Programmer's Guide to the GQ Facility

A Facility for Generalized Queues

David E. Beecher

Mallinckrodt Institute of Radiology
Electronic Radiology Laboratory
510 South Kingshighway Boulevard
St. Louis, Missouri 63110
314/362-6965 (Voice)
314/362-6971 (FAX)

Version 2.10.0

August 3, 1998

A Guide to using the Multiple Process, Generalized Queuing Facility.

---

1 Introduction

The GQ (Generalized Queuing) facility allows single or multiple processes to create distinct queues and use them for interprocess communication. This facility is an extension of the SNQ (Study Name Queue) facility from the DICOM '92 demonstration. The main difference is that this year the queue elements are completely arbitrary, and are specified at queue creation time. The queuing mechanisms have no idea what a queue element looks like, only that they are of a particular length (all elements of a particular queue must be the same length), and that there is some maximum number of elements for a particular queue. It is the user's responsibility to ensure that the data being enqueued and dequeued is of the proper size and type, since it is impossible for the queuing routines to perform any element type checking.

These queues are implemented using shared memory and semaphores, two UNIX resources that must be present for the queues to operate properly. Most UNIX systems will supply the exact routines used or a reasonable facsimile. Access to these queues is through standard routines which are described below.

Queues are created and deleted with GQ_InitQueue and GQ_KillQueue. A particular queue can be attached with GQ_GetQueue. A process can attach multiple queues by calling GQ_GetQueue multiple times; each time with a different queue identifier. The responsibilty of freeing the resources allocated to the queues lies with the user. GQ_KillQueue with the specified queue identifier will deallocate all resources allocated to that queue. Two standard manipulation routines are included, GQ_Enqueue, to place a new element on the tail of the queue, and GQ_Dequeue, to remove an element from the head of the queue and return it to the user. One utility routine, GQ_PrintQueue, is used to dump all the elements of a queue with a specific queue identifier to standard output. A print routine must be supplied to GQ_PrintQueue so that it knows how to dump the elements of that particular queue.

Before these routines can be successfully executed, the user must be sure that his environment has the variable QUEUE_DIRECTORY set to some directory which is writable by the process to be executed. All processes wishing to use the shared queues must have this environment variable set to the same directory. This directory contains a file which holds the shared-memory and semaphore identifiers. As mentioned above, the GQ facility is implemented using shared memory and semaphore resources that are present on most machines. If these resources are not present, the GQ routines will return the value GQ_UNIMPLEMENTED.

A final note on resource allocation. It is very important to deallocate the shared memory and semaphore resources used after programs are finished with a particular queue. This is done automatically by calling the routine GQ_KillQueue. If this is not done, semaphore and shared memory resources will be quickly exhausted. Each queue created uses one semaphore and one shared memory segment. The user can examine how many are currently in use with the ipcs command. These segments can be removed manually with the ipcrm command. ipcrm -m <shared memory id> will remove a shared memory segment. ipcrm -s <semaphore id> will remove a semaphore. The ids needed are easily extracted from the ipcs command. When removing the queues manually, it is important to also remove the small communications file in the QUEUE_DIRECTORY

directory called gq.dat<qid>. For instance, if a queue was created with identifier 0, this file would be named “gq.dat0”.

2 Include Files

All applications that use the GQ facility should include these files in the following order:

```
#include “dicom.h”
#include “condition.h”
#include “gq.h”
```

3 Return Values

The following returns are possible from the GQ facility:

GQ_NORMAL	Specified operation was successful
GQ_QUEUEFULL	Attempt to enqueue another element to a full queue
GQ_QUEUEEMPTY	Attempt to dequeue an element from an empty queue
GQ_SHAREDMEMORYFAIL	The shared memory resource failed
GQ_SEMAPHOREFAIL	The semaphore resource failed
GQ_FILEACCESSFAIL	Could not access the communications file
GQ_NOMEMORY	Could not allocate memory
GQ_UNIMPLEMENTED	This facility is unimplemented
GQ_BADELEMSIZE	Inconsistent element size specification
GQ_MAXQUEUEEXCEEDED	Exceeded maximum allowed number of queues
GQ_FILECREATEFAILED	Failed creating file “gq.dat<id>” in QUEUE_DIRECTORY
GQ_MULTCREATEREQUEST	Request to create queue with already existing queue id.
GQ_NOPENQUEUE	There is currently no open queue

4 GQ Routines

This section provides detailed documentation for each GQ facility routine.

GQ_Enqueue

Name

GQ_Enqueue - place a new element at the tail of the queue with specified queue id

Synopsis

CONDITION GQ_Enqueue(int qid, void *element)

qid	unique id of the queue in which element is to be enqueued
element	a pointer to the data element to enqueue

Description

The element pointed to by the input parameter, element, is copied to the tail of the selected queue.

Notes

It is the users responsibility to ensure that the size of the element referred to with the pointer parameter is the correct size for this particular queue.

Return Values

```
GQ_NORMAL
GQ_SEMAPHOREFAIL
GQ_QUEUEFULL
GQ_NOPENQUEUE
GQ_UNIMPLEMENTED
```

GQ_Dequeue

Name

GQ_Dequeue - remove the next element from the head of the specified queue and return it

Synopsis

CONDITION GQ_Dequeue(int qid, void *element)

qid	unique id of the queue from which element is to be dequeued
element	element storage for the newly dequeued element

Description

The element pointed to by the input parameter, element, is replaced by the contents of the head of the queue.

Notes

It is the user's responsibility to ensure that the size of the element referred to with the pointer parameter is the correct size for this particular queue.

Return Values

```
GQ_NORMAL
GQ_SEMAPHOREFAIL
GQ_QUEUEMPTY
GQ_NOPENQUEUE
GQ_UNIMPLEMENTED
```

GQ_GetQueue

Name

GQ_GetQueue - select a new (already existing) queue to use

Synopsis

CONDITION GQ_GetQueue(int qid, int element_size);

qid The queue identifier.

element_size The size of the elements for this queue

Description

This routine attempts to access the already existing queue identified by qid, and make it the current queue for the calling routine. There can be problems with file access, semaphores, or shared memory. If all that goes well, then the size the user passes is checked against the known size of the existing queue's elements. The system assumes the element size passed by the caller in any subsequent enqueue or dequeue operations on this queue.

Notes

Problems with this routine usually indicate that the QUEUE_DIRECTORY environment variable has not been set or has been set incorrectly.

Return Values

```
GQ_NORMAL
GQ_SHAREDMEMORYFAIL
GQ_FILEACCESSFAIL
GQ_BADELEMSIZE
GQ_UNIMPLEMENTED
```

GQ_GetQueueSize

Name

GQ_GetQueueSize - return the number of elements in the specified queue.

Synopsis

CONDITION GQ_GetQueueSize(int qid, int *size);

qid The queue identifier.

size Pointer to a user defined int that will hold the size of the specified queue.

Description

This routine attempts to access the already existing queue identified by qid, and make it the current queue for the calling routine. If all checks are satisfied, it goes thru the queue and counts the number of elements currently in the queue..

Notes

This operation is atomic in the sense that no other process will be able to enqueue or dequeue elements from this queue while this routine counts the number of elements currently in the queue. Problems with this routine usually indicate that the QUEUE_DIRECTORY environment variable has not been set or has been set incorrectly.

Return Values

```
GQ_NORMAL
GQ_NOMEMORY
GQ_NOOPENQUEUE
GQ_SHAREDMEMORYFAIL
GQ_FILEACCESSFAIL
GQ_BADELEMSIZE
GQ_UNIMPLEMENTED
```

GQ_InitQueue

Name

GQ_InitQueue - create a new queue for use by the system

Synopsis

CONDITION GQ_InitQueue(int qid, int num_elements, int element_size)

qid The new queue identifier

num_elements The maximum number of elements this queue can hold

element_size The size of each of the above elements

Description

This routine attempts to create a new queue with the specified queue id (qid), with num_elements each of size element_size. It needs to allocate a chunk of shared memory the correct size, and allocate and initialize a semaphore for exclusive access, either of which may fail. If all these succeed, it creates a communications file and calls GQ_GetQueue before returning success.

Notes

Problems with this routine usually indicate that the QUEUE_DIRECTORY environment variable has not been set or has been set incorrectly.

Return Values

```
GQ_NORMAL
GQ_SHAREDMEMORYFAIL
GQ_SEMAPHOREFAIL
GQ_FILEACCESSFAIL
GQ_UNIMPLEMENTED
```

GQ_KillQueue

Name

GQ_KillQueue - remove an existing queue from the system

Synopsis

CONDITION GQ_KillQueue(int qid)

qid The queue identifier to remove

Description

This routine operates on the queue with queue identifier qid. It attempts to detach the shared memory segment holding the queue and then frees up the associated semaphore. As a final cleanup it removes the small ASCII file used to communicate these identifiers between processes.

Return Values

```
GQ_NORMAL
GQ_SHAREDMEMORYFAIL
GQ_SEMAPHOREFAIL
GQ_UNIMPLEMENTED
```

GQ_ModifyHeadElement

Name

GQ_ModifyHeadElement - Atomically modify element at the head of the queue

Synopsis

CONDITION GQ_ModifyHeadElement(int qid, void *element, void (*func)(void *element))

qid	The queue identifier of the queue
element	Pointer to the head element that will be passed to the function passed as the next parameter
func	Pointer to function that will be invoked by the routine in order to allow the user to handle the head element.

Description

The modification operation takes place atomically. NO other enqueues or dequeues can occur until the user supplied function has terminated and the routine returns.

Notes

User should allocate memory for the element.

Return Values

GQ_NORMAL
GQ_NOMEMORY
GQ_SHAREDMEMORYFAIL
GQ_SEMAPHOREFAIL
GQ_UNIMPLEMENTED

GQ_PeekQueue

Name

GQ_PeekQueue - allows the user to peek at the head element of the specified queue without removing it.

Synopsis

CONDITION GQ_PeekQueue(int qid, void *element)

qid	queue identifier of the queue whose head element is to be peeked at.
element	Pointer to user allocated element in which the element at the head of the specified queue will be copied and retruned to user.

Description

This routine allows users to peek at the head element of the specified queue without removing it from the queue.

Notes

This operation is atomic in the sense that no element can be enqueued or dequeued while the peek operation is in progress. The user is required to allocate memory for the element.

Return Values

GQ_NORMAL
GQ_NOMEMORY
GQ_NOPENQUEUE
GQ_SEMAPHOREFAIL
GQ_QUEUEEMPTY
GQ_UNIMPLEMENTED

GQ_PrintQueue

Name

GQ_PrintQueue - provide a mechanism to dump the contents of the specified queue.

Synopsis

CONDITION GQ_PrintQueue(int qid, void (print_func(void*)))

qid	queue identifier whose contents are to be printed
print_func	the user supplied function that knows how to print a queue element

Description

This function is a utility principally meant for use by developers. Since the queuing mechanisms knows nothing about an element's structure (except its size), a printing routine must be passed as an input parameter so that each element can be printed.

Return Values

```
GQ_NORMAL
GQ_NOPENQUEUE
GQ_SEMAPHOREFAIL
GQ_QUEUEEMPTY
GQ_UNIMPLEMENTED
```