This chapter describes how your application can directly add elements to and remove them from an operating-system queue. The Macintosh Operating System stores some of the information it uses in data structures called queues. The Queue Utilities allow you to manipulate those queues directly by adding and removing elements.

Ordinarily, you do not need to use the Queue Utilities. The Operating System itself is responsible for managing the various operating-system queues that it creates internally, and you should manipulate those queues only indirectly. For example, to add an element to the notification queue maintained by the Notification Manager, you should call the `NMInstall` function. To remove an element from that queue, you should call the `NMRemove` function. But if you discover some unusual need for adding or removing such elements directly, you can use the Queue Utilities routines. In addition, you can use the Queue Utilities routines for directly manipulating queues that you create.

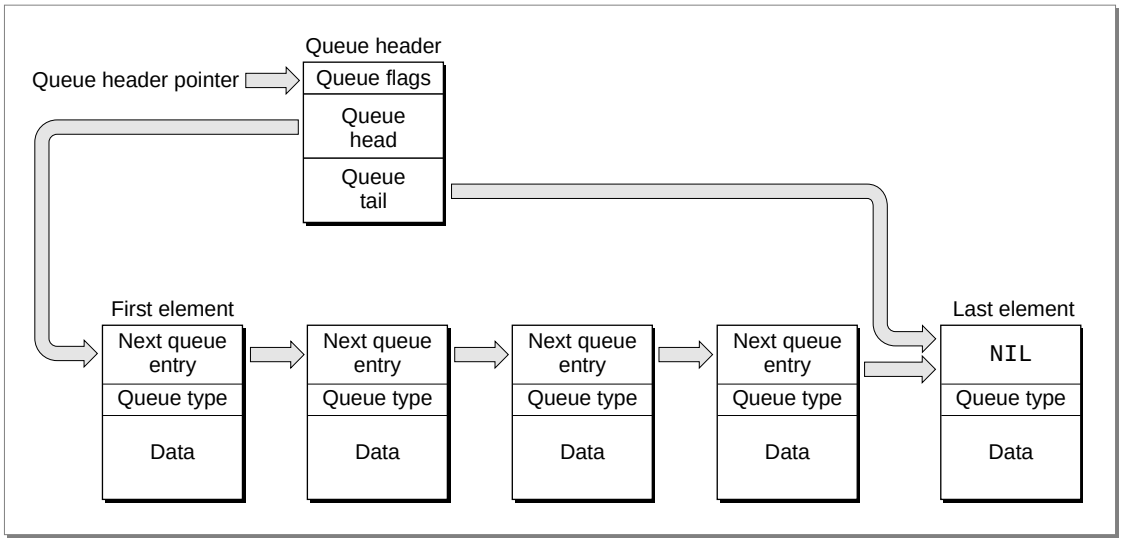
This chapter describes the general structure of operating-system queues and then

- lists the routines your application should use to manipulate an operating-system queue indirectly
- shows how your application can use the Queue Utilities for directly manipulating queues that you create.

About Queues

The Macintosh Operating System uses operating-system queues to keep track of a wide variety of items, including VBL tasks, notifications, I/O requests, events, mounted volumes, and disk drives (or other block-formatted devices). A ***queue*** is a list of identically structured entries linked together by pointers. A single entry in a queue is called a ***queue element.*** Figure 6-1 illustrates the general structure of an operating-system queue.

Figure 6-1 An operating-system queue



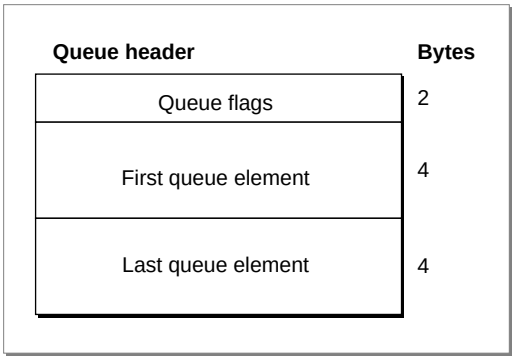
As you can see, the addresses of the first and last elements in the queue are stored in a ***queue header.*** The queue header also contains some queue flags, which contain information about the queue.

Each queue element contains the address of the next element in the queue (or the value `NIL` if there is no next element), an indication of the type of queue to which the next element belongs, and some data. The exact format and size of the data differs among the various queue types. In some cases, the data in the queue element contains the address of a routine to be executed. Table 6-1 on page 6-7 lists the different types of operating-system queues used by the Macintosh Operating System.

The Queue Header

The queue header is the head of a list of identically structured entries linked together by pointers. Figure 6-2 shows the format of a queue header.

Figure 6-2 The format of a queue header



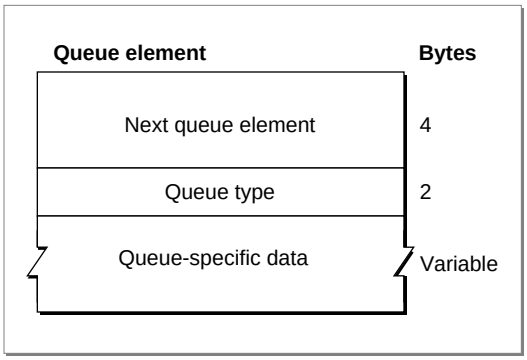
A queue header is a record defined by a data structure of type `QHdr`, which contains three fields: flags, a pointer to the first element in the queue (`qHead`), and a pointer to the last element in the queue (`qTail`). The flags field contains information specific to each queue. Ordinarily, these flags are for use by the system software only, and your application should not need to read or manipulate these flags. The `qHead` field is a pointer to the first element in a queue, and the `qTail` field is a pointer to the last element in a queue. If the queue has no elements, both of these fields are set to `NIL`. Thus, if you have access to a variable `myQueueHdr` of type `QHdrPtr`, you can access the corresponding first queue element of a non-empty queue with myQueueHdr^.qHead^ and access the last element with myQueueHdr^.qTail^.

Each queue element itself is a record of type `QElem`, which is described in the next section.

The Queue Element

The exact format of a queue element is not the same for all types of operating-system queues; thus, a queue element is defined by a variant record that is a data structure of type `QElem`. Figure 6-3 shows the format of a queue element.

Figure 6-3 The format of a queue element



Each queue element contains two fixed fields: a pointer to the next element in the queue (`qLink`), a value describing the queue type (`qType`), and a variable data field specific to each queue type.

The `qLink` field contains a pointer to the next element in the queue. All queue elements are linked through these pointers. Each pointer points to the `qLink` field in the next queue element, and the last queue element contains a `NIL` pointer. The data type of the pointer to the next queue element is always `QElemPtr`.

The `qType` field contains an integer that usually designates the queue type; for example, `ORD(evType)` for the event queue. Table 6-1 contains a list of all the supported operating-system queue types.

Table 6-1 Operating-system queue types

Constant	Queue type	Description
`vType`	Vertical retrace queue	A list of tasks to be executed during VBL interrupts
`ioQType`	File I/O queue (or driver I/O queue)	A list of parameter blocks for all asynchronous routines awaiting execution
`drvQType`	Drive queue	A list of all disk drives connected to the computer
`evType`	Event queue	A list of pending events
`fsQType`	Volume control block queue	A list of volume control blocks for each mounted volume
`sIQType`	Slot interrupt queue	A list of slot interrupts
`dtQType`	Deferred task queue	A list of deferred tasks
`nmQType`	Notification queue	A list of notification requests
`slpQType`	Sleep queue	A list of routines to be notified before a Macintosh Portable or a PowerBook is put into the sleep state

Often, you need to set the `qType` field of a queue element to an appropriate value before installing the queue element. However, some operating-system queues use this field for different purposes. For example, the Time Manager uses an operating-system queue to track Time Manager tasks. In the high bit of this field, the revised Time Manager places a flag to indicate whether a task timer is active. The Time Manager (along with other parts of the Operating System that use this field for their own purposes) shields you from the implementation-level details of operating a queue. Indeed, there is no way for you to access a Time Manager queue directly, and the `QElem` data type does not support access of Time Manager task records from Time Manager queue elements.

The third field contains data that is specific to the type of operating-system queue to which the queue element belongs. For example, a queue element in a vertical retrace queue, maintained by the Vertical Retrace Manager, includes information about the task procedure to be called, the number of interrupts, and the task phase. A queue element in a notification queue, maintained by the Notification Manager, includes information about the alert box, the sound response, the item to be marked in the Application menu, a response procedure, and some reserved values. Figure 6-4 shows the format of these two different types of queue elements.

Figure 6-4 Formats of a vertical retrace queue element and a notification queue element

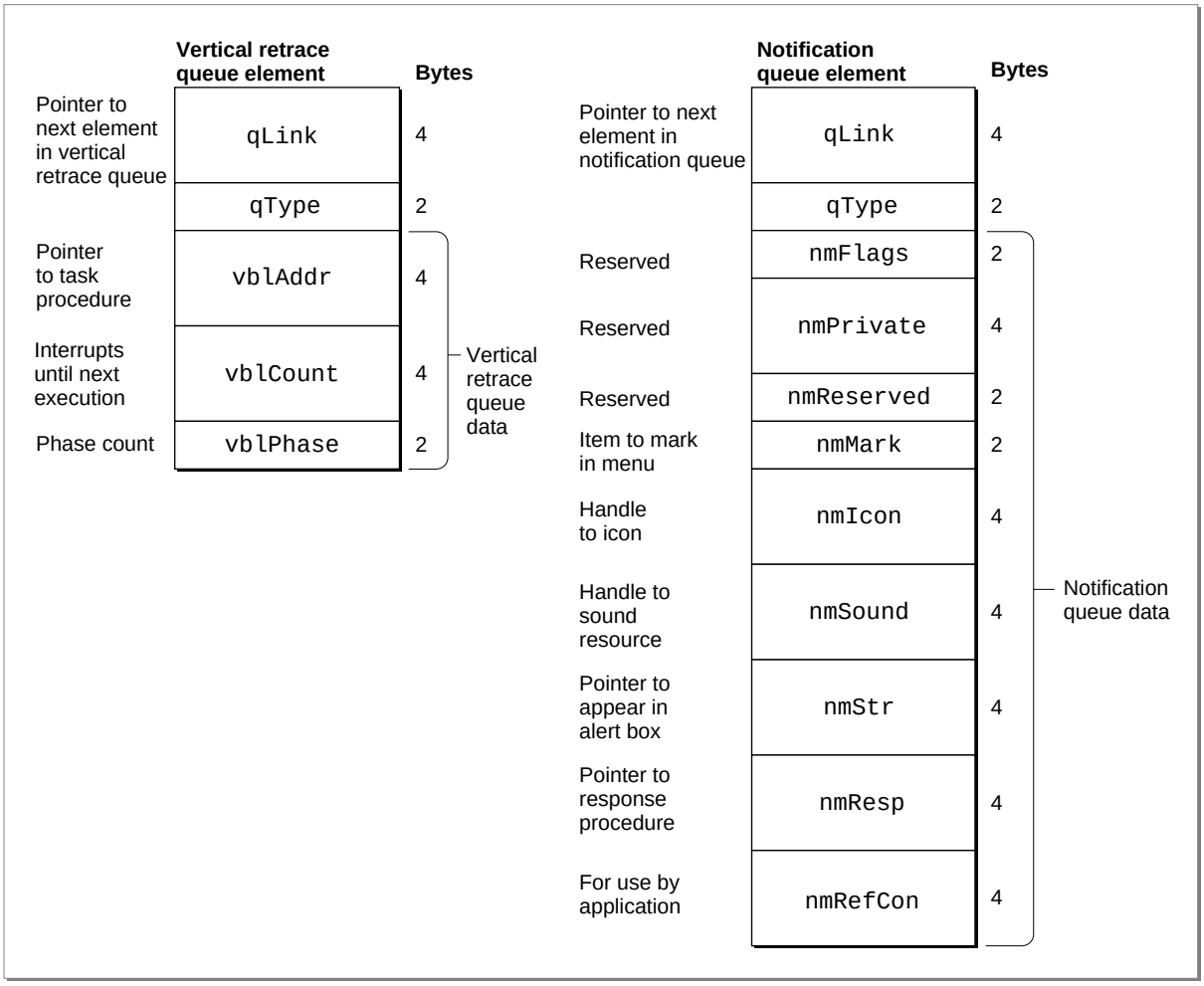


Figure 6-4 illustrates how the format and size of an operating-system queue element can vary because of the variable data field. For example, an element of type `vType` (a vertical retrace queue element) uses 10 bytes for VBL-specific data, whereas an element of type `nmType` (a notification queue element) uses 30 bytes for notification-specific data. All operating-system queue elements use at least 6 bytes: 4 bytes to store a pointer to the next element in the queue and 2 bytes to store a value indicating the queue type.

Using the Queue Utilities

The Queue Utilities provide routines for directly adding elements to a queue and removing them from a queue. The `Enqueue` procedure lets you add elements to the end of a queue, and the `Dequeue` function lets you remove elements from a queue.

You should manipulate an operating-system queue used by the Macintosh Operating System indirectly, by calling special-purpose routines. For example, to install a deferred task into a deferred task queue, your application should use the `DTInstall` function instead of the `Enqueue` procedure. However, if you create your own queues, you can use the `Enqueue` procedure and the `Dequeue` function to manipulate these queues directly. This section describes how to

- search for an element in an operating-system queue
- add an element to an operating-system queue
- remove an element from an operating-system queue

Searching for an Element in an Operating-System Queue

You can search an operating-system queue for a specific element or elements. For example, Listing 6-1 shows a simplified way to search a drive queue for all the drives connected to the computer. The application-defined function, `MySearchDriveQueue`, walks through the drive queue searches for all connected drives. If it finds any, it calls the application-defined function `DoDisplayDriveInfo` to display information about the connected drive.

Listing 6-1 Searching for drives in the drive queue

```
FUNCTION MySearchDriveQueue: Boolean;
VAR
   driveQHdr:     QHdrPtr;
   result:        Boolean;
BEGIN
   result := FALSE;                          {assume no drivers in the queue}
   driveQHdr := GetDrvQHdr;                  {get the drive queue header}
   driveQPtr := DrvQElPtr(driveQHdr^.qHead);
   WHILE (driveQPtr <> NIL) DO               {while drive queue is not empty}
   BEGIN
      result := TRUE;                        {found a drive}
      DoDisplayDriveInfo(driveQPtr);         {display drive information}
                                             {go to next drive in the queue}
      driveQPtr := DrvQElPtr(driveQPtr^.qLink);
   END; {of while}
   MySearchDriveQueue := result;             {return result of search}
END;
```

Adding Elements to an Operating-System Queue

You should avoid direct manipulation of an operating-system queue used by the Macintosh Operating System. Your application should, when possible, use the installation routines in Table 6-2 to add new elements to an operating-system queue.

Table 6-2 Installation routines for operating-system queue elements

Queue element	Installation routine	Additional information
Slot-based VBL task	`SlotVInstall`	The chapter "Vertical Retrace Manager" in *Inside Macintosh: Processes*
System-based VBL task	`VInstall`	The chapter "Vertical Retrace Manager" in *Inside Macintosh: Processes*
Parameter block for an asynchronous routine awaiting execution	*	The chapter "File Manager" in *Inside Macintosh: Files*
Disk drive	`AddDrive`	The chapter "File Manager" in *Inside Macintosh: Files*
Event	`PPostEvent` and `PostEvent`	The chapter "Event Manager" in *Inside Macintosh: Macintosh Toolbox Essentials*
Volume control block	*	The chapter "File Manager" in *Inside Macintosh: Files*
Deferred task	`DTInstall`	The chapter "Deferred Task Manager" in *Inside Macintosh: Processes*
Slot interrupt	`SIntInstall`	The chapter "Slot Manager" in *Inside Macintosh: Devices*
Notification request	`NMInstall`	The chapter "Notification Manager" in *Inside Macintosh: Processes*
Sleep	`SleepQInstall`	The chapter "Power Manager" in *Inside Macintosh: Devices*

* No comparative installation routine available.

IMPORTANT

It is not recommended that you directly add elements to an operating-system queue used by the Macintosh Operating System. If at all possible, your application should use the installation routines provided by the various managers. ▲

If you have created a queue for your own use, you can use the `Enqueue` procedure to add a new element to your queue. For example, Listing 6-2 presents the application-defined procedure `DoAddBankCustomer`, which uses the `Enqueue` procedure for directly installing a customer into a bank-teller queue.

Listing 6-2 Using the Enqueue procedure to add a bank customer to a teller queue

```
PROCEDURE DoAddBankCustomer(myQueueHdrPtr: QHdrPtr,
                            Var bankCustomer: MyCustomerRecord);
BEGIN
   WITH bankCustomer^ DO                {get bank customer data}
   BEGIN
      qType := kTellerQType;            {queue type for the bank-teller queue}
      account := MyGetNextAccount;      {get account number}
      action := MyGetBankAction;        {get action to perform}
      amount := MyGetAmount;            {get the amount}
   END;
   Enqueue(QElemPtr(bankCustomer), myQueueHdrPtr);    {add customer to queue}
END;
```

Note that you are responsible for allocating memory for a queue element before you insert into a queue and for deallocating that memory when you remove the queue element.

Removing Elements From an Operating-System Queue

This section describes how your application can remove elements from an operating-system queue. Whenever possible, your application should use the removal routines listed in Table 6-3 to remove elements indirectly from an operating-system queue used by the Macintosh Operating System.

Table 6-3 Removal routines for operating-system elements

Queue element	Removal routine	Additional information
Slot-based VBL task	`SlotVRemove`	The chapter "Vertical Retrace Manager" in *Inside Macintosh: Processes*
System-based VBL task	`VRemove`	The chapter "Vertical Retrace Manager" in *Inside Macintosh: Processes*
Parameter block for an asynchronous routine awaiting execution	*	The chapter "File Manager" in *Inside Macintosh: Files*
Disk drive	*	The chapter "File Manager" in *Inside Macintosh: Files*
Event	`WaitNextEvent`	The chapter "Event Manager" in *Inside Macintosh: Macintosh Toolbox Essentials*
Volume control block	*	The chapter "File Manager" in *Inside Macintosh: Files*
Deferred task	*	The chapter "Deferred Task Manager" in *Inside Macintosh: Processes*
Slot interrupt	`SIntRemove`	The chapter "Slot Manager" in *Inside Macintosh: Devices*
Notification request	`NMRemove`	The chapter "Notification Manager" in *Inside Macintosh: Processes*
Sleep	`SleepQRemove`	The chapter "Power Manager" in *Inside Macintosh: Devices*

* No comparative removal routine available.

IMPORTANT

It is not recommended that you directly remove queue elements from an operating-system queue used by the Macintosh Operating System. If at all possible, your application should use the removal routines provided by the various managers. ▲

If you have created a queue for your own use, you can use the `Dequeue` function to remove elements from that queue.

Listing 6-3 shows the application-defined function `DoRemoveBankCustomer`, which uses the `Dequeue` procedure for directly removing the first customer from a bank-teller queue. The `DoRemoveBankCustomer` function returns `TRUE` if it removes the customer.

Listing 6-3 Using `Dequeue` to remove the first customer in the bank-teller queue

```
FUNCTION DoRemoveBankCustomer (VAR myQueueHdr: QHdr): BOOLEAN;
VAR
   bankCustomerPtr: MyCustomerRecordPtr;
   customerRemoved: Boolean;

BEGIN
   customerRemoved := FALSE;
   bankCustomerPtr := MyCustomerRecordPtr(myQueueHdr.qHead);
   IF bankCustomerPtr <> NIL THEN      {Check for non-empty queue}
   BEGIN
      Dequeue(QElemPtr(bankCustomerPtr),&myQueueHdr) {remove customer}
      customerRemoved := TRUE;
   END; {of queue not empty}
   DoRemoveCustomer := customerRemoved;
END;
```

Queue Utilities Reference

This section describes the data structures of operating-system queues and two Queue Utilities routines for directly adding elements to and removing them from queues that you create.

Data Structures

Each operating-system queue created and maintained by the Macintosh Operating System consists of a queue header and a linked list of queue elements. This section describes the structure of queue headers and queue elements.

Queue Headers

A queue header is a block of data that contains information about a queue. The `QHdr` data type defines the structure of a queue header.

```
TYPE QHdr =
RECORD
   qFlags:     Integer;       {information on queue}
   qHead:      QElemPtr;      {pointer to first queue entry}
   qTail:      QElemPtr;      {pointer to last queue entry}
END;
```

Field descriptions

`qFlags`	Queue flags. This field contains information that is different for each queue type. Ordinarily, these flags are reserved for use by system software.
`qHead`	A pointer to the first element in the queue. If a queue has no elements, this field is set to `NIL`.
`qTail`	A pointer to the last element in the queue. If a queue has no elements, this field is set to `NIL`.

Queue Elements

A queue element is a single entry in a queue. The exact structure of an element in an operating-system queue depends on the type of the queue. The different queue types that are accessible to your application are defined by the `QTypes` data type.

```
TYPE QTypes =
   (dummyType,       {reserved}
   vType,            {vertical retrace queue type}
   ioQType,          {file I/O or driver I/O queue type}
   drvQType,         {drive queue type}
   evType,           {event queue type}
   fsQType,          {volume-control-block queue type}
   sIQType,          {slot interrupt queue type}
   dtQType,          {deferred task queue type}
   {nmType,}         {notification queue type}
   {slpQType}        {sleep queue type}
   );
```

Each of these enumerated queue types determines a different type of queue element. The `QElem` data type defines the available queue elements.

```
TYPE QElem =
RECORD
   CASE QTypes OF
      vType:      (vblQElem:  VBLTask);
      ioQType:    (ioQElem:   ParamBlockRec);
      drvQType:   (drvQElem:  DrvQEl);
      evType:     (evQElem:   EvQEl);
      fsQType:    (vcbQElem:  VCB);
      dtQType:    (dtQElem:   DeferredTask);
      {siQType:   (siQElem:   SlotIntQElement);}
      {nmType:    (nmQElem:   NMRec);}
      {slpQType:  (slpQElem:  SleepQRec);}
END;
QElemPtr = ^QElem;
```

Data type	Additional information
`VBLTask`	The chapter "Vertical Retrace Manager" in *Inside Macintosh: Processes*
`ParamBlockRec`	The chapter "File Manager" in *Inside Macintosh: Files*
`DrvQEl`	The chapter "File Manager" in *Inside Macintosh: Files*
`EvQEl`	The chapter "Event Manager" in *Inside Macintosh: Macintosh Toolbox Essentials*
`VCB`	The chapter "File Manager" in *Inside Macintosh: Files*
`DeferredTask`	The chapter "Deferred Task Manager" in *Inside Macintosh: Processes*
`SlotIntQElement`	The chapter "Slot Manager" in *Inside Macintosh: Devices*
`NMRec`	The chapter "Notification Manager" in *Inside Macintosh: Processes*
`SleepQRec`	The chapter "Power Manager" in *Inside Macintosh: Devices*

Routines

The Queue Utilities provide two routines: `Enqueue` and `Dequeue`. The `Enqueue` procedure allows you to add queue elements directly to an operating-system queue, and the `Dequeue` function allows you to remove the element. Ordinarily, these routines are used only by system software. If possible, you should manipulate an operating-system queue indirectly, by calling special-purpose routines. For example, to install a task record into a slot-based vertical retrace queue, your application should use the `SlotVInstall` function (provided by the Vertical Retrace Manager) instead of the `Enqueue` procedure. In addition, you can use the Queue Utilities routines for directly manipulating queues that you create.

Enqueue

You can use the `Enqueue` procedure to add elements directly to an operating-system queue or a queue that you create.

```
PROCEDURE Enqueue (qElement: QElemPtr; qHeader: QHdrPtr);
```

`qElement` A pointer to the queue element to add to a queue.

`qHeader` A pointer to a queue header.

DESCRIPTION

The `Enqueue` procedure adds the queue element specified by `qElement` parameter to the end of the queue specified by the `qHeader` parameter. The specified queue header is updated to reflect the new queue element.

SPECIAL CONSIDERATIONS

Because interrupt routines are likely to manipulate operating-system queues, interrupts are disabled for a short time while the specified queue is updated. You can call the `Enqueue` procedure at interrupt time. Whenever possible, use the installation routines listed in Table 6-2 on page 6-10 instead of the `Enqueue` procedure.

ASSEMBLY-LANGUAGE INFORMATION

The registers on entry and exit for the `Enqueue` procedure are

Registers on entry

A0 Pointer to the queue element to be added

A1 Pointer to the queue header

Registers on exit

A1 Pointer to the queue header

SEE ALSO

For a description of the `QElem` record, see page 6-14; for a description of the `QHdr` record, see page 6-13.

Dequeue

You can use the `Dequeue` function to remove a queue element directly from an operating-system queue or from a queue that you have created.

```
FUNCTION Dequeue (qElement: QElemPtr; qHeader: QHdrPtr): OSErr;
```

`qElement` A pointer to a queue element to remove from a queue.

`qHeader` A pointer to a queue header.

DESCRIPTION

The `Dequeue` function attempts to find the queue element specified by the `qElement` parameter in the queue specified by the `qHeader` parameter. If `Dequeue` finds the

element, it removes the element from the queue, adjusts the other elements in the queue accordingly, and returns `noErr`. Otherwise, it returns `qErr`, indicating that it could not find the element in the queue. The `Dequeue` function does not deallocate the memory occupied by the queue element.

SPECIAL CONSIDERATIONS

The `Dequeue` function disables interrupts as it searches through the queue for the element to be removed. The time during which interrupts are disabled depends on the length of the queue and the position of the entry in the queue. The `Dequeue` function can be called at interrupt time. Whenever possible, use the removal routines listed in Table 6-3 on page 6-12 instead the `Dequeue` function.

ASSEMBLY-LANGUAGE INFORMATION

The registers on entry and exit for the `Dequeue` function are

Registers on entry

A0	Pointer to the queue element to be removed
A1	Pointer to the queue header

Registers on exit

A1	Pointer to the queue header
D0	Result code

RESULT CODES

`noErr`	0	No error
`qErr`	–1	Entry is not in specified queue

SEE ALSO

For a description the `QElem` record, see page 6-14; for a description of the `QHdr` record, see page 6-13.

Summary of the Queue Utilities

Pascal Summary

Constants

```
CONST    {queue types}
   vType    = 1;           {vertical retrace queue type}
   ioQType  = 2;           {file I/O or driver I/O queue type}
   drvQType = 3;           {drive queue type}
   evType   = 4;           {event queue type}
   fsQType  = 5;           {volume-control-block queue type}
   sIQType  = 6;           {slot interrupt queue type}
   dtQType  = 7;           {deferred task queue type}
   nmType   = 8;           {notification queue type}
   slpQType = 16;          {sleep queue type}
```

Data Types

```
TYPE QHdr =    {queue header record}
   RECORD
      qFlags:     Integer;       {information on queue}
      qHead:      QElemPtr;      {pointer to the first queue element}
      qTail:      QElemPtr;      {pointer to the last queue element}
   END;
   QHdrPtr = ^QHdr;

   QTypes = (  {queue types}
      dummyType,                 {reserved}
      vType,                     {vertical retrace queue type}
      ioQType,                   {file I/O or driver I/O queue type}
      drvQType,                  {drive queue type}
      evType,                    {event queue type}
      fsQType,                   {volume-control-block queue type}
      sIQType,                   {slot interrupt queue type}
      dtQType,                   {deferred task queue type}
```

```
      {nmType,}                  {notification queue type}
      {slpQType}                 {sleep queue type}
   );

   QElem =     {queue element record}
   RECORD
      CASE QTypes OF
         dtQType:    (dtQElem:   DeferredTask);    {deferred task }
                                                   { queue element}
         vType:      (vblQElem:  VBLTask);         {vertical retrace }
                                                   { queue element}
         ioQType:    (ioQElem:   ParamBlockRec);   {file I/O queue element}
         drvQType:   (drvQElem:  DrvQEl);          {drive queue element}
         evType:     (evQElem:   EvQEl);           {event queue element}
         fsQType:    (vcbQElem:  VCB);             {volume-control-block }
                                                   { queue element}
         {sIQType:   (siQElem:   SlotIntQElement;} {slot interupt }
                                                   { queue element}
         {nmType:    (nmQElem:   NMRec);}          {notification }
                                                   { queue element}
         {slpQType:  (slpQElem:  SleepQRec);}      {sleep queue element}
   END;
   QElemPtr = ^QElem;
```

Routines

```
PROCEDURE Enqueue           (qElement: QElemPtr; qHeader: QHdrPtr);
FUNCTION Dequeue            (qElement: QElemPtr; qHeader: QHdrPtr): OSErr;
```

C Summary

Constants

```
enum {      /*queue types*/
   vType    = 1,              /*vertical retrace queue type*/
   ioQType  = 2,              /*file I/O or driver I/O queue type*/
   drvQType = 3,              /*drive queue type*/
   evType   = 4,              /*event queue type*/
   fsQType  = 5,              /*volume-control-block queue type*/
```

```
   sIQType  = 6,              /*slot interrupt queue type*/
   dtQType  = 7,              /*deferred task queue type*/
};

enum {      /*value for the notification queue type*/
   nmType   = 8               /*notification queue type*/
};

enum {      /*value for the sleep queue type*/
   slpQType = 16              /*sleep queue type*/
};
```

Data Types

```
struct QHdr {  /*queue header record*/
   short          qFlags;     /*information on queue*/
   QElemPtr       qHead;      /*pointer to the first queue element*/
   QElemPtr       qTail;      /*pointer to the last queue element*/
};
typedef struct QHdr QHdr;
typedef QHdr *QHdrPtr;

typedef unsigned short QTypes;   /*queue types*/

struct QElem { /*queue element record*/
   struct QElem   *qLink;     /*pointer to the next queue element*/
   short          qType;      /*type of queue element*/
   short          qData[1];   /*variable array of data; type of data and */
                              /* length depend on the queue type, */
                              /* specified in the qType field*/
};
typedef struct QElem QElem;
typedef QElem *QElemPtr;
```

Routines

```
pascal void Enqueue         (QElemPtr qElement, QHdrPtr qHeader);
pascal OSErr Dequeue        (QElemPtr qElement, QHdrPtr qHeader);
```

Assembly-Language Summary

QHdr Data Structure

0	`qFlags`	word	information on queue
2	`qHead`	long	pointer to first queue entry
6	`qTail`	long	pointer to last queue entry

QElem Data Structure

0	`qLink`	long	pointer to the next queue element
4	`qType`	word	type of queue element
6	`qData`	word	variable array of data; type of data and length depend on the queue type, specified in the `qType` field

Result Codes

`noErr`	0	No error
`qErr`	–1	Entry is not in specified queue