AND
** TRADE SECRETS OF MITSUBISHI ELECTRIC ITA.  USE, DISCLOSURE,
** OR REPRODUCTION IS PROHIBITED WITHOUT THE PRIOR
** EXPRESS WRITTEN PERMISSION OF MITSUBISHI ELECTRIC ITA.
**
** OpenMQ
**   Module: ramqcs.c
**   Author: Derek Schwenke 9/8/95
**
** ramq's client and server
**
*/

#include "qlib.h"
extern   char AskAnswer[LINESIZE];

main(int argc, char **argv) {
   int         status,sz;
   char        qname[NAMESIZE];
   MSGH        mh;
   lpQHANDLE   Q;

   ReadParms();

 // Name that q
   if(argc==2) strcpy(qname,argv[1]);
   else        strcpy(qname,"Q1"); // appl
   Say("Server=%s",qname);


 // Open the queue for putting
   if (!(Q = Qopen(qname,PUTTING)))
      Fail("APPC: could not open the que %s",qname);


 // Ask the user to Put or Get
   while( strlen(Ask("RAMQS: Put Get Open?"))) {

      if (*AskAnswer == 'p' || *AskAnswer == 'P') {
         Q->msgh.mode = PUTTING;
         if (strlen(Ask("RAMQS: Put messge")))
            if (QSUCCESS == Qpng(Q,flags,strlen(AskAnswer)+1,AskAnswer,
               LINESIZE,AskAnswer,&sz,&mh))
               Say("RAMQS: Succceded: (mode=%d) %s",mh.mode,AskAnswer);
            else
               Say("RAMQS: Qput failed");
      }

      else if (*AskAnswer == 'g' || *AskAnswer == 'G') {
         Q->msgh.mode = REQUESTING;
         if (QSUCCESS == Qpng(Q,flags,strlen(AskAnswer)+1,AskAnswer,
            LINESIZE,AskAnswer,&sz,&mh))
            Say("RAMQS: Succceded: (mode=%d) %d bytes %s",mh.mode,mh.size,AskAns
         else
            Say("RAMQS: Qput failed");
      }


      else if (*AskAnswer == 'o' || *AskAnswer == 'O' && strlen(AskAnswer) > 2)
         Say("RAMQS: Open");
         sscanf(AskAnswer,"%*s %s",qname);
         if (!(Q = Qopen(qname,PUTTING)))
            Fail("APPC: could not open the que %s",qname);
      }
   }

   Say("RAMQS: ALL DONE");
   return(0);

}
```

```
///////////////////////////////////////////////////////////////////////////////
//
// Common admin defs used by OpenMQ Qservers
//
///////////////////////////////////////////////////////////////////////////////

#define DISABLED                0L
#define ENABLED                 1L
#define QADM_REQ_STATS          2L
#define QADM_REQ_SEL_DATA       3L
#define QADM_REQ_MSG            7L
#define QADM_REQ_COM_DATA       8L
#define QADM_REQ_UNCOM_DATA     9L
#define QADM_SET_CONTROLS       10L

#define SHORT_SEARCH_TYPE       0L
#define INT_SEARCH_TYPE         1L
#define STR_SEARCH_TYPE         2L

#define SEARCH_ALL_ENT          0L
#define SEARCH_COM_ENT          1L
#define SEARCH_UNCOM_ENT        2L
#define KEY_STR_SIZE            50


typedef struct qadmstats {
   CHAR       logical_qname[NAMESIZE];   // logical que name
   CHAR       physical_qname[NAMESIZE];  // physical que server name
   CHAR       node_name[NAMESIZE];       // node name of que server
   int        node_address;              // IP address of que server
   int        max_entries_limit;         // upper limit on max value
   int        max_entries;               // current max value
   int        committed_entries;         // commited entries
   int        pending_gets;              // pending GET entries
   int        pending_puts;              // pending PUT entries
   int        holey_entries;             // #holes in the queue
   int        num_free_entries;          // # of free entries that can be used
   int        amt_free_dspace;           // amt of free_space left in queue
   short      qget_state;                // gets_are_enabled
   short      qput_state;                // puts_are_enabled
   int        num_puts;                  // committed puts only
   int        num_gets;                  // committed gets
   int        num_commits;               // # of committed operations
   int        num_aborts;                // # of aborted operations
   int        num_restarts;              // # of server restarts
   time_t     first_start_time;          // time of fresh start
   time_t     last_restart_time;         // time of last restart
//   int        num_recov_tries;
//   time_t     last_recov_time;
} QADMSTATS, *lpQADMSTATS;


typedef struct qadmctls {
   int        max_entries_value;     // extends max entries; later...
//   int        clear_all_tran_flag;  // cleans up stuck entries/compaction
   int        full_reset_flag;
   int        shutdown_flag;
   int        stats_reset_flag;
   int        enable_qputs_flag;
   int        enable_qgets_flag;
   int         halt_flag;
} QADMCTLS, *lpQADMCTLS;


typedef struct predstr {
   int         pred_type;                 // predicate type
   int         offset;                    // offset within QMSG structure
   short       min_switch;                // 1=ENABLED/DISABLED min switch
   short       max_switch;                // 1=ENABLED/DISABLED max switch
   short       min_sh_val;                // min short value
   short       max_sh_val;                // max short value
   int         min_int_val;               // min int value
   int         max_int_val;               // max int value
   char        min_str_val[KEY_STR_SIZE]; // min string value
   char        max_str_val[KEY_STR_SIZE]; // max string value
   int         min_str_len;               // min string length
   int         max_str_len;               // max string length

} PREDSTR, *lpPREDSTR;

typedef struct qadmsel {
   int         num_preds;         // number of predicates up to max of 3
   int         search_type;       // SEARCH_ALL_ENT, SEARCH_COM_ENT, SEARCH_UNCO
   PREDSTR     preds[3];          // array of predicates
} QADMSEL, *lpQADMSEL;

#define MAXSEL
```

```
/*
**
** Copyright(C)1995 MITSUBISHI ELECTRIC ITA.  ALL RIGHTS RESERVED.
** UNPUBLISHED -- RIGHTS RESERVED UNDER THE COPYRIGHT
** LAWS OF THE UNITED STATES.  USE OF A COPYRIGHT NOTICE
** IS PRECAUTIONARY ONLY AND DOES NOT IMPLY PUBLICATION
** OR DISCLOSURE.
**
** THIS SOFTWARE CONTAINS CONFIDENTIAL INFORMATION AND
** TRADE SECRETS OF MITSUBISHI ELECTRIC ITA.  USE, DISCLOSURE,
** OR REPRODUCTION IS PROHIBITED WITHOUT THE PRIOR
** EXPRESS WRITTEN PERMISSION OF MITSUBISHI ELECTRIC ITA.
**
** OpenMQ
**   Module: qget.c
**   Author: Derek Schwenke 9/8/95
**
**
** Qget() is made from two calls:
**    QlistenBeforeReply() // Get new data
**    QreplyAfterListen() // Return ACK and maybe some data
**   Both call must be made.
**
*/


#include "qlib.h"
   extern lpSMBUFH sm_base;

int Qget(         lpQHANDLE   q,          // Q handle returned by Qopen();
                  lpMSGH      messh,      // Pointer to message header to fill
                  char       *mess,       // Pointer to message buffer to fill
                  int         maxsize){   // size of message buffer

   // Always return 1st error found
   // Always call reply after listen

   int status = QlistenBeforeReply(q,messh,mess,maxsize);

   if (status == QSUCCESS) status = QreplyAfterListen(q,0,0,0,0,0);
   else                             QreplyAfterListen(q,0,0,0,0,0);
   return(status);
}



int QlistenBeforeReply(lpQHANDLE q,       // Q handle returned by Qopen();
                  lpMSGH      messh,      // Pointer to message header to fill
                  char       *mess,       // Pointer to message buffer to fill
                  int         max_size){  // size of message buffer

   int      objn, bnum, rtn, dsize, status = NOT_COMPLETE;
   lpSMBUF  buf;


   if(q == NULL)       Fail("Qlbr not passed handle");
   if(max_size == 0)   Fail("Qlbr not passed any message buffer");
```

```
    // When your Qopen(GETTING) you reserve buffers, so we want
    // to wait on one becoming READY.
    // When it is ready, we copy the data out.

    Diag("Qlbr[-]: waiting for one of %d buffers to become ready...",q->num_bu
    // Wait for a buffer
    rtn = WaitForMultipleObjects(q->num_bufs, q->readyh, FALSE,INFINITE);   //
    objn = rtn - WAIT_OBJECT_0;
    if ((objn < 0) || (objn > q->num_bufs))
       Fail("Qlbr[-]: WaitForMultipleObjects fails [%d]",rtn);
    bnum = q->bufs_found[objn];
    Diag("Qlbr[%d]: WaitForMultipleObjects got object %d mode=%d",bnum,objn,q->ms
    buf = SMBUFADDR(bnum);


    // Read the message out of the buffer
    dsize = __min(max_size, buf->msgh.size);
    memcpy(messh,&buf->msgh,sizeof(MSGH));
    memcpy(mess,&buf->mdata,dsize);
    status = QSUCCESS;


    // Return without notifying the sender i.e. without releaing "done"
    // Qral will be called next to do that

    q->cur_objn = objn; // Save for Qral() to use.
    Diag("Qlbr[%d]: returns",bnum);
    return(status);
}




int QreplyAfterListen( lpQHANDLE q,        // Handle returned by Qopen()
                  int       mode,          // Optional type of message
                  int       sub_mode,      // Optional User defined type of messa
                  char      *mess,         // Reply message to send
                  int       size,          // Size of reply messsage
                  lpMSGH    mh){           // Optional message header

    int      objn, bnum, status=QSUCCESS;
    lpSMBUF  buf;


    if(!q) Fail("Qral: not passed handle");
    if(q->cur_objn == -1) Fail("Qral called out of order");

    // Retrieve handels
    objn = q->cur_objn;
    bnum = q->bufs_found[objn];
    buf = SMBUFADDR(bnum);
    q->cur_objn = -1;
    Diag("Qral[%d] object %d",bnum,objn);


    // Consturct the message header
    buf->msgh.size = size;
    if (mh) {
      buf->msgh.mode = mh->mode;
      buf->msgh.sub_mode = mh->sub_mode;
   // buf->msgh.size = mh->size; // removed 12/27/95 by derek, passed size sho
      buf->msgh.reply_smbuf = mh->reply_smbuf;
   }
   if (mode)     buf->msgh.mode = mode;
   if (sub_mode) buf->msgh.sub_mode = sub_mode;

   // Copy any message data
   if (buf->msgh.size = __min(buf->msgh.size,MAXMSGDATA)) {
      memcpy(&buf->mdata,mess,buf->msgh.size);
   } else
      strcpy(buf->mdata,""); // Clear the whole buffer?


   Diag("Qral[%d]: releases done %d",bnum,objn);
   buf->status = SMBUF_RETURN_MESS;
   if (!ReleaseSemaphore(q->doneh[objn],1,0))
      Warn("Qral[%d]: ReleaseSemaphore done #%d",bnum,GetLastError());

   SHAREDATA(stat.gets)++; // Statistics
   Diag("Qral[%d]: returns(%d)",bnum,status);
   return(status);
}
```

```
////////////////////////////////////////////////////////////////////////////////
//
// Common routines and definitions used by OpenMQ
//
/////////////////////////////////////////////////////////////////////////////

// Size of shared memory
#define LINESIZE            1024
#define NAMESIZE            20
#define ROUTNAME            "c:\\q\\routing.txt"
#define PARMNAME            "c:\\q\\parms.txt"
#define MLOGNAME            "c:\\q\\qlogl.txt"
#define TLOGNAME            "c:\\q\\tranlog.txt"
#define SMEMNAME            "Q/SHAREDMEMORY"
#define SMEMTABLE           "Q/SHAREDTABLE"
#define MUTQOPEN            "Q/QOPEN"
#define MUTFREFMT           "Q/SMB_FREE_%d"
#define SEMRDYFMT           "Q/SMB_READY_%d"
#define SEMDONFMT           "Q/SMB_DONE_%d"
#define SEMACKFMT           "Q/SMB_ACK_%d"
#define CLASSNAMES          "class0 class1 class2 class3"
#define SMBUFADDR(X)        (lpSMBUF) ((int) sm_base + (X * sizeof(SMBUF)) + size
#define MAXSMSIZE(X)        sizeof(SMBUFH) + X * sizeof(SMBUF);
#define BITSET(X,Y)         ((X & Y) == X)
#define MAXMSGDATA          (MAXTXSIZE - sizeof(MSGH))
#define SHAREDATA(X)        (sm_base->X)
#define MAXTXSIZE           8192
#define MAXNSMBUF           32
#define MAXRTSIZE           ((MAXTXSIZE - sizeof(MSGH)) - 10)  // bytes
#define MAXFOLENT           8
#define FD_SETSIZE          64 /* Number of sockets */
//#define RTROOT              (lpRT) ((lpSMBUFH) sm_base)->RT
#define RTROOT              (lpRT) sm_base->RT[sm_base->rt_pingpong]


// Return codes
#define QSUCCESS_FAILOVER   1L
#define QSUCCESS            2L
#define NO_SUCH_QUEUE       11L
#define NO_MORE_SHARED_MEM  12L
#define NO_ACK              13L
#define NOT_COMPLETE        14L
#define WRONG_SMBUF_STATUS  15L
#define QFAIL               16L
#define QTOOBIGMESS         17L
#define NO_QNETD            18L
#define QMODE_MISSMATCH     19L


// Message Modes or message Types
#define PUT_MODE            2L
#define GET_MODE            3L

#define REQUEST_MODE        4L
#define ACK_MODE            8L
#define NACK_MODE           9L

#define COMMIT_MODE         10L
```

```
#define ABORT_MODE          11L

#define ADMINREQ_MODE       20L
#define ADMINREP_MODE       21L

#define QNETDREQ_MODE       30L
#define QNETDREP_MODE       31L


// Message sub_modes used by QNETD
#define QNETD_TRAN_INQ      2L
#define QNETD_RT_BROADCAST 3L
#define QNETD_BUF_STATUS    4L

// Message sub_modes used by QUE servers
#define SUB_MODE_OK         0L
#define SUB_MODE_EMPTY      11L
#define SUB_MODE_FULL       12L
#define SUB_MODE_DISABLED   13L
#define SUB_MODE_INV_INDX   14L     // invalid que entry number
#define SUB_MODE_INV_TID    15L     // invalid TID for pending list search
#define SUB_MODE_BAD_REQ    16L     //


// Transactio inquiry msg modes
#define Q_COMMIT            COMMIT_MODE
#define Q_ABORT             ABORT_MODE
#define Q_INPROGRESS        12L
#define Q_WRONGHOST         13L


// Shared Memory Buffer Status Codes
#define SMBUF_EMPTY         0L
#define SMBUF_SEND_MESS     1L
#define SMBUF_RETURN_MESS   2L
#define SMBUF_RETURN_FAIL   5L
#define SMBUF_GET_MESS      10L
#define SMBUF_GOT_MESS      11L
#define SMBUF_SEND_REMOTE   100L

// Qopen & Qput & Qget flags
#define Q_LOG               (0x00000001L)  // Log this message (not required if T
#define Q_TRACE             (0x00000002L)  // Generate trace entries for this mes
#define Q_ASYNC             (0x00000004L)  // Dont wait for the message to get to
#define Q_FAILOVER          (0x00000008L)  // Enable failover (if its defined in
#define Q_TRAN              (0x00000010L)  // Use two phase commit
#define Q_TRAN_BEGIN        (0x00000030L)  // Start the tran
#define Q_TRAN_END          (0x00000050L)  // End the tran (commit)
#define Q_TRAN_SINGLE       (0x000000F0L)  // Both start and end


// Includes ///////////////////////////////////////////////////////////////////
#include <stdlib.h>         // for exit() call
#include <malloc.h>         // for malloc() call
#include <stdio.h>          // for printf() call
#include <windows.h>        // HANDLE definition
#include <time.h>           // for mid time()
//#include <windowsx.h>     // ListBox_AddString

// Structures /////////////////////////////////////////////////////////////////
```

```
typedef struct mid {        // Message header
   int   host;              // IP of sender
   int   tid;               // incremnets on each tran. 0= not transactional
   int   uid;               // unique ID for each msg entry
} MID, *lpMID;


typedef struct mlist {      // List of MIDs
   struct mlist *next;
   MID    mid;
} MLIST, *lpMLIST;


typedef struct msgh {       // Message header
   int   type_coding;       // Header version, and machine coding format
   short mode;              // Message type (Putting or Getting)
   short sub_mode;          // User defined sub_mode
   int   flags;             // 32 bit message flags
   time_t time;             // Time (set but not used)
   MID   mid;               // Last Unique message id (struct?)
   int   to_node;           // ip target
   int   to_port;           // qnetd port on the target or sm buffer #
   int   to_smbuf;          // Shared memory buffer of receiver
   int   from_smbuf;        // Shared memory buffer of sender i.e. QNETD#1
   int   reply_smbuf;       // Shared memory buffer for reply messages
   char  to_server[NAMESIZE];  // Physical queue server name
   char  to_logical[NAMESIZE]; // Logical queue name
   int   size;              // size of the message
} MSGH, *lpMSGH;

typedef struct qhandle {    // Que Handle
   struct qhandle *next;    // Next handle if in chain
   MSGH  msgh;              // Prototype message header
   time_t open_time;        // Time the open() was made
   int   base_flags;        // Flags to be used in all ops
   short cur_objn;          // Current object (buff) number
   int   time_out;          // minimum time-out on this node
   short rt_rev;            // Rev of the Routing Table that was used
   short num_bufs;          // Number of buffers where service was found
   short bufs_found[64];    // Buffers where the service was found
   HANDLE freeh[64];        // Handles for the "free" mutex for each buffer
   HANDLE readyh[64];       // ??notset? Handles for the "ready" semephore for ea
   HANDLE doneh[64];        // ??notset? Handles for the "done" semephore for eac
} QHANDLE, *lpQHANDLE;

typedef struct smobjs {     // Sync Obj for 1 Shared Memory Buffer
   HANDLE freeh;            // Handle for the "free" mutex for each buffer
   HANDLE readyh;           // Handle for the "ready" semephore for each buffer
   HANDLE doneh;            // Handle for the "done" semephore for each buffer
   HANDLE ackh;             // Handle for the "ack" semephore for each buffer
} SMOBJS, *lpSMOBJS;

typedef  struct smbuf {     // Shared memory buffers
   char  name[NAMESIZE];
   short status;
   short sub_status;        // Currently not used!
   MSGH  msgh;
   char  mdata[MAXMSGDATA];
} SMBUF, *lpSMBUF;
```

```
typedef  struct aps {        // any app specific shared data
   int   app_num;            // this is not accessed by OpenMQ
   int   appdat1;            //
   int   appdat2;            //
   int   appdat3;            //
} APS,  *lpAPS;              //

typedef  struct stat {
   int   opens,openrep,closes;
   int   puts,gets;
   int   tx,rx;
   int   commit,abort;
   int   warn,fail;
} STAT,  *lpSTAT;

typedef  struct fol {
   char  name[NAMESIZE];     // Logical queue name
   int   puts,gets;          // Operations that fail
   int   ip;                 // Addres to fail to
} FOL,  *lpFOL;

/*
typedef struct st {              // QNETD Sockets Table
   struct    st *next;           //
   int       ip;                 //
   int       port;               //
   SOCKET    s;                  //         SOCKET def requires sock.lib
} ST, *lpST;
*/


// Shared memory
// -------------

typedef  struct smbufh {     // Shared memory header (global data area)
   int   tran_id;            // Current unique tran id for this node.
   int   unique_id;          // Current unique message id for this node.
   int   hostip;             // This nodes IP
   short diag;               // Diagnostics On/Off
   short nsbuf;              // Number of shared buffers
   int   time_out;           // minimal timeout on this node
   short failed_servers;     // number of failures detected
   int   time;               // now (watchdog's clock)
   int   start_time;         // time shared mem was initialized
   int   rt_rev;             // rev of last RT update (MINOR)
   int   rt_ver;             // version of last RT received  (MAJOR)
   int   rt_pingpong;        // RT Buffer to use
   int   tmp1;               // not used
   int   tmp2;               // not used
   int   tmp3;               // not used
   APS   ap;                 // not used (used by oentry)
   STAT  stat;               // Statistics
   char  hostname[NAMESIZE]; // Hostname
//       int      sockets                          // Number of active sockets
//    ST      ST[MAXFOLENT];                // Socket table
   FOL   FO[MAXFOLENT];      // Fail over list
   char  RT[2][MAXRTSIZE];       // Routing table
} SMBUFH, *lpSMBUFH;
```

```
// Routing table
// -------------

typedef struct rt {           // Routing Table in shared memory
   int   ip;                  //
   int   qnetd_port;          //
   int   next_offset;         //
   int   ntype_index;         //
   int   apps_index;          //
   char  s[4];                // node\0ntype\0apps\0 <<More than 4 char>>
} RT, *lpRT;

// Global data ///////////////////////////////////////////////////////////////
// C++ does not allow 2nd define of variables.
#ifndef Q_LIB
   #define Q_LIB
   lpSMBUFH  sm_base;             // Base address of shared memory segment
//   lpQHANDLE FLroot;              // Linked list of (any) failed servers
   char      AskAnswer[LINESIZE]; // Defined in print.c
#endif


// Routines /////////////////////////////////////////////////////////////////

#ifdef CPP
extern "C"
{
#endif

void        Say();        // Dont check any parameters
void        Fail();     // Dont check parameters
void        Warn();     // Dont check parameters
void        Diag();     // Dont check parameters
char        *Ask() ;    // Dont check parameters
//void        Mlog(char *msg);

// #define     Diag if (SHAREDATA(diag)) Say Made a call so "if" doesnt mess up

int ReserveSharedBuffer( int  seqno,
                  char        *bname,
                  int         bstatus,
                  lpSMOBJS    sync);

int UnReserveSharedBuffer(int bnum);


lpQHANDLE Qopen(  char        *que_name,  // Queue name (logical)
                  short       mode,       // Predefined Mode (PUT_MODE/GET_MODE)
                  short       sub_mode,   // Userdefined sub_mode
                  int         flags,      // Optional Flags (input)
                  int         *status,    // Optional detailed failure code (out
                  int         buffers,    // Optional number of buffers to make
                  int         index);     // Optional index of server to be open


lpQHANDLE QopenReply(lpQHANDLE q,         // Optional existing q handle to be re
                  lpMSGH      mh,         // Message to reply to
                  int         flags,      // Optional Flags for Qopen()
                  char        *name,      // Optional server name
                  int          *status);   // Optional detailed status code

int QreOpen(      lpQHANDLE q);            // Get new buffer and routing info

int Qclose(       lpQHANDLE    *ppq,       // Optional Queue handle OR
                  char         *bname);    // Optional name must be specifed

int Qput(         lpQHANDLE    q,          // Q handle returned by Qopen();
                  short        mode,       // Optional mode for this put;
                  short        sub_mode,   // Optional sub_mode for this put;
                  int          flags,      // Optional Extra flags for this Put()
                  int          size,       // Number of bytes to transfer
                  char         *buffer);   // Optional Buffer that holds the mess

int QsendAndReceive(lpQHANDLE q,           // Q handle returned by Qopen();
                  int          mode,       // Optional Mode for this put
                  int          submode,    // Optional SubMode for this put

                  int          putflags,   // Optional flags for this Put;
                  int          putsize,    // Number of bytes to transfer
                  char         *putbuf,    // Optional Buffer that holds the mess

                  int          maxgetsize, // Optional size of get buffer
                  char         *getbuf,    // Optional buffer fill with message d
                  int          *gotsize,   // Optional number of bytes filled.
                  lpMSGH       gothead);   // Optional got message header.


int Qget(         lpQHANDLE    q,          // Q handle returned by Qopen();
                  lpMSGH       messh,      // Pointer to message header to fil
                  char        *mess,       // Pointer to message buffer to fill
                  int          maxsize);   // size of message buffer


int QlistenBeforeReply(lpQHANDLE q,        // Q handle returned by Qopen();
                  lpMSGH       messh,      // Pointer to message header to fill
                  char        *mess,       // Pointer to message buffer to fill
                  int          max_size);  // size of message buffer


int QreplyAfterListen( lpQHANDLE q,        // Handle returned by Qopen()
                  int        mode,         // Optional type of message
                  int        sub_mode,     // Optional User defined type of messa
                  char       *mess,        // Reply message to send
                  int        size,         // Size of reply messsage
                  lpMSGH     mh);          // Optional message header


void Qcommit(     int        action);      // Commit or abort

int FindSMBuffers( lpQHANDLE  q,
                   int        targetsip,          // Local or Remote IP
                   char       *physical_name,     // Blank if not used
                   int        target_smbuf);      // -1 if not used

lpSMBUFH    AttachSharedMemory();          // Used by graphic apps

void SetWinPtr(HWND WinPtrX);              // Printing to GUI output
void ForcedBufferReset(int    bnum,        // Optional Buffer number (-1 = not us
                  lpQHANDLE    q,          // Optional Queue Handle
                  int          objn);      // Optional objet number in Queue Hand


#ifdef CPP
}
#endif


#ifdef QLIB   // Routines that only QLIB uses (Not API)


   lpRT ServerByName2(lpRT rtp, char *name, int index, char *physical);


   void     TLadd(lpQHANDLE q);      // From tran.c
   void     SetTID(lpMID m, int flags);
   lpRT     RTByIP(int IP);
   void     PrintRT(lpRT r,char * s);

#endif
```

```
////////////////////////////////////////////////////////////////////////////////
// Qnetd.h
////////////////////////////////////////////////////////////////////////////////

#include <winsock.h>

// Structures /////////////////////////////////////////////////////////////////
typedef struct st {             // QNETD Sockets Table
   struct   st *next;           //
   int      ip;                 //
   int      port;               //
   SOCKET   s;                  //          SOCKET def requires sock.lib
} ST, *lpST;

// Routines ///////////////////////////////////////////////////////////////////
//void     ReadRT();              // QLIB call move it to qnetd?
LPVOID   SharedMemInit(int nsbuf);
DWORD    ListenConn();
DWORD    ListenSMBuff();
DWORD    WatchDog();


int      SendRemoteMess(lpSMBUF buf, int pass);
lpST     AddSocketToTable(SOCKET sock,SOCKADDR_IN *sa);

lpRT     RTByName(char *name);    //QLIB call used by QNETD/tcp.c
int      ResolveTran(lpMID MID); //QLIB call used by QNETD to return admin messa
void     ReadParms(int *nsbuf);
int      Str2Port(char *str);
```

```
/*
**
** Copyright(C)1995 MITSUBISHI ELECTRIC ITA.  ALL RIGHTS RESERVED.
** UNPUBLISHED -- RIGHTS RESERVED UNDER THE COPYRIGHT
** LAWS OF THE UNITED STATES.  USE OF A COPYRIGHT NOTICE
** IS PRECAUTIONARY ONLY AND DOES NOT IMPLY PUBLICATION
** OR DISCLOSURE.
**
** THIS SOFTWARE CONTAINS CONFIDENTIAL INFORMATION AND
** TRADE SECRETS OF MITSUBISHI ELECTRIC ITA.  USE, DISCLOSURE,
** OR REPRODUCTION IS PROHIBITED WITHOUT THE PRIOR
** EXPRESS WRITTEN PERMISSION OF MITSUBISHI ELECTRIC ITA.
**
** OpenMQ
**   Module: qopen.c
**   Author: Derek Schwenke 9/8/95
**
** Qopen/Qclose API calls
*/


#include "qlib.h"
#include "rt.h"
   extern lpSMBUFH sm_base;


int FindSMBuffers(    lpQHANDLE q,
                      int   targetsip,          // Local or Remote IP
                      char  *physical_name,     // If local, name must be set
                      int   target_smbuf) {     // -1 if not used

   int   i = 0;
   int   found = 0;
   char  searchfor[NAMESIZE]; // Buffer name
   char  freen[NAMESIZE];     // Syncronization object name
   char  readyn[NAMESIZE];    // Syncronization object name
   char  donen[NAMESIZE];     // Syncronization object name

   // This routine finds the buffers to use ON THE LOCAL NODE, to
   // communicate. If the targetsip is remote, then it returns
   // the buffers that QNETD will listen to.

   // Messages can be sent to logical services
   // Reply messages are sent to specific smbuffers

   if (SHAREDATA(hostip) == targetsip) { // then its local
      strcpy(searchfor,physical_name);
      if (target_smbuf != -1) i = target_smbuf;
   } else { // its not local
      Diag("The physical server of %s is not local!",physical_name);
      strcpy(searchfor,"QNETD");
   }


   q->num_bufs = 0;
   while (i < SHAREDATA(nsbuf)) {
      lpSMBUF bp = SMBUFADDR(i);
      if ((!strcmp(bp->name,searchfor))||(i==target_smbuf)) { // Found a buffer
         // Get a handle for this buffer
         // Pack the handle into arrays so we can wait on any one of them.
```

```
        sprintf(freen, MUTFREFMT,i);
        sprintf(readyn,SEMRDYFMT,i);
        sprintf(donen, SEMDONFMT,i);
        if (!(q->freeh[q->num_bufs]  = OpenMutex(    SYNCHRONIZE, FALSE, freen)
           Fail("Could not OpenMutex %s Error=%d",freen,GetLastError());
        if (!(q->readyh[q->num_bufs] = OpenSemaphore(SEMAPHORE_MODIFY_STATE|SYN
           Fail("Could not OpenSemaphore %s Error=%d",readyn,GetLastError());
        if (!(q->doneh[q->num_bufs]  = OpenSemaphore(SEMAPHORE_MODIFY_STATE|SYN
           Fail("Could not OpenSemaphore %s Error=%d",donen,GetLastError());
        q->bufs_found[q->num_bufs++] = i;
        Diag(" Buffer %d [%s] found ",q->num_bufs,searchfor);
        if (q->num_bufs == 64) continue; // Stop looking
     }
     if (i++ == target_smbuf) i = SHAREDATA(nsbuf) ; // We are done with just o
  }

  if (q->num_bufs) {
     Diag("Qopen[%s] found %d buffers",searchfor,q->num_bufs);
  } else {
     Warn("Qopen[%s] No such server No buffers were found!",searchfor);
  }

  return(q->num_bufs);
}


lpQHANDLE Qopen( char      *que_name,  // Queue name (logical)
                 short     mode,       // Predefined Mode (PUT_MODE/GET_MODE)
                 short     sub_mode,   // Userdefined sub_mode
                 int       flags,      // Optional Flags (input)
                 int       *status,    // Optional detailed failure code (
                 int       buffers,    // Optional number of buffers to make
                 int       index){     // Optional index of server to be open


 // When called for PUT_MODE, just look for existing SMBUFers using this queue n
 // When called for GET_MODE, reserve a new set of buffers under the  name

  int      rtn, found = 0, stat = QSUCCESS;
  lpQHANDLE q = (lpQHANDLE) malloc(sizeof(QHANDLE));
  lpRT     rt=NULL;
  char     physical_name[NAMESIZE];
  HANDLE   mutex;


  // Attach to shared memory
  if(!sm_base){
     if(!( sm_base = AttachSharedMemory() )) {
        *status = NO_QNETD;
        return(0);
     }
  }

  Diag("Qopen(%s) opens mutex",que_name);

  if (!(mutex = OpenMutex(MUTEX_ALL_ACCESS, FALSE, MUTQOPEN)))         // Ser
     Fail("Qopen(%s): Cant OpenMutex %s %d",que_name,GetLastError()); // Fail

  if ((rtn = WaitForSingleObject(mutex,INFINITE)) != WAIT_OBJECT_0)
     Fail("Qopen(%s): Failed while waiting for mutex %s %d",que_name, &MUTQOPEN
if (!q)
   Fail("Qopen(%s) malloc error",que_name);
// memset(q,0,sizeof(QHANDLE));

// Find this queue in the RT  // If gettting look for servers on this node on


if (mode == GET_MODE) {
   q->msgh.to_node       = SHAREDATA(hostip);
   q->msgh.to_port       = 0;
   strcpy(physical_name,que_name);
   // rt = ServerByName2(RTByIP(SHAREDATA(hostip)),que_name,index,physical_na
   //
} else {  // PUT_MODE
   rt = ServerByName2(RTROOT,que_name,index,physical_name);
   if (!rt) // It's not in the routing table but check the local node anyway
      if (FindSMBuffers(q,SHAREDATA(hostip),que_name,-1)) {
         rt = RTByIP(SHAREDATA(hostip)); // Return this node
         strcpy(physical_name,que_name);
      }
}


// Assign values from RT into handle
if (rt || mode == GET_MODE) {
   if (rt) {
      q->msgh.to_node    = rt->ip;
      q->msgh.to_port    = rt->qnetd_port;
      strcpy(q->msgh.to_server,physical_name);
   }

   q->msgh.type_coding  = 1;
   q->msgh.flags        = 0;
   q->msgh.mode         = mode;
   q->msgh.sub_mode     = sub_mode;
   q->msgh.mid.host     = SHAREDATA(hostip);
   q->msgh.mid.tid      = 0;
   q->msgh.time         = 0;
   q->msgh.to_smbuf     = -1;
   strcpy(q->msgh.to_logical,que_name);
   q->msgh.from_smbuf   = 0;
   q->msgh.size         = 0;
   q->base_flags        = flags; // Flags used for all future access
   q->cur_objn          = -1;    // SMBUF number being used.
   q->num_bufs          = 0;     // number of buffers found.
   q->next              = NULL;  // Pointer to chain of handles
   q->open_time         = SHAREDATA(time);       //time(&q->open_time);


   if (mode == GET_MODE) {
      q->time_out       = INFINITE;  // 0x7FFFFFFF; // 24 days
   } else {
      q->time_out       = SHAREDATA(time_out);
   }


   if (mode == PUT_MODE){
      // Dont allocate, just find a list of existing buffers
      if( ! FindSMBuffers(q,q->msgh.to_node,physical_name,-1) )
```

```
            stat = NO_SUCH_QUEUE;
        } else if (mode == GET_MODE){
          // make a set of new buffers
          SMOBJS sync;
          int i,b, seqno = 0;
          int dontcheck = (buffers == -1); // Dont check for stale existing bu..e

          // Reserve Shared buffers

          if (buffers == -1)        seqno = 999;
          if (buffers < 1)          buffers = 1;
          if (buffers > SHAREDATA(nsbuf) - 1)  buffers = SHAREDATA(nsbuf) - 1;

          Diag("Qopen(%s) for GET_MODE: reserving %d buffers",que_name,buffers);
          for ( i = 0 ; i < buffers ; i++ ){
            if (-1 != (b = ReserveSharedBuffer(
              seqno++, physical_name, SMBUF_EMPTY, &sync) )) {
              q->bufs_found[i] = b;
              q->num_bufs++;
              q->freeh[i]  = sync.freeh;
              q->readyh[i] = sync.readyh;
              q->doneh[i]  = sync.doneh;
            } else {
              Say("Qopen(%s) for GET_MODE: Failed, got %d of %d buffers",que_na
              if (i==0) stat = NO_MORE_SHARED_MEM;
              break;
            }
          }
          Diag("Qopen(%s) for GET_MODE: reserved %d buffers",que_name,buffers);
        } // end of PUTTING or GETTING

    } else {  // No rt route was found
        PrintRT(RTROOT,"");
        stat = NO_SUCH_QUEUE;
        Warn("Qopen(%s) NO_SUCH_QUEUE",que_name);

    }

    if (stat != QSUCCESS) {
        free(q);
        q = NULL;
    }

    ReleaseMutex(mutex); // Mutex on all QOpen() calls

    if (status) *status = stat;
    return(q);
}


int Qclose(        lpQHANDLE   *ppq,        // Optional Queue handle OR
                   char        *bname){     // Optional name must be specifed
    lpQHANDLE qp;
    int i;

    // Attach to shared memory
    if(!sm_base){
      if(!( sm_base = AttachSharedMemory() ))
        return(NO_QNETD);
    }
```

```
   if (bname){
      Diag("Qclose(%s) by name",bname);
      for (i = 0; i < SHAREDATA(nsbuf) ; i++) // All buffers
          if (!strcmp((SMBUFADDR(i))->name,bname))
            UnReserveSharedBuffer(i);

   } else if (ppq) {
      if (qp = *ppq) {

         if (qp->msgh.mode == GET_MODE) {  // Release any buffers .. perhaps num

            Diag("Qclose(%s) by handel",(SMBUFADDR(qp->bufs_found[0]))->name);

            for (i = 0; i < qp->num_bufs ; i++)
               if (UnReserveSharedBuffer(qp->bufs_found[i]));
         }
         free(qp);
         *ppq = NULL;
      }
   }
   return(QSUCCESS);
}



// Used for putting messages back to the sender (use if the sender is listeni

lpQHANDLE QopenReply(lpQHANDLE q,          // Optional existing q handle to be re
                  lpMSGH      mh,          // Message to reply to
                  int         flags,       // Optional Flags for Qopen()
                  char        *name,       // Optional server name
                  int         *status){    // Optional detailed status code

   int stat = QSUCCESS, bnum;


   if(!sm_base){ // This cant happen. you must Qopen first.
      if(!( sm_base = AttachSharedMemory() ))
         if (status) *status = NO_QNETD;
         return(0);
   }

   if (!q) {
      q = (lpQHANDLE) malloc(sizeof(QHANDLE));
      if (!q) Fail("QopenReply Malloc error");
      memset(q,0,sizeof(QHANDLE));
   }

   if (mh) {
      // Construct que handle from messge header

      q->msgh.type_coding   = 1;
      q->msgh.mode          = PUT_MODE;
      q->msgh.sub_mode      = 0;
      q->msgh.flags         = flags;
      q->msgh.mid.host      = SHAREDATA(hostip);
      q->msgh.mid.tid        = 0;
      q->msgh.time           = 0;
      q->msgh.to_node        = mh->mid.host;
      q->msgh.to_port        = 0;
      q->msgh.to_smbuf       = mh->reply_smbuf;
      if (name) {
         strcpy(q->msgh.to_server,name);
         strcpy(q->msgh.to_logical,name);
      }
      q->msgh.from_smbuf     = 0;
      q->msgh.reply_smbuf    = 0;
      q->msgh.size           = 0;
      q->base_flags          = flags;
      q->cur_objn            = -1;
      q->num_bufs            = 0;
      q->next                = NULL;  // Pointer to chain of handles
      q->open_time           = SHAREDATA(time);
      q->time_out            = SHAREDATA(time_out);

      //time(&q->open_time);


// Get a list of buffers for this service
      bnum = mh->reply_smbuf;
      if (name)
         if (!strcmp(name,"QNETD")) bnum = 0;

      if( ! FindSMBuffers(q,mh->mid.host,"",bnum))
         stat = NO_SUCH_QUEUE;

   } else {
      Say("QopenReply() was not passed a message header");
      stat = NO_SUCH_QUEUE;
   }

   if (status) *status = stat;
   if (stat == QSUCCESS) return(q);
   free(q);
   return(0);
}

int QreOpen(      lpQHANDLE q){       // Existing q handle to be replaced
   lpRT     rt=NULL;

   if (!q)  return(NO_SUCH_QUEUE);

   if (q->msgh.mode == GET_MODE) return(QMODE_MISSMATCH);

   rt = RTByService(q);
   if (!rt) { // It's not in the routing table but check the local node anyway
      if (SMByName(q->msgh.to_logical)) {
         rt = RTByIP(SHAREDATA(hostip)); // Return this node
         strcpy(q->msgh.to_server,q->msgh.to_logical);
      }
   }

   // Assign values from RT into handle
   if (rt) {
      q->msgh.to_node    = rt->ip;
      q->msgh.to_port    = rt->qnetd_port;
   }

// Get a list of buffers for this service
   if( ! FindSMBuffers(q,q->msgh.to_node,q->msgh.to_server,-1))  // q->msgh.to.s
      return(NO_SUCH_QUEUE);

   return(0);
}



int ApClose( lpQHANDLE ap ){
   free(ap);
   return(QSUCCESS);
}
```

```
/*
**
** Copyright(C)1995 MITSUBISHI ELECTRIC ITA.  ALL RIGHTS RESERVED.
** UNPUBLISHED -- RIGHTS RESERVED UNDER THE COPYRIGHT
** LAWS OF THE UNITED STATES.  USE OF A COPYRIGHT NOTICE
** IS PRECAUTIONARY ONLY AND DOES NOT IMPLY PUBLICATION
** OR DISCLOSURE.
**
** THIS SOFTWARE CONTAINS CONFIDENTIAL INFORMATION AND
** TRADE SECRETS OF MITSUBISHI ELECTRIC ITA.  USE, DISCLOSURE,
** OR REPRODUCTION IS PROHIBITED WITHOUT THE PRIOR
** EXPRESS WRITTEN PERMISSION OF MITSUBISHI ELECTRIC ITA.
**
** OpenMQ
**    Module: qprint.c
**    Author: Derek Schwenke 9/8/95
**
**  Printing routines used by OpenMQ
*/

#include "qlib.h"
#include <windowsx.h>

//#define Diag(X) if(diag) {Say(X);}

// Global Data ////////////////////////////////////////////////////////////////
   FILE *MLOG = NULL;   // Message Log File
// HANDLE *MLOG = NULL;   // Message Log File
   HWND *LogWin;         // Global data - Handle for printing to window
   int  LogWinIndex=0; // Index of last lines written to the logwin ;
//   extern int diag;

void SetWinPtr(HWND WinPtr){
   LogWin = WinPtr;
   ListBox_ResetContent(LogWin);
   ListBox_SetCurSel(LogWin,LogWinIndex++);
}
// Routines ///////////////////////////////////////////////////////////////////

void Mlog(char *msg) {  // Log to a file
//   int len,i,j;
//   char line[LINESIZE];

#ifdef GUI
   //char line2[LINESIZE];
   //TRACE(msg); // ListBox_InsertString
     ListBox_AddString(LogWin,msg);
     ListBox_SetCurSel(LogWin,LogWinIndex++);
#endif

   if ( ! MLOG ) {
      if ((MLOG = fopen(MLOGNAME,"a")) == NULL )

/*       if ( NULL == ( MLOG = CreateFile( MLOGNAME ,
            GENERIC_WRITE|GENERIC_READ,
            FILE_SHARE_WRITE|FILE_SHARE_READ, 0,
            OPEN_ALWAYS,
            FILE_ATTRIBUTE_NORMAL, 0)))      //FILE_FLAG_WRITE_THROUGH
  */
         printf("ERROR CAN NOT OPEN LOG FILE %s",&MLOGNAME);
         return;
   }
   fprintf(MLOG,"%s\n",msg);
/*
   len = strlen(msg); // Make each \n into a \r\n
   for (i=j=0;i<len;i++,j++){
      if (msg[i] == '\n')
         line[j++] = '\r';
      line[j] = msg[i];
   }

   sprintf(line,"%s\r\n",msg);
   WriteFile(MLOG,line,strlen(line),&len,NULL);
*/
}


void Say(   char  *msg,
            void  *a1,
            void  *a2,
            void  *a3,
            void  *a4,
            void  *a5,
            void  *a6) {
   char line[LINESIZE];
   sprintf(line,msg,a1,a2,a3,a4,a5,a6);
#ifndef GUI
   printf("%s\n",line);
#endif
   Mlog(line);
}


// char AskAnswer[LINESIZE];
char * Ask( char *msg,
            void *a1,
            void *a2,
            void *a3,
            void *a4,
            void *a5,
            void *a6) {
   char line[LINESIZE];
   sprintf(line,msg,a1,a2,a3,a4,a5,a6);
   strcat(line," >");

#ifndef GUI
   printf("%s",line);
#endif
   Mlog(line);
#ifndef GUI
   gets(AskAnswer);
#endif
   Mlog(AskAnswer);
   return(AskAnswer);
}


void Diag(  char   *msg,
            void   *a1,
            void   *a2,
            void   *a3,
            void   *a4,
            void   *a5,
            void   *a6) {
   if (SHAREDATA(diag))
      Say(msg,a1,a2,a3,a4,a5,a6);
}

void Warn(  char   *msg,
            void   *a1,
            void   *a2,
            void   *a3,
            void   *a4,
            void   *a5,
            void   *a6) {
   char line[LINESIZE];
   if(sm_base) SHAREDATA(stat.warn)++;

   sprintf(line,"\nWARNING: %s\n",msg);
   Say(line,a1,a2,a3,a4,a5,a6);
}

void Fail(  char   *msg,
            void   *a1,
            void   *a2,
            void   *a3,
            void   *a4,
            void   *a5,
            void   *a6) {
   char line[LINESIZE];
   char mbline[LINESIZE];
   int r=0;
   if(sm_base) SHAREDATA(stat.fail)++;
   sprintf(line,"\nError: %s\n",msg);
   Say(line,a1,a2,a3,a4,a5,a6);

   sprintf(line,"\nError: %s\nClick 'Retry' or 'Ignore' to continue",msg);
   sprintf(mbline,line,a1,a2,a3,a4,a5,a6);

   r = MessageBox(NULL,mbline,"FATAL ERROR DETECTED",MB_ICONSTOP|MB_ABORTRETRYIG
   if ( r == IDABORT    ) {
      Say("Bye\n",0,0,0,0,0,0);
      ExitThread(0);
   } else
      Say("CONTINUES AFTER FATAL ERROR",0,0,0,0,0,0);


/*
   Ask("Type any character to continue or return to exit",0,0,0,0,0,0);
   if (strlen(AskAnswer)) {
      Say("CONTINUES AFTER FATAL ERROR",0,0,0,0,0,0);
   } else {
      Say("Bye\n",0,0,0,0,0,0);
//#ifndef GUI
      // exit(0);
      ExitThread(0); // Try this
//#endif
   }
*/
}
```

```
/*
**
** Copyright(C)1995 MITSUBISHI ELECTRIC ITA.  ALL RIGHTS RESERVED.
** UNPUBLISHED -- RIGHTS RESERVED UNDER THE COPYRIGHT
** LAWS OF THE UNITED STATES.  USE OF A COPYRIGHT NOTICE
** IS PRECAUTIONARY ONLY AND DOES NOT IMPLY PUBLICATION
** OR DISCLOSURE.
**
** THIS SOFTWARE CONTAINS CONFIDENTIAL INFORMATION AND
** TRADE SECRETS OF MITSUBISHI ELECTRIC ITA.  USE, DISCLOSURE,
** OR REPRODUCTION IS PROHIBITED WITHOUT THE PRIOR
** EXPRESS WRITTEN PERMISSION OF MITSUBISHI ELECTRIC ITA.
**
** OpenMQ
**   Module: qput.c
**   Author: Derek Schwenke 9/8/95
*/


#include "qlib.h"
#include "rt.h"
extern lpSMBUFH sm_base;
//extern lpQHANDLE FLroot;

////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////

int Qput(   lpQHANDLE    q,           // Q handle returned by Qopen();
            short        mode,        // Optional mode for this put;
            short        sub_mode,    // Optional sub_mode for this put;
            int          flags,       // Extra flags for this Put();
            int          size,        // Number of bytes to transfer
            char         *buffer) {   // Buffer that holds the messages.

   return(QsendAndReceive(q,mode,sub_mode,flags,size,buffer,0,0,0,0));
}

int QsendAndReceive(lpQHANDLE q,      // Q handle returned by Qopen();
            int          mode,        // Optional Mode for this put
            int          sub_mode,    // Optional SubMode for this put
            int          putflags,    // Optional flags for this Put;
            int          putsize,     // Number of bytes to transfer
            char         *putbuf,     // Buffer that holds the messages.
            int          maxgetsize,  // Optional size of get buffer
            char         *getbuf,     // Optional buffer fill with message data
            int          *gotsize,    // Optional number of bytes filled.
            lpMSGH       gothead) {   // Optional got message header.
   int      objn, bnum, rtn, status = NOT_COMPLETE, maxpass, pass = 0,ms;
   lpSMBUF  buf;


   if (q == NULL) {
      Warn("Qsar[-]: passed NULL handle");
      return(NO_SUCH_QUEUE);
   }

   if (q->num_bufs == 0) {
      if (QreOpen(q) != 0) {
         Warn("Qsar[-]: passed bad handle, no buffers after QreOpen()");
         return(NO_SUCH_QUEUE);
   }
}

if(putsize > MAXMSGDATA) {
   Warn("Qsar[-]: Message data too big %d",putsize);
   return(QTOOBIGMESS);
}


q->msgh.flags = q->base_flags | putflags;

if (q->msgh.flags & Q_FAILOVER)  maxpass = 3;
else                             maxpass = 1;


while(pass < maxpass) {   // loop "contine" if error "break" if good

   if (pass) MarkFailedServer(q);

   if (q->rt_rev < SHAREDATA(rt_rev))  UpdateHandle(q);

   pass++;


   // Wait for a buffer
   Diag("Qsar[-]: waiting for one of %d buffers to become free...",q->num_buf
   if (putflags) Diag("Qsar[-]: flags = %p",putflags);
   rtn = WaitForMultipleObjects(q->num_bufs,q->freeh,FALSE,q->time_out-500);

   objn = rtn - WAIT_OBJECT_0;
   if ((objn < 0 )||( objn > q->num_bufs)){
      int abn = rtn - WAIT_ABANDONED_0;
      if (q->num_bufs == 0) {
         Warn("Qsar[-]: Bad handle: has no buffers");
         continue;
      } else if (rtn == WAIT_TIMEOUT) {
         Warn("Qsar[-]: timeout");
         continue;
      } else if ( (abn >= 0 ) && (abn < q->num_bufs) ) {
         Warn("Qsar[-]: Waiting on abandoned obj %d",abn);
         continue;
      }
      Fail("Qsar[-]: WaitForMultipleObjects fails [%d] on %d objects",rtn,q->
      continue;
      // Should report some error here.
   }
   bnum = q->bufs_found[objn];
   buf = SMBUFADDR(bnum);
   // Diag("Qsar[%d=%s]: WaitForMultipleObjects got object %d",bnum,buf->name

   // Safty check the identy of the buffer
   if(*q->msgh.to_server) // if you know the name ("reply msgs only know the
      if(strcmp(q->msgh.to_server,buf->name))
         if( 0 != strncmp(buf->name,"QNETD",4)) {
            Warn("Qsar[%d=%s]: Stale handle. Wanted [%s] got [%s]",
               bnum,buf->name,q->msgh.to_server,buf->name);

            ForcedBufferReset(-1,q,objn);
            // Maybe re-open here?
            QreOpen(q);
            continue;
        }

        // Transfer the data to the buffer
        memcpy(&buf->msgh,&q->msgh,sizeof(MSGH));
        memcpy(&buf->mdata,putbuf,putsize);
        buf->msgh.size = putsize;
//      buf->msgh.flags = buf->msgh.flags | putflags;
        buf->status = SMBUF_SEND_MESS;

        SetTID(&buf->msgh.mid,buf->msgh.flags); // Set uid and not time()
        buf->msgh.time = SHAREDATA(time);


        if (mode)      buf->msgh.mode     = mode;     // Else use default from Qope
        if (sub_mode) buf->msgh.sub_mode = sub_mode; // Else use default from Qope

        // Notify the reciver
        Diag("Qsar[%d=%s]: ReleaseSemaphore ready object %d starts mode=%d",
           bnum,buf->name,objn,buf->msgh.mode);
        if (!ReleaseSemaphore(q->readyh[objn],1,NULL)) {
           Warn("Qsar[%d=%s]: ReleaseSemaphore ready failed handle %p #%d",
              bnum,buf->name,q->readyh[objn],GetLastError());

           ForcedBufferReset(-1,q,objn);

           status = QFAIL; // How to recover here?
        }

        // Wait for an answer
        if (buf->msgh.to_node == SHAREDATA(hostip))  //leave time for remote to
           ms = q->time_out;  // Local should be faster
        else
           ms = q->time_out + (q->time_out)/4;  // Remote is slower



        Diag("Qsar[%d=%s]: Waiting for done object %d starts %d ms",
           bnum,buf->name, objn,ms);
        if ((rtn = WaitForSingleObject(q->doneh[objn],ms)) != WAIT_OBJECT_0 ) {
           if (rtn == WAIT_TIMEOUT)
              Warn("Qsar[%d=%s]: WaitForSingleObject WAIT_TIMEOUT after %d ms",bnu
           else
              Warn("Qsar[%d=%s]: WaitForSingleObject GetLastError = %d",bnum,buf->
           status = QFAIL; // Add new desciptive status code here?

           ForcedBufferReset(-1,q,objn);

           continue;
        }


        // Check the answer
        if(buf->status != SMBUF_RETURN_MESS) {
           Warn("Qsar[%d=%s]: GOT A BAD BUFFER STATUS [%d]",bnum,buf->name,buf->st

           if (buf->status == SMBUF_RETURN_FAIL)
              status = QFAIL;
           else
              status = WRONG_SMBUF_STATUS;
```

```
         ForcedBufferReset(-1,q,objn);

         continue;
      }

      if (pass == 1) status = QSUCCESS;
      else           status = QSUCCESS_FAILOVER;

      // Get the reply's data
      if (getbuf && maxgetsize) {
         int gotsz = __min(maxgetsize,buf->msgh.size);
         if (gotsize)  *gotsize = gotsz;
         memcpy(getbuf,&buf->mdata,gotsz);
      }

      if (gothead) { // if the caller wants to see the header, copy it.
         memcpy(gothead,&buf->msgh,sizeof(MSGH));
      }

      // release the buffer
      if(!ReleaseMutex(q->freeh[objn]))
         Warn("Qsar[%d=%s]: ReleaseMutex object %d failed #%d",
            bnum,buf->name,objn,GetLastError());



      // If it's transactional, add to the open transaction list
      // Perhaps the single tran case can be optimised not to reget the buffer?
      if(putflags & Q_TRAN)
         TLadd(q);

      // If it's the end of a transaction, then COMMIT
      if ((putflags & Q_TRAN_END) == Q_TRAN_END)
         Qcommit(Q_COMMIT); // API call

      break; // We got this far, so it must have not failed.
   } // Loop 3 attempts to send

   if (status <= QSUCCESS) SHAREDATA(stat.puts)++;
   return(status);
}
```

```
/*
**
** Copyright(C)1995 MITSUBISHI ELECTRIC ITA.  ALL RIGHTS RESERVED.
** UNPUBLISHED -- RIGHTS RESERVED UNDER THE COPYRIGHT
** LAWS OF THE UNITED STATES.  USE OF A COPYRIGHT NOTICE
** IS PRECAUTIONARY ONLY AND DOES NOT IMPLY PUBLICATION
** OR DISCLOSURE.
**
** THIS SOFTWARE CONTAINS CONFIDENTIAL INFORMATION AND
** TRADE SECRETS OF MITSUBISHI ELECTRIC ITA.  USE, DISCLOSURE,
** OR REPRODUCTION IS PROHIBITED WITHOUT THE PRIOR
** EXPRESS WRITTEN PERMISSION OF MITSUBISHI ELECTRIC ITA.
**
** OpenMQ
**   Module: routab.c
**   Author: Derek Schwenke 9/8/95
**
** Routing table funtions
*/


#include "qlib.h"
#include "qnetd.h"
#include "rt.h"

#include <string.h>
#include <ctype.h>



int str2ip(char *str){
   int ip1,ip2,ip3,ip4;
   if ( 4 != sscanf(str,"%d.%d.%d.%d",&ip1,&ip2,&ip3,&ip4))    // Perhaps inet_ad
      return(0);
   else
      return((ip1 * 0x1000000) + (ip2 * 0x10000) + (ip3 * 0x100) + (ip4));}

int str2port(char *str){
   int port=0;
   if (sscanf(str,"%*[^\[][%d",&port))
      Say("str2port %s returns %d",str,port);
   else
      Say("str2port %s FAILED",str);
   return(port);
}


// Routing table entry,,,, to be deleted
typedef struct rte {          //
   int   ip;                  //
   char  node[NAMESIZE];      //
   char  ntype[NAMESIZE];     //
   char  cs[NAMESIZE];        //
   int   qnetd_port;          //
   char  serv_class[NAMESIZE];//
   char  servers[4*NAMESIZE]; //
   char  physical[NAMESIZE];  //
   char  logicals[4*NAMESIZE];//
   char  serv_path[NAMESIZE]; //
   char  serv_opts[NAMESIZE]; //
```

```
} RTE, *lpRTE;



static int is_ip(char *s) {
   return( isdigit(s[0]) );
}

static int is_stype(char *s) {
   return( strlen(s) > 3 && strstr(&CLASSNAMES,s) );
}

// The table looks like this:
//137.203.1.1   mars            NT3.5           Server[100]
// class0               router        c:/q/route      -master
// class1               QS1[Q1,Q2,Q3]  c:/q/qserv     -threads 66 -more_opts
//
// The node line is:
// IP# node_name node_type Client_or_server[QNETD address] Preferred_servers,,,
//
// The Services lines is:
// type physical_name[logical_names,,,] path_to_executable options

int parseRTnode(  char  *line,
                  int   line_no,
                  int   *ip,
                  char  *node,
                  char  *ntype,
                  char  *cs,
                  int   *port,
                  char  *servers) {
   char ip_str[NAMESIZE];
   char cs_str[LINESIZE];

   strcpy(servers,",");
   if ( 4 > sscanf(line,"%s %s %s %s %s",ip_str,node,ntype,cs_str,&servers[1]))
      Fail("Routing Table Line %d is not complete: %s",line_no,line);

   if (strlen(servers) > 1)
      strcat(servers,",");
   else
      strcpy(servers,"");

   if (!(*ip = str2ip(ip_str)))
      Fail("Routing Table Line %d ip does not scan: %s",line_no,line);

   if (2 != sscanf(cs_str,"%[^\[][%d]",cs,port))
      Fail("Routing Table Line %d QNETD client/server does not scan: %s",line_no

   return(1);
}

int parseRTservices(char  *line,
                  int   line_no,
                  char  *class,
                  char  *physical_name,
                  char  *logical_names,
                  char  *path,
                  char  *opts) {
```

```
   char services[LINESIZE];
   int len;
   strcpy(opts,"");
   if ( 3 > sscanf(line,"%s %s %s %[^]",class,services,path,opts))
      Fail("Routing Table: line %d is not complete: %s",line_no,line);
   strcpy(logical_names,",");
   sscanf(services,"%[^\[] [%s]",physical_name,&logical_names[1]);
   if((len = strlen(logical_names)) < 2) // No logical name
      sprintf(logical_names,",%s,",physical_name);
   else
      logical_names[--len] = ',';

   return(1);
}

/*
**
**char * NewStr(char * s) {
**   char * n = malloc(strlen(s)+1);
**   if (!n) Fail("Malloc error in NewStr");
**   return(strcpy(n,s));
**}
**
**
**void FreeNcopy(char ** o, char * s) {
**   char * n = malloc(strlen(s)+1);
**   if (!n) Fail("Malloc error in FreeNcopy");
**   strcpy(n,s);
**   free(*o);
**   *o = n;
**}
**
*/


void SwapRT(lpRT RT1, lpRT RT2) {
   lpRT P,R,PRT1=0,PRT2=0;

   Diag("SwapRT(%s,%s)",RT_NODE(RT1),RT_NODE(RT2));

   if (RT1 == RT2) return;

   // Scan for the pre1 and pre2 nodes
   P = RTROOT;
   while (R = NextRT(P)) { // Find both the previous nodes
      if (R == RT1) PRT1 = P;
      if (R == RT2) PRT2 = P;
      P = R;
   }
   if (!(PRT1 && PRT2)) return;
   // Swap pointers
   PRT1->next_offset = RT2C(RT2) - RT2C(PRT1);
   if (RT2->next_offset) PRT2->next_offset = RT2C(RT_NEXT(RT2)) - RT2C(PRT2);
   else                  PRT2->next_offset = 0;
   RT2->next_offset = RT2C(RT1) - RT2C(RT2);

}

int fgetsh(char *line,int max,HANDLE rth) {  // Aproximate fgets()
   int rd=1,rdn=0;
   while ((rdn < max) && ReadFile(rth,line,1,&rd,0)) {
      if (rd != 1) break;
      if (*line == '\n') break;
      line++; rdn++;
   }
   if (*line = '\n') line--;
   if (rd) line++;
   *line = 0;
   return(rdn);
}

void ReadRT() {    // Should only be called by QNETD.
   int    line_no = 0;
   char   line[LINESIZE];
   char   first[LINESIZE];
   char   preferred[LINESIZE];
   //FILE  *fp = fopen(ROUTNAME,"r");
   HANDLE rth = CreateFile( ROUTNAME ,
                      GENERIC_WRITE|GENERIC_READ,
                      FILE_SHARE_WRITE|FILE_SHARE_READ, 0,
                      OPEN_ALWAYS,
                      FILE_ATTRIBUTE_NORMAL, 0);
   RTE   rte;   // Current RTE
   lpRT  rt,rts;
   char *cp;


   if (SHAREDATA(rt_pingpong)) // Not the active RT
      rts = rt = (lpRT) sm_base->RT[0];
   else
      rts = rt = (lpRT) sm_base->RT[1];


   memset(rt,0,MAXRTSIZE);                  // Clear routing table (not required)
   cp = rt->s + 1;                          // 1st node is always nop
   rt->ntype_index = cp++ - (char *) rt->s; // 1st node is always nop
   rt->apps_index  = cp++ - (char *) rt->s; // 1st node is always nop

   strcpy(preferred,"");

   if (! rth )
      Fail("Cant open the Routing Table %s",&ROUTNAME);

   while (fgetsh(line,LINESIZE-1,rth)) {
      line_no++;
      sscanf(line,"%s",first);
      if (*first == '#') {
         line_no = line_no;
      } else if (is_ip(first)) {    // New node //
         // Perhaps we can filter other clinets out here to save space //
         parseRTnode(line,line_no,&rte.ip,rte.node,
                     rte.ntype,rte.cs,
                     &rte.qnetd_port,rte.servers);

         // Start a new node in the RT
         RT_NOFFSET(rt) = ++cp - RT2C(rt);
         rt = NextRT(rt);
         // Build RT in shared memory

         RT_IP(rt) = rte.ip;             // IP number
         RT_PORT(rt) = rte.qnetd_port;     // Port number
         RT_NOFFSET(rt) = 0;               // End of the chain
         strcpy(RT_NODE(rt),rte.node);     // Node string
         rt->ntype_index = strlen(RT_NODE(rt)) + 1;
         strcpy(RT_NTYPE(rt),rte.ntype);
         rt->apps_index = strlen(RT_NTYPE(rt)) + 1 + rt->ntype_index;
         cp = RT_APPS(rt);
         *cp = 0;


         if (!strcmp(SHAREDATA(hostname),RT_NODE(rt))) {
            strcpy(preferred,rte.servers);
            SHAREDATA(hostip) = RT_IP(rt);
            Diag("ParseRT this node is %s ip=%p",SHAREDATA(hostname),SHAREDATA(h
         }

      } else if (is_stype(first)) { // New service //
         parseRTservices(line,line_no,rte.serv_class,
                         rte.physical,rte.logicals,
                         rte.serv_path,rte.serv_opts);
      // format: [QS1],Q1,Q2,[QS2],Q4,
         sprintf(cp,"[%s]%s",rte.physical,rte.logicals);
         cp = strchr(cp,0);                // Advance the pointer

      } else if (!strcmp(first,"RT_VERSION")) {
         sscanf(line,"%*s %d",&SHAREDATA(rt_ver));
      }
   }
//    if (ferror(rth)){
//       Fail("ParseRT read error on %s.\n",&ROUTNAME);
//       clearerr(rth);
//    }
   CloseHandle(rth);

   // Sort this list according to the preferred servers
   PrintRT(rts,"Before sorting");

   if (strlen(preferred) > 2) {
      lpRT rtf,rtp = NextRT(rts);
      char *e,*p = preferred + 1;
      Diag("ReadRT: Sorting preferred servers:%s",preferred);
      // Swap entries to achive the right order
      while (rtp && ((strlen(p) > 2))) {
         if (e = strchr(p,',')) *e = 0;
         rtf = rts;
         while (rtf = NextRT(rtf)) if (!strcmp(RT_NODE(rtf),p)) break;
         if (!rtf) {
            Say("ReadRT: Cant find preferred server %s",p);
            break;
         }
         SwapRT(rtp,rtf);
         rtp = NextRT(rtp);

         if (e)    p = e + 1;
         else      break;

      }
      PrintRT(rts,"After sorting");
   }
   // Ping Pong the RT tables, make this one active
   if (SHAREDATA(rt_pingpong))
      SHAREDATA(rt_pingpong) = 0;
   else
      SHAREDATA(rt_pingpong) = 1;
   SHAREDATA(rt_rev)++;  // Every handle will re-load from new RT.

   // We must find some way to set the IP addr, and the RT is the only way to ge
   if(!SHAREDATA(hostip))
      Warn("ReadRT: could not find host %s in the file %s",SHAREDATA(hostname),&
}
```

```
/*
**
** Copyright(C)1995 MITSUBISHI ELECTRIC ITA.  ALL RIGHTS RESERVED.
** UNPUBLISHED -- RIGHTS RESERVED UNDER THE COPYRIGHT
** LAWS OF THE UNITED STATES.  USE OF A COPYRIGHT NOTICE
** IS PRECAUTIONARY ONLY AND DOES NOT IMPLY PUBLICATION
** OR DISCLOSURE.
**
** THIS SOFTWARE CONTAINS CONFIDENTIAL INFORMATION AND
** TRADE SECRETS OF MITSUBISHI ELECTRIC ITA.  USE, DISCLOSURE,
** OR REPRODUCTION IS PROHIBITED WITHOUT THE PRIOR
** EXPRESS WRITTEN PERMISSION OF MITSUBISHI ELECTRIC ITA.
**
** OpenMQ
**    Module: routab.c
**    Author: Derek Schwenke 9/8/95
**
** Routing table funtions
*/

#include "qlib.h"
#include "rt.h"
#include "netadmin.h"

#include <string.h>
#include <ctype.h>


lpRT NextRT(lpRT rt){
   if (rt->next_offset)
      return(RT_NEXT(rt));
   return(0);
}


void PrintRT(lpRT r,char * s) {
   Diag("\n\n\t\tRouting Table %s\n",s);
   while(r = NextRT(r)) {
      Diag("%8s %7s %p[%4d] %s",RT_NODE(r),RT_NTYPE(r),
         r->ip,r->qnetd_port,RT_APPS(r));
   } Diag("");
}


lpRT ServerByNode(char *node) {
   lpRT  rt = RTROOT;

   while (rt = NextRT(rt)) {
      if (!strcmp(RT_NODE(rt),node)) break;
   }
   return(rt);
}


lpRT RTByName(char *name) {
   lpRT  rt = RTROOT;

   while (rt = NextRT(rt)) {
      if (!strcmp(RT_NODE(rt),name)) break;
   }
   return(rt);
}

lpRT RTByIP(int IP) {
   lpRT  rt = RTROOT;

   while (rt = NextRT(rt)) {
      if (IP == rt->ip) break;
   }
   return(rt);
}

int Str2IP(char *str){
   int ip1,ip2,ip3,ip4;
   if ( 4 != sscanf(str,"%d.%d.%d.%d",&ip1,&ip2,&ip3,&ip4))    // Perhaps inet_ad
      return(0);
   else
      return((ip1 * 0x1000000) +
             (ip2 * 0x10000) +
             (ip3 * 0x100) +
             (ip4));
}

int Name2IP(const char *name) {
   lpRT rt;
   if (rt = RTByName((char *) name))
      return(RT_IP(rt));
   else
      return(Str2IP((char *) name));
}



char IP2NameStr[40];
char* IP2IPstr(int ip) {
   unsigned long i = ip;
   sprintf(IP2NameStr,"%d.%d.%d.%d",
      ((i/0x01000000L) & 0x00FFL),
      ((i/0x00010000L) & 0x00FFL),
      ((i/0x00000100L) & 0x00FFL),
      (i               & 0x00FFL));
   return(IP2NameStr);
}

char* IP2Name(int ip) {
   lpRT rt = RTByIP(ip);
   if (rt) return(RT_NODE(rt));
   else    return(IP2IPstr(ip));
}

char* IP2NameOrNull(int ip) {
   lpRT rt = RTByIP(ip);
   if (rt) return(RT_NODE(rt));
   else    return(0);
}



// Called by Qopen
```

```
lpRT RTByService(lpQHANDLE q) {
   char sname[LINESIZE];
   char *p;
   lpRT rtp = RTROOT;
   sprintf(sname,",%s,",q->msgh.to_logical);

   while (rtp) {
      if (p = strstr(RT_APPS(rtp),sname))
         if (!ServiceHasFailed(q->msgh.to_logical,RT_IP(rtp)))
            break;
      rtp = NextRT(rtp);
   }

   if (rtp) { // Extract the physical server name
      while (*p != '[') p--;
      sscanf(p,"[%[^\]]",q->msgh.to_server); // physical server
   } else { // Try the local node

      if (SMByName(q->msgh.to_logical)) {
         rtp = RTByIP(SHAREDATA(hostip)); // Return this node
         strcpy(q->msgh.to_server,q->msgh.to_logical); // physical = logical
      }
   }

   return(rtp); // Null == failed
}


// Called by Qopen
lpRT ServerByName2(lpRT rtp, char *name, int index, char *physical) {
   char sname[LINESIZE];
   char *p;
   sprintf(sname,",%s,",name);

   while (rtp) {
      if (p = strstr(RT_APPS(rtp),sname))
         if (!(index--)) break; // Find the Nth entry
      rtp = NextRT(rtp);
   }

   if (rtp && physical) { // Extract the physical server name
      while (*p != '[') p--;
      sscanf(p,"[%[^\]]",physical);
   }

   return(rtp); // Null == failed
}

lpRT GetRTroot(){  // Used by GUI apps that want to find the RT
   if (!sm_base) AttachSharedMemory();
   return(RTROOT);
}


#define TROUTNAME              "c:\\q\\routing.tmp"

int BroadcastRT() {
        FILE*                  rtf = fopen(ROUTNAME,"r");
        lpRT                   rt = RTROOT;
        lpQHANDLE       qh = Qopen("QNETD",PUT_MODE,0,0,0,0,0);
        SMBUF                   b;
        int                     sz;

        Diag("BroadCastRT");
        if (!qh || !rtf) return(0);

        memset(&b,0,sizeof(b));
        qh->time_out = 1000; // One second
        strcpy(qh->msgh.to_logical,"QNETD");
        strcpy(qh->msgh.to_server,"QNETD");

        sz = fread(&b.mdata, sizeof( char ), MAXMSGDATA, rtf );
        fclose( rtf );

        while (rt = NextRT(rt)) {
      Say("BroadCastRT Trying %d",RT_IP(rt));
                if (RT_IP(rt) == SHAREDATA(hostip))     continue; // skip this n
      if (strstr(RT_NTYPE(rt),"NOBROADCAST")) continue; // skip this node
      qh->msgh.to_node = RT_IP(rt);
                qh->msgh.to_port = RT_PORT(rt);
                Qput(qh,QNETDREQ_MODE,NETMAN_RT_NEW, 0,sz,b.mdata);
        }

    return(1);
}


int CopyRT(int ip) {
        lpRT        rt;
        lpQHANDLE   qh = Qopen("QNETD",PUT_MODE,0,0,0,0,0);
    SMBUF                 b;
        if (!qh) return(0);

        // qh->time_out = 1000; // One second
        strcpy(qh->msgh.to_logical,"QNETD");
        strcpy(qh->msgh.to_server,"QNETD");

        if (rt = RTByIP(ip)) {
      Say("CpyRT Trying %d",ip);
                qh->msgh.to_node = RT_IP(rt);
                qh->msgh.to_port = RT_PORT(rt);
                QsendAndReceive(qh,QNETDREQ_MODE,NETMAN_RT_GET, 0,0,0,
                        MAXMSGDATA,b.mdata,0,&b.msgh);
      // Write to file
      if (b.msgh.size > 10) {
         FILE * fp = fopen(ROUTNAME,"w");
         Say("CpyRT saving copied table %d bytes %s",b.msgh.size,b.mdata);
         fwrite(b.mdata,sizeof( char ),b.msgh.size,fp);
         fclose(fp);
         // ReadRT();  called by watchdog.c when file time changes
      }
        } else
      return(0);
    return(1);
}
```

```
/*
**
** Copyright(C)1995 MITSUBISHI ELECTRIC ITA.  ALL RIGHTS RESERVED.
** UNPUBLISHED -- RIGHTS RESERVED UNDER THE COPYRIGHT
** LAWS OF THE UNITED STATES.  USE OF A COPYRIGHT NOTICE
** IS PRECAUTIONARY ONLY AND DOES NOT IMPLY PUBLICATION
** OR DISCLOSURE.
**
** THIS SOFTWARE CONTAINS CONFIDENTIAL INFORMATION AND
** TRADE SECRETS OF MITSUBISHI ELECTRIC ITA.  USE, DISCLOSURE,
** OR REPRODUCTION IS PROHIBITED WITHOUT THE PRIOR
** EXPRESS WRITTEN PERMISSION OF MITSUBISHI ELECTRIC ITA.
**
** OpenMQ
**    Module: rt.h
**    Author: Derek Schwenke 10/8/95
**
** Routing table header
*/

// Routing table structure
/*
typedef struct rt {            // Routing Table in shard memory
   int   ip;                   //
   int   qnetd_port;           //
   int   next_offset;          //
   int   ntype_index;          //
   int   apps_index;           //
   char  s[0];                 // node\0ntype\0apps\0 <<More than 4 char>>
} RT, *lpRT;
*/

#define RT2C(rt)         (char *)(rt)
#define RT_NODE(rt)      rt->s
#define RT_IP(rt)        rt->ip
#define RT_PORT(rt)      rt->qnetd_port
#define RT_NOFFSET(rt)   rt->next_offset
#define RT_APINDEX(rt)   rt->apps_index
#define RT_NTINDEX(rt)   rt->ntype_index
#define RT_NEXT(rt)      (lpRT)  (RT2C(rt) + rt->next_offset)
#define RT_APPS(rt)      &(rt->s[rt->apps_index])
#define RT_NTYPE(rt)     &(rt->s[rt->ntype_index])

// From routtab.c
#ifdef CPP
extern "C"
{
#endif
int   MarkFailedServer(lpQHANDLE q);
void  UpdateHandle(lpQHANDLE q);
lpFOL ServiceHasFailed(char *name, int ip);
lpRT  RTByService(lpQHANDLE q);
lpSMBUF SMByName(char *name);  // move to qlib.h?

char* IP2Name(int ip);
int   Name2IP(const char *name);
void  ReadRT();
lpRT  NextRT(lpRT rt);
void  PrintRT(lpRT r,char * s);
lpRT  ServerByNode(char *node);
//lpRT  IPByName(char *name);
lpRT  RTByName(char *name);
lpRT  RTByIP(int IP);
lpRT  ServerByName2(lpRT rtp, char *name, int index, char *physical);
lpRT  GetRTroot(); // Used by GUI apps that want to find the RT
// routtab.c -
int     BroadcastRT(); // used by netman
int     CopyRT(int ip); // used by netman


#ifdef CPP
}
#endif
```

```
/*
**
** Copyright(C)1995 MITSUBISHI ELECTRIC ITA.  ALL RIGHTS RESERVED.
** UNPUBLISHED -- RIGHTS RESERVED UNDER THE COPYRIGHT
** LAWS OF THE UNITED STATES.  USE OF A COPYRIGHT NOTICE
** IS PRECAUTIONARY ONLY AND DOES NOT IMPLY PUBLICATION
** OR DISCLOSURE.
**
** THIS SOFTWARE CONTAINS CONFIDENTIAL INFORMATION AND
** TRADE SECRETS OF MITSUBISHI ELECTRIC ITA.  USE, DISCLOSURE,
** OR REPRODUCTION IS PROHIBITED WITHOUT THE PRIOR
** EXPRESS WRITTEN PERMISSION OF MITSUBISHI ELECTRIC ITA.
**
** OpenMQ
**   Module: shm_acc.c
**   Author: Derek Schwenke 9/8/95
**
**
*/

#include "qlib.h"
#define  TABLETIMEOUT   10000

void ForcedBufferReset(int       bnum,        // Optional Buffer number (-1 = not us
                 lpQHANDLE  q,           // Optional Queue Handle (if set also
                 int        objn){       // Optional object number in Queue Han
   // This routine called only as repair after some failure.

   SMOBJS    sync;
   lpSMBUF   buf = SMBUFADDR(bnum);
   int       found,rtn,opened=0;
   char      name[NAMESIZE];

   if ((q) && (objn < 64)) {
      sync.freeh  = q->freeh[objn];
      sync.readyh = q->readyh[objn];
      sync.doneh  = q->doneh[objn];
      sync.ackh = 0;
      bnum = q->bufs_found[objn];
      buf = SMBUFADDR(bnum);
   } else if ( (bnum >= 0) && (bnum < SHAREDATA(nsbuf))) {
      opened++;
      sprintf(name,MUTFREFMT,found);
      if (!(sync.freeh = OpenMutex(SYNCHRONIZE, FALSE, name)))
         Warn("ForcedBufferReset[%d=%s]: OpenMutex %s failed #%d",bnum,buf->name

      sprintf(name,SEMRDYFMT,found);
      if (!(sync.readyh = OpenSemaphore(SYNCHRONIZE, FALSE, name)))
         Warn("ForcedBufferReset[%d=%s]: OpenSemaphore  %s failed #%d",bnum,buf-

      sprintf(name,SEMDONFMT,found);
      if (!(sync.doneh = OpenSemaphore(SEMAPHORE_ALL_ACCESS, FALSE, name)))
         Warn("ForcedBufferReset[%d=%s]: OpenSemaphore %s failed #%d",bnum,buf->

      sprintf(name,SEMACKFMT,found);
      if (!(sync.ackh = OpenSemaphore(SEMAPHORE_ALL_ACCESS, FALSE, name)))
         Warn("ForcedBufferReset[%d=%s]: OpenSemaphore %s failed #%d",bnum,buf->
   } else {
      sync.ackh = sync.doneh = sync.readyh = sync.freeh = NULL;
      Warn("ForcedBufferReset Fails: bad args bnum=%d q=%p objn=%d",bnum,q,objn)
```

```
  }

      // These Semephores have two states.
      // To reset first release then reaquire them
      /*
      if (!ReleaseSemaphore(sync.readyh,1,NULL))
         Warn("ForcedBufferReset[%d=%s]: ReleaseSemaphore ready readyh #%d",bnum
      if (!ReleaseSemaphore(sync.doneh,1,NULL))
         Warn("ForcedBufferReset[%d=%s]: ReleaseSemaphore ready doneh #%d",bnum,
      if (!ReleaseSemaphore(sync.ackh,1,NULL))
         Warn("ForcedBufferReset[%d=%s]: ReleaseSemaphore ready ackh #%d",bnum,b
      */
   if (sync.readyh)
      if ( (rtn = WaitForSingleObject(sync.readyh,10)) != WAIT_TIMEOUT )
         Warn("ForcedBufferReset[%d=%s]: WaitForSingleObject readyh was reset",b
   if (sync.doneh)
      if ( (rtn = WaitForSingleObject(sync.doneh,10)) != WAIT_TIMEOUT )
         Warn("ForcedBufferReset[%d=%s]: WaitForSingleObject doneh  was reset",b
   if (sync.ackh)
      if ( (rtn = WaitForSingleObject(sync.ackh,10)) != WAIT_TIMEOUT )
         Warn("ForcedBufferReset[%d=%s]: WaitForSingleObject ackh was reset",bnu

   if (sync.freeh)
      if(!ReleaseMutex(sync.freeh))
         Warn("ForcedBufferReset(%d=%s): ReleaseMutex freeh fail GetLastError =
            bnum,buf->name,GetLastError());

   if (opened) {
      CloseHandle(sync.freeh);
      CloseHandle(sync.readyh);
      CloseHandle(sync.doneh);
      CloseHandle(sync.ackh);
   }

}


lpSMBUF SMByName(char *name) {
   int i;
   for (i = 0; i < SHAREDATA(nsbuf); i++)
      if (!strcmp(name,(SMBUFADDR(i))->name))
         return(SMBUFADDR(i));
   return(0);
}


// This routine is called at startup by QNETD and QSERV.
// It's not done often so efficency is no issue here.

// This routine allocates 1 buffer and returns the buffer number or -1 on error.
// seqno specifes the number of existing buffers to expect already allocated.

int ReserveSharedBuffer(int seqno, char *bname, int bstatus,
                        lpSMOBJS sync) {
   int      used,found,i,rtn;
   char     name[NAMESIZE];
   lpSMBUF  BUF;
   HANDLE   SMT = OpenMutex( SYNCHRONIZE, FALSE, SMEMTABLE );
```

```
Diag("ReserveSharedBuffer(%s) starts...",bname);

if (!SMT) // SMT control all changes to the shared memory buffers
   Fail("ReserveSharedBuffer(%s): Cant OpenMutex for the table %d", bname, Ge

if ((rtn = WaitForSingleObject(SMT,TABLETIMEOUT)) != WAIT_OBJECT_0 )
   Fail(" Failed while waiting for the table %d",rtn);

// Count the existing number of buffers registered under this name
used = 0;  // number of times we saw the name
found = -1; // first empty buffer found
for (i = 0; i < SHAREDATA(nsbuf) ; i++) {
   BUF = SMBUFADDR(i) ;
        Diag("Checking buf %s",BUF->name);

       if (!strcmp(BUF->name,bname)) {
          if (used >= seqno) {
             Say("ReserveSharedBuffer(%s): FOUND STALE EXISTING BUFFER [%d] "
                 strcpy(BUF->name,"empty");
                 BUF->status = SMBUF_EMPTY;
              } else {
            Diag("ReserveSharedBuffer(%s): found an existing buffer[%d] ",bna
            used++;
              }
  }
       if ((found == -1) && (!strcmp(BUF->name,"empty"))) found = i;
}

if (found == -1) {
   Warn("ReserveSharedBuffer(%s):Cant allocate a buffer - all %d currently
       return(found);
}

// Get handles for free ready and done objects.
// All have previously been created by sharmeminit
BUF = SMBUFADDR(found) ;
sprintf(name,MUTFREFMT,found);
if (!(sync->freeh = OpenMutex(SYNCHRONIZE, FALSE, name)))
   Fail("ReserveSharedBuffer(%s): OpenMutex %s failed #%d",bname,name,GetLast
if (sync->freeh) ReleaseMutex(sync->freeh); // I dont own this mutex, This ca
GetLastError(); // clear this error out

sprintf(name,SEMRDYFMT,found);
if (!(sync->readyh = OpenSemaphore(SYNCHRONIZE, FALSE, name)))
   Fail("ReserveSharedBuffer(%s): OpenSemaphore  %s failed #%d",bname,name,Ge
if (sync->readyh) WaitForSingleObject(sync->readyh,10); // Reset

sprintf(name,SEMDONFMT,found);
if (!(sync->doneh = OpenSemaphore(SEMAPHORE_ALL_ACCESS, FALSE, name)))
   Fail("ReserveSharedBuffer(%s): OpenSemaphore %s failed #%d",bname,name,Get
if (sync->doneh) WaitForSingleObject(sync->doneh,10); // Reset

sprintf(name,SEMACKFMT,found);
if (!(sync->ackh = OpenSemaphore(SEMAPHORE_ALL_ACCESS, FALSE, name)))
   Fail("ReserveSharedBuffer(%s): OpenSemaphore %s failed #%d",bname,name,Get
if (sync->ackh) WaitForSingleObject(sync->ackh,10); // Reset

// Set name and status
strcpy(BUF->name,bname);
BUF->status = bstatus;
```

```
   ReleaseMutex(SMT);
   Diag("ReserveSharedBuffer(%s) returns [%d] addr %p",bname,found,BUF);
   return(found);
}


int UnReserveSharedBuffer(int bnum) {
   int        rtn = 0;
   lpSMBUF    BUF = SMBUFADDR(bnum);
   HANDLE     SMT = OpenMutex( SYNCHRONIZE, FALSE, SMEMTABLE );

   Diag("UnReserveSharedBuffer(%s)", BUF->name);

   if (!SMT)
      Fail("UnReserveSharedBuffer(%s): Cant OpenMutex for the table %d", BUF->na

   if ((rtn = WaitForSingleObject(SMT,TABLETIMEOUT)) != WAIT_OBJECT_0 )
      Fail(" Failed while waiting for the table %d",rtn);

         strcpy(BUF->name,"empty");
         BUF->status = SMBUF_EMPTY;
   ReleaseMutex(SMT);


   return(rtn);
}


// This routine is called at startup by QNETD and QSERV.
// It's not done often so efficency is no issue here.
lpSMBUFH AttachSharedMemory() {
   HANDLE     hMap;        // Shared memory mapping

         //Say("AttachSharedMemory \n");

   // Open the shared memory object
   if ( ! (hMap = OpenFileMapping(FILE_MAP_WRITE, FALSE, SMEMNAME)))
      Warn("OpenFileMapping returned null - QNETD is not running");

   // Map the shared memory into my address space
   else {
      if (! (sm_base = (lpSMBUFH) MapViewOfFile( hMap, FILE_MAP_WRITE,0,0,0)))
         Warn("MapViewOfFile returned null");

   // Note: hMap is not returned because we dont expect anyone
   // to ever close this mapping once they have it.
      Diag("AttachSharedMemory %s returns %p",&SMEMNAME,sm_base);
   }
   return(sm_base);
}
```

```
/*
**
** Copyright(C)1995 MITSUBISHI ELECTRIC ITA.  ALL RIGHTS RESERVED.
** UNPUBLISHED -- RIGHTS RESERVED UNDER THE COPYRIGHT
** LAWS OF THE UNITED STATES.  USE OF A COPYRIGHT NOTICE
** IS PRECAUTIONARY ONLY AND DOES NOT IMPLY PUBLICATION
** OR DISCLOSURE.
**
** THIS SOFTWARE CONTAINS CONFIDENTIAL INFORMATION AND
** TRADE SECRETS OF MITSUBISHI ELECTRIC ITA.  USE, DISCLOSURE,
** OR REPRODUCTION IS PROHIBITED WITHOUT THE PRIOR
** EXPRESS WRITTEN PERMISSION OF MITSUBISHI ELECTRIC ITA.
**
** OpenMQ
**   Module: tran.c
**   Author: Derek Schwenke 9/8/95
**
** Tran routines for single threaded clients, or well behaved
** multi threaded clients. There will be problems if 2
** threads do different COMMITs So, this code is not thread safe.
*/

#include "qlib.h"

//typedef struct tl {  // Message header
//   struct tl *next;  // Next
//   QHANDLE   q;      // Header version, and machine coding format
//} TL, *lpTL;

lpQHANDLE TLroot = NULL; // Open transaction list
int   tran_id = 1;   // Transactional ID for this process
int   last_tran_id;  // Last commited ID

lpQHANDLE FindTL(lpQHANDLE q) {
   lpQHANDLE tl = TLroot;

   while (tl)
      if (strcmp(tl->msgh.to_logical,q->msgh.to_logical) == 0 &&
          strcmp(tl->msgh.to_server, q->msgh.to_server) == 0)
         break;
      else
         tl = tl->next;
   return(tl);
}

void TLadd(lpQHANDLE q) {
   lpQHANDLE n;

   if ( n = FindTL(q)) {
      n->msgh.sub_mode++;      // Inc # of times used
   } else {
      n = (lpQHANDLE) malloc(sizeof(QHANDLE));
      Diag("TLadd: Starting for %s %s",q->msgh.to_logical,q->msgh.to_server);

      memcpy(n,q,sizeof(QHANDLE));
      n->msgh.sub_mode = 1;   // # of times used
      n->msgh.mid_tid = tran_id; // Save for acks
      n->next = TLroot;
      TLroot = n;
   }
```

```
}


void TLdel() {
   lpQHANDLE nn,n = TLroot;
   while (nn = n) {
      n = n->next;
      free(nn);
   } TLroot = NULL;
}


char open_tran[1000];
char old_trans[1000];
FILE *TLOG = NULL;            // Message Log File

void LogTran(char *s){
   if (!TLOG)
      if (!(TLOG = fopen(TLOGNAME,"w")))
              Fail("ERROR CAN NOT OPEN LOG FILE %s",&TLOGNAME);

   fprintf(TLOG,"%s",s);
}



void Qcommit(int action){      // Commit or abort
   // 1. Mark it commited on DISK
   // 2. Add to RAM list
   // 3. Send commit messages
   char  transtr[80];
   int   trouble = 0;
   lpQHANDLE  tl = TLroot;
   MSGH  mh;

   Diag("Qcommit(%d)",action);
   // The Open Transaction List should have accumulated q's to commit
   if(!tl){
      Diag("Attempt to commit with no outstanding messages Ignored");
   } else {

      strcpy(old_trans,open_tran);
      sprintf(transtr,":%d=%d",TLroot->msgh.mid.tid,action);
      strcat(open_tran,transtr);
      LogTran(open_tran); // <-- This is the real commit

      Diag("Qcommit(%d) transtr=%s open_tran=%s",action,transtr,open_tran);

      // Try to notify everyone about the commit or abort
      while(tl) {

         mh.mode = 0;   // Clear incase no full header is recived.
         QsendAndReceive(tl,action,0,0, 0,0, 0,0,0,&mh);
         if(mh.mode == ACK_MODE)
            Diag("Qcommit: was acknoledged!");
         else
            Diag("Qcommit: WAS NOT ACKNOLEDGED",trouble++);

         tl = tl->next;
      }

      TLdel(); // Clean up

      if(!trouble) {  // Remove from the open tran list
         strcpy(open_tran,old_trans);
         if (*open_tran) LogTran(open_tran);
      }
      if (*open_tran) Say("Qcommit: open_tran=[%s]",open_tran);

      last_tran_id = tran_id;
   }
}


int ResolveTran(lpMID MID){
   int    i, state = Q_ABORT;
   char   *p,id[40];

   sprintf(id,":%d,",MID->tid);

   if (p = strstr(old_trans,id))        // If this mid is on the old
      scanf(p,":%d=%d",i,state);       // Read the tran state

   else if ( TLroot )                  // If there's a inflight transaction
      if (TLroot->msgh.mid.tid == MID->tid)  // And the msg# matches
         state = Q_INPROGRESS;         // Return "inprogress"

   if (SHAREDATA(hostip) != MID->host) {
      Warn("ResolveTran PASSED the wrong host");
      state = Q_WRONGHOST;
   }
   return(state);
}

// This code is executed in each client's address space.
// But it should return a unique number

void SetTID(lpMID m, int flags) {

   if(flags & Q_TRAN) { // It's transactional make a ID
      if(BITSET(Q_TRAN_BEGIN,flags)||(tran_id == last_tran_id))
         tran_id = ++(SHAREDATA(tran_id)); // Generate new tran ID for this NODE
      Diag("SetTID id=%d",tran_id);
      m->tid = tran_id;
   }

   // Always make a unique id for each message.
   m->uid = ++(SHAREDATA(unique_id));

}
```

```
/*
**
** Copyright(C)1995 MITSUBISHI ELECTRIC ITA.  ALL RIGHTS RESERVED.
** UNPUBLISHED -- RIGHTS RESERVED UNDER THE COPYRIGHT
** LAWS OF THE UNITED STATES.  USE OF A COPYRIGHT NOTICE
** IS PRECAUTIONARY ONLY AND DOES NOT IMPLY PUBLICATION
** OR DISCLOSURE.
**
** THIS SOFTWARE CONTAINS CONFIDENTIAL INFORMATION AND
** TRADE SECRETS OF MITSUBISHI ELECTRIC ITA.  USE, DISCLOSURE,
** OR REPRODUCTION IS PROHIBITED WITHOUT THE PRIOR
** EXPRESS WRITTEN PERMISSION OF MITSUBISHI ELECTRIC ITA.
**
** OpenMQ
**   Module: copyrt.c
**   Author: Derek Schwenke 11/20/95
**
** Routing table broadcast funtions
*/

#include "qlib.h"
#include "qnetd.h"
#include "rt.h"

#define TROUTNAME           "c:\\q\\routing.tmp"

void BroadCastRT() {
        FILE*                   rtf = fopen(ROUTNAME,"r");
        FILE*                   trt = fopen(TROUTNAME,"w");
        lpRT                    rt = RTROOT;
        lpQHANDLE       qh = Qopen("QNETD",PUT_MODE,0,0,0,0,0);
        SMBUF                   b;
        int                     rd;

        Say("BroadCastRT");
        if (!qh || !rtf) return;

        memset(&b,0,sizeof(b));
        qh->time_out = 1000; // One second
        strcpy(qh->msgh.to_logical,"QNETD");
        strcpy(qh->msgh.to_server,"QNETD");

        rd = fread(&b.mdata, sizeof( char ), MAXMSGDATA, rtf );
        Say("BroadCastRT read got %d bytes ",rd);
        fclose( rtf );
        Say("BroadCastRT closed",b.mdata);

        rd = fwrite(&b.mdata, sizeof( char ), MAXMSGDATA, rtf);
        Say("BroadCastRT wrote %d ",rd);
        /*
        while (rt = RT_NEXT(rt)) {
                qh->msgh.to_node = RT_IP(rt);
                qh->msgh.to_port = RT_PORT(rt);
                QsendAndReceive(g_q,QNETDREQ_MODE,NETMAN_SMBUFH, 0,0,0,
                        (sizeof(SMBUFH) - MAXRTSIZE),(char *) &g_smhead,0,0);
        }
        */
}
```

```
int CopyRT(int IP) {
        lpRT        rt = RTROOT;
        lpQHANDLE  qh = Qopen("QNETD",PUT_MODE,0,0,0,0,0);
        if (!qh) return(0);

        qh->time_out = 1000; // One second
        strcpy(qh->msgh.to_logical,"QNETD");
        strcpy(qh->msgh.to_server,"QNETD");

        /*
        while (rt = RT_NEXT(rt)) {
                qh->msgh.to_node = RT_IP(rt);
                qh->msgh.to_port = RT_PORT(rt);
                QsendAndReceive(g_q,QNETDREQ_MODE,NETMAN_SMBUFH, 0,0,0,
                        (sizeof(SMBUFH) - MAXRTSIZE),(char *) &g_smhead,0,0);
        }
        */
   return(1);
}
```

```
/*
**
** Copyright(C)1995 MITSUBISHI ELECTRIC ITA.  ALL RIGHTS RESERVED.
** UNPUBLISHED -- RIGHTS RESERVED UNDER THE COPYRIGHT
** LAWS OF THE UNITED STATES.  USE OF A COPYRIGHT NOTICE
** IS PRECAUTIONARY ONLY AND DOES NOT IMPLY PUBLICATION
** OR DISCLOSURE.
**
** THIS SOFTWARE CONTAINS CONFIDENTIAL INFORMATION AND
** TRADE SECRETS OF MITSUBISHI ELECTRIC ITA.  USE, DISCLOSURE,
** OR REPRODUCTION IS PROHIBITED WITHOUT THE PRIOR
** EXPRESS WRITTEN PERMISSION OF MITSUBISHI ELECTRIC ITA.
**
** OpenMQ
**   Module: failover.c
**   Author: Derek Schwenke 9/22/95
*/
#include "qlib.h"



// Make entry in the fail over list
int MarkFailedServer(lpQHANDLE q) {
   lpFOL    fl = &(sm_base->FO[SHAREDATA(failed_servers)]);
   if (SHAREDATA(failed_servers) < MAXFOLENT - 1){
      strcpy(fl->name,q->msgh.to_logical);
      fl->ip = q->msgh.to_node;
      SHAREDATA(failed_servers)++;
      SHAREDATA(rt_rev)++;
      return(1);
   }
   return(0);
}


// Returns lpFOL if this server has failed
lpFOL ServiceHasFailed(char *name, int ip) {
   int i;
   for (i=0;i<SHAREDATA(failed_servers);i++)  {
      lpFOL    fl = &(sm_base->FO[i]);
      if (!strcmp(name,fl->name) && fl->ip == ip)
         return(fl);
   }
   return(0);
}


void UpdateHandle(lpQHANDLE q){
   q->rt_rev = SHAREDATA(rt_rev);
   if ((Q_FAILOVER & q->msgh.flags) && q->msgh.mode == PUT_MODE)  {
      QreOpen(q);
   }
}

/*

int markfailed(  lpQHANDLE q, int pass){
   lpQHANDLE failover;
   int rtn;
// Note that this server has failed us, Find a new one
```

```
   Warn("markfailed(%s) server on %p in pass=%d looking for replacement",q->msgh
   failover = Qopen( q->msgh.to_logical,
                     q->msgh.mode,
                     q->msgh.sub_mode,
                     q->base_flags,
                     0,0,pass);

// Add to failed list
   if(failover) {
      failover->next = FLroot;
      FLroot = failover;
      rtn = 1; // good
   } else {
      Say("markfailed(%s) no replacement found pass=%d",q->msgh.to_logical,pass)
      free(failover);
      rtn = 0; // bad
   }
   return(rtn);
}


lpQHANDLE findreplacment(lpQHANDLE q){
   lpQHANDLE fop = FLroot;  // List of all failed servers
   while(fop) { // Find first failover that matches
      if (!strcmp(fop->msgh.to_logical,q->msgh.to_logical)) break;
         fop = fop->next;
   }
   if (fop) {
      Say("findreplacment(%s) Found a fail over",q->msgh.to_logical);
      q = fop;   // We found a fail over for this service
   }
   return(q);
}

void flushfailedlist() {
   lpQHANDLE pp, cp = FLroot;  // List of all failed servers
   if (!cp) return;
   FLroot = NULL;

   //Sleep(60000); // Let any current transactions end
   while(pp=cp)
      cp = cp->next;
      free(pp);
}
*/
```

```
User: root
Host: bunny
Class: bunny
Job: stdin
```

```
// QNETDREQ_MODE sub modes:
#define NETMAN_SMBUFH   100
#define NETMAN_SMBUFS   101
#define NETMAN_SOCKETS  102
#define NETMAN_TRAN     103
#define NETMAN_RT_READ  104
// #define NETMAN_RT_BROAD 105
#define NETMAN_RT_NEW   106
#define NETMAN_RT_GET   107
#define NETMAN_CLR_FOL  110


// Shared Buffers Status
// ---------------------

typedef  struct bs {     // Shared memory buffers
   char  name[NAMESIZE];
   short status;
   short sub_status;
   MSGH  msgh;
} BS, *lpBS;

typedef  struct bsa {    // Shared memory buffers
   int   nsbuf;
   BS    bs[MAXNSMBUF];
} BSA, *lpBSA;



// Socket Status
// -------------

typedef struct ss {
   int   ip;
   int   port;
} SS, *lpSS;

typedef  struct ssa {    // Shared memory buffers
   int   sockets;
   SS    ss[FD_SETSIZE];
} SSA, *lpSSA;


// RT
// -------------

typedef  struct rta {    // Shared memory buffers
   char  RT[MAXRTSIZE];
} RTA, *lpRTA;
```

```
// admindlg.cpp : implementation file
//


#include "stdafx.h"
#include "qman.h"
#include "admindlg.h"

#define Q_LIB
#include "qlib.h"
#include "qadmin.h"
#include "rt.h"
#define  ADMTIMER 102
extern lpSMBUFH   sm_base;
extern QADMSTATS  g_s[3];
extern CString    g_que[3];
extern lpQHANDLE  QS[3];
int  IDC_APICS[8] = {IDC_QNONE1,IDC_QDOWN1,IDC_QSTOP1,IDC_QNOPUT1,IDC_QNOGET1,ID
enum pics {QNONE,QDOWN,QSTOP,QNOPUT,QNOGET,QUP,QNOPG,QFULL};
extern int  g_pic[3+3+3];

QADMCTLS    g_ad;

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif


extern CFont g_text_font;
int ALL_TEXT_A[] = {IDC_STATS,IDC_OGROUP,IDC_SGROUP,
                    IDC_PUTC,IDC_GETC,IDC_HALTC,IDC_SRESETC,IDC_FRESETC,IDC_SHU
                    IDC_MAXSIZE,IDC_LIMLAB,IDC_QSIZE,
                    IDOK,IDC_SET,IDC_REFRESH,0};


/////////////////////////////////////////////////////////////////////////////
// CAdminDlg dialog


CAdminDlg::CAdminDlg(CWnd* pParent /*=NULL*/)
        : CDialog(CAdminDlg::IDD, pParent)
{
        //{{AFX_DATA_INIT(CAdminDlg)
                // NOTE: the ClassWizard will add member initialization here
        //}}AFX_DATA_INIT
}


void CAdminDlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CAdminDlg)
                // NOTE: the ClassWizard will add DDX and DDV calls here
        //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CAdminDlg, CDialog)
        //{{AFX_MSG_MAP(CAdminDlg)
        ON_BN_CLICKED(IDC_REFRESH, OnRefresh)
        ON_BN_CLICKED(IDC_SET, OnSet)
        ON_CBN_EDITCHANGE(IDC_QSIZE, OnEditchangeQsize)
        ON_WM_TIMER()
        ON_WM_RBUTTONDOWN()
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()


/////////////////////////////////////////////////////////////////////////////
// CAdminDlg message handlers
void CAdminDlg::SetDisplay(int init){
        CString sp,sg,s,t1,t2;

   if (init) {
      t1 = g_que[m_id] + " Status";
      t2 = g_que[m_id] + " Settings";

        SetDlgItemText(IDC_OGROUP,t1);
        SetDlgItemText(IDC_SGROUP,t2);
   }


   CListBox* lb = (CListBox*) GetDlgItem(IDC_STATS);
   lb->ResetContent();

   s.Format("%5d committed entries",g_s[m_id].committed_entries); lb->InsertStri
   s.Format("%5d uncommitted puts", g_s[m_id].pending_puts);      lb->InsertStri
   s.Format("%5d uncommitted gets", g_s[m_id].pending_gets);      lb->InsertStri
   s.Format("%5d holes",            g_s[m_id].holey_entries);     lb->InsertS' 
   s.Format("%5d max entries",      g_s[m_id].max_entries);       lb->InsertS

   sg = ctime(&g_s[m_id].first_start_time);
   sp = ctime(&g_s[m_id].last_restart_time);

   s.Format("%5d restarts",g_s[m_id].num_restarts);           lb->InsertString(-1,
   s.Format("Last restart time %s",LPCTSTR(sg.Left(24)));    lb->InsertString(-1,
   s.Format("First restart time %s",LPCTSTR(sp.Left(24)));  lb->InsertString(-1,


   if (init) {
      s.Format("(%d limit)",g_s[m_id].max_entries_limit);
      SetDlgItemText(IDC_LIMLAB,LPCTSTR(s));
      SetDlgItemInt(IDC_QSIZE,g_s[m_id].max_entries);
   }

   // Select a icon
//   if (init)
//      for (int i=0;i<6;i++)
//         if (i != g_pic[m_id])
//            GetDlgItem(IDC_APICS[i])->ShowWindow(SW_HIDE);


   if ((g_pic[m_id] != g_pic[m_id+3+3])) {
      GetDlgItem(IDC_APICS[g_pic[m_id]])->ShowWindow(SW_SHOW);
      GetDlgItem(IDC_APICS[g_pic[m_id+3+3]])->ShowWindow(SW_HIDE);
      g_pic[m_id+3+3] = g_pic[m_id];
   }

   // Set check box items
   if (init) {
      ((CButton *) GetDlgItem(IDC_GETC))->SetCheck(g_s[m_id].qget_state);
      ((CButton *) GetDlgItem(IDC_PUTC))->SetCheck(g_s[m_id].qput_state);
      ((CButton *) GetDlgItem(IDC_SRESETC))->SetCheck(g_ad.stats_reset_flag);
      ((CButton *) GetDlgItem(IDC_FRESETC))->SetCheck(g_ad.full_reset_flag);
      ((CButton *) GetDlgItem(IDC_HALTC))->SetCheck(g_ad.halt_flag);
   }
}

BOOL CAdminDlg::OnInitDialog()
{
        CDialog::OnInitDialog();

   g_pic[m_id+3+3] = QNONE; // history is invalid
   for (int i=0;i<6;i++)    // Turn all pics off.
      GetDlgItem(IDC_APICS[i])->ShowWindow(SW_HIDE);
   SetDisplay(1);

   SetTimer(ADMTIMER,1000,NULL); // 1 sec


   i = 0;
   while (ALL_TEXT_A[i])
      GetDlgItem(ALL_TEXT_A[i++])->SetFont(&g_text_font);




   return TRUE;  // return TRUE unless you set the focus to a control
                 // EXCEPTION: OCX Property Pages should return FALSE
}


void CAdminDlg::OnRefresh()
{
   SetDisplay(1);
}



DWORD SetCtl(LPVOID m_id)
{
int id = (int)m_id;

   QsendAndReceive(QS[id],ADMINREQ_MODE,QADM_SET_CONTROLS, 0,sizeof(g_ad),(char


   g_ad.stats_reset_flag = 0;
   g_ad.full_reset_flag = 0;
   g_ad.shutdown_flag = 0;
   g_ad.halt_flag = 0;
   return(0);


}


void CAdminDlg::OnSet()
{
   memset(&g_ad,0,sizeof(g_ad));

   if (IsDlgButtonChecked(IDC_PUTC))     g_ad.enable_qputs_flag++;
   if (IsDlgButtonChecked(IDC_GETC))     g_ad.enable_qgets_flag++;
   if (IsDlgButtonChecked(IDC_SRESETC)) g_ad.stats_reset_flag++;
   if (IsDlgButtonChecked(IDC_FRESETC)) g_ad.full_reset_flag++;
   if (IsDlgButtonChecked(IDC_SHUTDOWNC)) g_ad.shutdown_flag++;
   if (IsDlgButtonChecked(IDC_HALTC))    g_ad.halt_flag++;

   int qs = GetDlgItemInt(IDC_QSIZE,NULL,TRUE);
   if ((qs>0)&&(qs<=g_s[m_id].max_entries_limit))
      g_ad.max_entries_value = qs;
      DWORD id;

      CreateThread(NULL,0,(LPTHREAD_START_ROUTINE) SetCtl,(LPVOID) m_id,0,&id);
 //   QsendAndReceive(QS[m_id],ADMINREQ_MODE,QADM_SET_CONTROLS, 0,sizeof(g_ad),(
   if (g_ad.shutdown_flag) {
      g_pic[m_id] = QSTOP; // QSTOP=Yellow (QDOWN=RED) we will hit a comm error
      // SetDisplay(0); // 0=refresh only.
   }
   //Sleep(1000);
   //SetDisplay(0); // 0=refresh only.
   //Sleep(1000);

   //if (g_ad.shutdown_flag) CDialog::OnOK(); // Exit
   CDialog::OnOK(); // Exit


   //g_ad.stats_reset_flag = 0;
   //g_ad.full_reset_flag = 0;
   //g_ad.shutdown_flag = 0;
   //g_ad.halt_flag = 0;


   //
   //SetDisplay(1); // 1=init: set buttons
}

void CAdminDlg::OnEditchangeQsize()
{
   int qs = GetDlgItemInt(IDC_QSIZE,NULL,TRUE);
   if (!((qs>0)&&(qs<=g_s[m_id].max_entries_limit)))
      SetDlgItemText(IDC_QSIZE,"");
}


void CAdminDlg::OnTimer(UINT nIDEvent)
{
        if (nIDEvent == ADMTIMER)
   SetDisplay(0);
        else
        CDialog::OnTimer(nIDEvent);
}

void CAdminDlg::OnRButtonDown(UINT nFlags, CPoint point)
{
   GetParentFrame()->SetMessageText("");
   this->Invalidate();
```

```
        CDialog::OnRButtonDown(nFlags, point);
}
```

```
// admindlg.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CAdminDlg dialog

class CAdminDlg : public CDialog
{
// Construction
public:
        CAdminDlg(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
        //{{AFX_DATA(CAdminDlg)
        enum { IDD = IDD_ADMINDIALOG };
                // NOTE: the ClassWizard will add data members here
        //}}AFX_DATA
   int m_id;

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CAdminDlg)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        //}}AFX_VIRTUAL
   void SetDisplay(int i);

// Implementation
protected:

        // Generated message map functions
        //{{AFX_MSG(CAdminDlg)
        virtual BOOL OnInitDialog();
        afx_msg void OnRefresh();
        afx_msg void OnSet();
        afx_msg void OnEditchangeQsize();
        afx_msg void OnTimer(UINT nIDEvent);
        afx_msg void OnRButtonDown(UINT nFlags, CPoint point);
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};
```

```
User: root
Host: bunny
Class: bunny
Job: stdin
```

```
// datadlg.cpp : implementation file
//

#include "stdafx.h"
#include "qman.h"
#include "datadlg.h"
#include "KeySearch.h"

#define Q_LIB
#include "qlib.h"
#include "qadmin.h"
#include "rt.h"
#include "orderfm.h"

extern lpQHANDLE  QS[3];
extern QADMSTATS  g_s[3];
extern CString    g_que[3];

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CdataDlg dialog


CdataDlg::CdataDlg(CWnd* pParent /*=NULL*/)
        : CDialog(CdataDlg::IDD, pParent)
{
        //{{AFX_DATA_INIT(CdataDlg)
                // NOTE: the ClassWizard will add member initialization here
        //}}AFX_DATA_INIT
}


void CdataDlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CdataDlg)
                // NOTE: the ClassWizard will add DDX and DDV calls here
        //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CdataDlg, CDialog)
        //{{AFX_MSG_MAP(CdataDlg)
        ON_BN_CLICKED(IDREFRESHB, OnRefreshb)
        ON_BN_CLICKED(IDC_CMT, OnCmt)
        ON_BN_CLICKED(IDC_UNCMT, OnUncmt)
        ON_BN_CLICKED(IDR_SEARCHB, OnSearchb)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()



        SMBUF m;

void CdataDlg::ScreenInit() {
   CString s,ss;
   int   i,gotsize;
   pOFORM po = (pOFORM) &m.mdata;

   if (m_sub_mode == QADM_REQ_COM_DATA)
      s.Format("%s   %4d Commited %20d Uncommited entries",LPCTSTR(g_que[m_ic
            g_s[m_id].committed_entries,  g_s[m_id].pending_gets + g_s[m_id].p_n
   else
      s.Format("%s   %4d Uncommited %20d Commited entries",LPCTSTR(g_que[m_id]),
        g_s[m_id].pending_gets + g_s[m_id].pending_puts, g_s[m_id].committed_ent
   SetDlgItemText(IDC_TITLE,LPCTSTR(s));

   CListBox* lb = (CListBox*) GetDlgItem(IDC_DATAL);
   lb->ResetContent();

   for (i = 0; i < 200; i++) {
      *m.mdata = i;
      if (QSUCCESS == QsendAndReceive(QS[m_id],ADMINREQ_MODE,m_sub_mode,
          0,sizeof(int),m.mdata, sizeof(m.mdata),m.mdata,&gotsize,&m.msgh))

      ss = "%3d\t%s\t%s\t%s";
      if (gotsize == sizeof(OFORM)) {
         s.Format(LPCTSTR(ss),i,po->cust,po->item,po->qty,0);
         lb->InsertString(-1,s);
      } else if (gotsize) {
         s.Format("%3d -\t%s",i,m.mdata);
         lb->InsertString(-1,s);
      } else {
         break;
      }
   }
 }

#define FORMATNAME            "c:\\q\\formats.txt"

char * IsAddr(char *c){
   int off = 0;
   if (strstr(c,"msgh")){
      if (strstr(c,"size")) return((char *)m.msgh.size);
   } else if (strstr(c,"mdata")) {
      sscanf(c,"%*[^[]\[%d",off);
      return((m.mdata + off)) ;
   }
   return(0);
}


typedef struct fmts {  // Thread parameters
   void*   testa;
   int     testv;
   char    fmt[100];
   void*   a[10];
   fmts*   next;
} FMTS, *pFMTS;

void clearfmt( pFMTS p){
   p->testa = NULL;
   p->testv = 0;
  *p->fmt = 0;
   for (int i=0;i < 10;i++) p->a[i] = NULL;
   p->next = 0;
}


FMTS fmts;

void ParseFormats(){
   char line[LINESIZE];
   char test[LINESIZE];
   char format[LINESIZE];
   char ops[LINESIZE];
   char op1[LINESIZE];
   char op2[LINESIZE];
   char op[LINESIZE];
   FILE *fp = fopen(FORMATNAME,"r");

   if ( ! fp ) {
        //  Say("Cant open the data formats file %s",FORMATNAME);
          return;
        }

   while (fgets(line,LINESIZE,fp)) {

      if ( 3 == sscanf(line,"%[^;];%[^;];%[^;]",test,format,ops)) {
         if (strchr(test,'#')) continue;
         if ( 3 != sscanf(test,"%s %s %s",op1,op,op2)) continue;
         if (strstr(op1,"size")) fmts.testa = &m.msgh.size;
         sscanf(op2,"%d",&fmts.testv);
         strcpy(fmts.fmt,format);

      } else {
         ; //Say("ReadParms: Ignoring: %s",line);
      }
   }
   if (ferror(fp)){
     // Fail("read error in parameters file %s",PARMNAME);
      clearerr(fp);
   }
   fclose(fp);
}

/////////////////////////////////////////////////////////////////////////////
// CdataDlg message handlers

BOOL CdataDlg::OnInitDialog()
{
         CDialog::OnInitDialog();
         m_sub_mode = QADM_REQ_COM_DATA;
   ((CButton *) GetDlgItem(IDC_CMT))->SetCheck(TRUE);

   ScreenInit();

         return TRUE;  // return TRUE unless you set the focus to a control
                       // EXCEPTION: OCX Property Pages should return FALSE
}

void CdataDlg::OnRefreshb()
{
   ScreenInit();
}
```

```
void CdataDlg::OnCmt()
{
        m_sub_mode = QADM_REQ_COM_DATA;
   ScreenInit();

}

void CdataDlg::OnUncmt()
{
        m_sub_mode = QADM_REQ_UNCOM_DATA;
   ScreenInit();

}

void CdataDlg::OnSearchb()
{
   CKeySearch d;
   d.m_id = m_id;
   d.DoModal();
}
```

```
// datadlg.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CdataDlg dialog

class CdataDlg : public CDialog
{
// Construction
public:
        CdataDlg(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
        //{{AFX_DATA(CdataDlg)
        enum { IDD = IDD_DATADIALOG };
                // NOTE: the ClassWizard will add data members here
        //}}AFX_DATA
   int m_id;
   int m_sub_mode;

   void ScreenInit();
// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CdataDlg)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        //}}AFX_VIRTUAL

// Implementation
protected:

        // Generated message map functions
        //{{AFX_MSG(CdataDlg)
        virtual BOOL OnInitDialog();
        afx_msg void OnRefreshb();
        afx_msg void OnCmt();
        afx_msg void OnUncmt();
        afx_msg void OnSearchb();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};
```

```
// KeySearch.h : header file
//


/////////////////////////////////////////////////////////////////////////////
// CKeySearch dialog

class CKeySearch : public CDialog
{
// Construction
public:
	CKeySearch(CWnd* pParent = NULL);   // standard constructor
   //void CM_switch(int mode);
// Dialog Data
	//{{AFX_DATA(CKeySearch)
	enum { IDD = IDD_KEYSEARCH };
		// NOTE: the ClassWizard will add data members here
	//}}AFX_DATA
   //int   m_min_int,m_max_int,m_at_int;
   //CString m_min_str,m_max_str,m_at_str;
   int m_id; // 1-3
   int m_preds;
   int m_pred_type[3];
   int m_pic;
   int m_committed;
   int m_uncommitted;
   int m_total_entries;
   int m_search_type;

   void OnCompChange(int pred);
   void ChangePredView(int pred, int act);
   void CopyInput(int pred, int min, int max);


// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CKeySearch)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:

	// Generated message map functions
	//{{AFX_MSG(CKeySearch)
	virtual BOOL OnInitDialog();
	afx_msg void OnSearchb();
	afx_msg void OnAnd1();
	afx_msg void OnAnd2();
	afx_msg void OnSelchangeCompCb1();
	afx_msg void OnSelchangeCompCb2();
	afx_msg void OnSelchangeCompCb3();
	afx_msg void OnEditchangeMinCb1();
	afx_msg void OnEditchangeMinCb2();
	afx_msg void OnEditchangeMinCb3();
	afx_msg void OnEditchangeMaxCb1();
	afx_msg void OnEditchangeMaxCb2();
	afx_msg void OnEditchangeMaxCb3();
	afx_msg void OnTimer(UINT nIDEvent);
```

```
        afx_msg void OnAllR();
        afx_msg void OnComR();
        afx_msg void OnUncomR();
        afx_msg void OnRButtonDown(UINT nFlags, CPoint point);
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};
```

```
// mainfrm.cpp : implementation of the CMainFrame class
//

#include "stdafx.h"
#include "qman.h"

#include "mainfrm.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CMainFrame

IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
	//{{AFX_MSG_MAP(CMainFrame)
		// NOTE - the ClassWizard will add and remove mapping macros her
		//    DO NOT EDIT what you see in these blocks of generated code
	ON_WM_CREATE()
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// arrays of IDs used to initialize control bars

// toolbar buttons - IDs are command buttons
static UINT BASED_CODE buttons[] =
{
	// same order as in the bitmap 'toolbar.bmp'
	ID_FILE_NEW,
	ID_FILE_OPEN,
	ID_FILE_SAVE,
		ID_SEPARATOR,
	ID_EDIT_CUT,
	ID_EDIT_COPY,
	ID_EDIT_PASTE,
		ID_SEPARATOR,
	ID_FILE_PRINT,
	ID_APP_ABOUT,
};

static UINT BASED_CODE indicators[] =
{
	ID_SEPARATOR,           // status line indicator
	ID_INDICATOR_CAPS,
	ID_INDICATOR_NUM,
	ID_INDICATOR_SCRL,
};

/////////////////////////////////////////////////////////////////////////////
// CMainFrame construction/destruction

CMainFrame::CMainFrame()
{
	// TODO: add member initialization code here
}

CMainFrame::~CMainFrame()
{
}

int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
        if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
                return -1;

        if (!m_wndToolBar.Create(this) ||
                !m_wndToolBar.LoadBitmap(IDR_MAINFRAME) ||
                !m_wndToolBar.SetButtons(buttons,
                  sizeof(buttons)/sizeof(UINT)))
        {
                TRACE0("Failed to create toolbar\n");
                return -1;      // fail to create
        }

/* Derek's remove tool bar */
    m_wndToolBar.ShowWindow(SW_HIDE);

        if (!m_wndStatusBar.Create(this) ||
                !m_wndStatusBar.SetIndicators(indicators,
                  sizeof(indicators)/sizeof(UINT)))
        {
                TRACE0("Failed to create status bar\n");
                return -1;      // fail to create
        }

        // TODO: Delete these three lines if you don't want the toolbar to
        //  be dockable
        m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
        EnableDocking(CBRS_ALIGN_ANY);
        DockControlBar(&m_wndToolBar);

        // TODO: Remove this if you don't want tool tips
        m_wndToolBar.SetBarStyle(m_wndToolBar.GetBarStyle() |
                CBRS_TOOLTIPS | CBRS_FLYBY);

        return 0;
}

/////////////////////////////////////////////////////////////////////////////
// CMainFrame diagnostics

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
        CFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{
        CFrameWnd::Dump(dc);
}

#endif //_DEBUG
```

```
/////////////////////////////////////////////////////////////////////////////
// CMainFrame message handlers
// mainfrm.h : interface of the CMainFrame class
//
///////////////////////////////////////////////////////////////////////////////

class CMainFrame : public CFrameWnd
{
protected: // create from serialization only
        CMainFrame();
        DECLARE_DYNCREATE(CMainFrame)

// Attributes
public:

// Operations
public:

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CMainFrame)
        //}}AFX_VIRTUAL

// Implementation
public:
        virtual ~CMainFrame();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

protected:  // control bar embedded members
        CStatusBar  m_wndStatusBar;
        CToolBar    m_wndToolBar;

// Generated message map functions
protected:
        //{{AFX_MSG(CMainFrame)
        afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
                // NOTE - the ClassWizard will add and remove member functions h
                //    DO NOT EDIT what you see in these blocks of generated code
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

///////////////////////////////////////////////////////////////////////////////
```

```
// KeySearch.cpp : implementation file
//

#include "stdafx.h"
#include "qman.h"
#include "KeySearch.h"
#define Q_LIB
#include "qlib.h"
#include "qadmin.h"
#include "orderfm.h"
#include "rt.h"


#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

#define KEYTIMER 106
extern lpQHANDLE  QS[3];
extern QADMSTATS  g_s[3];
extern CString    g_que[3];

extern   lpSMBUFH sm_base;
extern int  g_pic[3+3+3];
/////////////////////////////////////////////////////////////////////////////
#define ID_OBJS   9

int    IDC_KPICS[8] = {IDC_QNONE1,IDC_QDOWN1,IDC_QSTOP1,IDC_QNOPUT1,IDC_QNOGET
enum   IDTYPE   {COMP,MIN,AT,MAX,AND,OPA,OPB,OPC,OPD};
int    g_id[3][ID_OBJS] = {{IDC_COMP_CB1,IDC_MIN_CB1,IDC_AT_CB1,IDC_MAX_CB1,IDC_A
                           {IDC_COMP_CB2,IDC_MIN_CB2,IDC_AT_CB2,IDC_MAX_CB2,IDC_A
                           {IDC_COMP_CB3,IDC_MIN_CB3,IDC_AT_CB3,IDC_MAX_CB3,IDC_A

// List of items to get big fonts
extern CFont g_text_font;
int ALL_TEXT_S[] = {IDC_ALL_R,IDC_COM_R,IDC_UNCOM_R,IDC_MODE,IDOK,IDSEARCHB,IDC_

/////////////////////////////////////////////////////////////////////////////
// CKeySearch dialog


CKeySearch::CKeySearch(CWnd* pParent /*=NULL*/)
        : CDialog(CKeySearch::IDD, pParent)
{
        //{{AFX_DATA_INIT(CKeySearch)
                // NOTE: the ClassWizard will add member initialization here
        //}}AFX_DATA_INIT
}


void CKeySearch::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CKeySearch)
                // NOTE: the ClassWizard will add DDX and DDV calls here
        //}}AFX_DATA_MAP
}
```

```
BEGIN_MESSAGE_MAP(CKeySearch, CDialog)
        //{{AFX_MSG_MAP(CKeySearch)
        ON_BN_CLICKED(IDSEARCHB, OnSearchb)
        ON_BN_CLICKED(IDC_AND1, OnAnd1)
        ON_BN_CLICKED(IDC_AND2, OnAnd2)
        ON_CBN_SELCHANGE(IDC_COMP_CB1, OnSelchangeCompCb1)
        ON_CBN_SELCHANGE(IDC_COMP_CB2, OnSelchangeCompCb2)
        ON_CBN_SELCHANGE(IDC_COMP_CB3, OnSelchangeCompCb3)
        ON_CBN_EDITCHANGE(IDC_MIN_CB1, OnEditchangeMinCb1)
        ON_CBN_EDITCHANGE(IDC_MIN_CB2, OnEditchangeMinCb2)
        ON_CBN_EDITCHANGE(IDC_MIN_CB3, OnEditchangeMinCb3)
        ON_CBN_EDITCHANGE(IDC_MAX_CB1, OnEditchangeMaxCb1)
        ON_CBN_EDITCHANGE(IDC_MAX_CB2, OnEditchangeMaxCb2)
        ON_CBN_EDITCHANGE(IDC_MAX_CB3, OnEditchangeMaxCb3)
        ON_WM_TIMER()
        ON_BN_CLICKED(IDC_ALL_R, OnAllR)
        ON_BN_CLICKED(IDC_COM_R, OnComR)
        ON_BN_CLICKED(IDC_UNCOM_R, OnUncomR)
        ON_WM_RBUTTONDOWN()
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CKeySearch message handlers

BOOL CKeySearch::OnInitDialog()
{
        CDialog::OnInitDialog();
        int i,id;

   m_preds = 1;
   m_pred_type[0] = m_pred_type[1] = m_pred_type[2] =  INT_SEARCH_TYPE;

   for (id = 1; id < OPA; id++)
      GetDlgItem(g_id[0][id])->EnableWindow(FALSE);

   m_pic = (g_pic[m_id] + 1) % 8;

   ((CButton *) GetDlgItem(IDC_ALL_R))->SetCheck(TRUE);
   m_search_type = SEARCH_ALL_ENT;

   m_committed = -1;
   m_uncommitted = -1;
   m_total_entries = -1;

   SetDlgItemText(IDC_MODE,"View " + g_que[m_id] + " Entries" );

   SetTimer(KEYTIMER,1000,NULL); // 4 sec


   i = 0;
   while (ALL_TEXT_S[i])
      GetDlgItem(ALL_TEXT_S[i++])->SetFont(&g_text_font);


        return TRUE;  // return TRUE unless you set the focus to a control
                      // EXCEPTION: OCX Property Pages should return FALSE
}
```

```
int String2Time(const char *s){  // Thu Nov 30 17:30:00 1995
   char mon[80];
   int  mday,hr,mn,sc,yr,args;
   tm TM;
   // If it starts with a day skip it.
   sscanf(s,"%s",mon);
   if (strstr("Mon Tue Wed Thu Fri Sat Sun",mon)) s = s + 4;

   args = sscanf(s,"%s %d %d:%d:%d %d",mon,&mday,&hr,&mn,&sc,&yr);
   if      (!strcmp(mon,"Jan")) TM.tm_mon = 0;
   else if (!strcmp(mon,"Feb")) TM.tm_mon = 1;
   else if (!strcmp(mon,"Mar")) TM.tm_mon = 2;
   else if (!strcmp(mon,"Apr")) TM.tm_mon = 3;
   else if (!strcmp(mon,"May")) TM.tm_mon = 4;
   else if (!strcmp(mon,"Jun")) TM.tm_mon = 5;
   else if (!strcmp(mon,"Jul")) TM.tm_mon = 6;
   else if (!strcmp(mon,"Aug")) TM.tm_mon = 7;
   else if (!strcmp(mon,"Sep")) TM.tm_mon = 8;
   else if (!strcmp(mon,"Oct")) TM.tm_mon = 9;
   else if (!strcmp(mon,"Nov")) TM.tm_mon = 10;
   else if (!strcmp(mon,"Dec")) TM.tm_mon = 11;
   else  TM.tm_mon = 12;

   if ((args == 6) && (TM.tm_mon != 12)) {
      TM.tm_sec = sc;
      TM.tm_min = mn;
      TM.tm_hour = hr;
      TM.tm_mday = mday;
      TM.tm_year = yr - 1900;
      TM.tm_isdst = -1;
      return(mktime(&TM));
   } else
   return(-1);

}
void Time2String(int time,char *s, int printday){
   if (printday)
      strcpy(s,ctime((time_t*)(&time)));
   else
      strcpy(s,(ctime((time_t*)(&time)) + 4)  ); // dont print day
   s[strlen(s) - 1] = 0;
}

void CKeySearch::OnCompChange(int pred) {
   CString type_str;
   GetDlgItemText(g_id[pred][COMP],type_str);
   int comptype = INT_SEARCH_TYPE;
   if (type_str == "String")  comptype = STR_SEARCH_TYPE;
   if (type_str == "Short")   comptype = SHORT_SEARCH_TYPE;
   m_pred_type[pred] = comptype;

   CComboBox * CBat  = (CComboBox *) this->GetDlgItem(g_id[pred][AT]);
   CComboBox * CBmin = (CComboBox *) this->GetDlgItem(g_id[pred][MIN]);
   CComboBox * CBmax = (CComboBox *) this->GetDlgItem(g_id[pred][MAX]);

   CBat->ResetContent();
   CBmin->ResetContent();
   CBmax->ResetContent();
   CBmin->AddString("");
   CBmin->AddString("");
   CBmin->AddString("ANY");
   CBmax->AddString("");
   CBmax->AddString("");
   CBmax->AddString("ANY");

   if (type_str == "Time") {
      char tim[80];
      Time2String(SHAREDATA(time),tim,0);
      CBat->AddString("TIME");

      CBmin->SetCurSel(-1);
      CBmin->SetWindowText(tim);
//    CBmin->ReplaceSel(tim);


      CBmin->AddString(tim);  //  add to edit box too?
      CBmax->AddString(tim);  //  add to edit box too?
   } else if (comptype == STR_SEARCH_TYPE) {
      CBat->AddString("CUSTOMER");
      CBat->AddString("ITEM");
      CBat->AddString("to_server");
      CBat->AddString("to_logical");
   } else if (comptype == INT_SEARCH_TYPE) {
      CBat->AddString("UID");
      CBat->AddString("TID");
      CBat->AddString("HOST");
      CBat->AddString("QUANTITY");
      lpRT rt = RTROOT;
      while (rt = NextRT(rt)) {
         CBmin->AddString(RT_NODE(rt));
         CBmax->AddString(RT_NODE(rt));
      }
   } else if (comptype == SHORT_SEARCH_TYPE) {
      CBat->AddString("MODE");
      CBat->AddString("SUB_MODE");
   }

   for (int id = 1; id < ID_OBJS; id++)
      GetDlgItem(g_id[pred][id])->EnableWindow(TRUE);
   GetDlgItem(IDSEARCHB)->EnableWindow(TRUE);

  // GetParentFrame()->SetMessageText(""); cedit
   SetDlgItemText(IDC_SERSTAT,"");

   ((CComboBox *) this->GetDlgItem(g_id[pred][AT]))->SetCurSel(0);
   ((CComboBox *) this->GetDlgItem(g_id[pred][MAX]))->SetCurSel(0);
   ((CComboBox *) this->GetDlgItem(g_id[pred][MIN]))->SetCurSel(0);

}

void CKeySearch::OnSelchangeCompCb1() {OnCompChange(0);}
void CKeySearch::OnSelchangeCompCb2() {OnCompChange(1);}
void CKeySearch::OnSelchangeCompCb3() {OnCompChange(2);}

void CKeySearch::ChangePredView(int pred, int act)
{
   int id;

   for (id = 0; id < ID_OBJS; id++)
```

```
      GetDlgItem(g_id[pred][id])->ShowWindow(act);
   for (id = 1; id < OPA; id++)
      GetDlgItem(g_id[pred][id])->EnableWindow(FALSE);
   if (act == SW_HIDE)
      for  (id = 0; id < AND; id++)
         ((CComboBox *) this->GetDlgItem(g_id[pred][id]))->SetCurSel(-1);

   if (act == SW_HIDE ) { // Get rid of the ghost of the combo box  MFC bug?
      this->Invalidate();
      this->UpdateWindow();
   }
}




void CKeySearch::OnAnd1()
{
        if (m_preds == 1) {
      ChangePredView(1,SW_SHOW);
      m_preds = 2;
   } else {
      ChangePredView(1,SW_HIDE);
      ChangePredView(2,SW_HIDE);
      m_preds = 1;
   }
}

void CKeySearch::OnAnd2()
{
        if (m_preds == 2) {
      ChangePredView(2,SW_SHOW);
      m_preds = 3;
   } else {
      ChangePredView(2,SW_HIDE);
      m_preds = 2;
   }
}

void CKeySearch::CopyInput(int pred, int from, int to)
{
   CString s; int pos;
   GetDlgItemText(g_id[pred][from],s.GetBuffer(100),100);
   s.ReleaseBuffer();

   pos = ((CComboBox *) this->GetDlgItem(g_id[pred][to]))->GetCurSel();
   ((CComboBox *) this->GetDlgItem(g_id[pred][to]))->DeleteString(0);
   ((CComboBox *) this->GetDlgItem(g_id[pred][to]))->InsertString(0,s);
   ((CComboBox *) this->GetDlgItem(g_id[pred][from]))->DeleteString(1);
   ((CComboBox *) this->GetDlgItem(g_id[pred][from]))->InsertString(1,s);
   if (pos == 0)
      ((CComboBox *) this->GetDlgItem(g_id[pred][to]))->SetCurSel(0);

}

void CKeySearch::OnEditchangeMinCb1() {CopyInput(0,MIN,MAX);}
void CKeySearch::OnEditchangeMinCb2() {CopyInput(1,MIN,MAX);}
void CKeySearch::OnEditchangeMinCb3() {CopyInput(2,MIN,MAX);}
void CKeySearch::OnEditchangeMaxCb1()  {CopyInput(0,MAX,MIN);}
void CKeySearch::OnEditchangeMaxCb2()  {CopyInput(1,MAX,MIN);}
void CKeySearch::OnEditchangeMaxCb3()  {CopyInput(2,MAX,MIN);}

void CKeySearch::OnSearchb()
{
   SMBUF b,B;
   QADMSEL key;
   OFORM order;

   CString ss,s,at_str,min_str, max_str;
   int i,at_int, sz, matches;

   GetDlgItem(IDSEARCHB)->EnableWindow(FALSE);
   CListBox* lb = (CListBox*) GetDlgItem(IDC_SEARCH_LB);
   lb->ResetContent();
   SetDlgItemText(IDC_SERSTAT,"");
   // lb->SetTabStops(100);

   key.num_preds = m_preds;
   key.search_type = m_search_type; // SEARCH_ALL_ENT
   for (int p = 0; p < m_preds; p++) {
      // Set min and max values
      min_str = "";
      max_str = "";
      GetDlgItemText(g_id[p][MIN],min_str.GetBuffer(100),100);
      GetDlgItemText(g_id[p][MAX],max_str.GetBuffer(100),100);
      GetDlgItemText(g_id[p][AT],at_str.GetBuffer(100),100);
      min_str.ReleaseBuffer();
      max_str.ReleaseBuffer();
      at_str.ReleaseBuffer();
      at_int = GetDlgItemInt(g_id[p][AT],NULL,TRUE);

      strcpy(key.preds[p].min_str_val,LPCTSTR(min_str));
      strcpy(key.preds[p].max_str_val,LPCTSTR(max_str));
      key.preds[p].min_int_val = GetDlgItemInt(g_id[p][MIN],NULL,TRUE); // min i
      key.preds[p].max_int_val = GetDlgItemInt(g_id[p][MAX],NULL,TRUE); // max i
      key.preds[p].min_sh_val  = key.preds[p].min_int_val; // min short
      key.preds[p].max_sh_val  = key.preds[p].max_int_val; // max short


      if (min_str == "ANY") {
         key.preds[p].min_switch = 0;
         key.preds[p].min_str_len = 0;
      } else {
         key.preds[p].min_switch = 1;
         key.preds[p].min_str_len = strlen(key.preds[p].min_str_val);
      }

      if (max_str == "ANY") {
         key.preds[p].max_switch = 0;
         key.preds[p].max_str_len = 0;
      } else {
         key.preds[p].max_switch = 1;
         key.preds[p].max_str_len = strlen(key.preds[p].max_str_val);
      }

      if (at_str == "TIME") {
         //char tim[80];
         if (max_str == "NOW")
         key.preds[p].max_int_val = SHAREDATA(time);
      if ((i = String2Time(LPCTSTR(max_str))) != -1)
         key.preds[p].max_int_val = i;
      if (min_str == "NOW")
         key.preds[p].min_int_val = SHAREDATA(time);
      if ((i = String2Time(LPCTSTR(min_str))) != -1)
         key.preds[p].min_int_val = i;

     // Time2String(key.preds[p].min_int_val,tim,1);
     // s.Format("Time value:%s",tim);
     // SetDlgItemText(IDC_SERSTAT,s);
   }

   if (at_str == "HOST") {
      if (i = Name2IP(LPCTSTR(max_str)))
         key.preds[p].max_int_val = i;
      if (i = Name2IP(LPCTSTR(min_str)))
         key.preds[p].min_int_val = i;
   }

   // Find the offset
   if (at_str == "TIME")
       at_int = ((char *) &b.msgh.time)        - ((char *) &b.msgh);

   if (at_str == "MODE")
       at_int = ((char *) &b.msgh.mode)        - ((char *) &b.msgh);

   if (at_str == "SUB_MODE")
       at_int = ((char *) &b.msgh.sub_mode)    - ((char *) &b.msgh);

   if (at_str == "UID")
       at_int = ((char *) &b.msgh.mid.uid)     - ((char *) &b.msgh);

   if (at_str == "TID")
       at_int = ((char *) &b.msgh.mid.tid)     - ((char *) &b.msgh);

   if (at_str == "HOST")
       at_int = ((char *) &b.msgh.mid.host)    - ((char *) &b.msgh);

   if (at_str == "SIZE")
       at_int = ((char *) &b.msgh.size)        - ((char *) &b.msgh);

   if (at_str == "to_server")
       at_int = ((char *) &b.msgh.to_server)   - ((char *) &b.msgh);

   if (at_str == "to_logical")
       at_int = ((char *) &b.msgh.to_logical) - ((char *) &b.msgh);

   if (at_str == "CUSTOMER")
       at_int = sizeof(MSGH) + ((char *) &order.cust) - ((char *) &order);

   if (at_str == "ITEM")
       at_int = sizeof(MSGH) + ((char *) &order.item) - ((char *) &order);

   if (at_str == "QUANTITY")
       at_int = sizeof(MSGH) + ((char *) &order.qty)  - ((char *) &order);


   key.preds[p].offset = at_int;
   key.preds[p].pred_type = m_pred_type[p];
```

```
    }


    // Get the list of key matches
    if (QSUCCESS == QsendAndReceive(QS[m_id],ADMINREQ_MODE,QADM_REQ_SEL_DATA,
       0,sizeof(key),(char *) &key, sizeof(b.mdata),b.mdata,&sz,&b.msgh)) {
       lpMID md = (lpMID) b.mdata;

       matches = sz / sizeof(MID);
       if (matches > 0) {
//           char * h = IP2Name(md->host);
          for (i = 0; i < matches ; i++) {
             ss.Format("%s(%d,%d)", IP2Name(md->host),md->uid,md->tid);
             s.Format("%4d \t%s",i+1,ss );
             lb->InsertString(-1,s);
             md = (lpMID)((char *)md + sizeof(MID));
          }
       } else {
          lb->InsertString(-1,"No messages");
       }

       lb->UpdateWindow();
       //Sleep(200);

       md = (lpMID) b.mdata;
       lpOFORM po = (lpOFORM)B.mdata;
       for (i = 0; i < matches ; i++) {
          memcpy(B.mdata,md,sizeof(MID)); // Copy one mid

          s.Format("%d Messages: Reading %d",matches,i);
          SetDlgItemText(IDC_SERSTAT,s);
          GetDlgItem(IDC_SERSTAT)->UpdateWindow();

          if (QSUCCESS == QsendAndReceive(QS[m_id],ADMINREQ_MODE,QADM_REQ_MSG,
             0,sizeof(mid),B.mdata,
             sizeof(B.mdata),B.mdata,&sz,&B.msgh)) {

             ss.Format("%s(%d,%d)", IP2Name(md->host),md->uid,md->tid);
             if (B.msgh.size == sizeof(OFORM))
                s.Format("%4d \t%s\t%-8s\t%2d %-10s",i + 1,ss,po->cust,po->qty,po
             else if (sz)
                s.Format("%4d \t%s\t%s",i + 1,ss,b.mdata);
             else
                s.Format("%4d \t%s\t<<Empty>>",i + 1,ss);

             lb->DeleteString(i);
             lb->InsertString(i,s);
          } else {
             s.Format("%d Error requesting %s(%d,%d)",i + 1,IP2Name(md->host),md-
             SetDlgItemText(IDC_SERSTAT,s);
             break;  // Stop here
          }
          md = (lpMID)((char *)md + sizeof(MID));
       }

    } else
      s.Format("Error - no reply") ;

    if (!strstr(LPCTSTR(s),"Error")) s.Format("%d Matches:",matches);
```

```
   SetDlgItemText(IDC_SERSTAT,s);


   Sleep(500);
   GetDlgItem(IDSEARCHB)->EnableWindow(TRUE);
}



void CKeySearch::OnTimer(UINT nIDEvent)
{
        if (nIDEvent == KEYTIMER) {
       int  com,uncom;
       CString s;

       if ((g_pic[m_id] != m_pic)) {
          GetDlgItem(IDC_KPICS[g_pic[m_id]])->ShowWindow(SW_SHOW);
          GetDlgItem(IDC_KPICS[m_pic])->ShowWindow(SW_HIDE);
          m_pic = g_pic[m_id];

          // Set Title
          this->SetWindowText("View " + g_que[m_id] );
       }

       if (m_committed != (com = g_s[m_id].committed_entries)) {
          s.Format("%3d Committed entries.", (m_committed = com));
          SetDlgItemText(IDC_COM_R,s);
       }
       if (m_uncommitted != (uncom = g_s[m_id].pending_gets + g_s[m_id].pending_p
          s.Format("%3d Uncommitted entries.",(m_uncommitted = uncom));
          SetDlgItemText(IDC_UNCOM_R,s);
       }
       if (m_total_entries != (com + uncom)) {
          s.Format("All %d entries.",       (m_total_entries = com + uncom));
          SetDlgItemText(IDC_ALL_R,s);
       }
   } else
        CDialog::OnTimer(nIDEvent);
}


void CKeySearch::OnAllR()
{ m_search_type = SEARCH_ALL_ENT;}
void CKeySearch::OnComR()
{ m_search_type = SEARCH_COM_ENT;}
void CKeySearch::OnUncomR()
{ m_search_type = SEARCH_UNCOM_ENT;}

void CKeySearch::OnRButtonDown(UINT nFlags, CPoint point)
{
   GetParentFrame()->SetMessageText("");
   this->Invalidate();


        CDialog::OnRButtonDown(nFlags, point);
}
```

```
// qman.cpp : Defines the class behaviors for the application.
//

#include "stdafx.h"
#include "qman.h"

#include "mainfrm.h"
#include "qmandoc.h"
#include "qmanview.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CQmanApp

BEGIN_MESSAGE_MAP(CQmanApp, CWinApp)
        //{{AFX_MSG_MAP(CQmanApp)
        ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
                // NOTE - the ClassWizard will add and remove mapping macros her
                //    DO NOT EDIT what you see in these blocks of generated code
        //}}AFX_MSG_MAP
        // Standard file based document commands
        ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
        ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
        // Standard print setup command
        ON_COMMAND(ID_FILE_PRINT_SETUP, CWinApp::OnFilePrintSetup)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CQmanApp construction

CQmanApp::CQmanApp()
{
        // TODO: add construction code here,
        // Place all significant initialization in InitInstance
}

/////////////////////////////////////////////////////////////////////////////
// The one and only CQmanApp object

CQmanApp theApp;

/////////////////////////////////////////////////////////////////////////////
// CQmanApp initialization

BOOL CQmanApp::InitInstance()
{
        // Standard initialization
        // If you are not using these features and wish to reduce the size
        //  of your final executable, you should remove from the following
        //  the specific initialization routines you do not need.

        Enable3dControls();

        LoadStdProfileSettings();  // Load standard INI file options (including

        // Register the application's document templates.  Document templates
        //  serve as the connection between documents, frame windows and views.

        CSingleDocTemplate* pDocTemplate;
        pDocTemplate = new CSingleDocTemplate(
                IDR_MAINFRAME,
                RUNTIME_CLASS(CQmanDoc),
                RUNTIME_CLASS(CMainFrame),       // main SDI frame window
                RUNTIME_CLASS(CQmanView));
        AddDocTemplate(pDocTemplate);

        // create a new (empty) document
        OnFileNew();

        if (m_lpCmdLine[0] != '\0')
        {
                // TODO: add command line processing here
        }

        return TRUE;
}

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
        CAboutDlg();

// Dialog Data
        //{{AFX_DATA(CAboutDlg)
        enum { IDD = IDD_ABOUTBOX };
        //}}AFX_DATA


   CFont m_title_font;
// Implementation
protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        //{{AFX_MSG(CAboutDlg)
        virtual BOOL OnInitDialog();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
        //{{AFX_DATA_INIT(CAboutDlg)
        //}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CAboutDlg)
        //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
        //{{AFX_MSG_MAP(CAboutDlg)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

// App command to run the dialog
void CQmanApp::OnAppAbout()
{
        CAboutDlg aboutDlg;
        aboutDlg.DoModal();
}

/////////////////////////////////////////////////////////////////////////////
// CQmanApp commands

BOOL CAboutDlg::OnInitDialog()
{
        CDialog::OnInitDialog();


   LOGFONT lf;
   memset(&lf,0,sizeof(LOGFONT));
        strcpy(lf.lfFaceName,"Monotype Corsiva");
   lf.lfHeight = 24;
   m_title_font.CreateFontIndirect(&lf);
        GetDlgItem(IDC_ABOUT1)->SetFont(&m_title_font);


        // TODO: Add extra initialization here

        return TRUE;  // return TRUE unless you set the focus to a control
                      // EXCEPTION: OCX Property Pages should return FALSE
}
```

```
// qman.h : main header file for the QMAN application
//

#ifndef __AFXWIN_H__
        #error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"       // main symbols

/////////////////////////////////////////////////////////////////////////////
// CQmanApp:
// See qman.cpp for the implementation of this class
//

class CQmanApp : public CWinApp
{
public:
        CQmanApp();

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CQmanApp)
        public:
        virtual BOOL InitInstance();
        //}}AFX_VIRTUAL

// Implementation

        //{{AFX_MSG(CQmanApp)
        afx_msg void OnAppAbout();
                // NOTE - the ClassWizard will add and remove member function
                //    DO NOT EDIT what you see in these blocks of generated code
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};


/////////////////////////////////////////////////////////////////////////////
```

```
// qmandoc.cpp : implementation of the CQmanDoc class
//

#include "stdafx.h"
#include "qman.h"

#include "qmandoc.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CQmanDoc

IMPLEMENT_DYNCREATE(CQmanDoc, CDocument)

BEGIN_MESSAGE_MAP(CQmanDoc, CDocument)
        //{{AFX_MSG_MAP(CQmanDoc)
                // NOTE - the ClassWizard will add and remove mapping macros her
                //    DO NOT EDIT what you see in these blocks of generated code
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CQmanDoc construction/destruction

CQmanDoc::CQmanDoc()
{
        // TODO: add one-time construction code here

}

CQmanDoc::~CQmanDoc()
{
}

BOOL CQmanDoc::OnNewDocument()
{
        if (!CDocument::OnNewDocument())
                return FALSE;

        // TODO: add reinitialization code here
        // (SDI documents will reuse this document)

        return TRUE;
}

/////////////////////////////////////////////////////////////////////////////
// CQmanDoc serialization

void CQmanDoc::Serialize(CArchive& ar)
{
        if (ar.IsStoring())
        {
                // TODO: add storing code here
        }
        else
        {
                // TODO: add loading code here
        }
}

/////////////////////////////////////////////////////////////////////////////
// CQmanDoc diagnostics

#ifdef _DEBUG
void CQmanDoc::AssertValid() const
{
        CDocument::AssertValid();
}

void CQmanDoc::Dump(CDumpContext& dc) const
{
        CDocument::Dump(dc);
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CQmanDoc commands
// qmandoc.h : interface of the CQmanDoc class
//
/////////////////////////////////////////////////////////////////////////////

class CQmanDoc : public CDocument
{
protected: // create from serialization only
        CQmanDoc();
        DECLARE_DYNCREATE(CQmanDoc)

// Attributes
public:

// Operations
public:

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CQmanDoc)
        public:
        virtual BOOL OnNewDocument();
        //}}AFX_VIRTUAL

// Implementation
public:
        virtual ~CQmanDoc();
        virtual void Serialize(CArchive& ar);   // overridden for document i/o
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
        //{{AFX_MSG(CQmanDoc)
                // NOTE - the ClassWizard will add and remove member functions h
                //    DO NOT EDIT what you see in these blocks of generated code
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////
```

```
// qmanview.h : interface of the CQmanView class
//
/////////////////////////////////////////////////////////////////////////////

class CQmanView : public CFormView
{
protected: // create from serialization only
        CQmanView();
        DECLARE_DYNCREATE(CQmanView)

public:
        //{{AFX_DATA(CQmanView)
        enum{ IDD = IDD_QMAN_FORM };
                // NOTE: the ClassWizard will add data members here
        //}}AFX_DATA

   // Attributes
   int m_maxtrack;
   CString m_que[3];

public:
        CQmanDoc* GetDocument();

// Operations
public:

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CQmanView)
        public:
        virtual void OnInitialUpdate();
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
        virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
        virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);
        virtual void OnPrint(CDC* pDC, CPrintInfo*);
        virtual void OnDraw(CDC* pDC);
        //}}AFX_VIRTUAL
   //void InitTrackBar(HWND hTrack, int IDMIN, int TMIN, int IDMAX, int TMAX, in
   //DWORD Poll();
   //DWORD CQmanView::Poll();
   void CQmanView::OpenQue(int i, int IDC_QUES);

   // meter painting
   void DrawKey();
   void GetMeterBoxes();
   void TestMeters();
   void CQmanView::LoadList(int QN, int IDC_QUES);
   void CQmanView::DrawMeter(int i);
   void CQmanView::CmdLine(int pass);
   void CQmanView::DrawKeyColor(int ID, int HS, COLORREF COL);

   // font
   CFont m_title_font;

// Implementation
public:
        virtual ~CQmanView();
#ifdef _DEBUG
```

```
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
        //{{AFX_MSG(CQmanView)
        afx_msg void OnExit();
        afx_msg void OnHScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar);
        afx_msg void OnSelchangeQues1();
        afx_msg void OnSelchangeQues2();
        afx_msg void OnSelchangeQues3();
        afx_msg void OnAdminb1();
        afx_msg void OnAdminb2();
        afx_msg void OnAdminb3();
        afx_msg void OnDatab1();
        afx_msg void OnDatab2();
        afx_msg void OnDatab3();
        afx_msg void OnTimer(UINT nIDEvent);
        afx_msg void OnRButtonDown(UINT nFlags, CPoint point);
        afx_msg void OnSetfocusQues1();
        afx_msg void OnSetfocusQues2();
        afx_msg void OnSetfocusQues3();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

#ifndef _DEBUG  // debug version in qmanview.cpp
inline CQmanDoc* CQmanView::GetDocument()
   { return (CQmanDoc*)m_pDocument; }
#endif

////////////////////////////////////////////////////////////////////////////
```

```
// qmanview.cpp : implementation of the CQmanView class
//


#include "stdafx.h"
#include "qman.h"

#include "qmandoc.h"
#include "qmanview.h"
#include "admindlg.h"
//#include "datadlg.h"
#include "KeySearch.h"

 //+++++++++++++ QLIB ++++++++++++++++++
#include "qlib.h"
#include "qadmin.h"
#include "rt.h"



#define  MYTIMER 100
#define  TITLETIMER 101

#define  MIN_POLL_DLY 100
#define  MAX_POLL_DLY 5000


extern   lpSMBUFH sm_base;
lpQHANDLE   QS[3] = {NULL,NULL,NULL};
QADMSTATS   g_s[3+3]; // 3 Current + 3 Old
int         g_starting[3]; // forces first N updates of meter

//HWND g_track[3];
HWND g_tmin[3];
HWND g_tmax[3];
HWND g_tlab[3];
CString g_que[]   = {"","",""};

int  ALL_TEXT[] = {IDC_TMIN1,IDC_TMIN2,IDC_TMIN3,
                   IDC_TMAX1,IDC_TMAX2,IDC_TMAX3,
                   IDC_TLAB1,IDC_TLAB2,IDC_TLAB3,
                   IDC_ADMINB1,IDC_ADMINB2,IDC_ADMINB3,
                   IDC_DATAB1,IDC_DATAB2,IDC_DATAB3,
                   IDC_QUES1,IDC_QUES2,IDC_QUES3,
                   IDC_EXIT,0,0,0};


int  IDC_TMINS[]   = {IDC_TMIN1,IDC_TMIN2,IDC_TMIN3};
int  IDC_TMAXS[]   = {IDC_TMAX1,IDC_TMAX2,IDC_TMAX3};
int  IDC_TLABS[]   = {IDC_TLAB1,IDC_TLAB2,IDC_TLAB3};
int  IDC_METERS[]  = {IDC_METER1,IDC_METER2,IDC_METER3};
int  IDC_PICS[3][8]  = {{IDC_QNONE1,IDC_QDOWN1,IDC_QSTOP1,IDC_QNOPUT1,IDC_QNOGET1
                        {IDC_QNONE4,IDC_QDOWN4,IDC_QSTOP4,IDC_QNOPUT4,IDC_QNOGET4
                        {IDC_QNONE5,IDC_QDOWN5,IDC_QSTOP5,IDC_QNOPUT5,IDC_QNOGET5
enum pics {QNONE,QDOWN,QSTOP,QNOPUT,QNOGET,QUP,QNOPG,QFULL};
int  g_pic[3+3+3];
int  g_poll;
int  g_poll_delay = 2000;
CFont g_text_font;
```

```
typedef struct met {  // Thread parameters
   CRect    b_rec;
   CRect    c_rec;
   CRect    p_rec;
   CRect    g_rec;
   CRect    h_rec;
   CRect    f_rec;
   int      commit,pendp,pendg,hole,free,min,max;
} MET, *pMET;

MET g_met[3+4];



//+++++++++++++ QLIB ++++++++++++++++++


#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CQmanView

IMPLEMENT_DYNCREATE(CQmanView, CFormView)

BEGIN_MESSAGE_MAP(CQmanView, CFormView)
        //{{AFX_MSG_MAP(CQmanView)
        ON_BN_CLICKED(IDC_EXIT, OnExit)
        ON_WM_HSCROLL()
        ON_CBN_SELCHANGE(IDC_QUES1, OnSelchangeQues1)
        ON_CBN_SELCHANGE(IDC_QUES2, OnSelchangeQues2)
        ON_CBN_SELCHANGE(IDC_QUES3, OnSelchangeQues3)
        ON_BN_CLICKED(IDC_ADMINB1, OnAdminb1)
        ON_BN_CLICKED(IDC_ADMINB2, OnAdminb2)
        ON_BN_CLICKED(IDC_ADMINB3, OnAdminb3)
        ON_BN_CLICKED(IDC_DATAB1, OnDatab1)
        ON_BN_CLICKED(IDC_DATAB2, OnDatab2)
        ON_BN_CLICKED(IDC_DATAB3, OnDatab3)
        ON_WM_TIMER()
        ON_WM_RBUTTONDOWN()
        ON_CBN_SETFOCUS(IDC_QUES1, OnSetfocusQues1)
        ON_CBN_SETFOCUS(IDC_QUES2, OnSetfocusQues2)
        ON_CBN_SETFOCUS(IDC_QUES3, OnSetfocusQues3)
        //}}AFX_MSG_MAP
        // Standard printing commands
        ON_COMMAND(ID_FILE_PRINT, CFormView::OnFilePrint)
        ON_COMMAND(ID_FILE_PRINT_PREVIEW, CFormView::OnFilePrintPreview)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CQmanView construction/destruction

CQmanView::CQmanView()
        : CFormView(CQmanView::IDD)
{
        //{{AFX_DATA_INIT(CQmanView)
                // NOTE: the ClassWizard will add member initialization here
```

```
	//}}AFX_DATA_INIT
	// TODO: add construction code here
}

CQmanView::~CQmanView()
{
}

void CQmanView::DoDataExchange(CDataExchange* pDX)
{
	CFormView::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CQmanView)
		// NOTE: the ClassWizard will add DDX and DDV calls here
	//}}AFX_DATA_MAP
}

/////////////////////////////////////////////////////////////////////////////
// CQmanView printing

BOOL CQmanView::OnPreparePrinting(CPrintInfo* pInfo)
{
	// default preparation
	return DoPreparePrinting(pInfo);
}

void CQmanView::OnBeginPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
	// TODO: add extra initialization before printing
}

void CQmanView::OnEndPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
	// TODO: add cleanup after printing
}

void CQmanView::OnPrint(CDC* pDC, CPrintInfo*)
{
	// TODO: add code to print the controls
}

/////////////////////////////////////////////////////////////////////////////
// CQmanView diagnostics

#ifdef _DEBUG
void CQmanView::AssertValid() const
{
	CFormView::AssertValid();
}

void CQmanView::Dump(CDumpContext& dc) const
{
	CFormView::Dump(dc);
}

CQmanDoc* CQmanView::GetDocument() // non-debug version is inline
{
	ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CQmanDoc)));
	return (CQmanDoc*)m_pDocument;
}
#endif //_DEBUG
```

```
/////////////////////////////////////////////////////////////////////////////
// CQmanView message handlers

void CQmanView::OnExit()
{
   g_poll = 0;   // Stop the threads
   AfxGetMainWnd()->DestroyWindow();
}


// METER METER METER METER METER METER METER METER METER METER METER METER
// METER METER METER METER METER METER METER METER METER METER METER METER
// METER METER METER METER METER METER METER METER METER METER METER METER
//
//                  b_rec
// +---------+---------+---------+---------+
// | Committ | Pending | holes   | Free    |
// |         |         |         |         |
// +---------+---------+---------+---------+
//  c_rec      p_rec      h_rec    f_rec


void CQmanView::GetMeterBoxes(){
   for (int i=0;i<3;i++){
        GetDlgItem(IDC_METERS[i])->GetWindowRect(&g_met[i].b_rec);
      // Adjust coordantates for 0,0,l,b
      g_met[i].b_rec.right  -= g_met[i].b_rec.left;
      g_met[i].b_rec.bottom -= g_met[i].b_rec.top;
      g_met[i].b_rec.left    = g_met[i].b_rec.top = 0;

      g_met[i].c_rec.left    = 0; // Commit always starts at 0,0
      g_met[i].f_rec.right   = g_met[i].b_rec.right; // Free always ends at max

      g_met[i].c_rec.top     =
      g_met[i].p_rec.top     =
      g_met[i].g_rec.top     =
      g_met[i].h_rec.top     =
      g_met[i].f_rec.top     = g_met[i].b_rec.top; // 0

      g_met[i].c_rec.bottom  =
      g_met[i].p_rec.bottom  =
      g_met[i].g_rec.bottom  =
      g_met[i].h_rec.bottom  =
      g_met[i].f_rec.bottom  = g_met[i].b_rec.bottom; // All the same hight


   }
}


void CQmanView::DrawKeyColor(int ID, int HS, COLORREF COL){
   CRect    c_rec;
   CBrush *pCBrush;

   if ( HS == -1 )
      pCBrush = new CBrush(COL);
   else
      pCBrush = new CBrush(HS,COL);
   GetDlgItem(ID)->GetWindowRect(&c_rec);

   // Adjust coordantates for 0,0,l,b
      c_rec.right  -= c_rec.left;
      c_rec.bottom -= c_rec.top;
      c_rec.left    = c_rec.top = 0;

      // The DC for the meter
   CDC* pCOLORDC = GetDlgItem(ID)->GetDC();

   // Select this brush, save the old
   CBrush *pOldBrush =
   pCOLORDC->SelectObject(pCBrush);
   pCOLORDC->Rectangle(c_rec);

   GetDlgItem(ID)->Invalidate();
   delete(pCBrush);
   ReleaseDC(pCOLORDC);
}


void CQmanView::DrawKey(){

   DrawKeyColor(IDC_KEY_COM,     -1,            RGB(127,255,255));
   DrawKeyColor(IDC_KEY_PENPUT, HS_BDIAGONAL, RGB(000,182,255));
   DrawKeyColor(IDC_KEY_PENGET, HS_FDIAGONAL, RGB(000,182,255));
   DrawKeyColor(IDC_KEY_HOLE,   HS_DIAGCROSS, RGB(255,128,128));
}



#define MAXPIXELS g_met[i].b_rec.right
void CQmanView::TestMeters(){
   int c,pg,pp,h; // Pixel width of each rectangle

   for (int i=0;i<3;i++) { // If any change
      if((g_met[i].commit  != g_s[i].committed_entries) ||
         (g_met[i].pendg   != g_s[i].pending_gets ) ||
         (g_met[i].pendp   != g_s[i].pending_puts) ||
         (g_met[i].hole    != g_s[i].holey_entries ) ||
         (g_met[i].free    != g_s[i].num_free_entries) ||
         (g_met[i].max     != g_s[i].max_entries) ) {

         if (!g_s[i].max_entries) g_s[i].max_entries=1; // No divide by zero

         // Update old values
         g_met[i].commit  = g_s[i].committed_entries;
         g_met[i].pendg   = g_s[i].pending_gets;
         g_met[i].pendp   = g_s[i].pending_puts;
         g_met[i].hole    = g_s[i].holey_entries;
         g_met[i].free    = g_s[i].num_free_entries;
         g_met[i].max     = g_s[i].max_entries;

         // c,pp,pg,h,f are points on a line between min, max scaled to #pixe
         c = 0 + (MAXPIXELS * g_met[i].commit)/g_met[i].max;
         pp= c + (MAXPIXELS * g_met[i].pendp)/g_met[i].max;
         pg=pp + (MAXPIXELS * g_met[i].pendg)/g_met[i].max;
         h =pg + (MAXPIXELS * g_met[i].hole)/g_met[i].max;
      // f = h + (MAXPIXELS * g_met[i].free)/g_met[i].max; // constant

         // Make 5 rectangles to fill with colors
      // g_met[i].c_rec.left = 0; // constant
         g_met[i].c_rec.right = c;

         g_met[i].p_rec.left  = c;
         g_met[i].p_rec.right = pp;

         g_met[i].g_rec.left  = pp;
         g_met[i].g_rec.right = pg;

         g_met[i].h_rec.left  = pg;
         g_met[i].h_rec.right = h;

         g_met[i].f_rec.left = h;
      // g_met[i].f_rec.right = g_met[i].max; // constant

         DrawMeter(i);
      }
   }
}

void CQmanView::DrawMeter(int i){

   // Create the Brush
   CBrush *pCBrush = new CBrush(                RGB(127,255,255) );
   CBrush *pPBrush = new CBrush( HS_BDIAGONAL, RGB(000,182,255) );
   CBrush *pGBrush = new CBrush( HS_FDIAGONAL, RGB(000,182,255) );
   CBrush *pHBrush = new CBrush( HS_DIAGCROSS, RGB(255,128,128) );
        // The DC for the meter
   CDC* pCOLORDC = GetDlgItem(IDC_METERS[i])->GetDC();

  // Create a PEN
  // CPen *pQPen = new CPen(PS_SOLID, 3,RGB(0,0,255));
  // CPen *pOldPen = pXDC->SelectObject(pQPen);

   // Select this brush, save the old
   CBrush *pOldBrush =
   pCOLORDC->SelectObject(pCBrush);
   pCOLORDC->Rectangle(g_met[i].c_rec);

   pCOLORDC->SelectObject(pPBrush);
   pCOLORDC->Rectangle(g_met[i].p_rec);

   pCOLORDC->SelectObject(pGBrush);
   pCOLORDC->Rectangle(g_met[i].g_rec);

   pCOLORDC->SelectObject(pHBrush);
   pCOLORDC->Rectangle(g_met[i].h_rec);

   pCOLORDC->SelectStockObject(WHITE_BRUSH);
   pCOLORDC->Rectangle(g_met[i].f_rec);


   pCOLORDC->SelectObject(&pOldBrush);  // Reset the brush
   GetDlgItem(IDC_METERS[i])->Invalidate();

   delete(pCBrush);
   delete(pPBrush);
   delete(pGBrush);
   delete(pHBrush);
   ReleaseDC(pCOLORDC);

}

/*     //////////////////////////////////////
       // Init the track bar              //
       //  0          TPS          100   //
       //  |------------[]------------|  //
       // MIN                        MAX //
       //////////////////////////////////////


void InitTrackBar(HWND hTrack, HWND HMIN, int TMIN, HWND HMAX, int TMAX) {
   CString s;

   //HWND hTrack = GetDlgItem(ID)->m_hWnd;
   ::SendMessage(hTrack,TBM_SETRANGEMIN,TRUE,TMIN);
   ::SendMessage(hTrack,TBM_SETRANGEMAX,TRUE,TMAX);
   ::SendMessage(hTrack,TBM_SETTICFREQ,1,TRUE);
   ::SendMessage(hTrack,TBM_SETPOS,TRUE,TMIN);
   ::SendMessage(hTrack,TBM_SETSELSTART,TRUE,TMIN); // Select from start

   // Track Bar lables
  //s.Format("%d",TMIN); SetDlgItemText(IDMIN,s);
  // s.Format("%d",TMAX); SetDlgItemText(IDMAX,s);
   s.Format("%d",TMIN);  ::SendMessage(HMIN,WM_SETTEXT,0,(LPARAM)(LPCTSTR) s);
   s.Format("%d",TMAX);  ::SendMessage(HMAX,WM_SETTEXT,0,(LPARAM)(LPCTSTR) s);

}
*/


DWORD Poll(LPVOID qnum)
{
   MSGH mh;

   CString s,s1;
   int used = 1; // Clear the track bar if not in use
   int i = (int)(qnum);
   int repeated_error = 0;

   g_poll = 1;
   while(g_poll) {

      if (QS[i]) {
         int  sz;

         used++;
         if (QS[i]->open_time)
            ::SendMessage(g_tlab[i],WM_SETTEXT,0,(LPARAM)(LPCTSTR) "STARTING..."

         if (QSUCCESS >= (QsendAndReceive(QS[i],ADMINREQ_MODE,QADM_REQ_STATS, 0,
               sizeof(g_s[i]),(char *)&g_s[i],&sz,&mh))){
            if ((mh.mode == ADMINREP_MODE || mh.mode == ACK_MODE ) && (sz > 0
                                    // Note: now, mode is ADMINREP_MODE if local c

               if (!g_starting[i] && !memcmp( &g_s[i+3], &g_s[i],sizeof(QADMSTAT
                  goto skip;
            memcpy( &g_s[i+3], &g_s[i],sizeof(QADMSTATS));  // set history=cu
            if (g_starting[i]) g_starting[i]--;             // stop forcing m
            repeated_error = 0;

            // Assign a picture
            if (g_s[i].qget_state && g_s[i].qput_state)
               g_pic[i] = QUP;
            else if (g_s[i].qget_state)
               g_pic[i] = QNOPUT;
            else if (g_s[i].qput_state)
               g_pic[i] = QNOGET;
            else
               g_pic[i] = QNOPG;

            if (g_s[i].num_free_entries == 0)
               g_pic[i] = QFULL;


            if (QS[i]->open_time) QS[i]->open_time = 0;  // Get rid of "START

             s.Format("%s:%s at %s has %d entries",&g_s[i].physical_qname,&g_
             s1.Format("%d",g_s[i].max_entries);
            ::SendMessage(g_tlab[i],WM_SETTEXT,0,(LPARAM)(LPCTSTR) s);
            ::SendMessage(g_tmax[i],WM_SETTEXT,0,(LPARAM)(LPCTSTR) s1);
            ::SendMessage(g_tmin[i],WM_SETTEXT,0,(LPARAM)(LPCTSTR) "0");

         } else {
            ::SendMessage(g_tlab[i],WM_SETTEXT,0,(LPARAM)(LPCTSTR) "BAD REPLY
            // QS[i] = NULL;
            if (repeated_error++ > 0) {
               g_pic[i] = QDOWN;
               Sleep(5000);
            } else
                g_pic[i] = QSTOP;
         }
      } else {
         ::SendMessage(g_tlab[i],WM_SETTEXT,0,(LPARAM)(LPCTSTR) "COMM ERROR")
        // QS[i] = NULL;
         if (repeated_error++ > 0) {
            g_pic[i] = QDOWN;
            Sleep(5000);
         } else
            g_pic[i] = QSTOP;

      }
   } else { // No open que
     // if (g_que[i] == "")
    //     g_pic[i] = QNONE;
      if (used) {
         used = 0; // The track bar will now be clear
         memset(&g_s[i],0,sizeof(g_s[i]));
         g_s[i+3].max_entries = 1;
         //::SendMessage(g_track[i],TBM_SETPOS,TRUE,0);
         //::SendMessage(g_track[i],TBM_SETSELEND,TRUE,0);
         ::SendMessage(g_tlab[i],WM_SETTEXT,0,(LPARAM)(LPCTSTR) "No Que");
         ::SendMessage(g_tmin[i],WM_SETTEXT,0,(LPARAM)(LPCTSTR) "");
         ::SendMessage(g_tmax[i],WM_SETTEXT,0,(LPARAM)(LPCTSTR) "");
      }
   }
skip: Sleep(g_poll_delay);
```

```
   } // While loop
   return(1);
}


void CQmanView::CmdLine(int pass){
   CString parm,value,line,nline;
   int i,poll;

   line = AfxGetApp()->m_lpCmdLine;

   while (2 <= (i = sscanf(LPCTSTR(line),"%s %s %[^@]",
      parm.GetBuffer(100),value.GetBuffer(100),nline.GetBuffer(100) ))) {
      parm.MakeUpper();

      if (parm == "POLL") {
         sscanf(LPCTSTR(value),"%d",&poll);
         if((MIN_POLL_DLY <= poll) && (poll <= MAX_POLL_DLY))
            g_poll_delay  = poll;
      }

      if (parm == "1") {
         CComboBox * CB = (CComboBox *) this->GetDlgItem(IDC_QUES1);
         CB->SelectString(-1,value);
         OnSelchangeQues1();
      }
      if (parm == "2") {
         CComboBox * CB = (CComboBox *) this->GetDlgItem(IDC_QUES2);
         CB->SelectString(-1,value);
         OnSelchangeQues2();
      }
      if (parm == "3") {
         CComboBox * CB = (CComboBox *) this->GetDlgItem(IDC_QUES3);
         CB->SelectString(-1,value);
         OnSelchangeQues3();
      }

      line = LPCTSTR(nline);
      nline = "";
   }
}


int g_poll_delay_old;
void CQmanView::OnTimer(UINT nIDEvent)
{

// Select ICONS
        if (nIDEvent == MYTIMER){
      for (int i=0;i<3;i++){

         if (g_pic[i] != g_pic[i+3]){
            GetDlgItem(IDC_PICS[i] [g_pic[i]])->ShowWindow(SW_SHOW);
            GetDlgItem(IDC_PICS[i] [g_pic[i+3]])->ShowWindow(SW_HIDE);
            g_pic[i+3] = g_pic[i];
            }
        }
        TestMeters();

           } else if (nIDEvent == TITLETIMER) {
        GetParentFrame()->SetWindowText("QMAN");
        KillTimer(TITLETIMER);
        DrawKey();
           } else
           CFormView::OnTimer(nIDEvent);
}



void CQmanView::LoadList(int QN, int IDC_QL)  // Called every time the user pick
{
        CString s;
    int i;

    // LIST OF QUEUES   From Routing table and what is in Shared Memory.

    // CListBox* lb = (CListBox*) GetDlgItem(IDC_QUE);
    // lb->InsertString(-1,"Q1");
    // lb->InsertString(-1,"Q2");
    // lb->SetCurSel(0);

    CComboBox * CB1 = (CComboBox *) this->GetDlgItem(IDC_QL);
    CB1->ResetContent();


// List all logical queues from the Routing Table
// APPS: [physical],logical1,logical2,[physical],logical,
    if (sm_base = AttachSharedMemory()){
        lpRT  rt = RTROOT;
        while(rt = NextRT(rt)) {
            char *e,*s = RT_APPS(rt);     // Starts after the first letter
            while (s = strchr(s,',')) {  // Ends at next ","
                if (e = strchr(++s,',')) {
                    *e = 0;
                    if ((!strchr(s,'['))  && *s) {
                        CB1->AddString(s); // lb->InsertString(-1,s)
                    }
                    *e = ',';
                }
            }
        }
        // Look at shared memory (physical Q names) for "Q"s
        for ( i=0; i < SHAREDATA(nsbuf); i++ ){
            lpSMBUF b = SMBUFADDR(i);
            if ( strcmp(b->name,&"QNETD") && strchr(b->name,'Q') ) { // Anything th
                if (CB1->FindStringExact(-1,b->name) == CB_ERR) {  // String is not
                    CB1->AddString(b->name); // lb->InsertString(-1,s)
                }
            }
        }
    } else {
        GetParentFrame()->SetMessageText("QNETD not running(?) please start it.");
        MessageBox("QNETD not running, please start it.",0,MB_ICONSTOP);
        //AfxGetMainWnd()->DestroyWindow();
    }
    CB1->AddString("");

    // CB1->SetCurSel(g_que[QN]);
    CB1->SelectString(-1, g_que[QN]);
}



void CQmanView::OnInitialUpdate()
{
        LOGFONT lf;
        CString s;
        CFormView::OnInitialUpdate(); // Default from vc++

    // Set frame size = Form size
    GetParentFrame()->RecalcLayout();
    ResizeParentToFit(FALSE);
    ResizeParentToFit(TRUE);



    GetMeterBoxes();

    DrawKey();

    for (int i=0;i<3;i++){

 //     g_track[i] = GetDlgItem(IDC_TRACKS[i])->m_hWnd;
      g_tmin[i] = GetDlgItem(IDC_TMINS[i])->m_hWnd;
      g_tmax[i] = GetDlgItem(IDC_TMAXS[i])->m_hWnd;
      g_tlab[i] = GetDlgItem(IDC_TLABS[i])->m_hWnd;
      g_pic[i] = QNONE;    // man
      g_pic[i+3] = QSTOP;  // yellow
    }


    // Add selections in the list of quenames (needed now, so the CmdLine args wi
    LoadList(0,IDC_QUES1);
    LoadList(1,IDC_QUES2);
    LoadList(2,IDC_QUES3);


    DWORD id;
    CreateThread(NULL,0,(LPTHREAD_START_ROUTINE) Poll,(LPVOID) 0,0,&id);
    CreateThread(NULL,0,(LPTHREAD_START_ROUTINE) Poll,(LPVOID) 1,0,&id);
    CreateThread(NULL,0,(LPTHREAD_START_ROUTINE) Poll,(LPVOID) 2,0,&id);

    CmdLine(1);


    SetTimer(MYTIMER,250,NULL); // 1/4 second
    SetTimer(TITLETIMER,100,NULL);


    // Fonts
    memset(&lf,0,sizeof(LOGFONT));
#ifdef BIGFONT
    lf.lfHeight = 18;
```

```
#else
   lf.lfHeight = 13;
#endif
   g_text_font.CreateFontIndirect(&lf);
        strcpy(lf.lfFaceName,"Matura MT Script Capitals");
        strcpy(lf.lfFaceName,"Monotype Corsiva");
   lf.lfHeight = 32;
   m_title_font.CreateFontIndirect(&lf);


        GetDlgItem(IDC_TITLE)->SetFont(&m_title_font);

   i = 0;
   while (ALL_TEXT[i])
      GetDlgItem(ALL_TEXT[i++])->SetFont(&g_text_font);


}

void CQmanView::OnHScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar)
{
        // TODO: Add your message handler code here and/or call default

//      CFormView::OnHScroll(nSBCode, nPos, pScrollBar);  // Removed by derek
}


void CQmanView::OpenQue(int i, int IDC_QUES)
{
        CString s,que;
   int stat;

   GetDlgItemText(IDC_QUES,que.GetBuffer(100),100);
   que.ReleaseBuffer();

   if (que != g_que[i]) { // the user changed the open que name
      GetParentFrame()->SetWindowText("QMAN");

      if (que == "") {
         g_pic[i] = QNONE;
         g_que[i] = "";
         s = "No Que will be used";
         Qclose(&QS[i],0);
      } else {
         g_pic[i] = QSTOP;
         s.Format("Opening que %s",que);
         memset(&g_s[i],0,sizeof(QADMSTATS)); // clear
         g_starting[i] = 500; // Aprox 50 seconds.
         GetParentFrame()->SetMessageText(s); // MessageBox(s);  causes 2nd pass
         if (QS[i] = Qopen(que.GetBuffer(0),PUT_MODE,0,Q_FAILOVER,&stat,0,0) ) {
            g_que[i] = que;
            s.Format("Qopen(%s)",que);
  //           QS[i]->time_out = 1000; // 1 second
         } else {
            s.Format("Qopen(%s) Error %d",que,stat);
            g_pic[i] = QDOWN;

            CComboBox * CB = (CComboBox *) this->GetDlgItem(IDC_QUES);
            CB->SelectString(-1,"");
            g_que[i] = "";
            }
        }
        GetParentFrame()->SetMessageText(s);
    }
}


void CQmanView::OnSetfocusQues1(){ LoadList(0,IDC_QUES1);}
void CQmanView::OnSetfocusQues2(){ LoadList(1,IDC_QUES2);}
void CQmanView::OnSetfocusQues3(){ LoadList(2,IDC_QUES3);}

void CQmanView::OnSelchangeQues1(){      OpenQue(0,IDC_QUES1);}
void CQmanView::OnSelchangeQues2(){ OpenQue(1,IDC_QUES2);}
void CQmanView::OnSelchangeQues3(){ OpenQue(2,IDC_QUES3);}

void CallAdm(int id){
   if (QS[id]) {
          CAdminDlg adm;
      adm.m_id = id;
      adm.DoModal();
   }
}

void CallData(int id){
   if (QS[id]) {
          CKeySearch d;  // CdataDlg
      d.m_id = id;
      d.DoModal();
   }
}

void CQmanView::OnAdminb1()  { CallAdm(0); }
void CQmanView::OnAdminb2()  { CallAdm(1); }
void CQmanView::OnAdminb3()  { CallAdm(2); }


void CQmanView::OnDatab1()  { CallData(0); }
void CQmanView::OnDatab2()  { CallData(1); }
void CQmanView::OnDatab3()  { CallData(2); }

void CQmanView::OnDraw(CDC* pDC)
{
        DrawMeter(0);
        DrawMeter(1);
        DrawMeter(2);
   DrawKey();
        CFormView::OnDraw(pDC);
}

//      this->SetWindowText("Qman av");
//       GetParentFrame()->SetWindowText("QQMAN");


void CQmanView::OnRButtonDown(UINT nFlags, CPoint point)
{
   GetParentFrame()->SetMessageText("");
   this->Invalidate();
        CFormView::OnRButtonDown(nFlags, point);
}
```

```
// oentrvw.cpp : implementation of the COentryView class
//
/*
**
** Copyright(C)1995 MITSUBISHI ELECTRIC ITA.  ALL RIGHTS RESERVED.
** UNPUBLISHED -- RIGHTS RESERVED UNDER THE COPYRIGHT
** LAWS OF THE UNITED STATES.  USE OF A COPYRIGHT NOTICE
** IS PRECAUTIONARY ONLY AND DOES NOT IMPLY PUBLICATION
** OR DISCLOSURE.
**
** THIS SOFTWARE CONTAINS CONFIDENTIAL INFORMATION AND
** TRADE SECRETS OF MITSUBISHI ELECTRIC ITA.  USE, DISCLOSURE,
** OR REPRODUCTION IS PROHIBITED WITHOUT THE PRIOR
** EXPRESS WRITTEN PERMISSION OF MITSUBISHI ELECTRIC ITA.
**
** OpenMQ Demo
**   Module: oenrvw.cpp
**   Author: Derek Schwenke 10/8/95
**
*/

#include "stdafx.h"
#include "oentry.h"

#include "oentrdoc.h"
#include "dbdlg.h"
#include "odlg.h"

#include "oentrvw.h"
#include "orderfm.h"

#include "oraomq.h"


//+++++++++++++ QLIB ++++++++++++++++++
#include "qlib.h"
#include "rt.h"
extern   lpSMBUFH sm_base;
lpQHANDLE   Q,Qrep;
//+++++++++++++ QLIB ++++++++++++++++++
#define REPLY_SUBMODE      1
#define NOMSGQ_SUBMODE     2
#define DIRECT_FILL_TIME_OVER 121
#define PLACE_ORDER_NOQ    0
#define PLACE_ORDER        1
#define FILL_ORDER         2
#define FILL_FROM_ORACLE   3

#define DB_DONE_TIMER      201
#define WAIT_ANAMATE_TIMER 202
#define DIRECT_FILL_TIMER  203
#define FILL_DELAY_TIMER   204
#define FILL_ANAMATE_TIMER 205
#define SET_1ST_TITLE_TIMER 206
#define OPTIONS_DONE_TIMER 207
#define POLL_FILL_TIMER    208
#define PLACE_TIMER        209
```

```
#define NEXTBIT(X) (1 & ( X = X / 2 ))

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// COentryView

IMPLEMENT_DYNCREATE(COentryView, CFormView)

BEGIN_MESSAGE_MAP(COentryView, CFormView)
        //{{AFX_MSG_MAP(COentryView)
        ON_BN_CLICKED(IDC_EXITB, OnExitb)
        ON_BN_CLICKED(IDC_ORDERB, OnOrderb)
        ON_BN_CLICKED(IDC_AUTOB, OnAutob)
        ON_WM_CTLCOLOR()
        ON_BN_CLICKED(IDC_PLACER, OnPlacer)
        ON_BN_CLICKED(IDC_FILLR, OnFillr)
        ON_CBN_SELCHANGE(IDC_QUE, OnSelchangeQue)
   ON_MESSAGE(MESSAGEREADY,OnReplyMsg)
        ON_BN_CLICKED(IDC_SENDREPC, OnSendrepc)
        ON_BN_CLICKED(IDC_TRANB, OnTranb)
        ON_BN_CLICKED(IDC_COMMITB, OnCommitb)
        ON_BN_CLICKED(IDC_ABORTB, OnAbortb)
        ON_WM_LBUTTONDOWN()
        ON_WM_TIMER()
        ON_BN_CLICKED(IDC_SHOWDB, OnShowdb)
        ON_BN_CLICKED(IDC_FILLDB, OnFilldb)
        ON_BN_CLICKED(IDC_PLACENOQR, OnPlacenoqr)
        ON_CBN_EDITUPDATE(IDC_QUE, OnEditupdateQue)
        ON_BN_CLICKED(IDC_OPTIONSB, OnOptionsb)
        ON_WM_RBUTTONDOWN()
        //}}AFX_MSG_MAP
        // Standard printing commands
        ON_COMMAND(ID_FILE_PRINT, CFormView::OnFilePrint)
        ON_COMMAND(ID_FILE_PRINT_PREVIEW, CFormView::OnFilePrintPreview)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// COentryView construction/destruction

COentryView::COentryView()
        : CFormView(COentryView::IDD)
{
        //{{AFX_DATA_INIT(COentryView)
        m_OrderMode = 1;
        m_inst = _T("");
        m_sendreply = 0;
        //}}AFX_DATA_INIT
        // TODO: add construction code here
   m_runflag = 0;
   m_mes_sent = 0;
   m_mes_rec = 0;
   m_rec_sent = 0;
   m_rec_rec = 0;
   m_color = RGB(0,255,0);
```

000279

```
   m_box_color = m_color;
   m_que = "";
   m_order_num = 0;
   m_tran_state = 0;
   m_auto_tran = 0;

}


   CString g_cust[] = {"Jones","James","Johnson","Jacobs","Jaffe","Jackson"};
   CString g_item[] = {"Bolts","Buckets","Buttons","Belts","Bobbins","Boats"};
   int  g_price[] = {1,2,3,4,5,6};
   int  g_qty[]   = {1000,1000,1000,1000,1000,1000};
   int  g_purchases[] = {0,0,0,0,0,0};
   int  g_num_purchases[] = {0,0,0,0,0,0};
   int  g_total_sales = 0;
   int  g_db_run = 0;
   int  g_options_run = 0;
 //  int  g_delay = 0;
   int  g_directplace = 0; // Client
   int  g_wait_anamate = 0;  // Server
   int  g_direct_fill = 0;  // Server
   int  g_fill_anamate = 0;  // Server
   int  g_filling = 0;  // Server

   int  g_place_tpm   = 60;
   int  g_place_delay = 1000;

   int  g_fill_delay  = 0;
   int  g_poll_pps    = 10;
   int  g_poll_delay  = 100;
   int  g_clear_stats = 0;
   int  g_ora_state = 0;
   int  g_max_orders = 0;
   char g_oracle_con_str[80] = "scott/tiger@t:grampa:orcl";

   COLORREF g_new_color;
   CFont    g_title_font;
   CFont    g_text_font;

   int  WAITS[] = {IDC_WAIT0,IDC_WAIT1};
   int  FILLS[] = {IDC_FILL0,IDC_FILL1,IDC_FILL2};
   int  ALL_TEXT[] = { IDC_MODEBOX,IDC_PLACER,IDC_PLACENOQR,IDC_FILLR,IDC_FILLDB
   IDC_OPTIONSB, IDC_ORDERBOX,IDC_FILLTXT,IDC_SENDREPC,IDC_ORDERB,IDC_AUTOB,IDC_
   IDC_QUE,IDC_CUST,IDC_ITEM,IDC_QTY,
   IDC_QUELAB,IDC_CUSTLAB,IDC_ITEMLAB,IDC_QTYLAB,
   IDC_TRANBOX,IDC_TRANB,IDC_COMMITB,IDC_ABORTB,IDC_EXITB,
   IDC_STATBOX,IDC_RECIPTS_LAB,IDC_RECIPTS,IDC_RECIPT,IDC_MSGS_LAB,IDC_MSGS,0};


   lpQHANDLE g_LQ; // Listener que global for close OnExit()


COentryView::~COentryView()
{
}

void COentryView::DoDataExchange(CDataExchange* pDX)
{
        CFormView::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(COentryView)
        DDX_Radio(pDX, IDC_PLACER, m_OrderMode);
//      DDX_Text(pDX, IDC_INST, m_inst);
        DDX_Check(pDX, IDC_SENDREPC, m_sendreply);
        //}}AFX_DATA_MAP
}

/////////////////////////////////////////////////////////////////////////////
// COentryView printing

BOOL COentryView::OnPreparePrinting(CPrintInfo* pInfo)
{
        // default preparation
        return DoPreparePrinting(pInfo);
}

void COentryView::OnBeginPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
        // TODO: add extra initialization before printing
}

void COentryView::OnEndPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
        // TODO: add cleanup after printing
}

void COentryView::OnPrint(CDC* pDC, CPrintInfo*)
{
        // TODO: add code to print the controls
}

/////////////////////////////////////////////////////////////////////////////
// COentryView diagnostics

#ifdef _DEBUG
void COentryView::AssertValid() const
{
        CFormView::AssertValid();
}

void COentryView::Dump(CDumpContext& dc) const
{
        CFormView::Dump(dc);
}

COentryDoc* COentryView::GetDocument() // non-debug version is inline
{
        ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(COentryDoc)));
        return (COentryDoc*)m_pDocument;
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// COentryView message handlers

void COentryView::OnExitb()
{
        CString s;

   if (g_LQ) Qclose(&g_LQ,0);
```

```
    if (m_runflag) {
       m_runflag = 0;
       while( s != "Auto" ) { // Read the button until not runnning
          GetDlgItemText(IDC_AUTOB,s.GetBuffer(100),100);
       }
    }

    AfxGetMainWnd()->DestroyWindow();
}



//GetParentFrame()->SetMessageText("Unknown message");
long COentryView::OnReplyMsg(WPARAM  wParam, LPARAM  lParam){
    CString s,ss;

    switch(wParam) {
       case REPLY_SUBMODE:
          if (lParam)
             SetDlgItemText(IDC_RECIPT,(char *)lParam);
          else
             SetDlgItemText(IDC_RECIPT,"OnReplyMsg Unknown REPLY_SUBMODE Message"
          SetDlgItemInt(IDC_RECIPTS,++m_rec_rec);

       break;
       case NOMSGQ_SUBMODE:
          if (lParam)
             MessageBox("NOMSGQ_SUBMODE How did I get here?");

             /*

             SetDlgItemInt(IDC_MSGS,++m_mes_rec);

             if (g_delay) {
                SetTimer(FILL_DELAY_TIMER,g_delay,NULL);           //   g_direct_f
                SetTimer(FILL_ANAMATE_TIMER,100,NULL);
             } else
                g_direct_fill = 0;

             int instock = DbOrder(((pOFORM)lParam)->cust, ((pOFORM)lParam)->item

             if (instock == 1)        ss = "FILLED";
             else if (instock == 0)   ss = "OUT OF STOCK";
             else if (instock == -1)  ss = "WRONG CUSTOMER";
             else if (instock == -2)  ss = "WRONG ITEM";


             s.Format("Direct: %s: %s %s %d",LPCTSTR(ss),((pOFORM)lParam)->cust,(
             SetDlgItemText(IDC_FILLTXT,LPCTSTR(s));
             color_the_box( ((pOFORM)lParam)->color,0);
            */

       break;
       case DIRECT_FILL_TIME_OVER:
             GetParentFrame()->SetMessageText(lParam);


             // SetDlgItemInt(IDC_MSGS,--m_mes_sent);
```

```
      break;
      default:
         if (lParam)
             s.Format("ORM: %d %s",wParam,lParam);
         else
             s.Format("Unknown Message ORM");
         GetParentFrame()->SetMessageText(s);
   }
   return(0);
}



////////////////////////////////////////////////////////////////////////
////////////////////////   LISTENER THREAD  ////////////////////////////
////////////////////////////////////////////////////////////////////////

typedef struct tparam {  // Thread parameters
   HWND    hwnd;
   int     bnum; // smbuff num to send replys to.
   CString inst;
} TPARAM, *pTPARAM;

   TPARAM tp;  // <-- Global Data so it does not go away.



char g_thd_data[1000];        //////////////////////
MSGH g_thd_mh;                // Listener thread //
DWORD Listen(pTPARAM tp) {    //////////////////////
   int  max_dly,md,stat=0;
   CString s,n = tp->inst;


   if ( g_LQ = Qopen(n.GetBuffer(0),GET_MODE,0,  0,&stat,0,0)) {
      // Save the buffer I will listen to.
      tp->bnum = g_LQ->bufs_found[0]; // Replys will be sent to this buffer

      if (SHAREDATA(diag)) {
         s.Format("Thread Open(%s) bnum=%d",LPCTSTR(n),tp->bnum);
         ::PostMessage( tp->hwnd,MESSAGEREADY,0,(LPARAM) LPCTSTR(s) );
      }

/*                               // REPLY_SUBMODE (pass sub_mode as ORM_MES_REC
      while(QSUCCESS == Qget(g_LQ,&g_thd_mh,data,1000) ) {
         ::PostMessage(tp->hwnd,MESSAGEREADY, g_thd_mh.sub_mode ,(LPARAM) data);
//          ::PostMessage(tp->hwnd,MESSAGEREADY,ORM_MES_RECEIVED,(LPARAM) data);
*/


      while(QSUCCESS == QlistenBeforeReply(g_LQ,&g_thd_mh,g_thd_data,1000) ) {
         // Got a message could be Recipt or direct fill request
         if (g_thd_mh.sub_mode == REPLY_SUBMODE) {
            ::PostMessage(tp->hwnd,MESSAGEREADY, g_thd_mh.sub_mode ,(LPARAM)

         } else {

            g_direct_fill = 1;
```

```
          max_dly = 500; // 25 Seconds max   check 20 times a second
          while ((g_direct_fill || g_filling) && (--max_dly))  Sleep(50); //

          if (max_dly > 0)
             md = ACK_MODE;
          else {
             md = NACK_MODE;
             g_direct_fill = 0;
             MessageBox(NULL,"SER TIME OVER",NULL,NULL);
             s.Format("%s: SERVER did not reply to DIRECT FILL",n);
            ::PostMessage(tp->hwnd,MESSAGEREADY, DIRECT_FILL_TIME_OVER ,(LPARA

          }
        }
        QreplyAfterListen(g_LQ,md,0,g_thd_data,g_thd_mh.size,0);
      }.

   } else {
      s.Format("Cant Qopen(%s) ERROR=%d",LPCTSTR(n),stat);
      MessageBox(tp->hwnd,s,"Listener Thread",MB_ICONSTOP);
   }

   Qclose(&g_LQ,0);
   return(0);
}



void COentryView::CmdLine(int pass){
   CString parm,value,line,nline;
   int i,r,g,b;

   line = AfxGetApp()->m_lpCmdLine;

   while (2 <= (i = sscanf(LPCTSTR(line),"%s %s %[^@]",
      parm.GetBuffer(100),value.GetBuffer(100),nline.GetBuffer(100) ))) {
      parm.MakeUpper();

// oentry Name xxx Que ssss RGB r,g,b OrderMode m Fill x Poll x Order x
      if (pass == 1) {

         if (parm == "NAME")   {m_inst = LPCTSTR(value);}
         if (parm == "RGB")    {sscanf(LPCTSTR(value),"%d,%d,%d",&r,&g,&b);m_colo
         if (parm == "ORDERMODE") {sscanf(LPCTSTR(value),"%d",&m_OrderMode);m_Or
                               if (m_OrderMode < 0 || m_OrderMode > 3) m_OrderMod
         if (parm == "FILL")  {sscanf(LPCTSTR(value),"%d",&g_fill_delay);}
         if (parm == "POLL")  {sscanf(LPCTSTR(value),"%d",&g_poll_pps);
                               if((MIN_POLL <= g_poll_pps) && (g_poll_pps <= MAX_
                                  g_poll_delay  = 1000/g_poll_pps;}
         if (parm == "ORDER")  {sscanf(LPCTSTR(value),"%d",&g_place_tpm);
                               if((MIN_AUTO <= g_place_tpm) && (g_place_tpm <= MA
                                  g_place_delay = 60000/g_place_tpm;}
         if (parm == "ORACLE") { strcpy(g_oracle_con_str ,LPCTSTR(value)); }

      } else {
         if (parm == "QUE" || parm == "SERVER") {
            CComboBox * CB = (CComboBox *) this->GetDlgItem(IDC_QUE);
            CB->SelectString(-1,value);
            OnSelchangeQue();
         }
      }

      line = LPCTSTR(nline);
      nline = "";
   }
}


void COentryView::LoadQList(int mode)
{
   // mode=0 : List Non-Ques
   // mode=1 : List Ques
   int i;

   CComboBox * CB = (CComboBox *) this->GetDlgItem(IDC_QUE);
   CB->ResetContent();

// List all logical queues
// APPS: [physical],logical1,logical2,[physical],logical,
   if (sm_base = AttachSharedMemory()){
      lpRT  rt = RTROOT;
      while(rt = NextRT(rt)) {
         char *e,*s = RT_APPS(rt);     // Starts after the first letter
         while (s = strchr(s,',')) {  // Ends at next ","
            if (e = strchr(++s,',')) {
               *e = 0;
               if ((!strchr(s,'['))  && *s) {
                  if ((mode && strchr(s,'Q')) || (!mode && !strchr(s,'Q')))
                     CB->AddString(s); // lb->InsertString(-1,s)
               }
               *e = ',';
            }
         }
      }

      // Now add any extra entries from the local node that were not in the RT
  // lpSMBUF  BUF;
      for (i = 0; i < SHAREDATA(nsbuf) ; i++) {
         if (!strcmp((SMBUFADDR(i))->name,"empty")) continue;
         if (!strcmp((SMBUFADDR(i))->name,"QNETD")) continue;
         if (CB_ERR != CB->FindStringExact(-1,(SMBUFADDR(i))->name)) continue; /
         if ((mode && strchr((SMBUFADDR(i))->name,'Q')) || (!mode && !strchr((SM
            CB->AddString((SMBUFADDR(i))->name); // lb->InsertString(-1,s)
      }

   } else {
      MessageBox("QNETD not running. Please restart.");
   }
   CB->AddString("");

   if ( CB_ERR == (i = CB->FindStringExact(-1,m_que)))
      CB->SetCurSel(0);
   else
      CB->SetCurSel(i);

   OnSelchangeQue();

}
```

```
void COentryView::OnInitialUpdate()
{
        CString s;
        CFormView::OnInitialUpdate();

   // Set frame size = Form size
   GetParentFrame()->RecalcLayout();
   ResizeParentToFit(FALSE);
   ResizeParentToFit(TRUE);

   m_OrderMode = PLACE_ORDER;

   LoadQList(1);

   CmdLine(1);

   // Set max size on data fields
   (  (CEdit *) this->GetDlgItem(IDC_CUST)  )->LimitText(9);
   (  (CEdit *) this->GetDlgItem(IDC_ITEM)  )->LimitText(9);
   (  (CEdit *) this->GetDlgItem(IDC_QTY)   )->LimitText(9);

   // Generate unique instance name on this node
   if (m_inst == "") // May be set by command line "name"

      m_inst.Format("OE%d",SHAREDATA(ap.app_num)++);

   OnAnyUserAction();

   SetTimer(SET_1ST_TITLE_TIMER,100,NULL);  // Calls OnAnyUserAction(); to set t


   // Start the backgound thread to Qget() messages
   DWORD id;
   tp.hwnd = m_hWnd;
   tp.inst = m_inst;
   CreateThread(NULL,0,(LPTHREAD_START_ROUTINE) Listen, (LPVOID) &tp,0,&id);


   CmdLine(2);

   ( (CButton *) this->GetDlgItem(IDC_PLACER) )->SetCheck(0);
   ( (CButton *) this->GetDlgItem(IDC_PLACENOQR) )->SetCheck(0);
   ( (CButton *) this->GetDlgItem(IDC_FILLR) )->SetCheck(0);
   ( (CButton *) this->GetDlgItem(IDC_FILLDB) )->SetCheck(0);

   if (m_OrderMode == PLACE_ORDER)
      ( (CButton *) this->GetDlgItem(IDC_PLACER) )->SetCheck(1);
   if (m_OrderMode == PLACE_ORDER_NOQ)
      ( (CButton *) this->GetDlgItem(IDC_PLACENOQR) )->SetCheck(1);
   if (m_OrderMode == FILL_ORDER)
      ( (CButton *) this->GetDlgItem(IDC_FILLR) )->SetCheck(1);
   if (m_OrderMode == FILL_FROM_ORACLE)
      ( (CButton *) this->GetDlgItem(IDC_FILLDB) )->SetCheck(1);

#ifdef ORACLE
      ((CButton *) this->GetDlgItem(IDC_FILLDB))->EnableWindow(TRUE);
```

```
#else
      ((CButton *) this->GetDlgItem(IDC_FILLDB))->EnableWindow(FALSE);
#endif


   PlaceOrFillMode(m_OrderMode);


   // Fonts
   LOGFONT lf;
   memset(&lf,0,sizeof(LOGFONT));

#ifdef BIGFONT
   lf.lfHeight = 18;
#else
   lf.lfHeight = 13;
#endif
   g_text_font.CreateFontIndirect(&lf);

        strcpy(lf.lfFaceName,"Monotype Corsiva");
   lf.lfHeight = 32;
   g_title_font.CreateFontIndirect(&lf);


   GetDlgItem(IDC_BIG_TITLE)->SetFont(&g_title_font);

   int i = 0;
   while (ALL_TEXT[i])
      GetDlgItem(ALL_TEXT[i++])->SetFont(&g_text_font);
}



//void COentryView::OnEditchangeQue()
void COentryView::OnEditupdateQue()
{
        OnSelchangeQue();
}

void COentryView::OnSelchangeQue()
{
   CString que,s;
   int stat=0;

   // Get the que name
   GetDlgItemText(IDC_QUE,que.GetBuffer(100),100);
   que.ReleaseBuffer();


   if (que != m_que) { // New value
      if (Q) Qclose(&Q,0);

      if (que == "") { // No que
         m_que = "";
         s = "No Que will be used";
         GetDlgItem(IDC_AUTOB)->EnableWindow(FALSE);
         GetDlgItem(IDC_ORDERB)->EnableWindow(FALSE);

      } else {          // Set value
         s.Format("Opening que %s",que);
         GetParentFrame()->SetMessageText(s);

         if (Q = Qopen(que.GetBuffer(0),PUT_MODE,0,0,&stat,0,0) ) {
            if (m_sendreply)
               Q->msgh.reply_smbuf = tp.bnum;
            m_que = que;
            s.Format("Qopen(%s)",que);

            GetDlgItem(IDC_AUTOB)->EnableWindow(TRUE);
            GetDlgItem(IDC_ORDERB)->EnableWindow(TRUE);

            if (m_OrderMode == PLACE_ORDER_NOQ)
               Q->msgh.sub_mode = NOMSGQ_SUBMODE;


         } else {
            s.Format("Qopen(%s) Error %d",que,stat);
            GetDlgItem(IDC_AUTOB)->EnableWindow(FALSE);
            GetDlgItem(IDC_ORDERB)->EnableWindow(FALSE);

         }
      }
      GetParentFrame()->SetMessageText(s);

   }
   OnAnyUserAction();
}

OFORM g_buf ;
DWORD DirectPlace() {
   int flags = 0;
   QsendAndReceive(Q,0,SUB_MODE_OK, flags,sizeof(g_buf),(char *) &g_buf,  0,0,0,
   g_directplace = 0;
   return(0);
}

//ORDERORDERORDERORDERORDERORDERORDERORDERORDERORDERORDERORDERORDERORDER
//ORDERORDERORDERORDERORDERORDERORDERORDERORDERORDERORDERORDERORDERORDER
//ORDERORDERORDERORDERORDERORDERORDERORDERORDERORDERORDERORDERORDERORDER
//ORDERORDERORDERORDERORDERORDERORDERORDERORDERORDERORDERORDERORDERORDER
//ORDERORDERORDERORDERORDERORDERORDERORDERORDERORDERORDERORDERORDERORDER

void COentryView::OnOrderb()
{  // Check m_ordermode = 0 to place order or = 1 to fill the order
        CString que,str,s = "";
   int   gsize,stat=0,flags,instock,filled=0;
//    OFORM buf ;
   char  line[sizeof(OFORM)+10];
   MSGH  mh;  // To get detailed put status
   DWORD id;

   if (g_directplace) return; // direct place thread is already working
   if (g_filling) return; // Wait for fill done is already working
   if (Q) {
      if (!m_runflag) OnAnyUserAction();

      if (m_auto_tran) {
         if(m_auto_rand < 16) m_auto_rand = rand();

         // Begin a tran?
   if ((m_auto_tran == 1) && NEXTBIT(m_auto_rand)) {
           if (!m_tran_state) m_tran_state= 1;  // Done by OnTran (causes
      m_auto_tran = 2;
      GetDlgItem(IDC_TRANB)->ShowWindow(SW_HIDE);

      if (m_auto_commit = NEXTBIT(m_auto_rand))
         GetDlgItem(IDC_COMMITB)->ShowWindow(SW_SHOW);
      else
         GetDlgItem(IDC_ABORTB)->ShowWindow(SW_SHOW);

   }
}


// Set the flags for get or put
if (m_tran_state)
   if (m_tran_state++ == 1)
      flags = Q_TRAN_BEGIN;
   else
      flags = Q_TRAN;
else
   flags = 0;


if (m_OrderMode <= PLACE_ORDER) { // Place the order

   // Read the data to be sent
   GetDlgItemText(IDC_CUST,g_buf.cust,10);
   GetDlgItemText(IDC_ITEM,g_buf.item,10);
   g_buf.qty = GetDlgItemInt(IDC_QTY,NULL,TRUE);
   g_buf.color = m_color;
   g_buf.reply_to = SHAREDATA(hostip);

   SetDlgItemInt(IDC_MSGS,++m_mes_sent);
   if (m_OrderMode == PLACE_ORDER_NOQ) {
      g_directplace = 1;
      GetDlgItem(IDC_ORDERB)->EnableWindow(FALSE);

      SetTimer(WAIT_ANAMATE_TIMER,400,NULL);
      CreateThread(NULL,0,(LPTHREAD_START_ROUTINE) DirectPlace, (LPVOID) 0

   } else if (QSUCCESS == (stat = QsendAndReceive(Q,0,SUB_MODE_OK, flags,s

      if (mh.mode == ACK_MODE && mh.sub_mode == SUB_MODE_OK){
         m_tran_sent++;
         s.Format("%s: %8s %8s %d",m_que.GetBuffer(0),g_buf.cust,g_buf.ite
      } else {
         if (mh.sub_mode == SUB_MODE_FULL)
            if (m_runflag)
               s.Format("que is FULL");
            else
               MessageBox("que is FULL");
         else
            s.Format("Qput() error %d",mh.sub_mode);
      }
   } else {  // not QSUCCESS (communications error)
      SetDlgItemInt(IDC_MSGS,--m_mes_sent);
      s.Format("Communication error code %d",stat);
   }
```

```
      GetParentFrame()->SetMessageText(s);

   } else { // FILL_ORDER Fill the order
      *g_buf.cust = *g_buf.item = 0;
      g_buf.qty = 0;

      if (g_direct_fill) {  // There was a direct placed order to fill
         pOFORM po = (pOFORM) &g_thd_data;
         CString ss;

         SetDlgItemInt(IDC_MSGS,++m_mes_rec);

         if (g_fill_delay) {
            g_filling = 1; // Do not fill another until the timer expires
            SetTimer(DIRECT_FILL_TIMER,g_fill_delay,NULL);          //   g_di
            SetTimer(FILL_ANAMATE_TIMER,100,NULL);
         }

         color_the_box( po->color,0);
         int instock = DbOrder(po->cust, po->item, po->qty);

         if (instock == 1)       ss = "FILLED";
         else if (instock == 0)  ss = "OUT OF STOCK";
         else if (instock == -1) ss = "WRONG CUSTOMER";
         else if (instock == -2) ss = "WRONG ITEM";
         filled++;

         s.Format("Direct: %s: %s %s %d",LPCTSTR(ss), po->cust, po->item, po-
         SetDlgItemText(IDC_FILLTXT,LPCTSTR(s));


         g_direct_fill = 0;
         if (g_filling) return;
      }



      if (QSUCCESS == (stat = QsendAndReceive(Q,REQUEST_MODE,SUB_MODE_OK,flag
         sizeof(g_buf),(char *) &g_buf,&gsize,&mh))) {

         if (mh.mode == ACK_MODE && mh.sub_mode == SUB_MODE_OK) {
            SetDlgItemInt(IDC_MSGS,++m_mes_rec);
            if (gsize < sizeof(OFORM))
               s.Format("Bad reply length");
            else { // you got valid message

               m_tran_rec++;
               color_the_box(g_buf.color,0); // But dont force it

               if (m_OrderMode == FILL_FROM_ORACLE)
                  instock = OraOrder(g_buf.cust,g_buf.item,g_buf.qty); // Ora
               else
                  instock = DbOrder(g_buf.cust,g_buf.item,g_buf.qty);
               if (g_fill_delay) {
                  g_filling = 1; // Do not fill another until the timer expir
                  SetTimer(FILL_DELAY_TIMER,g_fill_delay,NULL);         /
                  SetTimer(FILL_ANAMATE_TIMER,100,NULL);
               }
            }
         } else if (mh.sub_mode == SUB_MODE_EMPTY) {
            if (m_runflag || filled)
               s.Format("Que Empty");
            else
               MessageBox("Que Empty");
         } else
            s.Format("Bad reply mode %d:%d",mh.mode,mh.sub_mode);

      } else { // not QSUCCESS (communications error)
         mh.mode = 0;// Not ACK_MODE
         s.Format("Communication Error Code %d",stat);
      }


      if (gsize == sizeof(OFORM)) {
         if (instock == 1 )
            sprintf(line,"FILLED:%9s %9s %d",g_buf.cust,g_buf.item,g_buf.qty)
         else if ( instock == 0)
            sprintf(line,"OUT OF STOCK:%9s %9s %d",g_buf.cust,g_buf.item,g_bu
         else if ( instock == -1)
            sprintf(line,"NO SUCH ITEM:%9s %9s %d",g_buf.cust,g_buf.item,g_bu
         else // instock == -2
            sprintf(line,"NO SUCH CUSTOMER:%9s %9s %d",g_buf.cust,g_buf.item,
         SetDlgItemText(IDC_FILLTXT,line);
      }
        // else
        // strcpy(line,"");


      // Send Reply if requested
      if ((mh.reply_smbuf >= 0 ) && (mh.reply_smbuf < SHAREDATA(nsbuf)) && (s
         mh.mid.host = g_buf.reply_to; // HACK! Makes sure sending host ge
         Qrep = QopenReply(Qrep,&mh,0,0,0);


         if (QSUCCESS != (stat = Qput(Qrep,0,REPLY_SUBMODE,0,sizeof(line),lin
            s.Format("Qput(%s) Error %d",Qrep->msgh.to_server,stat);
         else
            s.Format("Replying Qput(%s) to smbuf %d",Qrep->msgh.to_server,Qre
         SetDlgItemInt(IDC_RECIPTS,++m_rec_sent);

      }

      GetParentFrame()->SetMessageText(s);
    /*
      if (m_runflag && mh.mode == ACK_MODE) // We got something
         OnOrderb(); // Check again for next message, no need to wait.
      else // We didnt get anything
         color_the_box( m_color,0); // restore old color
      */
      if (!m_runflag && !g_fill_delay) color_the_box( m_color,0);


   } // place or fill


   if ((m_OrderMode <= PLACE_ORDER) || mh.mode == ACK_MODE) // Placeing or
   if (m_auto_tran > 1) {
      if ( ( NEXTBIT(m_auto_rand) && NEXTBIT(m_auto_rand) )
         || (m_auto_tran++ > 8) ) {
            // Commit or Abort tran?
            if (m_auto_commit)
               OnCommitb();
            else
               OnAbortb();
            m_auto_tran = 1;
         }
      }


      if ((m_OrderMode >= FILL_ORDER) && m_runflag &&
         mh.mode == ACK_MODE && mh.sub_mode == SUB_MODE_OK)
         if (g_max_orders++ < 200)
                        OnOrderb(); // Try to get one more item.
               else
                         g_max_orders = 0;
      } else
                g_max_orders = 0;

   // The order or fill button will be active until we are done.
   // CButton * CB = (CButton *) this->GetDlgItem(IDC_ORDERB);
   // CB->SetState(TRUE);

}

//
// m_auto_tran:
// 0 No auto trans
// 1 No tran yet
// 2 After OnAutob()
//
int g_poll_delay_old;
int g_place_delay_old;
void COentryView::OnTimer(UINT nIDEvent)
{


        if (nIDEvent == PLACE_TIMER ) {

      if (g_directplace) return; // direct place thread is already working
      if (g_filling) return; // Wait for fill done is already working
      if (g_place_delay_old != g_place_delay) {
         KillTimer(PLACE_TIMER);
         SetTimer(PLACE_TIMER,g_place_delay_old = g_place_delay,NULL);
      }

      SetDlgItemText(IDC_CUST,g_cust[m_order_num % 6]);
      SetDlgItemText(IDC_ITEM,g_item[m_order_num % 5]);
      SetDlgItemInt(IDC_QTY,1 + (m_order_num++ % 9),FALSE);

      OnOrderb();
      //CButton * CB = (CButton *) this->GetDlgItem(IDC_ORDERB);
      //CB->SetState(TRUE);


        } else if (nIDEvent == POLL_FILL_TIMER) {

      if (!g_filling) {
         if (g_poll_delay != g_poll_delay_old) {
            KillTimer(POLL_FILL_TIMER);
         SetTimer(POLL_FILL_TIMER,g_poll_delay_old = g_poll_delay,NULL);
      }

      OnOrderb();
   }

     } else if (nIDEvent == WAIT_ANAMATE_TIMER) {

   if (g_directplace) {

      // Anomate the wait  0,1 0,1
      GetDlgItem(WAITS[g_wait_anamate++])->ShowWindow(SW_HIDE);
      if (g_wait_anamate > 1) g_wait_anamate = 0;
      GetDlgItem(WAITS[g_wait_anamate])->ShowWindow(SW_SHOW);
   } else {
      KillTimer(WAIT_ANAMATE_TIMER);
      GetDlgItem(IDC_ORDERB)->EnableWindow(TRUE);
      GetDlgItem(WAITS[g_wait_anamate])->ShowWindow(SW_HIDE);
   }

     } else if (nIDEvent == FILL_ANAMATE_TIMER) {

   if (g_filling == 0) {    // End Anamation
      KillTimer(FILL_ANAMATE_TIMER);
      GetDlgItem(FILLS[g_fill_anamate])->ShowWindow(SW_HIDE);

   } else { // Anamate  0,1,2

      GetDlgItem(FILLS[g_fill_anamate++])->ShowWindow(SW_HIDE);
      if (g_fill_anamate > 2) g_fill_anamate = 0;
      GetDlgItem(FILLS[g_fill_anamate])->ShowWindow(SW_SHOW);

      if (1) {   // g_progress
           // ( (CButton *) this->GetDlgItem(IDC_PLACER) )->SetCheck(0);
//         GetDlgItem(IDC_PROG))->SetRange(0,100);
 //        ((CProgressCtrl *) this->GetDlgItem(IDC_PROG))->SetRange(0,100);
 //        ((CProgressCtrl *) this->GetDlgItem(IDC_PROG))->SetPos(50);
 //        ((CProgressCtrl *) this->GetDlgItem(IDC_PROG))->ShowWindow(SW_SHOW
      }


   }

     } else if (nIDEvent == DIRECT_FILL_TIMER) {

   g_direct_fill = 0; // let Qsar() return.
   g_filling = 0;      // End Anamation, let next order in
   KillTimer(DIRECT_FILL_TIMER);
  color_the_box( m_color,0);   // Restore the old color

  } else if (nIDEvent == FILL_DELAY_TIMER) {

   g_filling = 0;      // End Anamation, let next order in
   KillTimer(FILL_DELAY_TIMER);
  color_the_box( m_color,0);   // Restore the old color

     } else if (nIDEvent == SET_1ST_TITLE_TIMER) {  // Only done once

      KillTimer(SET_1ST_TITLE_TIMER);
```

```
            OnAnyUserAction();

      } else if (nIDEvent == DB_DONE_TIMER) {
         if (g_db_run == 40) {
            g_db_run = 0;
            GetDlgItem(IDC_SHOWDB)->EnableWindow(TRUE);
            KillTimer(DB_DONE_TIMER);
         }
           } else if (nIDEvent == OPTIONS_DONE_TIMER) {

         if (g_options_run == 40) { // End
            g_options_run = 0;
            GetDlgItem(IDC_OPTIONSB)->EnableWindow(TRUE);
            KillTimer(OPTIONS_DONE_TIMER);
         }

         if (m_color != g_new_color) {
            color_the_box(m_color = g_new_color,1);
         }

         if (g_clear_stats) {
            g_clear_stats = 0;
            m_mes_sent = 0;
            m_mes_rec = 0;
            m_rec_sent = 0;
            m_rec_rec = 0;
            SetDlgItemInt(IDC_MSGS,     0);
            SetDlgItemInt(IDC_RECIPTS, 0);
         }
      } else

           CFormView::OnTimer(nIDEvent);
}


void COentryView::OnAutob()
{
      if (m_runflag) {
         KillTimer(PLACE_TIMER);
         KillTimer(POLL_FILL_TIMER);
         m_runflag = 0; // You are no longer running
         if (m_auto_tran) OnAbortb();
         m_auto_tran = 0;
         SetDlgItemText(IDC_AUTOB,"Auto");


         GetDlgItem(IDC_EXITB)->EnableWindow(TRUE);
         ((CButton *) this->GetDlgItem(IDC_PLACER))->EnableWindow(TRUE);
         ((CButton *) this->GetDlgItem(IDC_PLACENOQR))->EnableWindow(TRUE);
         ((CButton *) this->GetDlgItem(IDC_FILLR))->EnableWindow(TRUE);
#ifdef ORACLE
         ((CButton *) this->GetDlgItem(IDC_FILLDB))->EnableWindow(TRUE);
#endif
         color_the_box( m_color,1);  // Restore default color

      } else {
         m_runflag = 1;   // You will be running
         m_order_num = 0; // 1st order number Controls order selections
         m_auto_tran = m_tran_state; // make random transactions too?.
      if ( m_OrderMode > PLACE_ORDER ) // FILL
         SetTimer(POLL_FILL_TIMER,g_poll_delay_old = g_poll_delay,NULL);
      else
         SetTimer(PLACE_TIMER,g_place_delay_old = g_place_delay,NULL);


      SetDlgItemText(IDC_AUTOB,"    Stop    ");

      GetDlgItem(IDC_EXITB)->EnableWindow(FALSE);
      ( (CButton *) this->GetDlgItem(IDC_PLACER) )->EnableWindow(FALSE);
      ( (CButton *) this->GetDlgItem(IDC_PLACENOQR) )->EnableWindow(FALSE);
      ( (CButton *) this->GetDlgItem(IDC_FILLR) )->EnableWindow(FALSE);
      ( (CButton *) this->GetDlgItem(IDC_FILLDB) )->EnableWindow(FALSE);
  }
  OnAnyUserAction();
}


// Color Color Color Color Color Color Color Color Color Color
// Color Color Color Color Color Color Color Color Color Color
// Color Color Color Color Color Color Color Color Color Color


 void COentryView::OnColor()
{
   CHOOSECOLOR cc;      // common dialog box structure
   COLORREF acrCustClr[16];

   // Setup the custom colors as a grey scale
   for (int v=0,i=0; i < 16; v=17 * i++)
      acrCustClr[i] = RGB(v,v,v);

   // Initialize the necessary members.
   cc.lStructSize = sizeof(CHOOSECOLOR);
   cc.hwndOwner = NULL; //  = hwnd;
   cc.lpCustColors = (LPDWORD) acrCustClr;
   cc.Flags = CC_FULLOPEN;  //  CC_PREVENTFULLOPEN


   if (ChooseColor(&cc)){
      CString s;

      m_box_color = m_color = cc.rgbResult; // lpCustColors
      Invalidate(); // Display the new color
   } else {
      GetParentFrame()->SetMessageText("Color was not changed");
   }
   OnAnyUserAction();
}


//Hinit = 1;
HBRUSH COentryView::OnCtlColor(CDC* pDC, CWnd* pWnd, UINT nCtlColor)
{

   if (nCtlColor == CTLCOLOR_EDIT) {
      if (pWnd->GetDlgCtrlID() == IDC_COLORBOX) {

         pDC->SetBkColor(m_color);
         m_brush = CreateSolidBrush(m_color);
         return(m_brush);

      }
   } else if (nCtlColor == CTLCOLOR_BTN ) {  // CTLCOLOR_EDIT
      if (pWnd->GetDlgCtrlID() == IDC_AUTOB)
      if (m_runflag) {
         pDC->SetBkColor(RGB(255,0,0));          //  SetBkColor SetTextColor
         pDC->SetTextColor(RGB(255,255,255));    //  SetBkColor SetTextColor
         m_brush = CreateSolidBrush(m_color);
         return(m_brush);
      }
//    } else if (nCtlColor == CTLCOLOR_STATIC ) {  //  CTLCOLOR_EDIT
//       if (pWnd->GetDlgCtrlID() == IDC_INST){
//          // pDC->SetBkColor(RGB(255,0,0));          //  SetBkColor SetTextColor
//          // pDC->SetTextColor(RGB(255,255,255));    //  SetBkColor SetTextColor
//          // m_brush = CreateSolidBrush(m_color);
//          m_brush = CreateSolidBrush(m_box_color); // empty non-text background
//          pDC->SetBkColor(m_box_color);            // behind the letters
//          return(m_brush);
//       }
   }

   HBRUSH hbr = CFormView::OnCtlColor(pDC, pWnd, nCtlColor);

        return hbr;
}


int COentryView::DbOrder(char *cust, char *item, int qty) {
   int custn=0,itemn=0;
   while(strcmp(item,g_item[itemn])) if (++itemn == 6) return(-1);
   while(strcmp(cust,g_cust[custn])) if (++custn == 6) return(-2);

   if (g_qty[itemn] >= qty) {
      g_qty[itemn] -= qty;
      g_purchases[custn] += qty * g_price[itemn];
      g_num_purchases[custn]++;
      g_total_sales += qty * g_price[itemn];

   } else
      return(0);
   return(1);
}



int COentryView::OraOrder(char *cust, char *item, int qty) {
   int custn=0,itemn=0, price,stock,rc,cust_orders,cust_sales;
   CString s;
   while(strcmp(item,g_item[itemn])) if (++itemn == 6) return(-1);
   while(strcmp(cust,g_cust[custn])) if (++custn == 6) return(-2);


   if (g_ora_state == 0) {
      GetParentFrame()->SetMessageText("Cant Order from oracle: not connected");
      return(0);
   }
   // Read Oracle
   if (  (rc = oraread(item, &price, &stock))  ) {
      s.Format("OraRead Error %d",rc);
      GetParentFrame()->SetMessageText(s);
      return(0);
   }
   if (  (rc = oracustr(cust, &cust_orders, &cust_sales))  ) {
      s.Format("OraCustRead Error %d",rc);
      GetParentFrame()->SetMessageText(s);
   }


   if (qty > stock) {
      return(0);  // MessageBox("OutOfStock");
   }  else {

      stock -= qty;

      g_total_sales += qty * price;
      cust_sales += qty * price;
      cust_orders++;


      // Write Oracle
      if ( (rc = orawrite(item, stock))  ) {
         s.Format("OraWrite Error %d",rc);
         GetParentFrame()->SetMessageText(s);
         return(0);
      }
      if (  (rc = oracustw(cust, cust_orders, cust_sales))  ) {
         s.Format("OraCustWrite Error %d",rc);
         GetParentFrame()->SetMessageText(s);
      }

   }

   return(1);
}


void COentryView::PlaceOrFillMode(int mode)
{
   m_OrderMode = mode;
   int SHOW, HIDE, IsPlaceMode;

   if (mode <= PLACE_ORDER) {
      HIDE = SW_HIDE;
      SHOW = SW_SHOW;
      IsPlaceMode = TRUE;
   } else {
      HIDE = SW_SHOW;
      SHOW = SW_HIDE;
      IsPlaceMode = FALSE;
   }

   GetDlgItem(IDC_SENDREPC)->ShowWindow(SHOW); // Show sendreply checkbox

   if (m_OrderMode <= PLACE_ORDER){
      SetDlgItemText(IDC_ORDERB,"Order");              // Order = Order
      SetDlgItemText(IDC_ORDERBOX,"Place Orders");            // Order = Order
      SetDlgItemText(IDC_BIG_TITLE,"OpenMQ  Place Orders");          // Order =
   } else {
      SetDlgItemText(IDC_ORDERB,"Fill");
      SetDlgItemText(IDC_ORDERBOX,"Fill Orders");
      SetDlgItemText(IDC_BIG_TITLE,"OpenMQ  Fill Orders");
   }

   GetDlgItem(IDC_SHOWDB)->ShowWindow(HIDE);

   GetDlgItem(IDC_CUST)->ShowWindow(SHOW);
   GetDlgItem(IDC_ITEM)->ShowWindow(SHOW);
   GetDlgItem(IDC_QTY)->ShowWindow(SHOW);
   GetDlgItem(IDC_CUSTLAB)->ShowWindow(SHOW);
   GetDlgItem(IDC_ITEMLAB)->ShowWindow(SHOW);
   GetDlgItem(IDC_QTYLAB)->ShowWindow(SHOW);

   GetDlgItem(IDC_FILLTXT)->ShowWindow(HIDE);

//    (  (CEdit *) this->GetDlgItem(IDC_CUST)   )->SetReadOnly(FALSE);
//    (  (CEdit *) this->GetDlgItem(IDC_ITEM)   )->SetReadOnly(FALSE);
//    (  (CEdit *) this->GetDlgItem(IDC_QTY)    )->SetReadOnly(FALSE);

   // Stats
   if (mode <= PLACE_ORDER) {

      SetDlgItemText(IDC_MSGS_LAB,"Sent:");
      SetDlgItemText(IDC_RECIPTS_LAB,"Received:");
      SetDlgItemInt(IDC_MSGS,    m_mes_sent);
      SetDlgItemInt(IDC_RECIPTS, m_rec_rec);
   } else {
      SetDlgItemText(IDC_MSGS_LAB,"Received:");
      SetDlgItemText(IDC_RECIPTS_LAB,"Sent:");
      SetDlgItemInt(IDC_MSGS,    m_mes_rec);
      SetDlgItemInt(IDC_RECIPTS, m_rec_sent);
   }

   if (mode != PLACE_ORDER_NOQ) {
      GetDlgItem(IDC_TRANBOX)->ShowWindow(SW_SHOW);
      if (m_tran_state) {
         GetDlgItem(IDC_COMMITB)->ShowWindow(SW_SHOW);
         GetDlgItem(IDC_ABORTB)->ShowWindow(SW_SHOW);
      } else
         GetDlgItem(IDC_TRANB)->ShowWindow(SW_SHOW);
   } else {
      GetDlgItem(IDC_TRANBOX)->ShowWindow(SW_HIDE);
      GetDlgItem(IDC_TRANB)->ShowWindow(SW_HIDE);
      GetDlgItem(IDC_COMMITB)->ShowWindow(SW_HIDE);
      GetDlgItem(IDC_ABORTB)->ShowWindow(SW_HIDE);
      GetDlgItem(IDC_SENDREPC)->ShowWindow(SW_HIDE);
   }


   if (mode == PLACE_ORDER_NOQ) {
      LoadQList(0); // Load non-Que
   } else {
      LoadQList(1); // Load Que names
   }
   // DB
   if (mode <= PLACE_ORDER)
      if (g_db_run) g_db_run = 40; // Turn off the DB display if it was left on.


   if (mode == FILL_FROM_ORACLE) {
      if (g_ora_state == 0)
         if(oraconn(g_oracle_con_str))
            MessageBox("Oracle Connect Failed");
         else
            g_ora_state = 1;
   } else {
      if (g_ora_state == 1)
         if(oradisc())
            MessageBox("Oracle DisConnect Failed");
         else
            g_ora_state = 0;
   }


   OnAnyUserAction();
}

void COentryView::OnPlacenoqr()
{
   if (!((CButton *)GetDlgItem(IDC_PLACENOQR))->GetCheck())
      return;
   PlaceOrFillMode(PLACE_ORDER_NOQ);
   OnAnyUserAction();
}
void COentryView::OnPlacer()
{
   if (!((CButton *)GetDlgItem(IDC_PLACER))->GetCheck())
      return;
   PlaceOrFillMode(PLACE_ORDER);
   OnAnyUserAction();
}

void COentryView::OnFillr()
{
   if (!((CButton *)GetDlgItem(IDC_FILLR))->GetCheck())
      return;

   PlaceOrFillMode(FILL_ORDER);
   OnAnyUserAction();
}

void COentryView::OnFilldb()
{
   if (!((CButton *)GetDlgItem(IDC_FILLDB))->GetCheck())
      return;

   PlaceOrFillMode(FILL_FROM_ORACLE);
   OnAnyUserAction();
}



void COentryView::OnAnyUserAction()
{
   SetDlgItemText(IDC_RECIPT,"");
   SetDlgItemText(IDC_FILLTXT,"");

   if (m_que == "")
      GetParentFrame()->SetWindowText(m_inst + ": No server is selected" );
   else if (m_OrderMode == PLACE_ORDER)
      GetParentFrame()->SetWindowText(m_inst + ": Place orders into queue " + m_
   else if (m_OrderMode == PLACE_ORDER_NOQ)
      GetParentFrame()->SetWindowText(m_inst + ": Place orders directly to " + m
   else // FILL
      GetParentFrame()->SetWindowText(m_inst + ": Fill orders from queue " + m_q

   m_box_color = m_color;
}

void COentryView::OnSendrepc()
{
   m_sendreply = !m_sendreply; // How can I get this info?

   if ( Q )
      if ( m_sendreply )
         Q->msgh.reply_smbuf = tp.bnum;
      else
         Q->msgh.reply_smbuf = -1;
}

void COentryView::color_the_box(COLORREF color, BOOL forceit){
   if (( m_box_color != color ) || forceit) {
      m_box_color = color;

        // The DC for the color box
      CDC* pCOLORDC = GetDlgItem(IDC_COLORBOX)->GetDC();
      // pCOLORDC->GetRect();

     // Create a PEN
     // CPen *pQPen = new CPen(PS_SOLID, 3,RGB(0,0,255));
     // CPen *pOldPen = pXDC->SelectObject(pQPen);

      // Create the Brush
      CBrush *pQBrush = new CBrush(color);

      // Select this brush, save the old
      CBrush *pOldBrush = pCOLORDC->SelectObject(pQBrush);

      // Draw the box

    //  pCOLORDC->Rectangle(CRect(1,1,65,65));

      pCOLORDC->Rectangle(m_color_box_rec);

      pCOLORDC->SelectObject(&pOldBrush);  // Reset the brush

      delete(pQBrush);
      ReleaseDC(pCOLORDC);

 //     GetDlgItem(IDC_LOGO_Q)->Invalidate();
//      GetDlgItem(IDC_INST)->Invalidate();
   }
}
```

```
void COentryView::OnDraw(CDC* pDC)
{

/*
   pControlDC->SelectStockObject(WHITE_BRUSH);
*/

   // The DC for the Instance name
 /*
   CDC* pINSTDC = GetDlgItem(IDC_INST)->GetDC();

   pINSTDC->SetBkColor(m_color);
   pINSTDC->SetTextColor(m_color);
   pINSTDC->SetBkMode(TRANSPARENT);
   pINSTDC->SelectStockObject(HOLLOW_BRUSH);
   */
        GetDlgItem(IDC_COLORBOX)->GetWindowRect(&m_color_box_rec);
      m_color_box_rec.right  -= m_color_box_rec.left;  //ScreenToClient(&rec);
      m_color_box_rec.bottom -= m_color_box_rec.top;
      m_color_box_rec.left    = m_color_box_rec.top = 0;

   color_the_box(m_box_color,1);

        CFormView::OnDraw(pDC);
}

/*void COentryView::OnResetb()
{
   m_mes_sent = 0;
   m_mes_rec = 0;
   m_rec_sent = 0;
   m_rec_rec = 0;
   SetDlgItemInt(IDC_MES_REC,0);
   SetDlgItemInt(IDC_REC_REC,0);
   SetDlgItemInt(IDC_MES_SENT,0);
   SetDlgItemInt(IDC_REC_SENT,0);

   OnAnyUserAction();
}
*/


// TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT
// TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT
// TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT

void COentryView::OnTranb()
{
        m_tran_state = 1;
   GetDlgItem(IDC_TRANB)->ShowWindow(SW_HIDE);
   GetDlgItem(IDC_ABORTB)->ShowWindow(SW_SHOW);
   GetDlgItem(IDC_COMMITB)->ShowWindow(SW_SHOW);
   OnAnyUserAction();
}

void COentryView::OnCommit(int action)
{

   // Anyway get rid of the commit/abort buttons
   GetDlgItem(IDC_TRANB)->ShowWindow(SW_SHOW);
   GetDlgItem(IDC_ABORTB)->ShowWindow(SW_HIDE);
   GetDlgItem(IDC_COMMITB)->ShowWindow(SW_HIDE);

   // Do the commit
   if ((Q) && (m_tran_state > 1))
      Qcommit(action);

   m_tran_rec = 0;
   m_tran_sent = 0;
        m_tran_state = 0;
}

void COentryView::OnCommitb()
{
        OnCommit(Q_COMMIT);
}

void COentryView::OnAbortb()
{
        OnCommit(Q_ABORT);
   m_mes_sent -= m_tran_sent;
   m_rec_rec  -= m_tran_rec;
}

void COentryView::OnLButtonDown(UINT nFlags, CPoint point)
{
   CRect rec;
//      WINDOWPLACEMENT GetWindowPlacement  CRect
//      MessageBeep(0);
        GetDlgItem(IDC_COLORBOX)->GetWindowRect(&rec);
        ScreenToClient(&rec);
        if (rec.PtInRect(point)) OnColor();
        CFormView::OnLButtonDown(nFlags, point);
}


void COentryView::OnRButtonDown(UINT nFlags, CPoint point)
{
   ClearDisplay();
        CFormView::OnRButtonDown(nFlags, point);
}


void COentryView::ClearDisplay()
{
   GetParentFrame()->SetMessageText("");
   this->Invalidate();
   OnAnyUserAction();
}



void COentryView::OnShowdb()
{
        GetDlgItem(IDC_SHOWDB)->EnableWindow(FALSE);
   SetTimer(DB_DONE_TIMER,2000,NULL); // When the DB is done re-enable
  // CDbDlg dbd;
  // dbd.DoModal();
        dbd.Create(IDD_DBDLG);
        //GetParentFrame()->SetMessageText("After dbd.Create ");
```

```
}

void COentryView::OnOptionsb()
{
        GetDlgItem(IDC_OPTIONSB)->EnableWindow(FALSE);
   SetTimer(OPTIONS_DONE_TIMER,2000,NULL); // When the options are done re-en


//   o_dlg.Create(IDD_O_DLG);

   if (o_dlg.m_inst != m_inst) { // Init
      o_dlg.m_inst = m_inst;
//       o_dlg.m_parentptr = this;
            o_dlg.Create(IDD_O_DLG);
   }

   g_new_color = m_color;

   o_dlg.ShowWindow(SW_SHOW);

}
```

```
// oentrvw.h : interface of the COentryView class
//
//////////////////////////////////////////////////////////////////////////////
#define MESSAGEREADY WM_USER + 3000
//#define IDD_TPSTIMER          100

class COentryView : public CFormView
{
protected: // create from serialization only
        COentryView();
        DECLARE_DYNCREATE(COentryView)

public:
        //{{AFX_DATA(COentryView)
        enum { IDD = IDD_OENTRY_FORM };
        int             m_OrderMode;
        CString m_inst;
        BOOL    m_sendreply;
        //}}AFX_DATA
   BOOL     m_runflag;  // Running or not
   int      m_millisec; // Delay when ordering
   int      m_mes_sent;
   int      m_mes_rec;
   int      m_rec_sent;
   int      m_rec_rec;

   HBRUSH   m_brush;
   COLORREF m_color;
   COLORREF m_box_color;
   CString  m_que;
   CRect    m_color_box_rec;


   int      m_order;
   int      m_order_num;
   int      m_tran_state;
   int      m_tran_rec;
   int      m_tran_sent;
   int      m_auto_tran;
   int      m_auto_rand;
   int      m_auto_commit;
   void     AutoRun();
   void     color_the_box(COLORREF c, BOOL update);
   void     CmdLine(int pass);
   void     OnCommit(int action);
   void     OnAnyUserAction();
   void     OnColor();
   void     ShowRateBar(int act);
   void     PlaceOrFillMode(int mode);
   int      DbOrder(char *cust, char *item, int qty);
   int      OraOrder(char *cust, char *item, int qty);
   void     LoadQList(int mode);
   void     ClearDisplay();

   CDbDlg dbd;
   COdlg  o_dlg;


// Attributes
public:
        COentryDoc* GetDocument();

// Operations
public:

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(COentryView)
        public:
        virtual void OnInitialUpdate();
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
        virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
        virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);
        virtual void OnPrint(CDC* pDC, CPrintInfo*);
        virtual void OnDraw(CDC* pDC);
        //}}AFX_VIRTUAL

// Implementation
public:
        virtual ~COentryView();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
        //{{AFX_MSG(COentryView)
        afx_msg void OnExitb();
        afx_msg void OnOrderb();
        afx_msg void OnAutob();
        afx_msg HBRUSH OnCtlColor(CDC* pDC, CWnd* pWnd, UINT nCtlColor);
        afx_msg void OnPlacer();
        afx_msg void OnFillr();
        afx_msg void OnSelchangeQue();
        afx_msg long OnReplyMsg(WPARAM  wParam, LPARAM  lParam);
        afx_msg void OnSendrepc();
        afx_msg void OnTranb();
        afx_msg void OnCommitb();
        afx_msg void OnAbortb();
        afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
        afx_msg void OnTimer(UINT nIDEvent);
        afx_msg void OnShowdb();
        afx_msg void OnFilldb();
        afx_msg void OnPlacenoqr();
        afx_msg void OnEditupdateQue();
        afx_msg void OnOptionsb();
        afx_msg void OnRButtonDown(UINT nFlags, CPoint point);
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

#ifndef _DEBUG  // debug version in oentrvw.cpp
inline COentryDoc* COentryView::GetDocument()
   { return (COentryDoc*)m_pDocument; }
```

000305

```
#endif
////////////////////////////////////////////////////////////////////////////////
```

```
// oentrvw.cpp : implementation of the COentryView class
//
/*
**
** Copyright(C)1995 MITSUBISHI ELECTRIC ITA.  ALL RIGHTS RESERVED.
** UNPUBLISHED -- RIGHTS RESERVED UNDER THE COPYRIGHT
** LAWS OF THE UNITED STATES.  USE OF A COPYRIGHT NOTICE
** IS PRECAUTIONARY ONLY AND DOES NOT IMPLY PUBLICATION
** OR DISCLOSURE.
**
** THIS SOFTWARE CONTAINS CONFIDENTIAL INFORMATION AND
** TRADE SECRETS OF MITSUBISHI ELECTRIC ITA.  USE, DISCLOSURE,
** OR REPRODUCTION IS PROHIBITED WITHOUT THE PRIOR
** EXPRESS WRITTEN PERMISSION OF MITSUBISHI ELECTRIC ITA.
**
** OpenMQ Demo
**   Module: oenrvw.cpp
**   Author: Derek Schwenke 10/8/95
**
*/

#include "stdafx.h"
#include "oentry.h"

#include "oentrdoc.h"
#include "dbdlg.h"
#include "odlg.h"

#include "oentrvw.h"
#include "orderfm.h"

#include "oraomq.h"


//+++++++++++++ QLIB ++++++++++++++++++
#include "qlib.h"
#include "rt.h"
extern   lpSMBUFH sm_base;
lpQHANDLE   Q,Qrep;
//+++++++++++++ QLIB ++++++++++++++++++
#define REPLY_SUBMODE       1
#define NOMSGQ_SUBMODE      2
#define DIRECT_FILL_TIME_OVER 121
#define PLACE_ORDER_NOQ     0
#define PLACE_ORDER         1
#define FILL_ORDER          2
#define FILL_FROM_ORACLE    3

#define DB_DONE_TIMER       201
#define WAIT_ANAMATE_TIMER 202
#define DIRECT_FILL_TIMER  203
#define FILL_DELAY_TIMER    204
#define FILL_ANAMATE_TIMER 205
#define SET_1ST_TITLE_TIMER 206
#define OPTIONS_DONE_TIMER 207
#define POLL_FILL_TIMER     208
#define PLACE_TIMER         209
```

```
#define NEXTBIT(X) (1 & ( X = X / 2 ))

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// COentryView

IMPLEMENT_DYNCREATE(COentryView, CFormView)

BEGIN_MESSAGE_MAP(COentryView, CFormView)
	//{{AFX_MSG_MAP(COentryView)
	ON_BN_CLICKED(IDC_EXITB, OnExitb)
	ON_BN_CLICKED(IDC_ORDERB, OnOrderb)
	ON_BN_CLICKED(IDC_AUTOB, OnAutob)
	ON_WM_CTLCOLOR()
	ON_BN_CLICKED(IDC_PLACER, OnPlacer)
	ON_BN_CLICKED(IDC_FILLR, OnFillr)
	ON_CBN_SELCHANGE(IDC_QUE, OnSelchangeQue)
   ON_MESSAGE(MESSAGEREADY,OnReplyMsg)
	ON_BN_CLICKED(IDC_SENDREPC, OnSendrepc)
	ON_BN_CLICKED(IDC_TRANB, OnTranb)
	ON_BN_CLICKED(IDC_COMMITB, OnCommitb)
	ON_BN_CLICKED(IDC_ABORTB, OnAbortb)
	ON_WM_LBUTTONDOWN()
	ON_WM_TIMER()
	ON_BN_CLICKED(IDC_SHOWDB, OnShowdb)
	ON_BN_CLICKED(IDC_FILLDB, OnFilldb)
	ON_BN_CLICKED(IDC_PLACENOQR, OnPlacenoqr)
	ON_CBN_EDITUPDATE(IDC_QUE, OnEditupdateQue)
	ON_BN_CLICKED(IDC_OPTIONSB, OnOptionsb)
	ON_WM_RBUTTONDOWN()
	ON_CBN_SETFOCUS(IDC_QUE, OnSetfocusQue)
	//}}AFX_MSG_MAP
	// Standard printing commands
	ON_COMMAND(ID_FILE_PRINT, CFormView::OnFilePrint)
	ON_COMMAND(ID_FILE_PRINT_PREVIEW, CFormView::OnFilePrintPreview)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// COentryView construction/destruction

COentryView::COentryView()
	: CFormView(COentryView::IDD)
{
	//{{AFX_DATA_INIT(COentryView)
	m_OrderMode = 1;
	m_inst = _T("");
	m_sendreply = 0;
	//}}AFX_DATA_INIT
	// TODO: add construction code here
   m_runflag = 0;
   m_mes_sent = 0;
   m_mes_rec = 0;
   m_rec_sent = 0;
   m_rec_rec = 0;
```

```
    m_color = RGB(0,255,0);
    m_box_color = m_color;
    m_que = "";
    m_order_num = 0;
    m_tran_state = 0;
    m_auto_tran = 0;

}


    CString g_cust[] = {"Jones","James","Johnson","Jacobs","Jaffe","Jackson"};
    CString g_item[] = {"Bolts","Buckets","Buttons","Belts","Bobbins","Boats"};
    int  g_price[] = {1,2,3,4,5,6};
    int  g_qty[]   = {1000,1000,1000,1000,1000,1000};
    int  g_purchases[] = {0,0,0,0,0,0};
    int  g_num_purchases[] = {0,0,0,0,0,0};
    int  g_total_sales = 0;
    int  g_db_run = 0;
    int  g_options_run = 0;
  //  int  g_delay = 0;
    int  g_directplace = 0; // Client
    int  g_wait_anamate = 0;  // Server
    int  g_direct_fill = 0;  // Server
    int  g_fill_anamate = 0;  // Server
    int  g_filling = 0;  // Server

    int  g_place_tpm   = 60;
    int  g_place_delay = 1000;

    int  g_fill_delay  = 0;
    int  g_poll_pps    = 10;
    int  g_poll_delay  = 100;
    int  g_clear_stats = 0;
    int  g_ora_state = 0;
    int  g_max_orders = 0;
    char g_oracle_con_str[80] = "scott/tiger@t:grampa:orcl";

    COLORREF g_new_color;
    CFont    g_title_font;
    CFont    g_text_font;

    int  WAITS[] = {IDC_WAIT0,IDC_WAIT1};
    int  FILLS[] = {IDC_FILL0,IDC_FILL1,IDC_FILL2};
    int  ALL_TEXT[] = { IDC_MODEBOX,IDC_PLACER,IDC_PLACENOQR,IDC_FILLR,IDC_FILLDB
    IDC_OPTIONSB, IDC_ORDERBOX,IDC_FILLTXT,IDC_SENDREPC,IDC_ORDERB,IDC_AUTOB,IDC_
    IDC_QUE,IDC_CUST,IDC_ITEM,IDC_QTY,
    IDC_QUELAB,IDC_CUSTLAB,IDC_ITEMLAB,IDC_QTYLAB,
    IDC_TRANBOX,IDC_TRANB,IDC_COMMITB,IDC_ABORTB,IDC_EXITB,
    IDC_STATBOX,IDC_RECIPTS_LAB,IDC_RECIPTS,IDC_RECIPT,IDC_MSGS_LAB,IDC_MSGS,0};


    lpQHANDLE g_LQ; // Listener que global for close OnExit()


COentryView::~COentryView()
{
}

void COentryView::DoDataExchange(CDataExchange* pDX)
{
        CFormView::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(COentryView)
        DDX_Radio(pDX, IDC_PLACER, m_OrderMode);
//      DDX_Text(pDX, IDC_INST, m_inst);
        DDX_Check(pDX, IDC_SENDREPC, m_sendreply);
        //}}AFX_DATA_MAP
}

/////////////////////////////////////////////////////////////////////////////
// COentryView printing

BOOL COentryView::OnPreparePrinting(CPrintInfo* pInfo)
{
        // default preparation
        return DoPreparePrinting(pInfo);
}

void COentryView::OnBeginPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
        // TODO: add extra initialization before printing
}

void COentryView::OnEndPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
        // TODO: add cleanup after printing
}

void COentryView::OnPrint(CDC* pDC, CPrintInfo*)
{
        // TODO: add code to print the controls
}

/////////////////////////////////////////////////////////////////////////////
// COentryView diagnostics

#ifdef _DEBUG
void COentryView::AssertValid() const
{
        CFormView::AssertValid();
}

void COentryView::Dump(CDumpContext& dc) const
{
        CFormView::Dump(dc);
}

COentryDoc* COentryView::GetDocument() // non-debug version is inline
{
        ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(COentryDoc)));
        return (COentryDoc*)m_pDocument;
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// COentryView message handlers

void COentryView::OnExitb()
{
        CString s;
```

```
   if (g_LQ) Qclose(&g_LQ,0);
   if (m_runflag) {
      m_runflag = 0;
      while( s != "Auto" ) { // Read the button until not runnning
         GetDlgItemText(IDC_AUTOB,s.GetBuffer(100),100);
      }
   }

   AfxGetMainWnd()->DestroyWindow();
}



//GetParentFrame()->SetMessageText("Unknown message");
long COentryView::OnReplyMsg(WPARAM  wParam, LPARAM  lParam){
   CString s,ss;

   switch(wParam) {
      case REPLY_SUBMODE:
         if (lParam)
            SetDlgItemText(IDC_RECIPT,(char *)lParam);
         else
            SetDlgItemText(IDC_RECIPT,"OnReplyMsg Unknown REPLY_SUBMODE Message"
         SetDlgItemInt(IDC_RECIPTS,++m_rec_rec);

      break;
      case NOMSGQ_SUBMODE:
         if (lParam)
            MessageBox("NOMSGQ_SUBMODE How did I get here?");

            /*

            SetDlgItemInt(IDC_MSGS,++m_mes_rec);

            if (g_delay) {
               SetTimer(FILL_DELAY_TIMER,g_delay,NULL);          //   g_direct_f
               SetTimer(FILL_ANAMATE_TIMER,100,NULL);
            } else
               g_direct_fill = 0;

            int instock = DbOrder(((pOFORM)lParam)->cust, ((pOFORM)lParam)->item

            if (instock == 1)       ss = "FILLED";
            else if (instock == 0)  ss = "OUT OF STOCK";
            else if (instock == -1) ss = "WRONG CUSTOMER";
            else if (instock == -2) ss = "WRONG ITEM";


            s.Format("Direct: %s: %s %s %d",LPCTSTR(ss),((pOFORM)lParam)->cust,(
            SetDlgItemText(IDC_FILLTXT,LPCTSTR(s));
            color_the_box( ((pOFORM)lParam)->color,0);
          */

      break;
      case DIRECT_FILL_TIME_OVER:
            GetParentFrame()->SetMessageText(lParam);


            // SetDlgItemInt(IDC_MSGS,--m_mes_sent);
      break;
      default:
         if (lParam)
            s.Format("ORM: %d %s",wParam,lParam);
         else
            s.Format("Unknown Message ORM");
         GetParentFrame()->SetMessageText(s);
   }
   return(0);
}



/////////////////////////////////////////////////////////////////////////////
///////////////////////   LISTENER THREAD  //////////////////////////////
/////////////////////////////////////////////////////////////////////////////

typedef struct tparam {  // Thread parameters
   HWND     hwnd;
   int      bnum; // smbuff num to send replys to.
   CString inst;
} TPARAM, *pTPARAM;

   TPARAM tp;  // <-- Global Data so it does not go away.



char g_thd_data[1000];         //////////////////////
MSGH g_thd_mh;                 // Listener thread //
DWORD Listen(pTPARAM tp) {     //////////////////////
   int  max_dly,md,stat=0;
   CString s,n = tp->inst;


   if ( g_LQ = Qopen(n.GetBuffer(0),GET_MODE,0,  0,&stat,0,0)) {
      // Save the buffer I will listen to.
      tp->bnum = g_LQ->bufs_found[0]; // Replys will be sent to this buffer

      if (SHAREDATA(diag)) {
         s.Format("Thread Open(%s) bnum=%d",LPCTSTR(n),tp->bnum);
         ::PostMessage( tp->hwnd,MESSAGEREADY,0,(LPARAM) LPCTSTR(s) );
      }

/*                                 // REPLY_SUBMODE (pass sub_mode as ORM_MES_REC
      while(QSUCCESS == Qget(g_LQ,&g_thd_mh,data,1000) ) {
         ::PostMessage(tp->hwnd,MESSAGEREADY, g_thd_mh.sub_mode ,(LPARAM) data);
//          ::PostMessage(tp->hwnd,MESSAGEREADY,ORM_MES_RECEIVED,(LPARAM) data);
*/


      while(QSUCCESS == QlistenBeforeReply(g_LQ,&g_thd_mh,g_thd_data,1000) ) {
         // Got a message could be Recipt or direct fill request
         if (g_thd_mh.sub_mode == REPLY_SUBMODE) {
            ::PostMessage(tp->hwnd,MESSAGEREADY, g_thd_mh.sub_mode ,(LPARAM) g_c

         } else {
            g_direct_fill = 1;
            max_dly = 500; // 25 Seconds max   check 20 times a second
            while ((g_direct_fill || g_filling) && (--max_dly))  Sleep(50); //

            if (max_dly > 0)
               md = ACK_MODE;
            else {
               md = NACK_MODE;
               g_direct_fill = 0;
               MessageBox(NULL,"SER TIME OVER",NULL,NULL);
               s.Format("%s: SERVER did not reply to DIRECT FILL",n);
              ::PostMessage(tp->hwnd,MESSAGEREADY, DIRECT_FILL_TIME_OVER ,(LPARA

            }
         }
         QreplyAfterListen(g_LQ,md,0,g_thd_data,g_thd_mh.size,0);
      }

   } else {
      s.Format("Cant Qopen(%s) ERROR=%d",LPCTSTR(n),stat);
      MessageBox(tp->hwnd,s,"Listener Thread",MB_ICONSTOP);
   }

   Qclose(&g_LQ,0);
   return(0);
}



void COentryView::CmdLine(int pass){
   CString parm,value,line,nline;
   int i,r,g,b;

   line = AfxGetApp()->m_lpCmdLine;

   while (2 <= (i = sscanf(LPCTSTR(line),"%s %s %[^@]",
      parm.GetBuffer(100),value.GetBuffer(100),nline.GetBuffer(100) ))) {
      parm.MakeUpper();

// oentry Name xxx Que ssss RGB r,g,b OrderMode m Fill x Poll x Order x
      if (pass == 1) {

         if (parm == "NAME")   {m_inst = LPCTSTR(value);}
         if (parm == "RGB")    {sscanf(LPCTSTR(value),"%d,%d,%d",&r,&g,&b);m_colo
         if (parm == "ORDERMODE") {sscanf(LPCTSTR(value),"%d",&m_OrderMode);m_Or
                               if (m_OrderMode < 0 || m_OrderMode > 3) m_OrderMod
         if (parm == "FILL")  {sscanf(LPCTSTR(value),"%d",&g_fill_delay);}
         if (parm == "POLL")  {sscanf(LPCTSTR(value),"%d",&g_poll_pps);
                               if((MIN_POLL <= g_poll_pps) && (g_poll_pps <= MAX_
                                  g_poll_delay  = 1000/g_poll_pps;}
         if (parm == "ORDER") {sscanf(LPCTSTR(value),"%d",&g_place_tpm);
                               if((MIN_AUTO <= g_place_tpm) && (g_place_tpm <= MA
                                  g_place_delay = 60000/g_place_tpm;}
         if (parm == "ORACLE") { strcpy(g_oracle_con_str ,LPCTSTR(value)); }

      } else {
         if (parm == "QUE" || parm == "SERVER") {
            CComboBox * CB = (CComboBox *) this->GetDlgItem(IDC_QUE);
            CB->SelectString(-1,value);
            OnSelchangeQue();
         }
      }

      line = LPCTSTR(nline);
      nline = "";
   }
}


void COentryView::LoadQList(int mode)
{
   // mode=0 : List Non-Ques
   // mode=1 : List Ques
   int i;

   CComboBox * CB = (CComboBox *) this->GetDlgItem(IDC_QUE);
   CB->ResetContent();

// List all logical queues
// APPS: [physical],logical1,logical2,[physical],logical,
   if (sm_base = AttachSharedMemory()){
      lpRT  rt = RTROOT;
      while(rt = NextRT(rt)) {
         char *e,*s = RT_APPS(rt);    // Starts after the first letter
         while (s = strchr(s,',')) {  // Ends at next ","
            if (e = strchr(++s,',')) {
               *e = 0;
               if ((!strchr(s,'[')) && *s) {
                  if ((mode && strchr(s,'Q')) || (!mode && !strchr(s,'Q')))
                     CB->AddString(s); // lb->InsertString(-1,s)
               }
               *e = ',';
            }
         }
      }

      // Now add any extra entries from the local node that were not in the RT
  // lpSMBUF  BUF;
      for (i = 0; i < SHAREDATA(nsbuf) ; i++) {
         if (!strcmp((SMBUFADDR(i))->name,"empty")) continue;
         if (!strcmp((SMBUFADDR(i))->name,"QNETD")) continue;
         if (CB_ERR != CB->FindStringExact(-1,(SMBUFADDR(i))->name)) continue; /
         if ((mode && strchr((SMBUFADDR(i))->name,'Q')) || (!mode && !strchr((SM
            CB->AddString((SMBUFADDR(i))->name); // lb->InsertString(-1,s)
      }

   } else {
      MessageBox("QNETD not running. Please restart.");
   }
   CB->AddString("");

   if ( CB_ERR == (i = CB->FindStringExact(-1,m_que)))
      CB->SetCurSel(0);
   else
      CB->SetCurSel(i);

   OnSelchangeQue();
```

```
}



void COentryView::OnInitialUpdate()
{
        CString s;
        CFormView::OnInitialUpdate();

   // Set frame size = Form size
   GetParentFrame()->RecalcLayout();
   ResizeParentToFit(FALSE);
   ResizeParentToFit(TRUE);

   m_OrderMode = PLACE_ORDER;

   LoadQList(1);

   CmdLine(1);

   // Set max size on data fields
   (  (CEdit *) this->GetDlgItem(IDC_CUST)   )->LimitText(9);
   (  (CEdit *) this->GetDlgItem(IDC_ITEM)   )->LimitText(9);
   (  (CEdit *) this->GetDlgItem(IDC_QTY)    )->LimitText(9);

   // Generate unique instance name on this node
   if (m_inst == "") // May be set by command line "name"

      m_inst.Format("OE%d",SHAREDATA(ap.app_num)++);

   OnAnyUserAction();

   SetTimer(SET_1ST_TITLE_TIMER,100,NULL);  // Calls OnAnyUserAction(); to set t


   // Start the backgound thread to Qget() messages
   DWORD id;
   tp.hwnd = m_hWnd;
   tp.inst = m_inst;
   CreateThread(NULL,0,(LPTHREAD_START_ROUTINE) Listen, (LPVOID) &tp,0,&id);


   CmdLine(2);

   ( (CButton *) this->GetDlgItem(IDC_PLACER) )->SetCheck(0);
   ( (CButton *) this->GetDlgItem(IDC_PLACENOQR) )->SetCheck(0);
   ( (CButton *) this->GetDlgItem(IDC_FILLR) )->SetCheck(0);
   ( (CButton *) this->GetDlgItem(IDC_FILLDB) )->SetCheck(0);

   if (m_OrderMode == PLACE_ORDER)
      ( (CButton *) this->GetDlgItem(IDC_PLACER) )->SetCheck(1);
   if (m_OrderMode == PLACE_ORDER_NOQ)
      ( (CButton *) this->GetDlgItem(IDC_PLACENOQR) )->SetCheck(1);
   if (m_OrderMode == FILL_ORDER)
      ( (CButton *) this->GetDlgItem(IDC_FILLR) )->SetCheck(1);
   if (m_OrderMode == FILL_FROM_ORACLE)
      ( (CButton *) this->GetDlgItem(IDC_FILLDB) )->SetCheck(1);

#ifdef ORACLE
```

```
      ((CButton *) this->GetDlgItem(IDC_FILLDB))->EnableWindow(TRUE);
#else
      ((CButton *) this->GetDlgItem(IDC_FILLDB))->EnableWindow(FALSE);
#endif


   PlaceOrFillMode(m_OrderMode);


   // Fonts
   LOGFONT lf;
   memset(&lf,0,sizeof(LOGFONT));

#ifdef BIGFONT
   lf.lfHeight = 18;
#else
   lf.lfHeight = 13;
#endif
   g_text_font.CreateFontIndirect(&lf);

        strcpy(lf.lfFaceName,"Monotype Corsiva");
   lf.lfHeight = 32;
   g_title_font.CreateFontIndirect(&lf);


   GetDlgItem(IDC_BIG_TITLE)->SetFont(&g_title_font);

   int i = 0;
   while (ALL_TEXT[i])
      GetDlgItem(ALL_TEXT[i++])->SetFont(&g_text_font);
}



//void COentryView::OnEditchangeQue()
void COentryView::OnEditupdateQue()
{
        OnSelchangeQue();
}

void COentryView::OnSelchangeQue()
{
   CString que,s;
   int stat=0;

   // Get the que name
   GetDlgItemText(IDC_QUE,que.GetBuffer(100),100);
   que.ReleaseBuffer();


   if (que != m_que) { // New value
      if (Q) Qclose(&Q,0);

      if (que == "") { // No que
         m_que = "";
         s = "No Que will be used";
         GetDlgItem(IDC_AUTOB)->EnableWindow(FALSE);
         GetDlgItem(IDC_ORDERB)->EnableWindow(FALSE);

      } else {          // Set value
```

```
            s.Format("Opening que %s",que);
            GetParentFrame()->SetMessageText(s);

            if (Q = Qopen(que.GetBuffer(0),PUT_MODE,0,0,&stat,0,0) ) {
               if (m_sendreply)
                  Q->msgh.reply_smbuf = tp.bnum;
               m_que = que;
               s.Format("Qopen(%s)",que);

               GetDlgItem(IDC_AUTOB)->EnableWindow(TRUE);
               GetDlgItem(IDC_ORDERB)->EnableWindow(TRUE);

               if (m_OrderMode == PLACE_ORDER_NOQ)
                  Q->msgh.sub_mode = NOMSGQ_SUBMODE;


            } else {
               s.Format("Qopen(%s) Error %d",que,stat);
               GetDlgItem(IDC_AUTOB)->EnableWindow(FALSE);
               GetDlgItem(IDC_ORDERB)->EnableWindow(FALSE);

            }
         }
         GetParentFrame()->SetMessageText(s);

   }
   OnAnyUserAction();
}

OFORM g_buf ;
DWORD DirectPlace() {
   int flags = 0;
   QsendAndReceive(Q,0,SUB_MODE_OK, flags,sizeof(g_buf),(char *) &g_buf,  0,0,0,
   g_directplace = 0;
   return(0);
}

//ORDERORDERORDERORDERORDERORDERORDERORDERORDERORDERORDERORDERORDERORDER
//ORDERORDERORDERORDERORDERORDERORDERORDERORDERORDERORDERORDERORDERORDER
//ORDERORDERORDERORDERORDERORDERORDERORDERORDERORDERORDERORDERORDERORDER
//ORDERORDERORDERORDERORDERORDERORDERORDERORDERORDERORDERORDERORDERORDER
//ORDERORDERORDERORDERORDERORDERORDERORDERORDERORDERORDERORDERORDERORDER

void COentryView::OnOrderb()
{  // Check m_ordermode = 0 to place order or = 1 to fill the order
         CString que,str,s = "";
   int   gsize,stat=0,flags,instock,filled=0;
//    OFORM buf ;
   char  line[sizeof(OFORM)+10];
   MSGH  mh;  // To get detailed put status
   DWORD id;

   if (g_directplace) return; // direct place thread is already working
   if (g_filling) return; // Wait for fill done is already working
   if (Q) {
      if (!m_runflag) OnAnyUserAction();

      if (m_auto_tran) {
         if(m_auto_rand < 16) m_auto_rand = rand();
      // Begin a tran?
      if ((m_auto_tran == 1) && NEXTBIT(m_auto_rand)) {
              if (!m_tran_state) m_tran_state= 1;  // Done by OnTran (causes
         m_auto_tran = 2;
         GetDlgItem(IDC_TRANB)->ShowWindow(SW_HIDE);

         if (m_auto_commit = NEXTBIT(m_auto_rand))
            GetDlgItem(IDC_COMMITB)->ShowWindow(SW_SHOW);
         else
            GetDlgItem(IDC_ABORTB)->ShowWindow(SW_SHOW);

      }
   }


   // Set the flags for get or put
   if (m_tran_state)
      if (m_tran_state++ == 1)
         flags = Q_TRAN_BEGIN;
      else
         flags = Q_TRAN;
   else
      flags = 0;


   if (m_OrderMode <= PLACE_ORDER) { // Place the order

      // Read the data to be sent
      GetDlgItemText(IDC_CUST,g_buf.cust,10);
      GetDlgItemText(IDC_ITEM,g_buf.item,10);
      g_buf.qty = GetDlgItemInt(IDC_QTY,NULL,TRUE);
      g_buf.color = m_color;
      g_buf.reply_to = SHAREDATA(hostip);

      SetDlgItemInt(IDC_MSGS,++m_mes_sent);
      if (m_OrderMode == PLACE_ORDER_NOQ) {
         g_directplace = 1;
         GetDlgItem(IDC_ORDERB)->EnableWindow(FALSE);

         SetTimer(WAIT_ANAMATE_TIMER,400,NULL);
         CreateThread(NULL,0,(LPTHREAD_START_ROUTINE) DirectPlace, (LPVOID) 0

      } else if (QSUCCESS == (stat = QsendAndReceive(Q,0,SUB_MODE_OK, flags,s

         if (mh.mode == ACK_MODE && mh.sub_mode == SUB_MODE_OK){
            m_tran_sent++;
            s.Format("%s: %8s %8s %d",m_que.GetBuffer(0),g_buf.cust,g_buf.ite
         } else {
            if (mh.sub_mode == SUB_MODE_FULL)
               if (m_runflag)
                  s.Format("que is FULL");
               else
                  MessageBox("que is FULL");
            else
               s.Format("Qput() error mode=%d submode=%d",mh.sub_mode);
         }
      } else {  // not QSUCCESS (communications error)
         SetDlgItemInt(IDC_MSGS,--m_mes_sent);
         s.Format("Communication error code %d",stat);
      }
      GetParentFrame()->SetMessageText(s);

    } else { // FILL_ORDER Fill the order
      *g_buf.cust = *g_buf.item = 0;
      g_buf.qty = 0;

      if (g_direct_fill) {  // There was a direct placed order to fill
         pOFORM po = (pOFORM) &g_thd_data;
         CString ss;

         SetDlgItemInt(IDC_MSGS,++m_mes_rec);

         if (g_fill_delay) {
            g_filling = 1; // Do not fill another until the timer expires
            SetTimer(DIRECT_FILL_TIMER,g_fill_delay,NULL);          //   g_di
            SetTimer(FILL_ANAMATE_TIMER,100,NULL);
         }

         color_the_box( po->color,0);
         int instock = DbOrder(po->cust, po->item, po->qty);

         if (instock == 1)       ss = "FILLED";
         else if (instock == 0)  ss = "OUT OF STOCK";
         else if (instock == -1) ss = "WRONG CUSTOMER";
         else if (instock == -2) ss = "WRONG ITEM";
         filled++;

         s.Format("Direct: %s: %s %s %d",LPCTSTR(ss), po->cust, po->item, po-
         SetDlgItemText(IDC_FILLTXT,LPCTSTR(s));


         g_direct_fill = 0;
         if (g_filling) return;
      }



      if (QSUCCESS == (stat = QsendAndReceive(Q,REQUEST_MODE,SUB_MODE_OK,flag
         sizeof(g_buf),(char *) &g_buf,&gsize,&mh))) {

         if (mh.mode == ACK_MODE && mh.sub_mode == SUB_MODE_OK) {
            SetDlgItemInt(IDC_MSGS,++m_mes_rec);
            if (gsize < sizeof(OFORM))
               s.Format("Bad reply length");
            else { // you got valid message

               m_tran_rec++;
               color_the_box(g_buf.color,0); // But dont force it

               if (m_OrderMode == FILL_FROM_ORACLE)
                  instock = OraOrder(g_buf.cust,g_buf.item,g_buf.qty); // Ora
               else
                  instock = DbOrder(g_buf.cust,g_buf.item,g_buf.qty);
               if (g_fill_delay) {
                  g_filling = 1; // Do not fill another until the timer ex
                  SetTimer(FILL_DELAY_TIMER,g_fill_delay,NULL);         //
                  SetTimer(FILL_ANAMATE_TIMER,100,NULL);
               }
            }
      } else if (mh.sub_mode == SUB_MODE_EMPTY) {
         if (m_runflag || filled)
            s.Format("Que Empty");
         else
            MessageBox("Que Empty");
      } else
         s.Format("Bad reply mode %d:%d",mh.mode,mh.sub_mode);

   } else { // not QSUCCESS (communications error)
      mh.mode = 0;// Not ACK_MODE
      s.Format("Communication Error Code %d",stat);
   }


   if (gsize == sizeof(OFORM)) {
      if (instock == 1 )
         sprintf(line,"FILLED:%9s %9s %d",g_buf.cust,g_buf.item,g_buf.qty)
      else if ( instock == 0)
         sprintf(line,"OUT OF STOCK:%9s %9s %d",g_buf.cust,g_buf.item,g_bu
      else if ( instock == -1)
         sprintf(line,"NO SUCH ITEM:%9s %9s %d",g_buf.cust,g_buf.item,g_bu
      else // instock == -2
         sprintf(line,"NO SUCH CUSTOMER:%9s %9s %d",g_buf.cust,g_buf.item,
      SetDlgItemText(IDC_FILLTXT,line);
   }
     // else
     // strcpy(line,"");


   // Send Reply if requested
   if ((mh.reply_smbuf >= 0 ) && (mh.reply_smbuf < SHAREDATA(nsbuf)) &&
      mh.mid.host = g_buf.reply_to; // HACK! Makes sure sending host gets
      Qrep = QopenReply(Qrep,&mh,0,0,0);


      if (QSUCCESS != (stat = Qput(Qrep,0,REPLY_SUBMODE,0,sizeof(line),lin
         s.Format("Qput(%s) Error %d",Qrep->msgh.to_server,stat);
      else
         s.Format("Replying Qput(%s) to smbuf %d",Qrep->msgh.to_server,Qre
      SetDlgItemInt(IDC_RECIPTS,++m_rec_sent);

   }

   GetParentFrame()->SetMessageText(s);
 /*
   if (m_runflag && mh.mode == ACK_MODE) // We got something
      OnOrderb(); // Check again for next message, no need to wait.
   else // We didnt get anything
      color_the_box( m_color,0); // restore old color
   */
   if (!m_runflag && !g_fill_delay) color_the_box( m_color,0);


} // place or fill


if ((m_OrderMode <= PLACE_ORDER) || mh.mode == ACK_MODE) // Placeing or go
if (m_auto_tran > 1) {
   if ( ( NEXTBIT(m_auto_rand) && NEXTBIT(m_auto_rand) )
      || (m_auto_tran++ > 8) ) {
            // Commit or Abort tran?
            if (m_auto_commit)
               OnCommitb();
            else
               OnAbortb();
            m_auto_tran = 1;
         }
      }


      if ((m_OrderMode >= FILL_ORDER) && m_runflag &&
         mh.mode == ACK_MODE && mh.sub_mode == SUB_MODE_OK)
         if (g_max_orders++ < 200)
                        OnOrderb(); // Try to get one more item.
               else
                         g_max_orders = 0;
      } else
                  g_max_orders = 0;

   // The order or fill button will be active until we are done.
   // CButton * CB = (CButton *) this->GetDlgItem(IDC_ORDERB);
   // CB->SetState(TRUE);

}

//
// m_auto_tran:
// 0 No auto trans
// 1 No tran yet
// 2 After OnAutob()
//
int g_poll_delay_old;
int g_place_delay_old;
void COentryView::OnTimer(UINT nIDEvent)
{


        if (nIDEvent == PLACE_TIMER ) {

      if (g_directplace) return; // direct place thread is already working
      if (g_filling) return; // Wait for fill done is already working
      if (g_place_delay_old != g_place_delay) {
         KillTimer(PLACE_TIMER);
         SetTimer(PLACE_TIMER,g_place_delay_old = g_place_delay,NULL);
      }

      SetDlgItemText(IDC_CUST,g_cust[m_order_num % 6]);
      SetDlgItemText(IDC_ITEM,g_item[m_order_num % 5]);
      SetDlgItemInt(IDC_QTY,1 + (m_order_num++ % 9),FALSE);

      OnOrderb();
      //CButton * CB = (CButton *) this->GetDlgItem(IDC_ORDERB);
      //CB->SetState(TRUE);


        } else if (nIDEvent == POLL_FILL_TIMER) {

      if (!g_filling) {
         if (g_poll_delay != g_poll_delay_old) {
          KillTimer(POLL_FILL_TIMER);
          SetTimer(POLL_FILL_TIMER,g_poll_delay_old = g_poll_delay,NULL);
       }

       OnOrderb();
    }

      } else if (nIDEvent == WAIT_ANAMATE_TIMER) {

    if (g_directplace) {

       // Anomate the wait  0,1 0,1
       GetDlgItem(WAITS[g_wait_anamate++])->ShowWindow(SW_HIDE);
       if (g_wait_anamate > 1) g_wait_anamate = 0;
       GetDlgItem(WAITS[g_wait_anamate])->ShowWindow(SW_SHOW);
    } else {
       KillTimer(WAIT_ANAMATE_TIMER);
       GetDlgItem(IDC_ORDERB)->EnableWindow(TRUE);
       GetDlgItem(WAITS[g_wait_anamate])->ShowWindow(SW_HIDE);
    }

      } else if (nIDEvent == FILL_ANAMATE_TIMER) {

    if (g_filling == 0) {    // End Anamation
       KillTimer(FILL_ANAMATE_TIMER);
       GetDlgItem(FILLS[g_fill_anamate])->ShowWindow(SW_HIDE);

    } else { // Anamate  0,1,2

       GetDlgItem(FILLS[g_fill_anamate++])->ShowWindow(SW_HIDE);
       if (g_fill_anamate > 2) g_fill_anamate = 0;
       GetDlgItem(FILLS[g_fill_anamate])->ShowWindow(SW_SHOW);

       if (1) {  // g_progress
            // ( (CButton *) this->GetDlgItem(IDC_PLACER) )->SetCheck(0);
//          GetDlgItem(IDC_PROG))->SetRange(0,100);
 //          ((CProgressCtrl *) this->GetDlgItem(IDC_PROG))->SetRange(0,100);
 //          ((CProgressCtrl *) this->GetDlgItem(IDC_PROG))->SetPos(50);
 //          ((CProgressCtrl *) this->GetDlgItem(IDC_PROG))->ShowWindow(SW_SHOW
       }



    }

      } else if (nIDEvent == DIRECT_FILL_TIMER) {

    g_direct_fill = 0; // let Qsar() return.
    g_filling = 0;     // End Anamation, let next order in
    KillTimer(DIRECT_FILL_TIMER);
  color_the_box( m_color,0);   // Restore the old color

  } else if (nIDEvent == FILL_DELAY_TIMER) {

    g_filling = 0;      // End Anamation, let next order in
    KillTimer(FILL_DELAY_TIMER);
  color_the_box( m_color,0);   // Restore the old color

      } else if (nIDEvent == SET_1ST_TITLE_TIMER) {  // Only done once
```

```
            KillTimer(SET_1ST_TITLE_TIMER);
            OnAnyUserAction();

    } else if (nIDEvent == DB_DONE_TIMER) {
        if (g_db_run == 40) {
            g_db_run = 0;
            GetDlgItem(IDC_SHOWDB)->EnableWindow(TRUE);
            KillTimer(DB_DONE_TIMER);
        }
          } else if (nIDEvent == OPTIONS_DONE_TIMER) {

        if (g_options_run == 40) { // End
            g_options_run = 0;
            GetDlgItem(IDC_OPTIONSB)->EnableWindow(TRUE);
            KillTimer(OPTIONS_DONE_TIMER);
        }

        if (m_color != g_new_color) {
            color_the_box(m_color = g_new_color,1);
        }

        if (g_clear_stats) {
            g_clear_stats = 0;
            m_mes_sent = 0;
            m_mes_rec = 0;
            m_rec_sent = 0;
            m_rec_rec = 0;
            SetDlgItemInt(IDC_MSGS,     0);
            SetDlgItemInt(IDC_RECIPTS, 0);
        }
    } else

          CFormView::OnTimer(nIDEvent);
}


void COentryView::OnAutob()
{
    if (m_runflag) {
        KillTimer(PLACE_TIMER);
        KillTimer(POLL_FILL_TIMER);
        m_runflag = 0; // You are no longer running
        if (m_auto_tran) OnAbortb();
        m_auto_tran = 0;
        SetDlgItemText(IDC_AUTOB,"Auto");


        GetDlgItem(IDC_EXITB)->EnableWindow(TRUE);
        ((CButton *) this->GetDlgItem(IDC_PLACER))->EnableWindow(TRUE);
        ((CButton *) this->GetDlgItem(IDC_PLACENOQR))->EnableWindow(TRUE);
        ((CButton *) this->GetDlgItem(IDC_FILLR))->EnableWindow(TRUE);
#ifdef ORACLE
        ((CButton *) this->GetDlgItem(IDC_FILLDB))->EnableWindow(TRUE);
#endif
        color_the_box( m_color,1);  // Restore default color

    } else {
        m_runflag = 1;    // You will be running
        m_order_num = 0; // 1st order number Controls order selections
        m_auto_tran = m_tran_state; // make random transactions too?.
```

```
      if ( m_OrderMode > PLACE_ORDER ) // FILL
         SetTimer(POLL_FILL_TIMER,g_poll_delay_old = g_poll_delay,NULL);
      else
         SetTimer(PLACE_TIMER,g_place_delay_old = g_place_delay,NULL);


      SetDlgItemText(IDC_AUTOB,"    Stop    ");

      GetDlgItem(IDC_EXITB)->EnableWindow(FALSE);
      ( (CButton *) this->GetDlgItem(IDC_PLACER) )->EnableWindow(FALSE);
      ( (CButton *) this->GetDlgItem(IDC_PLACENOQR) )->EnableWindow(FALSE);
      ( (CButton *) this->GetDlgItem(IDC_FILLR) )->EnableWindow(FALSE);
      ( (CButton *) this->GetDlgItem(IDC_FILLDB) )->EnableWindow(FALSE);
   }
   OnAnyUserAction();
}


// Color Color Color Color Color Color Color Color Color Color
// Color Color Color Color Color Color Color Color Color Color
// Color Color Color Color Color Color Color Color Color Color


 void COentryView::OnColor()
{
   CHOOSECOLOR cc;     // common dialog box structure
   COLORREF acrCustClr[16];

   // Setup the custom colors as a grey scale
   for (int v=0,i=0; i < 16; v=17 * i++)
      acrCustClr[i] = RGB(v,v,v);

   // Initialize the necessary members.
   cc.lStructSize = sizeof(CHOOSECOLOR);
   cc.hwndOwner = NULL; //  = hwnd;
   cc.lpCustColors = (LPDWORD) acrCustClr;
   cc.Flags = CC_FULLOPEN;  //  CC_PREVENTFULLOPEN


   if (ChooseColor(&cc)){
      CString s;

      m_box_color = m_color = cc.rgbResult; // lpCustColors
      Invalidate(); // Display the new color
   } else {
      GetParentFrame()->SetMessageText("Color was not changed");
   }
   OnAnyUserAction();
}


//Hinit = 1;
HBRUSH COentryView::OnCtlColor(CDC* pDC, CWnd* pWnd, UINT nCtlColor)
{

   if (nCtlColor == CTLCOLOR_EDIT) {
      if (pWnd->GetDlgCtrlID() == IDC_COLORBOX) {
            pDC->SetBkColor(m_color);
            m_brush = CreateSolidBrush(m_color);
            return(m_brush);

         }
      } else if (nCtlColor == CTLCOLOR_BTN ) {  // CTLCOLOR_EDIT
         if (pWnd->GetDlgCtrlID() == IDC_AUTOB)
         if (m_runflag) {
            pDC->SetBkColor(RGB(255,0,0));          //  SetBkColor SetTextColor
            pDC->SetTextColor(RGB(255,255,255));    //  SetBkColor SetTextColor
            m_brush = CreateSolidBrush(m_color);
            return(m_brush);
         }
//    } else if (nCtlColor == CTLCOLOR_STATIC ) {  //  CTLCOLOR_EDIT
//       if (pWnd->GetDlgCtrlID() == IDC_INST){
//          // pDC->SetBkColor(RGB(255,0,0));          //  SetBkColor SetTextColor
//          // pDC->SetTextColor(RGB(255,255,255));    //  SetBkColor SetTextColor
//          // m_brush = CreateSolidBrush(m_color);
//          m_brush = CreateSolidBrush(m_box_color); // empty non-text background
//          pDC->SetBkColor(m_box_color);           // behind the letters
//          return(m_brush);
//       }
      }

   HBRUSH hbr = CFormView::OnCtlColor(pDC, pWnd, nCtlColor);

         return hbr;
}


int COentryView::DbOrder(char *cust, char *item, int qty) {
   int custn=0,itemn=0;
   while(strcmp(item,g_item[itemn])) if (++itemn == 6) return(-1);
   while(strcmp(cust,g_cust[custn])) if (++custn == 6) return(-2);

   if (g_qty[itemn] >= qty) {
      g_qty[itemn] -= qty;
      g_purchases[custn] += qty * g_price[itemn];
      g_num_purchases[custn]++;
      g_total_sales += qty * g_price[itemn];

   } else
      return(0);
   return(1);
}



int COentryView::OraOrder(char *cust, char *item, int qty) {
   int custn=0,itemn=0, price,stock,rc,cust_orders,cust_sales;
   CString s;
   while(strcmp(item,g_item[itemn])) if (++itemn == 6) return(-1);
   while(strcmp(cust,g_cust[custn])) if (++custn == 6) return(-2);


   if (g_ora_state == 0) {
      GetParentFrame()->SetMessageText("Cant Order from oracle: not connected");
      return(0);
   }

   // Read Oracle
   if (  (rc = oraread(item, &price, &stock))  ) {
      s.Format("OraRead Error %d",rc);
      GetParentFrame()->SetMessageText(s);
      return(0);
   }
   if (  (rc = oracustr(cust, &cust_orders, &cust_sales))  ) {
      s.Format("OraCustRead Error %d",rc);
      GetParentFrame()->SetMessageText(s);
   }


   if (qty > stock) {
      return(0);  // MessageBox("OutOfStock");
   }  else {

      stock -= qty;

      g_total_sales += qty * price;
      cust_sales += qty * price;
      cust_orders++;


      // Write Oracle
      if ( (rc = orawrite(item, stock))  ) {
         s.Format("OraWrite Error %d",rc);
         GetParentFrame()->SetMessageText(s);
         return(0);
      }
      if (  (rc = oracustw(cust, cust_orders, cust_sales))  ) {
         s.Format("OraCustWrite Error %d",rc);
         GetParentFrame()->SetMessageText(s);
      }

   }

   return(1);
}


void COentryView::PlaceOrFillMode(int mode)
{
   m_OrderMode = mode;
   int SHOW, HIDE, IsPlaceMode;

   if (mode <= PLACE_ORDER) {
      HIDE = SW_HIDE;
      SHOW = SW_SHOW;
      IsPlaceMode = TRUE;
   } else {
      HIDE = SW_SHOW;
      SHOW = SW_HIDE;
      IsPlaceMode = FALSE;
   }

   GetDlgItem(IDC_SENDREPC)->ShowWindow(SHOW); // Show sendreply checkbox
```

```
    if (m_OrderMode <= PLACE_ORDER){
       SetDlgItemText(IDC_ORDERB,"Order");            // Order = Order
       SetDlgItemText(IDC_ORDERBOX,"Place Orders");            // Order = Order
       SetDlgItemText(IDC_BIG_TITLE,"OpenMQ  Place Orders");            // Order =
    } else {
       SetDlgItemText(IDC_ORDERB,"Fill");
       SetDlgItemText(IDC_ORDERBOX,"Fill Orders");
       SetDlgItemText(IDC_BIG_TITLE,"OpenMQ  Fill Orders");
    }

    GetDlgItem(IDC_SHOWDB)->ShowWindow(HIDE);

    GetDlgItem(IDC_CUST)->ShowWindow(SHOW);
    GetDlgItem(IDC_ITEM)->ShowWindow(SHOW);
    GetDlgItem(IDC_QTY)->ShowWindow(SHOW);
    GetDlgItem(IDC_CUSTLAB)->ShowWindow(SHOW);
    GetDlgItem(IDC_ITEMLAB)->ShowWindow(SHOW);
    GetDlgItem(IDC_QTYLAB)->ShowWindow(SHOW);

    GetDlgItem(IDC_FILLTXT)->ShowWindow(HIDE);

//   (  (CEdit *) this->GetDlgItem(IDC_CUST)  )->SetReadOnly(FALSE);
//   (  (CEdit *) this->GetDlgItem(IDC_ITEM)  )->SetReadOnly(FALSE);
//   (  (CEdit *) this->GetDlgItem(IDC_QTY)   )->SetReadOnly(FALSE);

    // Stats
    if (mode <= PLACE_ORDER) {

       SetDlgItemText(IDC_MSGS_LAB,"Sent:");
       SetDlgItemText(IDC_RECIPTS_LAB,"Received:");
       SetDlgItemInt(IDC_MSGS,     m_mes_sent);
       SetDlgItemInt(IDC_RECIPTS, m_rec_rec);
    } else {
       SetDlgItemText(IDC_MSGS_LAB,"Received:");
       SetDlgItemText(IDC_RECIPTS_LAB,"Sent:");
       SetDlgItemInt(IDC_MSGS,     m_mes_rec);
       SetDlgItemInt(IDC_RECIPTS, m_rec_sent);
    }

    if (mode != PLACE_ORDER_NOQ) {
       GetDlgItem(IDC_TRANBOX)->ShowWindow(SW_SHOW);
       if (m_tran_state) {
          GetDlgItem(IDC_COMMITB)->ShowWindow(SW_SHOW);
          GetDlgItem(IDC_ABORTB)->ShowWindow(SW_SHOW);
       } else
          GetDlgItem(IDC_TRANB)->ShowWindow(SW_SHOW);
    } else {
       GetDlgItem(IDC_TRANBOX)->ShowWindow(SW_HIDE);
       GetDlgItem(IDC_TRANB)->ShowWindow(SW_HIDE);
       GetDlgItem(IDC_COMMITB)->ShowWindow(SW_HIDE);
       GetDlgItem(IDC_ABORTB)->ShowWindow(SW_HIDE);
       GetDlgItem(IDC_SENDREPC)->ShowWindow(SW_HIDE);
    }


    if (mode == PLACE_ORDER_NOQ) {
       LoadQList(0); // Load non-Que
    } else {
       LoadQList(1); // Load Que names
    }
```

```
   // DB
   if (mode <= PLACE_ORDER)
      if (g_db_run) g_db_run = 40; // Turn off the DB display if it was left on.


   if (mode == FILL_FROM_ORACLE) {
      if (g_ora_state == 0)
         if(oraconn(g_oracle_con_str))
            MessageBox("Oracle Connect Failed");
         else
            g_ora_state = 1;
   } else {
      if (g_ora_state == 1)
         if(oradisc())
            MessageBox("Oracle DisConnect Failed");
         else
            g_ora_state = 0;
   }

   OnAnyUserAction();
}


// Re load the que list just incase a new que was launched
void COentryView::OnSetfocusQue()
{
   if (m_OrderMode == PLACE_ORDER_NOQ) {
      LoadQList(0); // Load non-Que
   } else {
      LoadQList(1); // Load Que names
   }

}






void COentryView::OnPlacenoqr()
{
   if (!((CButton *)GetDlgItem(IDC_PLACENOQR))->GetCheck())
      return;
   PlaceOrFillMode(PLACE_ORDER_NOQ);
   OnAnyUserAction();
}
void COentryView::OnPlacer()
{
   if (!((CButton *)GetDlgItem(IDC_PLACER))->GetCheck())
      return;
   PlaceOrFillMode(PLACE_ORDER);
   OnAnyUserAction();
}

void COentryView::OnFillr()
{
   if (!((CButton *)GetDlgItem(IDC_FILLR))->GetCheck())
      return;

   PlaceOrFillMode(FILL_ORDER);
   OnAnyUserAction();
}

void COentryView::OnFilldb()
{
   if (!((CButton *)GetDlgItem(IDC_FILLDB))->GetCheck())
      return;

   PlaceOrFillMode(FILL_FROM_ORACLE);
   OnAnyUserAction();
}


void COentryView::OnAnyUserAction()
{
   SetDlgItemText(IDC_RECIPT,"");
   SetDlgItemText(IDC_FILLTXT,"");

   if (m_que == "")
      GetParentFrame()->SetWindowText(m_inst + ": No server is selected" );
   else if (m_OrderMode == PLACE_ORDER)
      GetParentFrame()->SetWindowText(m_inst + ": Place orders into queue " + m_
   else if (m_OrderMode == PLACE_ORDER_NOQ)
      GetParentFrame()->SetWindowText(m_inst + ": Place orders directly to " + m
   else // FILL
      GetParentFrame()->SetWindowText(m_inst + ": Fill orders from queue " +

   m_box_color = m_color;
}

void COentryView::OnSendrepc()
{
   m_sendreply = !m_sendreply; // How can I get this info?

   if ( Q )
      if ( m_sendreply )
         Q->msgh.reply_smbuf = tp.bnum;
      else
         Q->msgh.reply_smbuf = -1;
}

void COentryView::color_the_box(COLORREF color, BOOL forceit){
   if (( m_box_color != color ) || forceit) {
      m_box_color = color;

        // The DC for the color box
      CDC* pCOLORDC = GetDlgItem(IDC_COLORBOX)->GetDC();
      // pCOLORDC->GetRect();

     // Create a PEN
     // CPen *pQPen = new CPen(PS_SOLID, 3,RGB(0,0,255));
     // CPen *pOldPen = pXDC->SelectObject(pQPen);

      // Create the Brush
      CBrush *pQBrush = new CBrush(color);
```

```
        // Select this brush, save the old
        CBrush *pOldBrush = pCOLORDC->SelectObject(pQBrush);

        // Draw the box

    //  pCOLORDC->Rectangle(CRect(1,1,65,65));

        pCOLORDC->Rectangle(m_color_box_rec);

        pCOLORDC->SelectObject(&pOldBrush);  // Reset the brush

        delete(pQBrush);
        ReleaseDC(pCOLORDC);

 //      GetDlgItem(IDC_LOGO_Q)->Invalidate();
//       GetDlgItem(IDC_INST)->Invalidate();
     }
}

void COentryView::OnDraw(CDC* pDC)
{

/*
   pControlDC->SelectStockObject(WHITE_BRUSH);
*/

   // The DC for the Instance name
 /*
   CDC* pINSTDC = GetDlgItem(IDC_INST)->GetDC();

   pINSTDC->SetBkColor(m_color);
   pINSTDC->SetTextColor(m_color);
   pINSTDC->SetBkMode(TRANSPARENT);
   pINSTDC->SelectStockObject(HOLLOW_BRUSH);
   */
        GetDlgItem(IDC_COLORBOX)->GetWindowRect(&m_color_box_rec);
     m_color_box_rec.right  -= m_color_box_rec.left;  //ScreenToClient(&rec);
     m_color_box_rec.bottom -= m_color_box_rec.top;
     m_color_box_rec.left    = m_color_box_rec.top = 0;

   color_the_box(m_box_color,1);

        CFormView::OnDraw(pDC);
}

/*void COentryView::OnResetb()
{
   m_mes_sent = 0;
   m_mes_rec = 0;
   m_rec_sent = 0;
   m_rec_rec = 0;
   SetDlgItemInt(IDC_MES_REC,0);
   SetDlgItemInt(IDC_REC_REC,0);
   SetDlgItemInt(IDC_MES_SENT,0);
   SetDlgItemInt(IDC_REC_SENT,0);

   OnAnyUserAction();
}
*/
```

```
// TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT
// TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT
// TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT

void COentryView::OnTranb()
{
        m_tran_state = 1;
   GetDlgItem(IDC_TRANB)->ShowWindow(SW_HIDE);
   GetDlgItem(IDC_ABORTB)->ShowWindow(SW_SHOW);
   GetDlgItem(IDC_COMMITB)->ShowWindow(SW_SHOW);
   OnAnyUserAction();
}

void COentryView::OnCommit(int action)
{

   // Anyway get rid of the commit/abort buttons
   GetDlgItem(IDC_TRANB)->ShowWindow(SW_SHOW);
   GetDlgItem(IDC_ABORTB)->ShowWindow(SW_HIDE);
   GetDlgItem(IDC_COMMITB)->ShowWindow(SW_HIDE);

   // Do the commit
   if ((Q) && (m_tran_state > 1))
      Qcommit(action);

   m_tran_rec = 0;
   m_tran_sent = 0;
        m_tran_state = 0;
}

void COentryView::OnCommitb()
{
        OnCommit(Q_COMMIT);
}

void COentryView::OnAbortb()
{
        OnCommit(Q_ABORT);
   m_mes_sent -= m_tran_sent;
   m_rec_rec  -= m_tran_rec;
}

void COentryView::OnLButtonDown(UINT nFlags, CPoint point)
{
   CRect rec;
//      WINDOWPLACEMENT GetWindowPlacement  CRect
//      MessageBeep(0);
        GetDlgItem(IDC_COLORBOX)->GetWindowRect(&rec);
        ScreenToClient(&rec);
        if (rec.PtInRect(point)) OnColor();
        CFormView::OnLButtonDown(nFlags, point);
}


void COentryView::OnRButtonDown(UINT nFlags, CPoint point)
{
   ClearDisplay();
        CFormView::OnRButtonDown(nFlags, point);
}


void COentryView::ClearDisplay()
{
   GetParentFrame()->SetMessageText("");
   this->Invalidate();
   OnAnyUserAction();
}



void COentryView::OnShowdb()
{
        GetDlgItem(IDC_SHOWDB)->EnableWindow(FALSE);
   SetTimer(DB_DONE_TIMER,2000,NULL); // When the DB is done re-enable
  // CDbDlg dbd;
  // dbd.DoModal();
        dbd.Create(IDD_DBDLG);
        //GetParentFrame()->SetMessageText("After dbd.Create ");
}

void COentryView::OnOptionsb()
{
        GetDlgItem(IDC_OPTIONSB)->EnableWindow(FALSE);
   SetTimer(OPTIONS_DONE_TIMER,2000,NULL); // When the options are done re-enabl


//   o_dlg.Create(IDD_O_DLG);

   if (o_dlg.m_inst != m_inst) { // Init
      o_dlg.m_inst = m_inst;
//      o_dlg.m_parentptr = this;
            o_dlg.Create(IDD_O_DLG);
   }

   g_new_color = m_color;

   o_dlg.ShowWindow(SW_SHOW);

}
```

```
DROP TABLE omq_stock;
/

CREATE TABLE omq_stock
   (item  VARCHAR2(10)    PRIMARY KEY,
    price NUMBER,
    qty   NUMBER);
/

INSERT INTO omq_stock VALUES ('Bolts',1,1000);
/
INSERT INTO omq_stock VALUES ('Buckets',2,1000);
/
INSERT INTO omq_stock VALUES ('Buttons',3,1000);
/
INSERT INTO omq_stock VALUES ('Belts',4,1000);
/
INSERT INTO omq_stock VALUES ('Bobbins',5,1000);
/
INSERT INTO omq_stock VALUES ('Boats',6,1000);
/

COMMIT WORK;
/


DROP TABLE omq_cust;
/

CREATE TABLE omq_cust
   (customer  VARCHAR2(10)        PRIMARY KEY,
    orders    NUMBER,
    sales         NUMBER);
/

INSERT INTO omq_cust VALUES ('Jacobs',0,0);
/
INSERT INTO omq_cust VALUES ('Jackson',0,0);
/
INSERT INTO omq_cust VALUES ('Jaffe',0,0);
/
INSERT INTO omq_cust VALUES ('Johnson',0,0);
/
INSERT INTO omq_cust VALUES ('Jones',0,0);
/
INSERT INTO omq_cust VALUES ('James',0,0);
/

COMMIT WORK;
/
```

```
/* File name & Package Name */
struct sqlcxp
{
  unsigned short fillen;
           char  filnam[10];
};
static struct sqlcxp sqlfpn =
{
    9,
    "oraomq.pc"
};


static const unsigned long sqlctx = 822081471;


static struct sqlexd {
   unsigned long    sqlvsn;
   unsigned long   arrsiz;
   unsigned long   iters;
   unsigned short   offset;
   unsigned short   selerr;
   unsigned short   sqlety;
  unsigned short   unused;
            short   *cud;
   unsigned char    *sqlest;
            char    *stmt;
   unsigned char  * *sqphsv;
   unsigned long   *sqphsl;
            short  * *sqpind;
   unsigned long   *sqparm;
   unsigned long   * *sqparc;
   unsigned char    *sqhstv[3];
   unsigned long    sqhstl[3];
            short   *sqindv[3];
   unsigned long    sqharm[3];
   unsigned long   *sqharc[3];
} sqlstm = {8,3};
extern sqlcx2(/*_ unsigned long , struct sqlexd *, struct sqlcxp * _*/);
extern sqlcte(/*_ unsigned long , struct sqlexd *, struct sqlcxp * _*/);
extern sqlbuf(/*_ char * _*/);
extern sqlora(/*_ unsigned long *, void * _*/);

static int IAPSUCC = 0;
static int IAPFAIL = 1403;
static int IAPFTL  = 535;
extern    sqliem();

/* cud (compilation unit data) array */
static short sqlcud0[] =
{8,34,
2,0,8,0,0,0,0,0,0,0,0,0,0,0,0,27,88,0,3,3,0,1,0,1,9,0,0,1,10,0,0,1,10,0,0,
36,0,8,0,0,0,0,0,0,0,0,0,0,0,30,111,0,0,0,0,1,0,
58,0,8,0,0,0,0,0,0,0,0,1,20,0,1,151,0,0,0,0,1,0,
80,0,8,0,0,0,0,0,0,0,0,2,19,0,1,152,0,0,0,0,1,0,
102,0,8,0,0,0,0,0,0,0,0,3,76,0,44,157,0,0,0,0,1,0,
124,0,8,0,0,0,0,0,0,0,0,4,45,0,3,161,0,0,0,0,1,0,
146,0,8,0,0,0,0,0,0,0,0,5,47,0,3,162,0,0,0,0,1,0,
168,0,8,0,0,0,0,0,0,0,0,6,47,0,3,163,0,0,0,0,1,0,
190,0,8,0,0,0,0,0,0,0,0,7,45,0,3,164,0,0,0,0,1,0,
212,0,8,0,0,0,0,0,0,0,0,8,47,0,3,165,0,0,0,0,1,0,
234,0,8,0,0,0,0,0,0,0,0,9,45,0,3,166,0,0,0,0,1,0,
256,0,8,0,0,0,0,0,0,0,0,10,82,0,44,168,0,0,0,0,1,0,
278,0,8,0,0,0,0,0,0,0,0,11,41,0,3,172,0,0,0,0,1,0,
300,0,8,0,0,0,0,0,0,0,0,12,42,0,3,173,0,0,0,0,1,0,
322,0,8,0,0,0,0,0,0,0,0,13,40,0,3,174,0,0,0,0,1,0,
344,0,8,0,0,0,0,0,0,0,0,14,42,0,3,175,0,0,0,0,1,0,
366,0,8,0,0,0,0,0,0,0,0,15,40,0,3,176,0,0,0,0,1,0,
388,0,8,0,0,0,0,0,0,0,0,16,40,0,3,177,0,0,0,0,1,0,
410,0,8,0,0,0,0,0,0,0,0,16,0,0,29,179,0,0,0,0,1,0,
432,0,8,0,0,0,0,0,0,0,0,17,58,0,4,211,0,3,1,0,1,0,1,9,0,0,2,3,0,0,2,3,0,0,
466,0,8,0,0,0,0,0,0,0,0,18,43,0,5,259,0,2,2,0,1,0,1,3,0,0,1,9,0,0,
496,0,8,0,0,0,0,0,0,0,0,18,0,0,29,263,0,0,0,0,1,0,
518,0,8,0,0,0,0,0,0,0,0,19,64,0,4,306,0,3,1,0,1,0,1,9,0,0,2,3,0,0,2,3,0,0,
552,0,8,0,0,0,0,0,0,0,0,20,59,0,5,356,0,3,3,0,1,0,1,3,0,0,1,3,0,0,1,9,0,0,
586,0,8,0,0,0,0,0,0,0,0,20,0,0,29,360,0,0,0,0,1,0,
};

/*
**
** Copyright(C)1996 MITSUBISHI ELECTRIC ITA.  ALL RIGHTS RESERVED.
** UNPUBLISHED -- RIGHTS RESERVED UNDER THE COPYRIGHT
** LAWS OF THE UNITED STATES.  USE OF A COPYRIGHT NOTICE
** IS PRECAUTIONARY ONLY AND DOES NOT IMPLY PUBLICATION
** OR DISCLOSURE.
**
** THIS SOFTWARE CONTAINS CONFIDENTIAL INFORMATION AND
** TRADE SECRETS OF MITSUBISHI ELECTRIC ITA.  USE, DISCLOSURE,
** OR REPRODUCTION IS PROHIBITED WITHOUT THE PRIOR
** EXPRESS WRITTEN PERMISSION OF MITSUBISHI ELECTRIC ITA.
**
** OpenMQ Demo
**    Module: oraomq.pc
**    Author: Frederick J. Igo, Jr. 1/15/96
**
*/


#include <stdlib.h>
#include <stdio.h>
#include <ctype.h>
#include <string.h>

# include "sqlproto.h"
# include "ociproto.h"
# define ORACLE
# include "oraomq.h"

/* SQL stmt #1
EXEC SQL INCLUDE SQLCA.H;
*/
/* Copyright (c) 1985,1986 by Oracle Corporation. */

/*
NAME
  SQLCA : SQL Communications Area.
FUNCTION
  Contains no code. Oracle fills in the SQLCA with status info
  during the execution of a SQL stmt.
NOTES
```

```
  If the symbol SQLCA_STORAGE_CLASS is defined, then the SQLCA
  will be defined to have this storage class. For example:

    #define SQLCA_STORAGE_CLASS extern

  will define the SQLCA as an extern.

  If the symbol SQLCA_INIT is defined, then the SQLCA will be
  statically initialized. Although this is not necessary in order
  to use the SQLCA, it is a good pgming practice not to have
  unitialized variables. However, some C compilers/OS's don't
  allow automatic variables to be init'd in this manner. Therefore,
  if you are INCLUDE'ing the SQLCA in a place where it would be
  an automatic AND your C compiler/OS doesn't allow this style
  of initialization, then SQLCA_INIT should be left undefined --
  all others can define SQLCA_INIT if they wish.

  New rules for defining SQLCA_INIT, SQLCA_STORAGE_CLASS, and DLL in OS/2:
  Users should not define SQLCA_STORAGE_CLASS if defining DLL.
  SQLCA_STORAGE_CLASS is primarily used for single-threaded programs
  and for internal development.

MODIFIED
  Okamura    08/15/89 - OS/2: users must define SQLMT for multi-threaded case
  Okamura    06/23/89 - OS/2: modify for multi-threaded case
  Clare      12/06/84 - Ch SQLCA to not be an extern.
  Clare      10/21/85 - Add initialization.
  Bradbury   01/05/86 - Only initialize when SQLCA_INIT set
  Clare      06/12/86 - Add SQLCA_STORAGE_CLASS option.
*/

#ifndef SQLCA
#define SQLCA 1

struct   sqlca
         {
         /* ub1 */ char    sqlcaid[8];
         /* b4  */ long    sqlabc;
         /* b4  */ long    sqlcode;
         struct
           {
           /* ub2 */ unsigned short sqlerrml;
           /* ub1 */ char           sqlerrmc[70];
           } sqlerrm;
         /* ub1 */ char    sqlerrp[8];
         /* b4  */ long    sqlerrd[6];
         /* ub1 */ char    sqlwarn[8];
         /* ub1 */ char    sqlext[8];
         };

#ifdef SQLMT
   extern struct sqlca *sqlcamt();              /* For multi-threaded version */
#  define sqlca (*sqlcamt())
#else /* SQLMT */

#ifdef SQLCA_STORAGE_CLASS
  SQLCA_STORAGE_CLASS struct sqlca sqlca
# ifdef  SQLCA_INIT
         = {
         {'S', 'Q', 'L', 'C', 'A', ' ', ' ', ' '},
        sizeof(struct sqlca),
        0,
        { 0, {0}},
        {'N', 'O', 'T', ' ', 'S', 'E', 'T', ' '},
        {0, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, 0, 0}
        }
# endif /* SQLCA_INIT */
        ;

#else /* SQLCA_STORAGE_CLASS */
   struct sqlca sqlca                       /* For single-threaded version */

#  ifdef  SQLCA_INIT
        = {
        {'S', 'Q', 'L', 'C', 'A', ' ', ' ', ' '},
        sizeof(struct sqlca),
        0,
        { 0, {0}},
        {'N', 'O', 'T', ' ', 'S', 'E', 'T', ' '},
        {0, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, 0, 0}
        }
#  endif /* SQLCA_INIT */
        ;
#endif /* SQLCA_STORAGE_CLASS */

#endif /* SQLMT */

/* end SQLCA */
#endif /* SQLCA */
     /* #include <C:\ORANT\PRO16\C\sqlca.h> */


/*
NAME
  oraomq
FUNCTION
  Openmq Oracle Pro*C subroutines
NOTES

   oraerrrpt();         -- Prints SQL Errors msgs & codes
   oraconn(...);        -- CONNECTS to ORACLE using given oracle string.
   oradisc();           -- DISCONNECTS from ORACLE
   oracreate();         -- Creates omq_stock & omq_cust tables in ORACLE.
   oraread(...);        -- Given ITEM name, reads PRICE & QTY.
   orawrite(...);       -- Given ITEM name & QTY, updates QTY IN db.
   oracustr(...);       -- Given CUSTOMER name, reads #ORDERS & $SALES.
   oracustw(...);       -- Given CUSTOMER name, #ORDERS & $SALES, updates db.

   C:\ORANT\PRO16\C> nmake -f oraomq.mak

*/


/* ------------------------------------------------------------------------
   ORAERRRPT prints the ORACLE error msg and number.
```

```
------------------------------------------------------------------------ */

oraerrrpt()
   {
   printf("%.70s (%d)\n", sqlca.sqlerrm.sqlerrmc, -sqlca.sqlcode);
   return(0);
   }




/* ----------------------------------------------------------------------
   ORACONN connects to ORACLE as user DEMO.
   Oracle String is provided by caller as: "SCOTT/TIGER@T:GRAMPA:ORCL".
   returns 0 on success and 1 on SQL error.
------------------------------------------------------------------------ */

oraconn(orastring)
          char orastring[80];
   {
/* SQL stmt #2
   EXEC SQL BEGIN DECLARE SECTION;
*/
struct {
  unsigned short len;
  unsigned char arr[80];
  } oracleid;
/*
       VARCHAR oracleid[80];
              /o username/password@dbstring  o/
   EXEC SQL END DECLARE SECTION;
*/

/* SQL stmt #4
   EXEC SQL WHENEVER SQLERROR GOTO errexit;
*/

   strcpy((char *)oracleid.arr,orastring);
   oracleid.len = strlen((char *)oracleid.arr);

/* SQL stmt #5
   EXEC SQL CONNECT :oracleid;
*/
{
   struct sqlexd sqlstm={8,3};
   sqlstm.iters = (unsigned int  )10;
   sqlstm.offset = (unsigned int  )2;
   sqlstm.cud = sqlcud0;
   sqlstm.sqlest = (unsigned char  *)&sqlca;
   sqlstm.sqlety = (unsigned short)0;
   sqlstm.sqhstv[0] = (unsigned char  *)&oracleid;
   sqlstm.sqhstl[0] = (unsigned int  )82;
   sqlstm.sqindv[0] = (         short *)0;
   sqlstm.sqharm[0] = (unsigned int  )0;
   sqlstm.sqphsv = sqlstm.sqhstv;
   sqlstm.sqphsl = sqlstm.sqhstl;
   sqlstm.sqpind = sqlstm.sqindv;
   sqlstm.sqparm = sqlstm.sqharm;
   sqlstm.sqparc = sqlstm.sqharc;
   sqlcex(sqlctx, &sqlstm, &sqlfpn);
   if (sqlca.sqlcode < 0) goto errexit;
}


   return(0);

   /* Here if SQL Error */
   errexit:
/* SQL stmt #6
     EXEC SQL WHENEVER SQLERROR CONTINUE;
*/
     return(1);
   }




/* ---------------------------------------------------------------------------
   ORADISC disconnects from ORACLE.
   returns 0 on success and 1 on SQL error.
--------------------------------------------------------------------------- */

oradisc()
   {

/* SQL stmt #7
   EXEC SQL WHENEVER SQLERROR GOTO errexit;
*/

/* SQL stmt #8
   EXEC SQL COMMIT RELEASE;
*/
{
   struct sqlexd sqlstm={8,0};
   sqlstm.iters = (unsigned int  )1;
   sqlstm.offset = (unsigned int  )36;
   sqlstm.cud = sqlcud0;
   sqlstm.sqlest = (unsigned char  *)&sqlca;
   sqlstm.sqlety = (unsigned short)0;
   sqlcex(sqlctx, &sqlstm, &sqlfpn);
   if (sqlca.sqlcode < 0) goto errexit;
}

   return(0);

   /* Here if SQL Error */
   errexit:
/* SQL stmt #9
     EXEC SQL WHENEVER SQLERROR CONTINUE;
*/
     return(1);
   }




/* ---------------------------------------------------------------------------
   ORACREATE creates omq_stock & omq_cust tables in ORACLE as follows:
```

```
   omq_stock:   ITEM           PRICE   QTY
                               Bolts           $1            10000
                               Buckets         $2            10000
                               Buttons         $3            10000
                               Belts           $4            10000
                               Bobbins         $5            10000
                               Boats           $6            10000

   omq_cust:    CUSTOMER       ORDERS  SALES
                               Jacobs          0             0
                               Jackson         0             0
                               Jaffe           0             0
                               Johnson         0             0
                               Jones           0             0
                               James           0      0

   returns 0 on success and 1 on SQL error.
------------------------------------------------------------------------ */

oracreate()
   {

   /* Expect an error if table already dropped */
/* SQL stmt #10
   EXEC SQL WHENEVER SQLERROR GOTO ignore;
   EXEC SQL DROP TABLE omq_stock;
*/
{
   struct sqlexd sqlstm={8,0};
   sqlstm.stmt = "DROP TABLE OMQ_STOCK";
   sqlstm.iters = (unsigned int  )1;
   sqlstm.offset = (unsigned int  )58;
   sqlstm.cud = sqlcud0;
   sqlstm.sqlest = (unsigned char  *)&sqlca;
   sqlstm.sqlety = (unsigned short)0;
   sqlcex(sqlctx, &sqlstm, &sqlfpn);
   if (sqlca.sqlcode < 0) goto ignore;
}
/* SQL stmt #12
   EXEC SQL DROP TABLE omq_cust;
*/
{
   struct sqlexd sqlstm={8,0};
   sqlstm.stmt = "DROP TABLE OMQ_CUST";
   sqlstm.iters = (unsigned int  )1;
   sqlstm.offset = (unsigned int  )80;
   sqlstm.cud = sqlcud0;
   sqlstm.sqlest = (unsigned char  *)&sqlca;
   sqlstm.sqlety = (unsigned short)0;
   sqlcex(sqlctx, &sqlstm, &sqlfpn);
   if (sqlca.sqlcode < 0) goto ignore;
}

   ignore:
/* SQL stmt #13
     EXEC SQL WHENEVER SQLERROR GOTO errexit;
*/


/* SQL stmt #14
```

```
     EXEC SQL CREATE TABLE omq_stock
               (item   VARCHAR2(10)     PRIMARY KEY,
                price  NUMBER,
                qty    NUMBER);
*/
{
     struct sqlexd sqlstm={8,0};
     sqlstm.stmt = "CREATE TABLE OMQ_STOCK(ITEM VARCHAR2(10)PRIMARY KEY,PRIC\
E NUMBER,QTY NUMBER)";
     sqlstm.iters = (unsigned int  )1;
     sqlstm.offset = (unsigned int  )102;
     sqlstm.cud = sqlcud0;
     sqlstm.sqlest = (unsigned char  *)&sqlca;
     sqlstm.sqlety = (unsigned short)0;
     sqlcex(sqlctx, &sqlstm, &sqlfpn);
     if (sqlca.sqlcode < 0) goto errexit;
}

/* SQL stmt #15
     EXEC SQL INSERT INTO omq_stock VALUES('Bolts',1,10000);
*/
{
     struct sqlexd sqlstm={8,0};
     sqlstm.stmt = "INSERT INTO OMQ_STOCK VALUES('Bolts',1,10000)";
     sqlstm.iters = (unsigned int  )1;
     sqlstm.offset = (unsigned int  )124;
     sqlstm.cud = sqlcud0;
     sqlstm.sqlest = (unsigned char  *)&sqlca;
     sqlstm.sqlety = (unsigned short)0;
     sqlcex(sqlctx, &sqlstm, &sqlfpn);
     if (sqlca.sqlcode < 0) goto errexit;
}
/* SQL stmt #16
          EXEC SQL INSERT INTO omq_stock VALUES('Buckets',2,10000);
*/
{
          struct sqlexd sqlstm={8,0};
          sqlstm.stmt = "INSERT INTO OMQ_STOCK VALUES('Buckets',2,10000)";
          sqlstm.iters = (unsigned int  )1;
          sqlstm.offset = (unsigned int  )146;
          sqlstm.cud = sqlcud0;
          sqlstm.sqlest = (unsigned char  *)&sqlca;
          sqlstm.sqlety = (unsigned short)0;
          sqlcex(sqlctx, &sqlstm, &sqlfpn);
          if (sqlca.sqlcode < 0) goto errexit;
}
/* SQL stmt #17
          EXEC SQL INSERT INTO omq_stock VALUES('Buttons',3,10000);
*/
{
          struct sqlexd sqlstm={8,0};
          sqlstm.stmt = "INSERT INTO OMQ_STOCK VALUES('Buttons',3,10000)";
          sqlstm.iters = (unsigned int  )1;
          sqlstm.offset = (unsigned int  )168;
          sqlstm.cud = sqlcud0;
          sqlstm.sqlest = (unsigned char  *)&sqlca;
          sqlstm.sqlety = (unsigned short)0;
          sqlcex(sqlctx, &sqlstm, &sqlfpn);
          if (sqlca.sqlcode < 0) goto errexit;
}
```

```
/* SQL stmt #18
        EXEC SQL INSERT INTO omq_stock VALUES('Belts',4,10000);
*/
{
        struct sqlexd sqlstm={8,0};
        sqlstm.stmt = "INSERT INTO OMQ_STOCK VALUES('Belts',4,10000)";
        sqlstm.iters = (unsigned int  )1;
        sqlstm.offset = (unsigned int  )190;
        sqlstm.cud = sqlcud0;
        sqlstm.sqlest = (unsigned char  *)&sqlca;
        sqlstm.sqlety = (unsigned short)0;
        sqlcex(sqlctx, &sqlstm, &sqlfpn);
        if (sqlca.sqlcode < 0) goto errexit;
}
/* SQL stmt #19
        EXEC SQL INSERT INTO omq_stock VALUES('Bobbins',5,10000);
*/
{
        struct sqlexd sqlstm={8,0};
        sqlstm.stmt = "INSERT INTO OMQ_STOCK VALUES('Bobbins',5,10000)";
        sqlstm.iters = (unsigned int  )1;
        sqlstm.offset = (unsigned int  )212;
        sqlstm.cud = sqlcud0;
        sqlstm.sqlest = (unsigned char  *)&sqlca;
        sqlstm.sqlety = (unsigned short)0;
        sqlcex(sqlctx, &sqlstm, &sqlfpn);
        if (sqlca.sqlcode < 0) goto errexit;
}
/* SQL stmt #20
        EXEC SQL INSERT INTO omq_stock VALUES('Boats',6,10000);
*/
{
        struct sqlexd sqlstm={8,0};
        sqlstm.stmt = "INSERT INTO OMQ_STOCK VALUES('Boats',6,10000)";
        sqlstm.iters = (unsigned int  )1;
        sqlstm.offset = (unsigned int  )234;
        sqlstm.cud = sqlcud0;
        sqlstm.sqlest = (unsigned char  *)&sqlca;
        sqlstm.sqlety = (unsigned short)0;
        sqlcex(sqlctx, &sqlstm, &sqlfpn);
        if (sqlca.sqlcode < 0) goto errexit;
}

/* SQL stmt #21
        EXEC SQL CREATE TABLE omq_cust
               (customer        VARCHAR2(10)    PRIMARY KEY,
                orders          NUMBER,
                sales           NUMBER);
*/
{
        struct sqlexd sqlstm={8,0};
        sqlstm.stmt = "CREATE TABLE OMQ_CUST(CUSTOMER VARCHAR2(10)PRIMARY K\
EY,ORDERS NUMBER,SALES NUMBER)";
        sqlstm.iters = (unsigned int  )1;
        sqlstm.offset = (unsigned int  )256;
        sqlstm.cud = sqlcud0;
        sqlstm.sqlest = (unsigned char  *)&sqlca;
        sqlstm.sqlety = (unsigned short)0;
        sqlcex(sqlctx, &sqlstm, &sqlfpn);
        if (sqlca.sqlcode < 0) goto errexit;
}
/* SQL stmt #22
        EXEC SQL INSERT INTO omq_cust VALUES('Jacobs',0,0);
*/
{
        struct sqlexd sqlstm={8,0};
        sqlstm.stmt = "INSERT INTO OMQ_CUST VALUES('Jacobs',0,0)";
        sqlstm.iters = (unsigned int  )1;
        sqlstm.offset = (unsigned int  )278;
        sqlstm.cud = sqlcud0;
        sqlstm.sqlest = (unsigned char  *)&sqlca;
        sqlstm.sqlety = (unsigned short)0;
        sqlcex(sqlctx, &sqlstm, &sqlfpn);
        if (sqlca.sqlcode < 0) goto errexit;
}
/* SQL stmt #23
        EXEC SQL INSERT INTO omq_cust VALUES('Jackson',0,0);
*/
{
        struct sqlexd sqlstm={8,0};
        sqlstm.stmt = "INSERT INTO OMQ_CUST VALUES('Jackson',0,0)";
        sqlstm.iters = (unsigned int  )1;
        sqlstm.offset = (unsigned int  )300;
        sqlstm.cud = sqlcud0;
        sqlstm.sqlest = (unsigned char  *)&sqlca;
        sqlstm.sqlety = (unsigned short)0;
        sqlcex(sqlctx, &sqlstm, &sqlfpn);
        if (sqlca.sqlcode < 0) goto errexit;
}
/* SQL stmt #24
        EXEC SQL INSERT INTO omq_cust VALUES('Jaffe',0,0);
*/
{
        struct sqlexd sqlstm={8,0};
        sqlstm.stmt = "INSERT INTO OMQ_CUST VALUES('Jaffe',0,0)";
        sqlstm.iters = (unsigned int  )1;
        sqlstm.offset = (unsigned int  )322;
        sqlstm.cud = sqlcud0;
        sqlstm.sqlest = (unsigned char  *)&sqlca;
        sqlstm.sqlety = (unsigned short)0;
        sqlcex(sqlctx, &sqlstm, &sqlfpn);
        if (sqlca.sqlcode < 0) goto errexit;
}
/* SQL stmt #25
        EXEC SQL INSERT INTO omq_cust VALUES('Johnson',0,0);
*/
{
        struct sqlexd sqlstm={8,0};
        sqlstm.stmt = "INSERT INTO OMQ_CUST VALUES('Johnson',0,0)";
        sqlstm.iters = (unsigned int  )1;
        sqlstm.offset = (unsigned int  )344;
        sqlstm.cud = sqlcud0;
        sqlstm.sqlest = (unsigned char  *)&sqlca;
        sqlstm.sqlety = (unsigned short)0;
        sqlcex(sqlctx, &sqlstm, &sqlfpn);
        if (sqlca.sqlcode < 0) goto errexit;
}
/* SQL stmt #26
        EXEC SQL INSERT INTO omq_cust VALUES('Jones',0,0);
*/
```

```
{
        struct sqlexd sqlstm={8,0};
        sqlstm.stmt = "INSERT INTO OMQ_CUST VALUES('Jones',0,0)";
        sqlstm.iters = (unsigned int  )1;
        sqlstm.offset = (unsigned int  )366;
        sqlstm.cud = sqlcud0;
        sqlstm.sqlest = (unsigned char  *)&sqlca;
        sqlstm.sqlety = (unsigned short)0;
        sqlcex(sqlctx, &sqlstm, &sqlfpn);
        if (sqlca.sqlcode < 0) goto errexit;
}
/* SQL stmt #27
        EXEC SQL INSERT INTO omq_cust VALUES('James',0,0);
*/
{
        struct sqlexd sqlstm={8,0};
        sqlstm.stmt = "INSERT INTO OMQ_CUST VALUES('James',0,0)";
        sqlstm.iters = (unsigned int  )1;
        sqlstm.offset = (unsigned int  )388;
        sqlstm.cud = sqlcud0;
        sqlstm.sqlest = (unsigned char  *)&sqlca;
        sqlstm.sqlety = (unsigned short)0;
        sqlcex(sqlctx, &sqlstm, &sqlfpn);
        if (sqlca.sqlcode < 0) goto errexit;
}

/* SQL stmt #28
        EXEC SQL COMMIT WORK;
*/
{
        struct sqlexd sqlstm={8,0};
        sqlstm.iters = (unsigned int  )1;
        sqlstm.offset = (unsigned int  )410;
        sqlstm.cud = sqlcud0;
        sqlstm.sqlest = (unsigned char  *)&sqlca;
        sqlstm.sqlety = (unsigned short)0;
        sqlcex(sqlctx, &sqlstm, &sqlfpn);
        if (sqlca.sqlcode < 0) goto errexit;
}

    return(0);

  /* Here if SQL Error */
  errexit:
/* SQL stmt #29
    EXEC SQL WHENEVER SQLERROR CONTINUE;
*/
    return(1);
  }


/* ---------------------------------------------------------------------
  ORAREAD reads QTY & PRICE from omq_stock table given ITEM.
  returns 0 on success, 1 on SQL error and 2 on noitem.
------------------------------------------------------------------- */

int oraread(itemname, itemprice, itemqty)
         char itemname[10];
         int *itemprice, *itemqty;
  {
```

```
/* SQL stmt #30
   EXEC SQL BEGIN DECLARE SECTION;
*/

struct {
  unsigned short len;
  unsigned char arr[10];
  } item;
/*
       VARCHAR item[10];
                  /o item name      o/
*/
       int     price;                    /* item price    */
       int     qty;                      /* item quantity */
/* SQL stmt #31
   EXEC SQL END DECLARE SECTION;
*/


/* SQL stmt #32
   EXEC SQL WHENEVER NOT FOUND GOTO noitem;
   EXEC SQL WHENEVER SQLERROR GOTO errexit;
*/

   strcpy((char *)item.arr,itemname);
   item.len = strlen(itemname);

/* SQL stmt #34
   EXEC SQL SELECT PRICE, QTY
            INTO :price, :qty
            FROM omq_stock
            WHERE ITEM = :item;
*/
{
   struct sqlexd sqlstm={8,3};
   sqlstm.stmt = "SELECT PRICE,QTY INTO:b1,:b2 FROM OMQ_STOCK WHERE ITEM=:b3";
   sqlstm.iters = (unsigned int  )1;
   sqlstm.offset = (unsigned int  )432;
   sqlstm.selerr = (unsigned short)1;
   sqlstm.cud = sqlcud0;
   sqlstm.sqlest = (unsigned char  *)&sqlca;
   sqlstm.sqlety = (unsigned short)0;
   sqlstm.sqhstv[0] = (unsigned char  *)&item;
   sqlstm.sqhstl[0] = (unsigned int  )12;
   sqlstm.sqindv[0] = (         short *)0;
   sqlstm.sqharm[0] = (unsigned int  )0;
   sqlstm.sqhstv[1] = (unsigned char  *)&price;
   sqlstm.sqhstl[1] = (unsigned int  )4;
   sqlstm.sqindv[1] = (         short *)0;
   sqlstm.sqharm[1] = (unsigned int  )0;
   sqlstm.sqhstv[2] = (unsigned char  *)&qty;
   sqlstm.sqhstl[2] = (unsigned int  )4;
   sqlstm.sqindv[2] = (         short *)0;
   sqlstm.sqharm[2] = (unsigned int  )0;
   sqlstm.sqphsv = sqlstm.sqhstv;
   sqlstm.sqphsl = sqlstm.sqhstl;
   sqlstm.sqpind = sqlstm.sqindv;
   sqlstm.sqparm = sqlstm.sqharm;
   sqlstm.sqparc = sqlstm.sqharc;
   sqlcex(sqlctx, &sqlstm, &sqlfpn);
   if (sqlca.sqlcode == 1403) goto noitem;
   if (sqlca.sqlcode < 0) goto errexit;
}

/* SQL stmt #35
   EXEC SQL WHENEVER NOT FOUND STOP;
   EXEC SQL WHENEVER SQLERROR CONTINUE;
*/

   *itemprice = price;
   *itemqty = qty;

   return(0);

   /* Here if item NOT found in dbs */
   noitem:
/* SQL stmt #37
     EXEC SQL WHENEVER NOT FOUND STOP;
     EXEC SQL WHENEVER SQLERROR CONTINUE;
*/
     return(2);

   /* Here if SQL Error */
   errexit:
/* SQL stmt #39
     EXEC SQL WHENEVER NOT FOUND STOP;
     EXEC SQL WHENEVER SQLERROR CONTINUE;
*/
     return(1);
   }


/* ----------------------------------------------------------------------------
   ORAWRITE updates QTY for a given ITEM in the omq_stock table.
   returns 0 on success, 1 on SQL error and 2 on noitem.
---------------------------------------------------------------------------- */

int orawrite(itemname, itemqty)
          char itemname[10];
          int itemqty;
   {
/* SQL stmt #41
   EXEC SQL BEGIN DECLARE SECTION;
*/

struct {
  unsigned short len;
  unsigned char arr[10];
  } item;
/*
       VARCHAR item[10];
                 /o item name       o/
*/
       int     qty;                       /* item quantity  */
/* SQL stmt #42
   EXEC SQL END DECLARE SECTION;
*/


/* SQL stmt #43
```

```
   EXEC SQL WHENEVER NOT FOUND GOTO noitem;
   EXEC SQL WHENEVER SQLERROR GOTO errexit;
*/

   strcpy((char *)item.arr,itemname);
   item.len = strlen(itemname);
   qty = itemqty;

/* SQL stmt #45
   EXEC SQL UPDATE omq_stock
             SET QTY = :qty
             WHERE ITEM = :item;
*/
{
   struct sqlexd sqlstm={8,2};
   sqlstm.stmt = "UPDATE OMQ_STOCK SET QTY=:b1 WHERE ITEM=:b2";
   sqlstm.iters = (unsigned int  )1;
   sqlstm.offset = (unsigned int  )466;
   sqlstm.cud = sqlcud0;
   sqlstm.sqlest = (unsigned char  *)&sqlca;
   sqlstm.sqlety = (unsigned short)0;
   sqlstm.sqhstv[0] = (unsigned char  *)&qty;
   sqlstm.sqhstl[0] = (unsigned int  )4;
   sqlstm.sqindv[0] = (         short *)0;
   sqlstm.sqharm[0] = (unsigned int  )0;
   sqlstm.sqhstv[1] = (unsigned char  *)&item;
   sqlstm.sqhstl[1] = (unsigned int  )12;
   sqlstm.sqindv[1] = (         short *)0;
   sqlstm.sqharm[1] = (unsigned int  )0;
   sqlstm.sqphsv = sqlstm.sqhstv;
   sqlstm.sqphsl = sqlstm.sqhstl;
   sqlstm.sqpind = sqlstm.sqindv;
   sqlstm.sqparm = sqlstm.sqharm;
   sqlstm.sqparc = sqlstm.sqharc;
   sqlcex(sqlctx, &sqlstm, &sqlfpn);
   if (sqlca.sqlcode == 1403) goto noitem;
   if (sqlca.sqlcode < 0) goto errexit;
}

/* SQL stmt #46
   EXEC SQL COMMIT WORK;
*/
{
   struct sqlexd sqlstm={8,0};
   sqlstm.iters = (unsigned int  )1;
   sqlstm.offset = (unsigned int  )496;
   sqlstm.cud = sqlcud0;
   sqlstm.sqlest = (unsigned char  *)&sqlca;
   sqlstm.sqlety = (unsigned short)0;
   sqlcex(sqlctx, &sqlstm, &sqlfpn);
   if (sqlca.sqlcode < 0) goto errexit;
}

/* SQL stmt #47
   EXEC SQL WHENEVER NOT FOUND STOP;
   EXEC SQL WHENEVER SQLERROR CONTINUE;
*/

   return(0);
   /* Here if item NOT found in dbs */
   noitem:
/* SQL stmt #49
     EXEC SQL WHENEVER NOT FOUND STOP;
     EXEC SQL WHENEVER SQLERROR CONTINUE;
*/
     return(2);

   /* Here if SQL Error */
   errexit:
/* SQL stmt #51
     EXEC SQL WHENEVER NOT FOUND STOP;
     EXEC SQL WHENEVER SQLERROR CONTINUE;
*/
     return(1);
   }



/* ------------------------------------------------------------------------------
   ORACUSTR reads ORDERS & SALES from omq_cust table given customer name.
   returns 0 on success, 1 on SQL error and 2 on nocustomer.
--------------------------------------------------------------------------- */

int oracustr(custname, custorders, custsales)
       char custname[10];
       int *custorders, *custsales;
   {
/* SQL stmt #53
   EXEC SQL BEGIN DECLARE SECTION;
*/

struct {
  unsigned short len;
  unsigned char arr[10];
  } customer;
/*
      VARCHAR customer[10];
                 /o cust name      o/
*/
      int     orders;                       /* cust #orders   */
      int     sales;                        /* cust $sales    */
/* SQL stmt #54
   EXEC SQL END DECLARE SECTION;
*/


/* SQL stmt #55
   EXEC SQL WHENEVER NOT FOUND GOTO noitem;
   EXEC SQL WHENEVER SQLERROR GOTO errexit;
*/

   strcpy((char *)customer.arr,custname);
   customer.len = strlen(custname);

/* SQL stmt #57
   EXEC SQL SELECT ORDERS, SALES
              INTO :orders, :sales
              FROM omq_cust
              WHERE CUSTOMER = :customer;
```

```
*/
{
   struct sqlexd sqlstm={8,3};
   sqlstm.stmt = "SELECT ORDERS,SALES INTO:b1,:b2 FROM OMQ_CUST WHERE CUSTOM\
ER=:b3";
   sqlstm.iters = (unsigned int  )1;
   sqlstm.offset = (unsigned int  )518;
   sqlstm.selerr = (unsigned short)1;
   sqlstm.cud = sqlcud0;
   sqlstm.sqlest = (unsigned char  *)&sqlca;
   sqlstm.sqlety = (unsigned short)0;
   sqlstm.sqhstv[0] = (unsigned char  *)&customer;
   sqlstm.sqhstl[0] = (unsigned int  )12;
   sqlstm.sqindv[0] = (         short *)0;
   sqlstm.sqharm[0] = (unsigned int  )0;
   sqlstm.sqhstv[1] = (unsigned char  *)&orders;
   sqlstm.sqhstl[1] = (unsigned int  )4;
   sqlstm.sqindv[1] = (         short *)0;
   sqlstm.sqharm[1] = (unsigned int  )0;
   sqlstm.sqhstv[2] = (unsigned char  *)&sales;
   sqlstm.sqhstl[2] = (unsigned int  )4;
   sqlstm.sqindv[2] = (         short *)0;
   sqlstm.sqharm[2] = (unsigned int  )0;
   sqlstm.sqphsv = sqlstm.sqhstv;
   sqlstm.sqphsl = sqlstm.sqhstl;
   sqlstm.sqpind = sqlstm.sqindv;
   sqlstm.sqparm = sqlstm.sqharm;
   sqlstm.sqparc = sqlstm.sqharc;
   sqlcex(sqlctx, &sqlstm, &sqlfpn);
   if (sqlca.sqlcode == 1403) goto noitem;
   if (sqlca.sqlcode < 0) goto errexit;
}

/* SQL stmt #58
   EXEC SQL WHENEVER NOT FOUND STOP;
   EXEC SQL WHENEVER SQLERROR CONTINUE;
*/

   *custorders = orders;
   *custsales = sales;

   return(0);

   /* Here if item NOT found in dbs */
   noitem:
/* SQL stmt #60
     EXEC SQL WHENEVER NOT FOUND STOP;
     EXEC SQL WHENEVER SQLERROR CONTINUE;
*/
     return(2);

   /* Here if SQL Error */
   errexit:
/* SQL stmt #62
     EXEC SQL WHENEVER NOT FOUND STOP;
     EXEC SQL WHENEVER SQLERROR CONTINUE;
*/
     return(1);
   }
```

```
/* ---------------------------------------------------------------------------
   ORACUSTW updates ORDERS & SALES for a given CUSTOMER in the DB.
   returns 0 on success, 1 on SQL error and 2 on nocustomer.
-------------------------------------------------------------------------- */

int oracustw(custname, custorders, custsales)
          char custname[10];
          int custorders, custsales;
   {
/* SQL stmt #64
   EXEC SQL BEGIN DECLARE SECTION;
*/

struct {
  unsigned short len;
  unsigned char arr[10];
  } customer;
/*
       VARCHAR customer[10];
                  /o cust name       o/
*/
       int      orders;                        /* cust #orders    */
           int      sales;                         /* cust $sales      */
/* SQL stmt #65
   EXEC SQL END DECLARE SECTION;
*/


/* SQL stmt #66
   EXEC SQL WHENEVER NOT FOUND GOTO noitem;
   EXEC SQL WHENEVER SQLERROR GOTO errexit;
*/

   strcpy((char *)customer.arr,custname);
   customer.len = strlen(custname);
   orders = custorders;
   sales = custsales;

/* SQL stmt #68
   EXEC SQL UPDATE omq_cust
              SET ORDERS = :orders,     SALES = :sales
              WHERE CUSTOMER = :customer;
*/
{
   struct sqlexd sqlstm={8,3};
   sqlstm.stmt = "UPDATE OMQ_CUST SET ORDERS=:b1,SALES=:b2 WHERE CUSTOMER=:b\
3";
   sqlstm.iters = (unsigned int  )1;
   sqlstm.offset = (unsigned int  )552;
   sqlstm.cud = sqlcud0;
   sqlstm.sqlest = (unsigned char  *)&sqlca;
   sqlstm.sqlety = (unsigned short)0;
   sqlstm.sqhstv[0] = (unsigned char  *)&orders;
   sqlstm.sqhstl[0] = (unsigned int  )4;
   sqlstm.sqindv[0] = (         short *)0;
   sqlstm.sqharm[0] = (unsigned int  )0;
   sqlstm.sqhstv[1] = (unsigned char  *)&sales;
   sqlstm.sqhstl[1] = (unsigned int  )4;
   sqlstm.sqindv[1] = (         short *)0;
   sqlstm.sqharm[1] = (unsigned int  )0;
   sqlstm.sqhstv[2] = (unsigned char  *)&customer;
   sqlstm.sqhstl[2] = (unsigned int  )12;
   sqlstm.sqindv[2] = (         short *)0;
   sqlstm.sqharm[2] = (unsigned int  )0;
   sqlstm.sqphsv = sqlstm.sqhstv;
   sqlstm.sqphsl = sqlstm.sqhstl;
   sqlstm.sqpind = sqlstm.sqindv;
   sqlstm.sqparm = sqlstm.sqharm;
   sqlstm.sqparc = sqlstm.sqharc;
   sqlcex(sqlctx, &sqlstm, &sqlfpn);
   if (sqlca.sqlcode == 1403) goto noitem;
   if (sqlca.sqlcode < 0) goto errexit;
}

/* SQL stmt #69
   EXEC SQL COMMIT WORK;
*/
{
   struct sqlexd sqlstm={8,0};
   sqlstm.iters = (unsigned int  )1;
   sqlstm.offset = (unsigned int  )586;
   sqlstm.cud = sqlcud0;
   sqlstm.sqlest = (unsigned char  *)&sqlca;
   sqlstm.sqlety = (unsigned short)0;
   sqlcex(sqlctx, &sqlstm, &sqlfpn);
   if (sqlca.sqlcode < 0) goto errexit;
}

/* SQL stmt #70
   EXEC SQL WHENEVER NOT FOUND STOP;
   EXEC SQL WHENEVER SQLERROR CONTINUE;
*/

   return(0);

   /* Here if item NOT found in dbs */
   noitem:
/* SQL stmt #72
     EXEC SQL WHENEVER NOT FOUND STOP;
     EXEC SQL WHENEVER SQLERROR CONTINUE;
*/
     return(2);

   /* Here if SQL Error */
   errexit:
/* SQL stmt #74
     EXEC SQL WHENEVER NOT FOUND STOP;
     EXEC SQL WHENEVER SQLERROR CONTINUE;
*/
     return(1);
   }
```

```
/*
**
** Copyright(C)1996 MITSUBISHI ELECTRIC ITA.  ALL RIGHTS RESERVED.
** UNPUBLISHED -- RIGHTS RESERVED UNDER THE COPYRIGHT
** LAWS OF THE UNITED STATES.  USE OF A COPYRIGHT NOTICE
** IS PRECAUTIONARY ONLY AND DOES NOT IMPLY PUBLICATION
** OR DISCLOSURE.
**
** THIS SOFTWARE CONTAINS CONFIDENTIAL INFORMATION AND
** TRADE SECRETS OF MITSUBISHI ELECTRIC ITA.  USE, DISCLOSURE,
** OR REPRODUCTION IS PROHIBITED WITHOUT THE PRIOR
** EXPRESS WRITTEN PERMISSION OF MITSUBISHI ELECTRIC ITA.
**
** OpenMQ Demo
**   Module: oraomq.h
**   Author: Derek Schwenke 1/11/96
**
*/


#ifdef CPP
extern "C"
{
#endif


#ifdef ORACLE
   int orawrite( char itemname[10],int itemqty);
   int oraread(  char itemname[10],int *itemprice, int *itemqty);
   int oracustw( char custname[10],int custorders, int custsales);
   int oracustr( char custname[10],int *custorders, int *custsales);
   int oradisc();
   int oraconn(  char orastring[80]);
   int oracreate();
   int oraerrrpt();
#else
   int orawrite( char itemname[10],int itemqty)
   int oraread(  char itemname[10],int *itemprice, int *itemqty)
   int oracustw( char custname[10],int custorders, int custsales)
   int oracustr( char custname[10],int *custorders, int *custsales)
   int oradisc()
   int oraconn(  char orastring[80])
   int oracreate()
   int oraerrrpt()
#endif


#ifdef CPP
}
#endif
```

```
/*
**
** Copyright(C)1996 MITSUBISHI ELECTRIC ITA.  ALL RIGHTS RESERVED.
** UNPUBLISHED -- RIGHTS RESERVED UNDER THE COPYRIGHT
** LAWS OF THE UNITED STATES.  USE OF A COPYRIGHT NOTICE
** IS PRECAUTIONARY ONLY AND DOES NOT IMPLY PUBLICATION
** OR DISCLOSURE.
**
** THIS SOFTWARE CONTAINS CONFIDENTIAL INFORMATION AND
** TRADE SECRETS OF MITSUBISHI ELECTRIC ITA.  USE, DISCLOSURE,
** OR REPRODUCTION IS PROHIBITED WITHOUT THE PRIOR
** EXPRESS WRITTEN PERMISSION OF MITSUBISHI ELECTRIC ITA.
**
** OpenMQ Demo
**   Module: oraomq.pc
**   Author: Frederick J. Igo, Jr. 1/15/96
**
*/


#include <stdlib.h>
#include <stdio.h>
#include <ctype.h>
#include <string.h>

# include "sqlproto.h"
# include "ociproto.h"
# define ORACLE
# include "oraomq.h"

EXEC SQL INCLUDE SQLCA.H;      /* #include <C:\ORANT\PRO16\C\sqlca.h> */


/*
NAME
  oraomq
FUNCTION
  Openmq Oracle Pro*C subroutines
NOTES

   oraerrrpt();          -- Prints SQL Errors msgs & codes
   oraconn(...);         -- CONNECTS to ORACLE using given oracle string.
   oradisc();            -- DISCONNECTS from ORACLE
   oracreate();          -- Creates omq_stock & omq_cust tables in ORACLE.
   oraread(...);         -- Given ITEM name, reads PRICE & QTY.
   orawrite(...);        -- Given ITEM name & QTY, updates QTY IN db.
   oracustr(...);        -- Given CUSTOMER name, reads #ORDERS & $SALES.
   oracustw(...);        -- Given CUSTOMER name, #ORDERS & $SALES, updates db.

   C:\ORANT\PRO16\C> nmake -f oraomq.mak

*/


/* ------------------------------------------------------------------------
   ORAERRRPT prints the ORACLE error msg and number.
--------------------------------------------------------------------- */

oraerrrpt()
```

```
   {
   printf("%.70s (%d)\n", sqlca.sqlerrm.sqlerrmc, -sqlca.sqlcode);
   return(0);
   }



/* --------------------------------------------------------------------------
   ORACONN connects to ORACLE as user DEMO.
   Oracle String is provided by caller as: "SCOTT/TIGER@T:GRAMPA:ORCL".
   returns 0 on success and 1 on SQL error.
-------------------------------------------------------------------------- */

oraconn(orastring)
        char orastring[80];
   {
   EXEC SQL BEGIN DECLARE SECTION;
       VARCHAR oracleid[80];          /* username/password@dbstring  */
   EXEC SQL END DECLARE SECTION;

   EXEC SQL WHENEVER SQLERROR GOTO errexit;

   strcpy((char *)oracleid.arr,orastring);
   oracleid.len = strlen((char *)oracleid.arr);

   EXEC SQL CONNECT :oracleid;

   return(0);

   /* Here if SQL Error */
   errexit:
     EXEC SQL WHENEVER SQLERROR CONTINUE;
     return(1);
   }



/* --------------------------------------------------------------------------
   ORADISC disconnects from ORACLE.
   returns 0 on success and 1 on SQL error.
-------------------------------------------------------------------------- */

oradisc()
   {

   EXEC SQL WHENEVER SQLERROR GOTO errexit;

   EXEC SQL COMMIT RELEASE;

   return(0);

   /* Here if SQL Error */
   errexit:
     EXEC SQL WHENEVER SQLERROR CONTINUE;
     return(1);
   }
```

```
/* ---------------------------------------------------------------------------
   ORACREATE creates omq_stock & omq_cust tables in ORACLE as follows:

   omq_stock:    ITEM            PRICE   QTY
                                 Bolts          $1              10000
                                 Buckets        $2              10000
                                 Buttons        $3              10000
                                 Belts          $4              10000
                                 Bobbins        $5              10000
                                 Boats          $6              10000

   omq_cust:     CUSTOMER        ORDERS  SALES
                                 Jacobs         0               0
                                 Jackson        0               0
                                 Jaffe          0               0
                                 Johnson        0               0
                                 Jones          0               0
                                 James          0       0

   returns 0 on success and 1 on SQL error.
-------------------------------------------------------------------------- */

oracreate()
   {

   /* Expect an error if table already dropped */
   EXEC SQL WHENEVER SQLERROR GOTO ignore;
   EXEC SQL DROP TABLE omq_stock;
   EXEC SQL DROP TABLE omq_cust;

   ignore:
     EXEC SQL WHENEVER SQLERROR GOTO errexit;

     EXEC SQL CREATE TABLE omq_stock
                (item   VARCHAR2(10)     PRIMARY KEY,
                 price  NUMBER,
                 qty    NUMBER);
     EXEC SQL INSERT INTO omq_stock VALUES('Bolts',1,10000);
         EXEC SQL INSERT INTO omq_stock VALUES('Buckets',2,10000);
         EXEC SQL INSERT INTO omq_stock VALUES('Buttons',3,10000);
         EXEC SQL INSERT INTO omq_stock VALUES('Belts',4,10000);
         EXEC SQL INSERT INTO omq_stock VALUES('Bobbins',5,10000);
         EXEC SQL INSERT INTO omq_stock VALUES('Boats',6,10000);

         EXEC SQL CREATE TABLE omq_cust
                (customer       VARCHAR2(10)     PRIMARY KEY,
                 orders         NUMBER,
                 sales          NUMBER);
         EXEC SQL INSERT INTO omq_cust VALUES('Jacobs',0,0);
         EXEC SQL INSERT INTO omq_cust VALUES('Jackson',0,0);
         EXEC SQL INSERT INTO omq_cust VALUES('Jaffe',0,0);
         EXEC SQL INSERT INTO omq_cust VALUES('Johnson',0,0);
         EXEC SQL INSERT INTO omq_cust VALUES('Jones',0,0);
         EXEC SQL INSERT INTO omq_cust VALUES('James',0,0);

         EXEC SQL COMMIT WORK;
```

```
      return(0);

    /* Here if SQL Error */
    errexit:
      EXEC SQL WHENEVER SQLERROR CONTINUE;
      return(1);
    }

/* ---------------------------------------------------------------------
    ORAREAD reads QTY & PRICE from omq_stock table given ITEM.
    returns 0 on success, 1 on SQL error and 2 on noitem.
------------------------------------------------------------------------ */

int oraread(itemname, itemprice, itemqty)
          char itemname[10];
          int *itemprice, *itemqty;
    {
    EXEC SQL BEGIN DECLARE SECTION;
        VARCHAR item[10];                   /* item name      */
        int     price;                      /* item price     */
        int     qty;                        /* item quantity  */
    EXEC SQL END DECLARE SECTION;

    EXEC SQL WHENEVER NOT FOUND GOTO noitem;
    EXEC SQL WHENEVER SQLERROR GOTO errexit;

    strcpy((char *)item.arr,itemname);
    item.len = strlen(itemname);

    EXEC SQL SELECT PRICE, QTY
               INTO :price, :qty
               FROM omq_stock
               WHERE ITEM = :item;

    EXEC SQL WHENEVER NOT FOUND STOP;
    EXEC SQL WHENEVER SQLERROR CONTINUE;

    *itemprice = price;
    *itemqty = qty;

    return(0);

    /* Here if item NOT found in dbs */
    noitem:
      EXEC SQL WHENEVER NOT FOUND STOP;
      EXEC SQL WHENEVER SQLERROR CONTINUE;
      return(2);

    /* Here if SQL Error */
    errexit:
      EXEC SQL WHENEVER NOT FOUND STOP;
      EXEC SQL WHENEVER SQLERROR CONTINUE;
      return(1);
    }

/* ---------------------------------------------------------------------
    ORAWRITE updates QTY for a given ITEM in the omq_stock table.
    returns 0 on success, 1 on SQL error and 2 on noitem.
```

```
------------------------------------------------------------------------ */

int orawrite(itemname, itemqty)
          char itemname[10];
          int itemqty;
   {
   EXEC SQL BEGIN DECLARE SECTION;
       VARCHAR item[10];                  /* item name      */
       int     qty;                       /* item quantity  */
   EXEC SQL END DECLARE SECTION;

   EXEC SQL WHENEVER NOT FOUND GOTO noitem;
   EXEC SQL WHENEVER SQLERROR GOTO errexit;

   strcpy((char *)item.arr,itemname);
   item.len = strlen(itemname);
   qty = itemqty;

   EXEC SQL UPDATE omq_stock
              SET QTY = :qty
              WHERE ITEM = :item;

   EXEC SQL COMMIT WORK;

   EXEC SQL WHENEVER NOT FOUND STOP;
   EXEC SQL WHENEVER SQLERROR CONTINUE;

   return(0);

   /* Here if item NOT found in dbs */
   noitem:
     EXEC SQL WHENEVER NOT FOUND STOP;
     EXEC SQL WHENEVER SQLERROR CONTINUE;
     return(2);

   /* Here if SQL Error */
   errexit:
     EXEC SQL WHENEVER NOT FOUND STOP;
     EXEC SQL WHENEVER SQLERROR CONTINUE;
     return(1);
   }


/* ---------------------------------------------------------------------
   ORACUSTR reads ORDERS & SALES from omq_cust table given customer name.
   returns 0 on success, 1 on SQL error and 2 on nocustomer.
------------------------------------------------------------------------ */

int oracustr(custname, custorders, custsales)
          char custname[10];
          int *custorders, *custsales;
   {
   EXEC SQL BEGIN DECLARE SECTION;
       VARCHAR customer[10];                  /* cust name      */
       int     orders;                        /* cust #orders   */
       int     sales;                         /* cust $sales    */
   EXEC SQL END DECLARE SECTION;

   EXEC SQL WHENEVER NOT FOUND GOTO noitem;
   EXEC SQL WHENEVER SQLERROR GOTO errexit;

   strcpy((char *)customer.arr,custname);
   customer.len = strlen(custname);

   EXEC SQL SELECT ORDERS, SALES
              INTO :orders, :sales
              FROM omq_cust
              WHERE CUSTOMER = :customer;

   EXEC SQL WHENEVER NOT FOUND STOP;
   EXEC SQL WHENEVER SQLERROR CONTINUE;

   *custorders = orders;
   *custsales = sales;

   return(0);

   /* Here if item NOT found in dbs */
   noitem:
     EXEC SQL WHENEVER NOT FOUND STOP;
     EXEC SQL WHENEVER SQLERROR CONTINUE;
     return(2);

   /* Here if SQL Error */
   errexit:
     EXEC SQL WHENEVER NOT FOUND STOP;
     EXEC SQL WHENEVER SQLERROR CONTINUE;
     return(1);
   }

/* ----------------------------------------------------------------------------
   ORACUSTW updates ORDERS & SALES for a given CUSTOMER in the DB.
   returns 0 on success, 1 on SQL error and 2 on nocustomer.
---------------------------------------------------------------------------- */

int oracustw(custname, custorders, custsales)
          char custname[10];
          int custorders, custsales;
   {
   EXEC SQL BEGIN DECLARE SECTION;
       VARCHAR customer[10];                  /* cust name       */
       int     orders;                        /* cust #orders    */
           int     sales;                         /* cust $sales     */
   EXEC SQL END DECLARE SECTION;

   EXEC SQL WHENEVER NOT FOUND GOTO noitem;
   EXEC SQL WHENEVER SQLERROR GOTO errexit;

   strcpy((char *)customer.arr,custname);
   customer.len = strlen(custname);
   orders = custorders;
   sales = custsales;

   EXEC SQL UPDATE omq_cust
              SET ORDERS = :orders,     SALES = :sales
              WHERE CUSTOMER = :customer;

   EXEC SQL COMMIT WORK;
EXEC SQL WHENEVER NOT FOUND STOP;
EXEC SQL WHENEVER SQLERROR CONTINUE;

return(0);

/* Here if item NOT found in dbs */
noitem:
  EXEC SQL WHENEVER NOT FOUND STOP;
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  return(2);

/* Here if SQL Error */
errexit:
  EXEC SQL WHENEVER NOT FOUND STOP;
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  return(1);
}
```

```
/* File name & Package Name */
struct sqlcxp
{
  unsigned short fillen;
           char  filnam[11];
};
static struct sqlcxp sqlfpn =
{
    10,
    "oratest.pc"
};


static const unsigned long sqlctx = 822081478;


static struct sqlexd {
   unsigned long    sqlvsn;
   unsigned long   arrsiz;
   unsigned long   iters;
   unsigned short   offset;
   unsigned short   selerr;
   unsigned short   sqlety;
  unsigned short   unused;
            short   *cud;
   unsigned char    *sqlest;
            char    *stmt;
   unsigned char  * *sqphsv;
   unsigned long   *sqphsl;
            short  * *sqpind;
   unsigned long   *sqparm;
   unsigned long   * *sqparc;
   unsigned char    *sqhstv[1];
   unsigned long    sqhstl[1];
            short   *sqindv[1];
   unsigned long    sqharm[1];
   unsigned long   *sqharc[1];
} sqlstm = {8,1};
extern sqlcx2(/*_ unsigned long , struct sqlexd *, struct sqlcxp * _*/);
extern sqlcte(/*_ unsigned long , struct sqlexd *, struct sqlcxp * _*/);
extern sqlbuf(/*_ char * _*/);
extern sqlora(/*_ unsigned long *, void * _*/);

static int IAPSUCC = 0;
static int IAPFAIL = 1403;
static int IAPFTL  = 535;
extern    sqliem();

/* cud (compilation unit data) array */
static short sqlcud0[] =
{8,34,
};

/*
**
** Copyright(C)1996 MITSUBISHI ELECTRIC ITA.  ALL RIGHTS RESERVED.
** UNPUBLISHED -- RIGHTS RESERVED UNDER THE COPYRIGHT
** LAWS OF THE UNITED STATES.  USE OF A COPYRIGHT NOTICE
** IS PRECAUTIONARY ONLY AND DOES NOT IMPLY PUBLICATION
** OR DISCLOSURE.
```

```
**
** THIS SOFTWARE CONTAINS CONFIDENTIAL INFORMATION AND
** TRADE SECRETS OF MITSUBISHI ELECTRIC ITA.  USE, DISCLOSURE,
** OR REPRODUCTION IS PROHIBITED WITHOUT THE PRIOR
** EXPRESS WRITTEN PERMISSION OF MITSUBISHI ELECTRIC ITA.
**
** OpenMQ Demo
**   Module: oratest.pc
**   Author: Frederick J. Igo, Jr. 1/15/96
**
*/

#include <stdlib.h>
#include <stdio.h>
#include <ctype.h>
#include <string.h>

# include "sqlproto.h"
# include "ociproto.h"
# define ORACLE
# include "oraomq.h"


/*
NAME
  openmq
FUNCTION
  Simple openmq Pro*C sample program
NOTES

   openmq  is a simple example program which decrements the stock
           for ordered items.  Checking is done for sufficient
           stock.

           The program queries the user for data as follows:

                   Enter customer name:
           Enter item name:
           Enter quantity ordered:

           The program terminates if null string (<return> key)
           is entered when the customer name is requested.

           If the item qty is updated, the following
           is printed:

           "Order for QTY ITEM at PRICE has been placed"

           C:\ORANT\PRO16\C> nmake -f openmq.mak

OWNER
  Igo
DATE
  01/11/96
MODIFIED
    igo        01/11/96 - create from sample32.pc
*/
```

```
void main()
{
   char    itemname[10];                  /* item name      */
   int     itemprice;                     /* item price     */
   int     itemqty;                       /* item quantity  */
   char    custname[10];                  /* cust name      */
   int     custorders;                    /* cust #orders   */
   int     custsales;                     /* cust $sales    */
   char    orastring[80];                 /* Oracle ID      */
   char    orastrinp[80];                 /* Oracle ID      */

/* -----------------------------------------------------------------------------
   logon to ORACLE, and open the cursors.
   The program exits if any errors occur.
----------------------------------------------------------------------------- */

   strcpy((char *)orastring,"SCOTT/TIGER@T:GRAMPA:ORCL");
   if (asks("\nAlter connect string? (null keeps SCOTT/TIGER@T:ORACLE:ORCL) ",
            (char *)orastrinp) > 0 ) {
      strcpy((char *)orastring,(char *)orastrinp);
   }

   printf("\nConnecting to Oracle using string %s...",orastring);
   if (oraconn(orastring) == 1) {
      oraerrrpt();
      printf("SQL Error on CONNECT, Bye-bye.");
      return;
      }
   printf("connected.\n");

   /* Create/Fill DB */
   if ( asks("\nShall we create/clear the DB (non-null to create)?",
            (char *)custname) > 0 ) {
      printf("Creating omq_stock and omq_cust tables in Oracle...\n");
      if (oracreate() == 1) {
         oraerrrpt();
         printf("SQL Error on CREATE, Bye-bye.");
         return;
         }
   }

/* -----------------------------------------------------------------------------
   Read the user's input from STDIN.  If the item name is not entered, exit.
   Verify that the entered quantity is lessthan that item's stock.
----------------------------------------------------------------------------- */

   for( ; ; )
     {
     int l,tot,qtyo;


     /* Get customer name  */
     l = asks("\nEnter customer name (null to quit): ", (char *)custname);
     if ( l <= 0 )
       break;

        printf("      Checking Customer %s in DB... ",custname);
     switch (oracustr(custname,&custorders,&custsales)) {
     case 1: {
```

```
    oraerrrpt();
    printf("SQL Error on Customer DB Read, try again.\n");
    continue;
    }
case 2: {
    printf("No such customer, try again.\n");
    continue;
    }
}

/* Get item name to be ordered    */
asks("\nEnter item name        : ", (char *)itemname);

/* Read DB with given item name to get qty and price */
    printf("      Checking item in DB... ");
switch (oraread(itemname,&itemprice,&itemqty)) {
case 1: {
    oraerrrpt();
    printf("SQL Error on Customer DB Read, try again.\n");
    continue;
    }
case 2: {
    printf("No such item, try again.\n");
    continue;
    }
}

    askn("\nEnter quantity ordered: ",&qtyo);
printf("      Checking stock for %d %s... ",qtyo,itemname);

if (qtyo > itemqty)
   {
    printf("Insufficient stock: %d.\n",itemqty);
   continue;
   }

/* Here if item was found in dbs and quantity suffucient. */

itemqty -= qtyo;
    tot = qtyo * itemprice;

/* Update DB for given item name */
printf("Updating %s QTY in Stock DB.\n",itemname);
switch (orawrite(itemname,itemqty)) {
case 0: {
    printf("\nOrder for %d %s at $%d each placed, ",
      qtyo,itemname,itemprice);
    printf("Sale is $%d, %d %s remain.\n",
      tot,itemqty,itemname);
    break;
    }
case 1: {
    oraerrrpt();
    printf("SQL Error during DB update, order not placed.\n");
    break;
    }
case 2: {
    /* Shouldn't get this case, since read found item */
    printf("No such item on write, please start over...\n");
    break;
    }
}

    /* Update Cust DB */
    custorders += 1;
    custsales += tot;

switch (oracustw(custname,custorders,custsales)) {
case 0: {
    printf("%s has placed %d orders for $%d.\n",
      custname,custorders,custsales);
    break;
    }
case 1: {
    oraerrrpt();
    printf("SQL Error during cust DB update.\n");
    break;
    }
case 2: {
    /* Shouldn't get this case, since read found item */
    printf("No such customer on write, please start over\n");
    break;
    }
}



    printf("\n=============================\n");
    printf("ITEM        PRICE     QTY\n");

    strcpy((char *)itemname,"Bolts");
    oraread(itemname,&itemprice,&itemqty);
    printf("Bolts       $%d        %d\n",itemprice,itemqty);

    strcpy((char *)itemname,"Buckets");
    oraread(itemname,&itemprice,&itemqty);
    printf("Buckets     $%d        %d\n",itemprice,itemqty);

    strcpy((char *)itemname,"Buttons");
    oraread(itemname,&itemprice,&itemqty);
    printf("Buttons     $%d        %d\n",itemprice,itemqty);

    strcpy((char *)itemname,"Belts");
    oraread(itemname,&itemprice,&itemqty);
    printf("Belts       $%d        %d\n",itemprice,itemqty);

    strcpy((char *)itemname,"Bobbins");
    oraread(itemname,&itemprice,&itemqty);
    printf("Bobbins     $%d        %d\n",itemprice,itemqty);

    strcpy((char *)itemname,"Boats");
    oraread(itemname,&itemprice,&itemqty);
    printf("Boats       $%d        %d\n",itemprice,itemqty);


    printf("============================\n");
    printf("CUSTOMER    ORDERS    SALES\n");

    strcpy((char *)custname,"Jacobs");
```

```
        oracustr(custname,&custorders,&custsales);
        printf("Jacobs      %d          $%d\n",custorders,custsales);

        strcpy((char *)custname,"Jackson");
        oracustr(custname,&custorders,&custsales);
        printf("Jackson     %d          $%d\n",custorders,custsales);

        strcpy((char *)custname,"Jones");
        oracustr(custname,&custorders,&custsales);
        printf("Jones       %d          $%d\n",custorders,custsales);

        strcpy((char *)custname,"Johnson");
        oracustr(custname,&custorders,&custsales);
        printf("Johnson     %d          $%d\n",custorders,custsales);

        strcpy((char *)custname,"Jaffe");
        oracustr(custname,&custorders,&custsales);
        printf("Jaffe       %d          $%d\n",custorders,custsales);

        strcpy((char *)custname,"James");
        oracustr(custname,&custorders,&custsales);
        printf("James       %d          $%d\n",custorders,custsales);

   }
/* ------------------------------------------------------------------------
   close the cursors and log off from ORACLE
------------------------------------------------------------------------ */

   printf ("\nDisconnecting from Oracle.\n");
   oradisc();
   printf ("\nEnd of OpenMQ/Pro*C example.\n");
   return;

}

/*------------------------------------------------------------------------
COUNT askn(text,variable)

   print the 'text' on STDOUT and read an integer variable from
   SDTIN.

   text points to the null terminated string to be printed
   variable points to an integer variable

   askn returns a 1 if the variable was read successfully or a
       -1 if -eof- was encountered
------------------------------------------------------------------------ */

int askn(text,variable)
   char text[];
   int  *variable;
   {
   char s[20];
   printf(text);
   fflush(stdout);
   if ( gets(s) == (char *)0 )
     return(EOF);

   *variable = atoi(s);
   return(1);
   }

/* ----------------------------------------------------------------------------
COUNT asks(text,variable)

   print the 'text' on STDOUT and read up to 'len' characters into
   the buffer pointed to by variable from STDIN.

   text points to the null terminated string to be printed
   variable points to a buffer of at least 'len'+1 characters

   asks returns the number of character read into the string, or a
       -1 if -eof- was encountered
--------------------------------------------------------------------------- */

int asks(text,variable)
   char text[],variable[];
   {
   printf(text);
   fflush(stdout);
   return( gets(variable) == (char *)0 ? EOF : strlen(variable) );
   }
```

```
/*
**
** Copyright(C)1996 MITSUBISHI ELECTRIC ITA.  ALL RIGHTS RESERVED.
** UNPUBLISHED -- RIGHTS RESERVED UNDER THE COPYRIGHT
** LAWS OF THE UNITED STATES.  USE OF A COPYRIGHT NOTICE
** IS PRECAUTIONARY ONLY AND DOES NOT IMPLY PUBLICATION
** OR DISCLOSURE.
**
** THIS SOFTWARE CONTAINS CONFIDENTIAL INFORMATION AND
** TRADE SECRETS OF MITSUBISHI ELECTRIC ITA.  USE, DISCLOSURE,
** OR REPRODUCTION IS PROHIBITED WITHOUT THE PRIOR
** EXPRESS WRITTEN PERMISSION OF MITSUBISHI ELECTRIC ITA.
**
** OpenMQ Demo
**   Module: oratest.pc
**   Author: Frederick J. Igo, Jr. 1/15/96
**
*/

#include <stdlib.h>
#include <stdio.h>
#include <ctype.h>
#include <string.h>

# include "sqlproto.h"
# include "ociproto.h"
# define ORACLE
# include "oraomq.h"


/*
NAME
  openmq
FUNCTION
  Simple openmq Pro*C sample program
NOTES

   openmq  is a simple example program which decrements the stock
           for ordered items.  Checking is done for sufficient
           stock.

           The program queries the user for data as follows:

                   Enter customer name:
           Enter item name:
           Enter quantity ordered:

           The program terminates if null string (<return> key)
           is entered when the customer name is requested.

           If the item qty is updated, the following
           is printed:

           "Order for QTY ITEM at PRICE has been placed"

           C:\ORANT\PRO16\C> nmake -f openmq.mak

OWNER
  Igo
DATE
```

```
  01/11/96
MODIFIED
    igo         01/11/96 - create from sample32.pc
*/



void main()
{
   char    itemname[10];                   /* item name      */
   int     itemprice;                      /* item price     */
   int     itemqty;                        /* item quantity  */
   char    custname[10];                   /* cust name      */
   int     custorders;                     /* cust #orders   */
   int     custsales;                      /* cust $sales    */
   char    orastring[80];                  /* Oracle ID      */
   char    orastrinp[80];                  /* Oracle ID      */

/* -----------------------------------------------------------------------
   logon to ORACLE, and open the cursors.
   The program exits if any errors occur.
---------------------------------------------------------------------- */

   strcpy((char *)orastring,"SCOTT/TIGER@T:GRAMPA:ORCL");
   if (asks("\nAlter connect string? (null keeps SCOTT/TIGER@T:ORACLE:ORCL) ",
            (char *)orastrinp) > 0 ) {
      strcpy((char *)orastring,(char *)orastrinp);
   }

   printf("\nConnecting to Oracle using string %s...",orastring);
   if (oraconn(orastring) == 1) {
      oraerrrpt();
      printf("SQL Error on CONNECT, Bye-bye.");
      return;
      }
   printf("connected.\n");

   /* Create/Fill DB */
   if ( asks("\nShall we create/clear the DB (non-null to create)?",
            (char *)custname) > 0 ) {
      printf("Creating omq_stock and omq_cust tables in Oracle...\n");
      if (oracreate() == 1) {
         oraerrrpt();
         printf("SQL Error on CREATE, Bye-bye.");
         return;
         }
   }

/* -----------------------------------------------------------------------
   Read the user's input from STDIN.  If the item name is not entered, exit.
   Verify that the entered quantity is lessthan that item's stock.
---------------------------------------------------------------------- */

   for( ; ; )
     {
     int l,tot,qtyo;


     /* Get customer name  */
```

```
l = asks("\nEnter customer name (null to quit): ", (char *)custname);
if ( l <= 0 )
  break;

    printf("      Checking Customer %s in DB... ",custname);
switch (oracustr(custname,&custorders,&custsales)) {
case 1: {
    oraerrrpt();
    printf("SQL Error on Customer DB Read, try again.\n");
    continue;
    }
case 2: {
    printf("No such customer, try again.\n");
    continue;
    }
}

/* Get item name to be ordered   */
asks("\nEnter item name       : ", (char *)itemname);

/* Read DB with given item name to get qty and price */
    printf("      Checking item in DB... ");
switch (oraread(itemname,&itemprice,&itemqty)) {
case 1: {
    oraerrrpt();
    printf("SQL Error on Customer DB Read, try again.\n");
    continue;
    }
case 2: {
    printf("No such item, try again.\n");
    continue;
    }
}

    askn("\nEnter quantity ordered: ",&qtyo);
printf("      Checking stock for %d %s... ",qtyo,itemname);

if (qtyo > itemqty)
   {
    printf("Insufficient stock: %d.\n",itemqty);
   continue;
   }

/* Here if item was found in dbs and quantity suffucient. */

itemqty -= qtyo;
    tot = qtyo * itemprice;

/* Update DB for given item name */
printf("Updating %s QTY in Stock DB.\n",itemname);
switch (orawrite(itemname,itemqty)) {
case 0: {
    printf("\nOrder for %d %s at $%d each placed, ",
      qtyo,itemname,itemprice);
    printf("Sale is $%d, %d %s remain.\n",
      tot,itemqty,itemname);
    break;
    }
case 1: {
    oraerrrpt();
```

```
        printf("SQL Error during DB update, order not placed.\n");
        break;
        }
    case 2: {
        /* Shouldn't get this case, since read found item */
        printf("No such item on write, please start over...\n");
        break;
        }
    }

        /* Update Cust DB */
        custorders += 1;
        custsales += tot;

    switch (oracustw(custname,custorders,custsales)) {
    case 0: {
        printf("%s has placed %d orders for $%d.\n",
          custname,custorders,custsales);
        break;
        }
    case 1: {
        oraerrrpt();
        printf("SQL Error during cust DB update.\n");
        break;
        }
    case 2: {
        /* Shouldn't get this case, since read found item */
        printf("No such customer on write, please start over\n");
        break;
        }
    }



        printf("\n=============================\n");
        printf("ITEM        PRICE     QTY\n");

        strcpy((char *)itemname,"Bolts");
        oraread(itemname,&itemprice,&itemqty);
        printf("Bolts       $%d         %d\n",itemprice,itemqty);

        strcpy((char *)itemname,"Buckets");
        oraread(itemname,&itemprice,&itemqty);
        printf("Buckets     $%d         %d\n",itemprice,itemqty);

        strcpy((char *)itemname,"Buttons");
        oraread(itemname,&itemprice,&itemqty);
        printf("Buttons     $%d         %d\n",itemprice,itemqty);

        strcpy((char *)itemname,"Belts");
        oraread(itemname,&itemprice,&itemqty);
        printf("Belts       $%d         %d\n",itemprice,itemqty);

        strcpy((char *)itemname,"Bobbins");
        oraread(itemname,&itemprice,&itemqty);
        printf("Bobbins     $%d         %d\n",itemprice,itemqty);

        strcpy((char *)itemname,"Boats");
        oraread(itemname,&itemprice,&itemqty);
        printf("Boats        $%d         %d\n",itemprice,itemqty);


        printf("============================\n");
        printf("CUSTOMER    ORDERS    SALES\n");

        strcpy((char *)custname,"Jacobs");
        oracustr(custname,&custorders,&custsales);
        printf("Jacobs      %d          $%d\n",custorders,custsales);

        strcpy((char *)custname,"Jackson");
        oracustr(custname,&custorders,&custsales);
        printf("Jackson     %d          $%d\n",custorders,custsales);

        strcpy((char *)custname,"Jones");
        oracustr(custname,&custorders,&custsales);
        printf("Jones       %d          $%d\n",custorders,custsales);

        strcpy((char *)custname,"Johnson");
        oracustr(custname,&custorders,&custsales);
        printf("Johnson     %d          $%d\n",custorders,custsales);

        strcpy((char *)custname,"Jaffe");
        oracustr(custname,&custorders,&custsales);
        printf("Jaffe       %d          $%d\n",custorders,custsales);

        strcpy((char *)custname,"James");
        oracustr(custname,&custorders,&custsales);
        printf("James       %d          $%d\n",custorders,custsales);

   }
/* -------------------------------------------------------------------
   close the cursors and log off from ORACLE
------------------------------------------------------------------ */

   printf ("\nDisconnecting from Oracle.\n");
   oradisc();
   printf ("\nEnd of OpenMQ/Pro*C example.\n");
   return;

}

/*--------------------------------------------------------------------
COUNT askn(text,variable)

   print the 'text' on STDOUT and read an integer variable from
   SDTIN.

   text points to the null terminated string to be printed
   variable points to an integer variable

   askn returns a 1 if the variable was read successfully or a
       -1 if -eof- was encountered
------------------------------------------------------------------ */

int askn(text,variable)
   char text[];
   int  *variable;
   {
```

```
   char s[20];
   printf(text);
   fflush(stdout);
   if ( gets(s) == (char *)0 )
     return(EOF);

   *variable = atoi(s);
   return(1);
   }

/* ---------------------------------------------------------------------------
COUNT asks(text,variable)

   print the 'text' on STDOUT and read up to 'len' characters into
   the buffer pointed to by variable from STDIN.

   text points to the null terminated string to be printed
   variable points to a buffer of at least 'len'+1 characters

   asks returns the number of character read into the string, or a
       -1 if -eof- was encountered
------------------------------------------------------------------------ */

int asks(text,variable)
   char text[],variable[];
   {
   printf(text);
   fflush(stdout);
   return( gets(variable) == (char *)0 ? EOF : strlen(variable) );
   }
```

```
User: root
Host: bunny
Class: bunny
Job: stdin
```

```
SETUP:

Install SQL*Net TCP\IP Client 1.1.6.8 on client.

Install Pro*C 1.6.4.0.1.

  For execution you only need C:\ORANT\PRO16\LIB\SQLNT16.DLL.
  I've saved a copy here for execution of oratest, if Pro*C
  is not installed.

  For building a .pc file, you need Pro*C installed.


Add "C:\ORANT\BIN" to system Path variable to pickup Oracle DLLs.

  (A copy of MSVCR40.DLL is saved here for execution of oratest.exe,
  if MSVC 4.0 is not installed.)



NOTES:

ORATEST currently uses oracleid SCOTT/TIGER@T:GRAMPA:ORCL, but
this can be altered when prompted for a connect string.

ORATEST can be used to verify the install on SQL*NET.

ORATEST can create the omq_stock and omq_cust tables in Oracle.

ORATEST can be used to view the omq_stock and omq_cust tables.
```

```
/*
**
** Copyright(C)1995 MITSUBISHI ELECTRIC ITA.  ALL RIGHTS RESERVED.
** UNPUBLISHED -- RIGHTS RESERVED UNDER THE COPYRIGHT
** LAWS OF THE UNITED STATES.  USE OF A COPYRIGHT NOTICE
** IS PRECAUTIONARY ONLY AND DOES NOT IMPLY PUBLICATION
** OR DISCLOSURE.
**
** THIS SOFTWARE CONTAINS CONFIDENTIAL INFORMATION AND
** TRADE SECRETS OF MITSUBISHI ELECTRIC ITA.  USE, DISCLOSURE,
** OR REPRODUCTION IS PROHIBITED WITHOUT THE PRIOR
** EXPRESS WRITTEN PERMISSION OF MITSUBISHI ELECTRIC ITA.
**
** OpenMQ
**   Module: putu.cpp
**   Author: Derek Schwenke 9/8/95
*/
// putu.cpp : Defines the class behaviors for the application.
//

#include "stdafx.h"
#include "putu.h"
#include "putudlg.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CPutuApp

BEGIN_MESSAGE_MAP(CPutuApp, CWinApp)
        //{{AFX_MSG_MAP(CPutuApp)
                // NOTE - the ClassWizard will add and remove mapping macros her
                //    DO NOT EDIT what you see in these blocks of generated code
        //}}AFX_MSG
        ON_COMMAND(ID_HELP, CWinApp::OnHelp)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CPutuApp construction

CPutuApp::CPutuApp()
{
        // TODO: add construction code here,
        // Place all significant initialization in InitInstance
}

/////////////////////////////////////////////////////////////////////////////
// The one and only CPutuApp object

CPutuApp theApp;

/////////////////////////////////////////////////////////////////////////////
// CPutuApp initialization

BOOL CPutuApp::InitInstance()
{
        // Standard initialization
// If you are not using these features and wish to reduce the size
//  of your final executable, you should remove from the following
//  the specific initialization routines you do not need.


Enable3dControls();
LoadStdProfileSettings();  // Load standard INI file options (including

CPutuDlg dlg;
m_pMainWnd = &dlg;

int nResponse = dlg.DoModal();
if (nResponse == IDOK)
{
        // TODO: Place code here to handle when the dialog is
        //  dismissed with OK
}
else if (nResponse == IDCANCEL)
{
        // TODO: Place code here to handle when the dialog is
        //  dismissed with Cancel
}

// Since the dialog has been closed, return FALSE so that we exit the
//  application, rather than start the application's message pump.
return FALSE;
```

```
// putu.h : main header file for the PUTU application
//

#ifndef __AFXWIN_H__
	#error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"		// main symbols

/////////////////////////////////////////////////////////////////////////////
// CPutuApp:
// See putu.cpp for the implementation of this class
//

class CPutuApp : public CWinApp
{
public:
	CPutuApp();

// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CPutuApp)
	public:
	virtual BOOL InitInstance();
	//}}AFX_VIRTUAL
   //virtual void printer( char *mess );

// Implementation

	//{{AFX_MSG(CPutuApp)
		// NOTE - the ClassWizard will add and remove member functions
		//    DO NOT EDIT what you see in these blocks of generated code
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};


/////////////////////////////////////////////////////////////////////////////
```

```
// putudlg.cpp : implementation file
//

#include "stdafx.h"
#include "putu.h"
#include "putudlg.h"
#include "putuqopt.h"
#include "putumopt.h"
#include "qlib.h"
//#include <windowsx.h>       // ListBox_AddString

//extern lpRT RTroot;

////////////////////////////////////////
lpQHANDLE   Q;
extern      lpSMBUFH sm_base;
////////////////////////////////////////

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
        CAboutDlg();

// Dialog Data
        //{{AFX_DATA(CAboutDlg)
        enum { IDD = IDD_ABOUTBOX };
        //}}AFX_DATA

// Implementation
protected:
        virtual void DoDataExchange(CDataExchange* pDX);        // DDX/DDV suppo
        //{{AFX_MSG(CAboutDlg)
        virtual BOOL OnInitDialog();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
        //{{AFX_DATA_INIT(CAboutDlg)
        //}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CAboutDlg)
        //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
        //{{AFX_MSG_MAP(CAboutDlg)
```

```
                // No message handlers
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg message handlers

BOOL CAboutDlg::OnInitDialog()
{
        CDialog::OnInitDialog();
        CenterWindow();

        // TODO: Add extra about dlg initialization here

        return TRUE;  // return TRUE  unless you set the focus to a control
}

/////////////////////////////////////////////////////////////////////////////
// CPutuDlg dialog

CPutuDlg::CPutuDlg(CWnd* pParent /*=NULL*/)
        : CDialog(CPutuDlg::IDD, pParent)
{
        //{{AFX_DATA_INIT(CPutuDlg)
        m_qnames = _T("");
        m_message = _T("");
        m_qstatus = _T("");
        m_mstatus = _T("");
        //}}AFX_DATA_INIT
        // Note that LoadIcon does not require a subsequent DestroyIcon in Wir
        m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CPutuDlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CPutuDlg)
        DDX_CBString(pDX, IDC_QNAMES, m_qnames);
        DDX_Text(pDX, IDC_MESS, m_message);
        DDX_Text(pDX, IDC_QSTATUS, m_qstatus);
        DDX_Text(pDX, IDC_MSTATUS, m_mstatus);
        //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CPutuDlg, CDialog)
        //{{AFX_MSG_MAP(CPutuDlg)
        ON_WM_SYSCOMMAND()
        ON_WM_PAINT()
        ON_WM_QUERYDRAGICON()
        ON_BN_CLICKED(IDC_BUT_Q_OPEN, OnButQOpen)
        ON_BN_CLICKED(IDC_BUT_QCLOSE, OnButQclose)
        ON_BN_CLICKED(IDC_BUT_SEND, OnButSend)
        ON_BN_CLICKED(IDC_BUT_QOPTS, OnButQopts)
        ON_BN_CLICKED(IDC_BUT_MOPTS, OnButMopts)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CPutuDlg message handlers
```

```
BOOL CPutuDlg::OnInitDialog()
{
        CDialog::OnInitDialog();
        CenterWindow();

        // Add "About..." menu item to system menu.

        // IDM_ABOUTBOX must be in the system command range.
        ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
        ASSERT(IDM_ABOUTBOX < 0xF000);

        CMenu* pSysMenu = GetSystemMenu(FALSE);
        CString strAboutMenu;
        strAboutMenu.LoadString(IDS_ABOUTBOX);
        if (!strAboutMenu.IsEmpty())
        {
                pSysMenu->AppendMenu(MF_SEPARATOR);
                pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
        }

        // Set printing window for "C" code
   CWnd* pLogWin = this->GetDlgItem(IDC_PRINTER);
        SetWinPtr(pLogWin->m_hWnd);

   ////////////////////////////////////////////////
   ////////////// QUE PULL DOWN //////////////////
   ////////////////////////////////////////////////

   CComboBox * CB = (CComboBox *) this->GetDlgItem(IDC_QNAMES);
   //CB->AddString("INITAL1");
   lpRT rtp = GetRTroot();
   while (rtp) {
      char *e,*s = rtp->apps;      // Starts after the first letter
      while (e = strchr(s,',')) { // Ends at next ","
         *e = 0;
         if (!strchr(s,'['))
            CB->AddString(s);
         *e = ',';
         s = e + 1;
      }
      rtp = rtp->next;
   }



        return TRUE;  // return TRUE  unless you set the focus to a control
}

void CPutuDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
        if ((nID & 0xFFF0) == IDM_ABOUTBOX)
        {
                CAboutDlg dlgAbout;
                dlgAbout.DoModal();
        }
        else
        {
                CDialog::OnSysCommand(nID, lParam);
        }
```

```
}

// If you add a minimize button to your dialog, you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.


void CPutuDlg::OnPaint()
{
        if (IsIconic())
        {
                CPaintDC dc(this); // device context for painting

                SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

                // Center icon in client rectangle
                int cxIcon = GetSystemMetrics(SM_CXICON);
                int cyIcon = GetSystemMetrics(SM_CYICON);
                CRect rect;
                GetClientRect(&rect);
                int x = (rect.Width() - cxIcon + 1) / 2;
                int y = (rect.Height() - cyIcon + 1) / 2;

                // Draw the icon
                dc.DrawIcon(x, y, m_hIcon);
        }
        else
        {
                CDialog::OnPaint();
        }
}

// The system calls this to obtain the cursor to display while the user drags
//  the minimized window.
HCURSOR CPutuDlg::OnQueryDragIcon()
{
        return (HCURSOR) m_hIcon;
}

void CPutuDlg::OnButQOpen()
{
        int status;
   char qname[100],line[100];

   SetDlgItemText(IDC_QSTATUS,"Opening...");

   GetDlgItemText(IDC_QNAMES,qname,100);

   ReadParms();

   if (!(Q = Qopen(qname,PUTTING,0,0,0,0,0)))
      sprintf(line,"FAILED to open %s",qname);
   else
      sprintf(line,"OPENED %s",qname);

   SetDlgItemText(IDC_QSTATUS,line);
}

void CPutuDlg::OnButQclose()
{
        SetDlgItemText(IDC_QSTATUS,"CLOSED");
}

void CPutuDlg::OnButSend()
{
   char got[100], line[100];


   GetDlgItemText(IDC_MESS,got,90);
   sprintf(line,"Sending:%s",got);    // C++ does this better?
   SetDlgItemText(IDC_MSTATUS,line);
   SetDlgItemText(IDC_MESS,"");       // Blank out the message

   // How do I force these to print out here?
   // PeekMessage
   // AddItem();



   if (QSUCCESS != Qput(Q,0,0, 0,sizeof(got),got))
      sprintf(line,"FAILED to send:%s",got);
   else
      sprintf(line,"Sent:%s",got);
   SetDlgItemText(IDC_MSTATUS,line);

}
 /*
void CPutuDlg::printer(char *mess) {
       //   CEdit::SetWindowText
CEdit * CE = (CEdit *) this->GetDlgItem(IDC_PRINTER);
       //CEdit * CE = (CEdit *) GetDlgItem(IDC_PRINTER);
       //CE->GetWindowText("");
CE->SetWindowText("NEW TEXT 1 \n SECOND LINE \n");
}

void printer(char *mess) {
    SetDlgItemText(IDC_MESS,mess);
}

  */

void CPutuDlg::OnButQopts()
{
   CPutuQOpts qopt;
   TRACE("AT QOPTS BUTTON");
        int nResponse = qopt.DoModal();
        if (nResponse == IDOK)
        {
                // TODO: Place code here to handle when the dialog is
                //  dismissed with OK
        }
        else if (nResponse == IDCANCEL)
        {
                // TODO: Place code here to handle when the dialog is
                //  dismissed with Cancel
        }

}
```

```
void CPutuDlg::OnButMopts()
{
        // TODO: Add your control notification handler code here
   TRACE("AT MOPTS BUTTON");

   CPutuMOpts mopt;
        int nResponse = mopt.DoModal();
        if (nResponse == IDOK)
        {
                // TODO: Place code here to handle when the dialog is
                //  dismissed with OK
        }
        else if (nResponse == IDCANCEL)
        {
                // TODO: Place code here to handle when the dialog is
                //  dismissed with Cancel
        }
}
```

```
// putudlg.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CPutuDlg dialog

class CPutuDlg : public CDialog
{
// Construction
public:
	CPutuDlg(CWnd* pParent = NULL); // standard constructor

// Dialog Data
	//{{AFX_DATA(CPutuDlg)
	enum { IDD = IDD_PUTU_DIALOG };
	CString m_qnames;
	CString m_message;
	CString m_qstatus;
	CString m_mstatus;
	//}}AFX_DATA

	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CPutuDlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);	// DDX/DDV suppo
	//}}AFX_VIRTUAL
 //  void printer( char *mess );   // This worked

// Implementation
protected:
	HICON m_hIcon;

	// Generated message map functions
	//{{AFX_MSG(CPutuDlg)
	virtual BOOL OnInitDialog();
	afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
	afx_msg void OnPaint();
	afx_msg HCURSOR OnQueryDragIcon();
	afx_msg void OnButQOpen();
	afx_msg void OnButQclose();
	afx_msg void OnButSend();
	afx_msg void OnButQopts();
	afx_msg void OnButMopts();
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};
```

```
/*
**
** Copyright(C)1995 MITSUBISHI ELECTRIC ITA.  ALL RIGHTS RESERVED.
** UNPUBLISHED -- RIGHTS RESERVED UNDER THE COPYRIGHT
** LAWS OF THE UNITED STATES.  USE OF A COPYRIGHT NOTICE
** IS PRECAUTIONARY ONLY AND DOES NOT IMPLY PUBLICATION
** OR DISCLOSURE.
**
** THIS SOFTWARE CONTAINS CONFIDENTIAL INFORMATION AND
** TRADE SECRETS OF MITSUBISHI ELECTRIC ITA.  USE, DISCLOSURE,
** OR REPRODUCTION IS PROHIBITED WITHOUT THE PRIOR
** EXPRESS WRITTEN PERMISSION OF MITSUBISHI ELECTRIC ITA.
**
** OpenMQ
**   Module: putumopt.c
**   Author: Derek Schwenke 9/8/95
*/

// putumopt.cpp : implementation file
//

#include "stdafx.h"
#include "putu.h"
#include "putumopt.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CPutuMOpts dialog


CPutuMOpts::CPutuMOpts(CWnd* pParent /*=NULL*/)
        : CDialog(CPutuMOpts::IDD, pParent)
{
        //{{AFX_DATA_INIT(CPutuMOpts)
                // NOTE: the ClassWizard will add member initialization here
        //}}AFX_DATA_INIT
}


void CPutuMOpts::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CPutuMOpts)
                // NOTE: the ClassWizard will add DDX and DDV calls here
        //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CPutuMOpts, CDialog)
        //{{AFX_MSG_MAP(CPutuMOpts)
                // NOTE: the ClassWizard will add message map macros here
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()


/////////////////////////////////////////////////////////////////////////////
```

```
// CPutuMOpts message handlers
// putumopt.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CPutuMOpts dialog

class CPutuMOpts : public CDialog
{
// Construction
public:
	CPutuMOpts(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
	//{{AFX_DATA(CPutuMOpts)
	enum { IDD = IDD_MESS_OPTS };
		// NOTE: the ClassWizard will add data members here
	//}}AFX_DATA


// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CPutuMOpts)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:

	// Generated message map functions
	//{{AFX_MSG(CPutuMOpts)
		// NOTE: the ClassWizard will add member functions here
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};
```

```
/*
**
** Copyright(C)1995 MITSUBISHI ELECTRIC ITA.  ALL RIGHTS RESERVED.
** UNPUBLISHED -- RIGHTS RESERVED UNDER THE COPYRIGHT
** LAWS OF THE UNITED STATES.  USE OF A COPYRIGHT NOTICE
** IS PRECAUTIONARY ONLY AND DOES NOT IMPLY PUBLICATION
** OR DISCLOSURE.
**
** THIS SOFTWARE CONTAINS CONFIDENTIAL INFORMATION AND
** TRADE SECRETS OF MITSUBISHI ELECTRIC ITA.  USE, DISCLOSURE,
** OR REPRODUCTION IS PROHIBITED WITHOUT THE PRIOR
** EXPRESS WRITTEN PERMISSION OF MITSUBISHI ELECTRIC ITA.
**
** OpenMQ
**   Module: putuqopt.c
**   Author: Derek Schwenke 9/8/95
*/
// putuqopt.cpp : implementation file
//

#include "stdafx.h"
#include "putu.h"
#include "putuqopt.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CPutuQOpts dialog


CPutuQOpts::CPutuQOpts(CWnd* pParent /*=NULL*/)
        : CDialog(CPutuQOpts::IDD, pParent)
{
        //{{AFX_DATA_INIT(CPutuQOpts)
        m_log_sw = FALSE;
        m_trace_sw = FALSE;
        m_tran_sw = FALSE;
        //}}AFX_DATA_INIT
}


void CPutuQOpts::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CPutuQOpts)
        DDX_Check(pDX, IDC_LOG_SW, m_log_sw);
        DDX_Check(pDX, IDC_TRACE_SW, m_trace_sw);
        DDX_Check(pDX, IDC_TRAN_SW, m_tran_sw);
        //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CPutuQOpts, CDialog)
        //{{AFX_MSG_MAP(CPutuQOpts)
                // NOTE: the ClassWizard will add message map macros here
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

```
/////////////////////////////////////////////////////////////////////////////
// CPutuQOpts message handlers
// putuqopt.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CPutuQOpts dialog

class CPutuQOpts : public CDialog
{
// Construction
public:
	CPutuQOpts(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
	//{{AFX_DATA(CPutuQOpts)
	enum { IDD = IDD_OPEN_OPTS };
	BOOL	m_log_sw;
	BOOL	m_trace_sw;
	BOOL	m_tran_sw;
	//}}AFX_DATA


// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CPutuQOpts)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:

	// Generated message map functions
	//{{AFX_MSG(CPutuQOpts)
		// NOTE: the ClassWizard will add member functions here
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};
```

```
//{{NO_DEPENDENCIES}}
// Microsoft Visual C++ generated include file.
// Used by putu.rc
//
#define IDM_ABOUTBOX                    0x0010
#define IDD_ABOUTBOX                    100
#define IDS_ABOUTBOX                    101
#define IDD_PUTU_DIALOG                 102
#define IDC_PRINTER                     104
#define IDR_MAINFRAME                   128
#define IDD_OPEN_OPTS                   130
#define IDD_MESS_OPTS                   131
#define IDC_QNAMES                      1000
#define IDC_MESS                        1004
#define IDC_BUT_Q_OPEN                  1005
#define IDC_BUT_SEND                    1006
#define IDC_BUT_QCLOSE                  1007
#define IDC_QSTATUS                     1008
#define IDC_MSTATUS                     1009
#define IDC_BUT_QOPTS                   1010
#define IDC_BUT_MOPTS                   1011
#define IDC_LOG_SW                      1018
#define IDC_TRACE_SW                    1019
#define IDC_TRAN_SW                     1020
#define IDC_TRAN_SW2                    1021
#define IDC_TRAN_SW3                    1022

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE        132
#define _APS_NEXT_COMMAND_VALUE         32771
#define _APS_NEXT_CONTROL_VALUE         1019
#define _APS_NEXT_SYMED_VALUE           101
#endif
#endif
```

```
/*
**
** Copyright(C)1996 MITSUBISHI ELECTRIC ITA.  ALL RIGHTS RESERVED.
** UNPUBLISHED -- RIGHTS RESERVED UNDER THE COPYRIGHT
** LAWS OF THE UNITED STATES.  USE OF A COPYRIGHT NOTICE
** IS PRECAUTIONARY ONLY AND DOES NOT IMPLY PUBLICATION
** OR DISCLOSURE.
**
** THIS SOFTWARE CONTAINS CONFIDENTIAL INFORMATION AND
** TRADE SECRETS OF MITSUBISHI ELECTRIC ITA.  USE, DISCLOSURE,
** OR REPRODUCTION IS PROHIBITED WITHOUT THE PRIOR
** EXPRESS WRITTEN PERMISSION OF MITSUBISHI ELECTRIC ITA.
**
** OpenMQ Demo
**   Module: oraomq.h
**   Author: Derek Schwenke 1/11/96
**
*/


#ifdef CPP
extern "C"
{
#endif


#ifdef ORACLE
   int orawrite( char itemname[10],int itemqty);
   int oraread(  char itemname[10],int *itemprice, int *itemqty);
   int oracustw( char custname[10],int custorders, int custsales);
   int oracustr( char custname[10],int *custorders, int *custsales);
   int oradisc();
   int oraconn(  char orastring[80]);
   int oracreate();
   int oraerrrpt();
#else
   int orawrite( char itemname[10],int itemqty)
   int oraread(  char itemname[10],int *itemprice, int *itemqty)
   int oracustw( char custname[10],int custorders, int custsales)
   int oracustr( char custname[10],int *custorders, int *custsales)
   int oradisc()
   int oraconn(  char orastring[80])
   int oracreate()
   int oraerrrpt()
#endif


#ifdef CPP
}
#endif
```

```
// Demo app's order form

typedef struct oform {  // Thread parameters
   char     cust[10];
   char     item[10];
   int      qty;
   int      color;
   int      reply_to;
} OFORM, *pOFORM, *lpOFORM;
```

```
// dbdlg.cpp : implementation file
//

#include "stdafx.h"


//#include "oentrvw.h"




#include "oentry.h"
#include "dbdlg.h"
#include "Odlg.h"

// #define ORACLE causes oraread() orawrite() to be exteraly defined
#ifndef ORACLE
#define ORACLE
#endif
#include "oraomq.h"


#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif
#define DB_TIMER 200
#define INIT_TIMER 10
#define RESTOCK_QTY 10000

int g_orders_mode = 0; // Display orders or items
extern CString g_item[];
extern CString g_cust[];
extern int g_price[6];
extern int g_qty[6];
       int g_qty_old[6];
extern int g_purchases[6];
       int g_purchases_old[6];
extern int g_num_purchases[6];
       int g_num_purchases_old[6];
extern int g_total_sales;
       int g_total_sales_old;
extern int g_db_run;
extern int g_ora_state;
       int g_ora_state_old = -1;
extern CFont g_title_font;
extern CFont g_text_font;

enum dbIDC {qty_IDC,price_IDC,item_IDC};

int  ALL_TEXT_DB[] = {IDC_ORDERS_ITEMS,IDC_DB_REFILL,IDC_DB_BOX,IDOK,
                      IDC_DB_SALEST,IDC_DB_SALES,
                      IDC_T11,IDC_T12,IDC_T13,
                      IDC_T21,IDC_T22,IDC_T23,
                      IDC_DB_Q0,IDC_DB_Q1,IDC_DB_Q2,IDC_DB_Q3,IDC_DB_Q4,IDC_DB_Q5
                      IDC_DB_P0,IDC_DB_P1,IDC_DB_P2,IDC_DB_P3,IDC_DB_P4,IDC_DB
                      IDC_DB_I0,IDC_DB_I1,IDC_DB_I2,IDC_DB_I3,IDC_DB_I4,IDC_DB_

int  g_IDCt[2][3] ={{IDC_T11,IDC_T12,IDC_T13},
                    {IDC_T21,IDC_T22,IDC_T23}};
```

```
int  g_IDCs[3][6] = {{IDC_DB_Q0,IDC_DB_Q1,IDC_DB_Q2,IDC_DB_Q3,IDC_DB_Q4,IDC_DB_Q
                    {IDC_DB_P0,IDC_DB_P1,IDC_DB_P2,IDC_DB_P3,IDC_DB_P4,IDC_DB_P5
                    {IDC_DB_I0,IDC_DB_I1,IDC_DB_I2,IDC_DB_I3,IDC_DB_I4,IDC_DB_I5
CString  g_item_tites[] = {"Qty","Price","Item"};
CString  g_order_titles[] = {"#","Amt","Customer"};



/////////////////////////////////////////////////////////////////////////////
// CDbDlg dialog


CDbDlg::CDbDlg(CWnd* pParent /*=NULL*/)
        : CDialog(CDbDlg::IDD, pParent)
{
        //{{AFX_DATA_INIT(CDbDlg)
                // NOTE: the ClassWizard will add member initialization here
        //}}AFX_DATA_INIT
}


void CDbDlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CDbDlg)
                // NOTE: the ClassWizard will add DDX and DDV calls here
        //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CDbDlg, CDialog)
        //{{AFX_MSG_MAP(CDbDlg)
        ON_BN_CLICKED(IDC_ORDERS_ITEMS, OnOrdersItems)
        ON_WM_TIMER()
        ON_WM_CREATE()
        ON_WM_DESTROY()
        ON_BN_CLICKED(IDC_DB_REFILL, OnDbRefill)
        ON_WM_RBUTTONDOWN()
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CDbDlg message handlers

void CDbDlg::OnOrdersItems()
{
   int i,j;
   CString s;
        if (g_orders_mode) { // Go into Items mode
      g_orders_mode = 0;
      SetDlgItemText(IDC_DB_REFILL,"Refill");

      // Set DB titles
          SetDlgItemText(IDC_DB_BOX,"Database Items");
          SetDlgItemText(IDC_ORDERS_ITEMS,"Orders");
      for (i=0;i<2;i++)
         for (j=0;j<3;j++)
```

```
                SetDlgItemText(g_IDCt[i][j],g_item_tites[j]);
      // Set DB values
      for (i=0; i<6 ;i++){ // For each item
         SetDlgItemInt(g_IDCs[qty_IDC][i]   ,g_qty[i]);
         s.Format("$%d.00",g_price[i]);
         SetDlgItemText(g_IDCs[price_IDC][i],s);
         SetDlgItemText(g_IDCs[item_IDC][i],g_item[i]);
      }

        } else { // Go into Orders mode
      g_orders_mode = 1;
      SetDlgItemText(IDC_DB_REFILL,"Clear");

      // Set DB titles
          SetDlgItemText(IDC_DB_BOX,"Database Orders");
          SetDlgItemText(IDC_ORDERS_ITEMS,"Items");
      for (i=0;i<2;i++)
         for (j=0;j<3;j++)
                SetDlgItemText(g_IDCt[i][j],g_order_titles[j]);
      // Set DB values
      for (i=0; i<6 ;i++){ // For each item
         SetDlgItemInt(g_IDCs[qty_IDC][i]   ,g_num_purchases[i]);
         SetDlgItemInt(g_IDCs[price_IDC][i],g_purchases[i]);
         SetDlgItemText(g_IDCs[item_IDC][i],g_cust[i]);
      }

   }
   for (i=0; i<6 ;i++){ // Invalidate any histoy
      g_num_purchases_old[i] =
      g_purchases_old[i] =
      g_qty_old[i] = -1;
   }
}

void CDbDlg::OnTimer(UINT nIDEvent)
{
   int i,rc,price,stock,cust_orders,cust_sales;
   CString s;

        if (nIDEvent == DB_TIMER) {
      if (g_db_run < 20) {
         if (g_db_run == 0) this->DestroyWindow();
         if (g_db_run == 1) {g_orders_mode = 1; OnOrdersItems(); g_db_run = 20;}
      }
      if (g_ora_state) {  // oracle db
         if (g_orders_mode){

            for (i=0; i<6 ;i++){ // For each item
               if (  (rc = oracustr(g_cust[i].GetBuffer(0), &cust_orders, &cust_
                  s.Format("OraCustRead ErroR %d",rc);
                  GetParentFrame()->SetMessageText(s);
               }
               if (g_num_purchases_old[i] != cust_orders)
                  SetDlgItemInt(g_IDCs[qty_IDC][i]   ,(g_num_purchases_old[i] = c
               if (g_purchases_old[i] != cust_sales)
                  SetDlgItemInt(g_IDCs[price_IDC][i],(g_purchases_old[i] = cu
               }
         } else {

            for (i=0; i<6 ;i++){ // For each item
```

```
                if (  (rc = oraread(g_item[i].GetBuffer(0), &price, &stock))  ) {
                   s.Format("OraRead ErroR %d",rc);
                   GetParentFrame()->SetMessageText(s);
                }
                if (g_qty_old[i] != stock)
                   SetDlgItemInt(g_IDCs[qty_IDC][i]  ,(g_qty_old[i] = stock));
             }
          }
       } else if (g_orders_mode){ // NOT ora_state, so use the local db
          for (i=0; i<6 ;i++){ // For each item
             if (g_num_purchases_old[i] != g_num_purchases[i])
                SetDlgItemInt(g_IDCs[qty_IDC][i]  ,(g_num_purchases_old[i] = g_nu
             if (g_purchases_old[i] != g_purchases[i])
                SetDlgItemInt(g_IDCs[price_IDC][i],(g_purchases_old[i] = g_purcha
          }
       } else {   // local db
          for (i=0; i<6 ;i++){ // For each item
             if (g_qty[i] != g_qty_old[i])
             SetDlgItemInt(g_IDCs[qty_IDC][i]  ,(g_qty_old[i] = g_qty[i]));
          }
       }
       if (g_total_sales_old != g_total_sales)
          SetDlgItemInt(IDC_DB_SALES,(g_total_sales_old = g_total_sales));



       if (g_ora_state_old != g_ora_state){
          if (g_ora_state_old = g_ora_state)
             SetDlgItemText(IDC_BIG_TITLE,"Oracle");
          else
             SetDlgItemText(IDC_BIG_TITLE,"Local DB");
       }
    } else if (nIDEvent == INIT_TIMER) {
       KillTimer(INIT_TIMER);

            GetDlgItem(IDC_BIG_TITLE)->SetFont(&g_title_font);

       if (g_ora_state)
          SetDlgItemText(IDC_BIG_TITLE,"Oracle");
       else
          SetDlgItemText(IDC_BIG_TITLE,"Local DB");

    } else

          CDialog::OnTimer(nIDEvent);
}

int CDbDlg::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
        if (CDialog::OnCreate(lpCreateStruct) == -1)
                return -1;


    SetTimer(DB_TIMER,200,NULL);
    SetTimer(INIT_TIMER,200,NULL);
    g_db_run = 1; // Start

    return 0;
}
```

```
void CDbDlg::OnDestroy()
{
        g_db_run = 40;

        CDialog::OnDestroy();

   KillTimer(DB_TIMER);
}

void CDbDlg::OnOK()
{
        g_db_run = 40; // Re enable the show db call button
        this->DestroyWindow();

        // CDialog::OnOK();
}

void CDbDlg::OnDbRefill()
{
        int i,rc;
   CString s;

   if (g_ora_state) {
      for (i=0;i<6;i++) {
         if (g_orders_mode) {
            if (  (rc = oracustw(g_cust[i].GetBuffer(0), 0, 0))  ) {
               s.Format("OraCustWrite ERRor %d",rc);
               GetParentFrame()->SetMessageText(s);
            }
         } else { // in items mode
            if (rc = orawrite(g_item[i].GetBuffer(0), RESTOCK_QTY)) {
               s.Format("OraWrite ERRor %d",rc);
               GetParentFrame()->SetMessageText(s);
            }
         }
      }
   } else {
      for (i=0;i<6;i++) {
         if (g_orders_mode) {
            g_purchases[i] = 0;
            g_num_purchases[i] = 0;
         } else { // in items mode
            g_qty[i] = RESTOCK_QTY;
         }
      }
   }
}


void CDbDlg::OnRButtonDown(UINT nFlags, CPoint point)
{
   GetParentFrame()->SetMessageText("");
   this->Invalidate();

        CDialog::OnRButtonDown(nFlags, point);
}

BOOL CDbDlg::OnInitDialog()
```

```
{
        CDialog::OnInitDialog();

    // Fonts
   int i = 0;
   while (ALL_TEXT_DB[i])
      GetDlgItem(ALL_TEXT_DB[i++])->SetFont(&g_text_font);

        return TRUE;  // return TRUE unless you set the focus to a control
                      // EXCEPTION: OCX Property Pages should return FALSE
}
```

```
// dbdlg.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CDbDlg dialog

class CDbDlg : public CDialog
{
// Construction
public:
	CDbDlg(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
	//{{AFX_DATA(CDbDlg)
	enum { IDD = IDD_DBDLG };
		// NOTE: the ClassWizard will add data members here
	//}}AFX_DATA


// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CDbDlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
	//}}AFX_VIRTUAL


   CFont    m_title_font;
   int      m_was_inited;

   // Implementation
protected:

	// Generated message map functions
	//{{AFX_MSG(CDbDlg)
	afx_msg void OnOrdersItems();
	afx_msg void OnTimer(UINT nIDEvent);
	afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
	afx_msg void OnDestroy();
	virtual void OnOK();
	afx_msg void OnDbRefill();
	afx_msg void OnRButtonDown(UINT nFlags, CPoint point);
	virtual BOOL OnInitDialog();
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};
```

```
}

CMainFrame::~CMainFrame()
{
}

int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
        if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
                return -1;

        if (!m_wndToolBar.Create(this) ||
                !m_wndToolBar.LoadBitmap(IDR_MAINFRAME) ||
                !m_wndToolBar.SetButtons(buttons,
                  sizeof(buttons)/sizeof(UINT)))
        {
                TRACE0("Failed to create toolbar\n");
                return -1;      // fail to create
        }


/* Derek's remove tool bar */
    m_wndToolBar.ShowWindow(SW_HIDE);



        if (!m_wndStatusBar.Create(this) ||
                !m_wndStatusBar.SetIndicators(indicators,
                  sizeof(indicators)/sizeof(UINT)))
        {
                TRACE0("Failed to create status bar\n");
                return -1;      // fail to create
        }

        // TODO: Delete these three lines if you don't want the toolbar to
        //  be dockable
        m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
        EnableDocking(CBRS_ALIGN_ANY);
        DockControlBar(&m_wndToolBar);

        // TODO: Remove this if you don't want tool tips
        m_wndToolBar.SetBarStyle(m_wndToolBar.GetBarStyle() |
                CBRS_TOOLTIPS | CBRS_FLYBY);

        return 0;
}

/////////////////////////////////////////////////////////////////////////////
// CMainFrame diagnostics

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
        CFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{
        CFrameWnd::Dump(dc);
```

```
}

#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CMainFrame message handlers
// mainfrm.h : interface of the CMainFrame class
//
/////////////////////////////////////////////////////////////////////////////

class CMainFrame : public CFrameWnd
{
protected: // create from serialization only
        CMainFrame();
        DECLARE_DYNCREATE(CMainFrame)

// Attributes
public:

// Operations
public:

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CMainFrame)
        //}}AFX_VIRTUAL

// Implementation
public:
        virtual ~CMainFrame();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

protected:  // control bar embedded members
        CStatusBar  m_wndStatusBar;
        CToolBar    m_wndToolBar;

// Generated message map functions
protected:
        //{{AFX_MSG(CMainFrame)
        afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
                // NOTE - the ClassWizard will add and remove member functions h
                //    DO NOT EDIT what you see in these blocks of generated code
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////
```

```
// Odlg.cpp : implementation file
//

#include "stdafx.h"
#include "oentry.h"
#include "Odlg.h"



// #define ORACLE causes oraread() orawrite() to be exteraly defined
#ifndef ORACLE
#define ORACLE
#endif
#include "oraomq.h"



#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

extern int g_options_run;
extern int g_fill_delay;
extern int g_place_delay;
extern int g_place_tpm;
extern int g_poll_pps;
extern int g_poll_delay;
extern int g_clear_stats;
extern int g_ora_state;
extern COLORREF g_new_color;
extern char g_oracle_con_str[80];
extern CFont g_text_font;

int ALL_TEXT_O[] = { IDOK,IDC_COLOR,IDC_CLRSTATS,IDC_ORACREATE,
                     IDC_FILLBOX,IDC_DLY_EB,IDC_DLYMAX,IDC_DLYMIN,IDC_DLY_LAB,
                     IDC_POLL_BOX,IDC_POLL_EB,IDC_POLLMAX,IDC_POLLMIN,IDC_POLL_L
                     IDC_AUTOBOX,IDC_AUTO_EB,IDC_AUTOMAX,IDC_AUTOMIN,IDC_AUTO_LA
/////////////////////////////////////////////////////////////////////////////
// COdlg dialog


COdlg::COdlg(CWnd* pParent /*=NULL*/)
        : CDialog(COdlg::IDD, pParent)
{
        //{{AFX_DATA_INIT(COdlg)
                // NOTE: the ClassWizard will add member initialization here
        //}}AFX_DATA_INIT
}


void COdlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(COdlg)
                // NOTE: the ClassWizard will add DDX and DDV calls here
        //}}AFX_DATA_MAP
}
```

```
    if (nSBCode == SB_ENDSCROLL) {
        ;
    } else if (pScrollBar->GetDlgCtrlID() == IDC_DLY_SLD) {
            if (((int)nPos != g_fill_delay) && (MIN_DLY <= nPos) && (nPos <= M
          SetDlgItemInt(IDC_DLY_EB,g_fill_delay = MAX_DLY + MIN_DLY - nPos);
        }
    } else if (pScrollBar->GetDlgCtrlID() == IDC_AUTO_SLD) {
            if (((int)nPos != g_place_delay) && (MIN_AUTO <= nPos) && (nPos <= MA
          SetDlgItemInt(IDC_AUTO_EB,(g_place_tpm = MAX_AUTO + MIN_AUTO - nPos));
          g_place_delay = 60000/g_place_tpm;
        }
    } else if (pScrollBar->GetDlgCtrlID() == IDC_POLL_SLD) {
            if (((int)nPos != g_poll_pps) && (MIN_POLL <= nPos) && (nPos <= MAX_P
          SetDlgItemInt(IDC_POLL_EB,(g_poll_pps = MAX_POLL + MIN_POLL - nPos));
          g_poll_delay = 1000/g_poll_pps;
        }
    }
         CDialog::OnVScroll(nSBCode, nPos, pScrollBar);
}

BOOL COdlg::OnInitDialog()
{
         CDialog::OnInitDialog();

   // GetParentFrame()->SetWindowText(m_inst + " Options");
    this->SetWindowText(m_inst + "  Options");


         SetDlgItemInt(IDC_DLYMAX,MAX_DLY);
         SetDlgItemInt(IDC_DLYMIN,MIN_DLY);

    SetDlgItemInt(IDC_AUTOMAX,MAX_AUTO);
         SetDlgItemInt(IDC_AUTOMIN,MIN_AUTO);

    SetDlgItemInt(IDC_POLLMAX,MAX_POLL);
         SetDlgItemInt(IDC_POLLMIN,MIN_POLL);

    SetDlgItemInt(IDC_DLY_EB,g_fill_delay);
    SetDlgItemInt(IDC_POLL_EB,g_poll_pps);
    SetDlgItemInt(IDC_AUTO_EB,g_place_tpm);

   // (CSliderCtrl *) xxx =  GetDlgItem(IDC_DLY_SLD);
    HWND hTrack = GetDlgItem(IDC_DLY_SLD)->m_hWnd;
    ::SendMessage(hTrack,TBM_SETRANGEMIN,TRUE,MIN_DLY);// MINDLY
    ::SendMessage(hTrack,TBM_SETRANGEMAX,TRUE,MAX_DLY);// MAXDLY
    ::SendMessage(hTrack,TBM_SETTICFREQ,100,TRUE); // 10 ticks   (MAXDLY - MINDLY
    ::SendMessage(hTrack,TBM_SETPOS,TRUE,MIN_DLY + MAX_DLY - g_fill_delay); // 1

     hTrack = GetDlgItem(IDC_AUTO_SLD)->m_hWnd;
    ::SendMessage(hTrack,TBM_SETRANGEMIN,TRUE,MIN_AUTO);// MINDLY
    ::SendMessage(hTrack,TBM_SETRANGEMAX,TRUE,MAX_AUTO);// MAXDLY
    ::SendMessage(hTrack,TBM_SETTICFREQ,500,TRUE); // 10 ticks   (MAXDLY - MINDLY
    ::SendMessage(hTrack,TBM_SETPOS,TRUE,MAX_AUTO + MIN_AUTO - g_place_tpm); //

     hTrack = GetDlgItem(IDC_POLL_SLD)->m_hWnd;
    ::SendMessage(hTrack,TBM_SETRANGEMIN,TRUE,MIN_POLL);// MINDLY
    ::SendMessage(hTrack,TBM_SETRANGEMAX,TRUE,MAX_POLL);// MAXDLY
    ::SendMessage(hTrack,TBM_SETTICFREQ,10,TRUE); // 10 ticks   (MAXDLY - MINDLY)
   ::SendMessage(hTrack,TBM_SETPOS,TRUE,MAX_POLL + MIN_POLL - g_poll_pps); //  1


   // Fonts
   int i = 0;
   while (ALL_TEXT_O[i])
      GetDlgItem(ALL_TEXT_O[i++])->SetFont(&g_text_font);



   return TRUE;  // return TRUE unless you set the focus to a control
                    // EXCEPTION: OCX Property Pages should return FALSE
}


void COdlg::OnUpdateAutoEb()
{
   int tpm = GetDlgItemInt(IDC_AUTO_EB,NULL,TRUE);
   if ((g_place_tpm != tpm) && (tpm >= MIN_AUTO) && (tpm <= MAX_AUTO)) {
      g_place_delay = 60000/tpm;
      g_place_tpm = tpm;
      HWND hTrack = GetDlgItem(IDC_AUTO_SLD)->m_hWnd;
      ::SendMessage(hTrack,TBM_SETPOS,TRUE,MAX_AUTO + MIN_AUTO - tpm);

   }
}
void COdlg::OnUpdatePollEb()
{
   int poll = GetDlgItemInt(IDC_POLL_EB,NULL,TRUE);
   if ((g_place_tpm != poll) && (poll >= MIN_AUTO) && (poll <= MAX_AUTO)) {
      g_poll_delay = 1000/poll;
      g_poll_pps = poll;
      HWND hTrack = GetDlgItem(IDC_POLL_SLD)->m_hWnd;
      ::SendMessage(hTrack,TBM_SETPOS,TRUE,MAX_POLL + MIN_POLL - poll);
   }

}
void COdlg::OnUpdateDlyEb()
{
   int dly = GetDlgItemInt(IDC_DLY_EB,NULL,TRUE);
   if ((g_fill_delay != dly) && (dly >= MIN_DLY) && (dly <= MAX_DLY)) {
      g_fill_delay = dly;
      HWND hTrack = GetDlgItem(IDC_DLY_SLD)->m_hWnd;
      ::SendMessage(hTrack,TBM_SETPOS,TRUE,MAX_DLY + MIN_DLY - dly);
   }

}


void COdlg::OnColor()
{
   CHOOSECOLOR cc;     // common dialog box structure
   COLORREF acrCustClr[16];

   // Setup the custom colors as a grey scale
   for (int v=0,i=0; i < 16; v=17 * i++)
      acrCustClr[i] = RGB(v,v,v);
   // Initialize the necessary members.
   cc.lStructSize = sizeof(CHOOSECOLOR);
   cc.hwndOwner = NULL; //  = hwnd;
   cc.lpCustColors = (LPDWORD) acrCustClr;
   cc.Flags = CC_FULLOPEN;  //  CC_PREVENTFULLOPEN


   if (ChooseColor(&cc)){
      g_new_color = cc.rgbResult; // lpCustColors
   } else {
      GetParentFrame()->SetMessageText("Color was not changed");
   }
}


void COdlg::OnClrstats()
{
        g_clear_stats++;

}

void COdlg::OnOracreate()
{
// Create the database in oracle
   int org_ora_state =  g_ora_state;


   if (g_ora_state == 0)
      if(oraconn(g_oracle_con_str))
         MessageBox("Oracle Connect Failed");
      else
         g_ora_state = 1;

   if (g_ora_state) {

      if(oracreate())
         MessageBox("Oracle oracreate Failed");

      if (org_ora_state == 0)
      if(oradisc())
         MessageBox("Oracle DisConnect Failed");
      else
         g_ora_state = 0;
   }
}
```

```
// Odlg.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// COdlg dialog

class COdlg : public CDialog
{
// Construction
public:
	COdlg(CWnd* pParent = NULL);	// standard constructor

// Dialog Data
	//{{AFX_DATA(COdlg)
	enum { IDD = IDD_O_DLG };
		// NOTE: the ClassWizard will add data members here
	//}}AFX_DATA
   CString m_inst;
//    COentryView* m_parentptr;

// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(COdlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);	// DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:

	// Generated message map functions
	//{{AFX_MSG(COdlg)
	afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
	afx_msg void OnUpdateAutoEb();
	afx_msg void OnVScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar);
	virtual BOOL OnInitDialog();
	afx_msg void OnUpdatePollEb();
	afx_msg void OnUpdateDlyEb();
	afx_msg void OnColor();
	virtual void OnOK();
	afx_msg void OnClrstats();
	afx_msg void OnOracreate();
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};
```

```
// oentrdoc.cpp : implementation of the COentryDoc class
//

#include "stdafx.h"
#include "oentry.h"
//#include "OpDlg.h"

#include "oentrdoc.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// COentryDoc

IMPLEMENT_DYNCREATE(COentryDoc, CDocument)

BEGIN_MESSAGE_MAP(COentryDoc, CDocument)
        //{{AFX_MSG_MAP(COentryDoc)
                // NOTE - the ClassWizard will add and remove mapping macros her
                //    DO NOT EDIT what you see in these blocks of generated code
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// COentryDoc construction/destruction

COentryDoc::COentryDoc()
{
        // TODO: add one-time construction code here

}

COentryDoc::~COentryDoc()
{
}

BOOL COentryDoc::OnNewDocument()
{
        if (!CDocument::OnNewDocument())
                return FALSE;

        // TODO: add reinitialization code here
        // (SDI documents will reuse this document)

        return TRUE;
}

/////////////////////////////////////////////////////////////////////////////
// COentryDoc serialization

void COentryDoc::Serialize(CArchive& ar)
{
        if (ar.IsStoring())
        {
                // TODO: add storing code here
        }
        else
        {
                // TODO: add loading code here
        }
}

////////////////////////////////////////////////////////////////////////////
// COentryDoc diagnostics

#ifdef _DEBUG
void COentryDoc::AssertValid() const
{
        CDocument::AssertValid();
}

void COentryDoc::Dump(CDumpContext& dc) const
{
        CDocument::Dump(dc);
}
#endif //_DEBUG

////////////////////////////////////////////////////////////////////////////
// COentryDoc commands
// oentrdoc.h : interface of the COentryDoc class
//
/////////////////////////////////////////////////////////////////////////////

class COentryDoc : public CDocument
{
protected: // create from serialization only
        COentryDoc();
        DECLARE_DYNCREATE(COentryDoc)

// Attributes
public:

// Operations
public:

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(COentryDoc)
        public:
        virtual BOOL OnNewDocument();
        //}}AFX_VIRTUAL

// Implementation
public:
        virtual ~COentryDoc();
        virtual void Serialize(CArchive& ar);   // overridden for document i/o
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
        //{{AFX_MSG(COentryDoc)
                // NOTE - the ClassWizard will add and remove member functions h
                //    DO NOT EDIT what you see in these blocks of generated code
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////
```

```
// oentry.cpp : Defines the class behaviors for the application.
//

#include "stdafx.h"
#include "oentry.h"

#include "mainfrm.h"
#include "oentrdoc.h"
#include "dbdlg.h"
#include "Odlg.h"
#include "oentrvw.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// COentryApp

BEGIN_MESSAGE_MAP(COentryApp, CWinApp)
        //{{AFX_MSG_MAP(COentryApp)
        ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
                // NOTE - the ClassWizard will add and remove mapping macros her
                //    DO NOT EDIT what you see in these blocks of generated code
        //}}AFX_MSG_MAP
        // Standard file based document commands
        ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
        ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
        // Standard print setup command
        ON_COMMAND(ID_FILE_PRINT_SETUP, CWinApp::OnFilePrintSetup)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// COentryApp construction

COentryApp::COentryApp()
{
        // TODO: add construction code here,
        // Place all significant initialization in InitInstance
}

/////////////////////////////////////////////////////////////////////////////
// The one and only COentryApp object

COentryApp theApp;

/////////////////////////////////////////////////////////////////////////////
// COentryApp initialization

BOOL COentryApp::InitInstance()
{
        // Standard initialization
        // If you are not using these features and wish to reduce the size
        //  of your final executable, you should remove from the following
        //  the specific initialization routines you do not need.

        Enable3dControls();

        LoadStdProfileSettings();  // Load standard INI file options (including
        // Register the application's document templates.  Document templates
        //  serve as the connection between documents, frame windows and views.


        CSingleDocTemplate* pDocTemplate;
        pDocTemplate = new CSingleDocTemplate(
                IDR_MAINFRAME,
                RUNTIME_CLASS(COentryDoc),
                RUNTIME_CLASS(CMainFrame),       // main SDI frame window
                RUNTIME_CLASS(COentryView));
        AddDocTemplate(pDocTemplate);

        // create a new (empty) document

     OnFileNew();

        if (m_lpCmdLine[0] != '\0')
        {
                // TODO: add command line processing here
        }

        return TRUE;
}

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
        CAboutDlg();

// Dialog Data
        //{{AFX_DATA(CAboutDlg)
        enum { IDD = IDD_ABOUTBOX };
        //}}AFX_DATA


      CFont m_title_font;


// Implementation
protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        //{{AFX_MSG(CAboutDlg)
        virtual BOOL OnInitDialog();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
        //{{AFX_DATA_INIT(CAboutDlg)
        //}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CAboutDlg)
        //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
        //{{AFX_MSG_MAP(CAboutDlg)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

// App command to run the dialog
void COentryApp::OnAppAbout()
{
        CAboutDlg aboutDlg;
        aboutDlg.DoModal();
}

/////////////////////////////////////////////////////////////////////////////
// COentryApp commands

BOOL CAboutDlg::OnInitDialog()
{
        CDialog::OnInitDialog();



   LOGFONT lf;
   memset(&lf,0,sizeof(LOGFONT));
        strcpy(lf.lfFaceName,"Monotype Corsiva");
   lf.lfHeight = 24;
   m_title_font.CreateFontIndirect(&lf);
        GetDlgItem(IDC_ABOUT1)->SetFont(&m_title_font);



        return TRUE;  // return TRUE unless you set the focus to a control
                      // EXCEPTION: OCX Property Pages should return FALSE
}
```

```
// oentry.h : main header file for the OENTRY application
//
#define MIN_DLY  0
#define MAX_DLY  5000

#define MIN_AUTO 2
#define MAX_AUTO 6000

#define MIN_POLL 1
#define MAX_POLL 100

#ifndef __AFXWIN_H__
        #error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"       // main symbols

/////////////////////////////////////////////////////////////////////////////
// COentryApp:
// See oentry.cpp for the implementation of this class
//

class COentryApp : public CWinApp
{
public:
        COentryApp();

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(COentryApp)
        public:
        virtual BOOL InitInstance();
        //}}AFX_VIRTUAL

// Implementation

        //{{AFX_MSG(COentryApp)
        afx_msg void OnAppAbout();
                // NOTE - the ClassWizard will add and remove member functions h
                //    DO NOT EDIT what you see in these blocks of generated code
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};


/////////////////////////////////////////////////////////////////////////////
```

```
// OPDLG.cpp : implementation file
//

#include "stdafx.h"
#include "oentry.h"
#include "OPDLG.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

extern int g_options_run;


/////////////////////////////////////////////////////////////////////////////
// OPDLG dialog


OPDLG::OPDLG(CWnd* pParent /*=NULL*/)
        : CDialog(OPDLG::IDD_O_DLG, pParent)
{
        //{{AFX_DATA_INIT(OPDLG)
                // NOTE: the ClassWizard will add member initialization here
        //}}AFX_DATA_INIT
}


void OPDLG::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(OPDLG)
                // NOTE: the ClassWizard will add DDX and DDV calls here
        //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(OPDLG, CDialog)
        //{{AFX_MSG_MAP(OPDLG)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// OPDLG message handlers

void OPDLG::OnOK()
{
        g_options_run = 40;  // Re-enable the options call button
        this->DestroyWindow();
        CDialog::OnOK();
}
```

```
User: root
Host: bunny
Class: bunny
Job: stdin
```

```
// OPDLG.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// OPDLG dialog

class OPDLG : public CDialog
{
// Construction
public:
        OPDLG(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
        //{{AFX_DATA(OPDLG)
//      enum { IDD = IDD_OP_DLG };
                // NOTE: the ClassWizard will add data members here
        //}}AFX_DATA


// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(OPDLG)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        //}}AFX_VIRTUAL

// Implementation
protected:

        // Generated message map functions
        //{{AFX_MSG(OPDLG)
        virtual void OnOK();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};
```

```
User: root
Host: bunny
Class: bunny
Job: stdin
```

```
// OptDlg.cpp : implementation file
//

#include "stdafx.h"
#include "oentry.h"
#include "OptDlg.h"


#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif


extern   int  g_options_run ;


/////////////////////////////////////////////////////////////////////////////
// OptDlg dialog


OptDlg::OptDlg(CWnd* pParent /*=NULL*/)
        : CDialog(OptDlg::IDD, pParent)
{
        //{{AFX_DATA_INIT(OptDlg)
                // NOTE: the ClassWizard will add member initialization here
        //}}AFX_DATA_INIT
}


void OptDlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(OptDlg)
                // NOTE: the ClassWizard will add DDX and DDV calls here
        //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(OptDlg, CDialog)
        //{{AFX_MSG_MAP(OptDlg)
        ON_BN_CLICKED(IDC_DONE, OnDone)
        ON_WM_DESTROY()
        ON_WM_CREATE()
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// OptDlg message handlers

void OptDlg::OnDone()
{
        g_options_run = 40;  // Re-enable the options call button
        this->DestroyWindow();
}

void OptDlg::OnDestroy()
{
   g_options_run = 40;  // Re-enable the options call button
```

```
        CDialog::OnDestroy();
}

int OptDlg::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
        if (CDialog::OnCreate(lpCreateStruct) == -1)
                return -1;

   g_options_run = 1;  // Start

        return 0;
}
```

```
// OptDlg.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// OptDlg dialog

class OptDlg : public CDialog
{
// Construction
public:
        OptDlg(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
        //{{AFX_DATA(OptDlg)
//      enum { IDD = IDD_OPTIONS_DLG };
        //}}AFX_DATA


// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(OptDlg)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        //}}AFX_VIRTUAL

// Implementation
protected:

        // Generated message map functions
        //{{AFX_MSG(OptDlg)
        afx_msg void OnDone();
        afx_msg void OnDestroy();
        afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};
```

```
//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.
// Used by oentry.rc
//
#define IDD_ABOUTBOX                    100
#define IDD_OENTRY_FORM                 101
#define IDR_MAINFRAME                   128
#define IDR_OENTRYTYPE                  129
#define IDI_ICON_Q                      130
#define IDB_BITMAPTEST                  131
#define IDD_DBDLG                       133
#define IDD_O_DLG                       138
#define IDR_3DIMES                      145
#define IDR_3DMDS                       146
#define IDI_ICON_TRASH                  147
#define IDI_ICON_TRASH1                 148
#define IDI_WAIT0                       149
#define IDI_WAIT1                       150
#define IDI_WAIT2                       151
#define IDI_WAIT3                       152
#define IDI_FILL0                       153
#define IDI_FILL1                       154
#define IDI_FILL2                       155
#define IDI_WAIT4                       156
#define IDI_WAIT5                       157
#define IDI_WAIT6                       158
#define IDC_AUTOB                       1000
#define IDC_EXITB                       1001
#define IDC_ORDERB                      1002
#define IDC_CUST                        1003
#define IDC_ITEM                        1004
#define IDC_QTY                         1005
#define IDC_PICT                        1006
#define IDC_COLORB                      1006
#define IDC_GENERIC1                    1007
#define IDC_TPS_EB                      1008
#define IDC_TRANB                       1010
#define IDC_COLORBOX                    1011
#define IDC_LOGO_Q                      1012
#define IDC_ABORTB                      1013
#define IDC_COMMITB                     1014
#define IDC_TOTALR                      1015
#define IDC_MSGS                        1015
#define IDC_SHOWDB                      1016
#define IDC_TPS                         1017
#define IDC_QUE                         1018
#define IDC_DLYMIN                      1019
#define IDC_DLYMAX                      1020
#define IDC_PLACER                      1021
#define IDC_AUTOMAX                     1021
#define IDC_PLACENOQR                   1022
#define IDC_AUTOMIN                     1022
#define IDC_QUELAB                      1023
#define IDC_POLLMAX                     1023
#define IDC_CUSTLAB                     1024
#define IDC_POLLMIN                     1024
#define IDC_ITEMLAB                     1025
#define IDC_QTYLAB                      1026
#define IDC_SENDREPC                    1028
#define IDC_RECIPT                      1029
```

```
#define IDC_FILLDB                    1030
#define IDC_RECIPTS                   1031
#define IDC_MSGS_LAB                  1032
#define IDC_RECIPTS_LAB               1033
#define IDC_FILLR3                    1034
#define IDC_DB_I0                     1038
#define IDC_DB_Q0                     1039
#define IDC_DB_SALEST                 1040
#define IDC_DB_SALES                  1041
#define IDC_DB_BOX                    1042
#define IDC_DB_REFILL                 1043
#define IDC_ORDERS_ITEMS              1044
#define IDC_T11                       1045
#define IDC_T12                       1046
#define IDC_T13                       1047
#define IDC_T21                       1048
#define IDC_T22                       1049
#define IDC_DB_P0                     1050
#define IDC_T23                       1051
#define IDC_ORDERBOX                  1052
#define IDC_FILLTXT                   1054
#define IDC_AUTOBOX                   1055
#define IDC_dmd                       1057
#define IDC_POLL_BOX                  1057
#define IDC_TRANBOX                   1058
#define IDC_WAIT0                     1060
#define IDC_WAIT1                     1061
#define IDC_OPTIONSB                  1063
#define IDC_FILL0                     1064
#define IDC_FILL1                     1065
#define IDC_DB_I1                     1066
#define IDC_FILL2                     1066
#define IDC_DB_Q1                     1067
#define IDC_DB_P1                     1068
#define IDC_DLY_EB                    1068
#define IDC_DB_I2                     1069
#define IDC_DLY_SLD                   1069
#define IDC_DB_Q2                     1070
#define IDC_DLY_LAB                   1070
#define IDC_DB_P2                     1071
#define IDC_AUTO_EB                   1071
#define IDC_DB_I3                     1072
#define IDC_AUTO_SLD                  1072
#define IDC_DB_Q3                     1073
#define IDC_AUTO_LAB                  1073
#define IDC_DB_P3                     1074
#define IDC_FILLBOX                   1074
#define IDC_DB_I4                     1075
#define IDC_COLOR                     1075
#define IDC_DB_Q4                     1076
#define IDC_POLL_EB                   1076
#define IDC_DB_P4                     1077
#define IDC_POLL_SLD                  1077
#define IDC_DB_I5                     1078
#define IDC_POLL_LAB                  1078
#define IDC_DB_Q5                     1079
#define IDC_DB_P5                     1080
#define IDC_CLRSTATS                  1080
#define IDC_ABOUT1                    1081
#define IDC_ORACREATE                 1081
```

```
#define IDC_BIG_TITLE                     1082
#define IDC_MODEBOX                       1083
#define IDC_STATBOX                       1084
#define IDC_FILLR                         2000
#define IDC_TOTALS                        2001

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_3D_CONTROLS                       1
#define _APS_NEXT_RESOURCE_VALUE          140
#define _APS_NEXT_COMMAND_VALUE           32771
#define _APS_NEXT_CONTROL_VALUE           1085
#define _APS_NEXT_SYMED_VALUE             101
#endif
#endif
```

```
// stdafx.cpp : source file that includes just the standard includes
//      oentry.pch will be the pre-compiled header
//      stdafx.obj will contain the pre-compiled type information

#include "stdafx.h"
```

```
// stdafx.h : include file for standard system include files,
//  or project specific include files that are used frequently, but
//      are changed infrequently
//

#include <afxwin.h>         // MFC core and standard components
#include <afxext.h>         // MFC extensions
```

```
// admindlg.cpp : implementation file
//


#include "stdafx.h"
#include "qman.h"
#include "admindlg.h"

#define Q_LIB
#include "qlib.h"
#include "qadmin.h"
#include "rt.h"
#define  ADMTIMER 102
extern lpSMBUFH   sm_base;
extern QADMSTATS  g_s[3];
extern CString    g_que[3];
extern lpQHANDLE  QS[3];
int  IDC_APICS[8] = {IDC_QNONE1,IDC_QDOWN1,IDC_QSTOP1,IDC_QNOPUT1,IDC_QNOGET1,ID
enum pics {QNONE,QDOWN,QSTOP,QNOPUT,QNOGET,QUP,QNOPG,QFULL};
extern int  g_pic[3+3+3];

QADMCTLS    g_ad;

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif


extern CFont g_text_font;
int ALL_TEXT_A[] = {IDC_STATS,IDC_OGROUP,IDC_SGROUP,
                    IDC_PUTC,IDC_GETC,IDC_HALTC,IDC_SRESETC,IDC_FRESETC,IDC_SHU
                    IDC_MAXSIZE,IDC_LIMLAB,IDC_QSIZE,
                    IDOK,IDC_SET,IDC_REFRESH,0};




/////////////////////////////////////////////////////////////////////////////
// CAdminDlg dialog


CAdminDlg::CAdminDlg(CWnd* pParent /*=NULL*/)
        : CDialog(CAdminDlg::IDD, pParent)
{
        //{{AFX_DATA_INIT(CAdminDlg)
                // NOTE: the ClassWizard will add member initialization here
        //}}AFX_DATA_INIT
}


void CAdminDlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CAdminDlg)
                // NOTE: the ClassWizard will add DDX and DDV calls here
        //}}AFX_DATA_MAP
}
```

```
BEGIN_MESSAGE_MAP(CAdminDlg, CDialog)
        //{{AFX_MSG_MAP(CAdminDlg)
        ON_BN_CLICKED(IDC_REFRESH, OnRefresh)
        ON_BN_CLICKED(IDC_SET, OnSet)
        ON_CBN_EDITCHANGE(IDC_QSIZE, OnEditchangeQsize)
        ON_WM_TIMER()
        ON_WM_RBUTTONDOWN()
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()


/////////////////////////////////////////////////////////////////////////////
// CAdminDlg message handlers
void CAdminDlg::SetDisplay(int init){
        CString sp,sg,s,t1,t2;

   if (init) {
      t1 = g_que[m_id] + " Status";
      t2 = g_que[m_id] + " Settings";

        SetDlgItemText(IDC_OGROUP,t1);
        SetDlgItemText(IDC_SGROUP,t2);
   }


   CListBox* lb = (CListBox*) GetDlgItem(IDC_STATS);
   lb->ResetContent();

   s.Format("%5d committed entries",g_s[m_id].committed_entries); lb->InsertStri
   s.Format("%5d uncommitted puts", g_s[m_id].pending_puts);      lb->InsertStri
   s.Format("%5d uncommitted gets", g_s[m_id].pending_gets);      lb->InsertS[illegible])
   s.Format("%5d holes",            g_s[m_id].holey_entries);     lb->InsertStri
   s.Format("%5d max entries",      g_s[m_id].max_entries);       lb->InsertStri

   sg = ctime(&g_s[m_id].first_start_time);
   sp = ctime(&g_s[m_id].last_restart_time);

   s.Format("%5d restarts",g_s[m_id].num_restarts);           lb->InsertString(-1,
   s.Format("Last restart time %s",LPCTSTR(sg.Left(24)));     lb->InsertString(-1,
   s.Format("First restart time %s",LPCTSTR(sp.Left(24)));    lb->InsertString(-1,


   if (init) {
      s.Format("(%d limit)",g_s[m_id].max_entries_limit);
      SetDlgItemText(IDC_LIMLAB,LPCTSTR(s));
      SetDlgItemInt(IDC_QSIZE,g_s[m_id].max_entries);
   }

   // Select a icon
//   if (init)
//      for (int i=0;i<6;i++)
//         if (i != g_pic[m_id])
//            GetDlgItem(IDC_APICS[i])->ShowWindow(SW_HIDE);


   if ((g_pic[m_id] != g_pic[m_id+3+3])) {
      GetDlgItem(IDC_APICS[g_pic[m_id]])->ShowWindow(SW_SHOW);
      GetDlgItem(IDC_APICS[g_pic[m_id+3+3]])->ShowWindow(SW_HIDE);
      g_pic[m_id+3+3] = g_pic[m_id];
   }
   // Set check box items
   if (init) {
      ((CButton *) GetDlgItem(IDC_GETC))->SetCheck(g_s[m_id].qget_state);
      ((CButton *) GetDlgItem(IDC_PUTC))->SetCheck(g_s[m_id].qput_state);
      ((CButton *) GetDlgItem(IDC_SRESETC))->SetCheck(g_ad.stats_reset_flag);
      ((CButton *) GetDlgItem(IDC_FRESETC))->SetCheck(g_ad.full_reset_flag);
      ((CButton *) GetDlgItem(IDC_HALTC))->SetCheck(g_ad.halt_flag);
   }
}

BOOL CAdminDlg::OnInitDialog()
{
       CDialog::OnInitDialog();

   g_pic[m_id+3+3] = QNONE; // history is invalid
   for (int i=0;i<6;i++)    // Turn all pics off.
      GetDlgItem(IDC_APICS[i])->ShowWindow(SW_HIDE);
   SetDisplay(1);

   SetTimer(ADMTIMER,1000,NULL); // 1 sec


   i = 0;
   while (ALL_TEXT_A[i])
      GetDlgItem(ALL_TEXT_A[i++])->SetFont(&g_text_font);




   return TRUE;  // return TRUE unless you set the focus to a control
                 // EXCEPTION: OCX Property Pages should return FALSE
}


void CAdminDlg::OnRefresh()
{
   SetDisplay(1);
}


DWORD SetCtl(LPVOID m_id)
{
int id = (int)m_id;

   QsendAndReceive(QS[id],ADMINREQ_MODE,QADM_SET_CONTROLS, 0,sizeof(g_ad),(char
   return(0);
}


void CAdminDlg::OnSet()
{
   memset(&g_ad,0,sizeof(g_ad));

   if (IsDlgButtonChecked(IDC_PUTC))    g_ad.enable_qputs_flag++;
   if (IsDlgButtonChecked(IDC_GETC))    g_ad.enable_qgets_flag++;
   if (IsDlgButtonChecked(IDC_SRESETC)) g_ad.stats_reset_flag++;
   if (IsDlgButtonChecked(IDC_FRESETC)) g_ad.full_reset_flag++;
   if (IsDlgButtonChecked(IDC_SHUTDOWNC)) g_ad.shutdown_flag++;
   if (IsDlgButtonChecked(IDC_HALTC))   g_ad.halt_flag++;

   int qs = GetDlgItemInt(IDC_QSIZE,NULL,TRUE);
   if ((qs>0)&&(qs<=g_s[m_id].max_entries_limit))
      g_ad.max_entries_value = qs;
      DWORD id;

      CreateThread(NULL,0,(LPTHREAD_START_ROUTINE) SetCtl,(LPVOID) m_id,0,&id);
  //  QsendAndReceive(QS[m_id],ADMINREQ_MODE,QADM_SET_CONTROLS, 0,sizeof(g_ad),(
   if (g_ad.shutdown_flag) {
      g_pic[m_id] = QDOWN;
      SetDisplay(0); // 0=refresh only.
   }
   Sleep(1000);
   SetDisplay(0); // 0=refresh only.
   Sleep(1000);

   if (g_ad.shutdown_flag) CDialog::OnOK(); // Exit
   g_ad.stats_reset_flag = 0;
   g_ad.full_reset_flag = 0;
   g_ad.shutdown_flag = 0;
   g_ad.halt_flag = 0;

   SetDisplay(1); // 1=init: set buttons
}

void CAdminDlg::OnEditchangeQsize()
{
   int qs = GetDlgItemInt(IDC_QSIZE,NULL,TRUE);
   if (!((qs>0)&&(qs<=g_s[m_id].max_entries_limit)))
      SetDlgItemText(IDC_QSIZE,"");
}


void CAdminDlg::OnTimer(UINT nIDEvent)
{
         if (nIDEvent == ADMTIMER)
   SetDisplay(0);
         else
         CDialog::OnTimer(nIDEvent);
}

void CAdminDlg::OnRButtonDown(UINT nFlags, CPoint point)
{
   GetParentFrame()->SetMessageText("");
   this->Invalidate();


         CDialog::OnRButtonDown(nFlags, point);
}
```

```
// admindlg.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CAdminDlg dialog

class CAdminDlg : public CDialog
{
// Construction
public:
        CAdminDlg(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
        //{{AFX_DATA(CAdminDlg)
        enum { IDD = IDD_ADMINDIALOG };
                // NOTE: the ClassWizard will add data members here
        //}}AFX_DATA
   int m_id;

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CAdminDlg)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        //}}AFX_VIRTUAL
   void SetDisplay(int i);

// Implementation
protected:

        // Generated message map functions
        //{{AFX_MSG(CAdminDlg)
        virtual BOOL OnInitDialog();
        afx_msg void OnRefresh();
        afx_msg void OnSet();
        afx_msg void OnEditchangeQsize();
        afx_msg void OnTimer(UINT nIDEvent);
        afx_msg void OnRButtonDown(UINT nFlags, CPoint point);
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};
```

```
// datadlg.cpp : implementation file
//

#include "stdafx.h"
#include "qman.h"
#include "datadlg.h"
#include "KeySearch.h"

#define Q_LIB
#include "qlib.h"
#include "qadmin.h"
#include "rt.h"
#include "orderfm.h"

extern lpQHANDLE  QS[3];
extern QADMSTATS  g_s[3];
extern CString    g_que[3];

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CdataDlg dialog


CdataDlg::CdataDlg(CWnd* pParent /*=NULL*/)
        : CDialog(CdataDlg::IDD, pParent)
{
        //{{AFX_DATA_INIT(CdataDlg)
                // NOTE: the ClassWizard will add member initialization here
        //}}AFX_DATA_INIT
}


void CdataDlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CdataDlg)
                // NOTE: the ClassWizard will add DDX and DDV calls here
        //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CdataDlg, CDialog)
        //{{AFX_MSG_MAP(CdataDlg)
        ON_BN_CLICKED(IDREFRESHB, OnRefreshb)
        ON_BN_CLICKED(IDC_CMT, OnCmt)
        ON_BN_CLICKED(IDC_UNCMT, OnUncmt)
        ON_BN_CLICKED(IDR_SEARCHB, OnSearchb)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()



        SMBUF m;

void CdataDlg::ScreenInit() {
   CString s,ss;
   int   i,gotsize;
   pOFORM po = (pOFORM) &m.mdata;

   if (m_sub_mode == QADM_REQ_COM_DATA)
      s.Format("%s   %4d Commited %20d Uncommited entries",LPCTSTR(g_que[m_id
            g_s[m_id].committed_entries,  g_s[m_id].pending_gets + g_s[m_id].
   else
      s.Format("%s   %4d Uncommited %20d Commited entries",LPCTSTR(g_que[m_id]),
        g_s[m_id].pending_gets + g_s[m_id].pending_puts, g_s[m_id].committed_ent
   SetDlgItemText(IDC_TITLE,LPCTSTR(s));

   CListBox* lb = (CListBox*) GetDlgItem(IDC_DATAL);
   lb->ResetContent();

   for (i = 0; i < 200; i++) {
      *m.mdata = i;
      if (QSUCCESS == QsendAndReceive(QS[m_id],ADMINREQ_MODE,m_sub_mode,
         0,sizeof(int),m.mdata, sizeof(m.mdata),m.mdata,&gotsize,&m.msgh))

      ss = "%3d\t%s\t%s\t%s";
      if (gotsize == sizeof(OFORM)) {
         s.Format(LPCTSTR(ss),i,po->cust,po->item,po->qty,0);
         lb->InsertString(-1,s);
      } else if (gotsize) {
         s.Format("%3d -\t%s",i,m.mdata);
         lb->InsertString(-1,s);
      } else {
         break;
      }
   }
}


#define FORMATNAME            "c:\\q\\formats.txt"

char * IsAddr(char *c){
   int off = 0;
   if (strstr(c,"msgh")){
      if (strstr(c,"size")) return((char *)m.msgh.size);
   } else if (strstr(c,"mdata")) {
      sscanf(c,"%*[^[]\[%d",off);
      return((m.mdata + off)) ;
   }
   return(0);
}


typedef struct fmts {  // Thread parameters
   void*   testa;
   int     testv;
   char    fmt[100];
   void*   a[10];
   fmts*   next;
} FMTS, *pFMTS;

void clearfmt( pFMTS p){
   p->testa = NULL;
   p->testv = 0;
  *p->fmt = 0;
   for (int i=0;i < 10;i++) p->a[i] = NULL;
   p->next = 0;
}


FMTS fmts;

void ParseFormats(){
   char line[LINESIZE];
   char test[LINESIZE];
   char format[LINESIZE];
   char ops[LINESIZE];
   char op1[LINESIZE];
   char op2[LINESIZE];
   char op[LINESIZE];
   FILE *fp = fopen(FORMATNAME,"r");

   if ( ! fp ) {
         //  Say("Cant open the data formats file %s",FORMATNAME);
           return;
         }

   while (fgets(line,LINESIZE,fp)) {

      if ( 3 == sscanf(line,"%[^;];%[^;];%[^;]",test,format,ops)) {
         if (strchr(test,'#')) continue;
         if ( 3 != sscanf(test,"%s %s %s",op1,op,op2)) continue;
         if (strstr(op1,"size")) fmts.testa = &m.msgh.size;
         sscanf(op2,"%d",&fmts.testv);
         strcpy(fmts.fmt,format);

      } else {
         ; //Say("ReadParms: Ignoring: %s",line);
      }
   }
   if (ferror(fp)){
     // Fail("read error in parameters file %s",PARMNAME);
      clearerr(fp);
   }
   fclose(fp);
}

/////////////////////////////////////////////////////////////////////////////
// CdataDlg message handlers

BOOL CdataDlg::OnInitDialog()
{
        CDialog::OnInitDialog();
        m_sub_mode = QADM_REQ_COM_DATA;
   ((CButton *) GetDlgItem(IDC_CMT))->SetCheck(TRUE);

   ScreenInit();

        return TRUE;  // return TRUE unless you set the focus to a control
                      // EXCEPTION: OCX Property Pages should return FALSE
}

void CdataDlg::OnRefreshb()
{
   ScreenInit();
}
```

```
void CdataDlg::OnCmt()
{
        m_sub_mode = QADM_REQ_COM_DATA;
   ScreenInit();

}

void CdataDlg::OnUncmt()
{
        m_sub_mode = QADM_REQ_UNCOM_DATA;
   ScreenInit();

}

void CdataDlg::OnSearchb()
{
   CKeySearch d;
   d.m_id = m_id;
   d.DoModal();
}
```

```
// datadlg.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CdataDlg dialog

class CdataDlg : public CDialog
{
// Construction
public:
        CdataDlg(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
        //{{AFX_DATA(CdataDlg)
        enum { IDD = IDD_DATADIALOG };
                // NOTE: the ClassWizard will add data members here
        //}}AFX_DATA
   int m_id;
   int m_sub_mode;

   void ScreenInit();
// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CdataDlg)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        //}}AFX_VIRTUAL

// Implementation
protected:

        // Generated message map functions
        //{{AFX_MSG(CdataDlg)
        virtual BOOL OnInitDialog();
        afx_msg void OnRefreshb();
        afx_msg void OnCmt();
        afx_msg void OnUncmt();
        afx_msg void OnSearchb();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};
```

```
// KeySearch.h : header file
//


/////////////////////////////////////////////////////////////////////////////
// CKeySearch dialog

class CKeySearch : public CDialog
{
// Construction
public:
        CKeySearch(CWnd* pParent = NULL);   // standard constructor
   //void CM_switch(int mode);
// Dialog Data
        //{{AFX_DATA(CKeySearch)
        enum { IDD = IDD_KEYSEARCH };
                // NOTE: the ClassWizard will add data members here
        //}}AFX_DATA
   //int   m_min_int,m_max_int,m_at_int;
   //CString m_min_str,m_max_str,m_at_str;
   int m_id; // 1-3
   int m_preds;
   int m_pred_type[3];
   int m_pic;
   int m_committed;
   int m_uncommitted;
   int m_total_entries;
   int m_search_type;

   void OnCompChange(int pred);
   void ChangePredView(int pred, int act);
   void CopyInput(int pred, int min, int max);


// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CKeySearch)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        //}}AFX_VIRTUAL

// Implementation
protected:

        // Generated message map functions
        //{{AFX_MSG(CKeySearch)
        virtual BOOL OnInitDialog();
        afx_msg void OnSearchb();
        afx_msg void OnAnd1();
        afx_msg void OnAnd2();
        afx_msg void OnSelchangeCompCb1();
        afx_msg void OnSelchangeCompCb2();
        afx_msg void OnSelchangeCompCb3();
        afx_msg void OnEditchangeMinCb1();
        afx_msg void OnEditchangeMinCb2();
        afx_msg void OnEditchangeMinCb3();
        afx_msg void OnEditchangeMaxCb1();
        afx_msg void OnEditchangeMaxCb2();
        afx_msg void OnEditchangeMaxCb3();
        afx_msg void OnTimer(UINT nIDEvent);
        afx_msg void OnAllR();
        afx_msg void OnComR();
        afx_msg void OnUncomR();
        afx_msg void OnRButtonDown(UINT nFlags, CPoint point);
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};
```

```
// mainfrm.cpp : implementation of the CMainFrame class
//

#include "stdafx.h"
#include "qman.h"

#include "mainfrm.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CMainFrame

IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
        //{{AFX_MSG_MAP(CMainFrame)
                // NOTE - the ClassWizard will add and remove mapping macros her
                //    DO NOT EDIT what you see in these blocks of generated code
        ON_WM_CREATE()
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// arrays of IDs used to initialize control bars

// toolbar buttons - IDs are command buttons
static UINT BASED_CODE buttons[] =
{
        // same order as in the bitmap 'toolbar.bmp'
        ID_FILE_NEW,
        ID_FILE_OPEN,
        ID_FILE_SAVE,
                ID_SEPARATOR,
        ID_EDIT_CUT,
        ID_EDIT_COPY,
        ID_EDIT_PASTE,
                ID_SEPARATOR,
        ID_FILE_PRINT,
        ID_APP_ABOUT,
};

static UINT BASED_CODE indicators[] =
{
        ID_SEPARATOR,           // status line indicator
        ID_INDICATOR_CAPS,
        ID_INDICATOR_NUM,
        ID_INDICATOR_SCRL,
};

/////////////////////////////////////////////////////////////////////////////
// CMainFrame construction/destruction

CMainFrame::CMainFrame()
{
        // TODO: add member initialization code here
}

CMainFrame::~CMainFrame()
{
}

int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
        if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
                return -1;

        if (!m_wndToolBar.Create(this) ||
                !m_wndToolBar.LoadBitmap(IDR_MAINFRAME) ||
                !m_wndToolBar.SetButtons(buttons,
                  sizeof(buttons)/sizeof(UINT)))
        {
                TRACE0("Failed to create toolbar\n");
                return -1;      // fail to create
        }

/* Derek's remove tool bar */
    m_wndToolBar.ShowWindow(SW_HIDE);

        if (!m_wndStatusBar.Create(this) ||
                !m_wndStatusBar.SetIndicators(indicators,
                  sizeof(indicators)/sizeof(UINT)))
        {
                TRACE0("Failed to create status bar\n");
                return -1;      // fail to create
        }

        // TODO: Delete these three lines if you don't want the toolbar to
        //  be dockable
        m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
        EnableDocking(CBRS_ALIGN_ANY);
        DockControlBar(&m_wndToolBar);

        // TODO: Remove this if you don't want tool tips
        m_wndToolBar.SetBarStyle(m_wndToolBar.GetBarStyle() |
                CBRS_TOOLTIPS | CBRS_FLYBY);

        return 0;
}

/////////////////////////////////////////////////////////////////////////////
// CMainFrame diagnostics

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
        CFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{
        CFrameWnd::Dump(dc);
}

#endif //_DEBUG
```

```
/////////////////////////////////////////////////////////////////////////////
// CMainFrame message handlers
// mainfrm.h : interface of the CMainFrame class
//
/////////////////////////////////////////////////////////////////////////////

class CMainFrame : public CFrameWnd
{
protected: // create from serialization only
        CMainFrame();
        DECLARE_DYNCREATE(CMainFrame)

// Attributes
public:

// Operations
public:

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CMainFrame)
        //}}AFX_VIRTUAL

// Implementation
public:
        virtual ~CMainFrame();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

protected:  // control bar embedded members
        CStatusBar  m_wndStatusBar;
        CToolBar    m_wndToolBar;

// Generated message map functions
protected:
        //{{AFX_MSG(CMainFrame)
        afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
                // NOTE - the ClassWizard will add and remove member functions h
                //    DO NOT EDIT what you see in these blocks of generated code
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////
```

```
User: root
Host: bunny
Class: bunny
Job: stdin
```

```
// KeySearch.cpp : implementation file
//

#include "stdafx.h"
#include "qman.h"
#include "KeySearch.h"
#define Q_LIB
#include "qlib.h"
#include "qadmin.h"
#include "orderfm.h"
#include "rt.h"


#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

#define KEYTIMER 106
extern lpQHANDLE  QS[3];
extern QADMSTATS  g_s[3];
extern CString    g_que[3];

extern   lpSMBUFH sm_base;
extern int  g_pic[3+3+3];
//////////////////////////////////////////////////////////////////////////////
#define ID_OBJS   9

int    IDC_KPICS[8] = {IDC_QNONE1,IDC_QDOWN1,IDC_QSTOP1,IDC_QNOPUT1,IDC_QNOGET1 I
enum   IDTYPE   {COMP,MIN,AT,MAX,AND,OPA,OPB,OPC,OPD};
int    g_id[3][ID_OBJS] = {{IDC_COMP_CB1,IDC_MIN_CB1,IDC_AT_CB1,IDC_MAX_CB1,IDC_A
                           {IDC_COMP_CB2,IDC_MIN_CB2,IDC_AT_CB2,IDC_MAX_CB2,IDC_A
                           {IDC_COMP_CB3,IDC_MIN_CB3,IDC_AT_CB3,IDC_MAX_CB3,IDC_A

// List of items to get big fonts
extern CFont g_text_font;
int ALL_TEXT_S[] = {IDC_ALL_R,IDC_COM_R,IDC_UNCOM_R,IDC_MODE,IDOK,IDSEARCHB,IDC_

//////////////////////////////////////////////////////////////////////////////
// CKeySearch dialog


CKeySearch::CKeySearch(CWnd* pParent /*=NULL*/)
        : CDialog(CKeySearch::IDD, pParent)
{
        //{{AFX_DATA_INIT(CKeySearch)
                // NOTE: the ClassWizard will add member initialization here
        //}}AFX_DATA_INIT
}


void CKeySearch::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CKeySearch)
                // NOTE: the ClassWizard will add DDX and DDV calls here
        //}}AFX_DATA_MAP
}
```

```
BEGIN_MESSAGE_MAP(CKeySearch, CDialog)
        //{{AFX_MSG_MAP(CKeySearch)
        ON_BN_CLICKED(IDSEARCHB, OnSearchb)
        ON_BN_CLICKED(IDC_AND1, OnAnd1)
        ON_BN_CLICKED(IDC_AND2, OnAnd2)
        ON_CBN_SELCHANGE(IDC_COMP_CB1, OnSelchangeCompCb1)
        ON_CBN_SELCHANGE(IDC_COMP_CB2, OnSelchangeCompCb2)
        ON_CBN_SELCHANGE(IDC_COMP_CB3, OnSelchangeCompCb3)
        ON_CBN_EDITCHANGE(IDC_MIN_CB1, OnEditchangeMinCb1)
        ON_CBN_EDITCHANGE(IDC_MIN_CB2, OnEditchangeMinCb2)
        ON_CBN_EDITCHANGE(IDC_MIN_CB3, OnEditchangeMinCb3)
        ON_CBN_EDITCHANGE(IDC_MAX_CB1, OnEditchangeMaxCb1)
        ON_CBN_EDITCHANGE(IDC_MAX_CB2, OnEditchangeMaxCb2)
        ON_CBN_EDITCHANGE(IDC_MAX_CB3, OnEditchangeMaxCb3)
        ON_WM_TIMER()
        ON_BN_CLICKED(IDC_ALL_R, OnAllR)
        ON_BN_CLICKED(IDC_COM_R, OnComR)
        ON_BN_CLICKED(IDC_UNCOM_R, OnUncomR)
        ON_WM_RBUTTONDOWN()
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CKeySearch message handlers

BOOL CKeySearch::OnInitDialog()
{
        CDialog::OnInitDialog();
        int i,id;

   m_preds = 1;
   m_pred_type[0] = m_pred_type[1] = m_pred_type[2] =  INT_SEARCH_TYPE;

   for (id = 1; id < OPA; id++)
      GetDlgItem(g_id[0][id])->EnableWindow(FALSE);

   m_pic = (g_pic[m_id] + 1) % 8;

   ((CButton *) GetDlgItem(IDC_ALL_R))->SetCheck(TRUE);
   m_search_type = SEARCH_ALL_ENT;

   m_committed = -1;
   m_uncommitted = -1;
   m_total_entries = -1;

   SetDlgItemText(IDC_MODE,"View " + g_que[m_id] + " Entries" );

   SetTimer(KEYTIMER,1000,NULL); // 4 sec


   i = 0;
   while (ALL_TEXT_S[i])
      GetDlgItem(ALL_TEXT_S[i++])->SetFont(&g_text_font);


        return TRUE;  // return TRUE unless you set the focus to a control
                      // EXCEPTION: OCX Property Pages should return FALSE
}
```

```
int String2Time(const char *s){  // Thu Nov 30 17:30:00 1995
   char mon[80];
   int  mday,hr,mn,sc,yr,args;
   tm TM;
   // If it starts with a day skip it.
   sscanf(s,"%s",mon);
   if (strstr("Mon Tue Wed Thu Fri Sat Sun",mon)) s = s + 4;

   args = sscanf(s,"%s %d %d:%d:%d %d",mon,&mday,&hr,&mn,&sc,&yr);
   if      (!strcmp(mon,"Jan")) TM.tm_mon = 0;
   else if (!strcmp(mon,"Feb")) TM.tm_mon = 1;
   else if (!strcmp(mon,"Mar")) TM.tm_mon = 2;
   else if (!strcmp(mon,"Apr")) TM.tm_mon = 3;
   else if (!strcmp(mon,"May")) TM.tm_mon = 4;
   else if (!strcmp(mon,"Jun")) TM.tm_mon = 5;
   else if (!strcmp(mon,"Jul")) TM.tm_mon = 6;
   else if (!strcmp(mon,"Aug")) TM.tm_mon = 7;
   else if (!strcmp(mon,"Sep")) TM.tm_mon = 8;
   else if (!strcmp(mon,"Oct")) TM.tm_mon = 9;
   else if (!strcmp(mon,"Nov")) TM.tm_mon = 10;
   else if (!strcmp(mon,"Dec")) TM.tm_mon = 11;
   else  TM.tm_mon = 12;

   if ((args == 6) && (TM.tm_mon != 12)) {
      TM.tm_sec = sc;
      TM.tm_min = mn;
      TM.tm_hour = hr;
      TM.tm_mday = mday;
      TM.tm_year = yr - 1900;
      TM.tm_isdst = -1;
      return(mktime(&TM));
   } else
   return(-1);

}
void Time2String(int time,char *s, int printday){
   if (printday)
      strcpy(s,ctime((time_t*)(&time)));
   else
      strcpy(s,(ctime((time_t*)(&time)) + 4)  ); // dont print day
   s[strlen(s) - 1] = 0;
}

void CKeySearch::OnCompChange(int pred) {
   CString type_str;
   GetDlgItemText(g_id[pred][COMP],type_str);
   int comptype = INT_SEARCH_TYPE;
   if (type_str == "String")  comptype = STR_SEARCH_TYPE;
   if (type_str == "Short")   comptype = SHORT_SEARCH_TYPE;
   m_pred_type[pred] = comptype;

   CComboBox * CBat  = (CComboBox *) this->GetDlgItem(g_id[pred][AT]);
   CComboBox * CBmin = (CComboBox *) this->GetDlgItem(g_id[pred][MIN]);
   CComboBox * CBmax = (CComboBox *) this->GetDlgItem(g_id[pred][MAX]);

   CBat->ResetContent();
   CBmin->ResetContent();
   CBmax->ResetContent();
   CBmin->AddString("");
   CBmin->AddString("");
   CBmin->AddString("ANY");
   CBmax->AddString("");
   CBmax->AddString("");
   CBmax->AddString("ANY");

   if (type_str == "Time") {
      char tim[80];
      Time2String(SHAREDATA(time),tim,0);
      CBat->AddString("TIME");

      CBmin->SetCurSel(-1);
      CBmin->SetWindowText(tim);
//    CBmin->ReplaceSel(tim);


      CBmin->AddString(tim);  //  add to edit box too?
      CBmax->AddString(tim);  //  add to edit box too?
   } else if (comptype == STR_SEARCH_TYPE) {
      CBat->AddString("CUSTOMER");
      CBat->AddString("ITEM");
      CBat->AddString("to_server");
      CBat->AddString("to_logical");
   } else if (comptype == INT_SEARCH_TYPE) {
      CBat->AddString("UID");
      CBat->AddString("TID");
      CBat->AddString("HOST");
      CBat->AddString("QUANTITY");
      lpRT rt = RTROOT;
      while (rt = NextRT(rt)) {
         CBmin->AddString(RT_NODE(rt));
         CBmax->AddString(RT_NODE(rt));
      }
   } else if (comptype == SHORT_SEARCH_TYPE) {
      CBat->AddString("MODE");
      CBat->AddString("SUB_MODE");
   }

   for (int id = 1; id < ID_OBJS; id++)
      GetDlgItem(g_id[pred][id])->EnableWindow(TRUE);
   GetDlgItem(IDSEARCHB)->EnableWindow(TRUE);

  // GetParentFrame()->SetMessageText(""); cedit
   SetDlgItemText(IDC_SERSTAT,"");

   ((CComboBox *) this->GetDlgItem(g_id[pred][AT]))->SetCurSel(0);
   ((CComboBox *) this->GetDlgItem(g_id[pred][MAX]))->SetCurSel(0);
   ((CComboBox *) this->GetDlgItem(g_id[pred][MIN]))->SetCurSel(0);

}

void CKeySearch::OnSelchangeCompCb1() {OnCompChange(0);}
void CKeySearch::OnSelchangeCompCb2() {OnCompChange(1);}
void CKeySearch::OnSelchangeCompCb3() {OnCompChange(2);}

void CKeySearch::ChangePredView(int pred, int act)
{
   int id;

   for (id = 0; id < ID_OBJS; id++)
```

```
      GetDlgItem(g_id[pred][id])->ShowWindow(act);
   for (id = 1; id < OPA; id++)
      GetDlgItem(g_id[pred][id])->EnableWindow(FALSE);
   if (act == SW_HIDE)
      for  (id = 0; id < AND; id++)
         ((CComboBox *) this->GetDlgItem(g_id[pred][id]))->SetCurSel(-1);

   if (act == SW_HIDE ) { // Get rid of the ghost of the combo box  MFC bug?
      this->Invalidate();
      this->UpdateWindow();
   }
}




void CKeySearch::OnAnd1()
{
         if (m_preds == 1) {
      ChangePredView(1,SW_SHOW);
      m_preds = 2;
   } else {
      ChangePredView(1,SW_HIDE);
      ChangePredView(2,SW_HIDE);
      m_preds = 1;
   }
}

void CKeySearch::OnAnd2()
{
         if (m_preds == 2) {
      ChangePredView(2,SW_SHOW);
      m_preds = 3;
   } else {
      ChangePredView(2,SW_HIDE);
      m_preds = 2;
   }
}

void CKeySearch::CopyInput(int pred, int from, int to)
{
   CString s; int pos;
   GetDlgItemText(g_id[pred][from],s.GetBuffer(100),100);
   s.ReleaseBuffer();

   pos = ((CComboBox *) this->GetDlgItem(g_id[pred][to]))->GetCurSel();
   ((CComboBox *) this->GetDlgItem(g_id[pred][to]))->DeleteString(0);
   ((CComboBox *) this->GetDlgItem(g_id[pred][to]))->InsertString(0,s);
   ((CComboBox *) this->GetDlgItem(g_id[pred][from]))->DeleteString(1);
   ((CComboBox *) this->GetDlgItem(g_id[pred][from]))->InsertString(1,s);
   if (pos == 0)
      ((CComboBox *) this->GetDlgItem(g_id[pred][to]))->SetCurSel(0);

}

void CKeySearch::OnEditchangeMinCb1() {CopyInput(0,MIN,MAX);}
void CKeySearch::OnEditchangeMinCb2() {CopyInput(1,MIN,MAX);}
void CKeySearch::OnEditchangeMinCb3() {CopyInput(2,MIN,MAX);}
void CKeySearch::OnEditchangeMaxCb1() {CopyInput(0,MAX,MIN);}
void CKeySearch::OnEditchangeMaxCb2() {CopyInput(1,MAX,MIN);}
void CKeySearch::OnEditchangeMaxCb3() {CopyInput(2,MAX,MIN);}

void CKeySearch::OnSearchb()
{
   SMBUF b,B;
   QADMSEL key;
   OFORM order;

   CString ss,s,at_str,min_str, max_str;
   int i,at_int, sz, matches;

   GetDlgItem(IDSEARCHB)->EnableWindow(FALSE);
   CListBox* lb = (CListBox*) GetDlgItem(IDC_SEARCH_LB);
   lb->ResetContent();
   SetDlgItemText(IDC_SERSTAT,"");
   // lb->SetTabStops(100);

   key.num_preds = m_preds;
   key.search_type = m_search_type; // SEARCH_ALL_ENT
   for (int p = 0; p < m_preds; p++) {
      // Set min and max values
      min_str = "";
      max_str = "";
      GetDlgItemText(g_id[p][MIN],min_str.GetBuffer(100),100);
      GetDlgItemText(g_id[p][MAX],max_str.GetBuffer(100),100);
      GetDlgItemText(g_id[p][AT],at_str.GetBuffer(100),100);
      min_str.ReleaseBuffer();
      max_str.ReleaseBuffer();
      at_str.ReleaseBuffer();
      at_int = GetDlgItemInt(g_id[p][AT],NULL,TRUE);

      strcpy(key.preds[p].min_str_val,LPCTSTR(min_str));
      strcpy(key.preds[p].max_str_val,LPCTSTR(max_str));
      key.preds[p].min_int_val = GetDlgItemInt(g_id[p][MIN],NULL,TRUE); // min i
      key.preds[p].max_int_val = GetDlgItemInt(g_id[p][MAX],NULL,TRUE); // max i
      key.preds[p].min_sh_val  = key.preds[p].min_int_val; // min short
      key.preds[p].max_sh_val  = key.preds[p].max_int_val; // max short


      if (min_str == "ANY") {
         key.preds[p].min_switch = 0;
         key.preds[p].min_str_len = 0;
      } else {
         key.preds[p].min_switch = 1;
         key.preds[p].min_str_len = strlen(key.preds[p].min_str_val);
      }

      if (max_str == "ANY") {
         key.preds[p].max_switch = 0;
         key.preds[p].max_str_len = 0;
      } else {
         key.preds[p].max_switch = 1;
         key.preds[p].max_str_len = strlen(key.preds[p].max_str_val);
      }

      if (at_str == "TIME") {
         char tim[80];
         if (max_str == "NOW")
```

```
            key.preds[p].max_int_val = SHAREDATA(time);
         if ((i = String2Time(LPCTSTR(max_str))) != -1)
            key.preds[p].max_int_val = i;
         if (min_str == "NOW")
            key.preds[p].min_int_val = SHAREDATA(time);
         if ((i = String2Time(LPCTSTR(min_str))) != -1)
            key.preds[p].min_int_val = i;

       // Time2String(key.preds[p].min_int_val,tim,1);
       // s.Format("Time value:%s",tim);
       // SetDlgItemText(IDC_SERSTAT,s);
      }

      if (at_str == "HOST") {
         if (i = Name2IP(LPCTSTR(max_str)))
            key.preds[p].max_int_val = i;
         if (i = Name2IP(LPCTSTR(min_str)))
            key.preds[p].min_int_val = i;
      }

      // Find the offset
      if (at_str == "TIME")
          at_int = ((char *) &b.msgh.time)        - ((char *) &b.msgh);

      if (at_str == "MODE")
          at_int = ((char *) &b.msgh.mode)        - ((char *) &b.msgh);

      if (at_str == "SUB_MODE")
          at_int = ((char *) &b.msgh.sub_mode)    - ((char *) &b.msgh);

      if (at_str == "UID")
          at_int = ((char *) &b.msgh.mid.uid)     - ((char *) &b.msgh);

      if (at_str == "TID")
          at_int = ((char *) &b.msgh.mid.tid)     - ((char *) &b.msgh);

      if (at_str == "HOST")
          at_int = ((char *) &b.msgh.mid.host)    - ((char *) &b.msgh);

      if (at_str == "SIZE")
          at_int = ((char *) &b.msgh.size)        - ((char *) &b.msgh);

      if (at_str == "to_server")
          at_int = ((char *) &b.msgh.to_server)   - ((char *) &b.msgh);

      if (at_str == "to_logical")
          at_int = ((char *) &b.msgh.to_logical) - ((char *) &b.msgh);

      if (at_str == "CUSTOMER")
          at_int = sizeof(MSGH) + ((char *) &order.cust) - ((char *) &order);

      if (at_str == "ITEM")
          at_int = sizeof(MSGH) + ((char *) &order.item) - ((char *) &order);

      if (at_str == "QUANTITY")
          at_int = sizeof(MSGH) + ((char *) &order.qty)  - ((char *) &order);


      key.preds[p].offset = at_int;
      key.preds[p].pred_type = m_pred_type[p];
```

```
}


// Get the list of key matches
if (QSUCCESS == QsendAndReceive(QS[m_id],ADMINREQ_MODE,QADM_REQ_SEL_DATA,
   0,sizeof(key),(char *) &key, sizeof(b.mdata),b.mdata,&sz,&b.msgh)) {
   lpMID md = (lpMID) b.mdata;

   matches = sz / sizeof(MID);
   if (matches > 0) {
//          char * h = IP2Name(md->host);
      for (i = 0; i < matches ; i++) {
         ss.Format("%s(%d,%d)", IP2Name(md->host),md->uid,md->tid);
         s.Format("%4d \t%s",i+1,ss );
         lb->InsertString(-1,s);
         md = (lpMID)((char *)md + sizeof(MID));
      }
   } else {
      lb->InsertString(-1,"No messages");
   }

   lb->UpdateWindow();
   //Sleep(200);

   md = (lpMID) b.mdata;
   lpOFORM po = (lpOFORM)B.mdata;
   for (i = 0; i < matches ; i++) {
      memcpy(B.mdata,md,sizeof(MID)); // Copy one mid

      s.Format("%d Messages: Reading %d",matches,i);
      SetDlgItemText(IDC_SERSTAT,s);
      GetDlgItem(IDC_SERSTAT)->UpdateWindow();

      if (QSUCCESS == QsendAndReceive(QS[m_id],ADMINREQ_MODE,QADM_REQ_MSG,
         0,sizeof(mid),B.mdata,
         sizeof(B.mdata),B.mdata,&sz,&B.msgh)) {

         ss.Format("%s(%d,%d)", IP2Name(md->host),md->uid,md->tid);
         if (B.msgh.size == sizeof(OFORM))
            s.Format("%4d \t%s\t%-8s\t%2d %-10s",i + 1,ss,po->cust,po->qty,po
         else if (sz)
            s.Format("%4d \t%s\t%s",i + 1,ss,b.mdata);
         else
            s.Format("%4d \t%s\t<<Empty>>",i + 1,ss);

         lb->DeleteString(i);
         lb->InsertString(i,s);
      } else {
         s.Format("%d Error requesting %s(%d,%d)",i + 1,IP2Name(md->host),md-
         SetDlgItemText(IDC_SERSTAT,s);
         break;  // Stop here
      }
      md = (lpMID)((char *)md + sizeof(MID));
   }

} else
  s.Format("Error - no reply") ;

if (!strstr(LPCTSTR(s),"Error")) s.Format("%d Matches:",matches);
```

```
   SetDlgItemText(IDC_SERSTAT,s);


   Sleep(500);
   GetDlgItem(IDSEARCHB)->EnableWindow(TRUE);
}



void CKeySearch::OnTimer(UINT nIDEvent)
{
        if (nIDEvent == KEYTIMER) {
      int  com,uncom;
      CString s;

      if ((g_pic[m_id] != m_pic)) {
         GetDlgItem(IDC_KPICS[g_pic[m_id]])->ShowWindow(SW_SHOW);
         GetDlgItem(IDC_KPICS[m_pic])->ShowWindow(SW_HIDE);
         m_pic = g_pic[m_id];

         // Set Title
         this->SetWindowText("View " + g_que[m_id] );
      }

      if (m_committed != (com = g_s[m_id].committed_entries)) {
         s.Format("%3d Committed entries.", (m_committed = com));
         SetDlgItemText(IDC_COM_R,s);
      }
      if (m_uncommitted != (uncom = g_s[m_id].pending_gets + g_s[m_id].pending_p
         s.Format("%3d Uncommitted entries.",(m_uncommitted = uncom));
         SetDlgItemText(IDC_UNCOM_R,s);
      }
      if (m_total_entries != (com + uncom)) {
         s.Format("All %d entries.",      (m_total_entries = com + uncom));
         SetDlgItemText(IDC_ALL_R,s);
      }
   } else
        CDialog::OnTimer(nIDEvent);
}


void CKeySearch::OnAllR()
{ m_search_type = SEARCH_ALL_ENT;}
void CKeySearch::OnComR()
{ m_search_type = SEARCH_COM_ENT;}
void CKeySearch::OnUncomR()
{ m_search_type = SEARCH_UNCOM_ENT;}

void CKeySearch::OnRButtonDown(UINT nFlags, CPoint point)
{
   GetParentFrame()->SetMessageText("");
   this->Invalidate();


        CDialog::OnRButtonDown(nFlags, point);
}
```

```
// qman.cpp : Defines the class behaviors for the application.
//

#include "stdafx.h"
#include "qman.h"

#include "mainfrm.h"
#include "qmandoc.h"
#include "qmanview.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CQmanApp

BEGIN_MESSAGE_MAP(CQmanApp, CWinApp)
        //{{AFX_MSG_MAP(CQmanApp)
        ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
                // NOTE - the ClassWizard will add and remove mapping macros her
                //    DO NOT EDIT what you see in these blocks of generated code
        //}}AFX_MSG_MAP
        // Standard file based document commands
        ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
        ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
        // Standard print setup command
        ON_COMMAND(ID_FILE_PRINT_SETUP, CWinApp::OnFilePrintSetup)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CQmanApp construction

CQmanApp::CQmanApp()
{
        // TODO: add construction code here,
        // Place all significant initialization in InitInstance
}

/////////////////////////////////////////////////////////////////////////////
// The one and only CQmanApp object

CQmanApp theApp;

/////////////////////////////////////////////////////////////////////////////
// CQmanApp initialization

BOOL CQmanApp::InitInstance()
{
        // Standard initialization
        // If you are not using these features and wish to reduce the size
        //  of your final executable, you should remove from the following
        //  the specific initialization routines you do not need.

        Enable3dControls();

        LoadStdProfileSettings();  // Load standard INI file options (including

        // Register the application's document templates.  Document templates
        //  serve as the connection between documents, frame windows and views.

        CSingleDocTemplate* pDocTemplate;
        pDocTemplate = new CSingleDocTemplate(
                IDR_MAINFRAME,
                RUNTIME_CLASS(CQmanDoc),
                RUNTIME_CLASS(CMainFrame),       // main SDI frame window
                RUNTIME_CLASS(CQmanView));
        AddDocTemplate(pDocTemplate);

        // create a new (empty) document
        OnFileNew();

        if (m_lpCmdLine[0] != '\0')
        {
                // TODO: add command line processing here
        }

        return TRUE;
}

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
        CAboutDlg();

// Dialog Data
        //{{AFX_DATA(CAboutDlg)
        enum { IDD = IDD_ABOUTBOX };
        //}}AFX_DATA


   CFont m_title_font;
// Implementation
protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        //{{AFX_MSG(CAboutDlg)
        virtual BOOL OnInitDialog();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
        //{{AFX_DATA_INIT(CAboutDlg)
        //}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CAboutDlg)
        //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
        //{{AFX_MSG_MAP(CAboutDlg)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

// App command to run the dialog
void CQmanApp::OnAppAbout()
{
        CAboutDlg aboutDlg;
        aboutDlg.DoModal();
}

/////////////////////////////////////////////////////////////////////////////
// CQmanApp commands

BOOL CAboutDlg::OnInitDialog()
{
        CDialog::OnInitDialog();


   LOGFONT lf;
   memset(&lf,0,sizeof(LOGFONT));
        strcpy(lf.lfFaceName,"Monotype Corsiva");
   lf.lfHeight = 24;
   m_title_font.CreateFontIndirect(&lf);
        GetDlgItem(IDC_ABOUT1)->SetFont(&m_title_font);


        // TODO: Add extra initialization here

        return TRUE;  // return TRUE unless you set the focus to a control
                      // EXCEPTION: OCX Property Pages should return FALSE
}
```

```
// qman.h : main header file for the QMAN application
//

#ifndef __AFXWIN_H__
	#error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"		// main symbols

/////////////////////////////////////////////////////////////////////////////
// CQmanApp:
// See qman.cpp for the implementation of this class
//

class CQmanApp : public CWinApp
{
public:
	CQmanApp();

// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CQmanApp)
	public:
	virtual BOOL InitInstance();
	//}}AFX_VIRTUAL

// Implementation

	//{{AFX_MSG(CQmanApp)
	afx_msg void OnAppAbout();
		// NOTE - the ClassWizard will add and remove member functior
		//    DO NOT EDIT what you see in these blocks of generated coue
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////
```

```
// qmandoc.cpp : implementation of the CQmanDoc class
//

#include "stdafx.h"
#include "qman.h"

#include "qmandoc.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CQmanDoc

IMPLEMENT_DYNCREATE(CQmanDoc, CDocument)

BEGIN_MESSAGE_MAP(CQmanDoc, CDocument)
	//{{AFX_MSG_MAP(CQmanDoc)
		// NOTE - the ClassWizard will add and remove mapping macros her
		//    DO NOT EDIT what you see in these blocks of generated code
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CQmanDoc construction/destruction

CQmanDoc::CQmanDoc()
{
	// TODO: add one-time construction code here

}

CQmanDoc::~CQmanDoc()
{
}

BOOL CQmanDoc::OnNewDocument()
{
	if (!CDocument::OnNewDocument())
		return FALSE;

	// TODO: add reinitialization code here
	// (SDI documents will reuse this document)

	return TRUE;
}

/////////////////////////////////////////////////////////////////////////////
// CQmanDoc serialization

void CQmanDoc::Serialize(CArchive& ar)
{
	if (ar.IsStoring())
	{
		// TODO: add storing code here
	}
	else
	{
```

```
                // TODO: add loading code here
        }
}

/////////////////////////////////////////////////////////////////////////////
// CQmanDoc diagnostics

#ifdef _DEBUG
void CQmanDoc::AssertValid() const
{
        CDocument::AssertValid();
}

void CQmanDoc::Dump(CDumpContext& dc) const
{
        CDocument::Dump(dc);
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CQmanDoc commands
// qmandoc.h : interface of the CQmanDoc class
//
/////////////////////////////////////////////////////////////////////////////

class CQmanDoc : public CDocument
{
protected: // create from serialization only
        CQmanDoc();
        DECLARE_DYNCREATE(CQmanDoc)

// Attributes
public:

// Operations
public:

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CQmanDoc)
        public:
        virtual BOOL OnNewDocument();
        //}}AFX_VIRTUAL

// Implementation
public:
        virtual ~CQmanDoc();
        virtual void Serialize(CArchive& ar);   // overridden for document i/o
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
        //{{AFX_MSG(CQmanDoc)
                // NOTE - the ClassWizard will add and remove member functions h
                //    DO NOT EDIT what you see in these blocks of generated code
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////
```

```
// qmanview.h : interface of the CQmanView class
//
/////////////////////////////////////////////////////////////////////////////

class CQmanView : public CFormView
{
protected: // create from serialization only
        CQmanView();
        DECLARE_DYNCREATE(CQmanView)

public:
        //{{AFX_DATA(CQmanView)
        enum{ IDD = IDD_QMAN_FORM };
                // NOTE: the ClassWizard will add data members here
        //}}AFX_DATA

   // Attributes
   int m_maxtrack;
   CString m_que[3];

public:
        CQmanDoc* GetDocument();

// Operations
public:

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CQmanView)
        public:
        virtual void OnInitialUpdate();
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
        virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
        virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);
        virtual void OnPrint(CDC* pDC, CPrintInfo*);
        virtual void OnDraw(CDC* pDC);
        //}}AFX_VIRTUAL
   //void InitTrackBar(HWND hTrack, int IDMIN, int TMIN, int IDMAX, int TMAX, in
   //DWORD Poll();
   //DWORD CQmanView::Poll();
   void CQmanView::OpenQue(int i, int IDC_QUES);

   // meter painting
   void DrawKey();
   void GetMeterBoxes();
   void TestMeters();
   void CQmanView::DrawMeter(int i);
   void CQmanView::CmdLine(int pass);
   void CQmanView::DrawKeyColor(int ID, int HS, COLORREF COL);

   // font
   CFont m_title_font;

// Implementation
public:
        virtual ~CQmanView();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
        //{{AFX_MSG(CQmanView)
        afx_msg void OnExit();
        afx_msg void OnHScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar);
        afx_msg void OnSelchangeQues1();
        afx_msg void OnSelchangeQues2();
        afx_msg void OnSelchangeQues3();
        afx_msg void OnAdminb1();
        afx_msg void OnAdminb2();
        afx_msg void OnAdminb3();
        afx_msg void OnDatab1();
        afx_msg void OnDatab2();
        afx_msg void OnDatab3();
        afx_msg void OnTimer(UINT nIDEvent);
        afx_msg void OnRButtonDown(UINT nFlags, CPoint point);
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

#ifndef _DEBUG  // debug version in qmanview.cpp
inline CQmanDoc* CQmanView::GetDocument()
   { return (CQmanDoc*)m_pDocument; }
#endif

/////////////////////////////////////////////////////////////////////////////
```

```
User: root
Host: bunny
Class: bunny
Job: stdin
```

```
// qmanview.cpp : implementation of the CQmanView class
//


#include "stdafx.h"
#include "qman.h"

#include "qmandoc.h"
#include "qmanview.h"
#include "admindlg.h"
//#include "datadlg.h"
#include "KeySearch.h"

 //+++++++++++++ QLIB ++++++++++++++++++
#include "qlib.h"
#include "qadmin.h"
#include "rt.h"



#define  MYTIMER 100
#define  TITLETIMER 101

#define  MIN_POLL_DLY 100
#define  MAX_POLL_DLY 5000


extern   lpSMBUFH sm_base;
lpQHANDLE   QS[3] = {NULL,NULL,NULL};
QADMSTATS   g_s[3+3]; // 3 Current + 3 Old

//HWND g_track[3];
HWND g_tmin[3];
HWND g_tmax[3];
HWND g_tlab[3];
CString g_que[]   = {"","",""};

int  ALL_TEXT[] = {IDC_TMIN1,IDC_TMIN2,IDC_TMIN3,
                   IDC_TMAX1,IDC_TMAX2,IDC_TMAX3,
                   IDC_TLAB1,IDC_TLAB2,IDC_TLAB3,
                   IDC_ADMINB1,IDC_ADMINB2,IDC_ADMINB3,
                   IDC_DATAB1,IDC_DATAB2,IDC_DATAB3,
                   IDC_QUES1,IDC_QUES2,IDC_QUES3,
                   IDC_EXIT,0,0,0};


int  IDC_TMINS[]  = {IDC_TMIN1,IDC_TMIN2,IDC_TMIN3};
int  IDC_TMAXS[]  = {IDC_TMAX1,IDC_TMAX2,IDC_TMAX3};
int  IDC_TLABS[]  = {IDC_TLAB1,IDC_TLAB2,IDC_TLAB3};
int  IDC_METERS[] = {IDC_METER1,IDC_METER2,IDC_METER3};
int  IDC_PICS[3][8] = {{IDC_QNONE1,IDC_QDOWN1,IDC_QSTOP1,IDC_QNOPUT1,IDC_QNOGET1
                       {IDC_QNONE4,IDC_QDOWN4,IDC_QSTOP4,IDC_QNOPUT4,IDC_QNOGET4
                       {IDC_QNONE5,IDC_QDOWN5,IDC_QSTOP5,IDC_QNOPUT5,IDC_QNOGET5
enum pics {QNONE,QDOWN,QSTOP,QNOPUT,QNOGET,QUP,QNOPG,QFULL};
int  g_pic[3+3+3];
int  g_poll;
int  g_poll_delay = 2000;
CFont g_text_font;
```

```
typedef struct met {  // Thread parameters
   CRect    b_rec;
   CRect    c_rec;
   CRect    p_rec;
   CRect    g_rec;
   CRect    h_rec;
   CRect    f_rec;
   int      commit,pendp,pendg,hole,free,min,max;
} MET, *pMET;

MET g_met[3+4];



//+++++++++++++ QLIB ++++++++++++++++++


#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CQmanView

IMPLEMENT_DYNCREATE(CQmanView, CFormView)

BEGIN_MESSAGE_MAP(CQmanView, CFormView)
        //{{AFX_MSG_MAP(CQmanView)
        ON_BN_CLICKED(IDC_EXIT, OnExit)
        ON_WM_HSCROLL()
        ON_CBN_SELCHANGE(IDC_QUES1, OnSelchangeQues1)
        ON_CBN_SELCHANGE(IDC_QUES2, OnSelchangeQues2)
        ON_CBN_SELCHANGE(IDC_QUES3, OnSelchangeQues3)
        ON_BN_CLICKED(IDC_ADMINB1, OnAdminb1)
        ON_BN_CLICKED(IDC_ADMINB2, OnAdminb2)
        ON_BN_CLICKED(IDC_ADMINB3, OnAdminb3)
        ON_BN_CLICKED(IDC_DATAB1, OnDatab1)
        ON_BN_CLICKED(IDC_DATAB2, OnDatab2)
        ON_BN_CLICKED(IDC_DATAB3, OnDatab3)
        ON_WM_TIMER()
        ON_WM_RBUTTONDOWN()
        //}}AFX_MSG_MAP
        // Standard printing commands
        ON_COMMAND(ID_FILE_PRINT, CFormView::OnFilePrint)
        ON_COMMAND(ID_FILE_PRINT_PREVIEW, CFormView::OnFilePrintPreview)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CQmanView construction/destruction

CQmanView::CQmanView()
        : CFormView(CQmanView::IDD)
{
        //{{AFX_DATA_INIT(CQmanView)
                // NOTE: the ClassWizard will add member initialization here
        //}}AFX_DATA_INIT
        // TODO: add construction code here
}
```

```
CQmanView::~CQmanView()
{
}

void CQmanView::DoDataExchange(CDataExchange* pDX)
{
        CFormView::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CQmanView)
                // NOTE: the ClassWizard will add DDX and DDV calls here
        //}}AFX_DATA_MAP
}

/////////////////////////////////////////////////////////////////////////////
// CQmanView printing

BOOL CQmanView::OnPreparePrinting(CPrintInfo* pInfo)
{
        // default preparation
        return DoPreparePrinting(pInfo);
}

void CQmanView::OnBeginPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
        // TODO: add extra initialization before printing
}

void CQmanView::OnEndPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
        // TODO: add cleanup after printing
}

void CQmanView::OnPrint(CDC* pDC, CPrintInfo*)
{
        // TODO: add code to print the controls
}

/////////////////////////////////////////////////////////////////////////////
// CQmanView diagnostics

#ifdef _DEBUG
void CQmanView::AssertValid() const
{
        CFormView::AssertValid();
}

void CQmanView::Dump(CDumpContext& dc) const
{
        CFormView::Dump(dc);
}

CQmanDoc* CQmanView::GetDocument() // non-debug version is inline
{
        ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CQmanDoc)));
        return (CQmanDoc*)m_pDocument;
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CQmanView message handlers
```

```
void CQmanView::OnExit()
{
   g_poll = 0;   // Stop the threads
   AfxGetMainWnd()->DestroyWindow();
}


// METER METER METER METER METER METER METER METER METER METER METER METER
// METER METER METER METER METER METER METER METER METER METER METER METER
// METER METER METER METER METER METER METER METER METER METER METER METER
//
//                    b_rec
// +---------+---------+---------+---------+
// | Committ | Pending | holes   | Free    |
// |         |         |         |         |
// +---------+---------+---------+---------+
//  c_rec      p_rec      h_rec    f_rec


void CQmanView::GetMeterBoxes(){
   for (int i=0;i<3;i++){
        GetDlgItem(IDC_METERS[i])->GetWindowRect(&g_met[i].b_rec);
      // Adjust coordantates for 0,0,1,b
      g_met[i].b_rec.right  -= g_met[i].b_rec.left;
      g_met[i].b_rec.bottom -= g_met[i].b_rec.top;
      g_met[i].b_rec.left    = g_met[i].b_rec.top = 0;

      g_met[i].c_rec.left    = 0; // Commit always starts at 0,0
      g_met[i].f_rec.right   = g_met[i].b_rec.right; // Free always ends at max

      g_met[i].c_rec.top     =
      g_met[i].p_rec.top     =
      g_met[i].g_rec.top     =
      g_met[i].h_rec.top     =
      g_met[i].f_rec.top     = g_met[i].b_rec.top; // 0

      g_met[i].c_rec.bottom  =
      g_met[i].p_rec.bottom  =
      g_met[i].g_rec.bottom  =
      g_met[i].h_rec.bottom  =
      g_met[i].f_rec.bottom  = g_met[i].b_rec.bottom; // All the same hight


   }
}


void CQmanView::DrawKeyColor(int ID, int HS, COLORREF COL){
   CRect    c_rec;
   CBrush *pCBrush;

   if ( HS == -1 )
      pCBrush = new CBrush(COL);
   else
      pCBrush = new CBrush(HS,COL);

   GetDlgItem(ID)->GetWindowRect(&c_rec);

   // Adjust coordantates for 0,0,1,b
      c_rec.right  -= c_rec.left;
      c_rec.bottom -= c_rec.top;
      c_rec.left    = c_rec.top = 0;

      // The DC for the meter
   CDC* pCOLORDC = GetDlgItem(ID)->GetDC();

   // Select this brush, save the old
   CBrush *pOldBrush =
   pCOLORDC->SelectObject(pCBrush);
   pCOLORDC->Rectangle(c_rec);

   GetDlgItem(ID)->Invalidate();
   delete(pCBrush);
   ReleaseDC(pCOLORDC);
}


void CQmanView::DrawKey(){

   DrawKeyColor(IDC_KEY_COM,     -1,            RGB(127,255,255));
   DrawKeyColor(IDC_KEY_PENPUT, HS_BDIAGONAL, RGB(000,182,255));
   DrawKeyColor(IDC_KEY_PENGET, HS_FDIAGONAL, RGB(000,182,255));
   DrawKeyColor(IDC_KEY_HOLE,   HS_DIAGCROSS, RGB(255,128,128));
}



#define MAXPIXELS g_met[i].b_rec.right
void CQmanView::TestMeters(){
   int c,pg,pp,h; // Pixel width of each rectangle

   for (int i=0;i<3;i++) { // If any change
      if((g_met[i].commit  != g_s[i].committed_entries) ||
         (g_met[i].pendg   != g_s[i].pending_gets ) ||
         (g_met[i].pendp   != g_s[i].pending_puts) ||
         (g_met[i].hole    != g_s[i].holey_entries ) ||
         (g_met[i].free    != g_s[i].num_free_entries) ||
         (g_met[i].max     != g_s[i].max_entries) ) {

         if (!g_s[i].max_entries) g_s[i].max_entries=1; // No divide by zero

         // Update old values
         g_met[i].commit  = g_s[i].committed_entries;
         g_met[i].pendg   = g_s[i].pending_gets;
         g_met[i].pendp   = g_s[i].pending_puts;
         g_met[i].hole    = g_s[i].holey_entries;
         g_met[i].free    = g_s[i].num_free_entries;
         g_met[i].max     = g_s[i].max_entries;

         // c,pp,pg,h,f are points on a line between min, max scaled to #pixels
         c = 0 + (MAXPIXELS * g_met[i].commit)/g_met[i].max;
         pp= c + (MAXPIXELS * g_met[i].pendp)/g_met[i].max;
         pg=pp + (MAXPIXELS * g_met[i].pendg)/g_met[i].max;
         h =pg + (MAXPIXELS * g_met[i].hole)/g_met[i].max;
      // f = h + (MAXPIXELS * g_met[i].free)/g_met[i].max; // constant

         // Make 5 rectangles to fill with colors
      // g_met[i].c_rec.left = 0; // constant
```

```
            g_met[i].c_rec.right = c;

            g_met[i].p_rec.left  = c;
            g_met[i].p_rec.right = pp;

            g_met[i].g_rec.left  = pp;
            g_met[i].g_rec.right = pg;

            g_met[i].h_rec.left  = pg;
            g_met[i].h_rec.right = h;

            g_met[i].f_rec.left = h;
        //  g_met[i].f_rec.right = g_met[i].max; // constant

            DrawMeter(i);
        }
    }
}

void CQmanView::DrawMeter(int i){

    // Create the Brush
    CBrush *pCBrush = new CBrush(                RGB(127,255,255) );
    CBrush *pPBrush = new CBrush( HS_BDIAGONAL, RGB(000,182,255) );
    CBrush *pGBrush = new CBrush( HS_FDIAGONAL, RGB(000,182,255) );
    CBrush *pHBrush = new CBrush( HS_DIAGCROSS, RGB(255,128,128) );
        // The DC for the meter
    CDC* pCOLORDC = GetDlgItem(IDC_METERS[i])->GetDC();

   // Create a PEN
   // CPen *pQPen = new CPen(PS_SOLID, 3,RGB(0,0,255));
   // CPen *pOldPen = pXDC->SelectObject(pQPen);

    // Select this brush, save the old
    CBrush *pOldBrush =
    pCOLORDC->SelectObject(pCBrush);
    pCOLORDC->Rectangle(g_met[i].c_rec);

    pCOLORDC->SelectObject(pPBrush);
    pCOLORDC->Rectangle(g_met[i].p_rec);

    pCOLORDC->SelectObject(pGBrush);
    pCOLORDC->Rectangle(g_met[i].g_rec);

    pCOLORDC->SelectObject(pHBrush);
    pCOLORDC->Rectangle(g_met[i].h_rec);

    pCOLORDC->SelectStockObject(WHITE_BRUSH);
    pCOLORDC->Rectangle(g_met[i].f_rec);


    pCOLORDC->SelectObject(&pOldBrush);  // Reset the brush
    GetDlgItem(IDC_METERS[i])->Invalidate();

    delete(pCBrush);
    delete(pPBrush);
    delete(pGBrush);
    delete(pHBrush);
    ReleaseDC(pCOLORDC);
```

```
}

/*    //////////////////////////////////////
      // Init the track bar               //
      //  0            TPS           100  //
      //  |------------[]------------|   //
      // MIN                          MAX //
      //////////////////////////////////////


void InitTrackBar(HWND hTrack, HWND HMIN, int TMIN, HWND HMAX, int TMAX) {
   CString s;

   //HWND hTrack = GetDlgItem(ID)->m_hWnd;
   ::SendMessage(hTrack,TBM_SETRANGEMIN,TRUE,TMIN);
   ::SendMessage(hTrack,TBM_SETRANGEMAX,TRUE,TMAX);
   ::SendMessage(hTrack,TBM_SETTICFREQ,1,TRUE);
   ::SendMessage(hTrack,TBM_SETPOS,TRUE,TMIN);
   ::SendMessage(hTrack,TBM_SETSELSTART,TRUE,TMIN); // Select from start

   // Track Bar lables
  //s.Format("%d",TMIN); SetDlgItemText(IDMIN,s);
  // s.Format("%d",TMAX); SetDlgItemText(IDMAX,s);
   s.Format("%d",TMIN);  ::SendMessage(HMIN,WM_SETTEXT,0,(LPARAM)(LPCTSTR) s);
   s.Format("%d",TMAX);  ::SendMessage(HMAX,WM_SETTEXT,0,(LPARAM)(LPCTSTR) s);

}
*/


DWORD Poll(LPVOID qnum)
{
   MSGH mh;

   CString s,s1;
   int used = 1; // Clear the track bar if not in use
   int i = (int)(qnum);

   g_poll = 1;
   while(g_poll) {

      if (QS[i]) {
         int  sz;

         used++;
         if (QS[i]->open_time)
            ::SendMessage(g_tlab[i],WM_SETTEXT,0,(LPARAM)(LPCTSTR) "STARTING..."

         if (QSUCCESS >= (QsendAndReceive(QS[i],ADMINREQ_MODE,QADM_REQ_STATS, 0,
               sizeof(g_s[i]),(char *)&g_s[i],&sz,&mh))){
            if ((mh.mode == ADMINREP_MODE || mh.mode == ACK_MODE ) && (sz > 0)){
                                 // Note: now, mode is ADMINREP_MODE if local or

               if (!memcmp( &g_s[i+3], &g_s[i],sizeof(QADMSTATS) )) // nothing c
                  goto skip;
               memcpy( &g_s[i+3], &g_s[i],sizeof(QADMSTATS)); // set history

               // Assign a picture
               if (g_s[i].qget_state && g_s[i].qput_state)
                  g_pic[i] = QUP;
            else if (g_s[i].qget_state)
               g_pic[i] = QNOPUT;
            else if (g_s[i].qput_state)
               g_pic[i] = QNOGET;
            else
               g_pic[i] = QNOPG;

            if (g_s[i].num_free_entries == 0)
               g_pic[i] = QFULL;


            if (QS[i]->open_time) QS[i]->open_time = 0;  // Get rid of "START

             s.Format("%s:%s at %s has %d entries",&g_s[i].physical_qname,&g_
             s1.Format("%d",g_s[i].max_entries);
            ::SendMessage(g_tlab[i],WM_SETTEXT,0,(LPARAM)(LPCTSTR) s);
            ::SendMessage(g_tmax[i],WM_SETTEXT,0,(LPARAM)(LPCTSTR) s1);
            ::SendMessage(g_tmin[i],WM_SETTEXT,0,(LPARAM)(LPCTSTR) "0");

         } else {
            g_pic[i] = QDOWN;
            ::SendMessage(g_tlab[i],WM_SETTEXT,0,(LPARAM)(LPCTSTR) "BAD REPLY
            QS[i] = NULL;  Sleep(3000);
         }
      } else {
         g_pic[i] = QDOWN;
         ::SendMessage(g_tlab[i],WM_SETTEXT,0,(LPARAM)(LPCTSTR) "TIME OUT");
         QS[i] = NULL;  Sleep(3000);

      }
   } else { // No open que
     // if (g_que[i] == "")
    //     g_pic[i] = QNONE;
      if (used) {
         used = 0; // The track bar will now be clear
         memset(&g_s[i],0,sizeof(g_s[i]));
         g_s[i+3].max_entries = 1;
         //::SendMessage(g_track[i],TBM_SETPOS,TRUE,0);
         //::SendMessage(g_track[i],TBM_SETSELEND,TRUE,0);
         ::SendMessage(g_tlab[i],WM_SETTEXT,0,(LPARAM)(LPCTSTR) "No Que");
         ::SendMessage(g_tmin[i],WM_SETTEXT,0,(LPARAM)(LPCTSTR) "");
         ::SendMessage(g_tmax[i],WM_SETTEXT,0,(LPARAM)(LPCTSTR) "");
      }
   }
skip: Sleep(g_poll_delay);
   } // While loop
   return(1);
}


void CQmanView::CmdLine(int pass){
   CString parm,value,line,nline;
   int i,poll;

   line = AfxGetApp()->m_lpCmdLine;

   while (2 <= (i = sscanf(LPCTSTR(line),"%s %s %[^@]",
      parm.GetBuffer(100),value.GetBuffer(100),nline.GetBuffer(100) ))) {
      parm.MakeUpper();
      if (parm == "POLL") {
         sscanf(LPCTSTR(value),"%d",&poll);
         if((MIN_POLL_DLY <= poll) && (poll <= MAX_POLL_DLY))
            g_poll_delay  = poll;
      }

      if (parm == "1") {
         CComboBox * CB = (CComboBox *) this->GetDlgItem(IDC_QUES1);
         CB->SelectString(-1,value);
         OnSelchangeQues1();
      }
      if (parm == "2") {
         CComboBox * CB = (CComboBox *) this->GetDlgItem(IDC_QUES2);
         CB->SelectString(-1,value);
         OnSelchangeQues2();
      }
      if (parm == "3") {
         CComboBox * CB = (CComboBox *) this->GetDlgItem(IDC_QUES3);
         CB->SelectString(-1,value);
         OnSelchangeQues3();
      }

      line = LPCTSTR(nline);
      nline = "";
   }
}




int g_poll_delay_old;
void CQmanView::OnTimer(UINT nIDEvent)
{

// Select ICONS
        if (nIDEvent == MYTIMER){
      for (int i=0;i<3;i++){

         if (g_pic[i] != g_pic[i+3]){
            GetDlgItem(IDC_PICS[i][g_pic[i]])->ShowWindow(SW_SHOW);
            GetDlgItem(IDC_PICS[i][g_pic[i+3]])->ShowWindow(SW_HIDE);
            g_pic[i+3] = g_pic[i];
         }
      }
      TestMeters();

        } else if (nIDEvent == TITLETIMER) {
      GetParentFrame()->SetWindowText("QMAN");
      KillTimer(TITLETIMER);
      DrawKey();
        } else
        CFormView::OnTimer(nIDEvent);
}


void CQmanView::OnInitialUpdate()
{
      LOGFONT lf;
      CString s;
      CFormView::OnInitialUpdate(); // Default from vc++

  // Set frame size = Form size
  GetParentFrame()->RecalcLayout();
  ResizeParentToFit(FALSE);
  ResizeParentToFit(TRUE);



  GetMeterBoxes();

  DrawKey();

 for (int i=0;i<3;i++){

//      g_track[i] = GetDlgItem(IDC_TRACKS[i])->m_hWnd;
     g_tmin[i] = GetDlgItem(IDC_TMINS[i])->m_hWnd;
     g_tmax[i] = GetDlgItem(IDC_TMAXS[i])->m_hWnd;
     g_tlab[i] = GetDlgItem(IDC_TLABS[i])->m_hWnd;
     g_pic[i] = QNONE;     // man
     g_pic[i+3] = QSTOP;   // yellow
}



  // LIST OF QUEUES

  // CListBox* lb = (CListBox*) GetDlgItem(IDC_QUE);
  // lb->InsertString(-1,"Q1");
  // lb->InsertString(-1,"Q2");
  // lb->SetCurSel(0);
  //Q = Qopen("Q1",PUTTING,0,0,0,0,0);

  CComboBox * CB1 = (CComboBox *) this->GetDlgItem(IDC_QUES1);
  CComboBox * CB2 = (CComboBox *) this->GetDlgItem(IDC_QUES2);
  CComboBox * CB3 = (CComboBox *) this->GetDlgItem(IDC_QUES3);
  CB1->ResetContent();
  CB2->ResetContent();
  CB3->ResetContent();


// List all logical queues
// APPS: [physical],logical1,logical2,[physical],logical,
   if (sm_base = AttachSharedMemory()){
      lpRT  rt = RTROOT;
      while(rt = NextRT(rt)) {
         char *e,*s = RT_APPS(rt);     // Starts after the first letter
         while (s = strchr(s,',')) {  // Ends at next ","
            if (e = strchr(++s,',')) {
               *e = 0;
               if ((!strchr(s,'[')) && *s) {
                  CB1->AddString(s); // lb->InsertString(-1,s)
                  CB2->AddString(s); // lb->InsertString(-1,s)
                  CB3->AddString(s); // lb->InsertString(-1,s)
               }
               *e = ',';
            }
         }
      }
   } else {
      GetParentFrame()->SetMessageText("QNETD not running(?) please start it.");
      MessageBox("QNETD not running, please start it.",0,MB_ICONSTOP);
      //AfxGetMainWnd()->DestroyWindow();
   }
   CB1->AddString("");  //CB1->SetCurSel(1);
   CB2->AddString("");  //CB2->SetCurSel(1);
   CB3->AddString("");  //CB3->SetCurSel(1);


   DWORD id;
   CreateThread(NULL,0,(LPTHREAD_START_ROUTINE) Poll,(LPVOID) 0,0,&id);
   CreateThread(NULL,0,(LPTHREAD_START_ROUTINE) Poll,(LPVOID) 1,0,&id);
   CreateThread(NULL,0,(LPTHREAD_START_ROUTINE) Poll,(LPVOID) 2,0,&id);

   CmdLine(1);


   SetTimer(MYTIMER,250,NULL); // 1/4 second
   SetTimer(TITLETIMER,100,NULL);


   // Fonts
   memset(&lf,0,sizeof(LOGFONT));
#ifdef BIGFONT
   lf.lfHeight = 18;
#else
   lf.lfHeight = 13;
#endif
   g_text_font.CreateFontIndirect(&lf);
        strcpy(lf.lfFaceName,"Matura MT Script Capitals");
        strcpy(lf.lfFaceName,"Monotype Corsiva");
   lf.lfHeight = 32;
   m_title_font.CreateFontIndirect(&lf);


        GetDlgItem(IDC_TITLE)->SetFont(&m_title_font);

   i = 0;
   while (ALL_TEXT[i])
      GetDlgItem(ALL_TEXT[i++])->SetFont(&g_text_font);


}

void CQmanView::OnHScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar)
{
        // TODO: Add your message handler code here and/or call default

//      CFormView::OnHScroll(nSBCode, nPos, pScrollBar);  // Removed by derek
}


void CQmanView::OpenQue(int i, int IDC_QUES)
{
        CString s,que;
   int stat;

   GetDlgItemText(IDC_QUES,que.GetBuffer(100),100);
   que.ReleaseBuffer();

   if (que != g_que[i]) { // the user changed the open que name
      GetParentFrame()->SetWindowText("QMAN");

      if (que == "") {
         g_pic[i] = QNONE;
         g_que[i] = "";
         s = "No Que will be used";
         Qclose(&QS[i],0);
      } else {
         g_pic[i] = QSTOP;
         s.Format("Opening que %s",que);
         memset(&g_s[i],0,sizeof(QADMSTATS));
         GetParentFrame()->SetMessageText(s); // MessageBox(s);  causes 2nd pass
         if (QS[i] = Qopen(que.GetBuffer(0),PUT_MODE,0,Q_FAILOVER,&stat,0,0) ) {
            g_que[i] = que;
            s.Format("Qopen(%s)",que);
  //          QS[i]->time_out = 1000; // 1 second
         } else {
            s.Format("Qopen(%s) Error %d",que,stat);
            CComboBox * CB = (CComboBox *) this->GetDlgItem(IDC_QUES);
            CB->SelectString(-1,""); g_que[i] = ""; g_pic[i] = QDOWN;
         }
      }
      GetParentFrame()->SetMessageText(s);
   }
}

void CQmanView::OnSelchangeQues1(){      OpenQue(0,IDC_QUES1);}
void CQmanView::OnSelchangeQues2(){ OpenQue(1,IDC_QUES2);}
void CQmanView::OnSelchangeQues3(){ OpenQue(2,IDC_QUES3);}

void CallAdm(int id){
   if (QS[id]) {
         CAdminDlg adm;
      adm.m_id = id;
      adm.DoModal();
   }
}

void CallData(int id){
   if (QS[id]) {
         CKeySearch d;  // CdataDlg
      d.m_id = id;
      d.DoModal();
   }
}

void CQmanView::OnAdminb1() { CallAdm(0); }
void CQmanView::OnAdminb2() { CallAdm(1); }
void CQmanView::OnAdminb3() { CallAdm(2); }


void CQmanView::OnDatab1() { CallData(0); }
void CQmanView::OnDatab2() { CallData(1); }
void CQmanView::OnDatab3() { CallData(2); }

void CQmanView::OnDraw(CDC* pDC)
```

```
{
        DrawMeter(0);
        DrawMeter(1);
        DrawMeter(2);
   DrawKey();
        CFormView::OnDraw(pDC);
}

//      this->SetWindowText("Qman av");
//       GetParentFrame()->SetWindowText("QQMAN");


void CQmanView::OnRButtonDown(UINT nFlags, CPoint point)
{
   GetParentFrame()->SetMessageText("");
   this->Invalidate();
        CFormView::OnRButtonDown(nFlags, point);
}
```

```
User: root
Host: bunny
Class: bunny
Job: stdin
```

```
// bufdlg.cpp : implementation file
//

#include "stdafx.h"
#include "netman.h"

#include "bufdlg.h"
// #include "buferdlg.h"



//+++++++++++++ QLIB ++++++++++++++++++
#define Q_LIB
#include "qlib.h"
#include "rt.h"
#include "netadmin.h"

#define  BUFTIMER 101
extern   lpSMBUFH    sm_base;
extern   SMBUFH      g_smhead;
extern   SMBUFH      g_smhead_old;
extern   CString     g_node;
extern   int         g_polling;
extern   int         g_remote_node;
extern   int         g_pic;
extern   lpQHANDLE   g_q;
extern   lpRT        g_rt;

         BS          g_bs_old;
         BSA         g_bsa, g_bsa_old;
         int         g_lbsel;

extern   int         g_CBufDlg_state;


//+++++++++++++ QLIB ++++++++++++++++++


#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CBufDlg dialog


CBufDlg::CBufDlg(CWnd* pParent /*=NULL*/)
        : CDialog(CBufDlg::IDD, pParent)
{
        //{{AFX_DATA_INIT(CBufDlg)
                // NOTE: the ClassWizard will add member initialization here
        //}}AFX_DATA_INIT
}


void CBufDlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CBufDlg)
                // NOTE: the ClassWizard will add DDX and DDV calls here
        //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CBufDlg, CDialog)
        //{{AFX_MSG_MAP(CBufDlg)
        ON_WM_TIMER()
        ON_LBN_DBLCLK(IDC_BUFLB, OnDblclkBuflb)
        ON_LBN_SELCHANGE(IDC_BUFLB, OnSelchangeBuflb)
        ON_WM_CREATE()
        ON_WM_RBUTTONDOWN()
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

//////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////////



//////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////////
// CBufDlg message handlers

BOOL CBufDlg::OnInitDialog()
{
        CDialog::OnInitDialog();

        return TRUE;  // return TRUE unless you set the focus to a control
                      // EXCEPTION: OCX Property Pages should return FALSE
}



int g_BufPoll = 0;
DWORD BufPoll(LPVOID ttype)
{
   int i,type = (int)(ttype);

   if (g_remote_node) {

      if ((g_BufPoll++ == 0 ) && g_q && g_rt) {

         QsendAndReceive(g_q,QNETDREQ_MODE,NETMAN_SMBUFS, 0,0,0,
           (sizeof(BSA)),(char *) &g_bsa,0,0);
      }

   } else { // pack local data into structure

      lpBSA b = &g_bsa;
      b->nsbuf = SHAREDATA(nsbuf);
      for (i=0; i < SHAREDATA(nsbuf); i++)
         memcpy(&b->bs[i], SMBUFADDR(i) , sizeof(BS) );
   }
   g_BufPoll = 0;
   return(1);
}
```

```
int   g_display_a_buf = 0;
#define BUFHS g_bsa.bs[g_lbsel]
#define MSGHS g_bsa.bs[g_lbsel].msgh
#define SHOWITEM(X) GetDlgItem(X)->ShowWindow(SW_SHOW);
#define HIDEITEM(X) GetDlgItem(X)->ShowWindow(SW_HIDE);

void CBufDlg::OnTimer(UINT nIDEvent)
{
        if (nIDEvent == BUFTIMER){
      DWORD id; int i,copy=0;;
      CString s;

      if (g_CBufDlg_state >= 10) OnOK();


      if ( memcmp(&g_bsa,&g_bsa_old, sizeof(BS) * g_bsa.nsbuf )){
         copy++; // will copy
         memcpy(&g_bsa_old,&g_bsa,sizeof(BSA));

         CListBox* lb = (CListBox*) GetDlgItem(IDC_BUFLB);
         lb->ResetContent();

         for (i=0; i < g_bsa.nsbuf; i++) {
            s.Format("%5d %d %d %s",i+1, g_bsa.bs[i].status, g_bsa.bs[i].sub_sta
            lb->InsertString(-1,s);
         }
      }

      // g_display_a_buf 1 = init; 2 = run ; 3 = stop;

      switch (g_display_a_buf) {
      case 1: // run
         g_display_a_buf = 2; // run
         SHOWITEM(IDC_DATE);
         SHOWITEM(IDC_DATEL);
         SHOWITEM(IDC_MID);
         SHOWITEM(IDC_MIDL);
         SHOWITEM(IDC_TO);
         SHOWITEM(IDC_TO2);
         SHOWITEM(IDC_TOL);
         SHOWITEM(IDC_FROM);
         SHOWITEM(IDC_FROML);
         SHOWITEM(IDC_PB0);
         SHOWITEM(IDC_PB1);
         SHOWITEM(IDC_PB2);
         SHOWITEM(IDC_PB3);
         SHOWITEM(IDC_PB4);
         SHOWITEM(IDC_PB5);
         SHOWITEM(IDC_PB6);
         SHOWITEM(IDC_PB7);
         SetDlgItemText(IDC_PB8,
            g_node + "'s " + BUFHS.name + " buffer");
      case 2: // run

         if (memcmp(&BUFHS,&g_bs_old,sizeof(BS))) { // some change
            memcpy(&g_bs_old,&BUFHS,sizeof(BS));
            if (MSGHS.time) SetDlgItemText(IDC_DATE,ctime(&MSGHS.time));
            else            SetDlgItemText(IDC_DATE,"No date");
            s.Format("from=%s uid=%d tid=%d",((char *)IP2Name(MSGHS.mid.host)),M
            SetDlgItemText(IDC_MID,s);
            s.Format("%s %s ",MSGHS.to_server,MSGHS.to_logical);
            SetDlgItemText(IDC_TO,s);
            s.Format("%s port=%d buf=%d",IP2Name(MSGHS.to_node),MSGHS.to_port
            SetDlgItemText(IDC_TO2,s);
            s.Format("buffer %d (reply to buffer %d)",MSGHS.from_smbuf,MSGHS.rep
            SetDlgItemText(IDC_FROM,s);
         }
         break;
      case 3:  // disable
         g_display_a_buf = 0; // off
         HIDEITEM(IDC_DATE);
         HIDEITEM(IDC_DATEL);
         HIDEITEM(IDC_MID);
         HIDEITEM(IDC_MIDL);
         HIDEITEM(IDC_TO);
         HIDEITEM(IDC_TO2);
         HIDEITEM(IDC_TOL);
         HIDEITEM(IDC_FROM);
         HIDEITEM(IDC_FROML);
         HIDEITEM(IDC_PB0);
         HIDEITEM(IDC_PB1);
         HIDEITEM(IDC_PB2);
         HIDEITEM(IDC_PB3);
         HIDEITEM(IDC_PB4);
         HIDEITEM(IDC_PB5);
         HIDEITEM(IDC_PB6);
         HIDEITEM(IDC_PB7);
         SetDlgItemText(IDC_PB8,"Detail");
      }


      if (!g_remote_node);
      CreateThread(NULL,0,(LPTHREAD_START_ROUTINE) BufPoll,(LPVOID) 0,0,&id);

         } else {
         CDialog::OnTimer(nIDEvent);
    }
}
  /*
void CBufDlg::OnLButtonDblClk(UINT nFlags, CPoint point)
{
// If inside the list box
   CListBox * LB= (CListBox *) this->GetDlgItem(IDC_BUFLB);
   g_lbsel = LB->GetCurSel();


   CBufDlg d;
   d.DoModal();


// else

         CDialog::OnLButtonDblClk(nFlags, point);
}
  */
```

```
void CBufDlg::OnDblclkBuflb()
{
   // CString s ;
   CListBox * LB= (CListBox *) this->GetDlgItem(IDC_BUFLB);
   g_lbsel = LB->GetCurSel();
   // Somtimes returns out-of-bounds nunmber

   if ((g_lbsel >=0 )&&(g_lbsel < g_bsa.nsbuf)) {
      if(strcmp(g_bsa.bs[g_lbsel].name,"empty")) {
         g_display_a_buf = 1; // init

      // CBuferDlg d;
      // d.DoModal();
      }
   }
}

void CBufDlg::OnSelchangeBuflb()
{
   if (g_display_a_buf)
      g_display_a_buf = 3; // stop
}

void CBufDlg::OnOK()
{
   g_CBufDlg_state = 20;
   KillTimer(BUFTIMER);
   this->DestroyWindow();
        //CDialog::OnOK();
}

int CBufDlg::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
        if (CDialog::OnCreate(lpCreateStruct) == -1)
                return -1;

      this->SetWindowText(g_node + " buffers");

      memset(&g_bsa_old,0,sizeof(BSA));

      SetTimer(BUFTIMER,1000,NULL); // 1 second

           g_CBufDlg_state = 2;
        return 0;
}

void CBufDlg::OnRButtonDown(UINT nFlags, CPoint point)
{
           GetParentFrame()->SetMessageText("");
   this->Invalidate();


        CDialog::OnRButtonDown(nFlags, point);
}
```

```
User: root
Host: bunny
Class: bunny
Job: stdin
```

```
// bufdlg.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CBufDlg dialog

class CBufDlg : public CDialog
{
// Construction
public:
        CBufDlg(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
        //{{AFX_DATA(CBufDlg)
        enum { IDD = IDD_BUF_DLG };
                // NOTE: the ClassWizard will add data members here
        //}}AFX_DATA


// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CBufDlg)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        //}}AFX_VIRTUAL

// Implementation
protected:

        // Generated message map functions
        //{{AFX_MSG(CBufDlg)
        virtual BOOL OnInitDialog();
        afx_msg void OnTimer(UINT nIDEvent);
        afx_msg void OnDblclkBuflb();
        afx_msg void OnSelchangeBuflb();
        virtual void OnOK();
        afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
        afx_msg void OnRButtonDown(UINT nFlags, CPoint point);
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};
```

```
// buferdlg.cpp : implementation file
//

#include "stdafx.h"
#include "netman.h"
#include "buferdlg.h"

#define Q_LIB
#define BUFFERTIMER   103
#include "qlib.h"
#include "netadmin.h"

extern  CString     g_node;
extern  BSA         g_bsa;
extern  int         g_lbsel;
extern  BS          g_bs_old;


#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CBuferDlg dialog


CBuferDlg::CBuferDlg(CWnd* pParent /*=NULL*/)
        : CDialog(CBuferDlg::IDD, pParent)
{
        //{{AFX_DATA_INIT(CBuferDlg)
                // NOTE: the ClassWizard will add member initialization here
        //}}AFX_DATA_INIT
}


void CBuferDlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CBuferDlg)
                // NOTE: the ClassWizard will add DDX and DDV calls here
        //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CBuferDlg, CDialog)
        //{{AFX_MSG_MAP(CBuferDlg)
        ON_WM_TIMER()
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()


/////////////////////////////////////////////////////////////////////////////
// CBuferDlg message handlers

BOOL CBuferDlg::OnInitDialog()
{
        CDialog::OnInitDialog();

        // TODO: Add extra initialization here
```

```
   memset(&g_bs_old,0,sizeof(BS));
   SetTimer(BUFFERTIMER,1000,NULL); // 1 second

        return TRUE;  // return TRUE unless you set the focus to a control
                      // EXCEPTION: OCX Property Pages should return FALSE
}

#define BUFHS g_bsa.bs[g_lbsel]
#define MSGHS g_bsa.bs[g_lbsel].msgh
#define POST  lb->InsertString(-1,s)

void CBuferDlg::OnTimer(UINT nIDEvent)
{
        if (nIDEvent == BUFFERTIMER){
      if (memcmp(&BUFHS,&g_bs_old,sizeof(BS))) { // some change
         memcpy(&g_bs_old,&BUFHS,sizeof(BS));
         CString s,t;
         CListBox * lb= (CListBox *) this->GetDlgItem(IDC_BUFLB);

         t = ctime(&MSGHS.time); s = "Message: " + t; POST;
         s.Format("Mode=%4d Sub=%4d",MSGHS.mode,MSGHS.sub_mode); POST;
         s.Format("MID host=%x tid=%d",MSGHS.mid.host,MSGHS.mid.tid); POST;
         s.Format("To %x Port %d Buffer %x",MSGHS.to_node,MSGHS.to_port,MSGHS.to
         s.Format("From %d Reply_to %d",MSGHS.from_smbuf,MSGHS.reply_smbuf); POS
         s.Format("To physical: %s logical: %s",MSGHS.to_server,MSGHS.to_logical

         this->SetWindowText(g_node + "'s " + BUFHS.name + " buffer");
      }
  } else
        CDialog::OnTimer(nIDEvent);
}
```

```
// buferdlg.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CBuferDlg dialog

class CBuferDlg : public CDialog
{
// Construction
public:
        CBuferDlg(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
        //{{AFX_DATA(CBuferDlg)
        enum { IDD = IDD_BUFFER };
                // NOTE: the ClassWizard will add data members here
        //}}AFX_DATA


// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CBuferDlg)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        //}}AFX_VIRTUAL

// Implementation
protected:

        // Generated message map functions
        //{{AFX_MSG(CBuferDlg)
        virtual BOOL OnInitDialog();
        afx_msg void OnTimer(UINT nIDEvent);
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};
```

```
// mainfrm.cpp : implementation of the CMainFrame class
//

#include "stdafx.h"
#include "netman.h"

#include "mainfrm.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CMainFrame

IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
        //{{AFX_MSG_MAP(CMainFrame)
                // NOTE - the ClassWizard will add and remove mapping macros her
                //    DO NOT EDIT what you see in these blocks of generated code
        ON_WM_CREATE()
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// arrays of IDs used to initialize control bars

// toolbar buttons - IDs are command buttons
static UINT BASED_CODE buttons[] =
{
        // same order as in the bitmap 'toolbar.bmp'
        ID_FILE_NEW,
        ID_FILE_OPEN,
        ID_FILE_SAVE,
                ID_SEPARATOR,
        ID_EDIT_CUT,
        ID_EDIT_COPY,
        ID_EDIT_PASTE,
                ID_SEPARATOR,
        ID_FILE_PRINT,
        ID_APP_ABOUT,
};

static UINT BASED_CODE indicators[] =
{
        ID_SEPARATOR,           // status line indicator
        ID_INDICATOR_CAPS,
        ID_INDICATOR_NUM,
        ID_INDICATOR_SCRL,
};

/////////////////////////////////////////////////////////////////////////////
// CMainFrame construction/destruction

CMainFrame::CMainFrame()
{
        // TODO: add member initialization code here
}

CMainFrame::~CMainFrame()
{
}

int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
        if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
                return -1;

        if (!m_wndToolBar.Create(this) ||
                !m_wndToolBar.LoadBitmap(IDR_MAINFRAME) ||
                !m_wndToolBar.SetButtons(buttons,
                  sizeof(buttons)/sizeof(UINT)))
        {
                TRACE0("Failed to create toolbar\n");
                return -1;      // fail to create
        }

/* Derek's remove tool bar */
    m_wndToolBar.ShowWindow(SW_HIDE);

        if (!m_wndStatusBar.Create(this) ||
                !m_wndStatusBar.SetIndicators(indicators,
                  sizeof(indicators)/sizeof(UINT)))
        {
                TRACE0("Failed to create status bar\n");
                return -1;      // fail to create
        }

        // TODO: Delete these three lines if you don't want the toolbar to
        //  be dockable
        m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
        EnableDocking(CBRS_ALIGN_ANY);
        DockControlBar(&m_wndToolBar);

        // TODO: Remove this if you don't want tool tips
        m_wndToolBar.SetBarStyle(m_wndToolBar.GetBarStyle() |
                CBRS_TOOLTIPS | CBRS_FLYBY);

        return 0;
}

/////////////////////////////////////////////////////////////////////////////
// CMainFrame diagnostics

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
        CFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{
        CFrameWnd::Dump(dc);
}

#endif //_DEBUG
```

```
/////////////////////////////////////////////////////////////////////////////
// CMainFrame message handlers
// mainfrm.h : interface of the CMainFrame class
//
/////////////////////////////////////////////////////////////////////////////

class CMainFrame : public CFrameWnd
{
protected: // create from serialization only
        CMainFrame();
        DECLARE_DYNCREATE(CMainFrame)

// Attributes
public:

// Operations
public:

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CMainFrame)
        //}}AFX_VIRTUAL

// Implementation
public:
        virtual ~CMainFrame();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

protected:  // control bar embedded members
        CStatusBar  m_wndStatusBar;
        CToolBar    m_wndToolBar;

// Generated message map functions
protected:
        //{{AFX_MSG(CMainFrame)
        afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
                // NOTE - the ClassWizard will add and remove member functions h
                //    DO NOT EDIT what you see in these blocks of generated code
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////
```

```
// netmadoc.cpp : implementation of the CNetmanDoc class
//

#include "stdafx.h"
#include "netman.h"

#include "netmadoc.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CNetmanDoc

IMPLEMENT_DYNCREATE(CNetmanDoc, CDocument)

BEGIN_MESSAGE_MAP(CNetmanDoc, CDocument)
	//{{AFX_MSG_MAP(CNetmanDoc)
		// NOTE - the ClassWizard will add and remove mapping macros her
		//    DO NOT EDIT what you see in these blocks of generated code
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CNetmanDoc construction/destruction

CNetmanDoc::CNetmanDoc()
{
	// TODO: add one-time construction code here

}

CNetmanDoc::~CNetmanDoc()
{
}

BOOL CNetmanDoc::OnNewDocument()
{
	if (!CDocument::OnNewDocument())
		return FALSE;

	// TODO: add reinitialization code here
	// (SDI documents will reuse this document)

	return TRUE;
}

/////////////////////////////////////////////////////////////////////////////
// CNetmanDoc serialization

void CNetmanDoc::Serialize(CArchive& ar)
{
	if (ar.IsStoring())
	{
		// TODO: add storing code here
	}
	else
	{
```

```
                // TODO: add loading code here
        }
}

/////////////////////////////////////////////////////////////////////////////
// CNetmanDoc diagnostics

#ifdef _DEBUG
void CNetmanDoc::AssertValid() const
{
        CDocument::AssertValid();
}

void CNetmanDoc::Dump(CDumpContext& dc) const
{
        CDocument::Dump(dc);
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CNetmanDoc commands
// netmadoc.h : interface of the CNetmanDoc class
//
/////////////////////////////////////////////////////////////////////////////

class CNetmanDoc : public CDocument
{
protected: // create from serialization only
        CNetmanDoc();
        DECLARE_DYNCREATE(CNetmanDoc)

// Attributes
public:

// Operations
public:

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CNetmanDoc)
        public:
        virtual BOOL OnNewDocument();
        //}}AFX_VIRTUAL

// Implementation
public:
        virtual ~CNetmanDoc();
        virtual void Serialize(CArchive& ar);   // overridden for document i/o
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
        //{{AFX_MSG(CNetmanDoc)
                // NOTE - the ClassWizard will add and remove member functions h
                //    DO NOT EDIT what you see in these blocks of generated code
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////
```

```
// netman.cpp : Defines the class behaviors for the application.
//

#include "stdafx.h"
#include "netman.h"

#include "mainfrm.h"
#include "netmadoc.h"

#include "bufdlg.h"
#include "netmavw.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CNetmanApp

BEGIN_MESSAGE_MAP(CNetmanApp, CWinApp)
        //{{AFX_MSG_MAP(CNetmanApp)
        ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
                // NOTE - the ClassWizard will add and remove mapping macros her
                //    DO NOT EDIT what you see in these blocks of generated code
        //}}AFX_MSG_MAP
        // Standard file based document commands
        ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
        ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
        // Standard print setup command
        ON_COMMAND(ID_FILE_PRINT_SETUP, CWinApp::OnFilePrintSetup)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CNetmanApp construction

CNetmanApp::CNetmanApp()
{
        // TODO: add construction code here,
        // Place all significant initialization in InitInstance
}

/////////////////////////////////////////////////////////////////////////////
// The one and only CNetmanApp object

CNetmanApp theApp;

/////////////////////////////////////////////////////////////////////////////
// CNetmanApp initialization

BOOL CNetmanApp::InitInstance()
{
        // Standard initialization
        // If you are not using these features and wish to reduce the size
        //  of your final executable, you should remove from the following
        //  the specific initialization routines you do not need.

        Enable3dControls();

        LoadStdProfileSettings();  // Load standard INI file options (including
        // Register the application's document templates.  Document templates
        //  serve as the connection between documents, frame windows and views.

        CSingleDocTemplate* pDocTemplate;
        pDocTemplate = new CSingleDocTemplate(
                IDR_MAINFRAME,
                RUNTIME_CLASS(CNetmanDoc),
                RUNTIME_CLASS(CMainFrame),       // main SDI frame window
                RUNTIME_CLASS(CNetmanView));
        AddDocTemplate(pDocTemplate);

        // create a new (empty) document
        OnFileNew();

        if (m_lpCmdLine[0] != '\0')
        {
                // TODO: add command line processing here
        }

        return TRUE;
}

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
        CAboutDlg();

// Dialog Data
        //{{AFX_DATA(CAboutDlg)
        enum { IDD = IDD_ABOUTBOX };
        //}}AFX_DATA
   CFont m_title_font;

// Implementation
protected:
        virtual void DoDataExchange(CDataExchange* pDX);     // DDX/DDV support
        //{{AFX_MSG(CAboutDlg)
        virtual BOOL OnInitDialog();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
        //{{AFX_DATA_INIT(CAboutDlg)
        //}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CAboutDlg)
        //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
```

```
        //{{AFX_MSG_MAP(CAboutDlg)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

// App command to run the dialog
void CNetmanApp::OnAppAbout()
{
        CAboutDlg aboutDlg;
        aboutDlg.DoModal();
}

/////////////////////////////////////////////////////////////////////////////
// CNetmanApp commands

BOOL CAboutDlg::OnInitDialog()
{
        CDialog::OnInitDialog();


   LOGFONT lf;
   memset(&lf,0,sizeof(LOGFONT));
        strcpy(lf.lfFaceName,"Monotype Corsiva");
   lf.lfHeight = 24;
   m_title_font.CreateFontIndirect(&lf);
        GetDlgItem(IDC_ABOUT1)->SetFont(&m_title_font);



        return TRUE;  // return TRUE unless you set the focus to a control
                      // EXCEPTION: OCX Property Pages should return FALSE
}
```

```
// netman.h : main header file for the NETMAN application
//

#ifndef __AFXWIN_H__
	#error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"       // main symbols

/////////////////////////////////////////////////////////////////////////////
// CNetmanApp:
// See netman.cpp for the implementation of this class
//

class CNetmanApp : public CWinApp
{
public:
	CNetmanApp();

// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CNetmanApp)
	public:
	virtual BOOL InitInstance();
	//}}AFX_VIRTUAL

// Implementation

	//{{AFX_MSG(CNetmanApp)
	afx_msg void OnAppAbout();
		// NOTE - the ClassWizard will add and remove member functior
		//    DO NOT EDIT what you see in these blocks of generated coue
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////
```

```
// netmavw.cpp : implementation of the CNetmanView class
//

#include "stdafx.h"
#include "netman.h"

#include "bufdlg.h"

#include "netmadoc.h"
#include "netmavw.h"

#include "rtdlg.h"


//+++++++++++++ QLIB ++++++++++++++++++
#include "qlib.h"
#include "rt.h"
#include "netadmin.h"
#define  MYTIMER 100
#define  SHUTDOWNTIME 101
extern   lpSMBUFH sm_base;
SMBUFH   g_smhead;
SMBUFH   g_smhead_old;
CString  g_node;
int      g_polling = 0;
int      g_remote_node = 0;
int      g_CBufDlg_state = 0;
int      g_pic = IDC_GREEN;
int      g_con = IDC_IS_CON;
lpQHANDLE  g_q = NULL;
lpRT       g_rt;

int      IDC_LAN[] = {IDC_GREEN,IDC_YELLOW,IDC_RED};

//+++++++++++++ QLIB ++++++++++++++++++


#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CNetmanView

IMPLEMENT_DYNCREATE(CNetmanView, CFormView)

BEGIN_MESSAGE_MAP(CNetmanView, CFormView)
        //{{AFX_MSG_MAP(CNetmanView)
        ON_CBN_SELCHANGE(IDC_NODES, OnSelchangeNodes)
        ON_WM_TIMER()
        ON_BN_CLICKED(IDC_EXITB, OnExitb)
        ON_BN_CLICKED(IDC_BUFB, OnBufb)
        ON_BN_CLICKED(IDC_ROUTEB, OnRouteb)
        ON_BN_CLICKED(IDC_FOLCLEARB, OnFolclearb)
        ON_WM_RBUTTONDOWN()
        ON_CBN_SETFOCUS(IDC_NODES, OnSetfocusNodes)
        //}}AFX_MSG_MAP
        // Standard printing commands
        ON_COMMAND(ID_FILE_PRINT, CFormView::OnFilePrint)
```

```
	ON_COMMAND(ID_FILE_PRINT_PREVIEW, CFormView::OnFilePrintPreview)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CNetmanView construction/destruction

CNetmanView::CNetmanView()
	: CFormView(CNetmanView::IDD)
{
	//{{AFX_DATA_INIT(CNetmanView)
		// NOTE: the ClassWizard will add member initialization here
	//}}AFX_DATA_INIT
	// TODO: add construction code here

}

CNetmanView::~CNetmanView()
{
}

void CNetmanView::DoDataExchange(CDataExchange* pDX)
{
	CFormView::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CNetmanView)
		// NOTE: the ClassWizard will add DDX and DDV calls here
	//}}AFX_DATA_MAP
}

/////////////////////////////////////////////////////////////////////////////
// CNetmanView printing

BOOL CNetmanView::OnPreparePrinting(CPrintInfo* pInfo)
{
	// default preparation
	return DoPreparePrinting(pInfo);
}

void CNetmanView::OnBeginPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
	// TODO: add extra initialization before printing
}

void CNetmanView::OnEndPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
	// TODO: add cleanup after printing
}

void CNetmanView::OnPrint(CDC* pDC, CPrintInfo*)
{
	// TODO: add code to print the controls
}

/////////////////////////////////////////////////////////////////////////////
// CNetmanView diagnostics

#ifdef _DEBUG
void CNetmanView::AssertValid() const
{
	CFormView::AssertValid();
}
```

```
void CNetmanView::Dump(CDumpContext& dc) const
{
        CFormView::Dump(dc);
}

CNetmanDoc* CNetmanView::GetDocument() // non-debug version is inline
{
        ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CNetmanDoc)));
        return (CNetmanDoc*)m_pDocument;
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CNetmanView message handlers

void CNetmanView::LoadList()  // List the possible nodes
{
   CComboBox * CB1 = (CComboBox *) this->GetDlgItem(IDC_NODES);
   CB1->ResetContent();

   if (sm_base = AttachSharedMemory()) {
      lpRT  rt = RTROOT;
      while(rt = NextRT(rt))
         CB1->AddString(RT_NODE(rt)); // lb->InsertString(-1,s)
   }

}

void CNetmanView::OnInitialUpdate()
{

   // Set frame size = Form size
   GetParentFrame()->RecalcLayout();
   ResizeParentToFit(FALSE);
   ResizeParentToFit(TRUE);

   m_d = NULL;
/*
   */
   LoadList();

   if (!sm_base) {
      MessageBox("QNETD not running, please start it.",0,MB_ICONSTOP);
      SetTimer(SHUTDOWNTIME,10,NULL);
   }

   // Show one PIC
   g_pic = IDC_RED;
   g_con = IDC_IS_CON;
   g_node = SHAREDATA(hostname);
   Light(IDC_YELLOW);
   Con(IDC_NO_CON);

   CComboBox * CB1 = (CComboBox *) this->GetDlgItem(IDC_NODES);
   CB1->SelectString(-1,SHAREDATA(hostname));

   SetTimer(MYTIMER,1000,NULL); // 1 second

        CFormView::OnInitialUpdate();
```

```
}


void CNetmanView::Light(int pic){
   if (g_pic != pic) {
      GetDlgItem(g_pic)->ShowWindow(SW_HIDE);
      GetDlgItem(pic)->ShowWindow(SW_SHOW);
      g_pic = pic;
   }
}

void CNetmanView::Con(int con){
   if (g_con != con) {
      GetDlgItem(g_con)->ShowWindow(SW_HIDE);
      GetDlgItem(con)->ShowWindow(SW_SHOW);
      g_con = con;
   }
}


int clock_stop;
void CNetmanView::OnSelchangeNodes()
{
   CComboBox * CB= (CComboBox *) this->GetDlgItem(IDC_NODES);
   g_CBufDlg_state = 10;

   CB->GetLBText( CB->GetCurSel() ,g_node.GetBuffer(100));
   g_node.ReleaseBuffer();
   if (g_node == SHAREDATA(hostname))
      g_remote_node = 0;
   else
      g_remote_node = 1;

   CString s; s.Format("%s's NETMAN",LPCTSTR(g_node));
   GetParentFrame()->SetWindowText(s);

   SetDlgItemText(IDC_FOLBOX, g_node + "'s fail over list");
  // Light(IDC_YELLOW);
   clock_stop = 5;
   g_smhead.time = g_smhead_old.time = 0;
}



void CNetmanView::UpdateStats(){
   int i;
   CString s;
   lpSMBUFH b,bb = &g_smhead_old;
   // int local = (g_node == SHAREDATA(hostname));

   if (g_remote_node)   b = &g_smhead;
   else                 b = sm_base;

   // Is the date moving?
   if(b->time == bb->time) clock_stop++;
   else clock_stop = 0;
   bb->time = b->time;
```

```
if (b->stat.warn != bb->stat.warn)
   SetDlgItemInt(IDC_WARN,(bb->stat.warn = b->stat.warn) );

if (b->stat.fail != bb->stat.fail)
   SetDlgItemInt(IDC_WARN,(bb->stat.fail = b->stat.fail) );


if (b->stat.puts != bb->stat.puts)
   SetDlgItemInt(IDC_PUTS,(bb->stat.puts = b->stat.puts) );

if (b->stat.gets != bb->stat.gets)
   SetDlgItemInt(IDC_GETS,(bb->stat.gets = b->stat.gets) );



if (b->stat.tx != bb->stat.tx)
   SetDlgItemInt(IDC_SENT,(bb->stat.tx = b->stat.tx) );

if (b->stat.rx != bb->stat.rx)
   SetDlgItemInt(IDC_RECEIVED,(bb->stat.rx = b->stat.rx) );


if (b->rt_rev != bb->rt_rev)
   SetDlgItemInt(IDC_RTREV,(bb->rt_rev = b->rt_rev) );

if (b->rt_ver != bb->rt_ver)
   SetDlgItemInt(IDC_RTVER,(bb->rt_ver = b->rt_ver) );



if ((b->rt_rev != bb->rt_rev)) { // RT and FOL update:
   if (!b->failed_servers) {    // No Failed servers
      GetDlgItem(IDC_FOL)->ShowWindow(SW_HIDE);
//      GetDlgItem(IDC_FOLBOX)->ShowWindow(SW_HIDE);
      GetDlgItem(IDC_FOLCLEARB)->ShowWindow(SW_HIDE);
   } else {
      GetDlgItem(IDC_FOL)->ShowWindow(SW_SHOW);
      GetDlgItem(IDC_FOLBOX)->ShowWindow(SW_SHOW);
      GetDlgItem(IDC_FOLCLEARB)->ShowWindow(SW_SHOW);

      CListBox* lb = (CListBox*) GetDlgItem(IDC_FOL);
      lb->ResetContent();
      lb->InsertString(-1,"==== FAILOVERS IN USE ====");
      for (i=0; i<b->failed_servers; i++) {
         s.Format("%5d %6s %x %d %d",i+1,
            b->FO[i].name,b->FO[i].ip,b->FO[i].puts,b->FO[i].gets);
         lb->InsertString(-1,s);
      }
   }
}

/*
if(!memcmp(&b->stat,&bb->stat,sizeof(b->stat)) || moved) {
   CListBox* lb = (CListBox*) GetDlgItem(IDC_STATS);
   lb->ResetContent();
   s.Format("opens=%d closes=%d openrep=%d",b->stat.opens,b->stat.closes,b
   s.Format("puts=%d gets=%d tx=%d rx=%d",b->stat.puts,b->stat.gets,b->stat.
   s.Format("commit=%d abort=%d warn=%d fail=%d",b->stat.commit,b->stat.abort
   moved++;
}
```

```
   /*
   if(local)
      for(i = 0; i<b->nsbuf;i++){
         CListBox* lb = (CListBox*) GetDlgItem(IDC_STATS);
         lb->ResetContent();
         s.Format("%5d %d %s",);lb->InsertString(-1,s);
      }
   */
   /*
   if((b->hostip != bb->hostip) ||
      (b->diag != bb->diag) || moved ) {
      s.Format("%s @ %x",b->hostname,b->hostip); SetDlgItemText(IDC_T1,s);
      s.Format("nsbuf=%d diag=%d timeout=%d sec",b->nsbuf,b->diag,(b->time_out/1
      s.Format("failed=%d rt_rev=%d",b->failed_servers,b->rt_rev); SetDlgItemTex


      memcpy(bb,b,sizeof(*b));
   }
   */

   if (clock_stop > 10 )       Light(IDC_RED);
   else if (clock_stop > 5 )   Light(IDC_YELLOW);
   else                        Light(IDC_GREEN);

   if (g_remote_node) Con(IDC_ISCON_PIC);
   else               Con(IDC_ISNOTCON_PIC);

}



DWORD Poll(LPVOID ttype)
{
   int type = (int)(ttype);

   if(g_polling++) return(1);

   if (g_node == "" || (!g_remote_node)
    )  // Not null or local
      goto BYE;

   if (!g_q) {
      g_q = Qopen("QNETD",PUT_MODE,0,0,0,0,0);
      if (!g_q) goto BYE;
   }

   if (!g_rt || RT_NODE(g_rt) != g_node)
      if (!(g_rt = ServerByNode(g_node.GetBuffer(0))))
         goto BYE;

   g_q->msgh.to_node = RT_IP(g_rt);
   g_q->msgh.to_port = RT_PORT(g_rt);
   strcpy(g_q->msgh.to_logical,"QNETD");
   strcpy(g_q->msgh.to_server,"QNETD");

   QsendAndReceive(g_q,QNETDREQ_MODE,NETMAN_SMBUFH, 0,0,0,
      (sizeof(SMBUFH) - MAXRTSIZE),(char *) &g_smhead,0,0);

BYE: g_polling = 0;
```

```
return(1);

}


void CNetmanView::OnTimer(UINT nIDEvent)
{
        if (nIDEvent == MYTIMER){
      DWORD id;


      if (g_CBufDlg_state == 20) {
         g_CBufDlg_state = 0;
         GetDlgItem(IDC_BUFB)->EnableWindow(TRUE);
      }

      UpdateStats();

      if (!g_remote_node);
      CreateThread(NULL,0,(LPTHREAD_START_ROUTINE) Poll,(LPVOID) 0,0,&id);

        } else if (nIDEvent == SHUTDOWNTIME) {
      OnExitb();
        } else {
           CFormView::OnTimer(nIDEvent);
   }
}

void CNetmanView::OnExitb()
{
   AfxGetMainWnd()->DestroyWindow();
}



           // CBufDlg d;
     // d.DoModal();

void CNetmanView::OnBufb()
{
   GetDlgItem(IDC_BUFB)->EnableWindow(FALSE);
   g_CBufDlg_state = 1; // Starting
   if (m_d == NULL) m_d = new CBufDlg(this);
   m_d->Create(IDD_BUF_DLG);
}


void CNetmanView::OnRouteb()
{
        CRtDlg d;
        d.DoModal();
}

void CNetmanView::OnFolclearb()
{
        if (g_remote_node) {
      QsendAndReceive(g_q,QNETDREQ_MODE,NETMAN_CLR_FOL,
      0,0,0,  0,0,0,0);
        } else
```

```
            SHAREDATA(failed_servers) = 0;
}

void CNetmanView::OnRButtonDown(UINT nFlags, CPoint point)
{
        GetParentFrame()->SetMessageText("");
   this->Invalidate();


        CFormView::OnRButtonDown(nFlags, point);
}

void CNetmanView::OnSetfocusNodes()
{
        LoadList();
}
```

```
// netmavw.h : interface of the CNetmanView class
//
/////////////////////////////////////////////////////////////////////////////

class CNetmanView : public CFormView
{
protected: // create from serialization only
        CNetmanView();
        DECLARE_DYNCREATE(CNetmanView)

public:
        //{{AFX_DATA(CNetmanView)
        enum{ IDD = IDD_NETMAN_FORM };
                // NOTE: the ClassWizard will add data members here
        //}}AFX_DATA
//   CString m_node;
   int g_pic;
   CBufDlg *m_d;
// Attributes
public:
        CNetmanDoc* GetDocument();

// Operations
public:

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CNetmanView)
        public:
        virtual void OnInitialUpdate();
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
        virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
        virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);
        virtual void OnPrint(CDC* pDC, CPrintInfo*);
        //}}AFX_VIRTUAL
   void UpdateStats();
   void LoadList();
   void Light(int pic);
   void Con(int pic);
// Implementation
public:
        virtual ~CNetmanView();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
        //{{AFX_MSG(CNetmanView)
        afx_msg void OnSelchangeNodes();
        afx_msg void OnTimer(UINT nIDEvent);
        afx_msg void OnExitb();
        afx_msg void OnBufb();
        afx_msg void OnRouteb();
        afx_msg void OnFolclearb();
        afx_msg void OnRButtonDown(UINT nFlags, CPoint point);
        afx_msg void OnSetfocusNodes();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

#ifndef _DEBUG  // debug version in netmavw.cpp
inline CNetmanDoc* CNetmanView::GetDocument()
   { return (CNetmanDoc*)m_pDocument; }
#endif

/////////////////////////////////////////////////////////////////////////////
```

```
//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.
// Used by netman.rc
//
#define IDD_ABOUTBOX                    100
#define IDD_NETMAN_FORM                 101
#define IDR_MAINFRAME                   128
#define IDR_NETMANTYPE                  129
#define IDD_BUF_DLG                     130
#define IDR_QUP                         131
#define IDD_BUFFER                      131
#define IDD_RTDLG                       132
#define IDR_QDOWN                       134
#define IDC_DOWN_PIC                    134
#define IDR_DOWN_PIC                    134
#define IDR_QNONE                       136
#define IDR_MAN                         136
#define IDR_NET                         137
#define IDR_NETMAN                      138
#define IDR_LANTERN                     139
#define IDR_LAN_RED                     140
#define IDR_LAN_GRN                     141
#define IDR_LAN_YEL                     142
#define IDR_NO_CON                      143
#define IDR_IS_CON                      144
#define IDR_3DIMES                      145
#define IDR_3DMDS                       146
#define IDC_NODES                       1000
#define IDC_ISCON_PIC                   1001
#define IDC_IS_CON                      1001
#define IDC_ISNOTCON_PIC                1002
#define IDC_NO_CON                      1002
#define IDC_GREEN                       1003
#define IDC_RED                         1004
#define IDC_YELLOW                      1005
#define IDC_EXITB                       1009
#define IDC_ROUTEB                      1010
#define IDC_BUFB                        1011
#define IDC_TRANB                       1012
#define IDC_FOLCLEARB                   1013
#define IDC_GETS                        1016
#define IDC_PUTS                        1017
#define IDC_SENT                        1018
#define IDC_RECEIVED                    1019
#define IDC_FAIL                        1020
#define IDC_WARN                        1021
#define IDC_RTREV                       1022
#define IDC_RTVER                       1023
#define IDC_FOL                         1024
#define IDC_BUFLB                       1025
#define IDC_BUFER_LB                    1025
#define IDC_RTREE                       1026
#define IDC_GETB                        1027
#define IDC_COPYB                       1027
#define IDC_BROADCASTB                  1028
#define IDC_FOLBOX                      1028
#define IDC_TREELB                      1029
#define IDC_PB1                         1030
#define IDC_CONLB                       1030
#define IDC_PB2                         1031
```

```
#define IDC_PB3                         1032
#define IDC_PB4                         1033
#define IDC_PB5                         1034
#define IDC_PB6                         1035
#define IDC_PB7                         1036
#define IDC_PB8                         1037
#define IDC_DATE                        1038
#define IDC_MID                         1039
#define IDC_TO                          1040
#define IDC_FROM                        1041
#define IDC_PB0                         1042
#define IDC_DATEL                       1043
#define IDC_MIDL                        1044
#define IDC_TOL                         1045
#define IDC_FROML                       1046
#define IDC_TO2                         1047
#define IDC_RT_TITLE                    1048
#define IDC_LOGO                        1049
#define IDC_ABOUT1                      1050

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_3D_CONTROLS                     1
#define _APS_NEXT_RESOURCE_VALUE        134
#define _APS_NEXT_COMMAND_VALUE         32771
#define _APS_NEXT_CONTROL_VALUE         1051
#define _APS_NEXT_SYMED_VALUE           101
#endif
#endif
```

```
// rtdlg.cpp : implementation file
//

#include "stdafx.h"
#include "netman.h"
#include "rtdlg.h"

//+++++++++++++ QLIB ++++++++++++++++++
#define Q_LIB
#include "qlib.h"
#include "rt.h"
#include "netadmin.h"
#include "qnetd.h"

#define  RTTIMER 104
extern   lpSMBUFH    sm_base;
extern   SMBUFH      g_smhead;
extern   SMBUFH      g_smhead_old;
extern   CString     g_node;
extern   int         g_polling;
extern   int         g_remote_node;
extern   int         g_pic;
extern   lpQHANDLE   g_q;
extern   lpRT        g_rt;


//+++++++++++++ QLIB ++++++++++++++++++



#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CRtDlg dialog


CRtDlg::CRtDlg(CWnd* pParent /*=NULL*/)
        : CDialog(CRtDlg::IDD, pParent)
{
        //{{AFX_DATA_INIT(CRtDlg)
                // NOTE: the ClassWizard will add member initialization here
        //}}AFX_DATA_INIT
}


void CRtDlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CRtDlg)
                // NOTE: the ClassWizard will add DDX and DDV calls here
        //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CRtDlg, CDialog)
        //{{AFX_MSG_MAP(CRtDlg)
        ON_BN_CLICKED(IDC_COPYB, OnCopyb)
        ON_BN_CLICKED(IDC_BROADCASTB, OnBroadcastb)
        ON_WM_RBUTTONDOWN()
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CRtDlg message handlers

   /*


 typedef struct _TV_ITEM {  tvi
    UINT       mask;
    HTREEITEM  hItem;           // item this structure refers to
    UINT       state;
    UINT       stateMask;
    LPSTR      pszText;
    int        cchTextMax;
    int        iImage;
    int        iSelectedImage;
    int        cChildren;
    LPARAM     lParam;          // 32-bit value to associate with item
} TV_ITEM;

    */

BOOL CRtDlg::OnInitDialog()
{
        CDialog::OnInitDialog();
   CString s;
//      int i;

/*
        TV_INSERTSTRUCT TreeCtrlItem;

        TreeCtrlItem.hParent = TVI_ROOT;
        TreeCtrlItem.hInsertAfter = TVI_LAST;
        TreeCtrlItem.item.mask = TVIF_TEXT | TVIF_PARAM;
        TreeCtrlItem.item.pszText = "Fire";
        TreeCtrlItem.item.lParam = 0;
        HTREEITEM hTreeItem1 = m_Property.InsertItem(&TreeCtrlItem);

        TreeCtrlItem.hParent = hTreeItem1;
        TreeCtrlItem.item.pszText = "Decay";
        TreeCtrlItem.item.lParam = 1;
        m_Property.InsertItem(&TreeCtrlItem);

        TreeCtrlItem.item.pszText = "Flammability";
        TreeCtrlItem.item.lParam = 2;
        m_Property.InsertItem(&TreeCtrlItem);

        TreeCtrlItem.item.pszText = "Maximum Heat";
        TreeCtrlItem.item.lParam = 3;
        m_Property.InsertItem(&TreeCtrlItem);


        TreeCtrlItem.hParent = TVI_ROOT;
        TreeCtrlItem.item.pszText = "Render";
        TreeCtrlItem.item.lParam = 0;
        HTREEITEM hTreeItem2 = m_Property.InsertItem(&TreeCtrlItem);
        TreeCtrlItem.hParent = hTreeItem2;
        TreeCtrlItem.item.pszText = "Smoothness";
        TreeCtrlItem.item.lParam = 6;
        m_Property.InsertItem(&TreeCtrlItem);

        m_Property.Expand(hTreeItem1,TVE_EXPAND);
        m_Property.Expand(hTreeItem2,TVE_EXPAND);
*/

/*
        TV_ITEM tvi;
        TV_INSERTSTRUCT tvins;

        tvins.hParent = TVI_ROOT;
    tvins.hInsertAfter = TVI_LAST;//  TVI_FIRST
    tvins.item = tvi;

    tvi.mask = TVIF_TEXT | TVIF_PARAM;
    tvi.hItem = NULL;
    tvi.state = NULL;
    tvi.stateMask = NULL;
    tvi.pszText = "TEXZT";
    tvi.cchTextMax = 6;
    tvi.iImage = NULL;
    tvi.iSelectedImage = NULL;
    tvi.cChildren = NULL;
    tvi.mask = NULL;
    tvi.lParam = NULL;


    HTREEITEM til, ti = NULL;
    CTreeCtrl * CT;

    CT = (CTreeCtrl *) this->GetDlgItem(IDC_RT_TREE);
        til = CT->InsertItem(&tvins);

        tvi.pszText = "TEXZT";
    tvi.lParam = 1;
        ti = CT->InsertItem(&tvins);

        tvi.pszText = "TEXZT";
    tvi.lParam = 2;
        ti = CT->InsertItem(&tvins);
        CT->Expand(til,TVE_EXPAND);

         */
                        // GetDlgItem(IDC_COPYB)->ShowWindow(SW_SHOW);
    s.Format("%s's Routing Table (rev %d/%d)",g_node,g_smhead.rt_ver,g_smhead.rt_
    SetDlgItemText(IDC_RT_TITLE,s);
    if (g_remote_node) {
                GetDlgItem(IDC_COPYB)->EnableWindow(TRUE);
                GetDlgItem(IDC_BROADCASTB)->EnableWindow(FALSE);
        } else {
                GetDlgItem(IDC_COPYB)->EnableWindow(FALSE);
                GetDlgItem(IDC_BROADCASTB)->EnableWindow(TRUE);
        }

        CListBox* lb = (CListBox*) GetDlgItem(IDC_TREELB);
    lb->ResetContent();
```

```
CString indent;
SMBUF b;
lpRT  rt = RTROOT;

      // Get the RT Data
if (g_remote_node) {
   memset(&b.mdata,0,(sizeof(b.mdata)));
   QsendAndReceive(g_q,QNETDREQ_MODE,NETMAN_RT_READ, 0,0,0,
      (sizeof(b.mdata)),(char*) &b.mdata,0,0);
   rt = (RT*) &b.mdata;
} else
   rt = RTROOT;

      // Load the box
while(rt = NextRT(rt)) {
   s.Format("%-12s %s",RT_NODE(rt),RT_NTYPE(rt));
   lb->InsertString(-1,s);
   char t, *e, *s = RT_APPS(rt);
   while (strlen(s) > 2 ) {

      if (*s == '[') {
         indent = "      "; s++;
      } else if (*s == ']') {
         indent = "           "; s++;
      } else if (*s == ',') {
         indent = "           "; s++;
      } else {

         if  (e = strpbrk(s,",]")) {
            t = *e;  *e = NULL;
            lb->InsertString(-1,indent + s);
            *e = t;
            s = e;
         } else {
            break; // should never happen
         }
      }
   }
}

/*
      // Load the sockets box
lb = (CListBox*) GetDlgItem(IDC_CONLB);
lb->ResetContent();

if (g_remote_node) {
   memset(&b.mdata,0,(sizeof(b.mdata)));
   QsendAndReceive(g_q,QNETDREQ_MODE,NETMAN_SOCKETS, 0,0,0,
      (sizeof(b.mdata)),(char*) &b.mdata,0,0);
   lpSSA sa = (SSA*) &b.mdata;

            for (i = 0;i < sa->sockets; i++) {
                    s.Format("%x %d",sa->ss[i].ip, sa->ss[i].port);
                    lb->InsertString(-1,indent);
            }

} else { // local
/*
   lpST stp = STroot;
```

```
      while (stp) {
                      s.Format("%x %d",stp->ip, stp->port);
                lb->InsertString(-1,indent);
      }

        }
*/
   return TRUE;  // return TRUE unless you set the focus to a control
                      // EXCEPTION: OCX Property Pages should return FALSE
}


void CRtDlg::OnCopyb()
{
   GetDlgItem(IDC_COPYB)->EnableWindow(FALSE);
   if (g_q)
      if (CopyRT(g_q->msgh.to_node))
         GetDlgItem(IDC_BROADCASTB)->EnableWindow(TRUE);
   GetDlgItem(IDC_COPYB)->EnableWindow(TRUE);

}

void CRtDlg::OnBroadcastb()
{
        GetDlgItem(IDC_BROADCASTB)->EnableWindow(FALSE);
   BroadcastRT();
        GetDlgItem(IDC_BROADCASTB)->EnableWindow(TRUE);
}

void CRtDlg::OnRButtonDown(UINT nFlags, CPoint point)
{
   GetParentFrame()->SetMessageText("");
   this->Invalidate();
        CDialog::OnRButtonDown(nFlags, point);
}
```

```
// rtdlg.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CRtDlg dialog

class CRtDlg : public CDialog
{
// Construction
public:
	CRtDlg(CWnd* pParent = NULL);	// standard constructor

// Dialog Data
	//{{AFX_DATA(CRtDlg)
	enum { IDD = IDD_RTDLG };
		// NOTE: the ClassWizard will add data members here
	//}}AFX_DATA


// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CRtDlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);	// DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:

	// Generated message map functions
	//{{AFX_MSG(CRtDlg)
	virtual BOOL OnInitDialog();
	afx_msg void OnCopyb();
	afx_msg void OnBroadcastb();
	afx_msg void OnRButtonDown(UINT nFlags, CPoint point);
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};
```

```
// stdafx.cpp : source file that includes just the standard includes
//      netman.pch will be the pre-compiled header
//      stdafx.obj will contain the pre-compiled type information

#include "stdafx.h"
```

```
// stdafx.h : include file for standard system include files,
//  or project specific include files that are used frequently, but
//      are changed infrequently
//

#include <afxwin.h>         // MFC core and standard components
#include <afxext.h>         // MFC extensions
#include <afxcmn.h>         // MFC TREE
```

```
// dbdlg.cpp : implementation file
//

#include "stdafx.h"


//#include "oentrvw.h"




#include "oentry.h"
#include "dbdlg.h"
#include "Odlg.h"

// #define ORACLE causes oraread() orawrite() to be exteraly defined
#ifndef ORACLE
#define ORACLE
#endif
#include "oraomq.h"


#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif
#define DB_TIMER 200
#define INIT_TIMER 10
#define RESTOCK_QTY 100000

int g_orders_mode = 0; // Display orders or items
extern CString g_item[];
extern CString g_cust[];
extern int g_price[6];
extern int g_qty[6];
       int g_qty_old[6];
extern int g_purchases[6];
       int g_purchases_old[6];
extern int g_num_purchases[6];
       int g_num_purchases_old[6];
extern int g_total_sales;
       int g_total_sales_old;
extern int g_db_run;
extern int g_ora_state;
       int g_ora_state_old = -1;
extern CFont g_title_font;
extern CFont g_text_font;

enum dbIDC {qty_IDC,price_IDC,item_IDC};

int  ALL_TEXT_DB[] = {IDC_ORDERS_ITEMS,IDC_DB_REFILL,IDC_DB_BOX,IDOK ,
                      IDC_T11,IDC_T12,IDC_T13,
                      IDC_T21,IDC_T22,IDC_T23,
                      IDC_DB_Q0,IDC_DB_Q1,IDC_DB_Q2,IDC_DB_Q3,IDC_DB_Q4,IDC_DB_Q5
                      IDC_DB_P0,IDC_DB_P1,IDC_DB_P2,IDC_DB_P3,IDC_DB_P4,IDC_DB_P5
                      IDC_DB_I0,IDC_DB_I1,IDC_DB_I2,IDC_DB_I3,IDC_DB_I4,IDC_DB

int  g_IDCt[2][3] ={{IDC_T11,IDC_T12,IDC_T13},
                    {IDC_T21,IDC_T22,IDC_T23}};
int  g_IDCs[3][6] = {{IDC_DB_Q0,IDC_DB_Q1,IDC_DB_Q2,IDC_DB_Q3,IDC_DB_Q4,IDC_DB_Q
```

```
                        {IDC_DB_P0,IDC_DB_P1,IDC_DB_P2,IDC_DB_P3,IDC_DB_P4,IDC_DB_P5
                        {IDC_DB_I0,IDC_DB_I1,IDC_DB_I2,IDC_DB_I3,IDC_DB_I4,IDC_DB_I5
CString  g_item_tites[] = {"Qty","Price","Item"};
CString  g_order_titles[] = {"#","Amt","Customer"};



/////////////////////////////////////////////////////////////////////////////
// CDbDlg dialog


CDbDlg::CDbDlg(CWnd* pParent /*=NULL*/)
        : CDialog(CDbDlg::IDD, pParent)
{
        //{{AFX_DATA_INIT(CDbDlg)
                // NOTE: the ClassWizard will add member initialization here
        //}}AFX_DATA_INIT
}


void CDbDlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CDbDlg)
                // NOTE: the ClassWizard will add DDX and DDV calls here
        //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CDbDlg, CDialog)
        //{{AFX_MSG_MAP(CDbDlg)
        ON_BN_CLICKED(IDC_ORDERS_ITEMS, OnOrdersItems)
        ON_WM_TIMER()
        ON_WM_CREATE()
        ON_WM_DESTROY()
        ON_BN_CLICKED(IDC_DB_REFILL, OnDbRefill)
        ON_WM_RBUTTONDOWN()
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CDbDlg message handlers

void CDbDlg::OnOrdersItems()
{
   int i,j;
   CString s;
        if (g_orders_mode) { // Go into Items mode
      g_orders_mode = 0;
      SetDlgItemText(IDC_DB_REFILL,"Refill");

      // Set DB titles
          SetDlgItemText(IDC_DB_BOX,"Database Items");
          SetDlgItemText(IDC_ORDERS_ITEMS,"Orders");
      for (i=0;i<2;i++)
         for (j=0;j<3;j++)
                SetDlgItemText(g_IDCt[i][j],g_item_tites[j]);
      // Set DB values
      for (i=0; i<6 ;i++){ // For each item
         SetDlgItemInt(g_IDCs[qty_IDC][i]   ,g_qty[i]);
         s.Format("$%d.00",g_price[i]);
         SetDlgItemText(g_IDCs[price_IDC][i],s);
         SetDlgItemText(g_IDCs[item_IDC][i],g_item[i]);
      }

        } else { // Go into Orders mode
      g_orders_mode = 1;
      SetDlgItemText(IDC_DB_REFILL,"Clear");

      // Set DB titles
           SetDlgItemText(IDC_DB_BOX,"Database Orders");
           SetDlgItemText(IDC_ORDERS_ITEMS,"Items");
      for (i=0;i<2;i++)
         for (j=0;j<3;j++)
                 SetDlgItemText(g_IDCt[i][j],g_order_titles[j]);
      // Set DB values
      for (i=0; i<6 ;i++){ // For each item
         SetDlgItemInt(g_IDCs[qty_IDC][i]   ,g_num_purchases[i]);
         SetDlgItemInt(g_IDCs[price_IDC][i],g_purchases[i]);
         SetDlgItemText(g_IDCs[item_IDC][i],g_cust[i]);
      }

   }
   for (i=0; i<6 ;i++){ // Invalidate any histoy
      g_num_purchases_old[i] =
      g_purchases_old[i] =
      g_qty_old[i] = -1;
   }
}

void CDbDlg::OnTimer(UINT nIDEvent)
{
   int i,rc,price,stock,cust_orders,cust_sales;
   CString s;

        if (nIDEvent == DB_TIMER) {
      if (g_db_run < 20) {
         if (g_db_run == 0) this->DestroyWindow();
         if (g_db_run == 1) {g_orders_mode = 1; OnOrdersItems(); g_db_run = 20;}
      }
      if (g_ora_state) {  // oracle db
         if (g_orders_mode){

            for (i=0; i<6 ;i++){ // For each item
               if (  (rc = oracustr(g_cust[i].GetBuffer(0), &cust_orders, &cust_
                  s.Format("OraCustRead ErroR %d",rc);
                  GetParentFrame()->SetMessageText(s);
               }
               if (g_num_purchases_old[i] != cust_orders)
                  SetDlgItemInt(g_IDCs[qty_IDC][i]   ,(g_num_purchases_old[i] = c
               if (g_purchases_old[i] != cust_sales)
                  SetDlgItemInt(g_IDCs[price_IDC][i],(g_purchases_old[i] = cust
            }
         } else {

            for (i=0; i<6 ;i++){ // For each item
               if (  (rc = oraread(g_item[i].GetBuffer(0), &price, &stock))   ) {
                s.Format("OraRead ErroR %d",rc);
                GetParentFrame()->SetMessageText(s);
             }
             if (g_qty_old[i] != stock)
                SetDlgItemInt(g_IDCs[qty_IDC][i]  ,(g_qty_old[i] = stock));
          }
       }
    } else if (g_orders_mode){ // NOT ora_state, so use the local db
       for (i=0; i<6 ;i++){ // For each item
          if (g_num_purchases_old[i] != g_num_purchases[i])
             SetDlgItemInt(g_IDCs[qty_IDC][i]  ,(g_num_purchases_old[i] = g_nu
          if (g_purchases_old[i] != g_purchases[i])
             SetDlgItemInt(g_IDCs[price_IDC][i],(g_purchases_old[i] = g_purcha
       }
    } else {   // local db
       for (i=0; i<6 ;i++){ // For each item
          if (g_qty[i] != g_qty_old[i])
          SetDlgItemInt(g_IDCs[qty_IDC][i]  ,(g_qty_old[i] = g_qty[i]));
       }
    }
    if (g_total_sales_old != g_total_sales)
       SetDlgItemInt(IDC_DB_SALES,(g_total_sales_old = g_total_sales));



    if (g_ora_state_old != g_ora_state){
       if (g_ora_state_old = g_ora_state)
          SetDlgItemText(IDC_BIG_TITLE,"Oracle");
       else
          SetDlgItemText(IDC_BIG_TITLE,"Local DB");
    }
 } else if (nIDEvent == INIT_TIMER) {
    KillTimer(INIT_TIMER);

         GetDlgItem(IDC_BIG_TITLE)->SetFont(&g_title_font);

    if (g_ora_state)
       SetDlgItemText(IDC_BIG_TITLE,"Oracle");
    else
       SetDlgItemText(IDC_BIG_TITLE,"Local DB");

 } else

       CDialog::OnTimer(nIDEvent);
}

int CDbDlg::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
        if (CDialog::OnCreate(lpCreateStruct) == -1)
                return -1;


   SetTimer(DB_TIMER,200,NULL);
   SetTimer(INIT_TIMER,200,NULL);
   g_db_run = 1; // Start

   return 0;
}

void CDbDlg::OnDestroy()
```

```
{
        g_db_run = 40;

        CDialog::OnDestroy();

   KillTimer(DB_TIMER);
}

void CDbDlg::OnOK()
{
        g_db_run = 40; // Re enable the show db call button
        this->DestroyWindow();

        // CDialog::OnOK();
}

void CDbDlg::OnDbRefill()
{
        int i,rc;
   CString s;

   if (g_ora_state) {
      for (i=0;i<6;i++) {
         if (g_orders_mode) {
            if (  (rc = oracustw(g_cust[i].GetBuffer(0), 0, 0))  ) {
               s.Format("OraCustWrite ERRor %d",rc);
               GetParentFrame()->SetMessageText(s);
            }
         } else { // in items mode
            if (rc = orawrite(g_item[i].GetBuffer(0), RESTOCK_QTY)) {
               s.Format("OraWrite ERRor %d",rc);
               GetParentFrame()->SetMessageText(s);
            }
         }
      }
   } else {
      for (i=0;i<6;i++) {
         if (g_orders_mode) {
            g_purchases[i] = 0;
            g_num_purchases[i] = 0;
         } else { // in items mode
            g_qty[i] = RESTOCK_QTY;
         }
      }
   }
}


void CDbDlg::OnRButtonDown(UINT nFlags, CPoint point)
{
   GetParentFrame()->SetMessageText("");
   this->Invalidate();

        CDialog::OnRButtonDown(nFlags, point);
}


BOOL CDbDlg::OnInitDialog()
{
        CDialog::OnInitDialog();

    // Fonts
   int i = 0;
   while (ALL_TEXT_DB[i])
      GetDlgItem(ALL_TEXT_DB[i++])->SetFont(&g_text_font);

        return TRUE;  // return TRUE unless you set the focus to a control
                      // EXCEPTION: OCX Property Pages should return FALSE
}
```

```
// dbdlg.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CDbDlg dialog

class CDbDlg : public CDialog
{
// Construction
public:
	CDbDlg(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
	//{{AFX_DATA(CDbDlg)
	enum { IDD = IDD_DBDLG };
		// NOTE: the ClassWizard will add data members here
	//}}AFX_DATA


// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CDbDlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
	//}}AFX_VIRTUAL


   CFont    m_title_font;
   int      m_was_inited;

   // Implementation
protected:

	// Generated message map functions
	//{{AFX_MSG(CDbDlg)
	afx_msg void OnOrdersItems();
	afx_msg void OnTimer(UINT nIDEvent);
	afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
	afx_msg void OnDestroy();
	virtual void OnOK();
	afx_msg void OnDbRefill();
	afx_msg void OnRButtonDown(UINT nFlags, CPoint point);
	virtual BOOL OnInitDialog();
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};
```

```
// mainfrm.cpp : implementation of the CMainFrame class
//

#include "stdafx.h"
#include "oentry.h"
//#include "OpDlg.h"

#include "mainfrm.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CMainFrame

IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
	//{{AFX_MSG_MAP(CMainFrame)
		// NOTE - the ClassWizard will add and remove mapping macros her
		//    DO NOT EDIT what you see in these blocks of generated code
	ON_WM_CREATE()
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// arrays of IDs used to initialize control bars

// toolbar buttons - IDs are command buttons
static UINT BASED_CODE buttons[] =
{
	// same order as in the bitmap 'toolbar.bmp'
	ID_FILE_NEW,
	ID_FILE_OPEN,
	ID_FILE_SAVE,
		ID_SEPARATOR,
	ID_EDIT_CUT,
	ID_EDIT_COPY,
	ID_EDIT_PASTE,
		ID_SEPARATOR,
	ID_FILE_PRINT,
	ID_APP_ABOUT,
};

static UINT BASED_CODE indicators[] =
{
	ID_SEPARATOR,           // status line indicator
	ID_INDICATOR_CAPS,
	ID_INDICATOR_NUM,
	ID_INDICATOR_SCRL,
};

/////////////////////////////////////////////////////////////////////////////
// CMainFrame construction/destruction

CMainFrame::CMainFrame()
{
	// TODO: add member initialization code here
```

```
}

CMainFrame::~CMainFrame()
{
}

int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
        if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
                return -1;

        if (!m_wndToolBar.Create(this) ||
                !m_wndToolBar.LoadBitmap(IDR_MAINFRAME) ||
                !m_wndToolBar.SetButtons(buttons,
                  sizeof(buttons)/sizeof(UINT)))
        {
                TRACE0("Failed to create toolbar\n");
                return -1;      // fail to create
        }


/* Derek's remove tool bar */
    m_wndToolBar.ShowWindow(SW_HIDE);



        if (!m_wndStatusBar.Create(this) ||
                !m_wndStatusBar.SetIndicators(indicators,
                  sizeof(indicators)/sizeof(UINT)))
        {
                TRACE0("Failed to create status bar\n");
                return -1;      // fail to create
        }

        // TODO: Delete these three lines if you don't want the toolbar to
        //  be dockable
        m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
        EnableDocking(CBRS_ALIGN_ANY);
        DockControlBar(&m_wndToolBar);

        // TODO: Remove this if you don't want tool tips
        m_wndToolBar.SetBarStyle(m_wndToolBar.GetBarStyle() |
                CBRS_TOOLTIPS | CBRS_FLYBY);

        return 0;
}

/////////////////////////////////////////////////////////////////////////////
// CMainFrame diagnostics

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
        CFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{
        CFrameWnd::Dump(dc);
```

```
}

#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CMainFrame message handlers
// mainfrm.h : interface of the CMainFrame class
//
/////////////////////////////////////////////////////////////////////////////

class CMainFrame : public CFrameWnd
{
protected: // create from serialization only
        CMainFrame();
        DECLARE_DYNCREATE(CMainFrame)

// Attributes
public:

// Operations
public:

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CMainFrame)
        //}}AFX_VIRTUAL

// Implementation
public:
        virtual ~CMainFrame();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

protected:  // control bar embedded members
        CStatusBar  m_wndStatusBar;
        CToolBar    m_wndToolBar;

// Generated message map functions
protected:
        //{{AFX_MSG(CMainFrame)
        afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
                // NOTE - the ClassWizard will add and remove member functions h
                //    DO NOT EDIT what you see in these blocks of generated code
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////
```

```
// Odlg.cpp : implementation file
//

#include "stdafx.h"
#include "oentry.h"
#include "Odlg.h"



// #define ORACLE causes oraread() orawrite() to be exteraly defined
#ifndef ORACLE
#define ORACLE
#endif
#include "oraomq.h"



#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

extern int g_options_run;
extern int g_fill_delay;
extern int g_place_delay;
extern int g_place_tpm;
extern int g_poll_pps;
extern int g_poll_delay;
extern int g_clear_stats;
extern int g_ora_state;
extern COLORREF g_new_color;
extern char g_oracle_con_str[80];
extern CFont g_text_font;

int ALL_TEXT_O[] = { IDOK,IDC_COLOR,IDC_CLRSTATS,IDC_ORACREATE,
                     IDC_FILLBOX,IDC_DLY_EB,IDC_DLYMAX,IDC_DLYMIN,IDC_DLY_LAB,
                     IDC_POLL_BOX,IDC_POLL_EB,IDC_POLLMAX,IDC_POLLMIN,IDC_POLL_L
                     IDC_AUTOBOX,IDC_AUTO_EB,IDC_AUTOMAX,IDC_AUTOMIN,IDC_AUTO_LA
/////////////////////////////////////////////////////////////////////////////
// COdlg dialog


COdlg::COdlg(CWnd* pParent /*=NULL*/)
        : CDialog(COdlg::IDD, pParent)
{
        //{{AFX_DATA_INIT(COdlg)
                // NOTE: the ClassWizard will add member initialization here
        //}}AFX_DATA_INIT
}


void COdlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(COdlg)
                // NOTE: the ClassWizard will add DDX and DDV calls here
        //}}AFX_DATA_MAP
}
```

```
BEGIN_MESSAGE_MAP(COdlg, CDialog)
        //{{AFX_MSG_MAP(COdlg)
        ON_WM_CREATE()
        ON_EN_UPDATE(IDC_AUTO_EB, OnUpdateAutoEb)
        ON_WM_VSCROLL()
        ON_EN_UPDATE(IDC_POLL_EB, OnUpdatePollEb)
        ON_EN_UPDATE(IDC_DLY_EB, OnUpdateDlyEb)
        ON_BN_CLICKED(IDC_COLOR, OnColor)
        ON_BN_CLICKED(IDC_CLRSTATS, OnClrstats)
        ON_BN_CLICKED(IDC_ORACREATE, OnOracreate)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

//////////////////////////////////////////////////////////////////////////////
// COdlg message handlers

int COdlg::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
        if (CDialog::OnCreate(lpCreateStruct) == -1)
                return -1;

        g_options_run = 1;  // start
        return 0;
}

void COdlg::OnOK()
{
   g_options_run = 40;  // Re-enable the options call button

   this->ShowWindow(SW_HIDE);

   //   this->DestroyWindow();
//      CDialog::OnOK();
}

 /*

void COdlg::OnDestroy()
{
        g_options_run = 40;  // Re-enable the options call button
  // CDialog::OnDestroy();


}

*/




void COdlg::OnVScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar)
{
   /*
   if ((nSBCode == SB_PAGEUP) || (nSBCode == SB_PAGEDOWN)) {
      if (pScrollBar->GetDlgCtrlID() == IDC_DLY_SLD)
         pScrollBar->SetScrollPos(MAX_DLY + MIN_DLY - g_fill_delay,TRUE);
   }
   */
```

```
    if (nSBCode == SB_ENDSCROLL) {
        ;
    } else if (pScrollBar->GetDlgCtrlID() == IDC_DLY_SLD) {
            if (((int)nPos != g_fill_delay) && (MIN_DLY <= nPos) && (nPos <= M
          SetDlgItemInt(IDC_DLY_EB,g_fill_delay = MAX_DLY + MIN_DLY - nPos);
        }
    } else if (pScrollBar->GetDlgCtrlID() == IDC_AUTO_SLD) {
            if (((int)nPos != g_place_delay) && (MIN_AUTO <= nPos) && (nPos <= MA
          SetDlgItemInt(IDC_AUTO_EB,(g_place_tpm = MAX_AUTO + MIN_AUTO - nPos));
          g_place_delay = 60000/g_place_tpm;
        }
    } else if (pScrollBar->GetDlgCtrlID() == IDC_POLL_SLD) {
            if (((int)nPos != g_poll_pps) && (MIN_POLL <= nPos) && (nPos <= MAX_P
          SetDlgItemInt(IDC_POLL_EB,(g_poll_pps = MAX_POLL + MIN_POLL - nPos));
          g_poll_delay = 1000/g_poll_pps;
        }
    }
          CDialog::OnVScroll(nSBCode, nPos, pScrollBar);
}

BOOL COdlg::OnInitDialog()
{
          CDialog::OnInitDialog();

   // GetParentFrame()->SetWindowText(m_inst + " Options");
    this->SetWindowText(m_inst + "  Options");


          SetDlgItemInt(IDC_DLYMAX,MAX_DLY);
          SetDlgItemInt(IDC_DLYMIN,MIN_DLY);

    SetDlgItemInt(IDC_AUTOMAX,MAX_AUTO);
          SetDlgItemInt(IDC_AUTOMIN,MIN_AUTO);

    SetDlgItemInt(IDC_POLLMAX,MAX_POLL);
          SetDlgItemInt(IDC_POLLMIN,MIN_POLL);

    SetDlgItemInt(IDC_DLY_EB,g_fill_delay);
    SetDlgItemInt(IDC_POLL_EB,g_poll_pps);
    SetDlgItemInt(IDC_AUTO_EB,g_place_tpm);

   // (CSliderCtrl *) xxx =  GetDlgItem(IDC_DLY_SLD);
    HWND hTrack = GetDlgItem(IDC_DLY_SLD)->m_hWnd;
    ::SendMessage(hTrack,TBM_SETRANGEMIN,TRUE,MIN_DLY);// MINDLY
    ::SendMessage(hTrack,TBM_SETRANGEMAX,TRUE,MAX_DLY);// MAXDLY
    ::SendMessage(hTrack,TBM_SETTICFREQ,100,TRUE); // 10 ticks   (MAXDLY - MINDLY
    ::SendMessage(hTrack,TBM_SETPOS,TRUE,MIN_DLY + MAX_DLY - g_fill_delay); // 1

     hTrack = GetDlgItem(IDC_AUTO_SLD)->m_hWnd;
    ::SendMessage(hTrack,TBM_SETRANGEMIN,TRUE,MIN_AUTO);// MINDLY
    ::SendMessage(hTrack,TBM_SETRANGEMAX,TRUE,MAX_AUTO);// MAXDLY
    ::SendMessage(hTrack,TBM_SETTICFREQ,500,TRUE); // 10 ticks   (MAXDLY - MINDLY
    ::SendMessage(hTrack,TBM_SETPOS,TRUE,MAX_AUTO + MIN_AUTO - g_place_tpm); //

     hTrack = GetDlgItem(IDC_POLL_SLD)->m_hWnd;
    ::SendMessage(hTrack,TBM_SETRANGEMIN,TRUE,MIN_POLL);// MINDLY
    ::SendMessage(hTrack,TBM_SETRANGEMAX,TRUE,MAX_POLL);// MAXDLY
    ::SendMessage(hTrack,TBM_SETTICFREQ,10,TRUE); // 10 ticks   (MAXDLY - MINDLY)
   ::SendMessage(hTrack,TBM_SETPOS,TRUE,MAX_POLL + MIN_POLL - g_poll_pps); //  1


   // Fonts
   int i = 0;
   while (ALL_TEXT_O[i])
      GetDlgItem(ALL_TEXT_O[i++])->SetFont(&g_text_font);



   return TRUE;  // return TRUE unless you set the focus to a control
                     // EXCEPTION: OCX Property Pages should return FALSE
}


void COdlg::OnUpdateAutoEb()
{
   int tpm = GetDlgItemInt(IDC_AUTO_EB,NULL,TRUE);
   if ((g_place_tpm != tpm) && (tpm >= MIN_AUTO) && (tpm <= MAX_AUTO)) {
      g_place_delay = 60000/tpm;
      g_place_tpm = tpm;
      HWND hTrack = GetDlgItem(IDC_AUTO_SLD)->m_hWnd;
      ::SendMessage(hTrack,TBM_SETPOS,TRUE,MAX_AUTO + MIN_AUTO - tpm);

   }
}
void COdlg::OnUpdatePollEb()
{
   int poll = GetDlgItemInt(IDC_POLL_EB,NULL,TRUE);
   if ((g_place_tpm != poll) && (poll >= MIN_AUTO) && (poll <= MAX_AUTO)) {
      g_poll_delay = 1000/poll;
      g_poll_pps = poll;
      HWND hTrack = GetDlgItem(IDC_POLL_SLD)->m_hWnd;
      ::SendMessage(hTrack,TBM_SETPOS,TRUE,MAX_POLL + MIN_POLL - poll);
   }

}
void COdlg::OnUpdateDlyEb()
{
   int dly = GetDlgItemInt(IDC_DLY_EB,NULL,TRUE);
   if ((g_fill_delay != dly) && (dly >= MIN_DLY) && (dly <= MAX_DLY)) {
      g_fill_delay = dly;
      HWND hTrack = GetDlgItem(IDC_DLY_SLD)->m_hWnd;
      ::SendMessage(hTrack,TBM_SETPOS,TRUE,MAX_DLY + MIN_DLY - dly);
   }

}


void COdlg::OnColor()
{
   CHOOSECOLOR cc;      // common dialog box structure
   COLORREF acrCustClr[16];

   // Setup the custom colors as a grey scale
   for (int v=0,i=0; i < 16; v=17 * i++)
      acrCustClr[i] = RGB(v,v,v);
   // Initialize the necessary members.
   cc.lStructSize = sizeof(CHOOSECOLOR);
   cc.hwndOwner = NULL; //  = hwnd;
   cc.lpCustColors = (LPDWORD) acrCustClr;
   cc.Flags = CC_FULLOPEN;  //  CC_PREVENTFULLOPEN


   if (ChooseColor(&cc)){
      g_new_color = cc.rgbResult; // lpCustColors
   } else {
      GetParentFrame()->SetMessageText("Color was not changed");
   }
}


void COdlg::OnClrstats()
{
        g_clear_stats++;

}

void COdlg::OnOracreate()
{
// Create the database in oracle
   int org_ora_state =  g_ora_state;


   if (g_ora_state == 0)
      if(oraconn(g_oracle_con_str))
         MessageBox("Oracle Connect Failed");
      else
         g_ora_state = 1;

   if (g_ora_state) {

      if(oracreate())
         MessageBox("Oracle oracreate Failed");

      if (org_ora_state == 0)
      if(oradisc())
         MessageBox("Oracle DisConnect Failed");
      else
         g_ora_state = 0;
   }
}
```

```
// Odlg.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// COdlg dialog

class COdlg : public CDialog
{
// Construction
public:
	COdlg(CWnd* pParent = NULL);	// standard constructor

// Dialog Data
	//{{AFX_DATA(COdlg)
	enum { IDD = IDD_O_DLG };
		// NOTE: the ClassWizard will add data members here
	//}}AFX_DATA
   CString m_inst;
//    COentryView* m_parentptr;

// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(COdlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:

	// Generated message map functions
	//{{AFX_MSG(COdlg)
	afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
	afx_msg void OnUpdateAutoEb();
	afx_msg void OnVScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar);
	virtual BOOL OnInitDialog();
	afx_msg void OnUpdatePollEb();
	afx_msg void OnUpdateDlyEb();
	afx_msg void OnColor();
	virtual void OnOK();
	afx_msg void OnClrstats();
	afx_msg void OnOracreate();
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};
```

```
User: root
Host: bunny
Class: bunny
Job: stdin
```

```
// oentrdoc.cpp : implementation of the COentryDoc class
//

#include "stdafx.h"
#include "oentry.h"
//#include "OpDlg.h"

#include "oentrdoc.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// COentryDoc

IMPLEMENT_DYNCREATE(COentryDoc, CDocument)

BEGIN_MESSAGE_MAP(COentryDoc, CDocument)
        //{{AFX_MSG_MAP(COentryDoc)
                // NOTE - the ClassWizard will add and remove mapping macros her
                //    DO NOT EDIT what you see in these blocks of generated code
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// COentryDoc construction/destruction

COentryDoc::COentryDoc()
{
        // TODO: add one-time construction code here

}

COentryDoc::~COentryDoc()
{
}

BOOL COentryDoc::OnNewDocument()
{
        if (!CDocument::OnNewDocument())
                return FALSE;

        // TODO: add reinitialization code here
        // (SDI documents will reuse this document)

        return TRUE;
}

/////////////////////////////////////////////////////////////////////////////
// COentryDoc serialization

void COentryDoc::Serialize(CArchive& ar)
{
        if (ar.IsStoring())
        {
                // TODO: add storing code here
        }
        else
        {
                // TODO: add loading code here
        }
}

/////////////////////////////////////////////////////////////////////////////
// COentryDoc diagnostics

#ifdef _DEBUG
void COentryDoc::AssertValid() const
{
        CDocument::AssertValid();
}

void COentryDoc::Dump(CDumpContext& dc) const
{
        CDocument::Dump(dc);
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// COentryDoc commands
// oentrdoc.h : interface of the COentryDoc class
//
/////////////////////////////////////////////////////////////////////////////

class COentryDoc : public CDocument
{
protected: // create from serialization only
        COentryDoc();
        DECLARE_DYNCREATE(COentryDoc)

// Attributes
public:

// Operations
public:

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(COentryDoc)
        public:
        virtual BOOL OnNewDocument();
        //}}AFX_VIRTUAL

// Implementation
public:
        virtual ~COentryDoc();
        virtual void Serialize(CArchive& ar);   // overridden for document i/o
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
        //{{AFX_MSG(COentryDoc)
                // NOTE - the ClassWizard will add and remove member functions h
                //    DO NOT EDIT what you see in these blocks of generated code
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////
```

```
// oentrvw.h : interface of the COentryView class
//
/////////////////////////////////////////////////////////////////////////////
#define MESSAGEREADY WM_USER + 3000
//#define IDD_TPSTIMER          100

class COentryView : public CFormView
{
protected: // create from serialization only
        COentryView();
        DECLARE_DYNCREATE(COentryView)

public:
        //{{AFX_DATA(COentryView)
        enum { IDD = IDD_OENTRY_FORM };
        int             m_OrderMode;
        CString m_inst;
        BOOL    m_sendreply;
        //}}AFX_DATA
   BOOL     m_runflag;  // Running or not
   int      m_millisec; // Delay when ordering
   int      m_mes_sent;
   int      m_mes_rec;
   int      m_rec_sent;
   int      m_rec_rec;

   HBRUSH   m_brush;
   COLORREF m_color;
   COLORREF m_box_color;
   CString  m_que;
   CRect    m_color_box_rec;


   int      m_order;
   int      m_order_num;
   int      m_tran_state;
   int      m_tran_rec;
   int      m_tran_sent;
   int      m_auto_tran;
   int      m_auto_rand;
   int      m_auto_commit;
   void     AutoRun();
   void     color_the_box(COLORREF c, BOOL update);
   void     CmdLine(int pass);
   void     OnCommit(int action);
   void     OnAnyUserAction();
   void     OnColor();
   void     ShowRateBar(int act);
   void     PlaceOrFillMode(int mode);
   int      DbOrder(char *cust, char *item, int qty);
   int      OraOrder(char *cust, char *item, int qty);
   void     LoadQList(int mode);
   void     ClearDisplay();

   CDbDlg dbd;
   COdlg  o_dlg;
```

```
// Attributes
```

```
public:
        COentryDoc* GetDocument();

// Operations
public:

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(COentryView)
        public:
        virtual void OnInitialUpdate();
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
        virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
        virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);
        virtual void OnPrint(CDC* pDC, CPrintInfo*);
        virtual void OnDraw(CDC* pDC);
        //}}AFX_VIRTUAL

// Implementation
public:
        virtual ~COentryView();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
        //{{AFX_MSG(COentryView)
        afx_msg void OnExitb();
        afx_msg void OnOrderb();
        afx_msg void OnAutob();
        afx_msg HBRUSH OnCtlColor(CDC* pDC, CWnd* pWnd, UINT nCtlColor);
        afx_msg void OnPlacer();
        afx_msg void OnFillr();
        afx_msg void OnSelchangeQue();
        afx_msg long OnReplyMsg(WPARAM  wParam, LPARAM  lParam);
        afx_msg void OnSendrepc();
        afx_msg void OnTranb();
        afx_msg void OnCommitb();
        afx_msg void OnAbortb();
        afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
        afx_msg void OnTimer(UINT nIDEvent);
        afx_msg void OnShowdb();
        afx_msg void OnFilldb();
        afx_msg void OnPlacenoqr();
        afx_msg void OnEditupdateQue();
        afx_msg void OnOptionsb();
        afx_msg void OnRButtonDown(UINT nFlags, CPoint point);
        afx_msg void OnSetfocusQue();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

#ifndef _DEBUG  // debug version in oentrvw.cpp
inline COentryDoc* COentryView::GetDocument()
```

```
    { return (COentryDoc*)m_pDocument; }
#endif

////////////////////////////////////////////////////////////////////////////
```

```
// oentry.cpp : Defines the class behaviors for the application.
//

#include "stdafx.h"
#include "oentry.h"

#include "mainfrm.h"
#include "oentrdoc.h"
#include "dbdlg.h"
#include "Odlg.h"
#include "oentrvw.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// COentryApp

BEGIN_MESSAGE_MAP(COentryApp, CWinApp)
        //{{AFX_MSG_MAP(COentryApp)
        ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
                // NOTE - the ClassWizard will add and remove mapping macros her
                //    DO NOT EDIT what you see in these blocks of generated code
        //}}AFX_MSG_MAP
        // Standard file based document commands
        ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
        ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
        // Standard print setup command
        ON_COMMAND(ID_FILE_PRINT_SETUP, CWinApp::OnFilePrintSetup)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// COentryApp construction

COentryApp::COentryApp()
{
        // TODO: add construction code here,
        // Place all significant initialization in InitInstance
}

/////////////////////////////////////////////////////////////////////////////
// The one and only COentryApp object

COentryApp theApp;

/////////////////////////////////////////////////////////////////////////////
// COentryApp initialization

BOOL COentryApp::InitInstance()
{
        // Standard initialization
        // If you are not using these features and wish to reduce the size
        //  of your final executable, you should remove from the following
        //  the specific initialization routines you do not need.

        Enable3dControls();

        LoadStdProfileSettings();  // Load standard INI file options (including
        // Register the application's document templates.  Document templates
        //  serve as the connection between documents, frame windows and views.


        CSingleDocTemplate* pDocTemplate;
        pDocTemplate = new CSingleDocTemplate(
                IDR_MAINFRAME,
                RUNTIME_CLASS(COentryDoc),
                RUNTIME_CLASS(CMainFrame),       // main SDI frame window
                RUNTIME_CLASS(COentryView));
        AddDocTemplate(pDocTemplate);

        // create a new (empty) document

     OnFileNew();

        if (m_lpCmdLine[0] != '\0')
        {
                // TODO: add command line processing here
        }

        return TRUE;
}

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
        CAboutDlg();

// Dialog Data
        //{{AFX_DATA(CAboutDlg)
        enum { IDD = IDD_ABOUTBOX };
        //}}AFX_DATA


      CFont m_title_font;


// Implementation
protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        //{{AFX_MSG(CAboutDlg)
        virtual BOOL OnInitDialog();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
        //{{AFX_DATA_INIT(CAboutDlg)
        //}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CAboutDlg)
        //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
        //{{AFX_MSG_MAP(CAboutDlg)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

// App command to run the dialog
void COentryApp::OnAppAbout()
{
        CAboutDlg aboutDlg;
        aboutDlg.DoModal();
}

/////////////////////////////////////////////////////////////////////////////
// COentryApp commands

BOOL CAboutDlg::OnInitDialog()
{
        CDialog::OnInitDialog();



   LOGFONT lf;
   memset(&lf,0,sizeof(LOGFONT));
        strcpy(lf.lfFaceName,"Monotype Corsiva");
   lf.lfHeight = 24;
   m_title_font.CreateFontIndirect(&lf);
        GetDlgItem(IDC_ABOUT1)->SetFont(&m_title_font);



        return TRUE;  // return TRUE unless you set the focus to a control
                      // EXCEPTION: OCX Property Pages should return FALSE
}
```

```
// oentry.h : main header file for the OENTRY application
//
#define MIN_DLY  0
#define MAX_DLY  5000

#define MIN_AUTO 2
#define MAX_AUTO 3000

#define MIN_POLL 1
#define MAX_POLL 100

#ifndef __AFXWIN_H__
	#error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"       // main symbols

/////////////////////////////////////////////////////////////////////////////
// COentryApp:
// See oentry.cpp for the implementation of this class
//

class COentryApp : public CWinApp
{
public:
	COentryApp();

// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(COentryApp)
	public:
	virtual BOOL InitInstance();
	//}}AFX_VIRTUAL

// Implementation

	//{{AFX_MSG(COentryApp)
	afx_msg void OnAppAbout();
		// NOTE - the ClassWizard will add and remove member functions h
		//    DO NOT EDIT what you see in these blocks of generated code
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};


/////////////////////////////////////////////////////////////////////////////
```

```
// OPDLG.cpp : implementation file
//

#include "stdafx.h"
#include "oentry.h"
#include "OPDLG.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

extern int g_options_run;


/////////////////////////////////////////////////////////////////////////////
// OPDLG dialog


OPDLG::OPDLG(CWnd* pParent /*=NULL*/)
	: CDialog(OPDLG::IDD_O_DLG, pParent)
{
	//{{AFX_DATA_INIT(OPDLG)
		// NOTE: the ClassWizard will add member initialization here
	//}}AFX_DATA_INIT
}


void OPDLG::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(OPDLG)
		// NOTE: the ClassWizard will add DDX and DDV calls here
	//}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(OPDLG, CDialog)
	//{{AFX_MSG_MAP(OPDLG)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// OPDLG message handlers

void OPDLG::OnOK()
{
	g_options_run = 40;  // Re-enable the options call button
	this->DestroyWindow();
	CDialog::OnOK();
}
```

```
// OPDLG.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// OPDLG dialog

class OPDLG : public CDialog
{
// Construction
public:
        OPDLG(CWnd* pParent = NULL);    // standard constructor

// Dialog Data
        //{{AFX_DATA(OPDLG)
//      enum { IDD = IDD_OP_DLG };
                // NOTE: the ClassWizard will add data members here
        //}}AFX_DATA


// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(OPDLG)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        //}}AFX_VIRTUAL

// Implementation
protected:

        // Generated message map functions
        //{{AFX_MSG(OPDLG)
        virtual void OnOK();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};
```

```
// OptDlg.cpp : implementation file
//

#include "stdafx.h"
#include "oentry.h"
#include "OptDlg.h"


#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif


extern   int  g_options_run ;


/////////////////////////////////////////////////////////////////////////////
// OptDlg dialog


OptDlg::OptDlg(CWnd* pParent /*=NULL*/)
        : CDialog(OptDlg::IDD, pParent)
{
        //{{AFX_DATA_INIT(OptDlg)
                // NOTE: the ClassWizard will add member initialization here
        //}}AFX_DATA_INIT
}


void OptDlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(OptDlg)
                // NOTE: the ClassWizard will add DDX and DDV calls here
        //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(OptDlg, CDialog)
        //{{AFX_MSG_MAP(OptDlg)
        ON_BN_CLICKED(IDC_DONE, OnDone)
        ON_WM_DESTROY()
        ON_WM_CREATE()
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// OptDlg message handlers

void OptDlg::OnDone()
{
        g_options_run = 40;  // Re-enable the options call button
        this->DestroyWindow();
}

void OptDlg::OnDestroy()
{
   g_options_run = 40;  // Re-enable the options call button
```

```
        CDialog::OnDestroy();
}

int OptDlg::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
        if (CDialog::OnCreate(lpCreateStruct) == -1)
                return -1;

   g_options_run = 1;  // Start

        return 0;
}
```

```
// OptDlg.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// OptDlg dialog

class OptDlg : public CDialog
{
// Construction
public:
        OptDlg(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
        //{{AFX_DATA(OptDlg)
//      enum { IDD = IDD_OPTIONS_DLG };
        //}}AFX_DATA


// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(OptDlg)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        //}}AFX_VIRTUAL

// Implementation
protected:

        // Generated message map functions
        //{{AFX_MSG(OptDlg)
        afx_msg void OnDone();
        afx_msg void OnDestroy();
        afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};
```

```
//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.
// Used by oentry.rc
//
#define IDD_ABOUTBOX                    100
#define IDD_OENTRY_FORM                 101
#define IDR_MAINFRAME                   128
#define IDR_OENTRYTYPE                  129
#define IDI_ICON_Q                      130
#define IDB_BITMAPTEST                  131
#define IDD_DBDLG                       133
#define IDD_O_DLG                       138
#define IDR_3DIMES                      145
#define IDR_3DMDS                       146
#define IDI_ICON_TRASH                  147
#define IDI_ICON_TRASH1                 148
#define IDI_WAIT0                       149
#define IDI_WAIT1                       150
#define IDI_WAIT2                       151
#define IDI_WAIT3                       152
#define IDI_FILL0                       153
#define IDI_FILL1                       154
#define IDI_FILL2                       155
#define IDI_WAIT4                       156
#define IDI_WAIT5                       157
#define IDI_WAIT6                       158
#define IDC_AUTOB                       1000
#define IDC_EXITB                       1001
#define IDC_ORDERB                      1002
#define IDC_CUST                        1003
#define IDC_ITEM                        1004
#define IDC_QTY                         1005
#define IDC_PICT                        1006
#define IDC_COLORB                      1006
#define IDC_GENERIC1                    1007
#define IDC_TPS_EB                      1008
#define IDC_TRANB                       1010
#define IDC_COLORBOX                    1011
#define IDC_LOGO_Q                      1012
#define IDC_ABORTB                      1013
#define IDC_COMMITB                     1014
#define IDC_TOTALR                      1015
#define IDC_MSGS                        1015
#define IDC_SHOWDB                      1016
#define IDC_TPS                         1017
#define IDC_QUE                         1018
#define IDC_DLYMIN                      1019
#define IDC_DLYMAX                      1020
#define IDC_PLACER                      1021
#define IDC_AUTOMAX                     1021
#define IDC_PLACENOQR                   1022
#define IDC_AUTOMIN                     1022
#define IDC_QUELAB                      1023
#define IDC_POLLMAX                     1023
#define IDC_CUSTLAB                     1024
#define IDC_POLLMIN                     1024
#define IDC_ITEMLAB                     1025
#define IDC_QTYLAB                      1026
#define IDC_SENDREPC                    1028
#define IDC_RECIPT                      1029
```

```
#define IDC_FILLDB                    1030
#define IDC_RECIPTS                   1031
#define IDC_MSGS_LAB                  1032
#define IDC_RECIPTS_LAB               1033
#define IDC_FILLR3                    1034
#define IDC_DB_I0                     1038
#define IDC_DB_Q0                     1039
#define IDC_DB_SALEST                 1040
#define IDC_DB_SALES                  1041
#define IDC_DB_BOX                    1042
#define IDC_DB_REFILL                 1043
#define IDC_ORDERS_ITEMS              1044
#define IDC_T11                       1045
#define IDC_T12                       1046
#define IDC_T13                       1047
#define IDC_T21                       1048
#define IDC_T22                       1049
#define IDC_DB_P0                     1050
#define IDC_T23                       1051
#define IDC_ORDERBOX                  1052
#define IDC_FILLTXT                   1054
#define IDC_AUTOBOX                   1055
#define IDC_dmd                       1057
#define IDC_POLL_BOX                  1057
#define IDC_TRANBOX                   1058
#define IDC_WAIT0                     1060
#define IDC_WAIT1                     1061
#define IDC_OPTIONSB                  1063
#define IDC_FILL0                     1064
#define IDC_FILL1                     1065
#define IDC_DB_I1                     1066
#define IDC_FILL2                     1066
#define IDC_DB_Q1                     1067
#define IDC_DB_P1                     1068
#define IDC_DLY_EB                    1068
#define IDC_DB_I2                     1069
#define IDC_DLY_SLD                   1069
#define IDC_DB_Q2                     1070
#define IDC_DLY_LAB                   1070
#define IDC_DB_P2                     1071
#define IDC_AUTO_EB                   1071
#define IDC_DB_I3                     1072
#define IDC_AUTO_SLD                  1072
#define IDC_DB_Q3                     1073
#define IDC_AUTO_LAB                  1073
#define IDC_DB_P3                     1074
#define IDC_FILLBOX                   1074
#define IDC_DB_I4                     1075
#define IDC_COLOR                     1075
#define IDC_DB_Q4                     1076
#define IDC_POLL_EB                   1076
#define IDC_DB_P4                     1077
#define IDC_POLL_SLD                  1077
#define IDC_DB_I5                     1078
#define IDC_POLL_LAB                  1078
#define IDC_DB_Q5                     1079
#define IDC_DB_P5                     1080
#define IDC_CLRSTATS                  1080
#define IDC_ABOUT1                    1081
#define IDC_ORACREATE                 1081
```

```
#define IDC_BIG_TITLE                    1082
#define IDC_MODEBOX                      1083
#define IDC_STATBOX                      1084
#define IDC_FILLR                        2000
#define IDC_TOTALS                       2001

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_3D_CONTROLS                     1
#define _APS_NEXT_RESOURCE_VALUE        140
#define _APS_NEXT_COMMAND_VALUE         32771
#define _APS_NEXT_CONTROL_VALUE         1085
#define _APS_NEXT_SYMED_VALUE           101
#endif
#endif
```

Having above indicated several embodiments of the Subject Invention, it will occur to those skilled in the art that modifications and alternatives can be practiced within the spirit of the invention, it is accordingly intended to define the scope of the invention only as indicated in the following claims.

WHAT IS CLAIMED IS:

1. A message queuing system, comprising:

means for transmitting a transactional message having an associated message queue including the state of the queue, message queue data and log records; and,

means at a recipient site for storing said transactional message on a single disk in a single file utilizing a combined on-disk file structure for said message queue data and said log records.

2. The system of Claim 1, and further including a read/write head for accessing said single disk and means for driving said head in a single forward direction during a write operation.

3. The system of Claim 1, and further including a queue entry management table placed on said disk at preselected locations, said table having a control information block, at least one message block and at least one log record.

4. The system of Claim 3, wherein said preselected locations correspond to a fixed offset from the beginning of said file, thus to permit rapid identification of the most recent state of the message queue data.

5. The system of Claim 4 and further including means at said recipient site for recovering said message queue upon interruption of said transmission responsive to the most recent queue entry management table prior to said information, whereby the last valid information received and stored is located from information contained in said most recent queue entry management table.

6. The system of Claim 5, wherein said file is divided into sectors, and wherein said offset places a queue entry management table at the beginning of a sector such that said table constitutes a checkpoint for the location of a sector having valid information, whereby the last valid information prior to said interruption may be rapidly located through the identification of the sector containing said most recent table.

7. The system of Claim 2, wherein said management queue table is written in contiguous blocks, the result of a single file, forward write direction and contiguous message queue data block system being minimization of seek time, increase in throughout and rapid recovery from a transmission interruption.

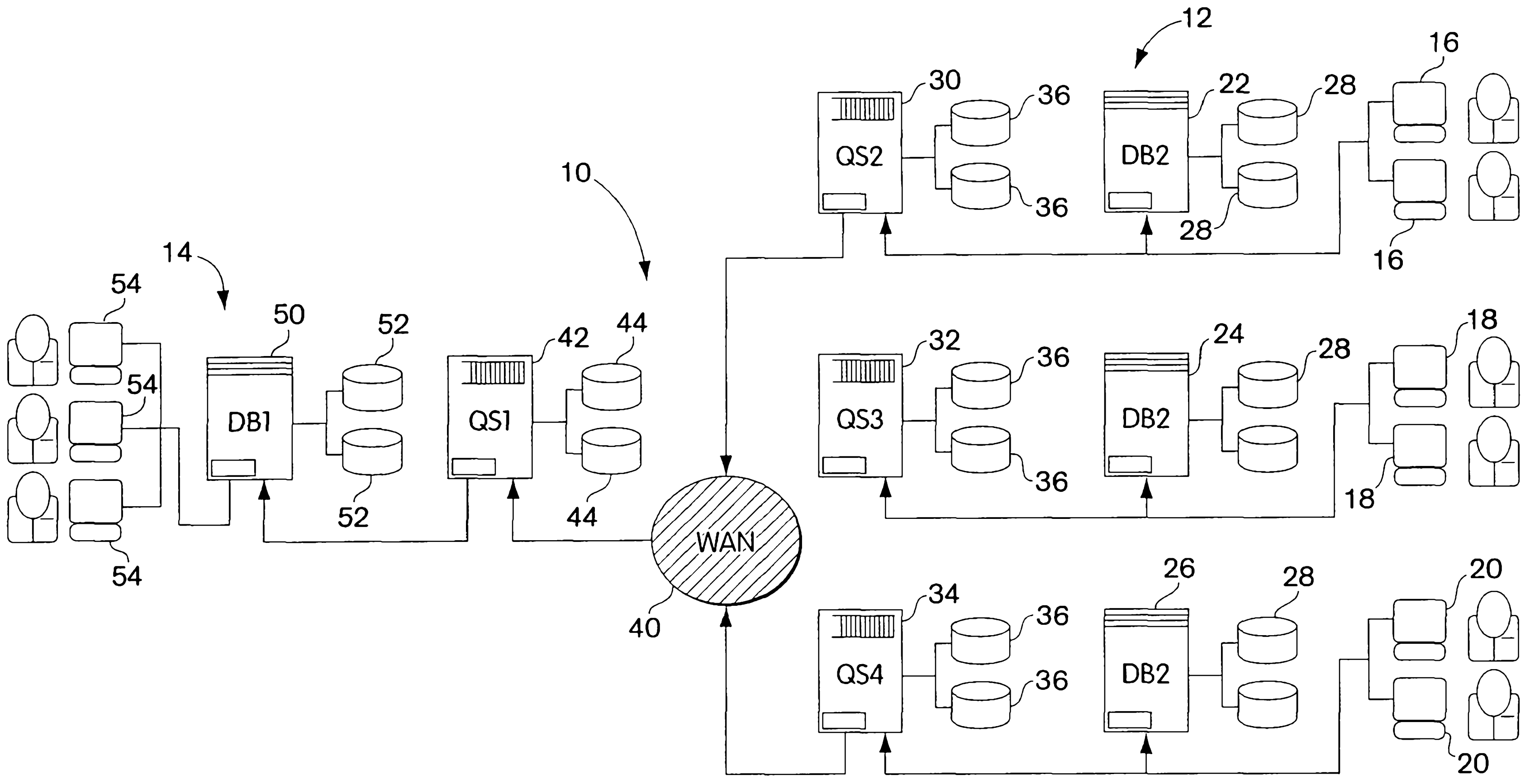


Fig. 1
(Prior Art)

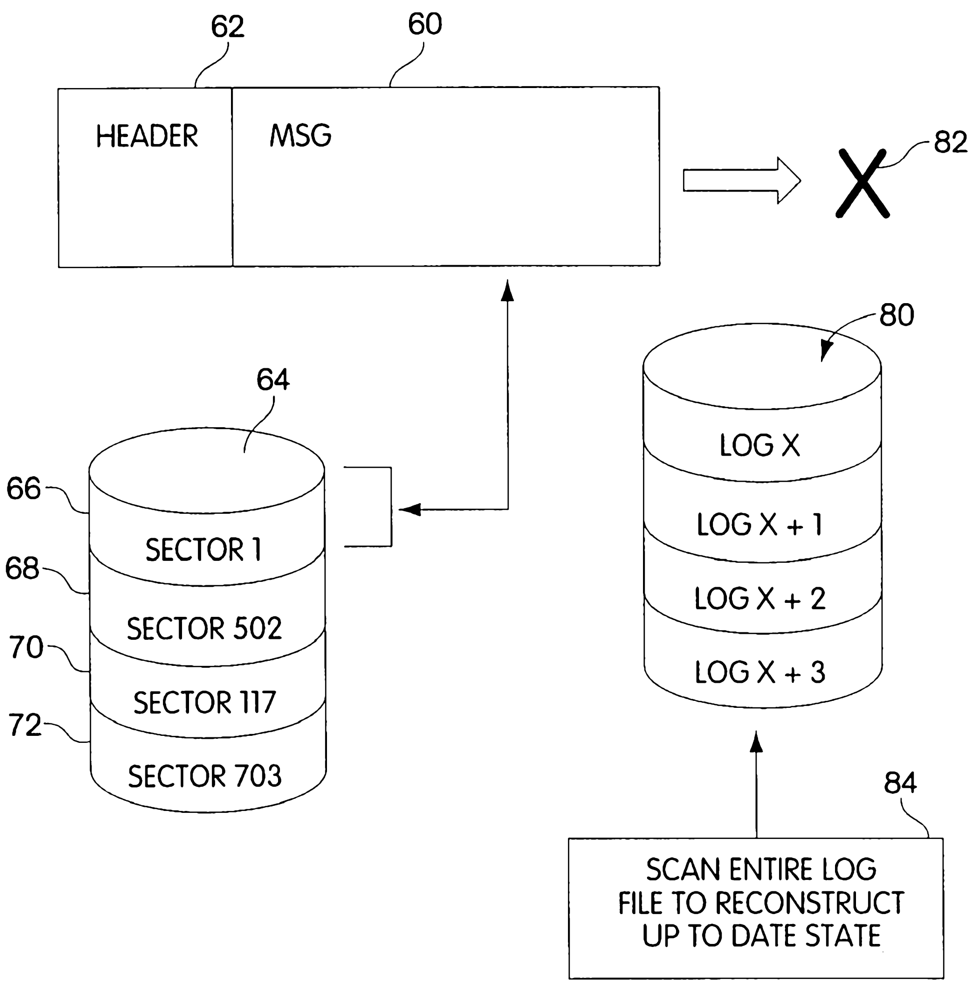


Fig. 2
(Prior Art)

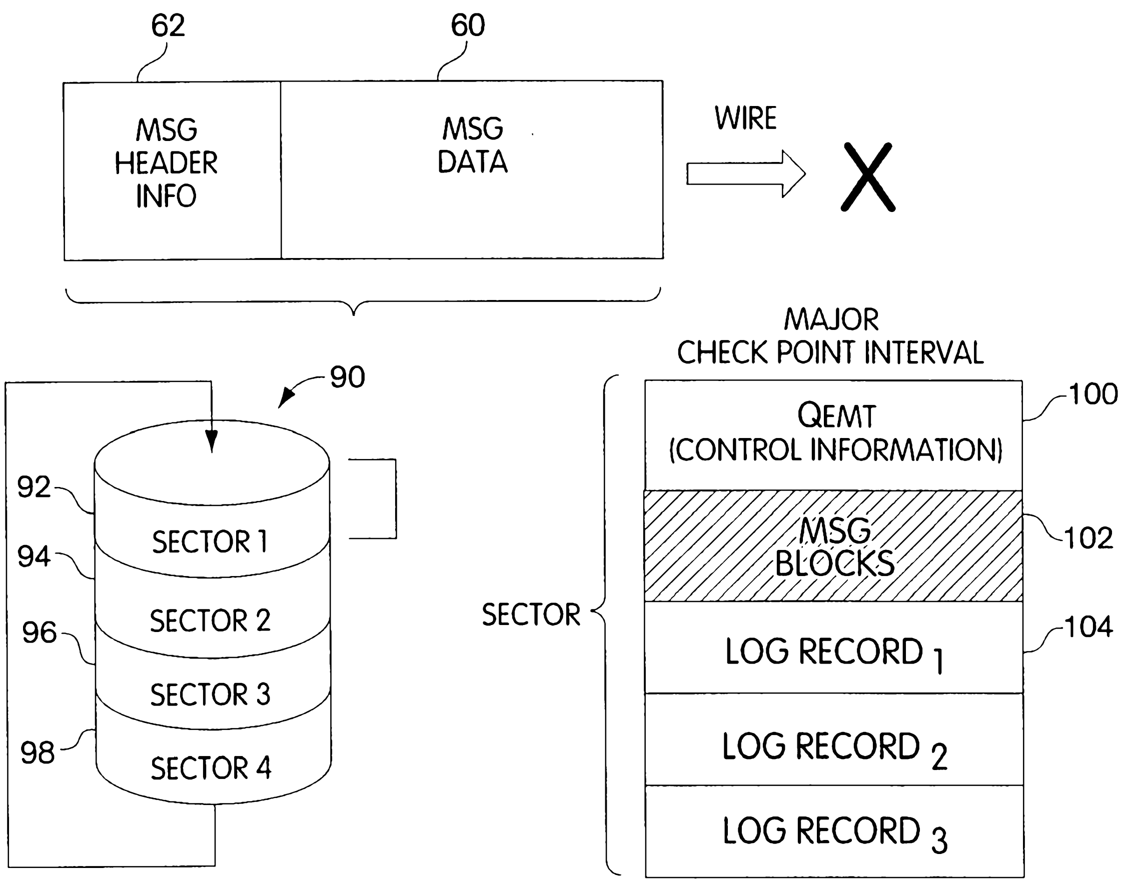

62
60
MSG HEADER INFO
MSG DATA
WIRE
X
MAJOR CHECK POINT INTERVAL
90
92
94
96
98
SECTOR 1
SECTOR 2
SECTOR 3
SECTOR 4
SECTOR
QEMT (CONTROL INFORMATION)
MSG BLOCKS
LOG RECORD 1
LOG RECORD 2
LOG RECORD 3
100
102
104

Fig. 3

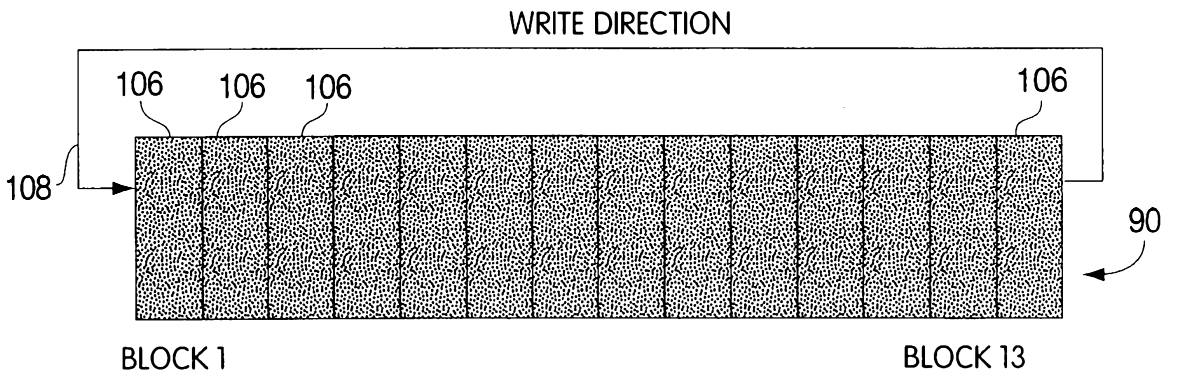


Fig. 4

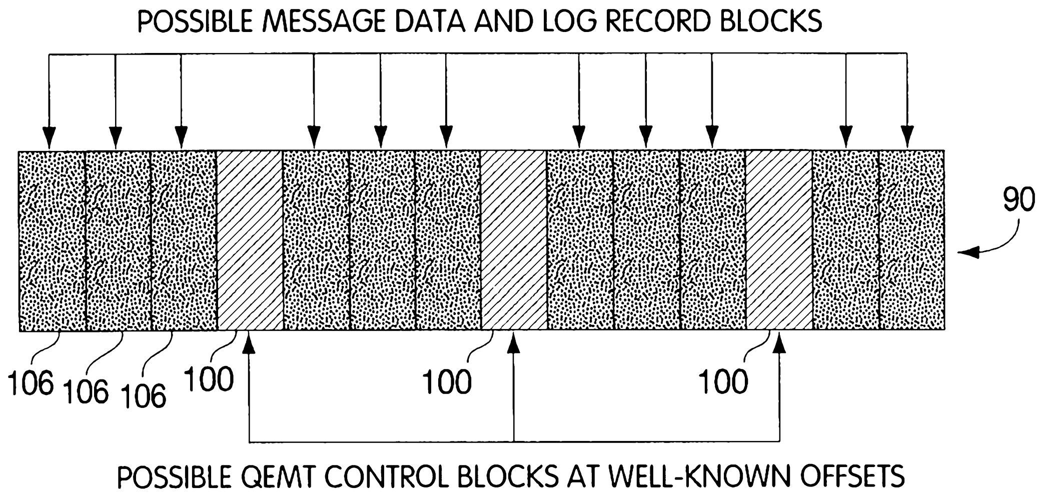

POSSIBLE MESSAGE DATA AND LOG RECORD BLOCKS
90
106
106
106
100
100
100
POSSIBLE QEMT CONTROL BLOCKS AT WELL-KNOWN OFFSETS

Fig. 5

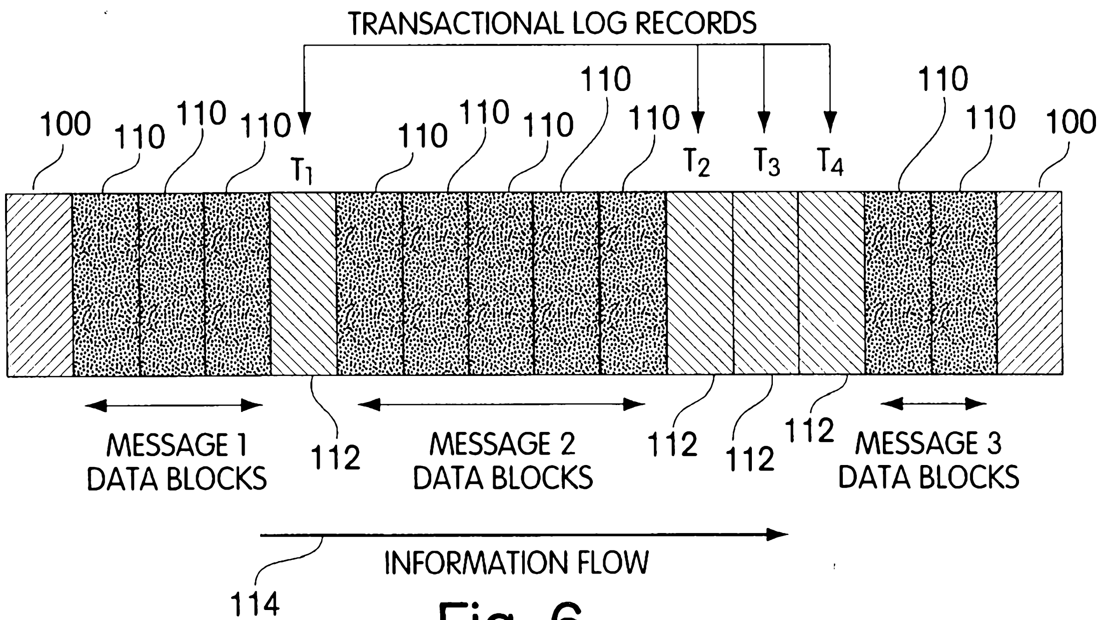


Fig. 6

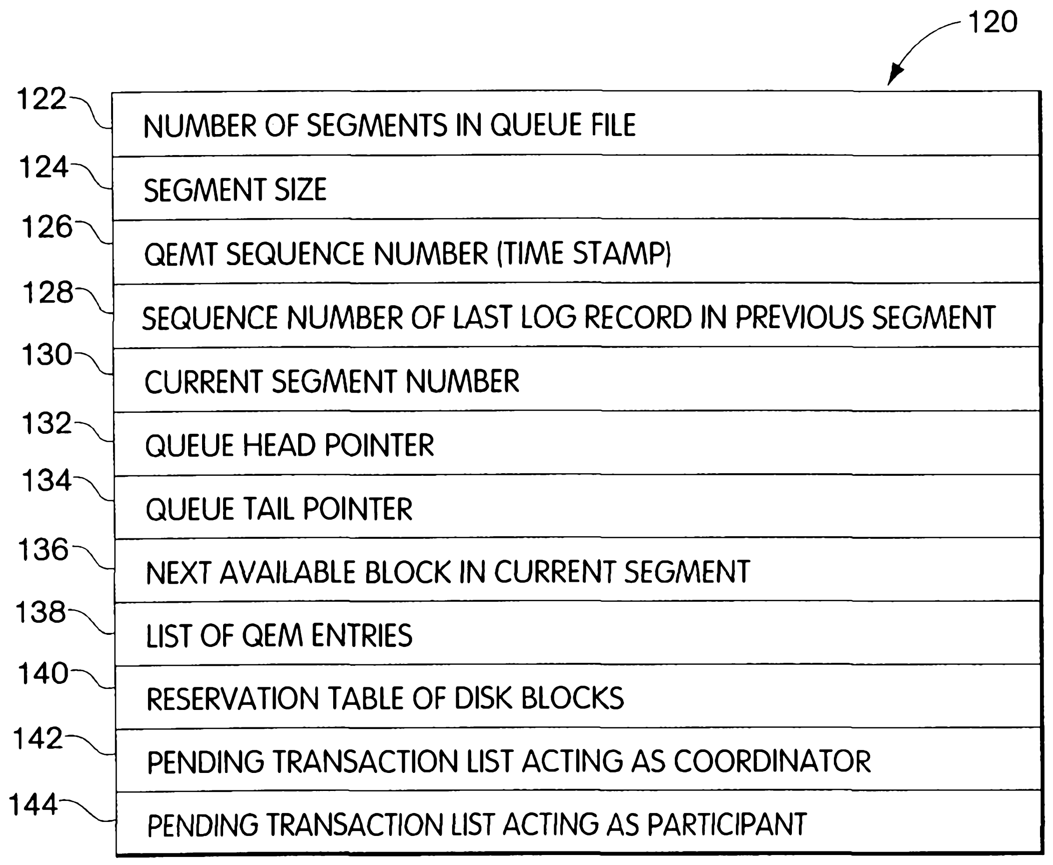

120
122
NUMBER OF SEGMENTS IN QUEUE FILE
124
SEGMENT SIZE
126
QEMT SEQUENCE NUMBER (TIME STAMP)
128
SEQUENCE NUMBER OF LAST LOG RECORD IN PREVIOUS SEGMENT
130
CURRENT SEGMENT NUMBER
132
QUEUE HEAD POINTER
134
QUEUE TAIL POINTER
136
NEXT AVAILABLE BLOCK IN CURRENT SEGMENT
138
LIST OF QEM ENTRIES
140
RESERVATION TABLE OF DISK BLOCKS
142
PENDING TRANSACTION LIST ACTING AS COORDINATOR
144
PENDING TRANSACTION LIST ACTING AS PARTICIPANT

Fig. 7

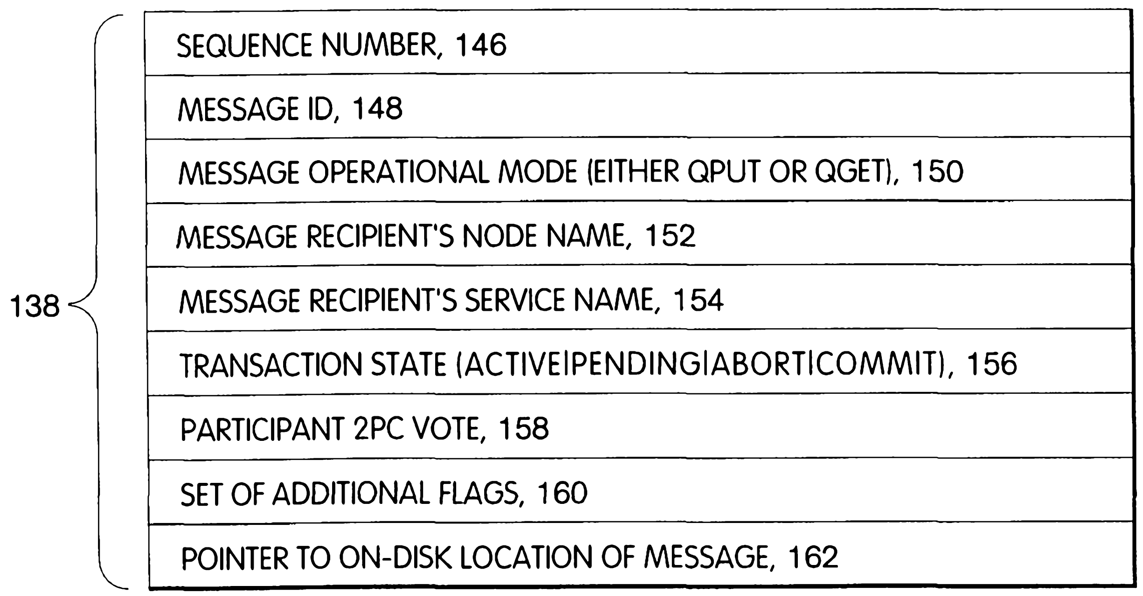

138
SEQUENCE NUMBER, 146
MESSAGE ID, 148
MESSAGE OPERATIONAL MODE (EITHER QPUT OR QGET), 150
MESSAGE RECIPIENT'S NODE NAME, 152
MESSAGE RECIPIENT'S SERVICE NAME, 154
TRANSACTION STATE (ACTIVE|PENDING|ABORT|COMMIT), 156
PARTICIPANT 2PC VOTE, 158
SET OF ADDITIONAL FLAGS, 160
POINTER TO ON-DISK LOCATION OF MESSAGE, 162

Fig. 8

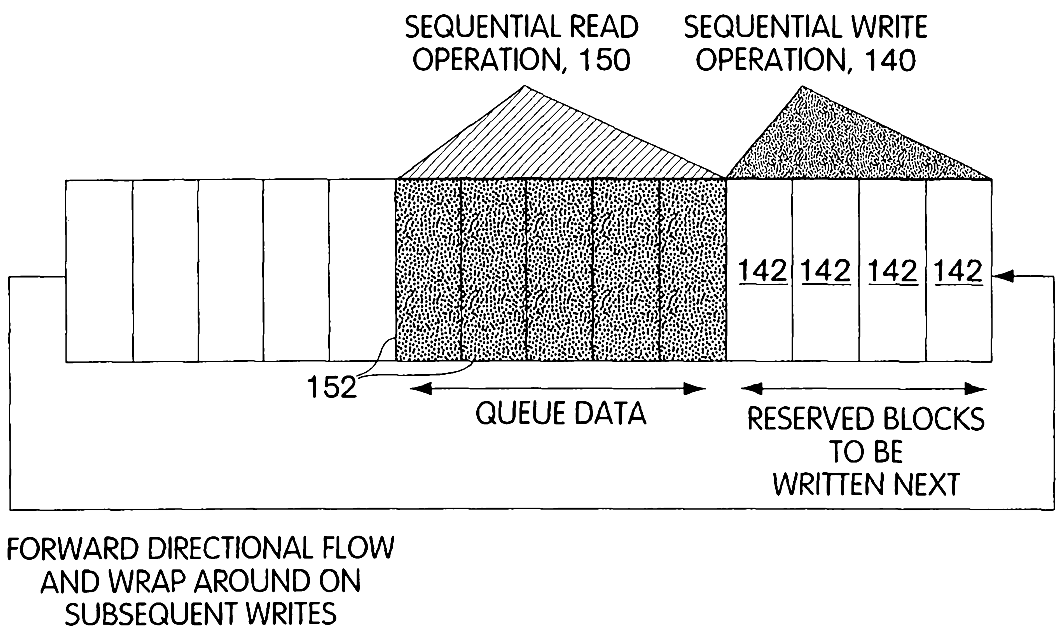

SEQUENTIAL READ OPERATION, 150
SEQUENTIAL WRITE OPERATION, 140
142
142
142
142
152
QUEUE DATA
RESERVED BLOCKS TO BE WRITTEN NEXT
FORWARD DIRECTIONAL FLOW AND WRAP AROUND ON SUBSEQUENT WRITES

Fig. 9

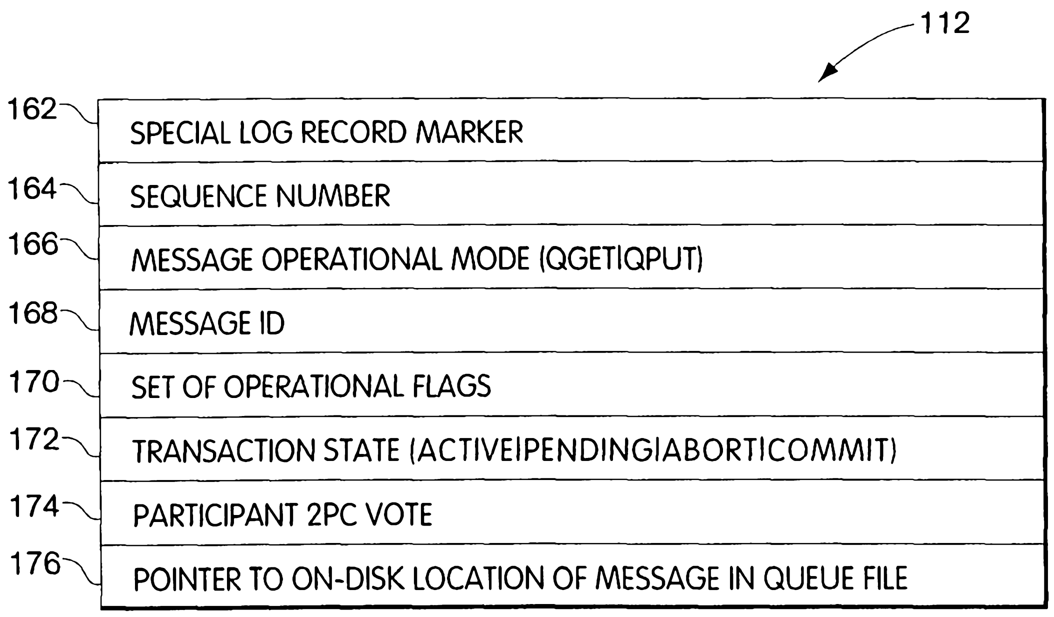
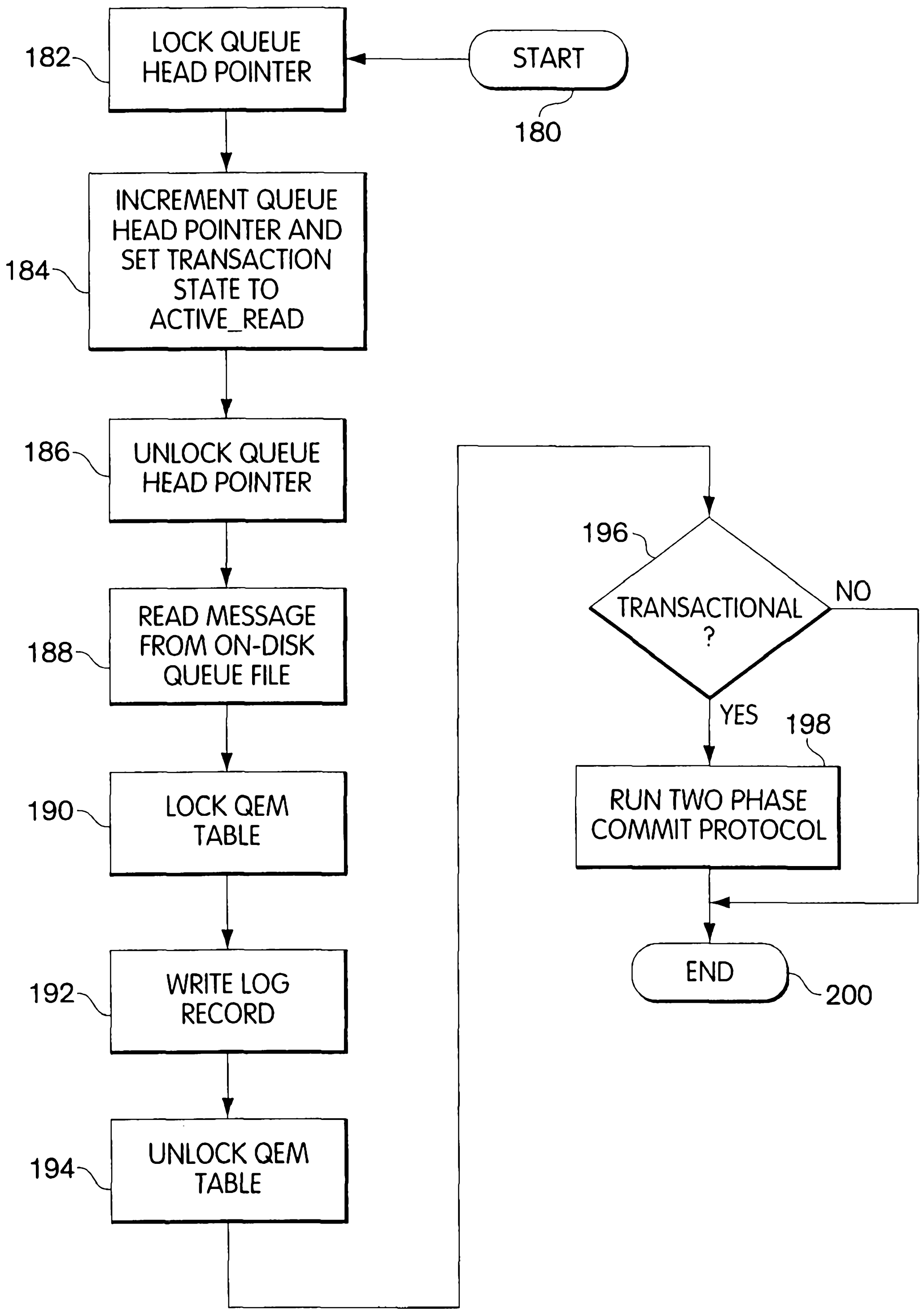

112
162
SPECIAL LOG RECORD MARKER
164
SEQUENCE NUMBER
166
MESSAGE OPERATIONAL MODE (QGET|QPUT)
168
MESSAGE ID
170
SET OF OPERATIONAL FLAGS
172
TRANSACTION STATE (ACTIVE|PENDING|ABORT|COMMIT)
174
PARTICIPANT 2PC VOTE
176
POINTER TO ON-DISK LOCATION OF MESSAGE IN QUEUE FILE

Fig. 10



START
180
LOCK QUEUE HEAD POINTER
182
INCREMENT QUEUE HEAD POINTER AND SET TRANSACTION STATE TO ACTIVE_READ
184
UNLOCK QUEUE HEAD POINTER
186
READ MESSAGE FROM ON-DISK QUEUE FILE
188
LOCK QEM TABLE
190
WRITE LOG RECORD
192
UNLOCK QEM TABLE
194
TRANSACTIONAL ?
196
NO
YES
RUN TWO PHASE COMMIT PROTOCOL
198
END
200

Fig. 11

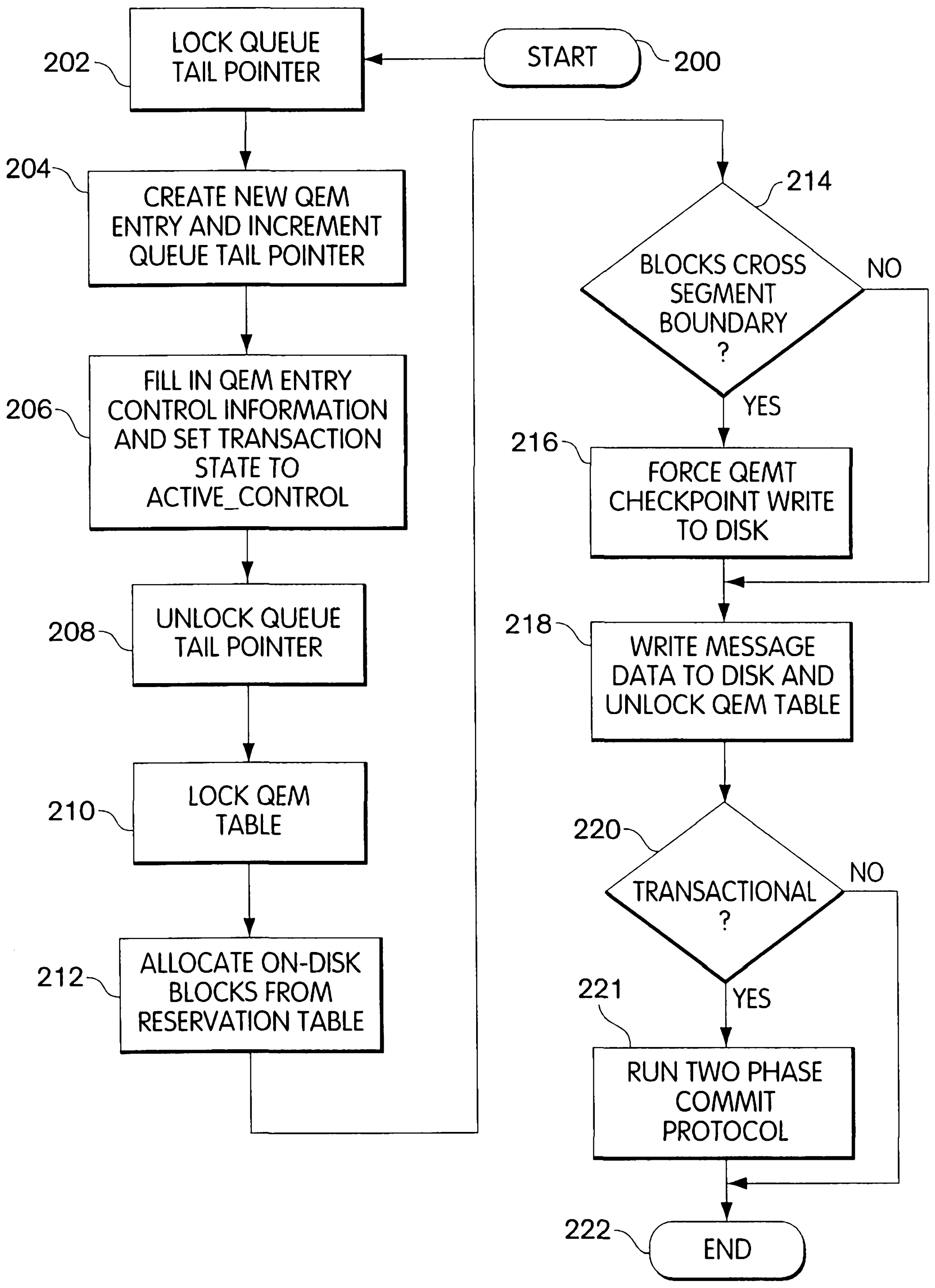

START
200
202
LOCK QUEUE TAIL POINTER
204
CREATE NEW QEM ENTRY AND INCREMENT QUEUE TAIL POINTER
206
FILL IN QEM ENTRY CONTROL INFORMATION AND SET TRANSACTION STATE TO ACTIVE_CONTROL
208
UNLOCK QUEUE TAIL POINTER
210
LOCK QEM TABLE
212
ALLOCATE ON-DISK BLOCKS FROM RESERVATION TABLE
214
BLOCKS CROSS SEGMENT BOUNDARY ?
NO
YES
216
FORCE QEMT CHECKPOINT WRITE TO DISK
218
WRITE MESSAGE DATA TO DISK AND UNLOCK QEM TABLE
220
TRANSACTIONAL ?
NO
YES
221
RUN TWO PHASE COMMIT PROTOCOL
222
END

Fig. 12

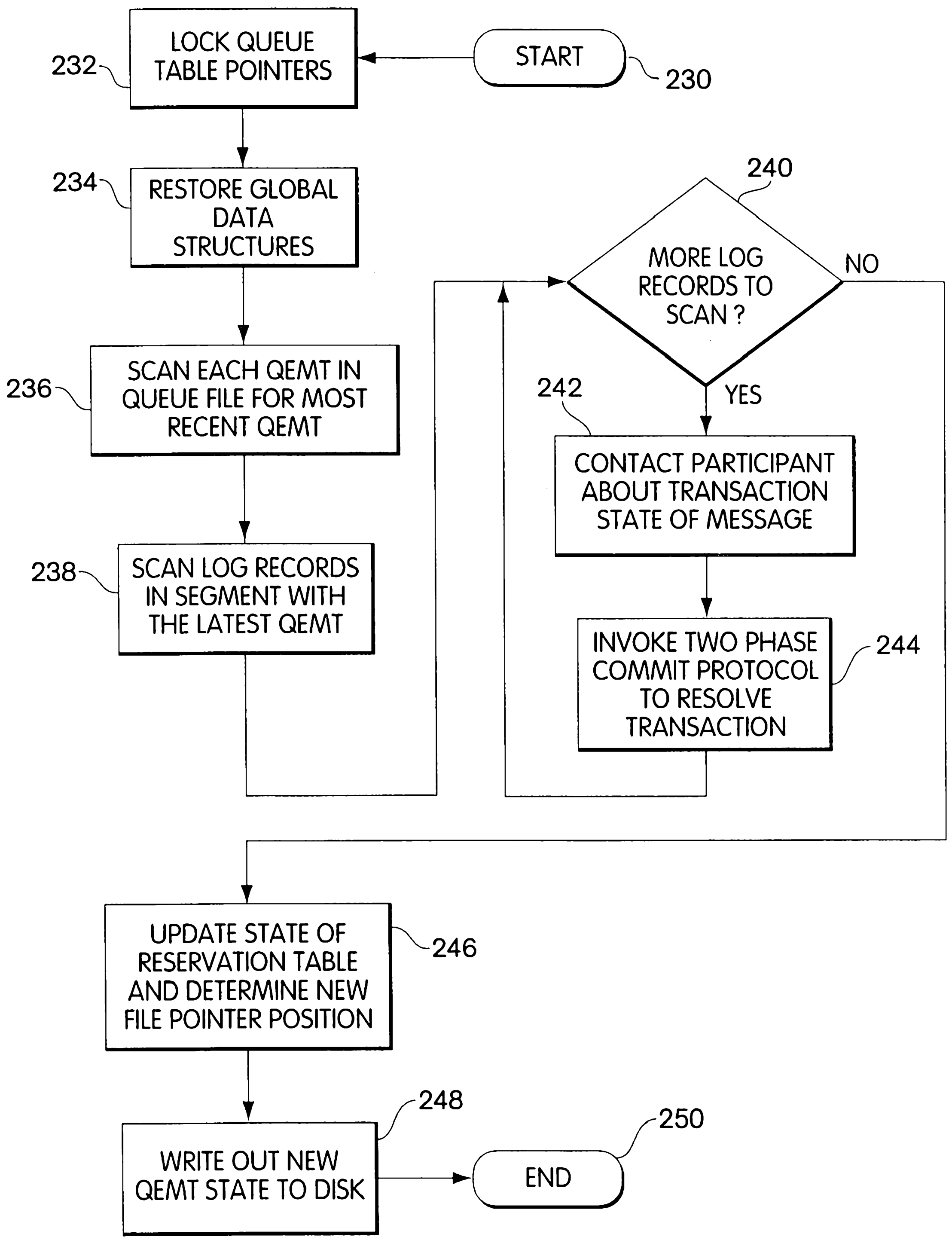

START
230
LOCK QUEUE TABLE POINTERS
232
RESTORE GLOBAL DATA STRUCTURES
234
SCAN EACH QEMT IN QUEUE FILE FOR MOST RECENT QEMT
236
SCAN LOG RECORDS IN SEGMENT WITH THE LATEST QEMT
238
MORE LOG RECORDS TO SCAN ?
240
NO
YES
CONTACT PARTICIPANT ABOUT TRANSACTION STATE OF MESSAGE
242
INVOKE TWO PHASE COMMIT PROTOCOL TO RESOLVE TRANSACTION
244
UPDATE STATE OF RESERVATION TABLE AND DETERMINE NEW FILE POINTER POSITION
246
WRITE OUT NEW QEMT STATE TO DISK
248
END
250

Fig. 13

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US97/20561

A. CLASSIFICATION OF SUBJECT MATTER

IPC(6) :G06F 11/00, 12/00
US CL : 395/182.16, 182.18
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 395/182.16, 182.18, 182.13, 427

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 5,465,328 A (DIEVENDORFF et al) 07 November 1995, Fig. 2, Col. 9 line 40 - Col. 15, line 41.	1-7
X	US 5,555,388 A (SHAUGHNESSY) 10 September 1996, Figures 3B, 4 and 6B, and Col. 15, line 23 -Col. 19, line 64.	1-7

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

* Special categories of cited documents:	"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"A" document defining the general state of the art which is not considered to be of particular relevance	
"E" earlier document published on or after the international filing date	"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"O" document referring to an oral disclosure, use, exhibition or other means	
"P" document published prior to the international filing date but later than the priority date claimed	"&" document member of the same patent family

Date of the actual completion of the international search 06 FEBRUARY 1998	Date of mailing of the international search report 20 MAR 1998
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231	Authorized officer [illegible] Thomas C. Lee
Facsimile No. (703) 305-3230	Telephone No. (703) 305-9717