Queue Services

Queue Services make NetWare's Queue Management System (QMS) available to developers. This chapter explains what QMS is and how developers can use Queue Services to distribute processing tasks over the network.

- Why Use QMS?
- Queues and the Bindery
- The Queue Process
- Using Queue Services

Why Use QMS?

Among the methods that developers can use to distribute processes on the network, QMS is perhaps the simplest and most direct. Although not a suitable solution for all situations, QMS offers some advantages over other methods.

Applications that need to do the following may want to consider using QMS:

- Exercise control over the flow and execution of large workloads
- Define specifications and protocols for processing jobs
- Provide broad safeguards to protect job information and network data

Control

Generally, QMS is suited to applications that are dealing with large workloads and need the flexibility and control inherent in the queuing process. Time and efficiency are somewhat less important to these applications than are security and reliability.

For example, an archive server is a good candidate for QMS. An archive server cares less about speed than about a dependable and secure operating environment. In addition, archive jobs may need to be sorted and managed according to size, priority, kind and other properties. QMS offers the regulatory features these tasks require.

Flexibility

Flexibility is another one of QMS's advantages. Although every queue must adhere to QMS's standard structure, QMS allows applications to define their own specifications within the QMS format. Thus, queues can transmit information that is specific to a specialized service. For example, a print server could use QMS to define its own protocol for handling diverse printing formats.

Security

QMS is based on NetWare's bindery and takes advantage of the bindery's extensive security structure. Along with NetWare's directory security, the bindery's organization clearly defines the relationship among queues, users, and queue servers. An application can ensure that only qualified users work with network data, whether it is placed temporarily in a queue or stored permanently on network disks.

In addition to the advantages listed so far, QMS is relatively easy for developers to use. Since QMS exploits NetWare's existing design and security features, it avoids possible conflicts with other NetWare services. Developers who are already familiar with NetWare's bindery should find QMS's design familiar and accessible. In addition, queue services contain an extensive group of function calls that help make QMS simple and convenient to manage.

Queues and the Bindery

Queues form the basis of QMS. Simply put, a queue is a group of jobs waiting to be serviced. On one end of the queue are users who submit jobs, and at the other end are queue servers that process the jobs. For the most part, jobs proceed sequentially through the queue, those arriving first being serviced before those that come after.

Queues are objects in a file server's bindery. Defining a queue as a bindery object allows QMS to guarantee that only authorized users will access and modify the queue. By defining users, operators and servers as properties of the queue, QMS provides extensive control over how the queue is used and who may use it. For more information about the structure of the bindery, see Bindery Services.

Queue Object

As with any bindery object, a queue has a name, type, ID number and security status. The queue's name and ID number identify it uniquely among queues in a file server's bindery. Particularly on an internetwork, it is important to note that a queue exists in the bindery of a specific file server.

QMS limits each queue to a single type of service. When an application creates a queue, it must define the type of jobs that queue intends to hold. Some queue types are designated by Novell. For example, a print queue is defined as NWOT_PRINT_QUEUE and will only accept jobs that correspond to that type.

When QMS creates a queue, the queue is assigned the security levels NWBS_ANY_READ (read anyone) and NWBS_SUPER_WRITE (write supervisor). The NWBS_ANY_READ level allows users to scan the bindery for a list of queues of a particular type. On the other hand, the NWBS_SUPER_WRITE level ensures that only the supervisor can modify the queue's properties.

Table 23 lists the features of a queue.

TABLE 23.

Name	Assigned by the application
ID Number	Assigned by QMS
Type	A number identifying the queue's work
Status	NWBS_ANY_READ | NWBS_SUPER_WRITE
Properties	Q_DIRECTOR Q_USERS Q_OPERATORS Q_SERVERS

The queue properties are described below.

Queue Properties

QMS attaches four properties to the queue object in the bindery: Q_DIRECTORY, Q_USERS, Q_SERVERS and Q_OPERATORS. These properties let an application control access to the queue through normal bindery routines.

The queue properties Q_USERS, Q_OPERATORS and Q_SERVERS define the role of queues and their relationship with other QMS components. In most cases, these properties are sufficient to accommodate a queue server's needs, although an application may define additional queue properties, as well. (For example, a queue could define as a property the type of hardware device that the queue is associated with--printer, plotter, tape, etc.)

Q_DIRECTORY Property

The first property, Q_DIRECTORY, has a security level of NWBS_SUPER_READ | NWBS_SUPER_WRITE (read supervisor and write supervisor). The other three properties are NWBS_ANY_READ | NWBS_SUPER_READ (read anyone and write supervisor). Thus, only the supervisor can see or modify the value of Q_DIRECTORY, but any logged in user can see the value of Q_USERS, Q_SERVERS and Q_OPERATORS.

When an application creates a queue, it must specify a directory path. QMS will use this path to create a queue directory that holds the system files and job files related to queue entries. The directory path SYS:SYSTEM is commonly used for the queue directory; however, you may specify any path.

QMS names the queue directory with the queue's ID number. The complete directory path, including the queue directory, is assigned to the Q_DIRECTORY property. The path string specified by the application cannot exceed NWMAX_PROPERTY_VALUE_LENGTH (128 characters, including the null terminator). Q_DIRECTORY is an item property; each queue has only one.

TABLE 24. Example of a Queue Directory

Path	SYS:\SYSTEM
Queue ID	015D03
Queue Directory	SYS:\SYSTEM\015D03

Q_USERS Property

QMS makes queues available to users through the Q_USERS property. This set property is a list of all the users who may place a job in a queue. The list can contain both user names and group names.

If a user is the security equivalent of any object listed in the Q_USERS property, that user has access to the queue. Likewise, when a group name is added to the Q_USERS property, all users in the group have access to the queue. Since the Q_USERS property is NWBS_SUPER_WRITE, only the supervisor can add or remove users from a queue. However, to use the queue, the supervisor must also be added to the Q_USERS property, just like any other object. If the supervisor does not want to place any restrictions on the queue, the group EVERYONE can be added to the Q_USERS property.

Q_SERVERS Property

For a queue to be serviced, a value must be assigned to the queue's Q_SERVERS property. This property holds a list of all the queue servers that can handle jobs placed in the queue. If an application does not assign any queue servers to the Q_SERVERS property, jobs may be placed in the queue, but they will not be serviced.

Before an application can assign a queue server to a queue, the queue server must exist as an object on the file server where the queue resides. As a bindery object, a queue server can log in to the file server and search for queues. QMS checks a queue server's login name and password against the servers listed in a queue's Q_SERVER property to determine if the queue server is authorized to service the queue.

When an application creates a queue server as a bindery object, it must specify the queue server's object type. As with queue types, Novell has designated object types for queue servers. Novell has defined the following object types for queue servers:

Object	**Type**
Print server	NWOT_PRINT_SERVER
Job server	NWOT_JOB_SERVER
Archive server	NWOT_ARCHIVE_SERVER

If you need an object type that is not defined, contact Novell's API Consulting Group.

An application must also assign an account balance to the queue server.

The Q_OPERATORS Property

QMS attaches a Q_OPERATORS property to each queue, which contains a list of all users who are authorized to manage the queue. An application must assign values to the Q_OPERATORS property before the queue can be managed. Operators can modify the specifications for a particular job as well as manipulate the order of jobs in the queue. Operators can also set the status of a queue, suspending the operation of the queue as needed.

As with the Q_USERS property, user or group names can be added to the Q_OPERATORS property. In other words, queue operators must exist in a file server's bindery as user and group objects. In order for the object SUPERVISOR to perform operator tasks, it must also be added as an object to the Q_OPERATORS property.

Supervisors and QMS

Developers should keep in mind the different security levels involved in a QMS application. An application needs supervisor equivalence to install and configure a queue server, since only supervisors can perform the bindery operation related to QMS.

However, queue users and queue operators need no special rights to exercise their privileges. Any user assigned to a queue's Q_USERS property can submit jobs, and any user assigned to a queue's Q_OPERATORS property can manage the operation of the queue.

Queue Process

When a queue is created in the bindery, QMS assigns it a Q_DIRECTORY property containing the queue directory path. Within this directory, QMS creates two system files that it uses to manage the queue. As jobs are submitted to the queue, temporary queue files are also placed in the queue directory.

QMS Files

Queue files are named with the characters Q$ followed by the last four digits of the queue's ID number. One file maintains a list of queue servers currently attached to the queue and has the extension .SRV. The other file maintains information about the jobs in the queue and has the extension .SYS. Both files are flagged HIDDEN when created.

A job is submitted to a queue as a queue file. Queue files use the same naming conventions as the queue system files, only each extension is a three digit number. The extension numbers of queue files reflect the sequence of queue jobs as they are created (.001, .002, .003, etc.).

When a queue file is created, a queue job structure (NWQueueJobStruct_t) is appended to the beginning of the file. The queue job structure takes up the first 255 bytes of the queue file. QMS uses this information to process the job. As jobs are processed, both the queue file and its reference within the .SYS file are deleted.

The queue file itself can contain any data needed for completing the job. If the queue supplies printing services, the queue file could contain the data to be printed. Or, if the queue supplies compiling services, the queue file might contain a compilation makefile. Developers are also free to use the queue file for transmitting their own protocol information.

NWQueueJobStruct_t Structure

All the information QMS requires to process a job is placed in the NWQueueJobStruct_t. Some of this information must be supplied by the application that creates the queue job; the rest is supplied by QMS. After the job is in the queue, the queue operator or the user who submitted the job can modify the information that indicates how to process the job.

NWQueueJobStruct_t Fields

Table 25 lists all of the fields in the NWQueueJobStruct_t structure. A description of each field follows the table.

TABLE 25. (Continued)NWQueueJobStruct_t Structure

Field	Type
clientStation	uint8
clientTask	uint8
clientID	uint32
targetServerID	uint32
targetExecutionTime	uint8 **[NWMAX_QUEUE_JOB_TIME_SIZE]**
jobEntryTime	uint8 **[NWMAX_QUEUE_JOB_TIME_SIZE]**
jobNumber	uint16
jobType	uint16
jobPosition	uint8

jobControlFlags	uint8
jobFileName	uint8 **[NWMAX_JOB_FILE_NAME_LENGTH]**
jobFileHandle	NWFileHandle_ta
servicingServerStation	uint8
servicingServerTaskNumber	uint8
servicingServerIDNumber	uint32
jobDescription	uint8 **[NWMAX_JOB_DESCRIPTION_LENGTH]**
queueRecord	NWClientRecord_ta

QMS automatically fills in all the fields in the structure except the following which must be filled in by the application:

targetServerID
targetExecutionTime
jobType
jobControlFlags
jobDescription
queueRecord

clientStation. This field contains the number of the station that placed the job in the queue.

clientTask. This field contains the number of the task the station was performing when it placed the job in the queue.

clientID. This field contains the user ID number of the station that placed the job in the queue.

targetServerID. This field contains the ID number of the queue server that is requested to service the job. A value of FF FF FF FF (4 bytes of 0xFF) indicates that any server may service the job. Otherwise, this field contains a queue server's ID number.

targetExecutionTime. This field contains the earliest time the client wants the job to be serviced. If this field is set to FF FF FF FF FF FF (6 bytes of 0xFF), the job will be serviced at the first opportunity.

This field is a 6 byte field.

First byte	Year	0 - 99 (90 for 1990)
Second byte	Month	1 - 12
Third byte	Day	1 - 31
Fourth byte	Hour	0 - 24
Fifth byte	Minute	0 - 59
Sixth byte	Second	0 - 59

jobEntryTime. This field contains the time that the job was placed in the queue. The time is taken from the system clock on the file server where the queue is found. This field is a 6 byte field. See ªtargetExecutionTimeº above for an explanation of the bytes.

jobNumber. This field contains the number that QMS assigns to the queue entry when the job is placed in the queue. A client should use this number when referring to the job.

jobType. This field contains a number that identifies the job by type. This field can contain any additional information that the client needs to transmit to the queue server. Applications are free to define and interpret this field as needed. Developers should avoid assigning -1 to a job type. If the queue server does not use the field, set the field to 0x00.

jobPosition. This field contains the position of the job in the queue. The entry at the head of the queue is number 1, the next entry is 2, and so on. As jobs are removed from the queue, the position numbers are updated to reflect the new position of the jobs that remain.

jobControlFlags. This field contains flag bits indicating the current status of the job. The bits are defined in Table

4.

TABLE 26. Job Control Flags

NWCF_SERVICE_AUTO_START	Setting this bit allows a job to go in the queue automatically if the connection is broken between the station submitting the job and the file server where the queue resides. Clearing this bit automatically removes a job from the queue if the connection breaks and the client has not yet released the job.
NWCF_SERVICE_RESTART	Setting this bit allows a job to remain in a queue in the event of a service failure. The queue server will attempt to service the job when service to the queue resumes. Clearing this bit allows the job to be removed from the queue if it has not been serviced successfully.
NWCF_ENTRY_OPEN	This bit is for QMS's internal management. It remains set as long as the client has not released the queue file to the queue. When the client has released the job, QMS clears the bit.
NWCF_USER_HOLD	Setting this bit places the job on hold. The job will continue to advance in the queue, but it will not be serviced until this bit is cleared.
NWCF_OPERATOR_HOLD	Setting this bit places the job on hold. The job will continue to advance in the queue, but it will not be serviced until this bit is cleared.

jobFileName. This field contains the name of the queue file created to process the job.

jobFileHandle. This field contains the file handle to the queue file that has been created in the queue. The handle can be used to write additional information to the file.

servicingServerStation. This field contains the station number of the queue server servicing the job. If no server is currently servicing the job, this field is undefined.

servicingServerTaskNumber. This field contains the task number of the queue server servicing a job. When no server is servicing a job, the value of this field is left undefined.

servicingServerIDNumber. This field contains the object ID of the queue server servicing a job. When no server is servicing the job, this field is set to zero. By testing this field, an application can determine whether a job is being serviced.

jobDescription. This field contains a null-terminated ASCII string. Applications can use this field to help identify the type or purpose of the job.

queueRecord. This field can contain any additional information that the client needs to transmit to the queue server. Applications are free to define and interpret this field as needed. Any values placed in this field will not be affected by QMS.

Managing the NWQueueJobStruct_t Structure

The NWQueueJobStruct_t structure plays several distinct roles in the queuing process. QMS, queue servers, clients and operators all depend on certain fields to transmit information at various points in the job process.

1) First, the following fields must be completed before the job can be placed in the queue.

   targetServerID
   targetExecutionTime
   jobType
   jobControlFlags

jobDescription
queueRecord

2) Next, when a job is placed in the queue, QMS records information in the following fields.

clientStation
clientTask
clientID
jobEntryTime
jobNumber
jobPosition
jobControlFlags **[NWCF_ENTRY_OPEN]**
jobFileName
jobFileHandle

3) Once the job is in the queue, the user who submitted the job or the queue operator can read and modify information in the following fields.

targetServerID
targetExecutionTime
jobType
jobPosition
jobControlFlags
jobDescription
queueRecord

4) When a server attaches to the queue, QMS checks the following fields to determine if the server is authorized to handle the job.

targetServerID
targetExecutionTime
servicingServerIDNumber
jobType
jobControlFlags The server cannot service the job if one of the following is set:

NWCF_OPERATOR_HOLD
NWCF_USER_HOLD
NWCF_ENTRY_OPEN

5) Finally, when QMS releases a job to a queue server, the following fields are filled in.

servicingServerStation
servicingServerTaskNumber
servicingServerIDNumber

Using Queue Services

Queue services has five areas of management:

- Creating queues in the bindery
- Submitting jobs to queues
- Monitoring and controlling the status of jobs and queues
- Creating queue servers in the bindery

- Servicing entries in queues as a queue server

Not all applications need to manage all five areas. For example, a simple word processing application could supply only the code to submit the job into a print queue. A more sophisticated application could also allow users to monitor and control their jobs in the queue. However, a print server would mainly be concerned with the last area, servicing entries in a print queue.

Each area of management is described below.

Creating Queues in the Bindery

Queues are created with NWCreateQueue. The client that creates the queue needs to be logged in with supervisor equivalency.

The NWCreateQueue creates the queue as a bindery object and as a directory. Typical bindery types for queues are listed below:

Object	Type
Print Queue	NWOT_PRINT_QUEUE
Archive Queue	NWOT_ARCHIVE_QUEUE
Job Queue	NWOT_JOB_QUEUE

It also creates four bindery property types and assigns them to the queue:

Q_DIRECTORY
Q_SERVERS
Q_OPERATORS
Q_USERS

The Q_DIRECTORY property stores the path information for the queue directory. Usually a standard naming convention is used for the directory. For example, the print queue directories created with PCONSOLE are subdirectories to SYS:SYSTEM, have a .NPQ extension and are named by giving them the character value of their hexadecimal object ID. The Q_DIRECTORY property is an item property.

The other three properties are set properties and allow you to assign bindery objects to them.

- The Q_SERVERS property stores the object IDs of the servers that have been added to this property. These servers, and only these servers, can service the queue. If no servers are assigned to the queue, no jobs will be serviced. The queue server must be created as a bindery object before the queue server can be assigned to this property.
- The Q_OPERATORS property stores the object IDs of the users or groups that can perform maintenance tasks on the queues. These objects are able to delete all jobs in the queue, rearrange the order of the jobs in the queue, change job servicing information and control the submission of new jobs. Usually SUPERVISOR is added to this group.
- The Q_USERS property stores the object IDs of the users or groups that have permission to use the queue. Each object can delete its own jobs or change the job servicing information. If no objects are assigned to the queue as queue users, no one can submit a job to the queue. Usually the group EVERYONE is added to this group.

Use NWAddObjectToSet to add bindery objects to the properties. Use NWDeleteObjectFromSet to remove a bindery object from the properties.

Use NWCreateProperty to create additional bindery properties for the queue. If you create a set property, use NWAddObjectToSet to add objects to the property. If you create an item property, use NWWritePropertyValue to write the value to the property.

Use NWDestroyQueue to delete a queue from the bindery.

Submitting Jobs to Queues

For a client to submit a job to a queue, the client must have its object ID (or the object ID of a group that client belongs to) in the Q_USERS property.

To submit a job to the queue, complete the following steps:

1) Assign values to the necessary fields in the NWQueueJobStruct_t structure.

2) If the queue server uses the queueRecord field, assign a value to the queueRecord field.
   - If the job is a NetWare print job, use NWConvertPrintStructToQueueStruct.
   - If the job is not a NetWare print job, use the queue server's function to fill in the queueRecord field.

3) Create a file in the queue. Use NWCreateQueueFile.

4) Write the file to the queue. Use NWWriteFile. Use the file handle that was created with NWCreateQueueFile.

5) Close the file. Use NWCloseFileAndStartQueueJob. If an error occurs, you can use NWCloseFileAndAbortQueueJob which closes the source file and removes the job from the queue.

The job is now ready to be serviced.

Monitoring and controlling the status of jobs and queues

The ability to monitor and control jobs in the queue varies.

- Clients which have been assigned to the Q_OPERATOR property can control the queue and the jobs.
- Clients which have been assigned to the Q_USERS property can control their own jobs and as users, they can view information about the queue and other users' jobs in the queue.
- Clients which have not been added to either property cannot view the queue status or queue job information.
- Queue servers which have been added to the Q_SERVERS property can read and service jobs in the queue.

The APIs allow you to do the following.

TABLE 27. (Continued)Controlling the Queue

Task	**API**	**Client**
Delete a job	NWRemoveJobFromQueue	Owner Operator
View job information	NWReadQueueJobEntry*	User Operator Server
Change job information	NWChangeQueueJobEntry	Owner Operator Server
List jobs in the queue	NWGetQueueJobList	User Operator Server

Change servicing order	NWChangeQueueJobPosition	Operator
View queue status flags	NWReadQueueCurrentStatus	User Server
Set queue status flags	NWSetQueueCurrentStatus	Operator
View queue server flags	NWReadQueueServerCurrentStatus	User
Set queue server flags	NWSetQueueServerCurrentStatus	Operator

*If the job is a NetWare print job, use NWConvertQueueStructToPrintStruct before reading the job. If the job is not a NetWare print job and the queue server required a function to change the job into a queue job, use the queue server's function to return the job to its original form.

Creating Queue Servers in the Bindery

To create a queue server, follow the steps outlined below.

1) Create the queue server as a bindery object. The client that creates the queue server needs to be logged in as supervisor equivalent. Use NWCreateObject to create the queue server. Assign the object an appropriate object type. Typical types for queue servers are listed below:

Object	Type
Job Server	NWOT_JOB_SERVER
Print Server	NWOT_PRINT_SERVER
Archive Server	NWOT_ARCHIVE_SERVER

2) Get the object ID of the queue server. Use NWGetObjectID. (The object ID number must be passed with some of the other API calls.)

3) Create a PASSWORD property for the queue server and assign the property a value. Use NWChangeObjectPassword.

4) Create properties for the queue server. Use NWCreateProperty. For example, Novell's print server creates two properties: PS_USERS and PS_OPERATORS. Both of these properties are of the type set and are given the access level of NWBS_LOGGED_READ, NWBS_LOGGED_WRITE, and NWBS_OBJECT_WRITE.

5) Check for accounting on the file server. Use NWScanProperty and look for a file server property of ACCOUNT_SERVERS. If this property exists, create an ACCOUNT_BALANCE property for the queue server and write an ACCOUNT_BALANCE value to the property. The ACCOUNT_BALANCE property is of type item so use NWWritePropertyValue.

6) If you need a directory to store information in, create a directory for the queue server. Use NWCreateDir. Assign the queue server to be a trustee of the directory. Use NWSetTrustee.

7) Add objects to the properties created in Step 4. Usually user SUPERVISOR is added to an operator property and group EVERYONE is added to a user property. For set properties, use NWAddObjectToSet. For item properties, use NWWritePropertyValue.

To delete a queue server from a file server, use the following APIs:

- Use NWDeleteObject to delete a queue server from the bindery.
- Use NWDeleteDir to delete any directories created for the queue server.

Servicing Entries in the Queue

To service jobs in a queue, the queue server needs to perform the following tasks.

1) Log the queue server into the file server. Use NWAttachToServerPlatform and NWLoginToServerPlatform.

2) Attach the queue server to the queue. Use NWAttachQueueServerToQueue.

Note: Before the attach call can be made, the queue server must be a member of the queue's Q_SERVERS property.

3) Get the current status of the queue. Use NWReadQueueCurrentStatus.

4) Check to see if there are jobs to be serviced. Use NWGetQueueJobList.

5) Find the next job that needs to be serviced. Use NWReadQueueJobEntry. If the queue server needs to know the size of the job, use NWGetQueueJobSize.

6) Select the job for servicing. Use NWServiceQueueJob.

7) Read the job. Use NWReadFile.

8) Process the job. This varies with the queue server. A print server would send the file to the printer with the appropriate printing codes. An archive server would send the file to a backup device with the appropriate formatting codes.

The process could also involve having the queue server assume the rights of the client. For example, an archive server could assume the rights of the client before restoring files to enforce the following:

- The clients could restore only files that they had the appropriate rights to.
- The clients could restore files only to directories that they had the Create right in.
- The clients could be assigned as the owner of the restored file.

You should use NWChangeToClientRights before processing the job and then use NWRestoreQueueServerRights when the job is finished.

9) Remove the job from the queue. Use NWFinishServicingQueueJob.

10) Repeat Step 4 through Step 9 for additional jobs in the queue.

NWDetachQueueServerFromQueue should be used before bringing down the queue server.

NWAbortServicingQueueJob can be used to interrupt the servicing of a job.