

# Message Handling APIs (V5R2)

---

## Table of Contents


- [Message Handling APIs](#)
- [Message Handling Terms and Concepts](#)
- APIs
  - [Change Exception Message](#) (QMHCHGEM)
  - [Control Job Log Output](#) (QMHCTLJL)
  - [List Job Log Messages](#) (QMHLJOBL)
  - [List Nonprogram Messages](#) (QMHLSTM)
  - [Move Program Messages](#) (QMOMOVP)
  - [Open List of Job Log Messages](#) (QGYOLJBL)
  - [Open List of Messages](#) (QGYOLMSG)
  - [Promote Message](#) (QMHPRMM)
  - [Receive Nonprogram Message](#) (QMHRMVM)
  - [Receive Program Message](#) (QMHRMVP)
  - [Remove Nonprogram Messages](#) (QMHRMVM)
  - [Remove Program Messages](#) (QMHRMVP)
  - [Resend Escape Message](#) (QMHRSNEM)
  - [Retrieve Message](#) (QMHRMVM)
  - [Retrieve Message File Attributes](#) (QMHRMFAT)
  - [Retrieve Nonprogram Message Queue Attributes](#) (QMHRMQAT)
  - [Retrieve Request Message](#) (QMHRMVRQ)
  - [Send Break Message](#) (QMHSNDBM)
  - [Send Nonprogram Message](#) (QMHSNDM)
  - [Send Program Message](#) (QMHSNDPM)
  - [Send Reply Message](#) (QMHSNDRM)
  - [Send Scope Message](#) (QMHSNDSM)
- Exit programs
  - [Break Handling](#)
  - [Default Handling](#)
  - [»Reply Handling«](#)

# Message Handling APIs

The Message Handling APIs let your applications work with iSeries messages. They allow you to do the following:

- Send messages to users or programs
- Receive messages from a message queue
- Handle errors
- Return message and message queue information
- Return message description and message file information

Before using the message handling APIs, read the following material:

- [Message Handling Terms and Concepts](#). This section briefly describes the elements of iSeries messages and message handling.
- [CL Programming](#).  For complete background information, see the sections that discuss defining and working with messages.


The Message Handling APIs are:

- [Change Exception Message](#) (QMCHGEM) changes an exception message on a call message queue. This API allows the current program to perform a variety of actions on an exception message that was sent to its caller, a previous caller, or itself.
- [Control Job Log Output](#) (QMHCTLJL) controls the production of a job log when the related job ends or when the job message queue becomes full and the print-wrap option is in effect for the job.
- [List Job Log Messages](#) (QMHJLJBL) lists messages from the job message queue of a job. This function gets the requested message information and returns it in a user space in the format specified in the parameter list.
- [List Nonprogram Messages](#) (QMHJLSTM) lists messages from one or two nonprogram message queues. This function gets the requested message information and returns it in a user space in the format specified in the parameter list.
- [Move Program Messages](#) (QMHMOVPM) moves messages from one call message queue to the message queue of an earlier call stack entry in the call stack. This is especially useful for error handling.
- [Open List of Job Log Messages](#) (QGYOLJBL) lists messages from a job log. Returned messages are sorted by their sending date and time unless the message being listed is a reply message to an inquiry, a sender's copy, or a notify type of message.
- [Open List of Messages](#) (QGYOLMSG) provides information on messages for the current user, a specific user, or one specific nonprogram message queue.
- [Promote Message](#) (QMHPRMM) promotes an escape or status message that was sent to a call stack entry. That is, the message is handled and replaced with a new escape or status message. You may promote an escape message to another escape message or to a status message. You may promote a status message to an escape message or to another status message.
- [Receive Nonprogram Message](#) (QMHRCVVM) receives a message from a nonprogram message queue, providing information about the sender of the message as well as the message itself. This API is similar in function to the Receive Message (RCVMSG) command with the MSGQ parameter.
- [Receive Program Message](#) (QMHRCVPM) receives a message from a call message queue, and provides information about the sender of the message as well as the message itself. This API is

similar in function to the Receive Message (RCVMSG) command with the PGMQ parameter.

- [Remove Nonprogram Messages](#) (QMHRMVM) removes messages from nonprogram message queues. This API is similar in function to the Remove Message (RMVMSG) command with the MSGQ parameter.
- [Remove Program Messages](#) (QMHRMVPM) removes messages from call message queues. This API is similar in function to the Remove Message (RMVMSG) command with the PGMQ parameter.
- [Resend Escape Message](#) (QMHRSNEM) resends an escape message from one call message queue to the message queue of the previous call stack entry in the call stack.
- [Retrieve Message](#) (QMHRMVM) retrieves the message text and other elements of a predefined message stored in a message file on your system. This API is similar to the Retrieve Message (RTVMSG) command.
- [Retrieve Message File Attributes](#) (QMHRMFAT) retrieves information about the attributes of a message file.
- [Retrieve Nonprogram Message Queue Attributes](#) (QMHRMQAT) provides information about the attributes of a nonprogram message queue.
- [Retrieve Request Message](#) (QMHRMVRQ) retrieves request messages from the current job's call message queue.
- [Send Break Message](#) (QMHSNDBM) sends a message to a work station for immediate display, interrupting the work station user's task. You can use break messages to warn users of impending system outages and the like. This API is similar in function to the Send Break Message (SNDBRKMSG) command.
- [Send Nonprogram Message](#) (QMHSNDM) sends a message to a system user or a message queue that is not associated with a specific program. This API is similar in function to the Send Program Message (SNDPGMMSG) command with the TOMSGQ parameter.
- [Send Program Message](#) (QMHSNDPM) sends a message to the message queue of a call stack entry in the call stack. This API is similar in function to the Send Program Message (SNDPGMMSG) command with the TOPGMQ parameter.
- [Send Reply Message](#) (QMHSNDRM) sends a response to an inquiry message. This API is similar in function to the Send Reply (SNDRPY) command.
- [Send Scope Message](#) (QMHSNDSM) Sends a scope message that allows a user to specify a program to run when your program or job is completed.


The Message Handling exit programs are:

- [Break Handling](#) provides the capability of handling messages that arrive at a message queue that is in \*BREAK mode.
- [Default Handling](#) provides a default message handling action that can be used if the program that is signaled an escape message does not monitor for or handle the escape message.
- [Reply Handling](#) is called when a reply is sent to an inquiry message. 

# Message Handling Terms and Concepts

This article describes the basic elements of iSeries messages and message handling, including:

- Immediate and predefined messages
- Message types, or the purposes assigned to messages when they are sent
- Message destinations, such as work station displays and call message queues

For details about these topics, see the [CL Programming](#)  book.

## Immediate and Predefined Messages

The OS/400 program uses two basic kinds of messages, immediate and predefined. **Immediate messages**, also known as **impromptu messages**, are created by the sender when they are sent. They consist completely of text that the sender supplies. They do not have identifying codes like CPF2469, and they are not stored in message files on the system.

In contrast, **predefined messages** are created and exist outside the programs that use them. Each predefined message has its own **message description**, which is stored in a message file (object type \*MSGF) on the system. The message description includes the message text and other items, such as message help, the default reply for the message, and its severity level.

Each message description has its own message identifier. The **message identifier** is a 7-character code, such as CPF2469, that refers to the message description as a whole. It is displayed with the text of the message and lets programs refer to the message easily. The term message identifier is usually abbreviated to MSGID in items such as CL commands.

Predefined messages have two types of text, message text and message help. The **message text** is the version of the predefined message that is first displayed at a work station or logged in a job log. The message text is also known as **first-level text**. It briefly describes a condition. **Message help**, also known as **second-level text**, is available to the user on request. It usually describes the cause of the condition and details any corrective action the user should take.

The message text and message help for predefined messages can consist of only constant text, only substitution variables, or both constant text and substitution variables. are variables for items such as file names in the message text or message help. They allow the message to better describe the recipient's individual circumstances. Substitution variables are also known as **replacement data**. **Substitution variables** The specific values you assign to these variables are called **message data**.

When either an immediate or predefined message is sent, the command or API used to send it usually assigns it a message key. The **message key**, also called the **message reference key**, is a unique string of characters that identifies a particular instance of a message in a queue. (In contrast, a message identifier identifies a message as it is stored in a message file, not as it is sent.) You can use the message key to refer to a specific instance of a message in order to move, receive, reply to, resend, or remove it.

## Message Types

The OS/400 program divides messages into types according to their use. These **message types** are categories based on the message's purpose. The sender of the message determines its type when sending the message. The message handling APIs use these message types in defining and working with messages:

**Completion (\*COMP).** Reports the successful completion of a task.

**Diagnostic (\*DIAG).** Describes errors in processes or input data. When an error occurs, a program usually sends an escape message, which causes the task to end abnormally. One or more diagnostic messages can be sent before the escape message to describe the error.

**Escape (\*ESCAPE).** Indicates a condition causing a program to end abnormally, without completing its work.

**Exception (\*EXCP).** Indicates a condition causing a program to end abnormally, without completing its work. An exception message can be either an escape or a notify message. The exception message type and the special value \*EXCP are used only with the Receive Program Message (QMHRCVPM) API.

**Informational (\*INFO).** Conveys information *without* asking for a reply.

**Inquiry (\*INQ).** Conveys information *and* asks for a reply.

**Notify (\*NOTIFY).** Describes a condition in the sending program requiring corrective action or a reply.

**Reply (\*RPY).** Responds to an inquiry or notify message.

**Request (\*RQS).** Requests a function from the receiving program.

**Sender's copy (\*COPY).** Is a copy of an inquiry or notify message. This copy is kept by the sender of the inquiry or notify message.

**Scope (\*SCOPE).** Specifies a program to run when the program this message is sent to completes. If the message is sent to \*EXT the program is to run when the job completes.

**Status (\*STATUS).** Describes the status of work being done by a program.

## Message Destinations

All iSeries messages are sent to message queues. The system provides the following message queues:

- A work station message queue for each work station on the system.
- A user profile message queue for each enrolled user.
- A job message queue for each job running on the system. The job message queue consists of two or more message queues:

1. An **external message queue (\*EXT)** to send messages between an interactive job and the work station user.

Any thread in a multithreaded job can send messages to the external message queue. Each thread can receive or remove only those messages sent to the external message queue from that thread.

2. A set of **call message queues** corresponding to the calls in the job. Each call within a job has a call message queue. If a call stack entry appears in the call stack twice, it has two separate call message queues.

A call message queue is also known as a program message queue. However, in the ILE model, you can call procedures as well as programs. Both ILE procedure message queues

and OPM program message queues are referred to as call message queues. A multithreaded job has a separate set of call message queues for each thread that is running in the job. Each thread can access only the messages for the call message queues within its own thread; messages cannot be sent to, received from, or removed from call message queues in other threads.

**Note:** An ILE procedure has an associated call message queue only if the related compiler does not place the procedure inline. An inline procedure does not have a call message queue, and because of this, cannot be referenced on any message handler API. For example, you cannot send a message to an inline procedure by specifying the name of that inline procedure.

3. All message queues other than external and call message queues are considered **nonprogram message queues**.

- The system operator's message queue, QSYSOPR.
- The history log, QHST.

**Note:** A message sent to a message queue remains on the queue until you use a command or API to remove it from the queue or move it to another queue. **Removing a message** from a message queue deletes that one instance of the message from the system. After removal, you can no longer receive or otherwise work with that instance of the message. However, other instances of the message might still be available in the same or different message queues. The message description of a predefined message still exists in a message file.

This exit program could be called from an exit point within a multithreaded job and should be written to be threadsafe.

---

[Top](#) | [Message Handling APIs](#) | [APIs by category](#)

# Change Exception Message (QMHCHGEM) API

## Required Parameter Group:

1	Invocation pointer	Input	Pointer
2	Call stack counter	Input	Binary(4)
3	Message key	Input	Char(4)
4	Modification option	Input	Char(10)
5	Reply text	Input	Char(*)
6	Reply text length	Input	Binary(4)
7	Error code	I/O	Char(*)

Default Public Authority: \*USE

Threadsafe: Yes

The Change Exception Message (QMHCHGEM) API changes an exception message on a call message queue. This API allows the current program to perform one of the following actions on an exception message that was sent to its caller, a previous caller, or itself:

- Handle the escape, status, or notify message. If the exception is a notify message that has not already received a reply, return the default reply.
- If it is a notify message that has not been replied to, return a reply to the call stack entry that sent this message. Handle the exception.
- Handle the escape, status, or notify message and remove it from the job log. If the exception is a notify message that has not been replied to, return its default reply or another reply to the call stack entry that sent it.

## Authorities and Locks

None.

## Required Parameter Group

### Invocation pointer

INPUT; POINTER

The invocation pointer to the call stack entry to which the exception message was sent. When using a value other than 0 for the call stack counter parameter, the invocation pointer points to the call stack entry from which to start counting in the call stack. This is the location of the call stack entry that received the exception message that was sent. The call stack entry you specify must be in the call stack. A null invocation pointer may be specified. If this pointer is not set, the call stack entry that called the QMHCHGEM API is used.

### Call stack counter

INPUT; BINARY(4)

A number that identifies the location in the call stack of the call stack entry that received the message you are changing. The number is relative to the call stack entry specified in the invocation pointer parameter. The number indicates how many calls earlier in the call stack the call stack entry is from the one specified in the invocation pointer parameter. Valid values follow:

- 0* Change a message in the message queue of the call stack entry specified in the invocation pointer parameter.
- 1* Change a message in the message queue of the call stack entry that called the call stack entry specified in the invocation pointer parameter.
- n* Change a message in the message queue of the nth call stack entry earlier up the stack from the call stack entry specified in the invocation pointer parameter.

You can use any positive number that offsets to an actual call stack entry in the call stack.

### **Message key**

INPUT; CHAR(4)

The message key of the exception message being changed. This parameter is ignored when \*CHANGEALL or \*CHANGELST is specified for the modification option parameter.

### **Modification option**

INPUT; CHAR(10)

The type of change to be done to the exception message. Valid values follow:

- \*HANDLE* Causes the exception to be handled. No error message is returned if the exception is already handled.  
  
If the exception is a status message, it is immediately removed from the job message queue and is no longer accessible. If the exception is an escape or notify message and is still in the job log, it is still accessible through its message key.  
  
If the exception is a notify message, the default reply is sent. If the notify message already received a reply but was not handled, the exception will still be handled. If the notify message was already handled but had not received a reply, the default reply will still be sent.
- \*CHANGE* Changes escape message to a diagnostic message and handles it. If the escape message has already been handled, it is still changed to a diagnostic message. If the exception is not an escape message, an error is returned to the caller of this API and the exception is not handled or changed.
- \*CHANGEALL* Changes all escape messages that were sent to the specified call stack entry. Each escape message is changed to a diagnostic message and handled. If the escape message has already been handled, it is still changed to a diagnostic message. If any messages are encountered that are not escape messages, they are ignored. When \*CHANGEALL is specified, the message key parameter value is ignored.



*\*CHANGELST* Changes the last escape message that was sent to the specified call stack entry. The escape message is changed to a diagnostic message and handled. If the escape message has already been handled, it is still changed to a diagnostic message. When *\*CHANGELST* is specified, the message key parameter value is ignored.

*\*REPLY* Replies to a notify message and handles it. The default reply or the reply specified in the reply text parameter is returned to the call stack entry that sent the notify message. If the reply text length parameter value is 0, the default reply is sent. The notify message is handled after its reply is sent.

If the notify message has already been handled, the reply is still sent.

If the notify message has already been replied to, an error is returned to the caller of this API, and the exception is not handled.

If the exception is not a notify message, an error is returned to the caller of this API, and the exception is not handled.

*\*REMOVE* Handles the escape, notify, or status exception and removes the message from the job log. If the exception is a notify message, its default reply or the reply specified in the reply text parameter is returned to the call stack entry that sent it. If the reply text length parameter value is 0, the default reply is sent. No error message is returned if the exception is already handled. The exception is immediately inaccessible for any further operations.

If the exception is a notify message that has already been replied to, it is handled and removed from the job log. No error message is returned to the caller of this API.

If the exception is not a notify message, the reply text length parameter value must be 0. If the reply text length parameter value is not 0, an error is returned to the caller of this API. The exception is not handled or removed from the job log.

## Reply text

INPUT; CHAR(\*)

The data that is returned as the reply to a notify message when *\*REPLY* or *\*REMOVE* is specified for the modification option parameter. This data must be compatible with the reply type, reply length, and valid reply values stored in the message description for the message. If it is not compatible, an error is returned to the caller of this API.

This parameter is ignored if:

- The message being changed is not a notify message.
- The modification option is not *\*REPLY* or *\*REMOVE*.
- The reply text length parameter is 0.

## Reply text length

INPUT; BINARY(4)

The length, in bytes, of the data that is returned as the reply to a notify message. This length must be compatible with the reply type and maximum reply length stored in the message description for

the message.

Valid values follow:

- 0 Return the notify message's default reply.
- 1-132 Return the reply specified in the reply text parameter. This is the number of bytes to return.

If the message being changed is not a notify message, or the modification option is not \*REPLY or \*REMOVE, this parameter is ignored.

### **Error code**

I/O; CHAR(\*)

The structure in which to return error information. For the format of the structure, see [Error Code Parameter](#).

## **Error Messages**

<b>Message ID</b>	<b>Error Message Text</b>
CPF24A3 E	Value for call stack counter parameter not valid.
CPF24B4 E	Severe error while addressing parameter list.
CPF24B6 E	Length of &1, not valid for message text or data.
CPF2401 E	Not authorized to library &1.
CPF2410 E	Message key not found in message queue &1.
CPF2411 E	Not authorized to message file &1 in &2.
CPF242D E	Modification option &1 not valid.
CPF242E E	Tried to change message which is not an exception.
CPF242F E	Message type must be ESCAPE for *CHANGE modification option.
CPF2420 E	Reply already sent for inquiry or notify message.
CPF2422 E	Reply not valid.
CPF243A E	Invocation pointer parameter not valid.
CPF2432 E	Cannot send reply to message type other than *INQ or *NOTIFY.
CPF2547 E	Damage to message file QCPFMSG.
CPF2548 E	Damage to message file &1 in &2.
CPF3C90 E	Literal value cannot be changed.
CPF3CF1 E	Error code parameter not valid.
CPF8100 E	All CPF81xx messages could be returned. xx is from 01 to FF.

CPF9830 E      Cannot assign library &1.

CPF9872 E      Program or service program &1 in library &2 ended. Reason code &3.

---

API introduced: V2R1

---

[Top](#) | [Message Handling APIs](#) | [APIs by category](#)

# Control Job Log Output (QMHCTLJL) API

## Required Parameter Group:

1	Output file names structure	Input	Char(65)
2	Output file name structure format	Input	Char(8)
3	Message filter array	Input	Char(*)
4	Number of message filter elements	Input	Binary(4)
5	Qualified error message queue name	Input	Char(20)
6	Error code	I/O	Char(*)

Default Public Authority: \*USE

Threadsafe: Conditional; see [Usage Notes](#).

The Control Job Log Output (QMHCTLJL) API controls the production of a job log when the related job ends or when the job message queue becomes full and the print-wrap option is in effect for the job. The API can only influence the job log for the job in which it is used. The API can be used to control:

- The destination of the job log.
- The amount of message information written to the job log.
- The occurrence of messages in the job log.

To have an effect on the job log, the API must be called before the job ends. It can be called in an initial program for the job.


This API does not produce a job log; rather, the API captures the control information specified on the parameters and retains the information until job log production. The API also prepares the output files for job log production. When job log production occurs, such as at end of job, the control information and the prepared output files direct the job log to the output file.

If the API is not used, normal job log production occurs. That is, all messages in the job message queue for the job are written to a spooled file from which the job log can be printed. If the API is used, no spooled file is produced. Instead, the messages are written into one or two output files, depending upon the amount of information requested. Once the API is used, the options selected remain in effect for the current job until the API is called again. Each time the API is used, the selected options can be changed. The options in effect when the job ends or when the job message queue becomes full are the ones used to produce the job log.

One or two output files can be produced at job log production time.

- The primary output file contains one record for each message selected for inclusion in the job log. A record in the primary output file contains the essential information of the message: message ID, message type, message severity, time sent, sender and receiver information, and the message data.
- Output to a secondary output file can also be produced. Secondary output is optional and is controlled by providing an output file name for the secondary file. Secondary output consists of the first and second level text for a message. The first and second level text has all message data inserted and is an image of the print line. Since there can be more than one print line for first and second level text, there can be more than one secondary record for a message. Thus, for a single message, the primary file contains one record and the secondary file contains one or more records.

For example, if the print image for first level text is 2 print lines and the second level is 4 print lines, there are 6 records produced for the message in the secondary file. With the 1 primary record, there would be a total of 7 records produced for that message.

Detailed information about the formats of the primary output file and secondary output file is provided in the [CL Programming](#)  book.

This API does not influence the DSPJOBLOG CL command when OUTPUT(\*OUTFILE) is specified on the command. If OUTPUT(\*OUTFILE) is specified on the command, the primary job log information is written to the file specified on the command, and no message filtering is performed. If OUTPUT(\*APIDFN) is specified on the command, the files specified by the call to the API are used for output, and message filtering is performed by the call to this API. The error message queue is not used so any errors are returned to the issuer of the DSPJOBLOG command.

If the DSPJOBLOG command is used with the files prepared by this API, note the following:

- The files specified on the API are prepared only once, when the API is called. Preparing the files includes clearing the members of any existing data if requested on the API. The files prepared by this API can be used on several calls of the DSPJOBLOG command. However, the members are not cleared between calls.
- Each time the DSPJOBLOG command is called, the output is added to whatever is already in the members. Additionally, at end of job, the final records for the job log are added to the members.
- If the members need to be cleared between calls of the DSPJOBLOG command or before the job ends, call the API again or use the CLRPFM (Clear Physical File Member) command.

This API does not override the LOG job attribute for the job or the LOG attribute on the SIGNOFF CL command. The LOG job attribute controls whether any job log is to be produced and controls whether second level text is to be logged.

## Authorities and Locks

Authorization to the specified output file(s) is checked when this API is used. If the specified output file(s) does not exist, this API creates them. Additionally, if the output file member(s) does not exist, this API adds a new member by the specified name to the output file(s). The authorization checking is performed using the user profile the API was called under. If an output file needs to be created, the user profile the API was called under becomes the owner of the output file. Public authority to files created by this API is controlled by the library containing those files. Public authority for objects created into a library is controlled by the CRTAUT parameter on the CRTLIB or CHGLIB command.

### Output file library authority:

- \*EXECUTE required in all cases.
- \*ADD required when the output file or member does not exist.

### Output file authority:

- \*OBJOPR, \*OBJMGT, \*ADD required in all cases.
- \*DLT required to clear the member of existing records.

## Required Parameter Group

### Output file names structure

INPUT; CHAR(65)

Provides information about the primary and secondary files and members.

### Output file names structure format

INPUT; CHAR(8)

The format of the output file names structure parameter. This must be set to CTLJ0100. See [CTLJ0100 Output File Names Structure](#) for details about this format.

### Message filter array

INPUT; CHAR(\*)

An array of zero or more elements. Each element in the array defines a test applied against each message before it is written to the output file. The tests are performed in the order they appear in the array. A message is rejected and not written to the output file if any one test defined in the array is true for the message. A message is selected and written to the output file if all tests defined in the array are false for the message.

Each array element specifies a message severity, message type, and message ID. If a message has a message severity less than or equal to the specified severity, has the specified type, and has a message ID equal to the specified ID, the test is true and the message is not written to the output file. Processing stops on the rejected message and the next message is processed.

If a message meets one or two of the criteria but not all three, the test is false and the test defined by the next array element is applied. If all array elements have been applied to the message and none have been found true, the message is written to the output file.

### Number of message filter elements

INPUT; BINARY(4)

The number of elements in the Message filter array parameter. The value must be greater than or equal to 0 but less than or equal to 255. If no message filtering is to be done during job log production, this parameter must be set to 0.

### Qualified error message queue name

INPUT; CHAR(20)

The qualified name of a message queue to which a message should be sent to record any errors detected during the production of the job log to an output file. The first 10 characters of this parameter contain the message queue name and the second 10 characters contain the library name. A library name must be specified. The special values \*CURLIB and \*LIBL cannot be used. QTEMP cannot be specified as the library name. The message queue name specified cannot be QHST or QSYSMSG.

The following special value can be used for the message queue name of this parameter:

- \*NONE No messages are sent. If errors occur during job log production to an output file, there is no record of what the error was or the action taken in response to the error.
- \*SYSOPR Send the messages to the system operator's message queue, QSYSOPR.

If a qualified message queue name is provided and that message queue does not exist or is unusable at job log production time, any messages are sent to \*SYSOPR.

## Error code

I/O; CHAR(\*)

The structure in which to return error information. For the format of the structure, see [Error Code Parameter](#).

## CTLJ0100 Output File Names Structure

The following table shows the layout of the output file names structure for format CTLJ0100.

Offset		Type	Field
Dec	Hex		
0	0	BIN(4)	Output file names structure length
4	4	CHAR(20)	Qualified primary file name
24	18	CHAR(10)	Primary file member name
34	22	CHAR(20)	Qualified secondary file name
54	36	CHAR(10)	Secondary file member name
64	40	CHAR(1)	Member replace option

## Message Filter Array Elements

The following table shows the structure of an array element in the message filter array parameter. This structure is repeated for each element in the array.

Offset		Type	Field
Dec	Hex		
0	0	CHAR(4)	Reserved
4	4	BINARY(4)	Message severity
8	8	CHAR(10)	Message Type
18	12	CHAR(7)	Message ID
25	19	CHAR(3)	Reserved

## Field Descriptions

**Member replace option.** Specifies if an existing member of either the primary or secondary file is to be cleared. If the member is to be cleared, it is cleared when this API is called. If a member contains existing records and is not cleared, at job log production time, the new records are added to the member and the existing records are left unchanged. Use one of the following special values:

- 0 Do not clear the member.
- 1 Clear the member of existing records.

**Message ID.** The message ID to test each message against. The following special values can be used in place of a message ID:

- \*ANY* Any message ID is considered equal. Message type and message severity determine if the message is selected or rejected. If message type is specified as *\*ANY*, message severity as 99, and message ID as *\*ANY*, all messages are rejected. There is no job log output to the output file(s).
- \*IMMED* Identifies immediate messages with no message ID.

A generic message ID can be used in place of a specific message ID. A generic message ID is an ID in which the last 2 or 4 characters are 0. For example, CPF2400 and CPF0000 are generic IDs. A generic ID represents a range of IDs rather than a single ID. A generic ID of the form XXXnn00 represents all IDs from XXXnn00 through XXXnnFF. A generic ID of the form XXX0000 represents all IDs from XXX0000 through XXXFFFF.

When a generic ID is specified, any message ID within the range of the generic ID is considered to match the generic ID. The message severity to test each message against. If severity is not important in rejecting a message use 99 as the value for this field. When 99 is used, each message's severity is less than or equal and, as a result, message type and message ID will determine if a message is selected or rejected.

**Message type.** The message type to test each message for. Use one of the following special values:

- \*ANY* Any message type. Use this when message type is not important in rejecting a message. Any message type is considered equal to this and, as a result, message severity and message ID determines if the message is selected or rejected.
- \*CMD* Commands that are logged from the running of a CL program.
- \*COMP* Completion message type.
- \*COPY* Sender's copy message type. If a sender's copy message is not written to the output file, its reply message is also not written.
- \*DIAG* Diagnostic message type.
- \*ESCAPE* Escape message type. The *\*ESCAPE* message type includes the function check message.
- \*EXCP* Exception message type. Includes both *\*ESCAPE* and *\*NOTIFY* message types.
- \*INFO* Information message type.
- \*INQ* Inquiry message type. If an inquiry message is not written to the output file, its reply message is also not written.
- \*NOTIFY* Notify message type. If a notify message is not written to the output file, its reply message is also not written.
- \*RQS* Request message type.
- \*RPY* Reply message type. If a reply message is not written to the output file, its inquiry, notify, or sender's copy message is also not written.

**Output file names structure length.** The length of this structure. This must be set to a value of 65.

**Primary file member name.** The name of the member within the primary file used for the primary job log information.



If a member of the specified name does not exist, a member by that name is added to the primary file by this API.

If a member by the specified name does exist, this API will clear the member if the Member replace option field of this structure specifies replace. Otherwise, the member is not cleared. At job log production time, new records are added to the member and any existing records are left unchanged.

The following special value can be used in this field:

*\*FIRST* The first member in the file receives the output. If a member does not exist, one is created with a name the same as the file name.

**Qualified primary file name.** The qualified name of the primary output file. This field consists of two parts. The first 10 positions contain the file name and the second 10 positions contain the library name. A specific library name can be provided or the special values *\*CURLIB* and *\*LIBL* can be used. *QTEMP* cannot be used as a library name.

If an output file by the specified name does not exist when this API is called, one is created by that name in the specified library. If the library was specified as *\*LIBL*, the file is created in the current library. If *\*LIBL* or *\*CURLIB* was specified and there is no current library, the file is created in *QGPL*. The file is created with the attributes *MAXRCDS* and *MAXMBRS* set to *\*NOMAX*.

The following special value can be used in this field:

*\*PRINT* Resets the job so the job log is written to a spooled file rather than an output file. When this special value is used the remaining parameters on this API are ignored.

The API does not allow the primary file specification to be overridden by the use of the Override Database File (*OVRDBF*) command. The primary file also cannot be overridden at job log production time. The file object determined to be the primary file used by this API is the one used at job log production time. If the file does not exist at job log production time, the job log is redirected to a spooled file.

The primary file must be a local physical file; it cannot be a *DDM* file.

**Qualified secondary file name.** The name of the file into which the secondary job log information is written. The parameter consists of two parts. The first 10 positions to contain the file name and the second 10 positions contain the library name. A library name can be provided or the special values *\*CURLIB* and *\*LIBL* can be used. *QTEMP* cannot be used for the library name.

If an output file by the specified name does not exist when the API is called, one is created by that name in the specified library. If the library was specified as *\*LIBL*, the file is created in the current library. If *\*LIBL* or *\*CURLIB* was specified and there is no current library, the file is created in *QGPL*. The file attributes *MAXRCDS* and *MAXMBRS* are set to *\*NOMAX*.

The following special value can be used in this field:

*\*NONE* No secondary file is provided. This special value prevents the generation of the secondary job log information. When this special value is used, the secondary library and member names are ignored.

The API does not allow the secondary file specification to be overridden by the use of the *OVRDBF* command. The secondary file also cannot be overridden at job log production time. The file object determined to be the secondary file by this API is used at job log production time. If the file does not exist at job log production time, the job log automatically redirects to a spooled file.

The secondary file must be a local physical file; it cannot be a *DDM* file.

**Reserved.** This field must be set to binary zeros.

**Secondary file member name.** The name of the member within the secondary file used for the secondary job log information.

If a member of the specified name does not exist, a member by that name is added to the secondary file.

If a member by the specified name does exist, the member is cleared if the Member replace option field of this structure specifies replace. Otherwise, the member is not cleared. At job log production time, new records are added to the member and any existing records are left unchanged.

The following special value can be used in this field:


*\*FIRST* The first member in the file receives the output. If a member does not exist, one is created with the same name as the file name.

## CCSID Considerations

CCSID conversion is allowed on any field in the primary file with the exception of the field QMHMDT, which contains the message data or immediate message text. The QMHMDT field is defined with a CCSID of 65535 so conversion does not occur on this field when the record is written. CCSID processing is done on the message data when the message is sent. The CCSID that the QMHMDT field data is in is stored in the record with the message data. The CCSID is placed in field QMHCID.

CCSID conversion is allowed for any field in the secondary file except for the field QMHLIN which contains a line of message text. This field is defined with CCSID 65535. The message text has completed CCSID processing before the record is written to the file. The CCSID the message text is in is stored in field QMHSID.

The model files QAMHJLPR and QAMHJLSC specify no CCSID definitions other than the two fields mentioned previously. CCSID processing for the remaining fields is allowed to default.

Detailed information about the fields mentioned in this discussion is provided in the [CL Programming](#)  book.

## Error Considerations

During job log production to an output file, different types of errors can be encountered. Some errors allow the job log to be written to the output file but the information in a record can be incomplete. Other errors can prevent the job log from being written to the output file. For example, when the output file has been deleted between the time this API has been run and job log production starts. Other errors, such as when a member becomes full, allow a certain number of records to be written but not all. Whenever an error is encountered, an informational message is written to the user or workstation queue specified by the API parameter. If the API parameter specifies \*NONE, these messages are not written.

If the error is such that the data can still be written to the output file, even though it can be incomplete, processing continues and at the end of job log production there are output file(s) and member(s) which contains the information that could be written. For example, if the message data for a message needs to be truncated to 3000 characters, production of the job log to the output file continues but a message is sent to record the fact that the message data was truncated.

If the error is such that no data can be written, the job log automatically redirects to a spooled file. This occurs if the objects, determined by the API to be the output files, are deleted, damaged, or otherwise unusable at the time job log production started. If any such error is encountered on the primary or secondary output file, this redirection to a spooled file occurs. A message is sent to the specified message queue to record that the job log has been redirected to a spooled file.

If only certain records are written to the output file, the remaining messages are redirected to a spooled file. A message is sent to notify of this.

## Job Message Queue Full Considerations

If the print-wrap option is used to control the action to take when a job message queue becomes full, consider the following when using this API to direct the output to an output file.

Each time the job message queue becomes full, records are written to the output file for those messages that are being removed from the job message queue. All new records are added to the member. The Member replace option specified on this API has an effect only when this API is preparing the output file for use. It has no effect when the records are written to the output file. For the print-wrap option, the member contains the complete set of messages for the job rather than a partial set.

For long running jobs, it is possible that the job message queue can become full and the output file member also becomes full. If the member becomes full, output is automatically redirected to a spooled file until the API is run another time to specify a new member to use.

Changing the action from print-wrap to wrap will stop the production of records to the file(s). When the job ends or if the action is changed back to print-wrap, records are again written to the file(s).

## Usage Notes

This API can be called from the initial thread of a multithreaded job to control the job log output for all threads. It should not be called from a secondary thread.

## Error Messages

Message ID	Error Message Text
CPF24D0 E	QTEMP is not a valid library to use for the file &1.
CPF24D1 E	Member replace option &1 is not valid.
CPF24D2 E	Number of elements &1 in message filter array is not valid.
CPF24D3 E	Message queue &1 not valid for use with the QMHCTLJL API.
CPF24D4 E	Message type &1 specified in filter element &2 is not valid.
CPF24D5 E	Message severity &1 specified in filter element &2 is not valid.
CPF24D6 E	Reserved field in filter element &1 does not contain a null value.
CPF24D7 E	File &1 in library &2 cannot be used for job log production.

CPF24D8 E	DDM file &1 in library &2 cannot be used for job log production.
CPF24D9 E	Error message queue library name &1 is not valid.
CPF24DA E	Failure while preparing for job log production to a physical file.
CPF3CF1 E	Error code parameter not valid.
CPF3C1D E	Length specified in parameter &1 not valid.
CPF3C21 E	Format name &1 is not valid.
CPF3C90 E	Literal value cannot be changed.
CPF9822 E	Not authorized to file &1 in library &2.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

## Job Log Production Messages

Message ID	Message Text
CPD241A E	Output file &1 member &3 moved or renamed.
CPD241B E	Error occurred on file &1 member &3 in &2.
CPD241C E	Job log file &1 member &3 in &2 has been deleted.
CPD241D E	Job log file &1 member &3 in &2 is not available.
CPD241E E	No records written to job log file &1 member &3 in &2.
CPD241F E	Not all records written to job log file &1 member &3 in &2.
CPD242B E	Message data truncated for message key &7.

---

API introduced: V3R1

---

[Top](#) | [Message Handling APIs](#) | [APIs by category](#)

# List Job Log Messages (QMHLJOB) API

## Required Parameter Group:

1	Qualified user space name	Input	Char(20)
2	Format name	Input	Char(8)
3	Message selection information	Input	Char(*)
4	Size of message selection information	Input	Binary(4)
5	Format of message selection information	Input	Char(8)
6	Error code	I/O	Char(*)

Default Public Authority: \*USE

Threadsafe: No

The List Job Log Messages (QMHLJOB) API lists messages sent to the job message queue of a job. This API gets the requested message information and returns it in a user space in the format specified in the parameter list. The following discusses how the list is sorted for nonbatch jobs and for batch jobs.

When the job whose messages are being listed is not a batch job, the returned messages are sorted by their sending date and time unless the message being listed is a reply message to an inquiry, a sender's copy, or a notify message. If it is a reply message, it is listed immediately following the inquiry, sender's copy, or notify message with which it is associated.

If the last message listed is an inquiry, a sender's copy, or a notify message, the user of the API must call the API again using parameters that would include listing the next later message after the inquiry, sender's copy, or notify message in order to obtain an available reply message.

When the job whose messages are listed is a batch job, the messages are grouped into two categories:

- Request messages that have been or are being processed, and the other messages that occurred during the processing of those requests.
- Request messages that are yet to be processed, and any diagnostic messages associated with these request messages.

The API treats unprocessed request messages as if they had a sending time later than all the request messages and their associated messages that have been or are being processed. The following two examples describe the sorting further.

For example, if the call to this API specifies to list the messages for a batch job from oldest to newest, the list consists of all requests and their associated messages that have been or are being processed. They are sorted as described above for a job that is not a batch job. They are followed by any request messages and any associated diagnostic messages that have not yet been processed (in the order that they will be processed).

As an opposite example, if the call to this API specifies to list the messages for a batch job from newest to oldest, the list consists of the request messages that remain to be processed. They are in the opposite order that they are processed. They are followed by the request messages that have been or are being processed and their associated messages. These are sorted backward through time as described above for nonbatch jobs.

The generated list replaces any existing information in the user space.

If the user space is not large enough to contain the data to be returned, the user space is increased to the maximum user space size allowed (16MB) or the maximum amount of storage allowed to the user of the API. If this is not large enough to contain the data to be returned, only the number of complete messages that fit in the user space are returned. The information status field in the generic header is set to P (partial but accurate). The user can then resubmit the request from the last message returned to obtain the additional messages. The key of the last message listed for each message queue is provided in the ending message key field in the header portion of the user space.

The maximum messages requested field and the number of fields to return field for each listed message increase the system resources required to create the list. Users of this API should use caution when specifying parameters that list many messages or request many identified fields to be returned for each listed message.

## Authorities and Locks

*User space*

\*CHANGE

*User space library*

\*EXECUTE

*User space lock*

\*EXCLRD

*Job authority*

- \*JOBCTL if the job for which messages are being retrieved has a user profile different from that of the job that calls the QMHLJOB API.
- \*ALLOBJ and \*JOBCTL if the job for which messages are being retrieved has a user class of \*SECOFR.

For additional information on job authorities, see [Basic system security and planning](#).

## Required Parameter Group

### Qualified user space name

INPUT;CHAR(20)

The user space that receives the generated list, and the library in which it is located. The first 10 characters contain the user space name, and the second 10 characters contain the user space library. You can use these special values for the library name:

\**CURLIB* The job's current library

\**LIBL* The library list

### Format name

INPUT; CHAR(8)

The format of the returned message information. You must use this format:

*LJOB0100* Basic message information with identified return fields. This format is described in [LJOB0100 Format](#).

### Message selection information

INPUT; CHAR(\*)

The information that determines the job message queue and messages to be listed. The format of this information is described in [JSLT0100 Format](#) and in [JSLT0200 Format](#).

### Size of message selection information

INPUT; BINARY(4)

The size in bytes of the message selection information parameter.

### Format of message selection information

INPUT; CHAR(8)

The format of the message selection information parameter. You must use one of these formats:

*JSLT0100* The specific information identifying the messages to be listed. This format is described in [JSLT0100 Format](#). Message text and data are returned in the CCSID of your job.

*JSLT0200* The specific information identifying the messages to be listed. This format is described in [JSLT0200 Format](#). This format is the same as JSLT0100 with the exception that you can specify the CCSID you want your message text and data returned in.

### Error code

I/O; CHAR(\*)

The structure in which to return error information. For the format of the structure, see [Error Code Parameter](#).

## Format of Generated Lists

The user space created consists of:

- A user area
- A generic header
- An input parameter section
- A header section
- LJOB0100 format

For details about the user area and generic header, see the [User Space Format for List APIs](#). For details about the remaining items, see the following sections. For descriptions of each field in the list returned, see [Field Descriptions](#).

## Input Parameter Section

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	User space name specified
10	A	CHAR(10)	User space library specified
20	14	CHAR(8)	Format name specified
28	1C	CHAR(8)	Format of message selection information specified
36	24	BINARY(4)	Size of message selection information specified
40	28	BINARY(4)	Maximum messages requested specified
44	2C	CHAR(10)	List direction specified
54	36	CHAR(10)	Job name specified
64	40	CHAR(10)	User profile specified
74	4A	CHAR(6)	Job number specified
80	50	CHAR(16)	Internal job identifier specified
96	60	CHAR(4)	Starting message key specified
100	64	BINARY(4)	Maximum message length specified
104	68	BINARY(4)	Maximum message help length specified
108	6C	BINARY(4)	Offset to identifiers of fields to return specified
112	70	BINARY(4)	Number of fields to return specified
116	74	BINARY(4)	Offset to call message queue specified
120	78	BINARY(4)	Length of call message queue specified
124	7C	BINARY(4)	Coded character set identifier (CCSID) specified
128	80	CHAR(*)	Reserved
The offsets to these fields are specified in the previous offset variables.		ARRAY(*) of BINARY(4)	Identifiers of fields to return specified
		CHAR(*)	Call message queue specified

## Header Section

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	User space name used
10	A	CHAR(10)	User space library used
20	14	CHAR(4)	Starting message key used
24	18	CHAR(4)	Ending message key



28	1C	CHAR(10)	Job name used
38	26	CHAR(10)	User profile used
48	30	CHAR(6)	Job number used
54	36	CHAR(2)	Reserved
56	38	BINARY(4)	Coded character set identifier (CCSID) used

## LJOB0100 Format

The following table shows the information returned in the list data section of the user space for the LJOB0100 format. The offsets listed are from the beginning of the user space. For a detailed description of each field, see [Field Descriptions](#).

The structure defined by this format is repeated for each message entry returned.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Offset to the next entry
4	4	BINARY(4)	Offset to fields returned
8	8	BINARY(4)	Number of fields returned
12	C	BINARY(4)	Message severity
16	10	CHAR(7)	Message identifier
23	17	CHAR(2)	Message type
25	19	CHAR(4)	Message key
29	1D	CHAR(10)	Message file name
39	27	CHAR(10)	Message file library specified at send time
49	31	CHAR(7)	Date sent
56	38	CHAR(6)	Time sent
» 62	3E	CHAR(6)	Microseconds «
» 68	44 «	CHAR(*)	Reserved
These fields repeat for each identifier field specified.		BINARY(4)	Offset to the next field information returned
		BINARY(4)	Length of field information returned
		BINARY(4)	Identifier field
		CHAR(1)	Type of data
		CHAR(1)	Status of data
		CHAR(14)	Reserved
		BINARY(4)	Length of data
		CHAR(*)	Data
		CHAR(*)	Reserved

## JSLT0100 Format

The organization of the JSLT0100 format of the message selection information parameter follows. For a detailed description of each field, see [Field Descriptions](#).

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Maximum messages requested
4	4	CHAR(10)	List direction
14	E	CHAR(10)	Qualified job name
24	18	CHAR(10)	Qualified user name
34	22	CHAR(6)	Qualified job number
40	28	CHAR(16)	Internal job identifier
56	38	CHAR(4)	Starting message key
60	3C	BINARY(4)	Maximum message length
64	40	BINARY(4)	Maximum message help length
68	44	BINARY(4)	Offset to identifiers of fields to return
72	48	BINARY(4)	Number of fields to return
76	4C	BINARY(4)	Offset to call message queue name
80	50	BINARY(4)	Length of call message queue name
The offsets to these fields are specified in the previous offset variables.		ARRAY(*) of BINARY(4)	Identifiers of fields to return
		CHAR(*)	Call message queue name

## JSLT0200 Format

The organization of the JSLT0200 format of the message selection information parameter follows. For a detailed description of each field, see [Field Descriptions](#).

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Maximum messages requested
4	4	CHAR(10)	List direction
14	E	CHAR(10)	Qualified job name
24	18	CHAR(10)	Qualified user name
34	22	CHAR(6)	Qualified job number
40	28	CHAR(16)	Internal job identifier
56	38	CHAR(4)	Starting message key
60	3C	BINARY(4)	Maximum message length
64	40	BINARY(4)	Maximum message help length
68	44	BINARY(4)	Offset to identifiers of fields to return
72	48	BINARY(4)	Number of fields to return

76	4C	BINARY(4)	Offset to call message queue name
80	50	BINARY(4)	Length of call message queue name
84	54	BINARY(4)	Coded character set identifier (CCSID) to return text and data in
88	58	BINARY(4)	Reserved (must be set to 0)
The offsets to these fields are specified in the previous offset variables.		ARRAY(*) of BINARY(4)	Identifiers of fields to return
		CHAR(*)	Call message queue name

## Valid Field Identifiers

The following table contains a list of the valid identifiers that can be specified in the message selection information parameter. For a detailed description of each field, see [Field Descriptions](#).


Identifier	Type	Description
0101	CHAR(9)	Alert option
0201	CHAR(*)	Replacement data or impromptu message text
0301	CHAR(*)	Message
0302	CHAR(*)	Message with replacement data
0401	CHAR(*)	Message help
0402	CHAR(*)	Message help with replacement data
0403	CHAR(*)	Message help with formatting characters
0404	CHAR(*)	Message help with replacement data and formatting characters
0501	CHAR(*)	Default reply
0601	CHAR(26)	Qualified sender job name
0602	CHAR(1)	Sender type
0603	CHAR(*)	Sending program name
0604	CHAR(10)	Sending module name
0605	CHAR(*)	Sending procedure name
0606	BINARY(4) followed by ARRAY(*) of CHAR(10)	Number of sending statement numbers or instruction numbers available followed by an array of the sending statement numbers or instruction numbers
0702	CHAR(1)	Receiving type
0703	CHAR(10)	Receiving program name
0704	CHAR(10)	Receiving module name
0705	CHAR(*)	Receiving procedure name
0706	BINARY(4) followed by ARRAY(*) of CHAR(10)	Number of receiving statement numbers or instruction numbers available followed by an array of the receiving statement numbers or instruction numbers

0801	CHAR(10)	Message file library used
0901	CHAR(30)	Problem identification
1001	CHAR(1)	Reply status
1101	CHAR(1)	Request status
1201	BINARY(4)	Request level
1301	BINARY(4)	Coded character set identifier (CCSID) for text
1302	BINARY(4)	CCSID conversion status indicator for text
1303	BINARY(4)	Coded character set identifier (CCSID) for data
1304	BINARY(4)	CCSID conversion status indicator for data

## Field Descriptions

**Alert option.** Whether and when an SNA alert is created and sent for the message. The value is one of the following:

- \**DEFER* An alert is sent after local problem analysis.
- \**IMMED* An alert is sent immediately when the message is sent to a message queue that has the allow alerts attribute set to \*YES.
- \**NO* No alert is sent.
- \**UNATTEND* An alert is sent immediately when the system is running in unattended mode (when the value of the alert status network attribute, ALRSTS, is \*UNATTEND).

For more information about alerts, see the [Alerts Support](#)  book.

This field is set to blanks if no alert option was specified when the message was sent.

**Call message queue name.** The name of the call message queue from which the messages are listed. You must use one of these values:

- \* Messages from every call stack entry of the job are listed. If the length of call message queue name field is greater than 1, the value must be left-justified and padded on the right with blanks.
- \**EXT* Only messages sent to the external message queue (\*EXT) of the job are to be listed. If the length of the call message queue name field is greater than 4. The value must be left-justified in the field and padded on the right with blanks.

**Call message queue specified.** The call message queue name as specified on the call to the API.

**CCSID conversion status indicator for data.** The following values may be returned:

- 0 No conversion was needed because the CCSID of the data matched the CCSID you wanted the data converted to.
- 1 No conversion occurred because either the data was 65535 or the CCSID you wanted the data converted to was 65535.

- 2 No conversion occurred because you did not ask for any message data to be returned or the data did not contain any \*CCHAR type data.
  - 3 The data was converted to the CCSID specified using the best fit conversion tables.
  - 4 A conversion error occurred using the best fit conversion tables so a default conversion was attempted. This completed without error.
- 1 An error occurred on both the best fit and default conversions. The text was not converted.

**CCSID conversion status indicator for text.** The following values may be returned:

- 0 No conversion was needed because the CCSID of the text matched the CCSID you wanted the text converted to.
  - 1 No conversion occurred because either the text was 65535 or the CCSID you wanted the text converted to was 65535.
  - 2 No conversion occurred because you did not ask for any text to be returned.
  - 3 The text was converted to the CCSID specified using the best fit conversion tables.
  - 4 A conversion error occurred using the best fit conversion tables so a default conversion was attempted. This completed without error.
- 1 An error occurred on both the best fit and default conversions. The data was not converted.

**Coded character set identifier (CCSID) for data.** The coded character set identifier that the replacement data is returned in. This only applies to the part of the replacement data that corresponds to a convertible character data type (\*CCHAR). All other replacement data will not be converted before it is returned and can be considered to have a CCSID of 65535. If a conversion error occurs or if the CCSID you requested the data to be converted to is 65535, the CCSID of the data is returned. If there is no \*CCHAR replacement data, 65535 is returned. Otherwise the CCSID you wanted the data converted to is returned.

For more information about message handler and its use of CCSIDs, see [CCSIDs: Message Support](#) in the Globalization topic. For more information about the \*CCHAR field type, see the [Add Message Description](#) (ADDMSGD) command.

**Coded character set identifier (CCSID) for text.** The coded character set identifier that the message text is returned in. If a conversion error occurs or if the CCSID you requested the message text to be converted to is 65535, the CCSID that the message text is stored in is returned. Otherwise the CCSID you wanted your message text converted to is returned. If you do not want the text converted before it is returned to you but you do want to know the CCSID that the message text is stored in, specify 65535 on the coded character set identifier to return text and data in parameter. The CCSID that the message text is stored in is returned in the coded character set identifier for text output field.

This applies to the following fields only:

- Message
- Message with replacement data
- Message help
- Message help with replacement data
- Message help with replacement data and formatting characters
- Message help with formatting characters

**Note:** This CCSID value does not apply to the replacement data that has been substituted into the text. See the coded character set identifier for data for this information.

**Coded character set identifier to return text and data in.** The CCSID that the text and data are converted to before they are returned. The following values are allowed:

*0* The text and data are converted to the CCSID of the job before being returned. This is the default value used when the JSLT0100 format is specified.

If the job is 65535 and the text or data is something other than EBCDIC single byte or EBCDIC mixed, the text and data are converted to the default job CCSID.

*65535* The text and data are not converted before being returned.

*CCSID* Specify a valid CCSID you want your text and data converted to before being returned. The CCSID must be between 1 and 65535. The CCSID is validated by this API. Only CCSIDs that a job can be changed to are accepted.

For a list of valid CCSIDs, see [CCSIDs: Message Support](#).

**Coded character set identifier specified.** The CCSID that was specified that the text and data are converted to before they are returned.

**Coded character set identifier used.** The CCSID that was used that the text and data are converted to before they are returned.

**Data.** The data returned for the specified identifier field.

**Date sent.** The date on which the message was sent, in CYYMMDD (century, year, month, and day) format.

**Default reply.** The text of the default reply when a stored message is being listed and a default reply exists. If this is not an inquiry message, or no default reply exists, this field is not used and the length of data field is 0.

**Ending message key.** The message key of the last message actually listed by the API. If no message is listed, the value returned is the same as the value specified in the starting message key specified field.

**Format of message selection information specified.** The format name of the message selection information parameter as specified on the call to the API.

**Format name specified.** The format name as specified on the call to the API.

**Identifier field.** The field ID actually returned in the LJOB0100 format. See [Valid Field Identifiers](#) for the list of valid field identifiers.

**Identifiers of fields to return.** The list of the field identifiers returned in the LJOB0100 format. For a list of the valid field identifiers, see [Valid Field Identifiers](#).

**Identifiers of fields to return specified.** The list of identifiers of fields to return as specified on the call to the API.

**Internal job identifier.** The internal name for the job. The List Job (QUSLJOB) API creates this identifier. If you do not specify \*INT for the qualified job name parameter, this parameter must contain blanks. If your application already has this information available from QUSLJOB, the QMHLJOB API can locate the job more quickly with this information than with a job name. However, calling QUSLJOB solely to obtain this parameter for use by QMHLJOB would result in poorer performance than using a job name in

calling QMHLJOBL.

**Internal job identifier specified.** The internal job identifier as specified on the call to the API.

**Job name specified.** The job name of the job that lists the messages as specified on the call to the API.

**Job name used.** The actual job name of the job that was used to list the messages.

**Job number specified.** The job number of the job that lists the messages as specified on the call to the API.

**Job number used.** The actual job number of the job that was used to list the messages.

**Length of data.** The length of the data returned for the data field, in bytes.

**Length of field information returned.** The total length of information returned for this field, in bytes.

**Length of call message queue name.** The length of the call message queue name field, in bytes. The maximum length that can be specified is 256. The minimum length is 1.

**Length of call message queue specified.** The length of the call message queue specified field, in bytes.

**List direction.** The direction to list messages. You must use one of these directions:

\**NEXT* Returns messages that are newer than the message specified by the starting message key field.

\**PRV* Returns messages that are older than the messages specified by the starting message key field.

When a batch job is being listed, request messages that have not yet been processed or received are considered to have a sending date and time later than all other messages on the job log. This is also true for any diagnostic messages associated with those request messages.

**List direction specified.** The direction to list messages as specified on the call to the API.

**Maximum message help length.** The maximum number of characters of text that this API returns for field identifiers 0401, 0402, 0403, and 0404. (See [Valid Field Identifiers](#).)

Specify a value to limit the number of characters returned for field identifiers 0401, 0402, 0403, and 0404. This value can be no smaller than 4. The maximum allowed value is 32765. To specify that the maximum length be used, use the special value of -1. This value is not checked if field identifiers 0401, 0402, 0403 or 0404 are not specified.

**Maximum message help length specified.** The maximum number of characters to return for field identifiers 0401, 0402, 0403 and 0404 as specified on the call to the API.

**Maximum message length.** The maximum number of characters of text that this API returns for field identifiers 0301 and 0302.

Specify a value to limit the number of characters returned for field identifiers 0301 and 0302. (See [Valid Field Identifiers](#).) This value can be no smaller than 4. The maximum allowed value is 32765. To specify that the maximum length be used, use the special value -1. This value is not checked if field identifiers 0301 or 0302 are not specified.

**Maximum message length specified.** The maximum number of characters to return for field identifiers 0301 and 0302 as specified on the call to the API.

**Maximum messages requested.** The maximum number of messages to be returned.

If fewer messages than the number requested exist on the job message queue, only the number of messages

that exist are returned. No error is signaled, and the information status field in the generic header would be marked as C for complete and accurate.

To list all messages in the job log in the specified list direction from the starting message key, use the special value of -1.

**Maximum messages requested specified.** The number of messages requested to be listed as specified on the call to the API.

**Message.** The text of a predefined message without replacement data substitution. If an impromptu message is listed, this field contains the impromptu message text.

**Message file library specified at send time.** The name of the library containing the message file as specified when the message was sent. If \*CURLIB or \*LIBL were specified for the library when the message was sent, that value is returned as the library here. For the actual library used when the message is sent, see the message file library used field.

**Message file library used.** The actual name of the library that contains the message file used to retrieve the message information. If an immediate message is listed, this field is set to blanks.

**Message file name.** The name of the message file containing the message listed.

**Message help.** The message help for the message listed without formatting characters and without replacement of data. If an impromptu message is listed, this field contains the impromptu message text.

**Message help with formatting characters.** The message help for the message listed, including formatting characters.

Three format control characters can be returned within the message. In the Add Message Description (ADDMSGD) command, they are defined to have these meanings:

- &N* Forces the text to a new line (column 2). If the text is longer than one line, the next lines are indented to column 4 until the end of text or another format control character is found.
- &P* Forces the text to a new line indented to column 6. If the text is longer than one line, the next lines start in column 4 until the end of text or another format control character is found.
- &B* Forces the text to a new line, starting in column 4. If the text is longer than one line, the next lines are indented to column 6 until the end of text or another format control character is found.

If an impromptu message is listed, this field contains the immediate message text.

**Message help with replacement data.** The message help for the message listed, including the replacement data. If an impromptu message is listed, this field contains the impromptu message text.

**Message help with replacement data and formatting characters.** The message help for the message listed, including the replacement data and the formatting characters. See the description of the message help with formatting characters field for an explanation of formatting characters. If an impromptu message is listed, this field contains the impromptu message text.

**Message identifier.** The identifying code of the message listed. If an impromptu message is listed, this field is set to blanks.

**Message key.** The message reference key of the message listed.

**Message severity.** The severity of the message listed. Possible values are 0 through 99.

**Message type.** The type of message listed. The possible values and their meanings follow:



<b>Value</b>	<b>Message Type</b>
01	Completion
02	Diagnostic
04	Informational
05	Inquiry
06	Sender's copy
08	Request
10	Request with prompting
14	Notify, exception already handled when API is called
15	Escape, exception already handled when API is called
16	Notify, exception not handled when API is called
17	Escape, exception not handled when API is called
21	Reply, not checked for validity
22	Reply, checked for validity
23	Reply, message default used
24	Reply, system default used
25	Reply, from system reply list

**Message with replacement data.** The text of a predefined message with the replacement data included. If an impromptu message is listed, this field contains the impromptu message text.

» **Microseconds.** The microseconds part of the time sent.«

**Number of fields returned.** The number of identifier fields returned to the application.

**Number of fields to return.** The number of fields to return in the LJOB0100 format (the number of entries in the identifiers of fields to return array).

**Number of fields to return specified.** The number of identifier fields to return as specified on the call to the API.

**Number of receiving statement numbers or instruction numbers available followed by an array of the receiving statement numbers.** The number of statement numbers or instruction numbers available for the receiving program or procedure.

For OPM programs and nonoptimized procedures, this count is 1.

For optimized procedures, this count can be greater than 1. In this case, each statement number represents a potential point at which the message could have been received. If the mapping table information has been removed from the program, this field returns a count of 0 and no statement numbers are available. The array of receiving statement numbers or instruction numbers immediately follows this field in the returned data.

**Number of sending statement numbers or instruction numbers available followed by an array of the**

**sending statement numbers or instruction numbers.** The number of statement numbers or instruction numbers available for the sending program or procedure.

For OPM programs and nonoptimized procedures, this count is 1.

For optimized procedures, this count can be greater than 1. In this case, each statement number represents a potential point at which the message could have been sent. If the mapping table information has been removed from the program, this field returns a count of 0, and no statement numbers are available. The array of sending statement numbers or instruction numbers immediately follows this field in the returned data.

**Offset of fields to return specified.** The offset, in bytes, from the beginning of the user space to the beginning of the identifiers of fields to return specified field.

**Offset to call message queue name.** The offset, in bytes, from the beginning of the message selection information parameter to the beginning of the call message queue name field.

**Offset to call message queue specified.** The offset, in bytes, from the beginning of the user space to the beginning of the call message queue specified field.

**Offset to fields returned.** The offset, in bytes, from the beginning of the user space to the beginning of the first repeating identified field of the LJOB0100 format.

**Offset to identifiers of fields to return.** The offset, in bytes, from the beginning of the message selection information parameter to the beginning of the identifiers of fields to return array.

**Offset to the next entry.** The offset, in bytes, from the beginning of the user space to the beginning of the next message entry.

**Offset, in bytes, to the next field information returned.** The offset from the beginning of the user space to the beginning of the next repeating identified field of the LJOB0100 format.

**Problem identification.** This field can be specified for the QMHLJOB L API, but it never returns any data and the length of data field is 0.

**Qualified job name.** The specific job name of the job whose messages are to be listed, or one of the following special values:

- \* The job that this program is running in. If this special value is used, the Qualified User Name and Qualified Job Number parameters must be blank.
- \**INT* The internal job identifier locates the job. If this special value is used, the Qualified User Name and Qualified Job Number parameters must be blank.

**Qualified user name.** A specific user profile name of the job whose messages are to be listed, or blanks when the Qualified Job Name parameter is the special value of \* or \*INT.

**Qualified job number.** A specific job number of the job whose messages are to be listed, or blanks when the Qualified Job Name parameter is the special value of \* or \*INT.

**Qualified sender job name.** This field can be specified for the QMHLJOB L API, but it never returns any data and the length of data field is 0.

**Receiving module name.** The name of the module that contains the procedure where the message was sent. If the message was not sent to a procedure within a ILE program, this field is not set and the length of data field is 0.

**Receiving procedure name.** The name of the procedure receiving the message. If the message was not sent

to a procedure within an ILE program, this field is not set and the length of data field is 0. A nested procedure name has each procedure name separated by a colon. The outermost procedure name is identified first followed by the procedures it contains. The innermost procedure is identified last in the string.

**Receiving program name.** The program name, or the ILE program name that contains the procedure that the message was sent to.

**Receiving type.** The type of the receiver (whether it is a program or a procedure). Possible values and their meanings follow:

- 0 Receiver is an original program model (OPM) program.
- 1 Receiver is a procedure within an ILE program, and the procedure name is up to and including 256 characters in length.
- 2 Receiver is a procedure within an ILE program, and the procedure name is 257 or more characters in length.

**Replacement data or impromptu message text.** The values for replacement variables in a predefined message, or the text of an impromptu message. If the message identifier field is not blank, this field contains message data. If the message identifier field is blank, this field contains impromptu message text.

Any pointer data in this field is marked as not valid if both:

- The API is called by a call stack entry not running in system state.
- The system security level is 50 or above.

**Reply status.** The reply status of the message (whether it accepts a reply, and if so, whether a reply has been sent). Possible values and their meanings follow:

- A Message accepts a reply, and a reply has been sent.
- W Message accepts a reply, and a reply has not been sent. (The message is waiting for a reply.)
- N Message does not accept a reply.

**Request level.** The level of the request-processing program that received the request message. If the message being listed is not a request, this field is set to 0.

**Request status.** Information regarding the processing status of the request message. Possible values and their meanings follow:

- O This request message has been received and processed.
- C This request message is currently being processed.
- N This request message has not yet been processed.

If the message being listed is not a request, this field is set to a blank character.

**Reserved.** An ignored field.

**Sender type.** The type of the sender (whether it is a program or procedure). Possible values and their meanings follow:

- 0 Sender is an OPM or a System Licensed Internal Code (SLIC) program with a name that is 12 characters or less.

- 1 Sender is a procedure within an ILE program, and the procedure name is up to and including 256 characters in length.
- 2 Sender is a procedure within an ILE program, and the procedure name is from 257 characters up to and including 4096 characters in length.
- 3 Sender is a SLIC program with a name that is from 13 characters up to and including 256 characters in length.

**Sending module name.** The name of the module that contains the procedure sending the message. If the message was not sent by a procedure within an ILE program, this field is not set and the length of data field is 0.

**Sending procedure name.** The name of the procedure sending the message. If the message was not sent by a procedure within an ILE program, this field is not set and the length of data field is 0. A nested procedure name has each procedure name separated by a colon. The outermost procedure name is identified first followed by the procedures it contains. The innermost procedure is identified last in the string.

**Sending program name.** The sending program name or ILE program name that contains the procedure sending the message.

**Size of message selection information specified.** The size of the message selection information field, in bytes, as specified in the call to the API.

**Starting message key.** The message key to begin searching for messages to list from the job. You can use these special values for the message key:

*'00000000'X* The first message to be returned is the oldest message in the queue.

*'FFFFFFFF'X* The first message to be returned is the newest message in the queue.

When the list direction field is \*NEXT, the first message listed is the first message with a message key equal to or greater than the key specified. If no message is found in this manner, an error is returned. When the list direction field is \*PRV, the first message listed is the first message with a message key equal or less than the key specified. If no message is found in this manner, an error is returned.

If a key of a reply message is specified, the message search begins with the inquiry, sender's copy, or notify message with which the reply is associated. It does not begin with the reply message itself.

**Note:** When a batch job is being listed, request messages that have not yet been processed or received are considered to have a sending date and time later than all other messages on the job log. This is also true for any diagnostic messages associated with those request messages. If the starting message key provided is to one of these messages, and the list direction is \*NEXT, only additional request messages and associated diagnostic messages that remain to be processed are listed. If the list direction is \*PRV, any earlier request messages or associated diagnostic messages are listed. These are followed by messages on the job log associated with request messages that have been or are being processed.

**Starting message key specified.** The starting message key as specified on the call to the API.

**Starting message key used.** The message keys of the first message actually listed by the API. If no message is listed, the value returned is the same as the value specified in the starting message key specified field.

**Status of data.** The status of the data listed for this message. Possible values and their meanings follow:

*blank* The data returned is complete.

- A* The caller of the API was not authorized to view the data. This occurs when the caller of the API is not authorized to the message file or the message file library containing a stored message being listed.
- D* The data was damaged. This occurs when the message file or library specified at send time for a stored message is damaged when the API is called.
- U* The data was unavailable. This occurs when the message file or library specified at send time for a stored message is exclusively used by another process when the API is called.
- N* The data was not found. This occurs when the message file or library specified at send time for a stored message cannot be found or resolved when the API is called.

This field is applicable to the field identifiers that are retrieved from the message file for a stored message. A description of the action that occurs for specific field identifiers when the status of data field is not blank follows:

- 0101* When the status of data field is not blank, the alert option field identifier contains blanks.
- 0301, 0302* When the status of data field is not blank, these message field identifiers contain message text regarding the problem encountered while attempting to access the message file. Both fields have the replacement data substituted.
- 0401, 0402, 0403, 0404* When the status of data field is not blank, these message help field identifiers contain message text regarding the problem encountered while attempting to access the message file. All fields have the replacement data substituted. The message help with formatting characters and message help with replacement data and formatting characters field identifiers also have the message formatting characters included.
- 0501* When the status of data field is not blank, the default reply field identifier contains the system default reply.
- 0801* When the status of data field is not blank, the message file library used field identifier contains blanks.

This field is also applicable to the various sending and receiving information fields when a problem is encountered while attempting to retrieve this information. This includes field identifiers 0602, 0603, 0604, 0605, 0606, 0702, 0703, 0704, 0705, and 0706. When one of these fields cannot be retrieved from the message, the status of data field is set to U and the field is set to blanks.

The status of data field is always blank for the other field identifiers.

**Time sent.** The time at which the message being listed was sent, in HHMMSS (hour, minute, and second) format.

**Type of data.** The type of data returned.

- C* The data is returned in character format.
- B* The data is returned in binary format.
- M* The data is returned in a mixed format. This value is returned for the field IDs 0606 and 0706, which contain a binary(4) value followed by an array of 10-character elements.

**User profile specified.** The user profile of the job from which to list messages as specified on the call to

the API.

**User profile used.** The actual user profile of the job used from which to list messages.

**User space library specified.** The name of the user space library as specified on the call to the API.

**User space library used.** The actual name of the library where the user space was found.

**User space name specified.** The name of the user space as specified on the call to the API.

**User space name used.** The actual name of the user space used to store the data listed.

## Error Messages

Message ID	Error Message Text
CPF1866 E	Value &1 for number of fields to return not valid.
CPF24B4 E	Severe error while addressing parameter list.
CPF24B7 E	Value &1 for call stack entry name length not valid.
CPF240D E	Message search direction specified is not valid.
CPF240E E	Format name of message selection information is not valid.
CPF240F E	Field identifier is not valid or is a duplicate of another field identifier specified.
CPF241E E	Call stack entry name is not valid.
CPF241F E	Length &1 specified for maximum message length is not valid.
CPF2410 E	Message key not found in message queue &1.
CPF2441 E	Not authorized to display job log.
CPF2443 E	Job log not displayed or listed because job has ended.
CPF247D E	Size of message selection information, &1, is not valid.
CPF247E E	CCSID &1 is not valid.
CPF2476 E	The maximum number of messages to list, &1, is not valid.
CPF252F E	Length &1 specified for maximum message help length is not valid.
CPF2532 E	Job message queue is damaged. Job log ended.
CPF3CAA E	List is too large for user space &1.
CPF3CF1 E	Error code parameter not valid.
CPF3C21 E	Format name &1 is not valid.
CPF3C51 E	Internal job identifier not valid.
CPF3C52 E	Internal job identifier no longer valid.

CPF3C53 E Job &3/&2/&1 not found.  
CPF3C55 E Job &3/&2/&1 does not exist.  
CPF3C58 E Job name specified is not valid.  
CPF3C59 E Internal identifier is not blanks and job name is not \*INT.  
CPF3C90 E Literal value cannot be changed.  
CPF9801 E Object &2 in library &3 not found.  
CPF9802 E Not authorized to object &2 in &3.  
CPF9803 E Cannot allocate object &2 in library &3.  
CPF9807 E One or more libraries in library list deleted.  
CPF9808 E Cannot allocate one or more libraries on library list.  
CPF9810 E Library &1 not found.  
CPF9811 E Program &1 in library &2 not found.  
CPF9812 E File &1 in library &2 not found.  
CPF9814 E Device &1 not found.  
CPF9820 E Not authorized to use library &1.  
CPF9821 E Not authorized to program &1 in library &2.  
CPF9822 E Not authorized to file &1 in library &2.  
CPF9825 E Not authorized to device &1.  
CPF9830 E Cannot assign library &1.  
CPF9831 E Cannot assign device &1.  
CPF9838 E User profile storage limit exceeded.  
CPF9872 E Program or service program &1 in library &2 ended. Reason code &3.

---

API introduced: V2R3

---

[Top](#) | [Message Handling APIs](#) | [APIs by category](#)

# List Nonprogram Messages (QMHLSTM) API

Required Parameter Group:

1	Qualified user space name	Input	Char(20)
2	Format name	Input	Char(8)
3	Message selection information	Input	Char(*)
4	Size of message selection information	Input	Binary(4)
5	Format of message selection information	Input	Char(8)
6	Error code	I/O	Char(*)

Default Public Authority: \*USE

Threadsafe: No

The List Nonprogram Messages (QMHLSTM) API lists messages from one or two nonprogram message queues. This API obtains the requested message information and returns it in a user space through the format specified in the parameter list.

If multiple message queues are specified, the messages are sorted by their sending dates and times. However, messages listed as a reply message to an inquiry or a sender's copy message are sorted differently. When a reply message exists for one of these types, it is listed immediately following the inquiry or sender's copy message with which it is associated.

If messages in multiple queues have identical sending dates and times, the message from the first queue in the qualified message queue names array is listed first.

If the last message listed is an inquiry or sender's copy message, the user must call the API again using parameters that would include listing the next later message after the inquiry or sender's copy message in order to determine and obtain an available reply message.

The QMHLSTM API lists messages from nonprogram message queues only. The QMHLSTM API cannot be used to list messages sent to a job log (including the external message (\*EXT) queue of a job). See [List Job Log Messages](#) (QMHLJOB) API for information on listing messages sent to a job log. QMHLSTM cannot be used to list messages sent to the history log (QHST).

The generated list replaces any existing information in the user space.

If the user space is not large enough to contain the amount of data that is to be returned, the user space is increased up to the maximum user space size allowed (16MB) or to the maximum amount of storage allowed to the user of the API. If this is not large enough to contain the data to be returned, only the number of complete messages that fit in the user space are returned. The information status field in the generic header is set to P (partial but accurate). The user can then resubmit the request from the last message returned to obtain the additional messages. The key of the last message listed for each message queue is provided in the ending message key field in the header portion of the user space.

New messages are prevented from being added to or removed from the message queues being listed during the use of this API. The maximum messages requested field and the number of fields to return field for each listed message increases the time that the queues are unavailable. Users of this API should use caution when listing many messages or many fields. Try to avoid causing lock-out situations on highly used message queues.



## Authorities and Locks

*Message queue*

\*USE

*Message queue library*

\*EXECUTE

*User space*

\*CHANGE

*User space library*

\*EXECUTE

*User space lock*

\*EXCLRD

## Required Parameter Group

### Qualified user space name

INPUT;CHAR(20)

The user space that receives the generated list, and the library in which it is located. The first 10 characters contain the user space name, and the second 10 characters contain the user space library. You can use these special values for the library name:

\**CURLIB* The job's current library

\**LIBL* The library list

### Format name

INPUT; CHAR(8)

The format of the returned message information. You must use this format:

*LSTM0100* Basic message information with identified return fields. This format is described in [LSTM0100 Format](#).

### Message selection information

INPUT; CHAR(\*)

The information that determines the message queues and messages to be listed. The format of this information is described in [MSLT0100 Format](#) and in [MSLT0200 Format](#).

### Size of message selection information

INPUT; BINARY(4)

The size, in bytes, of the message selection information parameter.

## Format of message selection information

INPUT; CHAR(8)

The format of the message selection information parameter. You must use one of these formats:

*MSLT0100* The specific information identifying the messages to be listed. This format is described in [MSLT0100 Format](#). Message text and data are returned in the CCSID of your job.

*MSLT0200* The specific information identifying the messages to be listed. This format is described in [MSLT0200 Format](#). This is the same as format MSLT0100 with the exception of the following:

- You are able to specify the CCSID in which you wish your message text and data returned.
- You may optionally specify a date and time with which to begin the list, relative to the beginning message key.

## Error code

I/O; CHAR(\*)

The structure in which to return error information. For the format of the structure, see [Error Code Parameter](#).

## Format of Generated Lists

The user space created consists of:

- A user area
- A generic header
- An input parameter section
- A header section
- LSTM0100 format

For details about the user area and generic header, see [User Space Format for List APIs](#). For details about the remaining items, see the following sections. For descriptions of each field in the list returned, see [Field Descriptions](#).

## Input Parameter Section

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	User space name specified
10	A	CHAR(10)	User space library specified
20	14	CHAR(8)	Format name specified
28	1C	CHAR(8)	Format of message selection information specified

36	24	BINARY(4)	Size of message selection information specified
40	28	BINARY(4)	Maximum messages requested specified
44	2C	CHAR(10)	List direction specified
54	36	CHAR(10)	Selection criteria specified
64	40	BINARY(4)	Severity criteria specified
68	44	BINARY(4)	Maximum message length specified
72	48	BINARY(4)	Maximum message help length specified
76	4C	BINARY(4)	Offset to message queue names specified
80	50	BINARY(4)	Offset to starting message keys specified
84	54	BINARY(4)	Number of message queues specified
88	58	BINARY(4)	Offset to identifiers of fields to return specified
92	5C	BINARY(4)	Number of fields to return specified
96	60	BINARY(4)	Coded character set identifier (CCSID) specified
100	64	CHAR(13)	Date and time criteria specified
113	71	CHAR(*)	Reserved
The offsets to these fields are specified in the previous offset variables.		ARRAY(*) of CHAR(20)	Message queue names specified
		ARRAY(*) of CHAR(4)	Starting message key specified
		ARRAY(*) of BINARY(4)	Identifiers of fields to return specified

## Header Section

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	User space name used
10	A	CHAR(10)	User space library used
20	14	BINARY(4)	Offset to message queue names used
24	18	BINARY(4)	Offset to starting message keys used
28	1C	BINARY(4)	Offset to ending message keys
32	20	BINARY(4)	Number of message queues used
36	24	BINARY(4)	Coded character set identifier (CCSID) used
40	28	CHAR(13)	Date and time of first message listed
53	35	CHAR(13)	Date and time of last message listed
66	42	CHAR(*)	Reserved
The offsets to these fields are specified in the previous offset variables.		ARRAY(*) of CHAR(20)	Message queue names used
		ARRAY(*) of CHAR(4)	Starting message key used

	ARRAY(*) of CHAR(4)	Ending message key used
--	---------------------	-------------------------

## LSTM0100 Format

The following table shows the information returned in the list data section of the user space for the LSTM0100 format. The offsets listed are from the beginning of the user space. For a detailed description of each field, see [Field Descriptions](#).

The structure defined by this format is repeated for each message entry returned.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Offset to the next entry
4	4	BINARY(4)	Offset to fields returned
8	8	BINARY(4)	Number of fields returned
12	C	BINARY(4)	Message severity
16	10	CHAR(7)	Message identifier
23	17	CHAR(2)	Message type
25	19	CHAR(4)	Message key
29	1D	CHAR(10)	Message file name
39	27	CHAR(10)	Message file library specified at send time
49	31	CHAR(10)	Message queue
59	3B	CHAR(10)	Message queue library used
69	45	CHAR(7)	Date sent
76	4C	CHAR(6)	Time sent
» 82	52	CHAR(6)	Microseconds «
» 88	58 «	CHAR(*)	Reserved
These fields repeat for each identifier field specified.		BINARY(4)	Offset to the next field information returned
		BINARY(4)	Length of field information returned
		BINARY(4)	Identifier field
		CHAR(1)	Type of data
		CHAR(1)	Status of data
		CHAR(14)	Reserved
		BINARY(4)	Length of data
		CHAR(*)	Data
		CHAR(*)	Reserved

## MSLT0100 Format

The organization of the MSLT0100 format of the message selection information parameter follows. For a detailed description of each field, see [Field Descriptions](#).

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Maximum messages requested
4	4	CHAR(10)	List direction
14	E	CHAR(10)	Selection criteria
24	18	BINARY(4)	Severity criteria
28	1C	BINARY(4)	Maximum message length
32	20	BINARY(4)	Maximum message help length
36	24	BINARY(4)	Offset to qualified message queue names
40	28	BINARY(4)	Offset to starting message keys
44	2C	BINARY(4)	Number of message queues
48	30	BINARY(4)	Offset to identifiers of fields to return
52	34	BINARY(4)	Number of fields to return
The offsets to these fields are specified in the previous offset variables.		ARRAY(*) of CHAR(20)	Qualified message queue names
		ARRAY(*) of CHAR(4)	Starting message key
		ARRAY(*) of BINARY(4)	Identifiers of fields to return

## MSLT0200 Format

The organization of the MSLT0200 format of the message selection information parameter follows. For a detailed description of each field, see [Field Descriptions](#).

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Maximum messages requested
4	4	CHAR(10)	List direction
14	E	CHAR(10)	Selection criteria
24	18	BINARY(4)	Severity criteria
28	1C	BINARY(4)	Maximum message length
32	20	BINARY(4)	Maximum message help length
36	24	BINARY(4)	Offset to qualified message queue names
40	28	BINARY(4)	Offset to starting message keys
44	2C	BINARY(4)	Number of message queues
48	30	BINARY(4)	Offset to identifiers of fields to return
52	34	BINARY(4)	Number of fields to return

56	38	BINARY(4)	Coded character set identifier (CCSID) to return text and data in
60	3C	CHAR(13)	Date and time criteria
73	49	CHAR(3)	Reserved (Must be blanks)
76	4C	BINARY(4)	Reserved (Must be 0)
The offsets to these fields are specified in the previous offset variables.		ARRAY(*) of CHAR(20)	Qualified message queue names
		ARRAY(*) of CHAR(4)	Starting message key
		ARRAY(*) of BINARY(4)	Identifiers of fields to return

## Valid Field Identifiers

The following table contains a list of the valid identifiers for the MSLT0100 and the MSLT0200 format.


Identifier	Type	Description
0101	CHAR(9)	Alert option
0201	CHAR(*)	Replacement data or impromptu message text
0301	CHAR(*)	Message
0302	CHAR(*)	Message with replacement data
0401	CHAR(*)	Message help
0402	CHAR(*)	Message help with replacement data
0403	CHAR(*)	Message help with formatting characters
0404	CHAR(*)	Message help with replacement data and formatting characters
0501	CHAR(*)	Default reply
0601	CHAR(26)	Qualified sender job name
0602	CHAR(1)	Sender type
0603	CHAR(12)	Sending program name
0604	CHAR(10)	Sending module name
0605	CHAR(256)	Sending procedure name
0606	BINARY(4) followed by ARRAY(*) of CHAR(10)	Number of sending statement numbers or instruction numbers available followed by an array of the sending statement numbers or instruction numbers
0702	CHAR(1)	Receiving type
0703	CHAR(10)	Receiving program name
0704	CHAR(10)	Receiving module name
0705	CHAR(256)	Receiving procedure name
0706	BINARY(4) followed by ARRAY(*) of CHAR(10)	Number of receiving statement numbers or instruction numbers available followed by an array of the receiving statement numbers or instruction numbers

0801	CHAR(10)	Message file library used
0901	CHAR(30)	Problem identification
1001	CHAR(1)	Reply status
1002	CHAR(1)	Critical break message status
1101	CHAR(1)	Request status
1201	BINARY(4)	Request level
1301	BINARY(4)	Coded character set identifier (CCSID) for text
1302	BINARY(4)	Coded character set identifier (CCSID) conversion status indicator for text
1303	BINARY(4)	Coded character set identifier (CCSID) for data
1304	BINARY(4)	Coded character set identifier (CCSID) conversion status indicator for data

## Field Descriptions

**Alert option.** Whether and when an SNA alert is created and sent for the message. If a message is listed, the value is one of the following:

- \**DEFER*      An alert is sent after local problem analysis.
- \**IMMED*      An alert is sent immediately when the message is sent to a message queue that has the allow alerts attribute set to \*YES.
- \**NO*            No alert is sent.
- \**UNATTEND*    An alert is sent immediately when the system is running in unattended mode (when the value of the alert status network attribute, ALRSTS, is \*UNATTEND).

For more information on alerts, see the [Alerts Support](#)  book.

This field is set to blanks if no alert option was specified when the message was sent.

**CCSID conversion status indicator for data.** The following values may be returned:

- 0    No conversion was needed because the CCSID of the data matched the CCSID you wanted the data converted to.
- 1    No conversion occurred because either the data was 65535 or the CCSID you wanted the data converted to was 65535.
- 2    No conversion occurred because you did ask for any data to be returned, or there was no \*CCHAR type data.
- 3    The data was converted to the CCSID specified using the best fit conversion tables.
- 4    A conversion error occurred using the best fit conversion tables so a default conversion was attempted. This completed without error.
- 1    An error occurred on both the best fit and default conversions. The data was not converted.

**CCSID conversion status indicator for text.** The following values may be returned:

- 0 No conversion was needed because the CCSID of the text matched the CCSID you wanted the text converted to.
- 1 No conversion occurred because either the text was 65535 or the CCSID you wanted the text converted to was 65535.
- 2 No conversion occurred because you did not ask for any text to be returned.
- 3 The text was converted to the CCSID specified using the best fit conversion tables.
- 4 A conversion error occurred using the best fit conversion tables so a default conversion was attempted. This completed without error.
- 1 An error occurred on both the best fit and default conversions. The data was not converted.

**Coded character set identifier (CCSID) for data.** The coded character set identifier that the data is returned in. If a conversion error occurs or if the CCSID you requested the data to be converted to is 65535, the CCSID of the data is returned. If there is no \*CCHAR replacement data, 65535 is returned. Otherwise the CCSID you wanted the data converted to is returned.

This only applies to the part of the replacement data that corresponds to a convertible character data type (\*CCHAR).

For more information about message handler and its use of CCSIDs, see [CCSIDs: Message Support](#) in the Globalization topic.

**Coded character set identifier (CCSID) for text.** The coded character set identifier that the message text is returned in. If a conversion error occurs or if the CCSID you requested the message text to be converted to is 65535, the CCSID that the message text is stored in is returned. Otherwise the CCSID you wanted your message text converted to is returned. If you do not want the message text converted before it is returned to you but you do want to know the CCSID that the message text is stored in, specify 65535 on the Coded character set identifier to return text and data in parameter. The CCSID that the message text is stored in is returned in the Coded character set identifier for text output field.

This applies to the following fields only:

- Message
- Message with replacement data
- Message help
- Message help with replacement data
- Message help with replacement data and formatting characters
- Message help with formatting characters

**Note:** This CCSID value does not apply to the replacement data that has been substituted into the text. See the Coded character set identifier for data for this information.

**Coded character set identifier (CCSID) to return text and data in.** The CCSID that the text and data are converted to before they are returned. The following values are allowed:

- 0 The text and data are converted to the CCSID of the job before being returned. This is the default value used when the MSLT0100 format is specified.

If the job is 65535 and the text or data is something other than EBCDIC single byte or EBCDIC mixed, the text and data are converted to the default job CCSID.



65535 The text and data are not converted before being returned.

*CCSID* Specify a valid CCSID you want your text and data converted to before being returned. The CCSID must be between 1 and 65535. Only CCSIDs that a job can be changed to are accepted. The CCSID is validated by this API. For a list of valid CCSIDs, see [CCSIDs: Message Support](#).

**Coded character set identifier (CCSID) specified.** The CCSID that was specified that the text and data are converted to before they are returned.

**Coded character set identifier (CCSID) used.** The CCSID that was used that the text and data are converted to before they are returned.

**Critical break message status.** Whether the message was sent by the operating system as a critical break message. The following values are returned:

- 1 The message was sent as a critical break message.
- 0 The message was not sent as a critical break message.

**Data.** The data returned for the specified identifier field.

**Date sent.** The date on which the message was sent, in CYYMMDD (century, year, month, and day) format.

**Date and time criteria.** The date and time starting point for messages to be listed. This is optional and must be blanks if not specified. The search for messages to list begins from the message that is specified by the starting message key. If the date and time criteria is specified in the call to the API and the list direction is \*PRV, then only messages whose time of arrival on the queue is equal to or less than the date and time criteria that is specified are listed. Similarly, if the direction is \*NEXT and the date and time criteria is specified on the call to the API, then only those messages whose time of arrival on the queue is equal to or greater than that specified are listed. The format of this field is in the CYYMMDDHHMMSS as follows:

- C Century, where 0 indicates years 19xx and 1 indicates years 20xx.
- YY Year
- MM Month
- DD Day
- HH Hour
- MM Minute
- SS Second

**Date and time criteria specified.** The date and time criteria as specified on the call to the API.

**Date and time of first message listed.** The date and time of the first message that is actually listed by the API. If no message is actually listed by the API, this field is blank.

**Date and time of last message listed.** The date and time of the last message that is actually listed by the API. If no message is actually listed by the API, this field is blank.

**Default reply.** The text of the default reply. When a stored message is being listed and a default reply exists. If this is not an inquiry message or no default reply exists, this field is not used and the length of data

field is 0.

**Ending message key used.** The message keys of the last message actually listed by the API. If no message is listed from a particular queue, the value returned for that queue is the same as that in the starting message key specified field.

**Format of message selection information specified.** The format name of the message selection information parameter as specified on the call to the API.

**Format name specified.** The format name as specified on the call to the API.

**Identifier field.** The field returned. See [Valid Field Identifiers](#) for the list of valid field identifiers.

**Identifiers of fields to return.** The list of the field identifiers to be returned in the LSTM0100 format. For a list of the valid field identifiers, see [Valid Field Identifiers](#). An error is returned if the identifier matches one specified earlier in the array, or if the identifier is not valid.

**Identifiers of fields to return specified.** The list of field identifiers to return as specified on the call to the API.

**Length of data.** The length of the data returned for the data field, in bytes.

**Length of field information returned.** The total length of information returned for this field, in bytes.

**List direction.** The direction to list messages. You must use one of these directions:

\**NEXT* Returns messages that are newer than the messages specified by the starting message key field.

\**PRV* Returns messages that are older than the message specified by the starting message key field.

If multiple message queues are to be listed, messages are intermixed and sorted by date and time in the specified manner.

**List direction specified.** The direction to list messages as specified on the call to the API.

**Maximum message help length.** The maximum number of characters of text that this API returns for field identifiers 0401, 0402, 0403, and 0404. (See [Valid Field Identifiers](#).)

Specify a value to limit the number of characters returned for field identifiers 0401, 0402, 0403, and 0404. This value can be no smaller than 4. The maximum allowed value is 32765. To specify that the maximum length be used, use the special value of -1. This value is not checked if field identifiers 0401, 0402, 0403, or 0404 are not specified.

**Maximum message help length specified.** The maximum number of characters to return for field identifiers 0401, 0402, 0403, and 0404 as specified on the call to the API.

**Maximum message length.** The maximum number of characters of text that this API returns for field identifiers 0301 and 0302. (See [Valid Field Identifiers](#).)

Specify a value to limit the number of characters returned for field identifiers 0301 and 0302. This value can be no smaller than 4. The maximum allowed value is 32765. To specify that the maximum length be used, use the special value of -1. This value is not checked if field identifiers 0301 or 0302 are not specified.

**Maximum message length specified.** The maximum number of characters to return for field identifiers 0301 and 0302 as specified on the call to the API.

**Maximum messages requested.** The maximum number of messages to be returned.

If fewer messages than the number requested exist on the queues, only the number of messages that exist are returned. No error is signaled, and the information status field in the generic header would be marked as C for complete and accurate.

Use the special value of -1 to list all messages on the queues in the specified list direction. The list runs from the starting message keys that meet the selection and severity criteria.

**Maximum messages requested specified.** The number of messages requested to be listed as specified on the call to the API.

**Message.** The text of a predefined message without replacement data substitution. If an impromptu message is listed, this field contains the impromptu message text.

**Message file library specified at send time.** The name of the library containing the message file as specified when the message was sent. If \*CURLIB or \*LIBL was specified for the library when the message was sent, that value is returned as the library here. For the actual library used when the message is sent, see the message file library used field.

**Message file library used.** The actual name of the library that contains the message file used to retrieve the message information. If an immediate message is listed, this field is set to blanks.

**Message file name.** The name of the message file containing the message listed.

**Message help.** The message help for the message listed without formatting characters and without replacement data. If an impromptu message is listed, this field contains the impromptu message text.

**Message help with formatting characters.** The message help for the message listed, including formatting characters.

Three format control characters can be returned within the message. They are defined in the online help for the Add Message Description (ADDMSGD) command to have these meanings:

*&N* Forces the text to a new line (column 2). If the text is longer than one line, the next lines are indented to column 4 until the end of text or another format control character is found.

*&P* Forces the text to a new line indented to column 6. If the text is longer than one line, the next lines start in column 4 until the end of text or another format control character is found.

*&B* Forces the text to a new line, starting in column 4. If the text is longer than one line, the next lines are indented to column 6 until the end of text or another format control character is found.

If an impromptu message is listed, this field contains the impromptu message text.

**Message help with replacement data.** The message help for the message listed, including the replacement data. If an impromptu message is listed, this field contains the impromptu message text.

**Message help with replacement data and formatting characters.** The message help for the message listed, including the replacement data and the formatting characters. See the message help with formatting characters field for an explanation of formatting characters. If an impromptu message is listed, this field contains the impromptu message text.

**Message identifier.** The identifying code of the message listed. If an impromptu message is listed, this field is set to blanks.

**Message key.** The key of the message listed.

**Message queue.** The name of the message queue where the message was listed.

**Message queue library used.** The actual library that contains the message queue.

**Message queue names specified.** The qualified message queue names specified on the call to the API.

**Message queue names used.** The actual message queue names used to list messages. The first 10 characters are the message queue name, and the second 10 characters are the message queue library.

**Message severity.** The severity of the message listed. Possible values are 0 through 99.

**Message type.** The type of message listed. The possible values and their meanings follow:

Value	Message Type
01	Completion
02	Diagnostic
04	Informational
05	Inquiry
06	Sender's copy
08	Request
10	Request with prompting
14	Notify, exception already handled when API is called
15	Escape, exception already handled when API is called
16	Notify, exception not handled when API is called
17	Escape, exception not handled when API is called
21	Reply, not checked for validity
22	Reply, checked for validity
23	Reply, message default used
24	Reply, system default used
25	Reply, from system reply list

**Message with replacement data.** The text of a predefined message with the replacement data included. If an impromptu message is listed, this field contains the impromptu message text.

➤ **Microseconds.** The microseconds part of the time sent. ⏪

**Number of fields returned.** The number of identifier fields returned to the application.

**Number of fields to return.** The number of fields to return in the LSTM0100 format (the number of entries in the identifiers of fields to return array).

**Number of fields to return specified.** The number of identifier fields to return as specified on the call to the API.

**Number of message queues.** The number of message queues to list. The valid values follow:

- 1 One message queue listed. Both the qualified message queue names field and the starting message key field contain one entry.
- 2 Two message queues listed. Both the qualified message queue names field and the starting message key field contain two entries.

**Number of message queues specified.** The number of message queues to be listed as specified on the call to the API. This is the size of the declared array of the message queue names specified and the starting message key specified fields.

**Number of message queues used.** The number of message queues listed. This is the number of elements in the message queue names used, starting message key used, and ending message key arrays.

**Number of receiving statement numbers or instruction numbers available followed by an array of the receiving statement numbers or instruction numbers.** This field can be specified for the QMHLSTM API, but it never returns any data and the length of data field is 0.

**Number of sending statement numbers or instruction numbers available followed by an array of the sending statement numbers or instruction numbers.** This field can be specified for the QMHLSTM API, but it never returns any data and the length of data field is 0.

**Offset to ending message key.** The offset, in bytes, from the beginning of the user space to the beginning of the ending message key field.

**Offset to fields returned.** The offset, in bytes, from the beginning of the user space to the beginning of the first repeating identified field of the LSTM0100 format.

**Offset to identifiers of fields to return.** The offset, in bytes, from the beginning of the message selection information parameter to the beginning of the identifiers of fields to return field.

**Offset to identifiers of fields to return specified.** The offset, in bytes, from the beginning of the user space to the beginning of the identifiers of fields to return specified field.

**Offset to message queue names specified.** The offset, in bytes, from the beginning of the user space to the beginning of the message queue names specified field.

**Offset to message queue names used.** The offset, in bytes, from the beginning of the user space to the beginning of the message queue names used field.

**Offset to qualified message queue names.** The offset, in bytes, from the beginning of the message selection information parameter to the beginning of the qualified message queue names field.

**Offset to starting message keys.** The offset, in bytes, from the beginning of the message selection information parameter to the beginning of the starting message key field.

**Offset to starting message key specified.** The offset, in bytes, from the beginning of the user space to the beginning of the starting message key specified field.

**Offset to starting message key used.** The offset, in bytes, from the beginning of the user space to the beginning of the starting message key used field.

**Offset to the next entry.** The offset, in bytes, from the beginning of the user space to the beginning of the next message entry.

**Offset to the next field information returned.** The offset, in bytes, from the beginning of the user space to the beginning of the next repeating identified field of the LSTM0100 format.

**Problem identification.** The number the system generates to identify a problem if problem analysis can be run for the message being listed. The problem identification is in the following format:

*CHAR 1-10* Problem ID number. The number the system generates to identify the problem.

*CHAR 11-30* Origin system in the format *network-ID.control-point-name*.

If problem analysis cannot be run, and this field is specified, the length of data field is 0.

**Qualified message queue names.** The list of message queues and the libraries where the message queues are located. The number of entries in the array must match the number specified in the number of message queues field. The first 10 characters of each entry contain the message queue name, and the second 10 characters of each entry contain the message queue library. You can use the following values for the library name:

*\*CURLIB* The job's current library

*\*LIBL* The library list

**Qualified sender job name.** The name of the job that sent the message. The job name has three parts:

*CHAR 1-10* The specific job name.

*CHAR 11-20* The specific user profile name.

*CHAR 21-26* The specific job number.

**Receiving module name.** This field can be specified for the QMHLSTM API, but it never returns any data and the length of data field is 0.

**Receiving procedure name.** This field can be specified for the QMHLSTM API, but it never returns any data and the length of data field is 0.

**Receiving program name.** This field can be specified for the QMHLSTM API, but it never returns any data and the length of data field is 0.

**Receiving type.** This field can be specified for the QMHLSTM API, but it never returns any data and the length of data field is 0.

**Replacement data or impromptu message text.** The values for replacement variables in a predefined message, or the text of an impromptu message. If the message identifier field is not blank, this field contains message data. If the message identifier field is blank, this field contains impromptu message text.

Any pointer data in this field is marked as not valid if both:

- The API is called by a call stack entry not running in system state.
- The system security level is 50 or above.

**Reply status.** The reply status of the message (whether it accepts a reply, and if so, whether a reply has been sent). Possible values and their meanings follow:

*A* Message accepts a reply, and a reply has been sent.

*W* Message accepts a reply, and a reply has not been sent. (The message is waiting for a reply.)

*N* Message does not accept a reply.

**Request level.** This field can be specified for the QMHLSTM API, but it never returns any data and the length of data field is 0.

**Request status.** This field can be specified for the QMHLSTM API, but it never returns any data the length of data field is 0.

**Reserved.** An ignored field.

**Selection criteria.** The type of messages to be listed. Valid values follow:

- \**ALL* All messages are to be listed.
- \**MNNR* Only messages not requiring a reply are listed. This includes informational, completion, diagnostic, request, notify, escape, reply, answered inquiry, and answered sender's copy messages.
- \**MNR* Only messages needing a reply are listed. This includes only unanswered inquiry messages.
- \**PAR* Only messages that can have problem analysis run against them are listed.
- \**SCNR* Only sender's copy messages requiring a reply are listed. This includes only unanswered sender's copy messages.

**Selection criteria specified.** The selection criteria as specified on the call to the API.

**Sender type.** This field can be specified for the QMHLSTM API, but it never returns any data and the length of data field is 0.

**Sending module name.** This field can be specified for the QMHLSTM API, but it never returns any data and the length of data field is 0.

**Sending procedure name.** This field can be specified for the QMHLSTM API, but it never returns any data and the length of data field is 0.

**Sending program name.** The sending program name or ILE program name that contains the procedure sending the message. Under certain conditions, the actual name of the program that sent the message is not known. In these cases, this field contains the 6-byte hexadecimal address of the program converted into 12 displayable characters. In all other cases, the 10-character program name is returned left-justified in the field; the final 2 characters contain blanks.

**Sending statement numbers or instruction numbers.** This field can be specified for the QMHLSTM API, but it never returns any data and the length of data field is 0.

**Severity criteria.** The minimum severity of a message to be included in the list. The value must be between 0 and 99. To retrieve all messages, specify a severity criteria of 0.

**Severity criteria specified.** The severity criteria as specified on the call to the API.

**Size of message selection information specified.** The size of the message selection information field, in bytes, as specified in the call to the API.

**Starting message key.** The message key used to begin searching for messages to list from the corresponding entry in the qualified message queue names field. You can use these special values for the message keys:

'00000000'X The first message to be returned is the oldest message in the queue.

'FFFFFFFF'X The first message to be returned is the newest message in the queue.

If a value other than X'00000000' or X'FFFFFFFF' is specified and a message with that key does not exist, an error is returned.

If the message that is specified by the starting message key exists but does not meet the selection criteria, the severity criteria, or the date and time criteria fields that are specified, no error is returned. The search for messages to list begins from the message that is specified by the starting message key.

**Starting message key specified.** The starting message keys as specified on the call to the API.

**Starting message key used.** The message keys of the first message actually listed by the API. If no message is listed from a particular message queue, the value returned for that queue is the same as that in the starting message key specified field.

**Status of data.** The status of the data listed for this message. Possible values and their meanings follow:

- blank* The data returned is complete.
- A* The caller of the API was not authorized to view the data. This occurs when the caller of the API is not authorized to the message file or message file library containing a stored message being listed.
- D* The data was damaged. This occurs when the message file or library specified at send time for a stored message is damaged when the API is called.
- U* The data was unavailable. This occurs when the message file or library specified at send time for a stored message is exclusively used by another process when the API is called.
- N* The data was not found. This occurs when the message file or library specified at send time for a stored message cannot be found or resolved when the API is called.

This field is applicable to the field identifiers that are retrieved from the message file for a stored message. A description of the action that occurs for specific field identifiers when the status of data field is not blank follows:

- 0101* When the status of data field is not blank, the alert option field identifier contains blanks.
- 0301, 0302* When the status of data field is not blank, these message field identifiers contain message text. regarding the problem encountered while attempting to access the message file. Both fields have the replacement data substituted.
- 0401, 0402, 0403, 0404* When the status of data field is not blank, these message help field identifiers contain the text of the message regarding the problem encountered while attempting to access the message file. All fields have the replacement data substituted. The message help with formatting characters and message help with replacement data and formatting characters field identifiers also have the message formatting characters included.
- 0501* When the status of data field is not blank, the default reply field identifier contains the system default reply.
- 0801* When the status of data field is not blank, the message file library used field identifier contains blanks.

This field is also applicable to the various sending information fields (identifiers 0601, 0603) when a problem is encountered while attempting to retrieve this information. When one of these fields cannot be



retrieved from the message:

- The status of data field is set to N.
- The length of data field is set to 0.

The status of data field is always blank for the other field identifiers. The length of data field is zero.

**Time sent.** The time at which the message being listed was sent, in HHMMSS (hour, minute, and second) format.

**Type of data.** The type of data returned.

*C* The data is returned in character format.

*B* The data is returned in binary format.

*M* The data is returned in a mixed format. This value is returned for the field IDs 0606 and 0706.

**User space library specified.** The name of the user space library as specified on the call to the API.

**User space library used.** The actual name of the library where this user space was found.

**User space name specified.** The name of the user space as specified on the call to the API.

**User space name used.** The actual name of the user space used to store the data listed.

## Error Messages

Message ID	Error Message Text
CPF1060 E	Date not valid.
CPF1061 E	Time not valid.
CPF1866 E	Value &1 for number of fields to return not valid.
CPF24B4 E	Severe error while addressing parameter list.
CPF240D E	Message search direction specified is not valid.
CPF240E E	Format name of message selection information is not valid.
CPF240F E	Field identifier is not valid or is a duplicate of another field
CPF2401 E	Not authorized to library &1.
CPF2403 E	Message queue &1 in &2 not found.
CPF2408 E	Not authorized to message queue &1.
CPF241D E	Severity criteria specified is not valid.
CPF241F E	Length &1 specified for maximum message length is not valid.
CPF2410 E	Message key not found in message queue &1.
CPF2433 E	Function not allowed for system log message queue &1.
CPF2444 E	Number of message queues, &1, is not valid.
CPF2467 E	&3 message queue &1 in library &2 logically damaged.
CPF247D E	Size of message selection information, &1, is not valid.

CPF247E E CCSID &1 is not valid.  
CPF2476 E The maximum number of messages to list, &1, is not valid.  
CPF2477 E Message queue &1 currently in use.  
CPF252F E Length &1 specified for maximum message help length is not valid.  
CPF2538 E Value for selection criteria not valid.  
CPF3CAA E List is too large for user space &1.  
CPF3CF1 E Error code parameter not valid.  
CPF3C21 E Format name &1 is not valid.  
CPF3C39 E Value for reserved field not valid.  
CPF3C90 E Literal value cannot be changed.  
CPF8198 E Damaged object found.  
CPF9807 E One or more libraries in library list deleted.  
CPF9808 E Cannot allocate one or more libraries on library list.  
CPF9811 E Program &1 in library &2 not found.  
CPF9812 E File &1 in library &2 not found.  
CPF9814 E Device &1 not found.  
CPF9821 E Not authorized to program &1 in library &2.  
CPF9822 E Not authorized to file &1 in library &2.  
CPF9825 E Not authorized to device &1.  
CPF9830 E Cannot assign library &1.  
CPF9831 E Cannot assign device &1.  
CPF9838 E User profile storage limit exceeded.  
CPF9872 E Program or service program &1 in library &2 ended. Reason code &3.

---

API introduced: V2R3

---

[Top](#) | [Message Handling APIs](#) | [APIs by category](#)

# Move Program Messages (QMHMOVPM) API

## Required Parameter Group:

1	Message key	Input	Char(4)
2	Message types	Input	Array of Char(10)
3	Number of message types	Input	Binary(4)
4	To call stack entry	Input	Char(*) or Pointer
5	To call stack counter	Input	Binary(4)
6	Error code	I/O	Char(*)

## Optional Parameter Group 1:

7	Length of to call stack entry	Input	Binary(4)
8	To call stack entry qualification	Input	Char(20)

## Optional Parameter Group 2:

9	To call stack entry data type	Input	Char(10)
10	From call stack entry address	Input	Char(16) or Pointer
11	From call stack counter	Input	Binary(4)

Default Public Authority: \*USE

Threadsafe: Yes

The Move Program Message (QMHMOVPM) API moves messages from a call message queue in the call stack to the call message queue of an earlier call stack entry. Moving a message does not change the sender information stored with the message. However, moving an escape message automatically changes it to a diagnostic message.

You can use the QMHMOVPM API to pass messages up the call stack, transferring important information to a previous program. When messages are sent to a call stack entry and the program running in that entry ends without moving its messages up the stack, the programs left in the stack cannot receive those messages unless:

- They use the message key of those messages.
- They use the List Job Log (QMHLJOBL) API.

For example, assume a program is sent several diagnostic messages and one escape message in response to an error. Another program earlier in the stack can handle the error. If the first program uses the QMHMOVPM API to move all the diagnostic messages to the program earlier in the stack, it then uses [Resend Escape Message](#) (QMHRSNEM) API to send the escape message to the calling program.

In a multithreaded job, messages can be moved only from one call message queue to another call message queue within the thread that calls this API. Messages cannot be moved to a call stack entry in another thread.

## Authorities and Locks

None.

## Required Parameter Group

### Message key

INPUT; CHAR(4)

When moving a single, specific message, use the key to that message for this parameter. The key is assigned by the command or API that sends the message.

When using the message types parameter to move a group of messages, use blanks for this parameter.

### Message types

INPUT; ARRAY of CHAR(10)

The type or types of the messages being moved.

When moving a group of messages, specify a list of one through four message types. You can use these values in the list to move all messages of one or more types:

- \**COMP* Completion
- \**DIAG* Diagnostic
- \**ESCAPE* Escape. After an escape message is moved, its message type changes to diagnostic.
- \**INFO* Informational

For descriptions of the message types, see [Message Types](#).

If there are no messages of a type you specify, the QMHMOVPM API does not return an error. It simply returns control to the calling program.

If the number of message types parameter specifies 0, this parameter is ignored.

### Number of message types

INPUT; BINARY(4)

The number of message types specified in the message types parameter.

When moving a single message by specifying the message key in the message key parameter, use 0 for the number of message types parameter.

If you use blanks for the message key parameter and specify one or more message types, the value must be 1 through 4.

### To call stack entry

INPUT; CHAR(\*) OR POINTER

The call stack entry to move the messages to, or the call stack entry to start counting from when using a value other than 0 for the To call stack counter parameter. The call stack entry specified

must be in the call stack; you cannot specify the external message queue.

You can specify a call stack entry by:

- Providing the name of the OPM program or ILE procedure running in the entry.
- Providing a pointer to the call stack entry, or
- Using one of the following special values:

\*           The current call stack entry (that is, the one in which the program or procedure using the API is running).

*\*PGMBDY*   The call stack entry is for the boundary of the specified program object. The program object is specified by providing a program name in the optional To call stack entry qualification parameter. The program object can be also be provided by not using the optional parameter or by specifying \*NONE for the program name. In this case, the program object is assumed to be the program that is using the API.

This option essentially identifies the oldest call stack entry which began a sequence of calls, where each call in this sequence involved the same program object. The call sequence could involve recursive calls to the same program or, in the case of ILE, calls to different procedures of the same ILE program or ILE service program.

For OPM programs, in most cases, using \*PGMBDY produces the same result as using \* or using an OPM program name in this parameter. A difference will appear when an OPM program calls itself recursively. In this case, using \* or an OPM program name identifies the current recursion level of the program. In contrast, using \*PGMBDY identifies the first recursion level.

For an ILE program, this option can be used to identify the first procedure of the ILE program that was called in the current sequence. If the ILE program was called using a dynamic call, this is the PEP (program entry procedure).

For ILE service programs, this option can be used to identify the call stack entry for the first procedure that was called in the identified service program.

*\*CTLBDY*   The call stack entry at the most recent control boundary. This call stack entry is in the same activation group as the one using the API.

If this keyword value is used and there is no control boundary in the current call stack entry, the error CPF24C8 is returned to the user of this API. This happens if the only entries on the call stack are for OPM programs.

In some cases, \*PGMBDY and \*CTLBDY identify the same call stack entry. The option \*CTLBDY does not care if all call stack entries in a call sequence involve the same program object. It only cares that a sequence started at a control boundary. Conversely, the start of a call sequence identified by \*PGMBDY may not fall on a control boundary.

*\*PGMNAME* The to call stack entry is identified by the program name and, optionally, module name that is provided by the optional To call stack entry qualification parameter.

For OPM programs, specifying this special value and the optional parameter is the same as specifying the OPM program name in this parameter.

For ILE programs or ILE service programs this special key value indicates that a procedure name is not being specified. Rather the call stack entry is identified by providing only the ILE program or service program name and optionally the ILE module name. This means that the call stack entry is the most recently called procedure that is part of the specified ILE program or ILE service program (and part of the ILE module if module name is also specified). The name of this most recently called procedure is not important in determining the correct call stack entry.

If this special value is specified with a program name only, then the call stack entry is the most recently called OPM program that has the specified program name or the most recently called ILE procedure that is part of an ILE program or ILE service program of the specified name, whichever occurs most recently on the call stack. If the module name is also specified, then the call stack entry is the most recently called ILE procedure that is part of specified the ILE program or service program and module. If a module name is given, the call stack entry cannot be an OPM program.

If the call stack entry is identified by pointer, the pointer specified must address a valid call stack entry within the same job as the one the API is used in. Alternatively, the pointer can be set to Null. The Optional Parameter Group 1 must be used and the Length of To call stack entry parameter must be set to 16. In addition, the Optional Parameter Group 2 must also be used and the Call stack entry format parameter must be set to \*PTR.

If the pointer provided is set to Null, this indicates that the call stack entry is the one in which the API is being used.

If the pointer does not address a valid call stack entry or is not a Null pointer, the error message CPF24C5 is sent to the user of the API.

The call stack entry can be a nested procedure name from 1 through 4096 characters in length. When specifying nested procedures, each procedure name must be separated by a colon, and the outermost procedure is identified first followed by the procedures it contains. The innermost procedure is the last procedure identified in the string.

The call stack entry can be a partial name. To specify a partial name, place three less-than signs (<<<) at the beginning of the call stack entry identifier, or place three greater-than signs (>>>) at the end of the call stack entry identifier, or place both the less-than signs and the greater-than signs at their respective ends of the call stack entry identifier. The value for the call stack entry excluding the less-than signs and the greater-than signs is used to search backward through the stack for the requested call stack entry name.

When searching for a partial call stack entry name:

- If the less-than signs (<<<) are specified only at the beginning of the call stack entry name, the less-than signs are truncated and the remaining character string is right-justified. The remaining string is then compared to the current call stack entry on the call stack. The

comparison starts at the end of the call stack entry name and backwardly compares the number of characters in the specified string.

- If the greater-than signs (>>>) are specified only at the end of the call stack entry name, the greater-than signs are truncated. The remaining character string is compared to the current call stack entry on the call stack. The comparison starts at position 1 of the call stack entry name and compares the number of characters in the specified string.
- If the less-than signs (<<<) are specified at the beginning of the call stack entry name and the greater-than signs (>>>) are specified at the end of the call stack entry name, both the less-than signs and the greater-than signs are truncated. The remaining characters are used to scan and to compare the entire length of the specified string and the current call stack entry on the call stack.

**Note:** If the optional parameters Length of to call stack entry and To call stack entry data type are not specified, this parameter is assumed to be CHAR(10).

### To call stack counter

INPUT; BINARY(4)

A number identifying the location in the call stack of the call stack entry to whose message queue the messages are to be moved. The number is relative to the call stack entry identified by the To call stack entry parameter. It indicates how many calls up the call stack the target entry is from the one identified by the To call stack entry parameter. Valid values follow:

- 0* Move the messages to the message queue of the entry specified by the to call stack entry parameter. You cannot use 0 when the to call stack entry parameter specifies \*, a Null pointer, or a pointer that points to the user of the API. This combination means that you want to move the message from yourself to yourself which is not a valid operation.
- 1* Move the messages to the message queue of the call stack entry one earlier on the stack than the one identified by the To call stack entry parameter.
- n (any positive number)* Move the messages to the queue of the nth call stack entry earlier in the stack from the one identified by the To call stack entry parameter.

You can use any positive number that does not exceed the actual number of call stack entries in the call stack. Do not include the external message queue in your count.

## Optional Parameter Group 1

### Length of to call stack entry

INPUT; BINARY(4)

The length of the value for the to call stack entry parameter.

Valid values for this parameter are as follows:

- 1 to and including 4096 if partial name indicators are not used.
- 16 if the call stack entry parameter is a pointer.
- 1 to and including 4102 if partial name indicators are used.

**Note:** The actual length of the call stack entry name cannot exceed 4096 characters. If this parameter is not used, the value for the call stack entry parameter is assumed to be 10 characters in length.

### To call stack entry qualification

INPUT; CHAR(20)

Further identifies the To call stack entry. The parameter consists of two 10 character parts. The first part is the module name qualifier and the second part is the program name qualifier. The values provided in this parameter are used as a qualifier for the value provided in the To call stack entry parameter. The values that can be specified here depend upon the value provided in the To call stack entry parameter.

The special values discussed with this parameter can be used to indicate that the module name qualifier or program name qualifier is not being specified:

If the To call stack entry parameter contains an ILE procedure name then this parameter can contain the name of the module and the ILE program to further qualify the procedure name. The first 10 characters specify the module name, and the second 10 characters specify the ILE program name. If this parameter is not used or if \*NONE is specified for both qualifiers, only the specified procedure name is used to determine the call stack entry.

If the required To call stack entry parameter contains the key value \*PGMNAME, then the program name qualifier must contain either an OPM program name or an ILE program name. For ILE, the module name qualifier may contain a module name; otherwise it must contain \*NONE.

When the To call stack entry parameter contains the special value \* or \*CTLBDY, the Optional Parameter Group 1 should either not be used or both the module name and program name qualifiers must be specified as \*NONE.

When the To call stack entry parameter contains an OPM program name, the Optional Parameter Group 1 should also either not be used or both the module name and program name qualifiers should be specified as \*NONE. If a module name or program name is specified here, the name specified as the To call stack entry is interpreted as an ILE procedure name rather than an OPM program name. Either the entry would not be found or an incorrect entry would be found.

When the special value \*PGMBDY is used, the Optional Parameter Group 1 may be used. In this case, the module name qualifier must be specified as \*NONE. For the program name qualifier, an OPM or ILE program name may be specified or \*NONE may be used.

If the To call stack entry is identified by pointer, this parameter must still be passed to the API. The value specified for the module name qualifier must be specified as \*NONE. The program name qualifier serves as a qualifier for the pointer and must be specified as one of the following special values:

*\*NONE*      The call stack entry addressed by the pointer is the one to which the message is to be moved or the one to start counting from when using a value other than 0 for the To call stack counter parameter.



*\*PGMBDY* The call stack entry to move the message to or to start counting from is an ILE program boundary. The ILE program is the one which contains the procedure that is running in the call stack entry addressed by the pointer.

If the ILE program was called using a dynamic call, using this special value with a pointer will identify the call stack entry for the PEP of that program. If a call was made using a procedure pointer, it will identify the call stack entry for the procedure that was pointed to.

If the pointer addresses a call stack entry that is running a procedure from an ILE service program, this option can be used to identify the call stack entry for the first procedure that was called in that service program.

If the call stack entry addressed by the pointer is running an OPM program than using the special value *\*PGMBDY* here will have the same effect as using *\*NONE* in most cases. A difference will occur if the OPM program called itself recursively. In this case using *\*PGMBDY* identifies the first recursion level while *\*NONE* identifies the current recursion level.

## Optional Parameter Group 2

Use these optional parameters when:

- The call stack entry to which the messages are to be moved is identified by a pointer.
- The messages to be moved are not on the current call stack entry's message queue.

### To call stack entry data type

INPUT; CHAR(10)

Whether the value of the To call stack entry parameter is a character string (a name or special value) or a pointer.

Use one of the following special values;

*\*CHAR* Value of the parameter is a character string (name or special value)

*\*PTR* Value of the parameter is a pointer.

If the above optional parameter is not specified, it is assumed that the value of the To call stack entry parameter is a character string.

### From call stack entry address

INPUT CHAR(16) OR POINTER

A pointer to the call stack entry where the messages to be moved exist or the call stack entry to start counting from if the from call stack counter is not 0.

The pointer that is specified must be an invocation pointer that addresses a valid call stack entry or must be set to Null.

For program languages that do not support pointers, a Null pointer can be presented by using a 16 byte variable. The first byte of the variable is set to the special value *\** and the remaining 15 bytes set to blanks.

If the pointer provided is set to Null, this indicates the current call stack entry (the one using the API).

If the pointer is not a valid pointer or if it is a valid pointer but is not Null nor does it address a valid call stack entry, the error message CPF24C5 is sent to the user of the API.

### **From call stack counter**

INPUT; BINARY(4)

A number identifying the location in the call stack of the call stack entry where the messages to be moved exist. The number is relative to the To call stack entry that is addressed by the From call stack entry address pointer. It indicates how many calls up the call stack the entry is from the one that is addressed by the pointer. Valid values follow:

- |                                |   |
|--------------------------------|---|
| <i>0</i>                       | The message to move exists at the call stack entry that is addressed by the pointer.  |
| <i>1</i>                       | The message to move exists at the call stack entry that is one earlier in the call stack than the one addressed by the pointer. |
| <i>n (any positive number)</i> | The message to move exists at the nth call stack entry earlier in the stack from the one addressed by the pointer.              |
- You can use any positive number that does not exceed the actual number of call stack entries in the call stack. Do not include the external message queue in your count.

The From call stack entry cannot be the same as the To call stack entry. Additionally, the To call stack entry must be earlier on the call stack than the From call stack entry.

## **Error Messages**

<b>Message ID</b>	<b>Error Message Text</b>
CPF24A3 E	Value for call stack counter parameter not valid.
CPF24A5 E	Value of &1, for number of message types, not valid.
CPF24BF E	Module or bound-program name is blank.
CPF24B3 E	Message type &1 not valid.
CPF24B4 E	Severe error while addressing parameter list.
CPF24B5 E	Not able to move message.
CPF24B7 E	Value &1 for call stack entry name length not valid.
CPF24B9 E	When call stack entry name is '*' or '*CTLBDY', module name and program name must be '*NONE'.
CPF24C5 E	Pointer to call stack entry not valid.
CPF24C6 E	Value of To call stack entry data type parameter not valid.

CPF24C8 E Control boundary not found on call stack.  
CPF24C9 E Program boundary not found on call stack.  
CPF24CB E \*PGMNAME requires a specified program name.  
CPF24CC E Call stack entry &2 for \*PGMNAME not found.  
CPF24CD E Module name cannot be specified when \*PGMBDY is used.  
CPF24CE E Qualifier &1 incorrect for use with pointer.  
CPF241B E Message type &1 in system program is not valid.  
CPF2410 E Message key not found in message queue &1.  
CPF247A E Call stack entry not found.  
CPF2471 E Length of field not valid.  
CPF2479 E Call stack entry not found.  
CPF2508 E Cannot move messages to same or later call stack entry.  
CPF2509 E Message key &2 refers to a message that is not on the mover's call stack entry.  
CPF3C90 E Literal value cannot be changed.  
CPF3CF1 E Error code parameter not valid.  
CPF3C36 E Number of parameters, &1, entered for this API was not valid.  
CPF9872 E Program or service program &1 in library &2 ended. Reason code &3.

---

API introduced: V2R1.1

---

[Top](#) | [Message Handling APIs](#) | [APIs by category](#)

# Open List of Job Log Messages (QGYOLJBL) API

Required Parameter Group:

1	Receiver variable	Output	Char(*)
2	Length of receiver variable	Input	Binary(4)
3	List Information	Output	Char(80)
4	Number of records to return	Input	Binary(4)
5	Message selection information	Input	Char(*)
6	Size of message selection information	Input	Binary(4)
7	Error code	I/O	CHAR(*)

Default Public Authority: \*USE

Threadsafe: No

The Open List of Job Log Messages (QGYOLJBL) API lists messages from a job log. The returned messages are sorted by their sending date and time unless the message being listed is a reply message to an inquiry, a sender's copy, or a notify type message. When a reply message exists for one of these messages, the reply message is not positioned according to the sending date and time of the reply. The reply message is, instead, listed immediately following the inquiry, the sender's copy, or the notify message that it is associated with.

**Note:** The QTEMP library and the system portion of the library list could be different between the main job and the server job when the list is being built asynchronously. If this is a problem, request that the list be built synchronously.

For more information, see [Process Open List APIs](#).

## Differences between QMHLJOB and QGYOLJBL

The differences between the QGYOLJBL API and the List Job Log Messages (QMHLJOB) API follow:

- The message information for the QGYOLJBL API is returned into a receiver variable instead of a user space.
- The request handle parameter was added to the QGYOLJBL API, so that a distinct value is associated with the list. This value ensures that a request for more messages from the list (call QGYGTLE API) is sent to the correct list.
- The total records parameter was added to the QGYOLJBL API, so that the caller of the API knows the size of the list from which information can be requested.
- The records returned parameter was added to the QGYOLJBL API. Now if the request is for X records and only Y records exist (where  $Y < X$ ), the requestor is aware that it did not get the number of records requested.

## Authorities and Locks

### *User Space Lock*

\*EXCLRD

### *Job Authority*

- \*JOBCTL if the job for which messages are being listed has a different user profile from that of the job that calls the QGYOLJBL API.
- \*ALLOBJ if the job for which messages are being listed has user class of \*SECOFR.

For additional information on job authorities, see [Basic system security and planning](#).

## Required Parameter Group

### **Receiver variable**

OUTPUT; CHAR(\*)

The variable used to return the requested number of records containing message information. See [OLJL0100 Format](#) for a description of this parameter.

### **Length of receiver variable**

INPUT; BINARY(4)

The length of the receiver variable.

### **List information**

OUTPUT; CHAR(80)

The variable used to return status information about the list of job log messages that was opened. For a description of this parameter, see [Format of Open List Information](#).

### **Number of records to return**

INPUT; BINARY(4)

The number of records in the list to be put into the receiver variable. Valid values for this field are:

- 1 All records are built synchronously in the list by the main job.
- 0 All records are built asynchronously in the list by a server job.

If a positive number of records is specified, at least that many records are built synchronously (in order to return those records immediately to the caller of this API), and the remainder are built asynchronously by a server job.

### **Message selection information**

INPUT; CHAR(\*)

The information that determines the job message queue and messages to be listed. The format of this information is described in [Message Selection Information Format](#).

### **Size of message selection information**

INPUT; BINARY(4)

The size, in bytes, of the message selection information parameter. The minimum value of this parameter is 85.

**Error code**

I/O; CHAR(\*)

The structure in which to return error information. For the format of the structure, see [Error Code Parameter](#).

**OLJL0100 Format**

For detailed descriptions of each field, see the [Field Descriptions](#).

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Offset to the next entry
4	4	BINARY(4)	Offset to fields returned
8	8	BINARY(4)	Number of fields returned
12	C	BINARY(4)	Message severity
16	10	CHAR(7)	Message identifier
23	17	CHAR(2)	Message type
25	19	CHAR(4)	Message key
29	1D	CHAR(10)	Message file name
39	27	CHAR(10)	Message file library specified at send time
49	31	CHAR(7)	Date sent
56	38	CHAR(6)	Time sent
➤ 62	3E	CHAR(6)	Microseconds ⚡
➤ 68	44 ⚡	CHAR(*)	Reserved
These fields repeat for each identifier field specified.		BINARY(4)	Offset to the next field information returned
		BINARY(4)	Length of field information returned
		BINARY(4)	Identifier field
		CHAR(1)	Type of data
		CHAR(1)	Status of data
		CHAR(14)	Reserved
		BINARY(4)	Length of data
		CHAR(*)	Data
		CHAR(*)	Reserved

**Field Descriptions**

**Data.** The data returned for the specified identifier field.

**Date sent.** The date on which the message was sent, in CYYMMDD (century, year, month, and day)

format.

**Identifier field.** The field ID actually returned in the OLJL0100 format. See [Valid Field Identifiers](#) for the list of valid field identifiers.

**Length of data.** The length of the data returned for the data field, in bytes.

**Length of field information returned.** The total length of information returned for this field, in bytes.

**Message file library specified at send time.** The name of the library containing the message file as specified when the message was sent. If \*CURLIB or \*LIBL were specified for the library when the message was sent, that value is returned as the library here. For the actual library used when the message is sent, see the message file library used field identifier.

**Message file name.** The name of the message file containing the message listed.

**Message identifier.** The identifying code of the message listed. If an impromptu message is listed, this field is set to blanks.

**Message key.** The message reference key of the message listed.

**Message severity.** The severity of the message listed. Possible values are 0 through 99.

**Message type.** The type of message listed. The possible values and their meanings follow:

Value	Message Type
01	Completion
02	Diagnostic
04	Informational
05	Inquiry
06	Sender's copy
08	Request
10	Request with prompting
14	Notify, exception already handled when API is called
15	Escape, exception already handled when API is called
16	Notify, exception not handled when API is called
17	Escape, exception not handled when API is called
21	Reply, not checked for validity
22	Reply, checked for validity
23	Reply, message default used
24	Reply, system default used
25	Reply, from system reply list

» **Microseconds.** The microseconds part of the time sent.«

**Number of fields returned.** The number of identifier fields returned to the application.

**Offset to fields returned.** The offset, in bytes, from the beginning of the receiver variable to the beginning of the first repeating identified field of the OLJL0100 format.

**Offset to the next entry.** The offset, in bytes, from the beginning of the receiver variable to the beginning of the next message entry.

**Offset to the next field information returned.** The offset, in bytes, from the beginning of the receiver variable to the beginning of the next repeating identified field of the OLJL0100 format.

**Reserved.** An ignored field.

**Status of data.** The status of the data listed for this message. Possible values and their meanings follow:

*blank* The data returned is complete.

*A* The caller of the API was not authorized to view the data. This occurs when the caller of the API is not authorized to the message file or the message file library containing a stored message being listed.

*D* The data was damaged. This occurs when the message file or library specified at send time for a stored message is damaged when the API is called.

*U* The data was unavailable. This occurs when the message file or library specified at send time for a stored message is exclusively used by another process when the API is called.

*N* The data was not found. This occurs when the message file or library specified at send time for a stored message cannot be found or resolved when the API is called.

This field is applicable to the field identifiers that are retrieved from the message file for a stored message. A description of the action that occurs for specific field identifiers when the status of data field is not blank follows:

*0101* When the status of data field is not blank, the alert option field identifier contains blanks.

*0301, 0302* When the status of data field is not blank, these message field identifiers contain message text regarding the problem encountered while attempting to access the message file. Both fields have the replacement data substituted.

*0401, 0402, 0403, 0404* When the status of data field is not blank, these message help field identifiers contain message text regarding the problem encountered while attempting to access the message file. All fields have the replacement data substituted. The message help with formatting characters and message help with replacement data and formatting characters field identifiers also have the message formatting characters included.

*0501* When the status of data field is not blank, the default reply field identifier contains the system default reply.

*0801* When the status of data field is not blank, the message file library used field identifier contains blanks.

This field is also applicable to the various sending and receiving information fields when a problem is encountered while attempting to retrieve this information. This includes field identifiers 0602, 0603, 0604,



0605, 0606, 0702, 0703, 0704, 0705, and 0706. When one of these fields cannot be retrieved from the message, the status of data field is set to U and the field is set to blanks.

The status of data field is always blank for the other field identifiers.

**Time sent.** The time at which the message being listed was sent, in HHMMSS (hour, minute, and second) format.

**Type of data.** The type of data returned.

- C* The data is returned in character format.
- B* The data is returned in binary format.
- M* The data is returned in a mixed format. This value is returned for the field IDs 0606 and 0706, which contain a binary(4) value followed by an array of 10-character elements.

## Message Selection Information Format

For detailed descriptions of each field, see the [Field Descriptions](#).

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	List direction
10	A	CHAR(26)	Qualified job name
36	24	CHAR(16)	Internal job identifier
52	34	CHAR(4)	Starting message key
56	38	BINARY(4)	Maximum message length
60	3C	BINARY(4)	Maximum message help length
64	40	BINARY(4)	Offset of identifiers of fields to return
68	44	BINARY(4)	Number of fields to return
72	48	BINARY(4)	Offset to call message queue name
76	4C	BINARY(4)	Size of call message queue name
The offsets to these fields are specified in the previous offset variables.		ARRAY(*) of BINARY(4)	Identifiers of fields to return
		CHAR(*)	Call message queue name

## Field Descriptions

**Call message queue name.** The name of the call message queue from which to list messages. You must use these values:

- \* Messages from every call of the job are listed.

*\*EXT* Only messages sent to *\*EXT* of the job are listed.

**Identifiers of fields to return.** The list of the identifiers of fields to be returned. For a list of the valid field identifiers, see [Valid Field Identifiers](#).

**Internal job identifier.** The internal name for the job. The List Job (QUSLJOB) API creates this identifier. If you do not specify *\*INT* for the job name parameter, this parameter must contain blanks. With this parameter, the system can locate the job more quickly than with a job name.

**List direction.** The direction to list messages. You must use one of the following directions:

*\*NEXT* Return messages that are newer than the message specified by the starting message key parameter and the starting offset parameter. This returns messages that start with the oldest first, and the most recently sent message last.

*\*PREV* Return messages that are older than the messages specified by the starting message key parameter. This returns messages that start with the most recently sent message first, and the oldest message last.

**Maximum message help length.** The maximum length of data to be returned for field identifiers 0401, 0402, 0403, and 0404.

**Maximum message length.** The maximum length of data to be returned for field identifiers 0301 and 0302.

**Number of fields to return.** The number of fields to return (the number of entries in the identifiers of fields to return array).

To avoid heavy utilization of system resources, users of this API must be cautious of selecting too many fields to return for the messages that are listed.

**Offset to call message queue name.** The offset, in bytes, from the beginning of the message selection information parameter to the beginning of the call message queue name field.

**Offset of identifiers of fields to return.** The offset, in bytes, from the beginning of the message selection information parameter to the beginning of the identifiers of fields to return array.

**Qualified job name.** The name of the job whose messages are to be listed. The qualified job name has three parts:

*Job name* CHAR(10) A specific job name or one of the following special values:

*\** The job that this program is running in. The rest of the qualified job name parameter must be blank.

*\*INT* The internal job identifier locates the job. The rest of the qualified job name parameter must be blank.

*User name* CHAR(10) A specific user profile name, or blanks when the job name is the special value of *\** or *\*INT*.

*Job number* CHAR(6) A specific job number, or blanks when the job name is the special value of *\** or *\*INT*.

**Size of call message queue name.** The length of the call message queue name parameter, in bytes. The maximum length that can be specified is 256; the minimum length is 1.

**Starting message key.** The message key to be used to begin searching for messages to list from the job. You can use these special values for the message key:

'00000000'X The first message to be returned is the oldest message in the queue.

'FFFFFFFF'X The first message to be returned is the newest message in the queue.

If the specified key does not exist, messages are listed with the next key that does exist in the search direction specified by the list direction parameter.

If a key of a reply message is specified, the message search begins with the inquiry or notify message that the reply is associated with, not the reply message itself.

If the message specified by the starting message key exists but does not meet the selection criteria parameter and the severity criteria parameter specified, no error is returned. The search for messages to list begins from the point that is specified by the starting message key.

## Valid Field Identifiers

For a detailed description of each field, see [Field Descriptions](#).


Identifier	Type	Description
0101	CHAR(9)	Alert option
0201	CHAR(*)	Replacement data or impromptu message text
0301	CHAR(*)	Message
0302	CHAR(*)	Message with replacement data
0401	CHAR(*)	Message help
0402	CHAR(*)	Message help with replacement data
0403	CHAR(*)	Message help with formatting characters
0404	CHAR(*)	Message help with replacement data and formatting characters
0501	CHAR(*)	Default reply
0601	CHAR(26)	Qualified sender job name
0602	CHAR(1)	Sender type
0603	CHAR(*)	Sending program name
0604	CHAR(10)	Sending module name
0605	CHAR(*)	Sending procedure name
0606	BINARY(4) followed by ARRAY(*) of CHAR(10)	Number of sending statement numbers or instruction numbers available followed by an array of the sending statement numbers or instruction numbers
0702	CHAR(1)	Receiving type
0703	CHAR(10)	Receiving program name
0704	CHAR(10)	Receiving module name
0705	CHAR(*)	Receiving procedure name

0706	BINARY(4) followed by ARRAY(*) of CHAR(10)	Number of receiving statement numbers or instruction numbers available followed by an array of the receiving statement numbers or instruction numbers
0801	CHAR(10)	Message file library used
0901	CHAR(30)	Problem identification
1001	CHAR(1)	Reply status
1101	CHAR(1)	Request status
1201	BINARY(4)	Request level
1301	BINARY(4)	Coded character set identifier (CCSID) for text
1302	BINARY(4)	CCSID conversion status indicator for text
1303	BINARY(4)	Coded character set identifier (CCSID) for data
1304	BINARY(4)	CCSID conversion status indicator for data

## Field Descriptions

**Alert option.** Whether and when an SNA alert is created and sent for the message. The value is one of the following:

- \**DEFER* An alert is sent after local problem analysis.
- \**IMMED* An alert is sent immediately when the message is sent to a message queue that has the allow alerts attribute set to \*YES.
- \**NO* No alert is sent.
- \**UNATTEND* An alert is sent immediately when the system is running in unattended mode (when the value of the alert status network attribute, ALRSTS, is \*UNATTEND).

For more information about alerts, see the [Alerts Support](#)  book.

This field is set to blanks if no alert option was specified when the message was sent.

**CCSID conversion status indicator for data.** The following values may be returned:

- 0 No conversion was needed because the CCSID of the data matched the CCSID that you wanted the data converted to.
- 1 No conversion occurred because either the data was 65535, or the CCSID you wanted the data converted to was 65535.
- 2 No conversion occurred because you did not ask for any message data to be returned or the data did not contain any \*CCHAR type data.
- 3 The data was converted to the CCSID specified using the best fit conversion tables.
- 4 A conversion error occurred using the best fit conversion tables so a default conversion was attempted. This completed without error.
- 1 An error occurred on both the best fit and default conversions. The data was not converted.

**CCSID conversion status indicator for text.** The following values may be returned:

- 0 No conversion was needed because the CCSID of the text matched the CCSID you wanted the text converted to.
  - 1 No conversion occurred because either the text was 65535 or the CCSID you wanted the text converted to was 65535.
  - 2 No conversion occurred because you did not ask for any text to be returned.
  - 3 The text was converted to the CCSID specified using the best fit conversion tables.
  - 4 A conversion error occurred using the best fit conversion tables so a default conversion was attempted. This completed without error.
- 1 An error occurred on both the best fit and default conversions. The data was not converted.

**Coded character set identifier (CCSID) for data.** The coded character set identifier that the replacement data is returned in. This only applies to the part of the replacement data that corresponds to a convertible character data type (\*CCHAR). All other replacement data will not be converted before it is returned and can be considered to have a CCSID of 65535. If a conversion error occurs or if the CCSID you requested the data to be converted to is 65535, the CCSID of the data is returned. If there is no \*CCHAR replacement data, 65535 is returned. Otherwise the CCSID you wanted the data converted to is returned.

For more information about message handler and its use of CCSIDs, see [CCSIDs: Message Support](#) in the Globalization topic. For more information about the \*CCHAR field type, see the [Add Message Description \(ADDMSGD\)](#) command.

**Coded character set identifier (CCSID) for text.** The coded character set identifier that the message text is returned in. If a conversion error occurs or if the CCSID you requested the message text to be converted to is 65535, the CCSID that the message text is stored in is returned. Otherwise, the CCSID you wanted your message text converted to is returned. If you do not want the text converted before it is returned to you but you want to know the CCSID that the message text is stored in, specify 65535 on the coded character set identifier to return text and data in parameter. The CCSID that the message text is stored in is returned in the coded character set identifier for text field.

This applies to the following fields only :

- Message
- Message with replacement data
- Message help
- Message help with replacement data
- Message help with replacement data and formatting characters
- Message help with formatting characters

**Note:** This CCSID value does not apply to the replacement data that has been substituted into the text. See the coded character set identifier for data for this information.

**Default reply.** The text of the default reply when a stored message is being listed, and a default reply exists. If this is not an inquiry message, or no default reply exists, this field is not used, and the length of data field is 0.

**Message.** The text of a predefined message without replacement data substitution. If an impromptu message is listed, this field contains the impromptu message text.

**Message file library used.** The actual name of the library that contains the message file that is used to retrieve the message information. If an immediate message is listed, this field is set to blanks.

**Message help.** The message help for the message listed without formatting characters and without replacement of data. If an impromptu message is listed, this field contains the impromptu message text.

**Message help with formatting characters.** The message help for the message listed, including formatting characters.

Three format control characters can be returned within the message. They are defined in the online help for the Add Message Description (ADDMSGD) command to have these meanings:

*&N* Forces the text to a new line (column 2). If the text is longer than one line, the next lines are indented to column 4 until the end of text or another format control character is found.

*&P* Forces the text to a new line indented to column 6. If the text is longer than one line, the next lines start in column 4 until the end of text or another format control character is found.

*&B* Forces the text to a new line, starting in column 4. If the text is longer than one line, the next lines are indented to column 6 until the end of text or another format control character is found.

If an impromptu message is listed, this field contains the immediate message text.

**Message help with replacement data.** The message help for the message listed, including the replacement data. If an impromptu message is listed, this field contains the impromptu message text.

**Message help with replacement data and formatting characters.** The message help for the message listed, including the replacement data and the formatting characters. See the description of the message help with formatting characters field for an explanation of formatting characters. If an impromptu message is listed, this field contains the impromptu message text.

**Message with replacement data.** The text of a predefined message with the replacement data included. If an impromptu message is listed, this field contains the impromptu message text.

**Number of the receiving statement numbers or instruction numbers available followed by an array of the receiving statement numbers.** The number of statement numbers or instruction numbers available for the receiving program or procedure.

For original program model (OPM) programs and nonoptimized procedures, this count is 1.

For optimized procedures, this count can be greater than 1. In this case, each statement number represents a potential point at which the message could have been received. If the mapping table information has been removed from the program, this field returns a count of 0 and no statement numbers are available. The array of receiving statement numbers or instruction numbers immediately follows this field in the returned data.

**Number of sending statement numbers or instruction numbers available followed by an array of the sending statement numbers or instruction numbers.** The number of statement numbers or instruction numbers available for the sending program or procedure.

For OPM programs and nonoptimized procedures, this count is 1.

For optimized procedures, this count can be greater than 1. In this case, each statement number represents a potential point at which the message could have been sent. If the mapping table information has been removed from the program, this field returns a count of 0, and no statement numbers are available. The array of sending statement numbers or instruction numbers immediately follows this field in the returned data.

**Problem identification.** This field can be specified for the QGYOLJBL API, but it never returns any data

and the length of data field is 0.

**Qualified sender job name.** This field can be specified for the QGYOLJBL API, but it never returns any data and the length of data field is 0.

**Receiving module name.** The name of the module that contains the procedure where the message was sent. If the message was not sent to a procedure within an Integrated Language Environment (ILE) program, this field is not set and the length of data field is 0.

**Receiving procedure name.** The name of the procedure receiving the message. If the message was not sent to a procedure within an ILE program, this field is not set and the length of data field is 0. A nested procedure name has each procedure name separated by a colon. The outermost procedure name is identified first followed by the procedures it contains. The innermost procedure is identified last in the string.

**Receiving program name.** The program name, or the ILE program name that contains the procedure that the message was sent to.

**Receiving type.** The type of the receiver (whether it is a program or a procedure). Possible values and their meanings follow:

- 0 Receiver is an original program model (OPM) program
- 1 Receiver is a procedure within an ILE program and the procedure name is up to and including 256 characters in length
- 2 Receiver is a procedure within an ILE program and the procedure name is 257 or more characters in length

**Replacement data or impromptu message text.** The values for replacement variables in a predefined message, or the text of an impromptu message. If the message identifier field is not blank, this field contains message data. If the message identifier field is blank, this field contains impromptu message text.

Any pointer data in this field is marked as not valid if both of the following are true:

- The API is called by a call stack entry not running in system state.
- The system security level is 50 or above.

**Reply status.** The reply status of the message (whether it accepts a reply, and if so, whether a reply has been sent). Possible values and their meanings follow:

- A Message accepts a reply, and a reply has been sent.
- W Message accepts a reply, and a reply has not been sent. (The message is waiting for a reply.)
- N Message does not accept a reply.

**Request level.** The level of the request-processing program that received the request message. If the message being listed is not a request, this field is set to 0.

**Request status.** Information regarding the processing status of the request message. Possible values and their meanings follow:

- O This request message has been received and processed.
- C This request message is currently being processed.
- N This request message has not yet been processed.

If the message being listed is not a request, this field is set to a blank character.

**Sender type.** The type of the sender (whether it is a program or procedure). Possible values and their meanings follow:

- 0 Sender is an OPM or a System Licensed Internal Code (SLIC) program with a name that is 12 characters or less.
- 1 Sender is a procedure within an ILE program and the procedure name is up to and including 256 characters in length.
- 2 Sender is a procedure within an ILE program and the procedure name is from 257 characters up to and including 4096 characters in length.
- 3 Sender is a SLIC program with a name that is from 13 characters up to and including 256 characters in length.

**Sending module name.** The name of the module that contains the procedure sending the message. If the message was not sent by a procedure within an ILE program, this field is not set and the length of data field is 0.

**Sending procedure name.** The name of the procedure sending the message. If the message was not sent by a procedure within an ILE program, this field is not set and the length of data field is 0. A nested procedure name has each procedure name separated by a colon. The outermost procedure name is identified first followed by the procedures it contains. The innermost procedure is identified last in the string.

**Sending program name.** The sending program name or ILE program name that contains the procedure sending the message.

## Error Messages

Message ID	Error Message Text
CPF1866 E	Value &1 for number of fields to return not valid.
CPF24B4 E	Severe error while addressing parameter list.
CPF24B7 E	Value &1 for call stack entry name length not valid.
CPF240D E	Message search direction specified is not valid.
CPF240F E	Field identifier is not valid or is a duplicate of another field identifier specified.
CPF2401 E	Not authorized to library &1.
CPF2410 E	Message key not found in message queue &1.
CPF241E E	Call stack entry name is not valid.
CPF241F E	Length &1 specified for maximum message length is not valid.
CPF2441 E	Not authorized to display job log.
CPF2443 E	Job log not displayed or listed because job has ended.
CPF252F E	Length &1 specified for maximum message help length is not valid.
CPF3CF1 E	Error code parameter not valid.
CPF3C19 E	Error occurred with receiver variable specified.
CPF3C21 E	Format name &1 is not valid.



CPF3C31 E Object type &1 is not valid.  
CPF3C51 E Internal job identifier not valid.  
CPF3C52 E Internal job identifier no longer valid.  
CPF3C53 E Job &3/&2/&1 not found.  
CPF3C55 E Job &3/&2/&1 does not exist.  
CPF3C58 E Job name specified is not valid.  
CPF3C59 E Internal identifier is not blanks and job name is not \*INT.  
CPF3C90 E Literal value cannot be changed.  
CPF6565 E User profile storage limit exceeded.  
CPF8100 E All CPF81xx messages could be returned. xx is from 01 to FF.  
CPF9801 E Object &2 in library &3 not found.  
CPF9802 E Not authorized to object &2 in &3.  
CPF9803 E Cannot allocate object &2 in library &3.  
CPF9807 E One or more libraries in library list deleted.  
CPF9808 E Cannot allocate one or more libraries on library list.  
CPF9811 E Program &1 in library &2 not found.  
CPF9812 E File &1 in library &2 not found.  
CPF9814 E Device &1 not found.  
CPF9821 E Not authorized to program &1 in library &2.  
CPF9822 E Not authorized to file &1 in library &2.  
CPF9825 E Not authorized to device &1.  
CPF9831 E Cannot assign device &1.  
CPF9838 E User profile storage limit exceeded.  
GUI0002 E &2 is not valid for length of receiver variable.  
GUI0027 E &1 is not valid for number of records to return.  
GUI0044 E &1 is not valid for length of message selection information.

---

API introduced: V3R6

---

[Top](#) | [Message Handling APIs](#) | [APIs by category](#)

# Open List of Messages (QGYOLMSG) API

## Required Parameter Group:

1	Receiver variable	Output	Char(*)
2	Length of receiver variable	Input	Binary(4)
3	List information	Output	Char(80)
4	Number of records to return	Input	Binary(4)
5	Sort information	Input	Char(1)
6	Message selection information	Input	Char(*)
7	Size of message selection information	Input	Binary(4)
8	User or queue information	Input	Char(21)
9	Message queues used	Output	Char(44)
10	Error code	I/O	Char(*)

Default Public Authority: \*USE

Threadsafe: No

The Open List of Messages (QGYOLMSG) API provides information on one of the following:

- Messages for the current user
- Messages for a specific user
- Messages on one specific message queue

If messages for the current user are to be listed, the user's messages are listed from both the user message queue and the APPC device message queue. The messages are sorted by their type and sending date and time, unless the message being listed is a message needing a reply or a reply message. When a reply message exists for a message needing a reply, the reply message is not positioned by the sending date and time of the reply. The reply message is listed immediately after the message that needs a reply.

This API only lists messages from nonprogram message queues. The Open List of Messages API cannot be used to list messages sent to a job log (including \*EXT) or to list messages sent to the QHST message queue.

New messages are prevented from being added to or removed from the message queues listed during the use of the QGYOLMSG API when it calls the List Nonprogram Messages (QMHLSTM) API.

Message file overrides are not honored by the server job when the list is built asynchronously by the server job.

**Note:** The QTEMP library and the system portion of the library list could be different between the main job and the server job when the list is being built asynchronously. If this is a problem, then request that the list be built synchronously.

For more information, see [Process Open List APIs](#).

## Differences between the QMHLSTM and QGYOLMSG APIs

The differences between the QGYOLMSG API and the List Nonprogram Messages (QMHLSTM) API follow:

- The message information for the QGYOLMSG API is returned into a receiver variable instead of a user space.
- The request handle parameter was added to the QGYOLMSG API so that a distinct value is associated with the list. This value ensures that a request for more messages from the list (Get List Entries (QGYGTLE) API) is sent to the correct list.
- The total records parameter was added to the QGYOLMSG API so that the caller of the API knows the size of the list from which information can be requested.
- The records returned parameter was added to the QGYOLMSG API. If the request is for X records and only Y records exist (where  $Y > X$ ), the requestor is aware that it did not get the number of records requested.

## Difference between the QSYRUSRI and QGYOLMSG APIs

The difference between the QGYOLMSG API and the Retrieve User Information (QSYRUSRI) API is:

- The user name parameter was added to the QGYOLMSG API so that the message queue name for that user can be retrieved by calling the QSYRUSRI API.

## Authorities and Locks

*Message Queue*

\*USE

*Message Queue Library*

\*EXECUTE

## Required Parameter Group

### Receiver variable

OUTPUT; CHAR(\*)

The receiver variable that receives the information requested. You can specify the size of the area to be smaller than the format requested as long as you specify the length parameter correctly. As a result, the API returns only the data that the area can hold.

The entries returned are in the LSTM0100 format. See [LSTM0100 Format](#) for information about the format.

### Length of receiver variable

INPUT; BINARY(4)

The length of the receiver variable provided. The length of receiver variable parameter may be specified up to the size of the receiver variable specified in the user program. If the length of receiver variable parameter specified is larger than the allocated size of the receiver variable

specified in the user program, the results are not predictable.

### **List information**

OUTPUT; CHAR(80)

The variable used to return status information about the list of messages that was opened. For a description of the layout of this parameter, see [Format of Open List Information](#).

### **Number of records to return**

INPUT; BINARY(4)

The number of records in the list to put into the receiver variable after the sorting has been done. The valid values for this field are as follows:

- 1 All records are built synchronously in the list by the main job.
- 0 All records are built asynchronously in the list by a server job.

If a positive number of records is specified, at least that many records are built synchronously (in order to return those records immediately to the caller of this API), and the remainder are built asynchronously by a server job.

### **Sort information**

INPUT; CHAR(1)

Indicates whether the list should be sorted by type if the selection criteria contains \*ALL. Possible values are as follows:

- 0 No sort
- 1 Sort by the following message types in the listed order:
  - \*MNR Unanswered, inquiry-type messages
  - \*SCNR Unanswered, sender's-copy type of messages
  - \*MNNR Informational, completion, diagnostic, request, notify, escape, request, answered inquiry, and answered, sender's-copy type messages.

### **Message selection information**

INPUT; CHAR(\*)

The information that determines the message queues and messages to be listed. The format of this information is described in [Message Selection Information Format](#).

### **Size of message selection information**

INPUT; BINARY(4)

The size, in bytes, of the message selection information parameter. The minimum value required for this parameter is 62.

### **User or queue information**

INPUT; CHAR(21)

The name that is given (the user name or the qualified user message queue name) and the value of that name. This parameter consists of the following parts:

#### *User or queue indicator*

CHAR(1)

Whether a value is given in the user name part of this parameter, or in the qualified-user message queue name part of this parameter. One of the following values must be specified:

- 0 The user name is given
- 1 The user message queue name is given

Depending on the value of this parameter, either the user name or the qualified-user message queue name contains a nonblank value.

#### *User or queue name*

CHAR(20)

If the user or queue indicator value is 0, the first 10 characters specify the user name for which information is returned; the last 10 characters must be blank. You can specify the following special value in the first 10 characters:

\**CURRENT* The information for the user running currently is returned.

This value is used to get the message queue name for the user.

If the user or queue indicator value is 1, this field contains the name of a message queue. The first 10 characters contain the name of the message queue, and the second 10 characters contain the name of the library where the message queue is located.

### **Message queues used**

OUTPUT; CHAR(44)

The name given (the user name or the qualified user message queue name) and the value of that name. This parameter consists of two parts:

#### *Number of queues used*

BINARY(4)

The number of message queues used to build the list. This value is 2 if both of the following list items are true:

- \**CURRENT* is specified for the user name in the user or queue information parameter
- Messages from both a user queue and a workstation message queue are included in the list.

Otherwise, this value is set to 1.

#### *Message queue names*

ARRAY(2) OF CHAR(20)

An array of qualified message queue names, which contain the names of the queues used to build the list. For each array element, the first 10 characters represent the message queue name, and the last 10 characters hold the library name.

### **Error code**

I/O; CHAR(\*)

The structure in which to return error information. For the format of the structure, see [Error Code](#)

[Parameter.](#)

## LSTM0100 Format

The following table shows the information returned in the receiver variable for the LSTM0100 format. The offsets listed are from the beginning of the receiver variable. For a detailed description of each field, see [Field Descriptions](#).

The structure defined by this format is repeated for each message entry returned.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Offset to the next entry
4	4	BINARY(4)	Offset to fields returned
8	8	BINARY(4)	Number of fields returned
12	C	BINARY(4)	Message severity
16	10	CHAR(7)	Message identifier
23	17	CHAR(2)	Message type
25	19	CHAR(4)	Message key
29	1D	CHAR(10)	Message file name
39	27	CHAR(10)	Message file library specified at send time
49	31	CHAR(10)	Message queue
59	3B	CHAR(10)	Message queue library used
69	45	CHAR(7)	Date sent
76	4C	CHAR(6)	Time sent
» 82	52	CHAR(6)	Microseconds «
» 88	58 «	CHAR(*)	Reserved
These fields repeat for each identifier field specified.		BINARY(4)	Offset to the next field information returned
		BINARY(4)	Length of field information returned
		BINARY(4)	Identifier field
		CHAR(1)	Type of data
		CHAR(1)	Status of data
		CHAR(14)	Reserved
		BINARY(4)	Length of data
		CHAR(*)	Data
		CHAR(*)	Reserved

## Field Descriptions

**Data.** The data returned for the specified identifier field.

**Date sent.** The date on which the message was sent, in CYYMMDD (century, year, month, and day)

format.

**Identifier field.** The field returned. See [Valid Field Identifiers](#) for the list of valid field identifiers.

**Length of data.** The length of the data returned for the data field, in bytes.

**Length of field information returned.** The total length of information returned for this field, in bytes.

**Message file library specified at send time.** The name of the library containing the message file as specified when the message was sent. If \*CURLIB or \*LIBL was specified for the library when the message was sent, that value is returned as the library here. For the actual library used when the message is sent, see the message file library used field identifier.

**Message file name.** The name of the message file containing the message listed.

**Message identifier.** The identifying code of the message listed. If an immediate message is listed, this field is set to blanks.

**Message key.** The key of the message listed.

**Message queue.** The name of the message queue where the message was listed.

**Message queue library used.** The actual library that contains the message queue.

**Message severity.** The severity of the message listed. Possible values are 0 through 99.

**Message type.** The type of message listed. The possible values and their meanings follow:

Value	Message Type
01	Completion
02	Diagnostic
04	Informational
05	Inquiry
06	Sender's copy
08	Request
10	Request with prompting
14	Notify, exception already handled when API is called
15	Escape, exception already handled when API is called
16	Notify, exception not handled when API is called
17	Escape, exception not handled when API is called
21	Reply, not checked for validity
22	Reply, checked for validity
23	Reply, message default used
24	Reply, system default used

» **Microseconds.** The microseconds part of the time sent.«

**Number of fields returned.** The number of identifier fields returned to the application.

**Offset to fields returned.** The offset, in bytes, from the beginning of the receiver variable to the beginning of the first repeating identified field of the LSTM0100 format.

**Offset to the next entry.** The offset, in bytes, from the beginning of the receiver variable to the beginning of the next message entry.

**Offset to the next field information returned.** The offset, in bytes, from the beginning of the receiver variable to the beginning of the next repeating identified field of the LSTM0100 format.

**Reserved.** An ignored field.

**Status of data.** The status of the data listed for this message. Possible values and their meanings follow:

- blank* The data returned is complete.
- A* The caller of the API was not authorized to view the data. This occurs when the caller of the API is not authorized to the message file or message file library containing a stored message being listed.
- D* The data was damaged. This occurs when the message file or library specified at send time for a stored message is damaged when the API is called.
- U* The data was unavailable. This occurs when the message file or library specified at send time for a stored message is exclusively used by another process when the API is called.
- N* The data was not found. This occurs when the message file or library specified at send time for a stored message cannot be found or resolved when the API is called.

This field is applicable to the field identifiers that are retrieved from the message file for a stored message. A description of the action that occurs for specific field identifiers when the status of data field is not blank follows:

- 0101* When the status of data field is not blank, the alert option field identifier contains blanks.
- 0301, 0302* When the status of data field is not blank, these message field identifiers contain message text about the problem encountered while attempting to access the message file. Both fields have the replacement data substituted.
- 0401, 0402, 0403, 0404* When the status of data field is not blank, these message help field identifiers contain the text of the message regarding the problem encountered while attempting to access the message file. All fields have the replacement data substituted. The message help with formatting characters and message help with replacement data and formatting characters field identifiers also have the message formatting characters included.
- 0501* When the status of data field is not blank, the default reply field identifier contains the system default reply.
- 0801* When the status of data field is not blank, the message file library used field identifier contains blanks.



This field is also applicable to the various sending information fields (identifiers 0601, 0603) when a problem is encountered while attempting to retrieve this information. When one of these fields cannot be retrieved from the message:

- The status of data field is set to N.
- The length of data field is set to 0.

The status of data field is always blank for the other field identifiers. The length of data field is zero.

**Time sent.** The time at which the message being listed was sent, in HHMMSS (hour, minute, and second) format.

**Type of data.** The type of data returned.

*C* The data is returned in character format.

*B* The data is returned in binary format.

*M* The data is returned in a mixed format. This value is returned for the field IDs 0606 and 0706.

## Message Selection Information Format

The organization of the message selection information parameter is shown below. For detailed descriptions of each field, see [Field Descriptions](#).

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	List direction
10	0A	CHAR(2)	Reserved
12	0C	BINARY(4)	Severity criteria
16	10	BINARY(4)	Maximum message length
20	14	BINARY(4)	Maximum message help length
24	18	BINARY(4)	Offset of selection criteria
28	1C	BINARY(4)	Number of selection criteria
32	20	BINARY(4)	Offset of starting message keys
36	24	BINARY(4)	Offset of identifiers of fields to return
40	28	BINARY(4)	Number of fields to return
The offsets to these fields are specified in the previous offset variables.		ARRAY(*) of CHAR(10)	Selection criteria
		ARRAY(*) of CHAR(4)	Starting message keys
		ARRAY(*) of BINARY(4)	Identifiers of fields to return

## Field Descriptions

**Identifiers of fields to return.** The list of the identifiers of fields to be returned in the receiver variable. For a list of the valid field identifiers, see [Valid Field Identifiers](#).

The reply status field must always be requested.

**List direction.** The direction to list messages. You must use one of these directions:

- \*NEXT* Return messages that are newer than the messages specified by the starting message keys parameter. *\*NEXT* lists the messages starting with the oldest message first in the list, and the most recently sent message last in the list.
- \*PREV* Return messages that are older than the messages specified by the starting message keys parameter. *\*PREV* lists the messages starting with the most recently sent message first in the list, and the oldest message last in the list.

If multiple message queues are to be listed, messages are intermixed and sorted by date and time in the specified manner.

**Maximum message help length.** The maximum length to be returned for fields 0401, 0402, 0403, and 0404.

**Maximum message length.** The maximum length to be returned for fields 0301 and 0302.

**Number of fields to return.** The number of fields to return. The reply status field must always be requested.

**Number of selection criteria.** The number of selection criteria specified. You must specify at least 1 and no more than 3.

**Offset of identifiers of fields to return.** The offset, in bytes, from the beginning of the message selection information parameter to the beginning of the identifiers of fields to return array.

**Offset of starting message keys.** The offset, in bytes, from the beginning of the message selection information parameter to the beginning of the starting message keys array.

**Offset to selection criteria.** The offset, in bytes, from the beginning of the message selection information parameter to the beginning of the selection criteria array.

**Reserved.** An ignored field.

**Selection criteria.** The type of messages to be listed. You must use at least one of these values and no more than three for this parameter. If more than one value is given and one of them is *\*ALL*, an error is given. If more than one value is given, the messages are grouped by type. Specifying all three values is equivalent to specifying *\*ALL* and requesting sorting.

- \*ALL* All messages are to be listed.
- \*MNR* Only messages that need a reply are listed. This includes only unanswered, inquiry-type messages.
- \*SCNR* Only the sender's copy messages that need a reply are listed. This includes only unanswered, sender's-copy type of messages.

*\*MNNR* Only messages that do not need a reply are listed. This includes informational, completion, diagnostic, request, notify, escape, request, answered inquiry, and answered, sender's-copy type of messages.

**Severity criteria.** The minimum severity of a message to be included in the list. The value must be in the range 0 to 99. To list all messages, specify a severity criteria of 0.

**Starting message keys.** The message key used to begin searching for messages to list from the corresponding entry in the message queue. If *\*CURRENT* is specified for the user name in the user or queue information parameter, two values must be specified. The first value is the starting message key for the current user's user message queue. The second value is the starting message key for the current user's workstation message queue. If either a specific user name or a specific message queue name is specified in the user or queue information parameter, only one message key is used.

You can use these special values for the message keys:

'00000000'X The first message to be returned is the oldest message in the queue.

'FFFFFFFF'X The first message to be returned is the newest message in the queue.

If a value other than '00000000'X or 'FFFFFFFF'X is specified and a message with that key does not exist, an error is returned. If a key of a reply message is specified, the message search begins with the inquiry or sender's copy message that the reply is associated with, not the reply message itself.

If the message specified by the starting message key exists but does not meet the selection criteria and severity criteria parameters specified, no error is returned and the search for messages to list begins from the message specified by the starting message key.

## Valid Field Identifiers

For a detailed description of each field, see [Field Descriptions](#).


Identifier	Type	Description
0101	CHAR(9)	Alert option
0201	CHAR(*)	Replacement data or immediate message text
0301	CHAR(*)	Message
0302	CHAR(*)	Message with replacement data
0401	CHAR(*)	Message help
0402	CHAR(*)	Message help with replacement data
0403	CHAR(*)	Message help with formatting characters
0404	CHAR(*)	Message help with replacement data and formatting characters
0501	CHAR(*)	Default reply
0601	CHAR(26)	Qualified sender job name
0602	CHAR(1)	Sender type
0603	CHAR(*)	Sending program name
0604	CHAR(10)	Sending module name
0605	CHAR(*)	Sending procedure name

0606	BINARY(4) followed by ARRAY(*) of CHAR(10)	Number of sending statement numbers or instruction numbers available followed by an array of the sending statement numbers or instruction numbers
0702	CHAR(1)	Receiving type
0703	CHAR(10)	Receiving program name
0704	CHAR(10)	Receiving module name
0705	CHAR(*)	Receiving procedure name
0706	BINARY(4) followed by ARRAY(*) of CHAR(10)	Number of receiving statement numbers or instruction numbers available followed by an array of the receiving statement numbers or instruction numbers
0801	CHAR(10)	Message file library used
0901	CHAR(30)	Problem identification
1001	CHAR(1)	Reply status
1002	CHAR(1)	Critical break message status
1101	CHAR(1)	Request status
1201	BINARY(4)	Request level
1301	BINARY(4)	Coded character set identifier (CCSID) for text
1302	BINARY(4)	CCSID conversion status indicator for text
1303	BINARY(4)	Coded character set identifier (CCSID) for data
1304	BINARY(4)	CCSID conversion status indicator for data

## Field Descriptions

**Alert option.** Whether and when an SNA alert is created and sent for the message. If a message is listed, the value is one of the following:

- \**DEFER*      An alert is sent after local problem analysis.
- \**IMMED*      An alert is sent immediately when the message is sent to a message queue that has the allow alerts attribute set to \*YES.
- \**NO*            No alert is sent.
- \**UNATTEND*    An alert is sent immediately when the system is running in unattended mode (when the value of the alert status network attribute, ALRSTS, is \*UNATTEND).

For more information about alerts, see the [Alerts Support](#)  book.

This field is set to blanks if no alert option was specified when the message was sent.

**CCSID conversion status indicator for data.** The following values may be returned:

- 0    No conversion was needed because the CCSID of the data matched the CCSID that you wanted the data converted to.

- 1 No conversion occurred because either the data was 65535, or the CCSID you wanted the data converted to was 65535.
  - 2 No conversion occurred because you did not ask for any message data to be returned or the data did not contain any \*CCHAR type data.
  - 3 The data was converted to the CCSID specified using the best fit conversion tables.
  - 4 A conversion error occurred using the best fit conversion tables so a default conversion was attempted. This completed without error.
- 1 An error occurred on both the best fit and default conversions. The data was not converted.

**CCSID conversion status indicator for text.** The following values may be returned:

- 0 No conversion was needed because the CCSID of the text matched the CCSID you wanted the text converted to.
  - 1 No conversion occurred because either the text was 65535 or the CCSID you wanted the text converted to was 65535.
  - 2 No conversion occurred because you did not ask for any text to be returned.
  - 3 The text was converted to the CCSID specified using the best fit conversion tables.
  - 4 A conversion error occurred using the best fit conversion tables so a default conversion was attempted. This completed without error.
- 1 An error occurred on both the best fit and default conversions. The data was not converted.

**Coded character set identifier (CCSID) for data.** The coded character set identifier that the replacement data is returned in. This only applies to the part of the replacement data that corresponds to a convertible character data type (\*CCHAR). All other replacement data will not be converted before it is returned and can be considered to have a CCSID of 65535. If a conversion error occurs or if the CCSID you requested the data to be converted to is 65535, the CCSID of the data is returned. If there is no \*CCHAR replacement data, 65535 is returned. Otherwise the CCSID you wanted the data converted to is returned.

For more information about message handler and its use of CCSIDs, see [CCSIDs: Message Support](#) in the Globalization topic. For more information about the \*CCHAR field type, see the [Add Message Description](#) (ADDMSGD) command.

**Coded character set identifier (CCSID) for text.** The coded character set identifier that the message text is returned in. If a conversion error occurs or if the CCSID you requested the message text to be converted to is 65535, the CCSID that the message text is stored in is returned. Otherwise the CCSID you wanted your message text converted to is returned. If you do not want the text converted before it is returned to you but you do want to know the CCSID that the message text is stored in, specify 65535 on the coded character set identifier to return text and data in parameter. The CCSID that the message text is stored in is returned in the coded character set identifier for text field.

This applies to the following fields only:

- Message
- Message with replacement data
- Message help
- Message help with replacement data
- Message help with replacement data and formatting characters

- Message help with formatting characters

**Note:** This CCSID value does not apply to the replacement data that has been substituted into the text. See the coded character set identifier (CCSID) for data field for this information.

**Critical break message status.** Whether the message was sent by the operating system as a critical break message. The following values are returned:

- 1* The message was sent as a critical break message.
- 0* The message was not sent as a critical break message.

**Default reply.** The text of the default reply when a stored message is being listed, and a default reply exists. If this is not an inquiry message, or no default reply exists, this field is not used, and the length of data field is 0.

**Message.** The text of a predefined message without replacement data substitution. If an immediate message is listed, this field contains the immediate message text.

**Message file library used.** The actual name of the library that contains the message file that is used to retrieve the message information. If an immediate message is listed, this field is set to blanks.

**Message help.** The message help for the message listed without formatting characters and without replacement of data. If an immediate message is listed, this field contains the immediate message text.

**Message help with formatting characters.** The message help for the message listed, including formatting characters.

Three format control characters can be returned within the message. They are defined in the online help of the Add Message Description (ADDMSGD) command to have these meanings:

- &N* Forces the text to a new line (column 2). If the text is longer than one line, the next lines are indented to column 4 until the end of text or another format control character is found.
- &P* Forces the text to a new line indented to column 6. If the text is longer than one line, the next lines start in column 4 until the end of text or another format control character is found.
- &B* Forces the text to a new line, starting in column 4. If the text is longer than one line, the next lines are indented to column 6 until the end of text or another format control character is found.

If an immediate message is listed, this field contains the immediate message text.

**Message help with replacement data.** The message help for the message listed, including the replacement data. If an immediate message is listed, this field contains the immediate message text.

**Message help with replacement data and formatting characters.** The message help for the message listed, including the replacement data and the formatting characters. See the description of the message help with formatting characters field for an explanation of formatting characters. If an immediate message is listed, this field contains the immediate message text.

**Message with replacement data.** The text of a predefined message with the replacement data included. If an immediate message is listed, this field contains the immediate message text.

**Number of receiving statement numbers or instruction numbers available followed by an array of the receiving statement numbers or instruction numbers.** This field can be specified for the QGYOLMSG API, but it never returns any data and the length of data field is 0.

**Number of sending statement numbers or instruction numbers available followed by an array of the sending statement numbers or instruction numbers.** This field can be specified for the QGYOLMSG API, but it never returns any data and the length of data field is 0.

**Problem identification.** The number the system generates to identify a problem if problem analysis can be run for the message being listed. The problem identification is in the following format:

*CHAR 1-10* Problem ID number. The number the system generates to identify the problem.

*CHAR 11-30* Origin system in the format *network-ID.control-point-name*.

If problem analysis cannot be run, and this field is specified, the length of data field is 0.

**Qualified sender job name.** The name of the job that sent the message. The job name has three parts:

*CHAR 1-10* The specific job name.

*CHAR 11-20* The specific user profile name.

*CHAR 21-26* The specific job number.

**Receiving module name.** This field can be specified for the QGYOLMSG API, but it never returns any data and the length of data field is 0.

**Receiving procedure name.** This field can be specified for the QGYOLMSG API, but it never returns any data and the length of data field is 0.

**Receiving program name.** This field can be specified for the QGYOLMSG API, but it never returns any data and the length of data field is 0.

**Receiving type.** This field can be specified for the QGYOLMSG API, but it never returns any data and the length of data field is 0.

**Replacement data or immediate message text.** The values for replacement variables in a predefined message, or the text of an immediate message. If the message identifier field is not blank, this field contains message data. If the message identifier field is blank, this field contains immediate message text.

Any pointer data in this field is marked as not valid if both of the following are true:

- The API is called by a call stack entry not running in system state.
- The system security level is 50 or above.

**Reply status.** The reply status of the message (whether it accepts a reply, and if so, whether a reply has been sent). This field must always be requested. Possible values and their meanings follow:

*A* Message accepts a reply, and a reply has been sent.

*W* Message accepts a reply, and a reply has not been sent. (The message is waiting for a reply.)

*N* Message does not accept a reply.

**Request level.** This field can be specified for the QGYOLMSG API, but it never returns any data and the length of data field is 0.

**Request status.** This field can be specified for the QGYOLMSG API, but it never returns any data and the length of data field is 0.

**Sender type.** This field can be specified for the QGYOLMSG API, but it never returns any data and the length of data field is 0.

**Sending module name.** This field can be specified for the QGYOLMSG API, but it never returns any data and the length of data field is 0.

**Sending procedure name.** This field can be specified for the QGYOLMSG API, but it never returns any data and the length of data field is 0.

**Sending program name.** The sending program name or ILE program name that contains the procedure sending the message.

## Error Messages

Message ID	Error Message Text
CPF2203 E	User profile &1 not correct.
CPF2204 E	User profile &1 not found.
CPF2207 E	Not authorized to use object &1 in library &3 type *&2.
CPF2213 E	Not able to allocate user profile &1.
CPF2217 E	Not authorized to user profile &1.
CPF2225 E	Not able to allocate internal system object.
CPF24B4 E	Severe error while addressing parameter list.
CPF3C19 E	Error occurred with receiver variable specified.
CPF3C90 E	Literal value cannot be changed.
CPF3CF1 E	Error code parameter not valid.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.
GUI0002 E	&2 is not valid for length of receiver variable.
GUI0017 E	&1 is not valid for user or queue indicator.
GUI0027 E	&1 is not valid for number of records to return.
GUI0040 E	User name cannot be blank.
GUI0043 E	&1 is not valid for sort indicator parameter.
GUI0044 E	&1 is not valid for length of message selection information.
GUI0045 E	&1 is not valid for number of selection criteria.
GUI0046 E	*ALL cannot be specified with another value in message selection criteria.
GUI004A E	Key 1001 for reply status field is not specified.
GUI004B E	Message queue associated with user name &1 does not exist.



GUI004C E      Message queue name and library must be specified.

---

API introduced: V3R6

---

[Top](#) | [Message Handling APIs](#) | [APIs by category](#)

# Promote Message (QMHPRMM) API

## Required Parameter Group:

1	Invocation pointer	Input	Pointer
2	Call stack counter	Input	Binary(4)
3	Message key	Input	Char(4)
4	Message identifier	Input	Char(7)
5	Qualified message file name	Input	Char(20)
6	Message data	Input	Char(*)
7	Length of message data	Input	Binary(4)
8	Message type	Input	Char(10)
9	Message severity	Input	Binary(4)
10	Log option	Input	Char(1)
11	Priority	Input	Char(10)
12	New message key	Output	Char(4)
13	Error code	I/O	Char(*)

Default Public Authority: \*USE

Threadsafe: Yes

The Promote Message (QMHPRMM) API promotes an escape or status message that has been sent to a call stack entry. The API replaces the message with a new escape or status message to the same call stack entry. An escape message may be promoted to another escape message or to a status message. A status message may be promoted to an escape message or to another status message. The stored severity of the new message may be overridden. The new message may be added to the job log, and the initial priority of the new message may be changed.

The message to be promoted must be an active escape message or status message, existing on the message queue of the specified call stack entry. This API causes the existing message to be handled. The source call stack entry of the new message is the same as the source call stack entry of the promoted message. The message key of the new message is returned to the caller of this API. If the new message is not monitored by the target call stack entry, an error may be returned to the caller of this API and the new message key may not be returned.

In a multithreaded job only messages on a call message queue in the thread that calls this API can be promoted. This API cannot promote messages on call message queues in other threads.

## Authorities and Locks

### *Message File Authority*

\*USE

### *Message File Library Authority*

\*EXECUTE

# Required Parameter Group

## Invocation pointer

INPUT; POINTER

The invocation pointer to the call stack entry receiving the escape message or status message that is to be promoted. When the call stack counter parameter is other than 0, the invocation pointer points to the call stack entry from which to start counting earlier in the call stack. This locates the call stack entry that received the escape message or status message being promoted. The call stack entry you specify must be in the call stack. A null invocation pointer may be specified. If this pointer is not set, the call stack entry that called the QMHPRMM API is used.

## Call stack counter

INPUT; BINARY(4)

A number identifying the location in the call stack of the call stack entry receiving the escape or status message that is to be promoted. The number is relative to the call stack entry specified in the invocation pointer parameter. It indicates how many calls up the call stack the call stack entry is from the one specified in the invocation pointer parameter. Valid values follow:

- |                                |   |
|--------------------------------|---|
| <i>0</i>                       | Promotes the message that was sent to the call stack entry specified in the invocation pointer parameter.   |
| <i>1</i>                       | Promotes the message sent to the call stack entry that called the call stack entry specified in the invocation pointer parameter.                   |
| <i>n (any positive number)</i> | Promotes the message sent to the nth call stack entry earlier in the stack from the call stack entry specified in the invocation pointer parameter. |

You can use any positive number that points to an actual call stack entry in the call stack.

## Message key

INPUT; CHAR(4)

The message key of the existing message that is to be promoted.

## Message identifier

INPUT; CHAR(7)

The identifying code for the new predefined message that replaces the message being promoted.

## Qualified message file name

INPUT; CHAR(20)

The name of the message file and the library in which the new predefined message resides. The first 10 characters specify the file name, and the second 10 characters specify the library. You must specify both the message file name and the library name. You can use these special values for the library name:

- \**CURLIB* The job's current library
- \**LIBL* The library list

### Message data

INPUT; CHAR(\*)

The data to insert in the predefined message's substitution variables.

If there is no data to insert, a value of 0 should be specified for the length of message data parameter.

If this parameter contains pointer data, each pointer must start on a 16-byte boundary to keep the data accurate.

### Length of message data

INPUT; BINARY(4)

The length of the message data, in bytes. Valid values are 0 through 32767.

### Message type

INPUT; CHAR(10)

The type of the new message.

You must specify one of these values:

*\*ESCAPE* Escape message

*\*STATUS* Status message


You may promote an escape message to another escape message or to a status message. You may promote a status message to an escape message or to another status message.

### Message severity

INPUT; BINARY(4)

A number identifying the message severity that overrides the severity that is stored in the message description for the message. Valid values follow:

*-1* The message severity is not to be overridden.

*0-99* The severity for the new message that replaces the message being promoted. For a list of the severity codes and their meanings, see the [CL Programming](#)  book.

### Log option

INPUT; CHAR(1)

A value that indicates whether the new message is to appear in the job log. Valid values follow:

*0* The new message is not to appear in the job log.

*1* The new message is to appear in the job log. However, if the exception monitor for the new message indicates that it is not to be enqueued, it does not appear in the job log. This parameter value is only valid when the message type parameter is *\*ESCAPE*. If the message type parameter is *\*STATUS*, an error is returned to the caller of this API.

## Priority

INPUT; CHAR(10)

The initial priority of the new message.

You must specify one of these values:

- \**CONTINUE* Continue. The initial priority of the new message is the same as the current priority of the message that is being promoted.
- \**LERETRY* Retry. The initial priority of the new message is 135, so that the first condition handler that will get control is the ILE condition manager.
- \**LEDFT* LE/400 Default. The initial priority of the new message is 225, which indicates LE default action. This causes the exception to be moved up to the next call stack entry where processing continues at the first handler condition.

## New message key

OUTPUT; CHAR(4)

The message key of the new message that replaces the promoted message.

## Error code

I/O; CHAR(\*)

The structure in which to return error information. For the format of the structure, see [Error Code Parameter](#).

## Error Messages

### Message ID Error Message Text

- CPF24A3 E Value for call stack counter parameter not valid.
- CPF24B3 E Message type &1 not valid.
- CPF24B4 E Severe error while addressing parameter list.
- CPF24B6 E Length of &1, not valid for message text or data.
- CPF2401 E Not authorized to library &1.
- CPF2407 E Message file &1 in &2 not found.
- CPF2410 E Message key not found in message queue &1.
- CPF2411 E Not authorized to message file &1 in &2.
- CPF2419 E Message identifier &1 not found in message file &2 in &3.
- CPF243A E Invocation pointer parameter not valid.
- CPF243B E Message severity &1 not valid.

CPF243C E Log option &1 not valid.  
CPF243D E Priority &1 not valid.  
CPF243E E Message is not an active exception.  
CPF243F E Cannot promote message of type other than \*ESCAPE or \*STATUS.  
CPF2499 E Message identifier &1 not allowed.  
CPF2531 E Message file &1 in &2 damaged for &3.  
CPF2548 E Damage to message file &1 in &2.  
CPF3C90 E Literal value cannot be changed.  
CPF3CF1 E Error code parameter not valid.  
CPF8100 E All CPF81xx messages could be returned. xx is from 01 to FF.  
CPF9830 E Cannot assign library &1.  
CPF9872 E Program or service program &1 in library &2 ended. Reason code &3.

---

API introduced: V2R3

---

[Top](#) | [Message Handling APIs](#) | [APIs by category](#)

# Receive Nonprogram Message (QMHRRCVM) API

## Required Parameter Group:

1	Message information	Output	Char(*)
2	Length of message information	Input	Binary(4)
3	Format name	Input	Char(8)
4	Qualified message queue name	Input	Char(20)
5	Message type	Input	Char(10)
6	Message key	Input	Char(4)
7	Wait time	Input	Binary(4)
8	Message action	Input	Char(10)
9	Error code	I/O	Char(*)

## Optional Parameter Group 1:

10	Coded character set identifier	Input	Binary(4)
----	--------------------------------	-------	-----------

## Optional Parameter Group 2:

11	Allow default reply rejection	Input	Char(10)
----	-------------------------------	-------	----------

Default Public Authority: \*USE

Threadsafe: Yes

The Receive Nonprogram Message (QMHRRCVM) API receives a message from a nonprogram message queue. To receive a message from a program message queue or from the external message queue, see [Receive Program Message](#) API.

## Authorities and Locks

### Message Queue Authority

\*USE and \*DLT if the message action parameter specifies \*REMOVE; \*USE for other message actions.

### Message Queue Library Authority

\*EXECUTE

### Message Queue Lock

If a wait time is specified and the API must wait, the message queue is allocated to the job calling the API for the duration of the wait.

Other jobs on the system can only send messages to the queue during the wait time. Other jobs cannot place the queue in \*BREAK or \*NOTIFY mode, remove messages from the queue, or do an additional receive message with wait operation during this time.

## Required Parameter Group

### Message information

OUTPUT; CHAR(\*)

The variable that receives the information returned, in the format specified by the format name parameter, of the length specified by the length of message information parameter.

### Length of message information

INPUT; BINARY(4)

The size of the area to contain the message information, in bytes. The minimum size is 8.

This parameter must specify the size of the variable you use for the message information parameter. If this parameter specifies a longer size, other parts of storage could be overwritten when the API returns the information.

The API returns as much information as it can fit in this length. If the available message information is longer, it is truncated. If the available message information is shorter, the unused output area is unchanged; whatever is already stored in that space remains there.

To determine how much information the API actually returns in response to this call, see the bytes returned field in the RCVMO100 or RCVMO200 output. To determine how much information the API could return if space were available, see the bytes available field.

### Format name

INPUT; CHAR(8)

The format to use for the message information. Specify one of these format names:

*RCVM0100* Brief message information. For details, see [RCVM0100 Format](#).

*RCVM0200* All message information. For details, see [RCVM0200 Format](#).

### Qualified message queue name

INPUT; CHAR(20)

The name of the message queue from which to receive the message, and the library in which it resides. The first 10 characters specify the message queue, and the second 10 characters specify the library. You can use these special values for the library name:

\**CURLIB* The job's current library

\**LIBL* The library list

You cannot receive messages from the history log, QHST.

### Message type

INPUT; CHAR(10)



The type of the message being received. The message type and message key parameters work together. Depending on the message type, the key can be required, optional, or disallowed. For a list of valid message types and information about how they work with the message key parameter, see [Message Types and Message Keys](#).

### Message key

INPUT; CHAR(4)

The key to the message being received. The key is assigned by the command or API that sends the message.

What you can use for this parameter depends on what you use for the message type. For details, see [Message Types and Message Keys](#).

If you are not receiving messages by key, use blanks for this parameter.

If you specify a key and the message queue does not contain a message with that key, an error is returned.

If you specify a key and the message queue does contain a message with that key, the API might or might not return a message. It never returns an error in this case. Whether or not the API returns a message depends on the value of the message type parameter. For example, if you specify the message type \*PRV and there is no message before the message with the key, the API does not return a message. Because the key you specified is valid, the API does not return an error either.



You can receive the reply to an inquiry message through the key to the sender's copy of the inquiry. If the reply is not available, no message is returned, and the API does not return an error.

When the message type is the special value \*NEXT, you can use the special value \*TOP for the message key. \*TOP returns the message at the top of the queue.

When the message type is the special value \*NEXT or \*PRV, you can use hexadecimal zeros for the message key for the first receive operation.

### Wait time

INPUT; BINARY(4)

The length of time  in seconds  to wait for the message to arrive in the queue so it can be received.

The system ignores this parameter when you specify both a message key and a message type other than reply (\*RPY). The parameter is used in only two cases:

1. The message type is reply (\*RPY), and the message key parameter specifies the key to the sender's copy of the message.
2. The message type is anything except reply (\*RPY), and the message key parameter is blank. In this case, the QMHRCVM API does not use the wait time parameter immediately. First, the API checks the queue for the first message of that type that has not been received. If no such message is found, the API then waits the specified length of time for a message to arrive.

Valid values follow:

0	Do not wait for the message. You must use 0 if you specify a message key and the message is not a reply message.
---	--

- 1 Wait until the message arrives in the queue and is received, no matter how long it takes. The system has no limit for the wait time.
- n (any positive number)* Wait **n** seconds for the message to arrive in the queue.

If you specify a value of zero or above and the message does not arrive in the specified time, most fields in the RCVM0100 or RCVM0200 output are unchanged. The bytes returned output field has a value of 8, and the bytes available output field has a value of 0. The remaining output fields are unchanged; they contain whatever was already stored in the space.

**Message action**

INPUT; CHAR(10)

The action to take after the message is received. Valid values follow:

- \*OLD* Keep the message in the message queue and mark it as an old message. You can receive the message again only by using the message key or by specifying the message type *\*NEXT*, *\*PRV* (previous), *\*FIRST*, or *\*LAST*.
- \*REMOVE* Remove the message from the message queue. The message key is no longer valid, so you cannot receive the message again.
- \*SAME* Keep the message in the message queue without changing its new or old designation. *\*SAME* lets you receive the message again later without using the message key.

**Error code**

I/O; CHAR(\*)

The structure in which to return error information. For the format of the structure, see [Error Code Parameter](#).

**Optional Parameter Group 1**

**Coded character set identifier**

INPUT; BINARY(4)

The coded character set identifier (CCSID) in which that you want your message text and replacement data returned. This applies to the message text and the parts of the replacement data defined as a convertible character field (*\*CCHAR*). For more information about *\*CCHAR* fields, see the [Add Message Description \(ADDMSGD\) Command](#). The following values are allowed:

- 0 The received message is converted to the CCSID of the job before being returned. This is the default value if this parameter is not coded.  
  
If the job is 65535 and the text or data is something other than EBCDIC single byte or EBCDIC mixed, the text and data are converted to the default job CCSID.
- 65535 The received message will not be converted before being returned.

*CCSID* Specify a CCSID you want your text and data converted to before being returned. Only CCSIDs that a job can be changed to are acceptable values. This API will validate the CCSID specified.

**Note:** If an invalid CCSID conversion is detected during the receive function, the data is not converted before it is returned. The CCSID conversion status field should be checked to determine if an error occurred.

For more information about message handler and its use of CCSIDs, see [CCSIDs: Message Support](#) in the Globalization topic.



## Optional Parameter Group 2

### Allow default reply rejection

INPUT; CHAR(10)

Removing an unanswered inquiry causes the default reply to be sent to the inquiry message. This value indicates whether a reply handling exit program will be allowed to reject a default reply that is sent as a result of using this function. A reply handling exit program can be registered via the system registration facility for exit point QIBM\_QMH\_REPLY\_INQ. If this parameter is not specified, a value of \*NO is used. This parameter is only applicable when \*REMOVE is specified for the message action parameter. Valid values are:

*\*NO* A reply handling exit program will not be allowed to reject a default reply.

*\*YES* A reply handling exit program will be allowed to reject a default reply. If an exit program rejects the reply, a CPD2476 (Reply rejected by a reply handling exit program) will be sent as a diagnostic message to the program using this function. The CPD2476 will be followed by a CPF2422 (Reply not valid) escape message that the program using this function should monitor for to handle and recover from error situations.



## Message Types and Message Keys

The message type and message key parameters work together. Depending on the message type, the key can be required, optional, or disallowed. The following table lists each message type you can specify, tells whether it requires a key, and gives other information about using that type.

When used without a key, most message types receive only new messages. When used with a key, most can receive old or new messages. Message types \*FIRST, \*LAST, \*PRV (previous), and \*NEXT always receive both old and new messages.

**New messages** are messages that have been sent to a queue and have not yet been received. **Old messages** are messages that have been received but have not yet been removed from the queue.

All message types listed in the following table are received in first-in first-out (FIFO) order.

The following terms are used to describe the message key in the following table:

*Disallowed* Do not specify a message key. Instead, use blanks for the message key parameter. Specifying a message key results in an error.

*Required* Specify a message key. Not specifying a message key results in an error.

*Optional* You can either specify a message key or use blanks for the message key parameter.

When you do not specify a message key, the first new message of the specified type is received. If a new message of that type is not in the message queue, no error is returned. The unused space allowed for the output in the message information parameter is unchanged.

When you specify a message key and the message in the message queue is of the type specified, the message is received. If the message is not found, or if the message found does not match the type specified, an error code or exception is returned.

There are two cases where the message is not found and no error is returned. In both cases, the bytes returned field equals 8 and the bytes available field equals 0. The two cases are:

- Receiving without a message key (the key is optional or disallowed). A message of the specified type is not found in the queue.
- Receiving with a message key (the key is required) and the message type is \*PRV or \*NEXT. The message with the key specified was found in the queue, but no \*PRV or \*NEXT message is found.

The message types you can specify in the QMHRCVM API follow:

Message Type	Message Key	Description
*ANY	Optional	Receives a message of any type except sender's copy.
*COMP	Optional	Receives a completion message.
*COPY	Required	Receives the sender's copy of a previously sent inquiry message. The qualified message queue name parameter must specify the reply message queue specified when the inquiry was sent.
*DIAG	Optional	Receives a diagnostic message.
*FIRST	Disallowed	Receives the first new or old message in the queue.
*INFO	Optional	Receives an informational message.
*INQ	Optional	Receives an inquiry message. If the message action is *REMOVE and a reply to the inquiry message has not been sent yet, the default reply is automatically sent when the inquiry message is received.
*LAST	Disallowed	Receives the last new or old message in the queue.
*NEXT	Required	Receives the next new or old message after the message with the specified key.  You can use the special value *TOP for the message key. *TOP designates the message at the top of the message queue.  You can use hexadecimal zeros (hex 00000000) for the message key for the first receive operation to receive the first message on the queue.
*PRV	Required	Receives the new or old message before the message with the specified key.  You can use hexadecimal zeros (hex 00000000) for the message key for the first receive operation to receive the last message on the queue.
*RPY	Optional	Receives the reply to an inquiry message. For the message key, you can use the key to the sender's copy of the inquiry or notify message.

## RCVM0100 Format

The following table lists the fields returned in the RCVM0100 format. For more information about each item of information, see [Field Descriptions](#).

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	BINARY(4)	Message severity
12	C	CHAR(7)	Message identifier
19	13	CHAR(2)	Message type
21	15	CHAR(4)	Message key
25	19	CHAR(7)	Reserved
32	20	BINARY(4)	CCSID conversion status indicator of message data or text
36	24	BINARY(4)	CCSID of message data or text
40	28	BINARY(4)	Length of replacement data or impromptu message text returned
44	2C	BINARY(4)	Length of replacement data or impromptu message text available
48	30	CHAR(*)	Replacement data or impromptu message text

## RCVM0200 Format

The following table lists the fields returned in the RCVM0200 format. For more information about each item of information, see [Field Descriptions](#).

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	BINARY(4)	Message severity
12	C	CHAR(7)	Message identifier
19	13	CHAR(2)	Message type
21	15	CHAR(4)	Message key
25	19	CHAR(10)	Message file name
35	23	CHAR(10)	Message file library specified
45	2D	CHAR(10)	Message file library used
55	37	CHAR(10)	Sending job
65	41	CHAR(10)	Sending user profile

75	4B	CHAR(6)	Sending job's number
81	51	CHAR(12)	Sending program name
93	5D	CHAR(4)	Reserved
97	61	CHAR(7)	Date sent
104	68	CHAR(6)	Time sent
» 110	6E	CHAR(6)	Microseconds «
» 116	74 «	CHAR(17)	Reserved
127	7F	BINARY(4)	CCSID conversion status indicator for text
131	83	BINARY(4)	CCSID conversion status indicator for data
135	87	CHAR(9)	Alert option
144	90	BINARY(4)	CCSID of message or message help
148	94	BINARY(4)	CCSID of replacement data or impromptu message text
152	98	BINARY(4)	Length of replacement data or impromptu message text returned
156	9C	BINARY(4)	Length of replacement data or impromptu message text available
160	A0	BINARY(4)	Length of message returned
164	A4	BINARY(4)	Length of message available
168	A8	BINARY(4)	Length of message help returned
172	AC	BINARY(4)	Length of message help available
176	B0	CHAR(*)	Replacement data or impromptu text
The offsets to these fields equal the offset to the last fixed-length field plus the length of the previous variable length fields.	CHAR(*)	Message	
	CHAR(*)	Message help	

## Field Descriptions


The following field descriptions apply only when a message is received. If no message is found, only the bytes available and bytes returned fields contain new values. The remaining fields contain whatever information was already stored in the space allowed for the output.

**Alert option.** Whether and when an SNA alert is created and sent for the message. If a message is received, the value is one of the following:

\**DEFER*      An alert is sent after local problem analysis.

\**IMMED*      An alert is sent immediately when the message is sent to a message queue that has the allow alerts attribute set to \*YES.

- \*NO* No alert is sent.
- \*UNATTEND* An alert is sent immediately when the system is running in unattended mode (when the value of the alert status network attribute, ALRSTS, is *\*UNATTEND*).

For more information, see the [Alerts Support](#)  book.

**Bytes available.** The length of all available information that could be returned for the format. Bytes available can be greater than the length specified in the API's length of message information parameter. If it is greater, the information returned in the message information parameter is truncated to the length specified.

**Bytes returned.** The length of all information returned in the format. The value of the bytes returned field is always less than or equal to the length of the message information parameter. Also, it is always less than or equal to the bytes available. There is one exception to this. When you attempt to receive a message and the message is not found, the following occurs:

- The value of the bytes returned field is 8.
- The value of the bytes available field is 0.
- The remaining fields are unchanged (that is, they contain whatever was already stored in that space).

If the bytes returned value is less than the length specified in the length of message information parameter, the extra space in the message information parameter is unchanged.

**CCSID conversion status indicator for text.** The following values may be returned:

- 0 No conversion was needed because the CCSID of the message or message help text matched the CCSID you wanted the message or message help text converted to.
- 1 No conversion occurred because either the message or message help text was 65535 or the CCSID you wanted the message or message help text converted to was 65535.
- 2 No conversion occurred because you did not supply enough space for the message or message help.
- 3 The message or message help text was converted to the CCSID specified using the best fit conversion tables.
- 4 A conversion error occurred using the best fit conversion tables so a default conversion was attempted. This completed without error.
- 1 An error occurred on both the best fit and default conversions. The text was not converted.

**CCSID conversion status indicator for data.** The following values may be returned:

- 0 No conversion was needed because the CCSID of the replacement data or impromptu message text matched the CCSID you wanted the data or text converted to.
- 1 No conversion occurred because either the data was 65535 or the CCSID you wanted the data converted to was 65535.
- 2 No conversion occurred because you did not supply enough space for the data.
- 3 The data was converted to the CCSID specified using the best fit conversion tables.

- 4 A conversion error occurred using the best fit conversion tables so a default conversion was attempted. This completed without error.
- 1 An error occurred on both the best fit and default conversions. The data was not converted.

**CCSID of message and message help.** The CCSID of the text in the message and message help fields is returned. The inserted replacement data may not be the same CCSID. Refer to the CCSID of the replacement data or impromptu message text field description for a more details.

If a conversion error occurs or the CCSID you requested the text to be converted to is 65535, the CCSID that the message description is stored in is returned. Otherwise, the CCSID you wanted your text converted to is returned. If you do not want the text converted before it is returned to you but you do want to know the CCSID that the message description is stored in, specify 65535 on the coded character set identifier parameter. The CCSID that the message description is stored in is returned in the CCSID of message and message help output field.

**CCSID of replacement data or impromptu message text.** The CCSID of the replacement data or impromptu message text is returned. If an impromptu message text is received, this is the CCSID of the impromptu message text. When replacement data is received, this is the CCSID of the replacement data fields defined as convertible character (\*CCHAR) in the message description. All other replacement data is not converted before it is returned. If a conversion error occurs or the CCSID you requested the data to be converted to is 65535, the CCSID of the data or text is returned. If replacement data is being returned and there is no \*CCHAR replacement data, 65535 is returned. Otherwise the CCSID you wanted the data converted to is returned.

**Date sent.** The date on which the message was sent, in CYYMMDD (century, year, month, day) format.

**Length (general information about the following length fields).** These formats use two types of length fields, each related to a single variable length text field. (The variable length text fields return information to the caller.) The first type of length field is returned length; the second is available length. **Returned length** is the actual length of the text in the variable length text field. **Available length** is the length of the text before it is placed in the variable length text field. It is always greater than or equal to the returned length. If the available length equals the returned length, all the message information is returned. If the text is truncated when placed in the variable length field, the available length is greater than the returned length by the number of characters truncated.

**Length of message available.** The length of the available message text, in bytes. If an immediate message is received, the value of this field is zero.

**Length of message help available.** The length of the available message help information, in bytes. If an impromptu message is received, the value of this field is zero.

**Length of message help returned.** The length of the message help information, in bytes. If an impromptu message is received, the value of this field is zero.

**Length of message returned.** The length of the returned text of a predefined message, in bytes. If an impromptu message is received, the value of this field is zero.

**Length of replacement data or impromptu message text available.** The length of the available impromptu message text or replacement data, in bytes. If the message identifier is not blank, this field contains the length of the available replacement data for a predefined message. If the message identifier is blank, this field contains the length of the available text of an impromptu message.

**Length of replacement data or impromptu message text returned.** The length of the returned impromptu message text or replacement data, in bytes. If the message identifier is not blank, this field contains the length of the replacement data. If the message identifier is blank, this field contains the length of the impromptu message text.



**Message.** The text of a predefined message. If an impromptu message is received, this field is blank.

The API can truncate the message to fit the available space. If truncation occurs in the middle of double-byte character set (DBCS) data, the API returns only complete DBCS characters. It ends the data with a DBCS shift-in character.

**Message file name.** The name of the message file containing the message received.

**Message file library specified.** The name of the library containing the message file, as specified in the call to this API. If you specify \*CURLIB or \*LIBL for the library when you send the message, that value is returned as the library here. For the actual library used when the message is sent, see the message file library used field.

**Message file library used.** The name of the library used to send the message. Because the library can contain override instructions, this is not necessarily the library in which the message actually resides.

**Message help.** The message help for the message received. If an immediate message is received, this field is blank.

The API can truncate the message help to fit the available space. If truncation occurs in the middle of double-byte character set (DBCS) data, the API returns only complete DBCS characters. It ends the data with a DBCS shift-in character.

**Message identifier.** The identifying code of the message received. If an impromptu message is received, this field is blank.

**Message key.** The key to the message received. The key is assigned by the command or API that sends the message. If the message action parameter specifies \*REMOVE, this field is blank.

**Message severity.** The severity of the message received. Possible values are 0 through 99.

**Message type.** The message type of the message received. The possible values and their meanings are:

<b>Value</b>	<b>Message Type</b>
01	Completion
02	Diagnostic
04	Informational
05	Inquiry
06	Sender's copy
08	Request
10	Request with prompting
14	Notify
15	Escape
21	Reply, not validity checked
22	Reply, validity checked
23	Reply, message default used

- 24 Reply, system default used
- 25 Reply, from system reply list

» **Microseconds.** The microseconds part of the time sent. «

**Replacement data or impromptu message text.** The values for substitution variables in a predefined message, or the text of an impromptu message. If the message identifier is not blank, this field contains message data. If the message identifier is blank, this field contains impromptu message text.

If this field contains message data that contains pointer data, each pointer must start on a 16-byte boundary. If you are running at security level 50, the pointer data is invalidated.

The API can truncate the data or text to fit the available space. If the field contains the text of an impromptu message and is truncated in the middle of double-byte character set (DBCS) data, the API returns only complete DBCS characters. It ends the data with a DBCS shift-in character. However, if the field contains data for a predefined message, the API does not check for DBCS data. This is because message data can contain pointers, and pointers can contain the same characters used to mark DBCS data.

**Reserved.** An ignored field.

**Sending job.** The name of the job in which the message being received was sent.

**Sending job's number.** The job number of the job in which the message being received was sent.

**Sending program name.** The original program model (OPM) program name or ILE bound program name that contains the procedure sending the message.

**Sending program's instruction number.** The number of the program instruction that issued the command or called the API used to send the message being received.

**Sending user profile.** The name of the user profile that sent the message being received.

**Time sent.** The time at which the message being received was sent, in HHMMSS (hour, minute, second) format.

## Error Messages

Message ID	Error Message Text
CPF24AF E	Message key not allowed with message type specified.
CPF24A7 E	Value for the length of message information not valid.
CPF24A8 E	Value for wait time not valid.
CPF24A9 E	Value for message action not valid.
CPF24B1 E	Message key required for message type specified.
CPF24B2 E	Message key of *TOP requires message type of *NEXT.
CPF24B3 E	Message type &1 not valid.
CPF24B4 E	Severe error while addressing parameter list.

CPF2401 E Not authorized to library &1.  
CPF2403 E Message queue &1 in &2 not found.  
CPF2407 E Message file &1 in &2 not found.  
CPF2408 E Not authorized to message queue &1.  
CPF2410 E Message key not found in message queue &1.  
CPF2411 E Not authorized to message file &1 in &2.  
»CPF2422 E Reply not valid.«  
CPF2433 E Function not allowed for system log message queue &1.  
CPF2450 E Work station message queue &1 not allocated to job.  
CPF2451 E Message queue &1 is allocated to another job.  
CPF247E E CCSID &1 is not valid.  
CPF2477 E Message queue &1 currently in use.  
CPF2531 E Message file &1 in &2 damaged for &3.  
CPF2548 E Damage to message file &1 in &2.  
CPF2551 E Message key and message type combination not valid.  
CPF3CF1 E Error code parameter not valid.  
CPF3C21 E Format name &1 is not valid.  
»CPF3C3A E Value for parameter &2 for API &1 not valid.«  
CPF3C36 E Number of parameters, &1, entered for this API was not valid.  
CPF3C90 E Literal value cannot be changed.  
CPF8100 E All CPF81xx messages could be returned. xx is from 01 to FF.  
CPF9830 E Cannot assign library &1.  
CPF9872 E Program or service program &1 in library &2 ended. Reason code &3.

---

API introduced: V2R1.1

---

[Top](#) | [Message Handling APIs](#) | [APIs by category](#)

# Receive Program Message (QMHRCVPM) API

## Required Parameter Group:

1	Message information	Output	Char(*)
2	Length of message information	Input	Binary(4)
3	Format name	Input	Char(8)
4	Call stack entry	Input	Char(*) or Pointer
5	Call stack counter	Input	Binary(4)
6	Message type	Input	Char(10)
7	Message key	Input	Char(4)
8	Wait time	Input	Binary(4)
9	Message action	Input	Char(10)
10	Error code	I/O	Char(*)

## Optional Parameter Group 1:

11	Length of call stack entry	Input	Binary(4)
12	Call stack entry qualification	Input	Char(20)

## Optional Parameter Group 2:

13	Call stack entry data type	Input	Char(10)
14	Coded character set identifier	Input	Binary(4)

Default Public Authority: \*USE

Threadsafe: Yes

The Receive Program Message (QMHRCVPM) API receives a message from a call message queue or external message queue and returns information describing the message.

To receive a message from nonprogram message queues, see [Receive Nonprogram Message](#) (QMHRVCM) API.

In a multithreaded job messages can be received from call message queues within the thread that calls this API, and messages sent from within the same thread can be received from the external message queue. Messages on call message queues in other threads or sent to the external message queue from other threads cannot be received.

## Authorities and Locks

None.

# Required Parameter Group

## Message information

OUTPUT; CHAR(\*)

The variable that receives the information returned, in the format specified in the format name parameter, of the length specified in the length of message information parameter.

## Length of message information

INPUT; BINARY(4)

The size of the area to contain the message information, in bytes. The minimum size is 8.

This parameter must specify the size of the variable you use for the message information parameter. If this parameter specifies a longer size, other parts of storage could be overwritten when the API returns the information.

The API returns as much information as it can fit in this length. If the available message information is longer, it is truncated. If the available message information is shorter, the unused output area is unchanged; whatever is already stored in that space remains there.

To determine how much information the API actually returns in response to this call, see the bytes returned field in the RCVMO100, RCVMO200, or RCVMO300 output. To determine how much information the API could return if space were available, see the bytes available field.

## Format name

INPUT; CHAR(8)

The format to use for the message information. Specify one of these format names:

*RCVM0100* Brief message information. For details, see [RCVM0100 Format](#).

*RCVM0200* All message information. For details, see [RCVM0200 Format](#).

*RCVM0300* All message information. Complete sender information for the message being received. For details, see [RCVM0300 Format](#).

## Call stack entry

INPUT; CHAR(\*) or Pointer

The call stack entry from whose message queue messages are to be received, or the call stack entry to start counting from when using a value other than 0 for the call stack counter parameter. The call stack entry you specify must be in the call stack or you can specify the external message queue instead of a call stack entry.

You can specify a call stack entry by providing the name of the OPM program or ILE procedure running in the entry, by providing a pointer to the call stack entry, or by using one of the following special values:

- \* The current call stack entry (that is, the one in which the API is being used).
- \**EXT* The external message queue. The call stack counter parameter is ignored. You cannot receive escape messages from this queue.

If the call stack entry is to be identified by pointer, the pointer that is specified must address a valid

call stack entry within the same job as the one the API is used in. Alternatively, the pointer can be set to Null. The pointer must be 16 byte aligned. The Optional Parameter Group 1 must be used and the Length of call stack entry parameter must be set to 16. In addition, the Optional Parameter Group 2 must also be used and the Call stack entry format parameter must be set to \*PTR.

If the pointer provided is set to Null, this indicates that the call stack entry is the one in which the API is being used.

If the pointer does not address a valid call stack entry or is not a Null pointer, the error message CPF24C5 is sent to the user of the API.

The call stack entry can be a nested procedure name from 1 through 4096 characters in length. When specifying nested procedures, each procedure name must be separated by a colon, and the outermost procedure is identified first followed by the procedures it contains. The innermost procedure is the last procedure identified in the string.

The call stack entry can be a partial name. To specify a partial name, place three less-than signs (<<<) at the beginning of the call stack entry identifier, or place three greater-than signs (>>>) at the end of the call stack entry identifier, or place both the less-than signs and the greater-than signs at their respective ends of the call stack entry identifier. The value for the call stack entry excluding the less-than signs and the greater-than signs is used to search backward through the stack for the requested call stack entry name.

When searching for a partial call stack entry name:

- If the less-than signs (<<<) are specified only at the beginning of the call stack entry name, the less-than signs are truncated and the remaining character string is right-justified. The remaining string is then compared to the current call stack entry on the call stack. The comparison starts at the end of the call stack entry name and backwardly compares the number of characters in the specified string.
- If the greater-than signs (>>>) are specified only at the end of the call stack entry name, the greater-than signs are truncated. The remaining character string is compared to the current call stack entry on the call stack. The comparison starts at position 1 of the call stack entry name and compares the number of characters in the specified string.
- If the less-than signs (<<<) are specified at the beginning of the call stack entry name and the greater-than signs (>>>) are specified at the end of the call stack entry name, both the less-than signs and the greater-than signs are truncated. The remaining characters are used to scan and to compare the entire length of the specified string and the current call stack entry on the call stack.

**Note:** If the optional parameters Length of to call stack entry and To call stack entry data type are not specified, this parameter is assumed to be CHAR(10).

### Call stack counter

INPUT; BINARY(4)

A number identifying the location in the call stack of the call stack entry from whose message queue the messages are to be received. The number is relative to the call stack entry identified by the Call stack entry parameter. It indicates how many calls up the call stack the target entry is from the one identified by the Call stack entry parameter. Valid values follow:

- 0 Receive the message from the message queue of the call stack entry specified in the Call stack entry parameter.

- 1* Receive the message from the message queue of the call stack entry that is one earlier than the entry identified by the Call stack entry parameter.
- n (any positive number)* Receive the message from the message queue of the nth call stack entry up the stack from the call stack entry specified in the Call stack entry parameter.
- You can use any positive number that does not exceed the actual number of call stack entries in the call stack, excluding the external message queue.

### **Message type**

INPUT; CHAR(10)

The type of the message being received. The message type and message key parameters work together. Depending on the message type, the key can be required, optional, or disallowed. For a list of valid message types and information about how they work with the message key parameter, see [Message Types and Message Keys](#).

### **Message key**

INPUT; CHAR(4)

The key to the message being received. The key is assigned by the command or API that sends the message.

What you can use for the message key parameter depends on what you use for the message type. For details, see [Message Types and Message Keys](#).

If you are not receiving messages by key, use blanks for this parameter.

If you specify a key and the message queue does not contain a message with that key, an error is returned.

If you specify a key and the message queue does contain a message with that key, the API might or might not return a message. It never returns an error in this case. Whether or not the API returns a message depends on the value of the message type parameter. For example, if you specify the message type \*PRV and there is no message before the message with the key, the API does not return a message. Because the key you specified is valid, the API does not return an error either.

You can receive the reply to a message through the key to the sender's copy of the message. If the reply is not available, no message is returned, and the API does not return an error.

When the message type is \*NEXT, you can use \*TOP for the message key to receive the next message after the last message received.

When the message type is \*NEXT or \*PRV, you can use hexadecimal zeros for the message key for the first receive operation.

If you know the message key of a message you want to receive, you can receive that message without regard to the call message queue containing the message. You can do this by specifying the key in this parameter, the special value '\*' for the Call stack entry parameter and the value '0' for the call stack counter parameter. This is useful if the message was sent to a call stack entry that is no longer in the call stack.

### **Wait time**

INPUT; BINARY(4)

The length of time to wait for the message to arrive in the queue so it can be received. Valid values follow:

- 0* Do not wait for the message. If the message is not in the queue and you specified a message key, an error is returned.
- 1* Wait until the message arrives in the queue and is received, no matter how long it takes. The system has no limit for the wait time.
- n (any positive number)* Wait **n** seconds for the message to arrive in the queue.

If you specify a value of zero or above and the message does not arrive in the specified time, most fields in the RCVMO100, RCVMO200 or RCVMO300 output are unchanged. The bytes returned output field has a value of 8, and the bytes available output field has a value of 0. The remaining output fields are unchanged; they contain whatever was already stored in the space.

### Message action

INPUT; CHAR(10)

The action to take after the message is received. Valid values follow:

- \*OLD* Keep the message in the message queue and mark it as an old message. You can receive the message again only by using the message key or by specifying the message type *\*NEXT*, *\*PRV* (previous), *\*FIRST*, or *\*LAST*.
- \*REMOVE* Remove the message from the message queue. This instance of the message is no longer available for you to work with. The message key is no longer valid; therefore, you cannot receive the message again.
- \*SAME* Keep the message in the message queue without changing its new or old designation. *\*SAME* lets you receive the message again later without using the message key.

For Integrated Language Environment (ILE) programs, the message action also indicates if an exception message should be handled. If the message action is *\*OLD* or *\*REMOVE*, an exception is handled. If *\*SAME* is specified, the exception is not handled.

### Error code

I/O; CHAR(\*)

The structure in which to return error information. For the format of the structure, see [Error Code Parameter](#).

## Optional Parameter Group 1

### Length of call stack entry

INPUT; BINARY(4)

The length of the value for the call stack entry parameter. Valid values for this parameter are as follows:

- 1 through 4096 if partial name indicators are not used.



- 16 if the call stack entry parameter is a pointer.
- 1 through 4102 if partial name indicators are used.

**Note:** The actual length of the call stack entry name cannot exceed 4096 characters. If this parameter is not used, the value for the call stack entry parameter is assumed to be 10 characters in length. If this parameter is not used, the value for the call stack entry parameter is assumed to be 10 characters in length.

### Call stack entry qualification

INPUT; CHAR(20)

The name of the module and the ILE program or service program to further qualify the procedure name specified by the Call stack entry parameter. The first 10 characters specify the module name, and the second 10 characters specify the ILE program or service program name. If this parameter is not used, only the Call stack entry parameter is used to determine the call stack entry. The following special value may be used:

*\*NONE* This value is used for the module or ILE program name or service program name, or both. When *\*NONE* is specified for one of the names, then that name is not used when searching for the call stack entry. If *\*NONE* is specified for both names, only the Call stack entry parameter is used to identify the call stack entry.

If the call stack entry is to be identified by pointer, this parameter must still be passed to the API but both the module name and program name qualifiers must be specified as *\*NONE*. When a pointer is used, the module and program name qualification is not applicable.

The module name and bound program name should be used only when identifying the call stack entry for an ILE procedure. When identifying the entry for an OPM program, the Optional Parameter Group 1 should either not be used or both the module name and bound program name should be specified as *\*NONE*.

## Optional Parameter Group 2

### Call Stack entry data type

INPUT; CHAR(10)

Whether the value of the call stack entry parameter is a character string (a name or special value) or a pointer. Use one of the following special values:

*\*CHAR* Value of the parameter is a character string (name or special value).

*\*PTR* Value of the parameter is a pointer.

If the above optional parameter is not specified, it is assumed that the value of the Call stack entry parameter is a character string.

### Coded character set identifier

INPUT; BINARY(4)

The coded character set identifier (CCSID) that you want your message text and message data returned in. This applies to the text and the parts of the message data defined as a convertible character field (*\*CCHAR*). For more information about *\*CCHAR* fields, see the [Add Message Description \(ADDMSGD\) Command](#). The following values are allowed:

- 0* The received message is converted to the CCSID of the job before being returned.
- If the job is 65535 and the text or data is something other than EBCDIC single byte or EBCDIC mixed, the text and data are converted to the default job CCSID.
- 65535* The received message will not be converted before being returned.
- CCSID* Specify a CCSID you want your text and data converted to before being returned. Valid values are between 1 and 65535. This API will validate the CCSID specified.
- Only CCSIDs that a job can be changed to are accepted. For a list of valid job CCSIDs, display the prompt for the CCSID parameter on the CHGJOB command.
- If this parameter is not specified, 0 is used and messages are returned in the CCSID of the job.

**Note:** If an invalid CCSID conversion is detected during the receive function, the data is not converted before it is returned. The CCSID conversion status field should be checked to determine if an error occurred.

For more information about message handler and its use of CCSIDs, see [CCSIDs: Message Support](#) in the Globalization topic.

## Message Types and Message Keys

The message type and message key parameters work together. Depending on the message type, the key can be required, optional, or disallowed. The following table lists each message type you can specify, tells whether it requires a key, and gives other information about using that type.

When used without a key, most message types receive only new messages. When used with a key, most can receive old or new messages. Message types \*FIRST, \*LAST, \*PRV (previous), and \*NEXT always receive both old and new messages.

**New messages** are messages that have been sent to a queue and have not yet been received. **Old messages** are messages that have been received but have not yet been removed from the queue.

Messages of type \*EXCP are received in last-in first-out (LIFO) order. Messages of all other types are received in first-in first-out (FIFO) order.

The following terms are used to describe the message key in the following table:

- Disallowed* Do not specify a message key. Instead, use blanks for the message key parameter. Specifying a message key results in an error.
- Required* Specify a message key. Not specifying a message key results in an error.

*Optional* You can either specify a message key or use blanks for the message key parameter.

When you do not specify a message key, the first new message of the specified type is received. If a new message of that type is not in the message queue, no error is returned. The unused space allowed for the output in the message information parameter is unchanged.

When you specify a message key and the message in the message queue is of the type specified, the message is received. If the message is not found, or if the message found does not match the type specified, an error code or exception is returned.

There are two cases where the message is not found and no error is returned. In both cases, the bytes returned field equals 8 and the bytes available field equals 0. The two cases are:

- Receiving without a message key (the key is optional or disallowed). A message of the specified type is not found in the queue.
- Receiving with a message key (the key is required) and the message type is \*PRV or \*NEXT. The message with the key specified was found in the queue, but no \*PRV or \*NEXT message is found.

The message types you can specify in the QMHRCVPM API are:

Message Type	Message Key	Description
*ANY	Optional	If message key is blanks, then this receives a message of any type except a sender's copy or request. If the message key is not blank, this receives the requested message. If the message is a sender's copy type message, the associated reply is received.
*COMP	Optional	Receives a completion message.
*COPY	Required	Receives a copy of a previously sent inquiry message. The program message queue parameter must specify the reply message queue specified when the inquiry was sent.
*DIAG	Optional	Receives a diagnostic message.
*ESCAPE	Optional	Receives an escape message.
*EXCP	Optional	Receives an escape or notify message. Both types of messages are received in last-in first-out (LIFO) order.
*FIRST	Disallowed	Receives the first new or old message in the queue, unless it is a request message. Request messages are skipped.
*INFO	Optional	Receives an informational message.
*LAST	Disallowed	Receives the last new or old message in the queue.
*NEXT	Required	Receives the next new or old message after the message with the specified key, unless it is a request message. Request messages are skipped.  You can use the special value *TOP for the message key. *TOP designates the message at the top of the message queue.  You can use hexadecimal zeros (hex 00000000) for the message key for the first receive operation.  If *TOP or hexadecimal zeros are specified, the search for the next message begins after the last request message is received. If no request messages have been received, the search starts at the top of the message queue.

*NOTIFY	Optional	Receives a notify message.
*PRV	Required	Receives the new or old message before the message with the specified key.  You can use hexadecimal zeros (hex 00000000) for the message key for the first receive operation to receive the last message on the queue.
*RPY	Optional	Receives the reply to an inquiry or notify message. For the message key, you can use the key to the sender's copy of the inquiry or notify message.
*RQS	Optional	Receives the next request in the program message queue for the program. Two actions are possible if no request exists: <ul style="list-style-type: none"> <li>1. If the job is interactive and the program queue is the external message queue, the Command Entry display is called to allow the work station user to type requests.</li> <li>2. In all other cases, an error code or exception is returned.</li> </ul>

## RCVM0100 Format

The following table lists the fields returned in the RCVM0100 format. For more information about each item of information, see [Field Descriptions](#).

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	BINARY(4)	Message severity
12	C	CHAR(7)	Message identifier
19	13	CHAR(2)	Message type
21	15	CHAR(4)	Message key
25	19	CHAR(7)	Reserved
32	20	BINARY(4)	CCSID conversion status indicator of message data or text
36	24	BINARY(4)	CCSID of replacement data or impromptu message text
40	28	BINARY(4)	Length of replacement data or impromptu message text returned
44	2C	BINARY(4)	Length of replacement data or impromptu message text available
48	30	CHAR(*)	Replacement data or impromptu message text

## RCVM0200 Format

The following table lists the fields returned in the RCVM0200 format. For more information about each item of information, see [Field Descriptions](#).

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	BINARY(4)	Message severity
12	C	CHAR(7)	Message identifier
19	13	CHAR(2)	Message type
21	15	CHAR(4)	Message key
25	19	CHAR(10)	Message file name
35	23	CHAR(10)	Message file library specified
45	2D	CHAR(10)	Message file library used
55	37	CHAR(10)	Sending job
65	41	CHAR(10)	Sending user profile
75	4B	CHAR(6)	Sending job's number
81	51	CHAR(12)	Sending program name
93	5D	CHAR(4)	Sending program's instruction number
97	61	CHAR(7)	Date sent
104	68	CHAR(6)	Time sent
110	6E	CHAR(10)	Receiving program name
120	78	CHAR(4)	Receiving program's instruction number
124	7C	CHAR(1)	Sending type
125	7D	CHAR(1)	Receiving type
126	7E	CHAR(1)	Reserved
127	7F	BINARY(4)	CCSID conversion status indicator for text
131	83	BINARY(4)	CCSID conversion status indicator for data
135	87	CHAR(9)	Alert option
144	90	BINARY(4)	CCSID of message and message help
148	94	BINARY(4)	CCSID of replacement data or impromptu message text
152	98	BINARY(4)	Length of replacement data or impromptu message text returned
156	9C	BINARY(4)	Length of replacement data or impromptu message text available
160	A0	BINARY(4)	Length of message returned
164	A4	BINARY(4)	Length of message available
168	A8	BINARY(4)	Length of message help returned
172	AC	BINARY(4)	Length of message help available
176	B0	CHAR(*)	Replacement data or impromptu message text

The offsets to these fields equal the offset to the last fixed-length field plus the length of the previous variable length fields.	CHAR(*)	Message
	CHAR(*)	Message help

## RCVM0300 Format

The following table lists the fields returned in the RCVM0300 format. For more information about each item of information, see [Field Descriptions](#).

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	BINARY(4)	Message severity
12	C	CHAR(7)	Message identifier
19	13	CHAR(2)	Message type
21	15	CHAR(4)	Message key
25	19	CHAR(10)	Message file name
35	23	CHAR(10)	Message file library specified
45	2D	CHAR(10)	Message file library used
55	37	CHAR(9)	Alert option
64	40	BINARY(4)	CCSID conversion status indicator of message and message help
68	44	BINARY(4)	CCSID conversion status indicator of message data or text
72	48	BINARY(4)	CCSID of replacement data or impromptu message text
76	4C	BINARY(4)	CCSID of replacement data and message help
80	50	BINARY(4)	Length of replacement data or impromptu message text returned
84	54	BINARY(4)	Length of replacement data or impromptu message text available
88	58	BINARY(4)	Length of message returned
92	5C	BINARY(4)	Length of message available
96	60	BINARY(4)	Length of message help returned
100	64	BINARY(4)	Length of message help available
104	68	BINARY(4)	Length of sender information returned
108	6C	BINARY(4)	Length of sender information available

112	70	CHAR(*)	Replacement data or impromptu message text
The offsets to these fields equal the offset to the last fixed-length field plus the length of the previous variable length fields.		CHAR(*)	Message
		CHAR(*)	Message help
		CHAR(*)	Sender information

## Sender Information Format

The following table lists the fields for the sender information format of the RCV0300 format. For more information about each item of information, see [Field Descriptions](#).

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	Sending job
10	A	CHAR(10)	Sending user profile
20	14	CHAR(6)	Sending job's number
26	1A	CHAR(7)	Date sent
33	21	CHAR(6)	Time sent
39	27	CHAR(1)	Sending type
40	28	CHAR(1)	Receiving type
41	29	CHAR(12)	Sending program name
53	35	CHAR(10)	Sending module name
63	3F	CHAR(256)	Sending procedure name
319	13F	CHAR(1)	Reserved
320	140	BINARY(4)	Number of statement numbers or instruction numbers available for the sending program or procedure
324	144	CHAR(30)	Sending program's statement numbers or instruction numbers
354	162	CHAR(10)	Receiving program name
364	16C	CHAR(10)	Receiving module name
374	176	CHAR(256)	Receiving procedure name
630	276	CHAR(10)	Reserved
640	280	BINARY(4)	Number of statement numbers or instruction numbers available for the receiving program or procedure
644	284	CHAR(30)	Receiving program's statement number or instruction number
674	2A2	CHAR(2)	Reserved


676	2A4	BINARY(4)	Displacement to long sending program name
680	2A8	BINARY(4)	Length of long sending program name
684	2AC	BINARY(4)	Displacement to long sending procedure name
688	2B0	BINARY(4)	Length of long sending procedure name
692	2B4	BINARY(4)	Displacement to long receiving procedure name
696	2B8	BINARY(4)	Length of long receiving procedure name
➤ 700	2BC	CHAR(6)	Microseconds ⏪
➤ 706	2C2 ⏪	CHAR(*)	Reserved
The offsets to these fields are found in the displacement fields identified in this table.		CHAR(*)	Long sending program name
		CHAR(*)	Long sending procedure name
		CHAR(*)	Long receiving procedure name

## Field Descriptions

The following field descriptions apply only when a message is received. If no message is found, only the bytes available and bytes returned fields contain new values. The remaining fields contain whatever information was already stored in the space allowed for the output.

**Alert option.** Whether and when an SNA alert is created and sent for the message. If a message is received, the value is one of the following:

- \**DEFER*      An alert is sent after local problem analysis.
- \**IMMED*      An alert is sent immediately when the message is sent to a message queue that has the allow alerts attribute set to \*YES.
- \**NO*            No alert is sent.
- \**UNATTEND*    An alert is sent immediately when the system is running in unattended mode (when the value of the alert status network attribute, ALRSTS, is \*UNATTEND).

For more information, see the [Alerts Support](#)  book.

**Bytes available.** The length of all available information that could be returned for the format. Bytes available can be greater than the length specified in the API's length of message information parameter. If it is greater, the information returned in the message information parameter is truncated to the length specified.

**Bytes returned.** The length of all information returned in the format. The value of the bytes returned field is always less than or equal to the length of the message information parameter. Also, it is always less than or equal to the bytes available. There is one exception to this. When you attempt to receive a message and the message is not found, the following occurs:

- The value of the bytes returned field is 8.
- The value of the bytes available field is 0.
- The remaining fields are unchanged (that is, they contain whatever was already stored in that



space).

If the bytes returned value is less than the length specified in the length of message information parameter, the extra space in the message information parameter is unchanged.

**CCSID conversion status indicator for data.** The following values may be returned:

- 0 No conversion was needed because the CCSID of the replacement data or impromptu message text matched the CCSID you wanted the data converted to.
- 1 No conversion occurred because either the data was 65535 or the CCSID you wanted the data converted to was 65535.
- 2 No conversion occurred because you did not supply enough space for the data.
- 3 The data was converted to the CCSID specified using the best fit conversion tables.
- 4 A conversion error occurred using the best fit conversion tables so a default conversion was attempted. This completed without error.
- 1 An error occurred on both the best fit and default conversions. The data was not converted.

**CCSID conversion status indicator for text.** The following values may be returned:

- 0 No conversion was needed because the CCSID of the message or message help text matched the CCSID you wanted the message or message help text converted to.
- 1 No conversion occurred because either the message or message help text was 65535 or the CCSID you wanted the message or message help text converted to was 65535.
- 2 No conversion occurred because you did not supply enough space for the message or message help text.
- 3 The message or message help text was converted to the CCSID specified using the best fit conversion tables.
- 4 A conversion error occurred using the best fit conversion tables so a default conversion was attempted. This completed without error.
- 1 An error occurred on both the best fit and default conversions. The text was not converted.

**CCSID of message and message help.** The CCSID of the text in the message and message help fields is returned. The inserted replacement data may not be the same CCSID. Refer to the CCSID of the replacement data or impromptu message text field description for a more details.

If a conversion error occurs or the CCSID you requested the text to be converted to is 65535, the CCSID that the message description is stored in is returned. Otherwise, the CCSID you wanted your text converted to is returned. If you do not want the text converted before it is returned to you but you do want to know the CCSID that the message description is stored in specify 65535 on the Coded character set identifier parameter. The CCSID that the message description is stored in is returned in the CCSID of message and message help output field.

**CCSID of replacement data or impromptu message text.** The CCSID of the impromptu message text or replacement data is returned. If an impromptu message is received, this is the CCSID of the impromptu message text. When replacement data is received, this is the CCSID of the message data fields defined as convertible character (\*CCHAR) in the message description. All other replacement data will not be converted before it is returned. If a conversion error occurs or if the CCSID you requested the data to be converted to is 65535, the CCSID of the data is returned. If message data is being returned and there is no

\*CCHAR replacement data, 65535 is returned. Otherwise, the CCSID you wanted the data converted to is returned.

**Date sent.** The date on which the message was sent, in CYYMMDD (century, year, month, day) format.

**Displacement to long receiving procedure name.** When the receiving type is 1 or 2, the offset to the long receiving procedure name, in bytes, from the beginning of the data returned in the sender information format. If there is not enough room to contain any of the long receiving procedure, this field is zero.

**Displacement to long sending procedure name.** When the sending type is 1 or 2, the offset to the long sending procedure name, in bytes, from the beginning of the data returned in the sender information format. If there is not enough room to contain any of the long sending procedure, this field is zero.

**Displacement to long sending program name.** The offset to the long sending program name, in bytes, from the beginning of the data returned in the sender information format. If there is not enough room to contain any of the long sending program, this field is zero.

**Length (general information about the following length fields).** These formats use two types of length fields, each related to a single variable length text field. (The variable length text fields return information to the caller.) The first type of length field is returned length; the second is available length. **Returned length** is the actual length of the text in the variable length text field. **Available length** is the length of the text before it is placed in the variable length text field. It is always greater than or equal to the returned length. If the available length equals the returned length, all the message information is returned. If the text is truncated when placed in the variable length field, the available length is greater than the returned length by the number of characters truncated.

**Length of long receiving procedure name.** For type 1 and 2, the length of the complete receiving procedure name, in bytes.

**Length of long sending procedure name.** For type 1 and 2, the length of the complete sending procedure name, in bytes.

**Length of long sending program name.** The length of the complete sending program name, in bytes.

**Length of message available.** The length of the available message text, in bytes. If an immediate message is received, the value of this field is zero.

**Length of message help available.** The length of the available message help information, in bytes. If an impromptu message is received, the value of this field is zero.

**Length of message help returned.** The length of the message help information, in bytes. If an impromptu message is received, the value of this field is zero.

**Length of message returned.** The length of the returned text of a predefined message, in bytes. If an impromptu message is received, the value of this field is zero.

**Length of replacement data or impromptu message available.** The length of the available impromptu message text or replacement data, in bytes. If the message identifier is not blank, this field contains the length of the available message data for a predefined message. If the message identifier is blank, this field contains the length of the available text of an impromptu message.

**Length of replacement data or impromptu message returned.** The length of the returned impromptu message text or replacement data, in bytes. If the message identifier is not blank, this field contains the length of the replacement data. If the message identifier is blank, this field contains the length of the impromptu message text.

**Length of sender information available.** The length, in bytes, of information available in the sender information format.

**Length of sender information returned.** The length, in bytes, of information returned in the sender information format.

**Long receiving procedure name.** For type 1 and 2, complete procedure name that received the message. A nested procedure name has each procedure name separated by a colon. The outermost procedure name is identified first followed by the procedures it contains. The innermost procedure is identified last in the string.

**Long sending procedure name.** For type 1 and 2, complete procedure name that sent the message. A nested procedure name has each procedure name separated by a colon. The outermost procedure name is identified first followed by the procedures it contains. The innermost procedure is identified last in the string.

**Long sending program name.** The program name that sent the message.

**Message.** The text of a predefined message. If an impromptu message is received, this field is blank.

The API can truncate the message to fit the available space. If truncation occurs in the middle of double-byte character set (DBCS) data, the API returns only complete DBCS characters. It ends the data with a DBCS shift-in character.

**Message file library specified.** The name of the library containing the message file, as specified in the call to this API. If you specify \*CURLIB or \*LIBL for the library when you send the message, that value is returned as the library here. For the actual library used when the message is sent, see the message file library used field.

**Message file library used.** The name of the library used to send the message. Because the library can contain override instructions, this is not necessarily the library in which the message actually resides.

**Message file name.** The name of the message file containing the message received.

**Message help.** The message help for the message received. If an immediate message is received, this field is blank.

The API can truncate the message help to fit the available space. If truncation occurs in the middle of double-byte character set (DBCS) data, the API returns only complete DBCS characters. It ends the data with a DBCS shift-in character.

**Message identifier.** The identifying code of the message received. If an immediate message is received, this field is blank.

**Message key.** The key to the message received. The key is assigned by the command or API that sends the message. If the message action parameter specifies \*REMOVE, this field is blank.

**Message severity.** The severity of the message received. Possible values are 0 through 99.

**Message type.** The message type of the message received. The possible values and their meanings are:

<b>Value</b>	<b>Message Type</b>
<i>01</i>	Completion
<i>02</i>	Diagnostic
<i>04</i>	Informational
<i>05</i>	Inquiry

06	Sender's copy
08	Request
10	Request with prompting
14	Notify
15	Escape
16	Notify, exception not handled when API is called
17	Escape, exception not handled when API is called
21	Reply, not validity checked
22	Reply, validity checked
23	Reply, message default used
24	Reply, system default used
25	Reply, from system reply list

» **Microseconds.** The microseconds part of the time sent. «

**Name of the procedure receiving the message.** This field is blank if the message was received by an original program model (OPM) program.

**Number of statement numbers or instruction numbers available for the receiving program or procedure.** For OPM programs and nonoptimized procedures, this count is 1. For optimized procedures, this count can be greater than 1. In this case, each statement number represents a potential point at which the message could have been received. If the mapping table information is removed from the program, this field returns a count of zero and no statement numbers are available.

**Number of statement numbers or instruction numbers available for the sending program or procedure.** For OPM programs and nonoptimized procedures, this count is 1. For optimized procedures, this count can be greater than 1. In this case, each statement number represents a potential point at which the message could have been sent. If the mapping table information is removed from the program, this field returns a count of zero and no statement numbers are available.

**Receiving module name.** For type 1 and 2, the name of the module receiving the message.

**Receiving procedure name.** For type 1 and 2, the name of the procedure receiving the message. A nested procedure name has each procedure name separated by a colon. The outermost procedure name is identified first followed by the procedures it contains. The innermost procedure is identified last in the string.

**Receiving program name.** The name of the program receiving the message, or the Integrated Language Environment (ILE) program name that contains the procedure receiving the message.

**Receiving program's statement number or instruction number.** This field can contain up to three statement numbers or instruction numbers. When the receiving type is 1 or 2, the field contains statement numbers. When the receiving type is 0, this field contains an instruction number. Each statement number can be up to 10 characters in length. The instruction number is 4 characters in length. Each statement number or instruction number is left-justified in a 10-character partition of the 30-character field. The first statement number or instruction number is in the leftmost 10 characters and the third in the rightmost 10 characters. The unused parts of this field are set to blanks.

**Receiving type.** The type of program that received the message. Valid values follow:

- 0 The message was sent to an original program model (OPM) program.
- 1 The message was sent to a procedure within an ILE program, and the procedure name is up to and including 256 characters in length.
- 2 The message was sent to a procedure within an ILE program, and the procedure name is from 257 characters up to and including 4096 characters in length. For this type, the receiving procedure name is blank, and the name can be obtained from the long receiving procedure name.

**Replacement data or impromptu message text.** The values for substitution variables in a predefined message, or the text of an impromptu message. If the message identifier is not blank, this field contains message data. If the message identifier is blank, this field contains impromptu message text.

If this field contains message data that contains pointer data, each pointer must start on a 16-byte boundary. If you are running at security level 50, the pointer data is invalidated.

The API can truncate the data or text to fit the available space. If the field contains the text of an impromptu message and is truncated in the middle of double-byte character set (DBCS) data, the API returns only complete DBCS characters. It ends the data with a DBCS shift-in character. However, if the field contains data for a predefined message, the API does not check for DBCS data. This is because message data can contain pointers, and pointers can contain the same characters used to mark DBCS data.

**Reserved.** An ignored field.

**Sending job.** The name of the job in which the message being received was sent. Because the sending job also must be the receiving job for call message queues, this field is always blank.

**Sending job's number.** The job number of the job in which the message being received was sent. Because the sending job also must be the receiving job for call message queues, this field is always blank.

**Sending module name.** For type 1 and 2, the name of the module that contains the sending message.

**Sending procedure name.** For type 1, the name of the procedure sending the message. A nested procedure name has each procedure name separated by a colon. The outermost procedure name is identified first followed by the procedures it contains. The innermost procedure is identified last in the string.

**Sending program name.** The program name or ILE program name that contains the procedure sending the message. This field is blank if the sending type is 3.

**Sending program's instruction number.** The number of the program instruction that issued the command or called the API used to send the message being received.

**Sending program's statement numbers or instruction numbers.** This field can contain up to three statement numbers or an instruction number. When the sending type is 0 or 3, this field contains an instruction number. The instruction number is 4 characters in length of original program model (OPM) programs and 6 characters in length for System Licensed Internal Code (SLIC) programs. For all other sending types, this field contains statement numbers. Each statement number can be up to 10 characters in length. The instruction number is 4 characters in length. Each statement number or instruction number is left-justified in a 10-character partition of the 30-character field. The first statement number or instruction number is in the leftmost 10 characters and the third in the rightmost 10 characters. The unused parts of this field are set to blanks.

**Sending type.** The type of the sender (whether it is a program or procedure). Possible values and their meanings are as follow:

- 0 Sender is an original program model (OPM) program or a System Licensed Internal Code (SLIC) program with up to and including 12 characters in its name.
- 1 Sender is a procedure within an ILE program, and the procedure name is up to and including 256 characters in length.
- 2 Sender is a procedure within an ILE program, and the procedure name is from 257 characters up to and including 4096 characters in length. For this type, the sending procedure name is blank and the name can be obtained from the long sending procedure name.
- 3 Sender is a SLIC program with 13 or more characters in its name. For this type, the sending program name is blank, and the name can be obtained from the long sending program name.

**Sending user profile.** The name of the user profile that sent the message being received. Because the sending job also must be the receiving job for call message queues, this field is always blank.

**Time sent.** The time at which the message being received was sent, in HHMMSS (hour, minute, second) format.

## Error Messages

Message ID	Error Message Text
CPF24A3 E	Value for call stack counter parameter not valid.
CPF24A7 E	Value for the length of message information not valid.
CPF24A8 E	Value for wait time not valid.
CPF24A9 E	Value for message action not valid.
CPF24AF E	Message key not allowed with message type specified.
CPF24B9 E	When call stack entry name is '*' or '*CTLBDY', module name and program name must be '*NONE'.
CPF24BF E	Module or bound-program name is blank.
CPF24B1 E	Message key required for message type specified.
CPF24B2 E	Message key of *TOP requires message type of *NEXT.
CPF24B3 E	Message type &1 not valid.
CPF24B4 E	Severe error while addressing parameter list.
CPF24B7 E	Value &1 for call stack entry name length not valid.
CPF24CE E	Qualifier &1 incorrect for use with pointer.
CPF24C5 E	Pointer to call stack entry not valid.
CPF24C6 E	Value of To call stack entry data type parameter not valid.
CPF2401 E	Not authorized to library &1.

CPF2407 E Message file &1 in &2 not found.

CPF2410 E Message key not found in message queue &1.

CPF2411 E Not authorized to message file &1 in &2.

CPF2415 E End of requests.

CPF2423 E Variable specified in SENDER parameter less than 80 bytes.

CPF2449 E Message that should be a reply, is not a reply.

CPF247A E Call stack entry not found.

CPF247E E CCSID &1 is not valid.

CPF2479 E Call stack entry not found.

CPF2531 E Message file &1 in &2 damaged for &3.

CPF2548 E Damage to message file &1 in &2.

CPF2551 E Message key and message type combination not valid.

CPF3CF1 E Error code parameter not valid.

CPF3C21 E Format name &1 is not valid.

CPF3C36 E Number of parameters, &1, entered for this API was not valid.

CPF3C90 E Literal value cannot be changed.

CPF8100 E All CPF81xx messages could be returned. xx is from 01 to FF.

CPF9830 E Cannot assign library &1.

CPF9872 E Program or service program &1 in library &2 ended. Reason code &3.

---

API introduced: V2R1.1

---

[Top](#) | [Message Handling APIs](#) | [APIs by category](#)

# Remove Nonprogram Messages (QMHRMVM) API

## Required Parameter Group:

1	Qualified message queue name	Input	Char(20)
2	Message key	Input	Char(4)
3	Messages to remove	Input	Char(10)
4	Error code	I/O	Char(*)

## Optional Parameter Group:

5	Allow default reply rejection	Input	Char(10)
---	-------------------------------	-------	----------

Default Public Authority: \*USE

Threadsafe: Yes

The Remove Nonprogram Messages (QMHRMVM) API removes messages from nonprogram message queues. The removed instances of the messages are no longer available for you to work with. However, other instances of the messages might still exist in other message queues, and the definitions of predefined messages are still in the message files.

You can use this API to remove a single message or a group of messages from a message queue, or to clear a message queue of all messages except unanswered inquiries.

To remove messages from program message queues, see [Remove Program Messages](#) (QMHRMVPM) API.

## Authorities and Locks

### *Message Queue Authority*

\*OBJOPR and \*DLT

### *Message Queue Library Authority*

\*EXECUTE

## Required Parameter Group

### Qualified message queue name

INPUT; CHAR(20)

The name of the message queue from which to remove messages, and the library in which it resides. The first 10 characters specify the message queue, and the second 10 characters specify the library.



You can use these special values for the library name:

*\*CURLIB* The job's current library

*\*LIBL* The library list

### Message key

INPUT; CHAR(4)

If the messages to remove parameter specifies *\*BYKEY*, the key to the single message being removed. The key is assigned by the command or API that sends the message.

If the messages to remove parameter does not specify *\*BYKEY*, use blanks for this parameter.

### Messages to remove

INPUT; CHAR(10)

The message or group of messages being removed. Valid values follow:

*\*ALL* All messages in the message queue.

*\*BYKEY* The single message specified in the message key parameter.

*\*KEEPUNANS* All messages in the message queue except unanswered inquiry and unanswered senders' copy messages.

*\*NEW* All new messages in the message queue. New messages are those that have not been received.

*\*OLD* All old messages in the message queue. Old messages are those that have already been received.

### Error code

I/O; CHAR(\*)

The structure in which to return error information. For the format of the structure, see [Error Code Parameter](#).



## Optional Parameter Group

### Allow default reply rejection

INPUT; CHAR(10)

Removing an unanswered inquiry causes the default reply to be sent to the inquiry message. This value indicates whether a reply handling exit program will be allowed to reject a default reply that is sent as a result of using this function. A reply handling exit program can be registered via the system registration facility for exit point QIBM\_QMH\_REPLY\_INQ. If the parameter is not specified, a value of *\*NO* is used. Valid values are:

*\*NO* A reply handling exit program will not be allowed to reject a default reply.

\*YES A reply handling exit program will be allowed to reject a default reply. If an exit program rejects the reply, a CPD2476 (Reply rejected by a reply handling exit program) will be sent as a diagnostic message to the program using this function. The CPD2476 will be followed by a CPF2422 (Reply not valid) escape message that the program using this function should monitor for to handle and recover from error situations.



## Error Messages

Message ID	Error Message Text
CPF24AE E	Message key and messages to remove are mutually dependent.
CPF24A6 E	Value for messages to remove not valid.
CPF24B4 E	Severe error while addressing parameter list.
CPF2401 E	Not authorized to library &1.
CPF2403 E	Message queue &1 in &2 not found.
CPF2408 E	Not authorized to message queue &1.
CPF2410 E	Message key not found in message queue &1.
»CPF2422 E	Reply not valid.◀
CPF2450 E	Work station message queue &1 not allocated to job.
CPF2451 E	Message queue &1 is allocated to another job.
CPF2477 E	Message queue &1 currently in use.
CPF3CF1 E	Error code parameter not valid.
»CPF3C3A E	Value for parameter &2 for API &1 not valid.◀
CPF3C36 E	Number of parameters, &1, entered for this API was not valid.
CPF3C90 E	Literal value cannot be changed.
CPF8127 E	&8 damage on message queue &4 in &9. VLIC log-&7.
CPF8176 E	Message queue for device description &4 damaged.
CPF9830 E	Cannot assign library &1.
CPF9838 E	User profile storage limit exceeded.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

---

API introduced: V2R1.1

---

# Remove Program Messages (QMHRMVPM) API

## Required Parameter Group:

1	Call stack entry	Input	Char(*) or Pointer
2	Call stack counter	Input	Binary(4)
3	Message key	Input	Char(4)
4	Messages to remove	Input	Char(10)
5	Error code	I/O	Char(*)

## Optional Parameter Group 1:

6	Length of call stack entry	Input	Binary(4)
7	Call stack entry qualification	Input	Char(20)
8	Remove unhandled exceptions	Input	Char(10)

## Optional Parameter Group 2:

9	Call stack entry data type	Input	Char(10)
---	----------------------------	-------	----------

Default Public Authority: \*USE

Threadsafe: Yes

The Remove Program Messages (QMHRMVPM) API removes messages from call message queues and the external message queue. The removed instances of the messages are no longer available for you to work with. However, other instances of the messages might still exist in other message queues, and the definitions of predefined messages are still in the message files.

To remove messages from nonprogram message queues, see [Remove Nonprogram Messages \(QMHRMVM\) API](#).

You can use the QMHRMVPM API to streamline program message handling. Assume that a program receives several diagnostic messages and an escape message from a called program. The program handles the escape message without receiving the diagnostic messages. The program could then use the QMHRMVPM API to remove all the new messages it has not received, including the unneeded diagnostic messages. This can prevent confusion if the program gets another escape message and must receive the diagnostic messages associated with the new escape to analyze the error.

In a multithreaded job messages can be removed from call message queues within the thread that calls this API, and messages sent from within the same thread can be removed from the external message queue. Messages on call message queues in other threads or sent to the external message queue from other threads cannot be removed.

## Authorities and Locks

None.

## Required Parameter Group

### Call stack entry

INPUT; CHAR(\*) or Pointer

The call stack entry from whose message queue the message are to be removed, or the call stack entry to start counting from when using the call stack counter parameter. The call stack entry you specify must be in the call stack or you can specify the external message queue instead of a call stack entry.

You can specify a call stack entry by providing the name of the OPM program or ILE procedure running in the entry, by providing a pointer to the call stack entry, or by using one of the following special values:

- \* The message queue of the current call stack entry (that is, the call stack entry removing the messages).
- \**ALLINACT* All message queues for inactive call stack entries. The messages to remove parameter must specify \*ALL. The call stack counter parameter is ignored.
- \**EXT* The external message queue. The call stack counter parameter is ignored.

If you specify a message key and the messages to remove parameter specifies \*BYKEY, this parameter is ignored.

If the call stack entry is to be identified by pointer, the pointer that is specified must address a valid call stack entry within the same job as the one the API is used in. Alternatively, the pointer can be set to Null. The Optional Parameter Group 1 must be used and the Length of Call stack entry parameter must be set to 16. In addition, the Optional Parameter Group 2 must also be used and the Call stack entry format parameter must be set to \*PTR.

If the pointer provided is set to Null, this indicates that the call stack entry is the one in which the API is being used.

If the pointer does not address a valid call stack entry or is not a Null pointer, the error message CPF24C5 is sent to the user of the API.

The call stack entry can be a nested procedure name from 1 through 4096 characters in length. When specifying nested procedures, each procedure name must be separated by a colon, and the outermost procedure is identified first followed by the procedures it contains. The innermost procedure is the last procedure identified in the string.

The call stack entry can be a partial name. To specify a partial name, place three less-than signs (<<<) at the beginning of the call stack entry identifier, or place three greater-than signs (>>>) at the end of the call stack entry identifier, or place both the less-than signs and the greater-than signs at their respective ends of the call stack entry identifier. The value for the call stack entry excluding the less-than signs and the greater-than signs is used to search backward through the stack for the requested call stack entry name.

When searching for a partial call stack entry name:

- If the less-than signs (<<<) are specified only at the beginning of the call stack entry name, the less-than signs are truncated and the remaining character string is right-justified. The remaining string is then compared to the current call stack entry on the call stack. The comparison starts at the end of the call stack entry name and backwardly compares the number of characters in the specified string.
- If the greater-than signs (>>>) are specified only at the end of the call stack entry name, the greater-than signs are truncated. The remaining character string is compared to the current call stack entry on the call stack. The comparison starts at position 1 of the call stack entry name and compares the number of characters in the specified string.
- If the less-than signs (<<<) are specified at the beginning of the call stack entry name and the greater-than signs (>>>) are specified at the end of the call stack entry name, both the less-than signs and the greater-than signs are truncated. The remaining characters are used to scan and to compare the entire length of the specified string and the current call stack entry on the call stack.

**Note:** If the optional parameters Length of to call stack entry and To call stack entry data type are not specified, this parameter is assumed to be CHAR(10).

### Call stack counter

INPUT; BINARY(4)

A number identifying the location in the call stack of the call stack entry from whose message queue the messages are to be removed. The number is relative to the call stack entry identified by the Call stack entry parameter. It indicates how many calls up the call stack the target entry is from the one identified by the Call stack entry parameter. Valid values follow:

- |                                |  |
|--------------------------------|--|
| <i>0</i>                       | Remove the messages from the message queue of the call stack entry specified in the Call stack entry parameter.  |
| <i>1</i>                       | Remove the messages from the message queue of the call stack entry that is one earlier than the entry identified by the Call stack entry parameter.        |
| <i>n (any positive number)</i> | Remove the messages from the message queue of the nth call stack entry up the stack from the call stack entry specified in the Call stack entry parameter. |

You can use any positive number that does not exceed the actual number of call stack entries in the call stack, excluding the external message queue.

This parameter is ignored in these cases:

- When the program message queue parameter specifies all queues for inactive call stack entries (\*ALLINACT) or the external message queue (\*EXT).
- When you specify a message key and the messages to remove parameter specifies \*BYKEY.

### Message key

INPUT; CHAR(4)

If the messages to remove parameter specifies \*BYKEY or \*SCOPE, the key to the single message being removed. (The key is assigned by the command or API that sends the message.) The call message queue and call stack counter parameters are ignored.

If the messages to remove parameter does not specify *\*BYKEY* or *\*SCOPE*, you must use blanks for this parameter.

If you know the message key of the message you want to remove, you can remove the message without regard to the call message queue that contains the message. To do this, specify the key, this parameter, the special value *\** for the call stack entry, and *0* for the call stack counter. This is useful if the message was sent to a call stack entry that is no longer in the call stack.

### Messages to remove

INPUT; CHAR(10)

The message or group of messages being removed. Valid values follow:

- \*ALL* All messages in the call message queue.
- \*BYKEY* The single message specified in the message key parameter. If the message to be removed is a *\*SCOPE* message, the *\*SCOPE* value must be used instead of *\*BYKEY*.
- \*KEEPRQS* All messages in the call message queue except requests.
- \*NEW* All new messages in the call message queue. New messages are those that have not been received.
- \*OLD* All old messages in the call message queue. Old messages are those that have already been received. The *\*SCOPE* type message identified by the message key specified in the Message key parameter.
- \*SCOPE* The scope message specified in the message key parameter.

### Error code

I/O; CHAR(\*)

The structure in which to return error information. For the format of the structure, see [Error Code Parameter](#).

## Optional Parameter Group 1

### Length of call stack entry

INPUT; BINARY(4)

The length of the value for the call stack entry parameter. Valid values for this parameter are as follows:

- 1 through 4096 if partial name indicators are not used.
- 16 if the call stack entry parameter is a pointer.
- 1 through 4102 if partial name indicators are used.

**Note:** The actual length of the call stack entry name cannot exceed 4096 characters. If this parameter is not used, the value for the call stack entry parameter is assumed to be 10 characters in length.

### Call stack entry qualification

INPUT; CHAR(20)

The name of the module and the ILE program or service program to further qualify the procedure name specified by the Call stack entry parameter. The first 10 characters specify the module name, and the second 10 characters specify the ILE program name or service program name. If this parameter is not used, only the Call stack entry parameter is used to determine the call stack entry. The following special value may be used:

*\*NONE* This value is used for the module or ILE program name or service program name, or both. When *\*NONE* is specified for one of the names, then that name is not used when searching for the call stack entry. If *\*NONE* is specified for both names, only the Call stack entry parameter is used to identify the call stack entry.

If the call stack entry is to be identified by pointer, this parameter must still be passed to the API but both the module and program qualifiers must be specified as *\*NONE*. When a pointer is used, the module and program name qualification is not applicable.

The module name and bound program name should be used only when identifying the call stack entry for an ILE procedure. When identifying the entry for an OPM program, the Optional Parameter Group 1 should either not be used or the module and program qualifiers should be specified as *\*NONE*.

### **Remove unhandled exceptions**

INPUT; CHAR(10)

Whether to remove unhandled exceptions in the identified call message queue. The exception handling support for some languages allows exceptions to not be handled. Valid values follow:

*\*YES* Remove any unhandled exceptions in the identified call message queue. This is the default when this parameter is not used.

*\*NO* Do not remove any unhandled exceptions in the identified call message queue.

## **Optional Parameter Group 2**

This parameter group is used when the Call stack entry parameter is specified as a pointer.

### **Call stack entry data type**

INPUT; CHAR(10)

Whether the value of the call stack entry parameter is a character string (a name or special value) or a pointer. Use one of the following special values;

*\*CHAR* Value of the parameter is a character string (name or special value)

*\*PTR* Value of the parameter is a pointer.

If the above optional parameter is not specified, it is assumed that the value of the call stack entry parameter is a character string.

## Error Messages

Message ID	Error Message Text
CPF24AD E	Messages to remove must be *ALL if program message queue is *ALLINACT.
CPF24AE E	Message key and messages to remove are mutually dependent.
CPF24A3 E	Value for call stack counter parameter not valid.
CPF24A6 E	Value for messages to remove not valid.
CPF24BF E	Module or bound-program name is blank.
CPF24B4 E	Severe error while addressing parameter list.
CPF24B7 E	Value &1 for call stack entry name length not valid.
CPF24B8 E	Value &1 for remove unhandled exceptions not valid.
CPF24C5 E	Pointer to call stack entry not valid.
CPF24C6 E	Value of To call stack entry data type parameter not valid.
CPF241A E	Clear option &1 in system program is not valid.
CPF2410 E	Message key not found in message queue &1.
CPF247A E	Call stack entry not found.
CPF2479 E	Call stack entry not found.
CPF3C36 E	Number of parameters, &1, entered for this API was not valid.
CPF3C90 E	Literal value cannot be changed.
CPF3CF1 E	Error code parameter not valid.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

---

API introduced: V2R1.1

---

[Top](#) | [Message Handling APIs](#) | [APIs by category](#)



# Resend Escape Message (QMHRSNEM) API

## Required Parameter Group:

1	Message key	Input	Char(4)
2	Error code	I/O	Char(*)

## Optional Parameter Group:

3	To call stack entry	Input	CHAR(*)
4	To call stack entry length	Input	Binary(4)
5	Format of to call stack entry parameter	Input	Char(8)
6	From call stack entry address	Input	CHAR(16) or Pointer
7	From call stack counter	Input	Binary(4)

Default Public Authority: \*USE

Threadsafe: Yes

The Resend Escape Message (QMHRSNEM) API resends an escape message from the current call stack entry to the previous call stack entry in the call stack or to any call stack entry that is earlier if the optional parameters are used. This API can also be used to resend an escape message from a call stack entry other than the current one.

You can use this API along with the Move Program Messages (QMHMOVPM) API to streamline exception message handling. If a call stack entry is sent diagnostic messages and an escape message but cannot handle the error itself, you can use the QMHMOVPM API to move the diagnostic messages. You can use the QMHRSNEM API to forward the escape message to the previous call stack entry in the call stack. The call stack entry does not need to send an escape message of its own. For details about the QMHMOVPM API, see [Move Program Messages \(QMHMOVPM\) API](#).

In a multithreaded job messages can be resent only from one call message queue to another call message queue within the thread that calls this API. Messages cannot be resent to a call stack entry in another thread.

## Authorities and Locks

None.

## Required Parameter Group

### Message key

INPUT; CHAR(4)

The key to the escape message being resent. The key is assigned by the command or API that first sends the message.

To resend the last new escape message, use blanks for this parameter.

A message is new until it is received. It then becomes an old message. You can resend an old message only if you specify the message key.

### **Error code**

I/O; CHAR(\*)

The structure in which to return error information. For the format of the structure, see [Error Code Parameter](#).

## **Optional Parameter Group**

Use these optional parameters if the call stack entry, to which the message is to be resent, is not just previous to the one that is using this API. Use these optional parameters also when the message to resend is not on the current call stack entry's message queue.

### **To call stack entry**

INPUT; CHAR(\*)

A structure that contains the information that this API will use to determine the call stack entry to resend the message to. The structure can be in one of two formats; RSNM0100 or RSNM0200.

### **To call stack entry length**

INPUT; BINARY(4)

The length of the To call stack entry parameter. The length specified here must be at least large enough to hold the specified format of the To call stack entry parameter. It may be larger. In case it is larger, the structure must be left justified in the area. If the length specified by this parameter is not adequate for the format being used, error CPF24C7 is sent to the caller of this API.

### **Format of to call stack entry parameter**

INPUT; CHAR(8)

The format of the to call stack entry parameter. Use one of the following format names:

*RSNM0100* The To call stack entry parameter is in the RSNM0100 format. This format is used when the to call stack entry is identified by name or by special value.

*RSNM0200* The To call stack entry parameter is in the RSNM0200 format. This format is used when the call stack entry is identified by a pointer.

### **From call stack entry address**

INPUT; Pointer or CHAR(16)

A pointer to the call stack entry where the escape message to resend exists or the call stack entry to start counting from if the From call stack counter is not 0.

The pointer that is specified must be an invocation pointer that addresses a valid call stack entry or must be set to Null.

For programming languages that do not support pointers, a Null pointer can be represented by a 16 byte variable in which the first byte is set to the special value '\*' and the remaining 15 bytes are set

to blanks.

If the pointer provided is set to Null, this indicates the current call stack entry.

If the pointer is not Null or it does not address a valid call stack entry, the error message CPF24C5 is sent to the user of the API.

### From call stack counter

INPUT; BINARY(4)

A number identifying the location in the call stack of the call stack entry from where the message is to be resent. The number is relative to the from call stack entry that is addressed by the pointer. It indicates how many calls earlier in the call stack the entry is from the one that is addressed by the pointer. Valid values follow:

*0* The message to resend exists at the call stack entry that is addressed by the pointer.

*1* The message to resend exists at the call stack entry that is one earlier in the call stack than the one addressed by the pointer.

*n (any positive number)* The message to resend exists at the nth call stack entry earlier in the stack from the one addressed by the pointer.

You can use any positive number that does not exceed the actual number of call stack entries in the call stack. Do not include the external message queue in your count.

## RSNM0100 Format

The following table shows the structure of the To call stack entry parameter when the RSNM0100 format is used. This format is used when the to call stack entry is identified by name or by special value.

The information passed in the RSNM0100 format is described in [Field Descriptions](#).

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	To call stack counter
4	4	CHAR(20)	To call stack entry qualification
24	18	BINARY(4)	Length of to call stack entry identifier
28	1C	CHAR(*)	To call stack entry identifier

## RSNM0200 Format

The following table shows the structure of the To call stack entry parameter when the RSNM0200 format is used. This format is used when the to call stack entry is identified by a pointer.

The information passed in the RSNM0200 format is described in [Field Descriptions](#).

Offset		Type	Field
Dec	Hex		

0	0	Pointer	To call stack entry address
16	10	BINARY(4)	To call stack counter
20	14	CHAR(10)	Pointer qualifier

## Field Descriptions

**Length of the to call stack entry identifier.** The length of the To call stack entry identifier. Valid values for this parameter are as follows:

- 1 to and including 4096 if partial name indicators are not used.
- 1 to and including 4102 if partial name indicators are used.

**Note:** The actual length of the call stack entry name cannot exceed 4096.

### Pointer qualifier.

Specify one of the following special values:

*\*NONE* Specify this special value if the call stack entry addressed by the pointer is the one to which the message is to be resent or the one to start counting from when using a value other than 0 for the To call stack counter parameter.

*\*PGMBDY* Specify this special value if call stack entry to resend the message to or to start counting from is an ILE program boundary. The ILE program is the one which contains the procedure that is running in the call stack entry addressed by the pointer.

If the ILE program was called using a dynamic call, using this special value with a pointer will identify the call stack entry for the PEP of that program. If a call was made using a procedure pointer, it will identify the call stack entry for the procedure that was pointed to.

If the pointer addresses a call stack entry that is running a procedure from an ILE service program, this option can be used to identify the call stack entry for the first procedure that was called in that service program.

If the call stack entry addressed by the pointer is running an OPM program, using the special value *\*PGMBDY* here will have the same effect as using *\*NONE* in most cases. A difference will occur if the OPM program called itself recursively. In this case using *\*PGMBDY* identifies the first recursion level while *\*NONE* identifies the current recursion level.

**To call stack counter.** A number identifying the location in the call stack of the call stack entry to which the message is to be resent. The number is relative to the call stack entry identified by the To call stack entry identifier field. It indicates how many calls up the call stack the target entry is from the one specified in the To call stack entry identifier field. Valid values follow:

*0* Resend the message to the message queue of call stack entry specified by the To call stack entry identifier field. You cannot use 0 when the To call stack identifier field specifies \* to designate the current call stack entry.

*1* Resend the message to the message queue of the call stack entry that is one earlier in the call stack than the one identified by the To call stack entry identifier field.

*n* (any positive number) Resend the message to the queue of the *n*th call stack entry earlier in the stack from the one specified in the to call stack entry field.

You can use any positive number that does not exceed the actual number of call stack entries in the call stack. Do not include the external message queue in your count.

The From call stack entry cannot be the same as the To call stack entry. Additionally, the To call stack entry must be earlier on the call stack than the From call stack entry.

**To call stack entry address.** A pointer to the call stack entry to resend the messages to, or the call stack entry to start counting from when using a value other than 0 for the To call stack counter parameter.

The pointer that is specified must be an invocation pointer that addresses a valid call stack entry or must be set to Null.

If the pointer provided is Null, this indicates the current call stack entry. If a Null pointer is provided, the To call stack counter parameter must be specified as a non-zero value.

If the pointer provided is not Null and does not address a valid call stack entry, the error message CPF24C5 is sent to the user of the API.

**To call stack entry identifier.** An identification of the call stack entry to resend the message to or the call stack entry to start counting from when using a value other than 0 for the To call stack counter parameter. The call stack entry identified by this field must be in the call stack, and you cannot specify the external message queue.

You can identify the To call stack entry by providing the name of the OPM program or ILE procedure running in the entry or by using one of the following special values:

\* The call stack entry of the program or procedure using the API.

*\*PGMBDY* The call stack entry is for the boundary of the specified program object. The program object is specified explicitly by providing a program name in the To call stack entry qualification field. The program object can be implicitly specified by specifying the program name qualification as *\*NONE*. In this case, the program object is assumed to be the program that is using the API.

This option essentially identifies the oldest call stack entry which began a sequence of calls, where each call in this sequence involved the same program object. The call sequence could involve recursive calls to the same program or, in the case of ILE, calls to different procedures of the same ILE program or ILE service program.

For OPM programs, in most cases using *\*PGMBDY* produces the same result as using *\** or an OPM program name directly in this field. A difference will appear when an OPM program calls itself recursively. In this case, using *\** or an OPM program name identifies the current recursion level. In contrast, using *\*PGMBDY* identifies the first recursion level.

For an ILE program, this option can be used to identify the first procedure of the ILE program that was called in the current sequence. If the ILE program was called using a dynamic call, this is the PEP (program entry procedure) of the ILE program. If sequence was started by calling using a procedure pointer, this is the call stack entry for the procedure that was pointed to.

For ILE service programs, this special value can be used to specify the call stack entry for the first procedure called in the identified service program.

*\*CTLBDY* The call stack entry at the most recent control boundary. This call stack entry is in the same activation group as the one that is using the API.

If this key word value is used and there is no control boundary in the current call stack, the error CPF24C8 is returned to the user of this API. This would happen if the only entries on the call stack are for OPM programs.

Note that in some cases, *\*PGMBDY* and *\*CTLBDY* will identify the same call stack entry but not in all cases. The option *\*CTLBDY* does not care if all call stack entries in a call sequence involve the same program object. It cares only that a sequence started at a control boundary. Conversely, the start of a call sequence identified by *\*PGMBDY* may not fall on a control boundary.

*\*PGMNAME* The call stack entry is identified entirely by the program name and optionally module name that is provided in the To call stack entry qualification field.

For OPM programs, specifying this special value here and the program name in the To call stack entry qualification field produces the same results as if the program name had been specified directly here.

For ILE programs or service programs, this special value is used to indicate that a procedure name is not being specified. Rather the call stack entry is identified by providing only the ILE program name or ILE service program name and optionally the ILE module name. This means that the call stack entry will be the most recently called procedure that is part of the specified ILE program (and part of the ILE module if module name is also specified). The name of this most recently called procedure is not important in determining the correct call stack entry.

If this key value is specified with a program name only, then the call stack entry is the most recently called OPM program that has the specified program name or the most recently called ILE procedure that is part of an ILE program or service program of the specified name, whichever is the most recent on the call stack. If the module name is also specified, then the call stack entry is the most recently called ILE procedure that is part of specified ILE program or service program and module. If module name is given then the call stack entry cannot be an OPM program.

The call stack entry can be a nested procedure name from 1 through 4096 characters in length. When specifying nested procedures, each procedure name must be separated by a colon, and the outermost procedure is identified first followed by the procedures it contains. The innermost procedure is the last procedure identified in the string.

The call stack entry can be a partial name. To specify a partial name, place three less-than signs (<<<) at the beginning of the call stack entry identifier, or place three greater-than signs (>>>) at the end of the call stack entry identifier, or place both the less-than signs and the greater-than signs at their respective ends of the call stack entry identifier. The value for the call stack entry excluding the less-than signs and the greater-than signs is used to search backward through the stack for the requested call stack entry name.

When searching for a partial call stack entry name:

- If the less-than signs (<<<) are specified only at the beginning of the call stack entry name, the less-than signs are truncated and the remaining character string is right-justified. The remaining string is then compared to the current call stack entry on the call stack. The comparison starts at the end of the call stack entry name and backwardly compares the number of characters in the specified string.
- If the greater-than signs (>>>) are specified only at the end of the call stack entry name, the greater-than signs are truncated. The remaining character string is compared to the current call

stack entry on the call stack. The comparison starts at position 1 of the call stack entry name and compares the number of characters in the specified string.

- If the less-than signs (<<<) are specified at the beginning of the call stack entry name and the greater-than signs (>>>) are specified at the end of the call stack entry name, both the less-than signs and the greater-than signs are truncated. The remaining characters are used to scan and to compare the entire length of the specified string and the current call stack entry on the call stack.

**To call stack entry qualification.** This field is used when it is necessary to further identify the To call stack entry. The parameter consists of two 10 character parts. The first part is the module name qualifier and the second part is the program name qualifier. The values provided in this field are used as a qualifier for the value provided in the To call stack entry identifier field. The values that can be specified here depend upon the value provided as the To call stack entry identifier.

The following special value may be used to indicate that the module name qualifier or program name qualifier is not being specified:

*\*NONE* This value can be used for the value of the module name qualifier, program name qualifier, or both to indicate that no qualifier is being specified.

If the To call stack entry identifier field contains an ILE procedure name then this field can contain the name of the module and program that the procedure was compiled and bound into. The module and program name qualifiers are used to distinguish the correct call stack entry in the case where different procedures of the same name are on the call stack at the same time. The first 10 characters specify the module name, and the second 10 characters specify the ILE program name or ILE Service program name. If *\*NONE* is specified for both the module and program name qualifiers, only the specified procedure name is used to determine the call stack entry.

If the To call stack entry identifier field contains the special value *\*PGMNAME* than the program name qualifier must contain either an OPM program name or an ILE program or Service program name. For ILE, the module name qualifier may contain a module name; otherwise it must contain *\*NONE*.

When the To call stack entry identifier field contains the special value *\** or *\*CTLBDY*, both the module name qualifier and program name qualifier must be specified as *\*NONE*.

When the To call stack entry identifier field contains an OPM program name, both the module name and program name qualifiers should be specified as *\*NONE*. If a module name or program name is specified here, the name specified as the To call stack entry identifier is interpreted as an ILE procedure name rather than an OPM program name. Either the entry would not be found or an incorrect entry would be found.

When the To call stack entry identifier field contains the special value *\*PGMBDY*, the module name qualifier must be specified as *\*NONE*. For the program name qualifier, an OPM, ILE program, or ILE Service program name may be specified or *\*NONE* may be used.

## Error Messages

Message ID	Error Message Text
CPF24A3 E	Value for call stack counter parameter not valid.
CPF24B4 E	Severe error while addressing parameter list.
CPF24B7 E	Value &1 for call stack entry name length not valid.

CPF24B9 E	When call stack entry name is '*' or '*CTLBDY', module name and program name must be '*NONE'.
CPF24BC E	No escape message to resend.
CPF24C5 E	Pointer to call stack entry not valid.
CPF24C7 E	The length &1 is not valid when the specified format is &2.
CPF24C8 E	Control boundary not found on call stack.
CPF24C9 E	Program boundary not found on call stack.
CPF24CA E	Call stack entry is not valid to resend message.
CPF24CB E	*PGMNAME requires a specified program name.
CPF24CC E	Call stack entry &2 for *PGMNAME not found.
CPF24CD E	Module name cannot be specified when *PGMBDY is used.
CPF24CE E	Qualifier &1 incorrect for use with pointer.
CPF24CF E	Pointer qualifier &1 incorrect for use with pointer.
CPF2410 E	Message key not found in message queue &1.
CPF246A E	Destination call stack entry not valid.
CPF247A E	Call stack entry not found.
CPF2479 E	Call stack entry not found.
CPF2524 E	Exception handler not available because of reason code &1.
CPF2550 E	Exception message sent to a deleted program or procedure.
CPF3CF1 E	Error code parameter not valid.
CPF3C21 E	Format name &1 is not valid.
CPF3C36 E	Number of parameters, &1, entered for this API was not valid.
CPF3C90 E	Literal value cannot be changed.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

---

API introduced: V2R1.1

---

[Top](#) | [Message Handling APIs](#) | [APIs by category](#)



# Retrieve Message (QMHRRTVM) API

## Required Parameter Group:

1	Message information	Output	Char(*)
2	Length of message information	Input	Binary(4)
3	Format name	Input	Char(8)
4	Message identifier	Input	Char(7)
5	Qualified message file name	Input	Char(20)
6	Replacement data	Input	Char(*)
7	Length of replacement data	Input	Binary(4)
8	Replace substitution values	Input	Char(10)
9	Return format control characters	Input	Char(10)
10	Error code	I/O	Char(*)

## Optional Parameter Group:

11	Retrieve Option	Input	Char(10)
12	CCSID to convert to	Input	Binary(4)
13	CCSID of replacement data	Input	Binary(4)

Default Public Authority: \*USE

Threadsafe: Yes

The Retrieve Message (QMHRRTVM) API retrieves the message description of a predefined message. The **message description** is created with the Add Message Description (ADDMSGD) command. It consists of the text of the message and other information, such as the message help and the default reply for the message. You can use the QMHRRTVM API to copy the text of predefined messages into a program.

Retrieving a message is not the same as receiving a message. Retrieving a message with this API returns the message and associated information stored in the message file. In contrast, receiving a message with the Receive Nonprogram Message (QMHRRCVM) or Receive Program Message (QMHRRCVPM) API focuses on the message as it is sent to a particular user or program at a particular time. The information returned from the QMHRRCVM or the QMHRRCVPM API includes details about who sent the message, when it was sent, why it was sent, and so on.

## Authorities and Locks

### *Message File Authority*

\*USE

### *Message File Library Authority*

\*EXECUTE

# Required Parameter Group

## Message information

OUTPUT; CHAR(\*)

The variable that receives information returned, in the format specified in the format name parameter, of the length specified in the length of message information parameter. If the user is retrieving the next message of the last message of a message file or the first message of an empty message file, blanks are returned for the message information field.

## Length of message information

INPUT; BINARY(4)

The size of the area to contain the message information, in bytes. The minimum size is 8.

This parameter must specify the size of the variable you use for the message information parameter. If this parameter specifies a longer size, other parts of storage could be overwritten when the API returns the information.

The API returns as much information as it can fit in this length. If the available message information is longer, it is truncated. If the available message information is shorter, the unused output area is unchanged; whatever is already stored in that space remains there.

To determine how much information the API actually returns in response to this call, see the bytes returned field in the message information format. To determine how much information the API could return if space were available, see the bytes available field.

## Format name

INPUT; CHAR(8)

The format to use for the message information. Specify one of these format names:

- RTVM0100* Brief message information. For details, see [RTVM0100 Format](#).
- RTVM0200* Similar to RTVM0100 plus message severity, alert index, alert option, log indicator, and default reply. For details, see [RTVM0200 Format](#).
- RTVM0300* Similar to RTVM0200 plus the message ID, CCSID information, and the replacement text formats. For details, see [RTVM0300 Format](#).
- RTVM0400* All message information. For details, see [RTVM0400 Format](#).

## Message identifier

INPUT; CHAR(7)

The identifying code for the predefined message being retrieved. If the retrieve option parameter is \*FIRST, this field is ignored and the first alphabetical message is retrieved from the message file.

## Qualified message file name

INPUT; CHAR(20)

The name of the message file from which to retrieve the message information and the library in which it resides. The first 10 characters specify the file name; the second 10 characters specify the library. You can use these special values for the library name:

*\*CURLIB* The job's current library

*\*LIBL* The library list

### **Replacement data**

INPUT; CHAR(\*)

The values to insert in the substitution variables in the predefined message and message help.

If you use blanks for this parameter, blanks are inserted for the substitution variables.

If this parameter contains pointer data, each pointer must start on a 16-byte boundary to keep the data accurate.

### **Length of replacement data**

INPUT; BINARY(4)

The length of the replacement data parameter, in bytes. Valid values are 0 through 32767.

### **Replace substitution variables**

INPUT; CHAR(10)

Whether to replace the substitution variables with the values given in the replacement data parameter. Specify one of these values:

*\*NO* Do not use the replacement data. Instead, return the substitution variable numbers (that is, &1, &2, and so on) in the message text.

*\*YES* Use the replacement data in the message text. If you specified blanks in the replacement data parameter, blanks are used for the substitution variables.

### **Return format control characters**

INPUT; CHAR(10)

Whether or not the format control characters are returned in the message help output field. Specify one of these values:

*\*NO* Do not return the format control characters in the text.

*\*YES* Return the format control characters in the text.

Three format control characters can be returned within the message. They are defined in the online help of the Add Message Description (ADDMSGD) command to have these meanings:

*&N* Forces the text to a new line (column 2). If the text is longer than one line, the next lines are indented to column 4 until the end of text or another format control character is found.

*&P* Forces the text to a new line indented to column 6. If the text is longer than one line, the next lines start in column 4 until the end of text or another format control character is found.

*&B* Forces the text to a new line, starting in column 4. If the text is longer than one line, the next lines are indented to column 6 until the end of text or another format control character is found.

## Error code

I/O; CHAR(\*)

The structure in which to return error information. For the format of the structure, see [Error Code Parameter](#).

## Optional Parameter Group

### Retrieve option

INPUT; CHAR(10)

If this parameter is not used, it is defaulted to \*MSGID. The following options are allowed:

- \*MSGID* Retrieve the message description specified by the message ID parameter. This is the default value of this parameter.
- \*NEXT* Retrieve the next message description in the message file that is after the message description specified on the message ID parameter.
- \*FIRST* Retrieve the first message description of the message file. Note that if \*FIRST is specified for this parameter, the message ID parameter is ignored.

**Note:** Although the last 4 characters of a message identifier can be thought of as 4 hexadecimal numbers, the values 'A' through 'F' are treated as characters when sorting. For example, CPFAAAA is listed before CPF0001.

### CCSID to convert to

INPUT; BINARY(4)

The coded character set identifier (CCSID) in which you want your message text returned. This only applies to text returned in the message and message help fields. The following values are allowed:

- 0* The retrieved message description is converted to the CCSID of the job. This is the default value if this parameter is not specified.  
  
If the job is 65535 and the text or data is something other than EBCDIC single byte or EBCDIC mixed, the text and data are converted to the default job CCSID.
- 65535* The retrieved message description will not be converted before it is returned.
- CCSID* A valid CCSID in which you want your message text returned. The CCSID must be between 1 and 65535. The CCSID will be validated by this API.  
  
Only CCSIDs that a job can be changed to are accepted. For a list of valid job CCSIDs, prompt the CCSID parameter on the Change Job (CHGJOB) CL command.

**Note:** If the text contains substitution variables, only the substitution variables that are \*CCHAR type data are converted to this CCSID. All other message data is not converted before being returned.

### CCSID of replacement data

INPUT; BINARY(4)

The coded character set identifier (CCSID) that the supplied replacement data is in. This only applies to parts of the replacement data that are defined as \*CCHAR. The \*CCHAR data is converted from this CCSID to the CCSID in which you want all your text converted. The following values are allowed:

- 0 The replacement data is assumed in the CCSID of the job. This is the default value if this parameter is not specified.
- 65535 The replacement data will not be converted.
- CCSID Specifies a valid CCSID that your replacement data is in. The CCSID must be between 1 and 65535. The CCSID is validated by this API. For a list of valid CCSIDs, see [CCSIDs: Message Support](#) in the Globalization topic.

## RTVM0100 Format

The following table lists the fields in the RTVM0100 format. For more information about each field, see [Field Descriptions](#).

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	BINARY(4)	Length of message returned
12	C	BINARY(4)	Length of message available
16	10	BINARY(4)	Length of message help returned
20	14	BINARY(4)	Length of message help available
24	18	CHAR(*)	Message
The offset to this field equals the last offset identified plus the length of the previous variable length fields.		CHAR(*)	Message help

## RTVM0200 Format

The following table lists the fields in the RTVM0200 format. For more information about each field, see [Field Descriptions](#).

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bytes returned

4	4	BINARY(4)	Bytes available
8	8	BINARY(4)	Message severity
12	C	BINARY(4)	Alert index
16	10	CHAR(9)	Alert option
25	19	CHAR(1)	Log indicator
26	1A	CHAR(2)	Reserved
28	1C	BINARY(4)	Length of default reply returned
32	20	BINARY(4)	Length of default reply available
36	24	BINARY(4)	Length of message returned
40	28	BINARY(4)	Length of message available
44	2C	BINARY(4)	Length of message help returned
48	30	BINARY(4)	Length of message help available
52	34	CHAR(*)	Default reply
The offset to this field equals the last offset identified plus the length of the previous variable length fields.		CHAR(*)	Message
		CHAR(*)	Message help

## RTVM0300 Format

The following table lists the fields in the RTVM0300 format. For more information about each field, see [Field Descriptions](#).

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	BINARY(4)	Message severity
12	C	BINARY(4)	Alert index
16	10	CHAR(9)	Alert option
25	19	CHAR(1)	Log indicator
26	1A	CHAR(7)	Message ID
33	21	CHAR(3)	Reserved
36	24	BINARY(4)	Number of substitution variable formats
40	28	BINARY(4)	CCSID conversion status indicator of text
44	2C	BINARY(4)	CCSID conversion status indicator of replacement data
48	30	BINARY(4)	CCSID of text returned
52	34	BINARY(4)	Offset of default reply
56	38	BINARY(4)	Length of default reply returned

60	3C	BINARY(4)	Length of default reply available
64	40	BINARY(4)	Offset of message
68	44	BINARY(4)	Length of message returned
72	48	BINARY(4)	Length of message available
76	4C	BINARY(4)	Offset of message help
80	50	BINARY(4)	Length of message help returned
84	54	BINARY(4)	Length of message help available
88	58	BINARY(4)	Offset of substitution variable formats
92	5C	BINARY(4)	Length of substitution variable formats returned
96	60	BINARY(4)	Length of substitution variable formats available
100	64	BINARY(4)	Length of substitution variable format element
104	68	CHAR(*)	Reserved
The offsets to these fields are specified in the previous offset variables.		CHAR(*)	Default reply
		CHAR(*)	Message
		CHAR(*)	Message help
		CHAR(*)	Substitution variable formats

## RTVM0400 Format

The following table lists the fields in the RTVM0400 format. For more information about each field, see [Field Descriptions](#).

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	BINARY(4)	Message severity
12	C	BINARY(4)	Alert index
16	10	CHAR(9)	Alert option
25	19	CHAR(1)	Log indicator
26	1A	CHAR(7)	Message ID
33	21	CHAR(3)	Reserved
36	24	BINARY(4)	Number of substitution variable formats
40	28	BINARY(4)	CCSID conversion status indicator of text
44	2C	BINARY(4)	CCSID conversion status indicator of replacement data
48	30	BINARY(4)	CCSID of text returned
52	34	BINARY(4)	Offset of default reply
56	38	BINARY(4)	Length of default reply returned
60	3C	BINARY(4)	Length of default reply available
64	40	BINARY(4)	Offset of message
68	44	BINARY(4)	Length of message returned

72	48	BINARY(4)	Length of message available
76	4C	BINARY(4)	Offset of message help
80	50	BINARY(4)	Length of message help returned
84	54	BINARY(4)	Length of message help available
88	58	BINARY(4)	Offset of substitution variable formats
92	5C	BINARY(4)	Length of substitution variable formats returned
96	60	BINARY(4)	Length of substitution variable formats available
100	64	BINARY(4)	Length of substitution variable format element
104	68	CHAR(10)	Reply type
114	72	CHAR(2)	Reserved
116	74	BINARY(4)	Maximum reply length
120	78	BINARY(4)	Maximum reply decimal positions
124	8C	BINARY(4)	Offset of valid reply value entries
128	80	BINARY(4)	Number of valid reply values entries returned
132	84	BINARY(4)	Length of valid reply value entries returned
136	88	BINARY(4)	Length of valid reply value entries available
140	8C	BINARY(4)	Length of valid reply value entry
144	90	BINARY(4)	Offset of special reply value entries
148	94	BINARY(4)	Number of special reply values returned
152	98	BINARY(4)	Length of special reply value entries returned
156	9C	BINARY(4)	Length of special reply value entries available
160	A0	BINARY(4)	Length of special reply value entry
164	A4	BINARY(4)	Offset of lower range reply value
168	A8	BINARY(4)	Length of lower range reply value returned
172	AC	BINARY(4)	Length of lower range reply value available
176	B0	BINARY(4)	Offset of upper range reply value
180	B4	BINARY(4)	Length of upper range reply value returned
184	B8	BINARY(4)	Length of upper range reply value available
188	BC	BINARY(4)	Offset of relational test entry
192	C0	BINARY(4)	Length of relational test entry returned
196	C4	BINARY(4)	Length of relational test entry available
200	C8	CHAR(7)	Message creation date
207	CF	CHAR(1)	Reserved
208	D0	BINARY(4)	Message creation level number
212	D4	CHAR(7)	Message modification date
219	DB	CHAR(1)	Reserved
220	DC	BINARY(4)	Message modification level number
224	E0	BINARY(4)	Stored CCSID of message
228	E4	BINARY(4)	Offset of dump list entries
232	E8	BINARY(4)	Number of dump list entries returned
236	EC	BINARY(4)	Length of dump list entries returned
240	F0	BINARY(4)	Length of dump list entries available



244	F4	CHAR(10)	Default program name
254	FE	CHAR(10)	Default program library name
The offsets to these fields are specified in the previous offset variables.		CHAR(*)	Default reply
		CHAR(*)	Message
		CHAR(*)	Message help
		CHAR(*)	Substitution variable formats
		CHAR(*)	Valid reply value entries
		CHAR(*)	Special reply value entries
		CHAR(*)	Lower range reply value
		CHAR(*)	Upper range reply value
		CHAR(*)	Relational test entry
		CHAR(*)	Dump list entries

## Relational Test Entry Format

The following table lists the fields in the Relational Test Entry format. For more information about each field, see [Field Descriptions](#).

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	Relational operator
10	A	CHAR(2)	Reserved
12	C	BINARY(4)	Length of relational value
16	10	CHAR(*)	Relational value

## Special Reply Value Entry Format

The following table lists the fields in the Special Reply Value Entry format. For more information about each field, see [Field Descriptions](#).

Offset		Type	Field
Dec	Hex		
0	0	CHAR(32)	From-value
32	20	CHAR(32)	To-value

## Substitution Variable Format

The following table lists the fields in the Substitution Variable format. For more information about each field, see [Field Descriptions](#).

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Length of replacement data for substitution variable
4	4	BINARY(4)	Field size or decimal positions
8	8	CHAR(10)	Substitution variable type
18	12	CHAR(*)	Reserved


## Valid Reply Entry Format

The following table lists the fields in the Valid Reply Entry format. For more information about each field, see [Field Descriptions](#).

Offset		Type	Field
Dec	Hex		
0	0	CHAR(32)	Valid reply value

## Field Descriptions

This section describes the fields returned in further detail. The fields are listed in alphabetical order.

**Alert index.** The format number for the message data field. This number is also called the resource name variable. For more information, see the [Alerts Support](#)  book.

**Alert option.** Whether and when an SNA alert is created and sent for the message. Valid values follow:

- \*DEFER* An alert is sent after local problem analysis.
- \*IMMED* An alert is sent immediately when the message is sent to a message queue that has the allow alerts attribute set to *\*YES*.
- \*NO* No alert is sent.
- \*UNATTEND* An alert is sent immediately when the system is running in unattended mode (when the value of the alert status network attribute, ALRSTS, is *\*UNATTEND*).

For more information, see the *Alerts Support* book.

**Bytes available.** The length of all available information about the format. Bytes available can be greater than the length specified in the APIs length of message information parameter. If it is greater, the information returned is truncated.

**Bytes returned.** The length of all information returned in the format. The value of the bytes returned field is always less than or equal to the value of the length of message information parameter, and less than or equal to the bytes available. If the bytes returned value is less than the length of message information value, the unused space is unchanged.

**CCSID conversion status indicator of replacement data.** This value indicates the status of any CCSID conversion that may have occurred when inserting the replacement data into the message description. The following values may be returned:

- 0 No conversion was needed because the CCSID of the data matched the CCSID you wanted the data converted to.
- 1 No conversion occurred because either the data was 65535 or the CCSID you wanted the data converted to was 65535.
- 2 No conversion occurred because you did not supply enough space for the data or the replacement data did not contain any \*CCHAR type substitution variables.
- 3 The data was converted to the CCSID specified using the best fit conversion tables.
- 4 A conversion error occurred using the best fit conversion tables so a default conversion was attempted. This completed without error.
- 1 An error occurred on both the best fit and default conversions. The data was not converted.

**CCSID conversion status indicator of text.** This value indicates the status of any CCSID conversion that may have occurred when retrieving the message description from the message file. The following values may be returned:

- 0 No conversion was needed because the CCSID of the text matched the CCSID you wanted the text converted to.
- 1 No conversion occurred because either the text was 65535 or the CCSID you wanted the text converted to was 65535.
- 2 No conversion occurred because you did not supply enough space for the text.
- 3 The text was converted to the CCSID specified using the best fit conversion tables.
- 4 A conversion error occurred using the best fit conversion tables so a default conversion was attempted. This completed without error.
- 1 An error occurred on the best fit conversion. The text was not converted.

**CCSID of text returned.** The coded character set identifier in which the text of the message description is returned.

**Note:** If a conversion error occurs or if the CCSID you requested the text to be converted to is 65535, the CCSID that the message description is stored in is returned. Otherwise the CCSID you wanted your text converted to is returned. If you do not want the text converted before it is returned but you do want to know the CCSID that the message description is stored in, specify 65535 on the coded character set identifier to convert to parameter. The CCSID the message description is stored in is returned in the coded character set identifier or text returned field.

The CCSID returned in the coded character set identifier for text returned may not apply to the replacement data substituted in the text returned. Only the \*CCHAR substitution variables are converted to the CCSID specified. The rest of the replacement data is not converted before it is returned. Also, if an error occurred trying to convert the replacement data from the CCSID of the message data to the CCSID to convert to, the

replacement data would not be converted before it is returned.

**Default program library name.** The library specified for the default program. This can be the actual library name, \*LIBL, or \*CURLIB.

**Default program name.** The name of the program called to take default action if this message is sent as an escape message to a program or procedure that is not monitoring for it.

**Default reply.** The default reply for the message identifier retrieved.

**Dump list entry.** Each dump list entry is a BINARY(4) value that specifies data to be dumped when the message is sent as an escape message to a program that is not monitoring for it. The following values may be returned:

- 1-99 The number of the message data field that is to be dumped.
- 1 The data areas of the job are dumped as specified by the Dump Job (DMPJOB) command. This corresponds to the special value \*JOBDMPL that can be specified in the DMPLST parameter of the ADDMSGD and CHGMSGD commands.
- 2 The internal machine data structures related to the machine process in which the job is running are dumped to the machine error log. This corresponds to the special value \*JOBINT that can be specified in the DMPLST parameter of the ADDMSGD and CHGMSGD commands.
- 4 The job information produced by the Display Job (DSPJOB) command is printed. This corresponds to the special value \*JOB that can be specified in the DMPLST parameter of the ADDMSGD and CHGMSGD commands.

**Field size or decimal positions.** This value is used in one of two ways, depending on the substitution variable type.

- If the substitution variable type is \*QTDCHAR, \*CHAR, \*CCHAR, \*HEX, or \*SPP and \*VARY is specified as the length of replacement data for the substitution variable, the size of the length portion of the replacement data is returned. This value is 2 if the length portion is 2 bytes long, or 4 if the length portion is 4 bytes long.
- When the substitution variable type is \*DEC, the number of decimal positions in the substitution variable is returned.
- In all other cases, 0 is returned.

**From-value.** The from-value in a special reply value entry. This is a value that can be entered as a reply to the message even though it may not meet the other reply validity checking specifications. It will be accepted and converted to a valid response specified in the to-value.

**Length (general information about the following length fields).** These formats use two types of length fields, each related to a single variable length text field. (The variable length text fields return information to the caller.) The first type of length field is returned length; the second is available length. **Returned length** is the actual length of the text in the variable length text field. **Available length** is the length of the text before it is placed in the variable length text field. It is always greater than or equal to the returned length. If the available length equals the returned length, all the message information is returned. If the text is truncated when placed in the variable length field, the available length is greater than the returned length by the number of characters truncated.

**Length of default reply available.** The length of the available default reply, in bytes.

**Length of default reply returned.** The length of the returned default reply, in bytes.

**Length of dump list entries available.** The length of the available dump list entries, in bytes.

**Length of dump list entries returned.** The length of the available dump list entries that were actually returned, in bytes.

**Length of lower range reply value available.** The length of the available lower range reply value, in bytes.

**Length of lower range reply value returned.** The length of the returned lower range reply value, in bytes.

**Length of message available.** The length of the available message text, in bytes.

**Length of message help available.** The length of the available help information for the message, in bytes.

**Length of message help returned.** The length of the returned help information for the message, in bytes.

**Length of message returned.** The length of the returned message text, in bytes.

**Length of relational test entry available.** The length of the available relational test entry, in bytes.

**Length of relational test entry returned.** The length of the returned relational test entry, in bytes.

**Length of relational value.** The length of the value to be compared to the reply entered for a relational test entry, in bytes.

**Length of replacement data for substitution variable.** The number of characters or digits that are needed in the message replacement data for this substitution variable. The value returned is dependant on the substitution variable type and length:

- -1 is returned if the length is \*VARY.
- The total number of decimal digits (including the fractional portion) is returned if the substitution variable type is \*DEC.
- In all other cases, the value returned is the size in bytes of the substitution variable.

**Length of special reply value entries available.** The total length in bytes of all the available special reply value entries.

**Length of special reply value entries returned.** The length in bytes of the special reply value entries that were actually returned.

**Length of special reply value entry.** The length of the special reply value entry, in bytes.

**Length of substitution variable format element.** The length in bytes of each substitution variable format element.

**Length of substitution variable formats available.** The total length in bytes of all the available substitution variable formats.

**Length of substitution variable formats returned.** The length in bytes of the substitution variable formats that were actually returned.

**Length of upper range reply value available.** The length of the available upper range reply value, in bytes.

**Length of upper range reply value returned.** The length of the returned upper range reply value, in bytes.

**Length of valid reply value entries available.** The total length in bytes of all the available valid reply value entries.

**Length of valid reply value entries returned.** The length in bytes of the valid reply value entries that were actually returned.

**Length of valid reply value entry.** The length of the valid reply value entry, in bytes.

**Log indicator.** The log problem indicator for the message retrieved. Possible values follow:

*N* Problems are not logged.

*Y* Problems are logged.

**Lower range reply value.** The lower value limit for the valid reply.

**Maximum reply decimal positions.** The maximum number of decimal positions allowed in the message reply.

**Maximum reply length.** The maximum length of a reply to an inquiry or notify message.

**Message.** The text of the message retrieved.

**Message creation date.** The date this the message was created, in CYYMMDD format.

**Message creation level number.** The level number of this message. This will be a value from 1 to 99.

**Message help.** The message help for the message retrieved.

**Message ID.** The message ID of the message retrieved.

**Message modification date.** The date this message was modified, in CYYMMDD format.

**Message modification level number.** The modification level number of this message. This will be a value from 1 to 99.

**Message severity.** The severity of the message retrieved.

**Number of dump list entries returned.** The number of dump list entries that were returned in the message information format. The dump list entry field is repeated once for each dump list entry.

**Number of special reply values returned.** The number of special reply values that were returned in the message information format. The special reply value entry field structure is repeated once for each special reply value.

**Number of substitution variable formats.** The number of substitution variables in the message description. The substitution variable formats field structure is repeated once for each substitution variable in the message.

**Number of valid reply values returned.** The number of valid reply values that were returned in the message information format. The valid reply value entry field structure is repeated once for each valid reply value.

**Offset of default reply.** The offset of the default reply, in bytes.

**Offset of dump list entries.** The offset of the dump list entries, in bytes.

**Offset of lower range reply value.** The offset of the lower range reply value, in bytes.

**Offset of message.** The offset of the first-level message, in bytes.

**Offset of message help.** The offset of the second-level message, in bytes.

**Offset of relational test entry.** The offset of the relational test entry, in bytes.

**Offset of special reply value entries.** The offset of the special reply value entries, in bytes.

**Offset of substitution variable formats.** The offset in bytes to the substitution variable formats.

**Offset of upper range reply value.** The offset of the upper range reply value, in bytes.

**Offset of valid reply value entries.** The offset of the valid reply value entries, in bytes.

**Relational operator.** The relational operator for a relational test entry. This value is one of the following:

*\*LT* Less than

*\*LE* Less than or equal to

*\*GT* Greater than

*\*GE* Greater than or equal to

*\*EQ* Equal to

*\*NE* Not equal to

**Relational test entry.** The relational test for the reply value that is defined in the message description. For more information, see [Relational Test Entry](#).

**Relational value.** The value to be compared to the reply entered for a relational test entry.

**Reply type.** The type of valid values that can be made to an inquiry or notify message. One of the following values will be returned:

*\*CHAR* Any character string is valid. If it is a quoted character string, the apostrophes are passed as part of the character string.

*\*NONE* No reply type is specified.

*\*DEC* Only a decimal number is a valid reply.

*\*ALPHA* Only an alphabetic string is valid. Blanks are not allowed.

*\*NAME* Only a simple name is a valid reply. The name does not have to be an object name, but it must start with an alphabetic character; the remaining characters must be alphanumeric.

**Reserved.** An ignored field.

**Special reply value entries.** Each special reply value that is defined in the message description is returned in a special reply value entry. For more information, see [Special Reply Value Entry](#).

**Stored CCSID of message.** The CCSID that was specified for the message. For messages created on a release prior to V3R1 the value 65535 will be returned.

**Substitution variable formats.** Each substitution variable that is defined in the message description is returned in a substitution variable format. For more information, see [Substitution Variable Format](#).

**Substitution variable type.** The type of data the substitution variable contains and how the data is formatted when substituted into the returned message text. The value is one of the following types:

- \*QTDCHAR** A character string formatted with enclosing apostrophes ('Monday, the 1st').
- \*CHAR** A character string formatted without enclosing apostrophes.
- \*CCHAR** A convertible character string.
- \*HEX** A string of bytes formatted as a hexadecimal value (X'C0F4').
- \*SPP** A 16-byte space pointer to data in a space object.
- \*DEC** A packed decimal number that is formatted in the message as a signed decimal value with a decimal point.
- \*BIN** A binary value that is 2, 4, or 8 bytes long (B'0000 0000 0011 1010') and is formatted in the message as a signed decimal value (58).
- \*UBIN** A binary value that is 2, 4 or 8 bytes long (B'0000 0000 0011 1010') and is formatted in the message as an unsigned decimal value (58).
- \*DTS** An 8-byte field that contains a system date/time stamp. The date/time stamp contains the date followed by one blank separator and then the time.
- \*SYP** A 16-byte system pointer to a system object.
- \*ITV** An 8-byte binary field that contains the time interval (in seconds) for wait time-out conditions.

**To-value.** The to-value in a special reply value entry. This is the value that will be used as the reply value when the value entered matches the from-value.

**Upper range reply value.** The upper value limit for the valid reply.

**Valid reply entries.** Each valid reply value that is defined in the message description is returned in a valid reply value entry. For more information, see [Valid Reply Entry](#).

**Valid reply value.** The valid reply value in a valid reply value entry.

## Error Messages

Message ID	Error Message Text
CPF24AA E	Value for replace substitution variables not valid.
CPF24AB E	Value for return format control characters not valid.
CPF24A7 E	Value for the length of message information not valid.
CPF24B4 E	Severe error while addressing parameter list.
CPF24B6 E	Length of &1, not valid for message text or data.
CPF2401 E	Not authorized to library &1.



CPF2407 E Message file &1 in &2 not found.

CPF2411 E Not authorized to message file &1 in &2.

CPF2419 E Message identifier &1 not found in message file &2 in &3.

CPF2465 E Replacement text of message &1 in &2 in &3 not valid for format specified.

CPF247E E CCSID &1 is not valid.

CPF247F E Retrieve option &1 not valid.

CPF2499 E Message identifier &1 not allowed.

CPF2531 E Message file &1 in &2 damaged for &3.

CPF2548 E Damage to message file &1 in &2.

CPF3CF1 E Error code parameter not valid.

CPF3C21 E Format name &1 is not valid.

CPF3C36 E Number of parameters, &1, entered for this API was not valid.

CPF3C90 E Literal value cannot be changed.

CPF8100 E All CPF81xx messages could be returned. xx is from 01 to FF.

CPF9830 E Cannot assign library &1.

CPF9872 E Program or service program &1 in library &2 ended. Reason code &3.

---

API introduced: V2R1.1

---

[Top](#) | [Message Handling APIs](#) | [APIs by category](#)

# Retrieve Message File Attributes (QMHRMFAT) API

Required Parameter Group:

1	Message file information	Output	Char(*)
2	Length of message file information	Input	Binary(4)
3	Format name	Input	Char(8)
4	Qualified message file name	Input	Char(20)
5	Error code	I/O	Char(*)

Default Public Authority: \*USE

Threadsafe: No

The Retrieve Message File Attributes (QMHRMFAT) API provides information about the attributes of a message file.

## Authorities and Locks

*Message file*

\*USE

*Message file library*

\*EXECUTE

## Required Parameter Group

### Message file information

OUTPUT; CHAR(\*)

The parameter to receive the message file information. The format of the returned data is specified in the format name parameter.

### Length of message file information

INPUT; BINARY(4)

The size of the area to contain the message information, in bytes. If the length specified is larger than the actual size of the message file information parameter, the results may not be predictable. The minimum length is 8 bytes.

### Format name

INPUT; CHAR(8)

The format of the information to be returned. You must use the following format name:

*RMFA0100* This format is described in [RMFA0100 Format](#).

### Qualified message file name

INPUT; CHAR(20)

The message file and the library in which the message file is located. The first 10 characters contain the message file name, and the second 10 characters contain the message file library. You can use these special values for the library name:

\**CURLIB* The job's current library

\**LIBL* The library list

### Error code

I/O; CHAR(\*)

The structure in which to return error information. For the format of the structure, see [Error Code Parameter](#).

## RMFA0100 Format

The following table shows the information returned in the message file information parameter for the RMFA0100 format. For a detailed description of each field, see [Field Descriptions](#).

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	CHAR(10)	Message file used
18	12	CHAR(10)	Message file library used
28	1C	BINARY(4)	Current storage size
32	20	BINARY(4)	Increment storage size
36	24	BINARY(4)	Number of increments
40	28	BINARY(4)	Maximum increments
44	2C	BINARY(4)	Coded character set identifier (CCSID)
48	30	CHAR(50)	Text description

## Field Descriptions

**Bytes available.** The number of bytes of data available to be returned. All available data is returned if enough space is provided.

**Bytes returned.** The number of bytes of data returned.

**Coded character set identifier (CCSID).** The coded character set identifier (CCSID) associated with this message file. A special value of 65535 means that no conversions are to occur when adding or retrieving message descriptions from this message file. A special value of 65534 means to use the CCSID associated with the message description when conversions are to occur. For more information about message handler use of CCSIDs, see [CCSIDs: Message Support](#) in the Globalization topic.

**Current storage size.** The current storage size of the message file in bytes.

**Increment storage size.** The number of bytes added each time the message file is increased in size.

**Maximum increments.** The maximum number of times the message file can be increased in size.

**Message file library used.** The actual library name that contains the message file.

**Message file used.** The name of the message file whose attributes were returned.

**Number of increments.** The number of times the message file has been increased in size.

**Reserved.** An ignored field.

**Text description.** Text that briefly describes the message file.

## Error Messages

Message ID	Error Message Text
CPF24B4 E	Severe error while addressing parameter list.
CPF2401 E	Not authorized to library &1.
CPF2407 E	Message file &1 in &2 not found.
CPF2411 E	Not authorized to message file &1 in &2.
CPF2483 E	Message file currently in use.
CPF2536 E	Value &1, for the length of message queue information not valid.
CPF3CF1 E	Error code parameter not valid.
CPF3C21 E	Format name &1 is not valid.
CPF3C90 E	Literal value cannot be changed.
CPF9830 E	Cannot assign library &1.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

---

API introduced: V3R1

---

[Top](#) | [Message Handling APIs](#) | [APIs by category](#)

# Retrieve Nonprogram Message Queue Attributes (QMHRMQAT) API

Required Parameter Group:

1	Message queue information	Output	Char(*)
2	Length of message queue information	Input	Binary(4)
3	Format name	Input	Char(8)
4	Qualified message queue name	Input	Char(20)
5	Error code	I/O	Char(*)

Default Public Authority: \*USE

Threadsafe: No

The Retrieve Nonprogram Message Queue Attributes (QMHRMQAT) API provides information about the attributes of a nonprogram message queue. For more information on the attributes of nonprogram message queues, see the Create Message Queue (CRTMSGQ) command and the Change Message Queue (CHGMSGQ) command documentation in the online help.

## Authorities and Locks

*Message queue*

\*USE

*Message queue library*

\*EXECUTE

## Required Parameter Group

### Message queue information

OUTPUT; CHAR(\*)

The parameter to receive the message queue information. The format of the returned data is specified in [RMQA0100 Format](#).

### Length of message queue information

INPUT; BINARY(4)

The size of the area to contain the message information, in bytes. If the length specified is larger than the actual size of the message queue information parameter, the results may not be predictable. The minimum length is 8 bytes.

### Format name

INPUT; CHAR(8)

The format of the information to be returned. You must use the following format name:

*RMQA0100* This format is described in [RMQA0100 Format](#).

### Qualified message queue name

INPUT; CHAR(20)

The message queue and the library in which the message queue is located. The first 10 characters contain the message queue name, and the second 10 characters contain the message queue library. You can use these special values for the library name:

\**CURLIB* The job's current library

\**LIBL* The library list

### Error code

I/O; CHAR(\*)

The structure in which to return error information. For the format of the structure, see [Error Code Parameter](#).

## RMQA0100 Format

The following table shows the information returned in the message queue information parameter for the RMQA0100 format. For a detailed description of each field, see [Field Descriptions](#).

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	CHAR(10)	Message queue used
18	12	CHAR(10)	Message queue library used
28	1C	BINARY(4)	Number of messages on queue
32	20	BINARY(4)	Current storage size
36	24	BINARY(4)	Increment storage size
40	28	BINARY(4)	Number of increments
44	2C	BINARY(4)	Maximum increments
48	30	BINARY(4)	Severity code filter
52	34	CHAR(7)	Delivery
59	3B	CHAR(10)	Break-handling program name
69	45	CHAR(10)	Break-handling program library
79	4F	CHAR(4)	Force to auxiliary storage
83	53	CHAR(50)	Text description
133	85	CHAR(1)	Allow alerts
134	86	CHAR(2)	Reserved

136	88	BINARY(4)	Coded character set identifier (CCSID)
140	8C	CHAR(10)	Message queue full action
150	96	CHAR(10)	Allow other jobs to reply

## Field Descriptions

**Allow alerts.** Whether this message queue allows alerts to be generated from alert messages sent to it. The values are:

- 1* Allow alerts to be generated
- 0* Do not allow alerts to be generated

See the [Alerts Support](#)  book for information on alerts.

**Allow other jobs to reply.** Whether other jobs can reply to messages on the message queue when it is in \*BREAK delivery mode with a break-handling program other than \*DSPMSG specified.

The field is set to \*ALWRPY if the queue is in break mode and \*DSPMSG is the break-handling program.

The field is set to blanks if the queue is not in break mode.

The possible values follow:

- \*NOALWRPY When the message queue is in \*BREAK mode with a break-handling program other than \*DSPMSG specified, other jobs can display the message queue but cannot reply to inquiry messages on the message queue.
- \*ALWRPY When the message queue is in \*BREAK mode with a break-handling program other than \*DSPMSG specified, other jobs can reply to inquiry messages on the message queue.

**Break-handling program library.** The library that is specified for the break-handling program.

If the message queue is not in break mode or the break-handling program name is specified as \*DSPMSG, this field is set to blanks.

The following special values or a specific library may be returned:

- \*LIBL The library list is used to locate the program.
- \*CURLIB The current library for the job is used to locate the program.

**Break-handling program name.** The program that is called when messages are sent to this message queue. The delivery field must be set to \*BREAK. The severity of the message sent to the queue must be equal to or greater than the value shown in the severity code filter field.

The field is set to \*DSPMSG if the queue is in break mode and there is no break-handling program set.

The field is set to blanks if the queue is not in break mode.

**Bytes available.** The number of bytes of data available to be returned. All available data is returned if enough space is provided.

**Bytes returned.** The number of bytes of data returned.

**Coded character set identifier (CCSID).** The coded character set identifier (CCSID) that is associated with this message queue. A special value of 65535 means that no conversions are to occur when sending or receiving messages from this message queue. A special value of 65534 means to use the CCSID associated with the message when conversions are to occur. For more information about the message handler use of CCSIDs, see [CCSIDs: Message Support](#) in the Globalization topic.

**Current storage size.** The current storage size of the message queue in bytes.

**Delivery.** How the messages that are sent to this message queue are delivered. The method of delivery is in effect only as long as the message queue is allocated to a job. When the queue is no longer allocated, the delivery mode is changed to \*HOLD for work-station, system-operator, and user message queues. The possible values follow:

- \*HOLD* The messages are held in the message queue until they are requested by a user or program.
- \*BREAK* When a message arrives at the message queue, the job to which the message queue is allocated is interrupted. Then the program specified on the break-handling program name field is called. If the break-handling program name is set to \*DSPMSG, the Display Message (DSPMSG) command is processed.
- \*NOTIFY* When a message arrives at the message queue, an interactive job to which the message queue is allocated is notified. The message light turns on and a buzzer sounds (if the feature is available).
- \*DFT* Messages requiring replies are answered with their default reply. No messages are added to the message queue unless the message queue is QSYSOPR.

**Force to auxiliary storage.** Whether changes made to the message queue description or messages added to or removed from the queue are immediately forced into auxiliary storage. If the value is \*YES and a system failure occurs, the changes to the message queue are not lost.

One of the following modes is returned:

- \*YES* All changes to the message queue description and to the messages in the queue are immediately forced into auxiliary storage.
- \*NO* Changes made to the message queue description, including its messages, are not immediately forced into auxiliary storage.

**Increment storage size.** The number of bytes added each time the message queue is increased in size.

**Maximum increments.** The maximum number of times the message queue can be incremented. If \*NOMAX is specified for this attribute when the message is created, the value determined by the system as the maximum number of times that the message queue can be incremented is returned.

**Message queue full action.** The action to take when the message queue becomes full. The possible values follow:

- \*SNDMSG* When the queue is full, message CPF2460 (Message queue could not be extended) will be sent to the program or user that sends a message to the full message queue.



**\*WRAP** When the queue is full, it is wrapped by removing messages from the queue to make space for new messages to be added to the queue.

**Message queue library used.** The actual library name that contains the message queue.

**Message queue used.** The name of the message queue whose attributes were returned.

**Number of increments.** The number of times that the message queue has been increased in size.

**Number of messages on queue.** The number of messages currently on the message queue.

**Severity code filter.** The lowest severity level that a message can have and still be delivered to a user in break or notify mode. Messages arriving at the message queue whose severities are lower than that specified here do not interrupt the job or turn on the message waiting light. Valid values are 0 through 99.

**Reserved.** An ignored field.

**Text description.** Text that briefly describes the message queue.

## Error Messages

Message ID	Error Message Text
CPF24B4 E	Severe error while addressing parameter list.
CPF2401 E	Not authorized to library &1.
CPF2403 E	Message queue &1 in &2 not found.
CPF2408 E	Not authorized to message queue &1.
CPF2477 E	Message queue &1 currently in use.
CPF2536 E	Value &1, for the length of message queue information not valid.
CPF3CF1 E	Error code parameter not valid.
CPF3C21 E	Format name &1 is not valid.
CPF3C90 E	Literal value cannot be changed.
CPF8127 E	&8 damage on message queue &4 in &9. VLIC log-&7.
CPF8176 E	Message queue for device description &4 damaged.
CPF9830 E	Cannot assign library &1.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

---

API introduced: V2R3

---

# Retrieve Request Message (QMHRTVRQ) API

## Required Parameter Group:

1	Message information	Output	Char(*)
2	Length of message information	Input	Binary(4)
3	Format name	Input	Char(8)
4	Message type	Input	Char(10)
5	Message key	Input	Char(4)
6	Error code	I/O	Char(*)

Default Public Authority: \*USE

Threadsafe: Yes

The Retrieve Request Message (QMHRTVRQ) API retrieves request messages from the current job's job message queue. Only request messages (commands) that have been received are retrieved; new request messages and all other types of messages are bypassed. One use for this API is in programs that support a retrieve key on a command line to retrieve commands previously run.

## Authorities and Locks

None.

## Required Parameter Group

### Message information

OUTPUT; CHAR(\*)

The variable that is to receive information about the request message. The minimum size for this area is 8 bytes. If the size of this area is smaller than the available message information, the API returns only the data that the area can hold. If no request message could be retrieved from the job message queue, no error is returned. (For example, there are no request messages prior to or after the one identified by the message key passed in.) Instead, bytes available is set to 0 to indicate no message was found.

### Length of message information

INPUT; BINARY(4)

The size of the area to contain the message information, in bytes. The minimum size is 8. If this value is larger than the actual size of the storage allocated for the message information parameter, the result may not be predictable.

### Format name

INPUT; CHAR(8)

The format of the message information to be returned.

Valid formats are:

*RTVQ0100* Basic request message information.

*RTVQ0200* All request message information.

### Message type

INPUT; CHAR(10)

The message to be retrieved.

The valid values follow:

*\*FIRST* Retrieve the first request message in the current job.

*\*LAST* Retrieve the last request message in the current job.

*\*NEXT* Retrieve the request message after the message indicated by the message key parameter. Message key is required when *\*NEXT* is used.

*\*PRV* Retrieve the request message before the message indicated by the message key parameter. Message key is required when *\*PRV* is used.

### Message key

INPUT; CHAR(4)

A value must be specified for this parameter when the message type parameter is *\*NEXT* or *\*PRV* and is used to retrieve the request message after or before the message with this key. This message key need not refer to a message on the job message queue. The search begins with the message on the queue that has the key closest to this value. This value must be blank when the message type parameter is *\*FIRST* or *\*LAST*.

### Error code

I/O; CHAR(\*)

The structure in which to return error information. For the format of the structure, see [Error Code Parameter](#).

## RTVQ0100 Format

The following table describes the information that is returned in the message information parameter for the RTVQ0100 format. For detailed descriptions of the fields, see [Field Descriptions](#).

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	CHAR(4)	Message key

12	C	CHAR(20)	Reserved
32	20	BINARY(4)	Length of request message text returned
36	24	BINARY(4)	Length of request message text available
40	28	CHAR(*)	Request message text

## RTVQ0200 Format

The following table describes the information that is returned in the message information parameter for the RTVQ0200 format. For detailed descriptions of the fields, see [Field Descriptions](#).

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	CHAR(4)	Message key
12	C	CHAR(10)	Program name or ILE service program name
22	16	CHAR(1)	Call stack entry type
23	17	CHAR(10)	Module name
33	21	CHAR(256)	Procedure name
289	121	CHAR(11)	Reserved
300	12C	BINARY(4)	Offset to long procedure name
304	130	BINARY(4)	Length of long procedure name
308	134	BINARY(4)	Length of request message text returned
312	138	BINARY(4)	Length of request message text available
316	13C	CHAR(*)	Request message text
		CHAR(*)	Long procedure name

## Field Descriptions

**Bytes available.** The length of all data available to return. All available data is returned if enough space is provided.

**Bytes returned.** The length of all data actually returned. If the data is truncated because the length of message information parameter did not specify an area large enough to hold the data, this value is less than bytes available. When no request messages could be retrieved from the job message queue, the following occurs:

- The value of the bytes returned field is set to 8.
- The value of the bytes available field is set to 0.
- The remaining fields are unchanged.

**Call stack entry type.** The type of call stack entry receiving the message.

The possible values follow:

- 0 The receiving call stack entry is an original program model (OPM) program.
- 1 The receiving call stack entry is a procedure within an ILE program, and the procedure name is up to and including 256 characters in length.
- 2 The receiving call stack entry is a procedure within an ILE program, and the procedure name is 257 through 4096 characters in length. For this type, the procedure name field is blank and the name can be obtained from the long procedure name.

**Length of long procedure name.** The length of the procedure name, in bytes. If the message was sent to an original program model (OPM) program, this field is zero.

**Length of request message text available.** The length of all available request message text, in bytes.

**Length of request message text returned.** The actual length of data in the request message text field. If the returned length is equal to the available length, all request message text was returned. If the message information parameter is not sufficiently large to hold all the text, the returned length is less than the available length and the text is truncated.

**Long procedure name.** The complete procedure name that received the message. A nested procedure name has each procedure name separated by a colon. The outermost procedure name is identified first followed by the procedures it contains. The innermost procedure is identified last in the string.

If the message was sent to an OPM program, this field is not used. The offset to long procedure name field and the length of long procedure name field are zero.

**Message key.** The key to the request message retrieved. This value can be used on subsequent calls to this API using \*NEXT or \*PRV in the message type parameter. This retrieves the next or previous request message on the job message queue. It can also be used on the Receive Program Message (QMHRCVPM) API to get more information about the request message.

**Module name.** The name of the module that contains the ILE procedure receiving the message. If the message was sent to an OPM program, this field is blank.

**Offset to long procedure name.** The offset to the procedure name, in bytes. If the message was sent to an OPM program, this field is zero. If the message information parameter is not large enough to contain any of the long procedure name field, this offset is set to zero.

**Procedure name.** The name of the ILE procedure receiving the message. When the call stack entry type is 0 or 2, this field is blank. A nested procedure name has each procedure name separated by a colon. The outermost procedure name is identified first followed by the procedures it contains. The innermost procedure is identified last in the string.

**Program name or ILE service program name.** The name of the program receiving the request message, or the name of the ILE program that contains the procedure receiving the message.

**Request message text.** The text of the request message. If you are retrieving CL commands, the maximum length of a CL command is 6000 bytes.

**Reserved.** An ignored field.

## Error Messages

<b>Message ID</b>	<b>Error Message Text</b>
CPF24AF E	Message key not allowed with message type specified.
CPF24A7 E	Value for the length of message information not valid.
CPF24B3 E	Message type &1 not valid.
CPF24B4 E	Severe error while addressing parameter list.
CPF3CF1 E	Error code parameter not valid.
CPF3C21 E	Format name &1 is not valid.
CPF3C36 E	Number of parameters, &1, entered for this API was not valid.
CPF3C90 E	Literal value cannot be changed.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

---

API introduced: V2R2

---

[Top](#) | [Message Handling APIs](#) | [APIs by category](#)

# Send Break Message (QMHSNDBM) API

## Required Parameter Group:

1	Message text	Input	Char(*)
2	Length of message text	Input	Binary(4)
3	Message type	Input	Char(10)
4	List of qualified messages	Input	Array of Char(20)
5	Number of message queues	Input	Binary(4)
6	Qualified name of the reply message queue	Input	Char(20)
7	Error code	I/O	Char(*)

## Optional Parameter:

8	Coded character set identifier	Input	Binary(4)
---	--------------------------------	-------	-----------

Default Public Authority: \*USE

Threadsafe: No

The Send Break Message (QMHSNDBM) API sends an immediate message to a work station message queue. This forces the message queue to be displayed, interrupting whatever is on the user's display. A message delivered in this way is called a **break message**. Use break messages to ensure that users see important information, such as a warning to sign off before you shut the system down for maintenance.

To keep break messages from interrupting a job, change the value of the BRKMSG parameter of the Change Job (CHGJOB) command to either \*NOTIFY or \*HOLD.

If your application attempts to diagnose and recover from errors, it might need to take additional action when using the QMHSNDBM API. The QMHSNDBM API sends a message to a list of message queues. When the API encounters an error in sending your message to a message queue in the list, it sends a diagnostic message to your program message queue. It then proceeds to the next message queue in the list. After trying to send the message to all specified message queues and if the API detected any errors, it returns a general escape message CPF2469. This message is returned as an exception or in the error code.

To diagnose and recover from these errors, your program should call the QMHRCVPM API to receive the diagnostic messages sent.

## Authorities and Locks

When the message type being sent is an inquiry, the sender needs the following authority to the reply message queue:

*Reply Message Queue Authority*

\*OBJOPR and \*ADD

*Reply Message Queue Library Authority*

\*EXECUTE

## Required Parameter Group

### Message text

INPUT; CHAR(\*)

The complete text of the immediate message being sent. You cannot include pointer data in this parameter.

**Coded Character Set Identifier (CCSID) Considerations:** The text supplied on the Message text parameter is assumed to be in the CCSID of the job executing this API unless the coded character set identifier is supplied in the CCSID parameter. For more information about message handler and its use of CCSIDs, see [CCSIDs: Message Support](#) in the Globalization topic.

### Length of message text

INPUT; BINARY(4)

The length of the message text, in bytes. Valid values are 1 through 6000.

### Message type

INPUT; CHAR(10)

The type of the message. You must specify one of these values:

- \**INFO* Informational. Conveys information *without* asking for a reply.
- \**INQ* Inquiry. Conveys information *and* asks for a reply.

### List of qualified message queue names

INPUT; ARRAY of CHAR(20)

A list of 1 through 50 work station message queues to which the message is being sent, and the libraries in which they reside.

When you send an informational message, you can specify 1 through 50 specific message queues or use this special value for the message queue name:

- \**ALLWS* All work station message queues. Use blanks for the library name, do not specify any other message queues, and specify 1 for the number of message queues parameter.

When you send an inquiry message, specify only one message queue.

When you send an informational or inquiry message to a specific message queue, use the first 10 characters for the message queue name. Use the second 10 characters for the library name.

You can use this special value for the library name:

- \**LIBL* The library list

If you send a break message to a specific work station message queue and that work station is not logged on, no error is returned. However, diagnostic message CPF2429, Message to work station did not break, appears in the job log.



### Number of message queues

INPUT; BINARY(4)

The number of message queues specified in the list of qualified message queue names parameter.

For informational (\*INFO) messages, valid values are 1 through 50.

For inquiry (\*INQ) messages or when using the special value \*ALLWS for the message queue, you must specify 1.

### Qualified name of the reply message queue

INPUT; CHAR(20)

For an inquiry message, the name of the message queue to receive the reply message, and the library in which it resides. The first 10 characters specify the message queue, and the second 10 characters specify the library. You can use this special value for the message queue name:

*\*PGMQ* The current call stack entry's call message queue. Use blanks for the library name.

You can use these special values for the library name:

*\*CURLIB* The job's current library

*\*LIBL* The library list

For an informational message, use blanks for this parameter. If you specify a message queue, the API ignores it and does not return an error.

### Error code

I/O; CHAR(\*)

The structure in which to return error information. For the format of the structure, see [Error Code Parameter](#).

## Optional Parameter

### Coded character set identifier

INPUT; BINARY(4)

The coded character set identifier (CCSID) that the supplied message text is in. The following values are allowed:

*0* The message text is assumed in the CCSID of the job running this API. This is the default value if this parameter is not specified.

*65535* The message text will not be converted.

*CCSID* Specify a valid CCSID that your message text is in. Valid values are between 1 and 65535. This API will validate the CCSID.

For a list of valid CCSIDs, see [CCSIDs: Message Support](#) in the Globalization topic.

## Error Messages

<b>Message ID</b>	<b>Error Message Text</b>
CPF24A2 E	Value for number of message queues not valid.
CPF24B3 E	Message type &1 not valid.
CPF24B4 E	Severe error while addressing parameter list.
CPF24B6 E	Length of &1, not valid for message text or data.
CPF2421 E	Message not sent. &1 in &2 not work station message queue.
CPF2428 E	Message queue parameter is not valid.
CPF2429 E	Message to work station &1 did not break.
CPF2469 E	Error occurred when sending message&1.
CPF2401 E	Not authorized to library &1.
CPF2403 E	Message queue &1 in &2 not found.
CPF2408 E	Not authorized to message queue &1.
CPF2421 E	Message not sent. &1 in &2 not work station message queue.
CPF2460 E	Message queue &1 could not be extended.
CPF2467 E	&3 message queue &1 in library &2 logically damaged.
CPF2477 E	Message queue &1 currently in use.
CPF247E E	CCSID &1 is not valid.
CPF2481 E	Work station message queue not available.
CPF8100 E	All CPF81xx messages could be returned. xx is from 01 to FF.
CPF3CF1 E	Error code parameter not valid.
CPF3C90 E	Literal value cannot be changed.
CPF9830 E	Cannot assign library &1.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

---

API introduced: V2R1

---

[Top](#) | [Message Handling APIs](#) | [APIs by category](#)

# Send Nonprogram Message (QMHSNDM) API

## Required Parameter Group:

1	Message identifier	Input	Char(7)
2	Qualified message file name	Input	Char(20)
3	Message data or immediate text	Input	Char(*)
4	Length of message data or immediate text	Input	Binary(4)
5	Message type	Input	Char(10)
6	List of qualified message queue names	Input	Array of Char(20)
7	Number of message queues	Input	Binary(4)
8	Qualified name of the reply message queue	Input	Char(20)
9	Message key	Output	Char(4)
10	Error code	I/O	Char(*)

## Optional Parameter Group:

11	Coded character set identifier	Input	Binary(4)
----	--------------------------------	-------	-----------

Default Public Authority: \*USE

Threadsafe: Yes

The Send Nonprogram Message (QMHSNDM) API sends a message to a nonprogram message queue so your program can communicate with another job or user.

To send a message to a call message queue or the external message queue, see [Send Program Message \(QMHSNDPM\) API](#).

Before coding your call to the QMHSNDM API, see [Dependencies among Parameters](#).

If your application attempts to diagnose and recover from errors, it might need to take additional action before calling the QMHSNDM API. The QMHSNDM API sends a message to a list of message queues. When the API encounters an error in sending your message to a message queue in the list, it sends a diagnostic message to your call message queue. The API then proceeds to the next message queue in the list. After trying to send the message to all specified message queues and if the API detected any errors, it returns the general escape message CPF2469. This message is sent as an exception or in the error code.

To diagnose and recover from these errors, your program should call the Receive Program Message (QMHRCVPM) API to receive the diagnostic messages sent.

## Authorities and Locks

*Message File Authority*

\*USE

*Message File Library Authority*

\*EXECUTE

*Message Queue Authority*

\*OBJOPR and \*ADD

*Message Queue Library Authority*

\*EXECUTE

*Reply Message Queue Authority*

\*OBJOPR and \*ADD

*Reply Message Queue Library Authority*

\*EXECUTE

## Required Parameter Group

### Message identifier

INPUT; CHAR(7)

The identifying code for the predefined message being sent, or blanks for an immediate message.

If you specify a message identifier, you must specify a qualified message file name. If you do not specify a message identifier, the qualified message file name parameter is ignored.

### Qualified message file name

INPUT; CHAR(20)

For a predefined message, the name of the message file and the library in which it resides. The first 10 characters specify the file name, and the second 10 characters specify the library.

You can use these special values for the library name:

\**CURLIB* The job's current library

\**LIBL* The library list

For an immediate message, use blanks for this parameter. If you specify a name, the API ignores it and does not return an error.

### Replacement data or impromptu message text

INPUT; CHAR(\*)

If a message identifier is specified, the data to insert in the predefined message's substitution variables. If no message identifier is specified, the complete text of an immediate message.

If this parameter contains pointer data, each pointer must start on a 16-byte boundary to keep the data accurate.

### Length of replacement data or impromptu message text

INPUT; BINARY(4)

The length of the message data or immediate text, in bytes. Valid values for each are:

*Replacement data* 0-32767

*Impromptu message text* 1-6000

## Message type

INPUT; CHAR(10)

The type of the message. You must specify one of these values:

\**COMP* Completion

\**DIAG* Diagnostic

\**INFO* Informational

\**INQ* Inquiry. When you send an inquiry message, a copy of the message is placed on the reply message queue. The message key returned by this API is the key to that copy. To receive the reply, use the [Receive Nonprogram Message \(QMHRVCVM\) API](#), specifying that key and a message type of reply. To receive the copy of the inquiry, specify the same key and a message type of \*COPY.

To send reply messages, see [Send Reply Message \(QMHSNDRM\) API](#).

For descriptions of the message types, see [Message Types](#). For details about coding the other parameters when sending a particular type of message, see [Dependencies among Parameters](#).

## List of qualified message queue names

INPUT; ARRAY of CHAR(20)

A list of 1 through 50 message queues to which the message is being sent, and the libraries in which they reside. When sending an inquiry message, you can list one message queue, or you can list two message queues if one of the queues is \*HSTLOG. The special value \*HSTLOG indicates the message will be sent to QSYS/QHST. When using the special value \*ALLACT, you can list only one queue. In all other cases, you can list up to 50 message queues.

You can specify user profile message queues or other nonprogram message queues, or you can use several special values.

To specify the default message queue associated with a user profile, use the first 10 characters for the user profile name. In the second 10 characters, use the special value \*USER.

To specify other nonprogram message queues, use the first 10 characters for the message queue name and the second 10 characters for the library name. You can use these special values for the library name:

\**CURLIB* The job's current library

\**LIBL* The library list

To specify other types of message queues, use one of the following special values for the first 10 characters, and leave the second 10 characters blank:

\**ALLACT* The default message queues of all active users. When you use this value, it must be the only item in the list. You cannot use this value with inquiry messages.

\**REQUESTER* In an interactive job, the current user's message queue. In a batch job, the system operator's message queue, QSYSOPR.

- \*SYSOPR* The system operator's message queue, QSYS/QSYSOPR. Any message sent to QSYSOPR automatically has a copy of the message sent to QHST.
- \*HSTLOG* The system history log message queue, QSYS/QHST. If *\*HSTLOG* is specified more than once, only one message will be sent to QSYS/QHST. If *\*HSTLOG* is specified with QSYSOPR, only one message is sent to QSYS/QHST.

### Number of message queues

INPUT; BINARY(4)

The number of message queues specified in the list of qualified message queue names parameter. Valid values are 1 through 50. When using the special value *\*ALLACT* for the message queue, you must specify 1. When sending an inquiry message, you can specify 1 or you can specify 2 if one of the values is *\*HSTLOG*.

### Qualified name of the reply message queue

INPUT; CHAR(20)

For an inquiry message only, the name of the message queue to receive the reply message, and the library in which it resides.

The first 10 characters specify the message queue, and the second 10 characters specify the library.

You can use these special values for the message queue name:

- \*PGMQ* The current call stack entry's call message queue. Use blanks for the library name.
- \*WRKSTN* The work station message queue. Use blanks for the library name. You cannot use this value when running in batch mode.

You can use these special values for the library name:

- \*CURLIB* The job's current library
- \*LIBL* The library list

For all message types except inquiry, use blanks for this parameter. If you specify a message queue, the API ignores it and does not return an error.

### Message key

OUTPUT; CHAR(4)

For an inquiry message only, the key to the sender's copy of the message in the reply message queue. This API assigns the key when it sends the message.

For all other message types, no key is returned.

### Error code

I/O; CHAR(\*)

The structure in which to return error information. For the format of the structure, see [Error Code Parameter](#).

## Optional Parameter Group

### Coded character set identifier

INPUT; BINARY(4)

The coded character set identifier (CCSID) that the supplied replacement data or impromptu message text is in. If a message identifier is specified, the text supplied by the replacement data or impromptu message text parameter that corresponds to \*CCHAR type fields is assumed to be in the CCSID supplied by this parameter. The data supplied that does not correspond to a \*CCHAR field is 65535 and is not converted. For more information about \*CCHAR type fields see the Add Message Description (ADDMSGD) command.

If no message identifier is specified, the impromptu message text supplied by the replacement data or impromptu message text parameter will all be assumed to be in the CCSID supplied by this parameter. For more information about message handler and its use of CCSIDs, see [CCSIDs: Message Support](#) in the Globalization topic. The following values are allowed:

- 0 The replacement data or impromptu message text is assumed in the CCSID of the job running this API. This is the default value if this parameter is not specified.
- 65535 The replacement data or impromptu message text will not be converted.
- CCSID Specify a valid CCSID that your replacement data or impromptu message is in. Valid values are between 1 and 65535. This API will validate the CCSID.

For a list of valid CCSIDs, see *CCSIDs: Message Support* in the Globalization topic.

## Dependencies among Parameters

You can use many different combinations of parameters when calling the QMHSNDM API. The values you specify for the message identifier and message type parameters determine which other input parameters you must specify and which you must code as blanks. They also determine whether the system returns the message key in the message key output parameter.

The following table summarizes the use of parameters for the QMHSNDM API.

Parameter	Message Type			
	*COMP, *DIAG, *INFO		*INQ	
	Predefined	Immediate	Predefined	Immediate
Message identifier	Message ID	Blank	Message ID	Blank
Qualified message file name	Required	Ignored	Required	Ignored
Message data or immediate text	Data	Text	Data	Text
Length of message data or immediate text	0-32767	1-6000	0-32767	1-6000
List of qualified message queue names	Required	Required	Required	Required
Number of message queues	1-50	1-50	1,2 if *HSTLOG	1,2 if *HSTLOG
Reply message queue	Ignored	Ignored	Required	Required
Message key	Not used	Not used	Returned	Returned
Error code	Required	Required	Required	Required

The terms used in the table to describe the parameter values are defined in the following list.

<b>Term</b>	<b>Meaning</b>
<i>Blank</i>	You must use blanks for this parameter.
<i>Data</i>	You must specify data for this parameter. The data is used as the values for a predefined message's substitution variables.
<i>Ignored</i>	The API ignores this parameter. You should use blanks for the value, but if you use another value, no error is returned.
<i>Message ID</i>	You must specify the message identifier of the predefined message being sent.
<i>Not used</i>	The API does not return data in this output parameter. The space remains unchanged.
<i>Required</i>	You must specify a valid value for this parameter.
<i>Returned</i>	The API returns data in this output parameter.
<i>Text</i>	You must specify text for this parameter. The text is used as the complete text of an immediate message.

## **Error Messages**

<b>Message ID</b>	<b>Error Message Text</b>
CPF2204 E	User profile &1 not found.
CPF24AC E	Either message identifier or message text must be specified.
CPF24A2 E	Value for number of message queues not valid.
CPF24B3 E	Message type &1 not valid.
CPF24B4 E	Severe error while addressing parameter list.
CPF24B6 E	Length of &1, not valid for message text or data.
CPF2428 E	Message queue parameter is not valid.
CPF2433 E	Function not allowed for system log message queue &1.
CPF2435 E	System reply list not found.
CPF2469 E	Error occurred when sending message&1.
CPF2401 E	Not authorized to library &1.
CPF2403 E	Message queue &1 in &2 not found.
CPF2407 E	Message file &1 in &2 not found.
CPF2408 E	Not authorized to message queue &1.
CPF2411 E	Not authorized to message file &1 in &2.
CPF2421 E	Message not sent. &1 in &2 not work station message queue.
CPF2460 E	Message queue &1 could not be extended.
CPF2467 E	&3 message queue &1 in library &2 logically damaged.



CPF2477 E Message queue &1 currently in use.  
CPF247E E CCSID &1 is not valid.  
CPF2548 E Damage to message file &1 in &2.  
CPF2557 E System reply list damaged.  
CPF2558 E System reply list currently in use.  
CPF8100 E All CPF81xx messages could be returned. xx is from 01 to FF.  
CPF2481 E Work station message queue not available.  
CPF2488 E Reply message queue \*WRKSTN not valid for batch job.  
CPF2499 E Message identifier &1 not allowed.  
CPF3C90 E Literal value cannot be changed.  
CPF3CF1 E Error code parameter not valid.  
CPF9830 E Cannot assign library &1.  
CPF9872 E Program or service program &1 in library &2 ended. Reason code &3.

---

API introduced: V3R6

---

[Top](#) | [Message Handling APIs](#) | [APIs by category](#)

# Send Program Message (QMHSNDPM) API

## Required Parameters Group:

1	Message identifier	Input	Char(7)
2	Qualified message file name	Input	Char(20)
3	Message data or immediate text	Input	Char(*)
4	Length of message data or immediate text	Input	Binary(4)
5	Message type	Input	Char(10)
6	Call stack entry	Input	Char(*) or Pointer
7	Call stack counter	Input	Binary(4)
8	Message key	Output	Char(4)
9	Error code	I/O	Char(*)

## Optional Parameter Group 1:

10	Length of call stack entry	Input	Binary(4)
11	Call stack entry qualification	Input	Char(20)
12	Display program messages screen wait time	Input	Binary(4)

## Optional Parameter Group 2:

13	Call stack entry data type	Input	Char(10)
14	Coded character set identifier	Input	Binary(4)

Default Public Authority: \*USE

Threadsafe: Yes

The Send Program Message (QMHSNDPM) API sends a message to a call message queue or the external message queue. (The external message queue is the part of the job message queue that handles messages between an interactive job and the work station user. It is not associated with a specific call stack entry.) This API allows the current call stack entry to send a message to its caller, a previous caller, or itself.

In a multithreaded job, messages can be sent only to call message queues in the thread in which this API is called or to the external message queue. Messages cannot be sent to call message queues in other threads.

To send a message to a nonprogram message queue, see [Send Nonprogram Message](#) (QMHSNDM) API.

Before coding your call to the QMHSNDPM API, see [Dependencies among Parameters](#).

## Authorities and Locks

*Message File Authority*

\*USE

*Message File Library Authority*

\*EXECUTE

## Required Parameter Group

### Message identifier

INPUT; CHAR(7)

The identifying code for the predefined message being sent, or blanks for an immediate message.

When sending an escape, notify, or status message, you must specify a message identifier. When sending a request message, you must use blanks. When sending other types of messages, you can use either a message identifier or blanks.

If you specify a message identifier, you must specify a qualified message file name. If you do not specify a message identifier, the API ignores the qualified message file name parameter.

### Qualified message file name

INPUT; CHAR(20)

For a predefined message, the name of the message file and the library in which it resides. The first 10 characters specify the file name, and the second 10 characters specify the library. You can use these special values for the library name:

\**CURLIB* The job's current library

\**LIBL* The library list

For an immediate message, use blanks for this parameter. If you specify a name, the API ignores it and does not return an error.

### Replacement data or impromptu message text

INPUT; CHAR(\*)

If a message identifier is specified, this parameter specifies the data to insert in the predefined message's substitution variables. If blanks are used for the message identifier, this parameter specifies the complete text of an immediate message.

If this parameter contains pointer data, each pointer must start on a 16-byte boundary to keep the data accurate.

### Length of replacement data or impromptu message text

INPUT; BINARY(4)

The length of the replacement data or impromptu message text, in bytes. Valid values for each are:

*Message data* 0-32767

*Immediate text* 1-6000

## Message type

INPUT;CHAR(10)

The type of the message. You must specify one of these values:

- \**COMP* Completion
- \**DIAG* Diagnostic
- \**ESCAPE* Escape
- \**INFO* Informational
- \**INQ* Inquiry. You can send inquiry messages only to the external message queue.
- \**NOTIFY* Notify
- \**RQS* Request
- \**STATUS* Status

## Call stack entry

INPUT; CHAR(\*) or Pointer

The call stack entry to send the message to, or the call stack entry to start counting from when using a value other than 0 for the Call stack counter parameter. The call stack entry you specify must be in the call stack. You can also specify the external message queue.

You can specify a call stack entry by providing the name of the OPM program or ILE procedure running in the entry, by providing a pointer to the call stack entry, or by using one of the following special values:

- \* The current call stack entry (that is, the one in which the program or procedure is using this API is running).
- \**PGMBDY* The call stack is for the boundary of the specified program object. The program object is identified explicitly by providing a program name in the optional Call stack entry qualification parameter. The program object can be specified by either not using the call stack entry qualification optional parameter or by specifying \**NONE* for the program name. In this case, the program object is assumed to be the program that is using this API.

This option essentially identifies the oldest call stack entry that began a sequence of calls, and each call in this sequence involved the same program object. The call sequence could involve recursive calls to the same program or, in the case of ILE, calls to different procedures of the same ILE program or ILE service program.

For OPM programs, in most cases using \**PGMBDY* produces the same results as using \* or an OPM program name. A difference will appear when an OPM program calls itself recursively. In this case, using \* or an OPM program name identifies the current recursion level of the program. In contrast, \**PGMBDY* identifies the first recursion level.

For an ILE program, this option can be used to identify the first procedure of the

ILE program that was called in the current sequence. If the ILE program was called using a dynamic call, this is the PEP (program entry procedure) of the ILE program. If sequence was started by calling by means of a procedure pointer, this is the call stack entry for the procedure that was pointed to.

For ILE service programs, this special value can be used to identify the call stack entry for the first procedure called in the specified service program.

*\*CTLBDY* The call stack entry at the most recent control boundary. This call stack entry is in the same activation group as the one that is using the API.

If this key word value is used and there is no control boundary in the current call stack, the error CPF24C8 is returned to the user of this API. This would happen if the only entries on the call stack are for OPM programs.

In some cases, \*PGMBDY and \*CTLBDY will identify the same call stack entry. The option \*CTLBDY does not care if all call stack entries in a call sequence involve the same program object. It cares only that a sequence started at a control boundary. Conversely, the start of a call sequence identified by \*PGMBDY may not fall on a control boundary.

*\*PGMNAME* The call stack entry is identified entirely by the program name and optionally module name that is provided by the optional Call stack entry qualification parameter.

For OPM programs, specifying this special value and the optional parameter is the same as specifying the OPM program name directly in this required parameter.

For ILE programs or ILE service programs, this special key value is used to indicate that a procedure name is not being specified. Rather the call stack entry is identified by providing only the ILE program or service program name and optionally the ILE module name. This means that the call stack entry will be the most recently called procedure that is part of the specified ILE program or service program (and part of the ILE module if module name is also specified). The name of this most recently called procedure is not important in determining the correct call stack entry.

If this key value is specified with a program name only, then the call stack entry is the most recently called OPM program that has the specified program name or the most recently called ILE procedure that is part of an ILE program or service program of the specified name, whichever is most recent on the call stack. If the module name is also specified, then the call stack entry is the most recently called ILE procedure that is part of the specified ILE program or service program and module. If module name is given then the call stack entry cannot be an OPM program.

*\*EXT* The external message queue. The Call stack counter parameter is ignored. You cannot send escape messages to this message queue. If you are sending an inquiry message, you must specify this message queue.

If the call stack entry is to be identified by pointer, the pointer that is specified must address a valid call stack entry within the same job as the one the API is used in. Alternatively, the pointer can be set to Null. The Optional Parameter Group 1 must be used and the Length of Call Stack Entry parameter must be set to 16. In addition, the Optional Parameter Group 2 must also be used and the Call Stack Entry Format parameter must be set to \*PTR.

If the pointer provided is set to Null, this indicates that the call stack entry is the one in which the API is being used.

The call stack entry can be a nested procedure name from 1 through 4096 characters in length. When specifying nested procedures, each procedure name must be separated by a colon, and the outermost procedure is identified first followed by the procedures it contains. The innermost procedure is the last procedure identified in the string.

The call stack entry can be a partial name. To specify a partial name, place three less-than signs (<<<) at the beginning of the call stack entry identifier, or place three greater-than signs (>>>) at the end of the call stack entry identifier, or place both the less-than signs and the greater-than signs at their respective ends of the call stack entry identifier. The value for the call stack entry excluding the less-than signs and the greater-than signs is used to search backward through the stack for the requested call stack entry name.

When searching for a partial call stack entry name:

- If the less than signs (<<<) are specified only at the beginning of the call stack entry name, the less-than signs are truncated and the remaining character string is right-justified. The remaining string is then compared to the current call stack entry on the call stack. The comparison starts at the end of the call stack entry name and backwardly compares the number of characters in the specified string.
- If the greater-than signs (>>>) are specified only at the end of the call stack entry name, the greater-than signs are truncated. The remaining character string is compared to the current call stack entry on the call stack. The comparison starts at position 1 of the call stack entry name and compares the number of characters in the specified string.
- If the less-than signs (<<<) are specified at the beginning of the call stack entry name and the greater-than signs (>>>) are specified at the end of the call stack entry name, both the less-than signs and the greater-than signs are truncated. The remaining characters are used to scan and to compare the entire length of the specified string and the current call stack entry on the call stack.

**Note:** If the optional parameters Length of to call stack entry and To call stack entry data type are not specified, this parameter is assumed to be CHAR(10).

### Call stack counter

INPUT; BINARY(4)

A number identifying the location in the call stack of the call stack entry to whose message queue the message is to be sent. The number is relative to the call stack entry identified by the Call stack entry parameter. It indicates how many calls up the call stack the target entry is from the one identified by the Call stack entry parameter. Valid values follow:

- |                                |  |
|--------------------------------|--|
| <i>0</i>                       | Send the messages to the message queue of the entry specified by the Call stack entry parameter.   |
| <i>1</i>                       | Send the messages to the message queue of the call stack entry one earlier on the stack than the one identified by the Call stack entry parameter. |
| <i>n (any positive number)</i> | Send the messages to the queue of the nth call stack entry earlier in the stack from the one identified by the Call stack entry parameter.         |

You can use any positive number that does not exceed the actual number of call stack entries in the call stack. Do not include the external message queue in your count.

## Message key

OUTPUT; CHAR(4)

The key to the message being sent. This API assigns the key when it sends a message of any type except status. For a status message, no key is returned and this parameter is not changed.

For inquiry and notify messages, this is the key to the sender's copy of the message in the sending program's message queue.

## Error code

I/O; CHAR(\*)

The structure in which to return error information. For the format of the structure, see [Error Code Parameter](#).

**Note:** The API does not return an error code for the message type of \*STATUS if the qualified message file name parameter is specified incorrectly, unless the call stack entry parameter is the special value \*EXT.

# Optional Parameter Group 1

## Length of call stack entry

INPUT; BINARY(4)

The length of the value for the Call stack entry parameter. Valid values for this parameter are as follows:

- 1 through 4096 if partial name indicators are not used.
- 16 if the call stack entry parameter is a pointer.
- 1 through 4102 if partial name indicators are used.

**Note:** The actual length of the call stack entry name cannot exceed 4096 characters. If this parameter is not used, the value for the call stack entry parameter is assumed to be 10 characters in length.

## Call stack entry qualification

INPUT; CHAR(20)

This parameter is used when it is necessary to further identify the call stack entry. The parameter consists of two 10 character parts. The first part is the module name qualifier and the second part is the program name qualifier. The values provided in this parameter are used as a qualifier for the value provided in the Call stack entry parameter. The values that can be specified here depend upon the value provided in the Call stack entry parameter.

The following special value may be used to indicate that the module name qualifier or program name qualifier is not being specified:

- \*NONE This value can be used for the value of the module name qualifier, program name qualifier, or both to indicate that no qualifier is being specified.

If the Call stack entry parameter contains an ILE procedure name then this parameter can contain the name of the module and program that the procedure was compiled and bound into. The module

and program name qualifiers are used to distinguish the correct call stack entry in the case where different procedures of the same name are on the call stack at the same time. The first 10 characters specify the module name, and the second 10 characters specify the ILE program or Service program name. If \*NONE is specified for both the module and program name qualifiers, only the specified procedure name is used to determine the call stack entry.

If the Call stack entry parameter contains the key value \*PGMNAME, then the program name qualifier must contain either an OPM program name or an ILE program or service program name. For ILE, the module name qualifier may contain a module name; otherwise it must contain \*NONE.

When the Call stack entry parameter specifies the special value \* or \*CTLBDY, both the module name qualifier and program name qualifier must be specified as \*NONE.

When the Call stack entry parameter specifies an OPM program name, both the module name and program name qualifiers should be specified as \*NONE. If a module name or program name is specified here, the name specified in the Call stack entry parameter is interpreted as an ILE procedure name rather than an OPM program name. Either the entry would not be found or an incorrect entry would be found.

When the Call stack entry parameter specifies the special value \*PGMBDY, the module name qualifier must be specified as \*NONE. For the program name qualifier, an OPM, ILE program, or ILE Service program name may be specified or \*NONE may be used.

If the Call stack entry is identified by pointer, this parameter must still be passed to the API. The value specified for the module name qualifier must be set to \*NONE. The program name qualifier serves as a qualifier for the pointer and must be specified as one of the following special values:

*\*NONE* The call stack entry addressed by the pointer is the one to which the message is to be sent or the one to start counting from when using a value other than 0 for the Call stack counter parameter.

*\*PGMBDY* The call stack entry to send the message to or to start counting from is an ILE program or Service program boundary. The ILE program or Service program is the one which contains the procedure that is running in the call stack entry addressed by the pointer.

If the ILE program was called using a dynamic call, using this special value with a pointer will identify the call stack entry for the PEP of that program. If the call was made using a procedure pointer, it will identify the call stack entry for the procedure that was pointed to.

If the pointer addresses a call stack entry that is running a procedure from an ILE service program, this option can be used to identify the call stack entry for the first procedure that was called in that service program.

If the call stack entry addressed by the pointer is running an OPM program than using the special value \*PGMBDY here will have the same effect as using \*NONE in most cases. A difference will occur if the OPM program called itself recursively. In this case using \*PGMBDY identifies the first recursion level while \*NONE identifies the current recursion level.

### **Display program messages screen wait time**

INPUT; BINARY(4)

The amount of time in seconds to wait on the display program messages screen for any user action



when a message is sent to an external message queue. If there is no user action within the specified time, the control returns to the statement following the call of this API. Valid values follow:

- 1 Wait on the display program messages screen for the maximum length of time allowed by the system (24 hours). If this parameter is not specified, this is the default.
- 0 Send the message, but do not show the display program messages screen.
- >0 The amount of time to wait in seconds for user action. If there is no user action within the time, then control returns. If the display file currently being displayed is defined with the restore display attribute of \*NO, the Display Program Messages screen wait time parameter is ignored.

## Optional Parameter Group 2

### Call stack entry data type

INPUT; CHAR(10)

The value of the Call stack entry parameter is a character string (a name or special value) or a pointer.

Use one of the following special values:

- \*CHAR Value of the parameter is a character string (name or special value)
- \*PTR Value of the parameter is a pointer.

If the above optional parameter is not specified, it is assumed that the value of the Call stack entry parameter is a character string.

### Coded character set identifier

INPUT; BINARY(4)

The coded character set identifier (CCSID) that the supplied replacement data or impromptu message text is in. If a message identifier is specified, the data supplied by the replacement data or impromptu message text parameter that corresponds to \*CCHAR type fields is assumed to be in the CCSID supplied by this parameter. The data supplied that does not correspond to a \*CCHAR field is 65535 and will not be converted. For more information about \*CCHAR type fields see the Add Message Description (ADDMSGD) command.

If no message identifier is specified, the text supplied by the message data or immediate text parameter will all be assumed to be in the CCSID supplied by this parameter. For more information about message handler and its use of CCSIDs, see [CCSIDs: Message Support](#) in the Globalization topic. The following values are allowed:

- 0 The message data or immediate text is assumed in the CCSID of the job running this API. This is the default value if this parameter is not specified.
- 65535 The message data or immediate text will not be converted.
- CCSID Specify a valid CCSID that your message data or immediate text is in. Valid values are between 1 and 65535. This API will validate the CCSID.

## Dependencies among Parameters

You can use many different combinations of parameters when calling the QMHSNDPM API. The values you specify for the message identifier and message type parameters determine which other input parameters you must specify and which you must code as blanks. They also determine whether the system returns the message key in the message key output parameter.

## Additional Information about Message Types

The following sections discuss additional considerations in using the different types of messages that you can specify when calling the QMHSNDPM API.

**Completion and Diagnostic Messages:** You can send completion and diagnostic messages as immediate messages or with message identifiers to call message queues or to the external message queue.

**Escape Messages:** You can send escape messages only with a message identifier and only to a call message queue. You cannot send them to the external message queue.

An escape message sent to a previous caller's call message queue ends the current call stack entry. Sending the escape message causes all the call stack entries up to but not including the call stack entry receiving the escape message to end. For example, if three call stack entries are active and the third call stack entry sends an escape message to the first, then the second and third call stack entries are ended.

**Informational Messages:** You can send informational messages as immediate messages or with message identifiers to call message queues or to the external message queue.

When you send an informational or inquiry message to the external message queue and the job is interactive, the message is displayed on the work station of the current job. The user sees the Display Program Messages display. Sending messages of these types to the external message queue is the only way to make the Display Program Messages display appear.

**Inquiry Messages:** You can send inquiry messages only to the external message queue. You cannot send them to call message queues. The sending call stack entry would have to end before the receiving call stack entry could receive the message and send a reply. At that point, the sending call stack entry is no longer active, so it cannot receive a reply.

When you send an inquiry message, it is displayed at the work station as described in [Informational Messages](#). In addition, a copy of the message is placed in the reply message queue, which is always the sending call stack entry's call message queue. Sending an inquiry message lets your call stack entry receive the reply to the message by using the message key returned by this API.

An inquiry message sent to the external message queue requires either the default reply or a reply from an interactive user. Call stack entries cannot reply to inquiry messages sent to the external message queue. In batch mode, the default reply is automatically sent as the reply.

The QMHSNDPM API returns the message key to the copy of the inquiry message in the reply message queue. You can receive the reply to the inquiry message by using the Receive Program Message (QMHRVPM) API specifying this key and the message type reply. You can receive the copy of the original inquiry message by specifying this key and the message type of \*COPY. You cannot use this key to receive the original message.

**Notify Messages** You can send notify messages only with a message identifier. You can send them to a call message queue or to the external message queue.

The reply message queue is always the sending call stack entry's call message queue (\*PGMQ). This API returns the message key to the copy of the message in the reply message queue. To receive the reply, use the Receive Program Message (QMHRVPM) API specifying this key and a message type of reply (\*RPY).

When a notify message is sent to a call message queue, the action taken depends on two things:

- Whether the receiving call stack entry monitors for the message.
- Whether the receiving call stack entry has specified an external exception handler to handle the message.

When a notify message is sent to the external message queue, the action taken is the same as if an inquiry message were sent, with this exception. Notify messages are not responded to by the system reply list.

<i>NOTIFY Message Results</i>			
<b>Receiving Call Stack Entry</b>	<b>Sending Call Stack Entry</b>	<b>Default Reply</b>	<b>Action Taken after Notify Message Is Sent</b>
Monitor defined with external handler	Does not end	Not sent	External handler is called
Monitor defined and no external handler	Ends	Not sent	Control is returned to receiving call stack entry
No monitor defined	Does not end	Sent	Control is returned to sending call stack entry

In an interactive job where a notify message is sent to \*EXT, the user must take action to send any reply, including the default reply. In a batch job, the default reply is automatically sent because programs cannot reply to notify messages sent to the external message queue.

**Request Messages:** You can send a request message only as an immediate message. You can send it to a call message queue or to the external message queue.

Request messages sent to the external message queue are never displayed on the Display Program Messages display. The system program QCMD receives them. Before the Command Entry display appears, the QCMD program attempts to perform the commands contained in the request messages. The QCMD program performs this function only with request messages on the external message queue; it does not work with request messages on other queues.

**Status Messages:** You must send status messages with a message identifier. You can send them to a call message queue or the external message queue.

When you send status messages to the external message queue, the system displays them at the bottom of the current job's display station. This shows the status of running programs. You can use this to reassure users that long programs are still running.

Status messages appear only on the display. Because they are never put in message queues and they do not have message keys, you cannot receive them.

Status messages are predefined messages. However, you can send the user what appears to be an immediate status message by using a message type of status and message CPF9898. CPF9898 is in message file QCPFMSG in library QSYS. This message consists only of a substitution variable; thus, it uses the message data that you supply as the full message text. To use it, specify CPF9898 in the message identifier parameter and the text you want to send in the replacement data or impromptu text parameter. To clear the message from the bottom of the display, you can send CPI9801, which only contains a blank line as the message.

## Error Messages

<b>Message ID</b>	<b>Error Message Text</b>
CPF24C5 E	Pointer to call stack entry not valid.
CPF24C6 E	Value of To call stack entry data type parameter not valid.
CPF24C8 E	Control boundary not found on call stack.
CPF24C9 E	Program boundary not found on call stack.
CPF24CB E	*PGMNAME requires a specified program name.
CPF24CC E	Call stack entry &2 for *PGMNAME not found.
CPF24CD E	Module name cannot be specified when *PGMBDY is used.
CPF24CE E	Qualifier &1 incorrect for use with pointer.
CPF24AC E	Either message identifier or message text must be specified.
CPF24AD E	Messages to remove must be *ALL if program message queue is *ALLINACT.
CPF24A3 E	Value for call stack counter parameter not valid.
CPF24BF E	Module or bound-program name is blank.
CPF24B3 E	Message type &1 not valid.
CPF24B4 E	Severe error while addressing parameter list.
CPF24B6 E	Length of &1, not valid for message text or data.
CPF24B7 E	Value &1 for call stack entry name length not valid.
CPF24B9 E	When call stack entry name is '*' or '*CTLBDY', module name and program name must be '*NONE'.
CPF24C2 E	Value &1 for display wait time not valid.
CPF2401 E	Not authorized to library &1.
CPF2407 E	Message file &1 in &2 not found.
CPF2409 E	Specified message type not valid with specified program message queue.
CPF2411 E	Not authorized to message file &1 in &2.
CPF2435 E	System reply list not found.
CPF246A E	Destination call stack entry not valid.
CPF2467 E	&3 message queue &1 in library &2 logically damaged.
CPF2469 E	Error occurred when sending message&1.
CPF247A E	Call stack entry not found.
CPF2479 E	Call stack entry not found.
CPF247E E	CCSID &1 is not valid.
CPF2489 E	Message identifier required for *NOTIFY, *ESCAPE, or *STATUS message types.
CPF2493 E	Message identifier not allowed with message type *RQS.
CPF2499 E	Message identifier &1 not allowed.
CPF2524 E	Exception handler not available because of reason code &1.
CPF2531 E	Message file &1 in &2 damaged for &3.

CPF2547 E      Damage to message file QCPFMSG.  
CPF2548 E      Damage to message file &1 in &2.  
CPF2557 E      System reply list damaged.  
CPF2558 E      System reply list currently in use.  
CPF3C90 E      Literal value cannot be changed.  
CPF3CF1 E      Error code parameter not valid.  
CPF8100 E      All CPF81xx messages could be returned. xx is from 01 to FF.  
CPF9830 E      Cannot assign library &1.  
CPF9845 E      Error occurred while opening file &1.  
CPF9846 E      Error while processing file &1 in library &2.  
CPF9847 E      Error occurred while closing file &1 in library &2.  
CPF9872 E      Program or service program &1 in library &2 ended. Reason code &3.

---

API introduced: V2R1.1

---

[Top](#) | [Message Handling APIs](#) | [APIs by category](#)

# Send Reply Message (QMHSNDRM) API

## Required Parameter Group:

1	Message key	Input	Char(4)
2	Qualified message queue name	Input	Char(20)
3	Reply text	Input	Char(*)
4	Length of reply text	Input	Binary(4)
5	Remove message	Input	Char(10)
6	Error code	I/O	Char(*)

## Optional Parameter Group 1:

7	Coded character set identifier	Input	Binary(4)
---	--------------------------------	-------	-----------

## Optional Parameter Group 2:

8	Allow default reply rejection	Input	Char(10)
---	-------------------------------	-------	----------

Default Public Authority: \*USE

Threadsafe: Yes

The Send Reply Message (QMHSNDRM) API sends a reply message to the sender of an inquiry message.

If your application attempts to diagnose and recover from errors, it might need to take additional action when using the QMHSNDRM API. When the QMHSNDRM API encounters errors in the reply being sent, it sends a diagnostic message describing each error to your program message queue. After the last diagnostic message, the QMHSNDRM API sends a general escape message, CPF2422.

To diagnose and recover from these errors, your application should call the QMHRCVPM API to receive the diagnostic messages sent.

## Authorities and Locks

### *Message File Authority*

\*USE

### *Message File Library Authority*

\*EXECUTE

### *Message Queue Authority*

\*USE and \*ADD. Also need \*DLT to remove a message.

### *Message Queue Library Authority*


\*EXECUTE

# Required Parameter Group

## Message key

INPUT; CHAR(4)

The key to the inquiry message that the reply answers. The key is assigned by the command or API that sends the message. You can obtain the key in these ways:

- By receiving an inquiry message with any of the following:
  - Receive Message (RCVMSG) command
  - Receive Program Message (QMHRCVPM) API
  - Receive Nonprogram Message (QMHRCVM) API
  - List Messages (QMHLSTM) API
  - List Job Log (QMHLJOBL) API
- Through a break-handling program. See the [CL Programming](#)  book for more information.

## Qualified message queue name

INPUT; CHAR(20)



The name of the message queue containing the inquiry message being answered, and the library in which it resides. The message queue is the one in which the inquiry--not the sender's copy--is located. The first 10 characters specify the message queue, and the second 10 characters specify the library. You can use these special values for the library name:

*\*CURLIB* The job's current library

*\*LIBL* The library list

## Reply text

INPUT; CHAR(\*)

The complete text of the reply being sent. To send the default reply stored in the message description, use blanks for this parameter.  A reply handling exit program can be used to limit the use of default replies. See the allow default reply rejection parameter below for more information. 

## Length of reply text

INPUT; BINARY(4)

The length of the reply text, in bytes. If you use blanks in the reply text parameter to indicate that you are using the default reply, you can specify any valid value for this parameter. Valid values are 1 through 132.

## Remove message

INPUT; CHAR(10)

Whether the inquiry message and the reply are removed from the message queue after the reply is sent. Valid values follow:

*\*NO* Keep the inquiry message and the reply.

*\*YES* Remove the inquiry message and the reply.

### **Error code**

I/O; CHAR(\*)

The structure in which to return error information. For the format of the structure, see [Error Code Parameter](#).

## **Optional Parameter Group 1**

### **Coded character set identifier**

INPUT; BINARY(4)

The coded character set identifier (CCSID) that the supplied message reply text is in. The following values are allowed:

*0* The message reply text is assumed to be in the CCSID of the job running this API. This is the default value if this parameter is not specified.

*65535* The message reply text will not be converted.

*CCSID* Specify a valid CCSID that your message reply text is in. Valid values are between 1 and 65535. This API will validate the CCSID.

For a list of valid CCSIDs, see [CCSIDs: Message Support](#) in the Globalization topic.



## **Optional Parameter Group 2**

### **Allow default reply rejection**

INPUT; CHAR(10)

Specifies whether a reply handling exit program will be allowed to reject a default reply. A default reply is requested by using blanks as the value for the reply text parameter. A reply handling exit program can be registered using the system registration facility for exit point QIBM\_QMH\_REPLY\_INQ. If this parameter is not specified, a value of *\*NO* is used. Valid values are:

*\*NO* A reply handling exit program will not be allowed to reject a default reply.

*\*YES* A reply handling exit program will be allowed to reject a default reply. If an exit program rejects the reply, a CPD2476 (Reply rejected by a reply handling exit program) will be sent as a diagnostic message to the program using this function. The CPD2476 will be followed by a CPF2422 (Reply not valid) escape message that the program using this function should monitor for to handle and recover from error situations.



**Note:** If a default reply is not being sent, this parameter is ignored and a reply handling exit program can reject or replace the reply value.



## Coded Character Set Identifier (CCSID) Considerations

If the inquiry message that this reply is being sent to is an impromptu message, the text supplied on the reply text parameter is assumed to be in the CCSID of the job executing this API unless the coded character set identifier is supplied in the CCSID parameter. If the inquiry message that this reply is being sent to is a predefined message, then the text supplied on the reply text parameter is considered 65535 and is not converted. For more information about message handler and its use of CCSIDs, see *CCSIDs: Message Support* in the Globalization topic.

## Error Messages

Message ID	Error Message Text
CPF24A4 E	Value for remove message not valid.
CPF24B4 E	Severe error while addressing parameter list.
CPF24B6 E	Length of &1, not valid for message text or data.
CPF2401 E	Not authorized to library &1.
CPF2403 E	Message queue &1 in &2 not found.
CPF2408 E	Not authorized to message queue &1.
CPF2410 E	Message key not found in message queue &1.
CPF2420 E	Reply already sent for inquiry or notify message.
CPF2422 E	Reply not valid.
CPF2432 E	Cannot send reply to message type other than *INQ or *NOTIFY.
CPF2433 E	Function not allowed for system log message queue &1.
CPF2439 E	Reply value entered is not valid.
CPF2440 E	Reply must be in range of &1 to &2.
CPF2442 E	Reply does not meet specified relations.
CPF2460 E	Message queue &1 could not be extended.
CPF2466 E	Reply length greater than &1.
CPF247E E	CCSID &1 is not valid.
CPF2477 E	Message queue &1 currently in use.
CPF2547 E	Damage to message file QCPFMSG.
CPF2548 E	Damage to message file &1 in &2.

CPF3CF1 E      Error code parameter not valid.

» CPF3C3A E      Value for parameter &2 for API &1 not valid.«

CPF3C36 E      Number of parameters, &1, entered for this API was not valid.

CPF3C90 E      Literal value cannot be changed.

CPF8100 E      All CPF81xx messages could be returned. xx is from 01 to FF.

CPF9830 E      Cannot assign library &1.

CPF9838 E      User profile storage limit exceeded.

CPF9872 E      Program or service program &1 in library &2 ended. Reason code &3.

---

API introduced: V2R1.1

---

[Top](#) | [Message Handling APIs](#) | [APIs by category](#)

# Send Scope Message (QMHSNDSM) API

Required Parameter Group:

1	Scope type	Input	Char(10)
2	Qualified scope program name	Input	Char(20)
3	Scope program data	Input	Char(*)
4	Scope program data length	Input	Binary(4)
5	Message key	Output	Char(4)
6	Error code	I/O	Char(*)

Default Public Authority: \*USE

Threadsafe: Yes

The Send Scope Message (QMHSNDSM) API sends a scope message to a call stack entry.

Scope messages are a way to call a program when the call stack entry that called the OMHSNDSM API ends or when the job using this API ends. The exit can be normal or abnormal. The three scope types, \*EXT, \*PGM, and \*CSE, indicate when the scope handling program should be called. To have the scope handling program run at job ending, specify \*EXT for the scope type parameter. The difference between program scoping (\*PGM) and call stack entry (\*CSE) scoping is how they handle a transfer control. When a transfer control occurs, the scope handling program is called for scope type \*PGM because the program or ILE procedure ended, but the scope handling program is not called for scope type \*CSE because the call stack is still active.

When a scope message is sent in a multithreaded job with a scope type of \*PGM or \*CSE, the scope handling program runs in the thread that used the OMHSNDSM API to send the scope message. When a multithreaded job uses a scope type of \*EXT, the scope handling program runs in the initial thread when the job ends.

You must also specify a scope-handling program to be run when the job, program or call stack entry exits. You must have at least \*EXECUTE authority to the scope-handling program library and at least \*USE authority to the program. If the user has below the minimum authorization, the scope message is sent but an authorization violation is signaled when the scope-handling program is called and it is not run. Any adopted authority the sending program may have does not affect the user's authority to the scope-handling program. The scope-handling program runs in the same state as the sender of the scope message.

This API returns the message key which can be used to end scoping by deleting the message.

## Authorities and Locks

*Scope program*

\*USE

*Scope program library*

\*EXECUTE

# Required Parameter Group

## Scope type

INPUT; CHAR(10)

The type of scoping to be done. Valid values are:

- \**EXT* Scope handling program is called when the caller's job ends.
- \**PGM* The scope-handling program is called when the caller's program ends or a transfer control is issued.
- \**CSE* The scope handling program is called only when the call stack entry ends. A transfer control does **not** cause the scope handling program to be called.

## Qualified scope program name

INPUT; CHAR(20)

The name of the scope handling program and the library in which it resides. The first ten characters represent the program name, the second ten characters the library name.

The library can have these special values:

- \**LIBL* The library list is used to find the scope handling program
- \**CURLIB* The current library is used to find the scope handling program

## Scope program data

INPUT; CHAR(\*)

Data that is passed to the scope handling program as its one and only parameter.

The user is responsible for the format of this data. Any pointers contained in the data must begin on a 16-byte boundary.

This data is copied to another area in storage when the scope message is sent. When the scope program is called, a space pointer pointing to the copied data is passed to the scope handling program.

## Scope program data length

INPUT; BINARY(4)

The length, in bytes, of the scope program data. The valid range is 0 through 32 767. If a data length of zero is specified, a null pointer is passed to the scope handling program.

## Message key

OUTPUT; CHAR(4)

The key to the scope message being sent.

## Error code

I/O; CHAR(\*)

The structure in which to return error information. For the format of the structure, see [Error Code Parameter](#).

## Error Messages

<b>Message ID</b>	<b>Error Message Text</b>
CPF24B4 E	Severe error while addressing parameter list.
CPF2401 E	Not authorized to library &1.
CPF3CF1 E	Error code parameter not valid.
CPF3C1D E	Length specified in parameter &1 not valid.
CPF3C90 E	Literal value cannot be changed.
CPF9801 E	Object &2 in library &3 not found.
CPF9805 E	Object &2 in library &3 destroyed.
CPF9810 E	Library &1 not found.
CPF9830 E	Cannot assign library &1.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

---

API introduced: V2R1


---

[Top](#) | [Message Handling APIs](#) | [APIs by category](#)

# Break Handling Exit Program

## Required Parameter Group:

1	Message queue name	Input	Char(10)
2	Message queue library name	Input	Char(10)
3	Message key	Input	Char(4)

The Break Handling exit program provides the capability of handling messages that arrive at a message queue that is in \*BREAK mode. A break handling program is defined to the system by specifying both break delivery and the name of the program on the same Change Message Queue (CHGMSGQ) command. A break handling program should remove the message after it is handled. Additional information and an example are provided in the [CL Programming](#)  book.

## Required Parameter Group

### Message queue name

INPUT; CHAR(10)

The name of the message queue to which the message was sent. The message queue name is a 10-character field that is left-justified and padded with blanks to the right, if necessary.

### Message queue library name

INPUT; CHAR(10)

The name of the library that contains the message queue. The library name is a 10-character field that is left-justified and padded with blanks to the right, if necessary.

### Message key

INPUT; CHAR(4)

The message reference key of the message that was sent.

## Usage Notes

This exit program could be called from an exit point within a multithreaded job and should be written to be threadsafe.


---

Exit program introduced: V3R6

---

# Default Handling Exit Program

Required Parameter Group:			
1	Receiving program information	Input	Char(* )
2	Message key	Input	Char(4)
QSYSINC Member Name: EMHDFTPG			

The Default Handling exit program provides a default message handling action that can be used if the program to which an escape message is sent does not monitor for the message. Additional information and an example are provided in the [CL Programming](#)  book. A default handling program is defined to the system through the Add Message Description (ADDMSGD) command and the Change Message Description (CHGMSGD) command.

## Required Parameter Group

### Receiving program information

INPUT; CHAR(\*)

The name of the call message queue to which the message was sent (this name is the same name as that of the program that did not monitor for the escape message). The structure for this parameter is described in [Format for Receiving Program Information](#).

### Message key

INPUT; CHAR(4)

The message reference key of the escape message on the call message queue.

## Format for Receiving Program Information

For detailed descriptions of each field, see the [Field Descriptions](#).

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	Receiving program name
10	A	CHAR(10)	Receiving module name
20	14	CHAR(256)	Receiving procedure name
276	114	CHAR(1)	Receiving program type
277	115	CHAR(3)	Reserved
280	118	BINARY(4)	Offset to long receiving procedure name
284	11C	BINARY(4)	Length of long receiving procedure name
288	120	CHAR(*)	Reserved

The offset to this field is found in the offset field identified above.	CHAR(*)	Long receiving procedure name
---	---------	-------------------------------

## Field Descriptions

**Length of long receiving procedure name.** For receiving program types 1 and 2, the length of the receiving procedure name, in bytes.

**Long receiving procedure name.** For receiving program types 1 and 2, the complete procedure name that did not monitor for the message. A nested procedure name has each procedure name separated by a colon. The outermost procedure name is identified first followed by its contained procedures. The innermost procedure is identified last in the string.

**Offset to long receiving procedure name.** When the receiving program type is 1 or 2, the offset to the long receiving procedure name, in bytes, from the beginning of this parameter structure.

**Receiving module name.** For receiving program types 1 and 2, the name of the module that did not monitor for the message. For type 0, this field contains blanks.

**Receiving procedure name.** For receiving program type 1, the name of the procedure that did not monitor for the message. A nested procedure name has each procedure name separated by a colon. The outermost procedure name is identified first followed by its contained procedures. The innermost procedure is identified last in the string. For types 0 and 2, this field contains blanks.

**Receiving program name.** The name of the program that did not monitor for the message.

**Receiving program type.** The type of program that received the message (that is, that did not monitor for the message). Valid values follow:

- 0 The message was sent to an OPM program.
- 1 The message was sent to a procedure within an ILE program, and the procedure name is 256 characters or less.
- 2 The message was sent to a procedure within an ILE program, and the procedure name is from 257 through 4096 characters. For this type, the receiving procedure name is blank, and the name can be obtained from the long receiving procedure name.

**Reserved.** An ignored field.

## Usage Notes

This exit program could be called from an exit point within a multithreaded job and should be written to be threadsafe.

---

Exit program introduced: V3R6

---



## » Reply Handling Exit Program

Required Parameter Group:

1	Type of call	Input	Binary(4)
2	Qualified message queue name	Input	Char(20)
3	Message key	Input	Char(4)
4	Message identifier	Input	Char(7)
5	Reply	I/O	Char(*)
6	Length of the reply	I/O	Binary(4)
7	CCSID of the reply	I/O	Binary(4)
8	Reply action return code	Output	Binary(4)

Exit Point Name: QIBM\_QMH\_REPLY\_INQ

Exit Point Format Name: RPYI0100

A reply handling exit program is called when a reply is sent to an inquiry message. The operating system calls the user-written exit program identified through the registration facility. The program is called after the system validates the reply, but before the reply is sent to the inquiry message. For some inquiry messages, a reply handling exit program can indicate that the reply should be rejected. When an exit program indicates that the reply should be rejected, a diagnostic message (CPD2476 Reply rejected by a reply handling exit program) is sent to the caller of the send reply function. An exit program can also replace the reply value to be sent. The first parameter sent to the exit program indicates the various reasons for which the exit program can be called.

The exit point supports an unlimited number of exit programs. If multiple exit programs are defined, all of the exit programs are called for each reply to an inquiry message until one of the exit programs rejects the reply or replaces the reply value. When a reply is rejected or replaced, any exit program that was previously notified of the reply is called again to be informed the reply is now being rejected or replaced. See [Required Parameters](#) for details on the types of calls the exit program must handle and the valid return codes that can be specified to return to the exit point. For information about adding or removing an exit program for an exit point, see [Registration Facility](#) documentation.

The exit program can use the receive message function to obtain additional information about the message being replied to. For example, the program can obtain the message data, the sender of the message, and so on. If the exit program attempts to obtain information from the sending job, it needs to take into account that the job that sent the inquiry could have ended before the inquiry is replied to, thus requiring the exit program to handle a not-found error. The exit program is executed within the same job as the user that entered a reply. It can use commands or APIs to obtain information about the current job and current user that provided the reply.

## Authorities and Locks

*User Profile Authority*

\*ALLOBJ and \*SECADM to add or remove exit programs in the registration facility.

# Required Parameter Group

## Type of call

INPUT; BINARY(4)

The reason the exit program is being called. The valid values are:

- 0 **Reply notification--no action allowed.** Informs an exit program that a reply is being sent to an inquiry message. The original reply value has been replaced by another exit program so any remaining exit programs are informed of the reply with this value, but the reply cannot be rejected or replaced. The return code parameter is ignored by the exit point.
- 1 **Reply validation requested.** The exit program should set the return code parameter to indicate whether the reply should be accepted, rejected, or replaced. A return code parameter with a value other than 0 or 2 will cause the reply to be accepted.
- 2 **Default reply validation requested.** The default reply is being sent to an inquiry message. The exit program should set the return code parameter to indicate whether the reply should be accepted or replaced. The exit program can set the return code parameter to indicate rejected, but this will cause the inquiry message to be left as unanswered (and possibly cause the job to hang). A return code parameter with a value other than 0 or 2 will cause the reply to be accepted.
- 3 **Default reply notification.** Informs an exit program that an inquiry message is being replied to with the default reply and the reply cannot be rejected, but the reply value can be replaced by an exit program. A return code parameter value other than 2 is ignored by the exit point and the reply is accepted. If a user or job requested a function that sends a default reply and the reply is not allowed to be rejected, this value would be sent to an exit program. Examples of when this value would be used:
  - deleting a message queue that contains unanswered inquiry messages.
  - sending the default reply to an unanswered inquiry and the parameter to allow default reply rejection was not specified.
  - sending an inquiry to a message queue that is in default mode.
  - sending an inquiry to \*EXT so it shows the Display Program Messages screen and the user presses F3, F12 or Enter to exit the screen without entering any reply.
  - sending an inquiry to \*EXT in a batch job.
  - a job uses an inquiry message reply job attribute of \*DFT.
  - wrapping a message queue.
- 4 **Reply rejected notification.** Informs an exit program that another exit program rejected the reply to an inquiry message. The return code parameter is ignored by the exit point and a reply is not sent to the inquiry message. When an exit program rejects a reply, any exit programs that previously accepted the reply are notified that a reply was rejected with the use of this value. Before the exit program is called with this value, CPD2476 (Reply rejected by a reply handling exit program) is sent as a diagnostic message to the caller of the send reply function. This enables an exit program to determine the exit program that rejected the reply. Any remaining exit programs will not be called.
- 5 **Replaced reply not valid.** Informs an exit program that the reply value it specified for replacement is not valid. The return code parameter will be ignored by the exit point and the original reply value, now specified in parameters 5 through 7, will be used as the reply value for the message. The next reply handling exit program will be called or the reply will be sent if there are no other exit programs to call.

- 6 **Reply replaced notification.** Informs an exit program, that has already accepted the reply, that another exit program replaced that reply. The return code parameter is ignored by the exit point. The reply cannot be rejected nor replaced. Parameters 5 through 7 contain the new reply value to be sent to the inquiry message.
- 7 **Reply cannot be sent notification.** The return code parameter is ignored by the exit point. The reply is not sent to the inquiry message. When an exit program accepts a reply, but then the system is not able to send that reply to the inquiry message, this value will be used to notify all the exit programs. This may indicate that something is wrong with the queue. After exit programs are notified, an error message is sent by the system reply function as it ends or returns.

**Note:** For values 0, 4, 5, and 6, parameter 8 will be ignored. For values 4 and 7, parameter 5 will be blank and parameters 6 and 7 will be zero.

### **Qualified message queue name**

INPUT; CHAR(20)

The name of the message queue and library containing the inquiry message. The message queue name is a 10-character field that is left-justified and padded with blanks if the name is less than 10 characters. The 10-character message queue name is followed by the 10-character library name that is padded with blanks to the right, if necessary. A special value of \*EXT will be used for the message queue name with blanks for the library name to indicate the inquiry message was sent to the job's external message queue.

### **Message key**

INPUT; CHAR(4)

The message reference key of the inquiry message that is being replied to.

### **Message identifier**

INPUT; CHAR(7)

The message identifier of the inquiry message that is being replied to. This value will be blank for an impromptu message.

### **Reply**

I/O; CHAR(\*)

The reply to be sent to an inquiry message. If a reply is being replaced by an exit program, the new reply value is specified in this parameter.

### **Length of the reply**

I/O; BINARY(4)

The length of the specified reply. The maximum value is 132. If a reply is being replaced by an exit program, the length of the new reply value is specified in this parameter.

### **CCSID of the reply**

I/O; BINARY(4)

The coded character set identifier of the specified reply. If a reply is being replaced, the CCSID of the reply is specified in this parameter.

### **Reply action return code**

OUTPUT; BINARY(4)

The return code to indicate whether the reply should be rejected, accepted, or replaced. The valid

values are:

- 0 The reply should be rejected. Any exit program that previously accepted the reply will be notified of the reply rejection. Any remaining exit programs will not be called.
- 1 The reply should be accepted. The next exit program will be called or the reply will be sent if there are no other exit programs.
- 2 The reply should be replaced using the values provided in parameters 5 through 7, reply, length of the reply, and CCSID of the reply. If the replaced reply value is not valid, or if the reply value is the same as the original reply value, the exit point will act as if the exit program accepted the original value and the next exit program will be called or the original reply will be sent if there are no other exit programs. If the replaced reply is valid, any exit program that previously accepted the reply will be notified of the reply replacement. The remaining reply handling exit programs will be called but they will not be allowed to reject or replace the reply. If there are no remaining exit programs to call, the new reply value will be sent.

**Note:** A value other than 0 or 2 will cause the reply to be accepted.

## Usage Notes

1. A reply handling exit program can send a message other than an inquiry and reply, but the program will get into a recursive loop if it tries to do functions like the following that affect inquiry messages and their reply:
  - send a reply to the inquiry message that it is being called to handle.
  - send a reply to any other inquiry message.
  - receive an unanswered inquiry message with the remove option set to yes.
  - remove an unanswered inquiry message.
  - send an inquiry message to \*EXT.

For example, a user enters a reply, so a reply handling exit program is called. The exit program sends a reply to the inquiry (or tries to remove the message, but the default reply must be sent first); this calls the exit program again, causing a loop.

2. The time between checking if there is a reply for an inquiry and actually sending the reply to the inquiry message will be greater with the use of reply handling exit programs. If the message queue is not already allocated to the job in which the reply is being sent, when the reply is sent it is implicitly allocated internally for the duration of the send. This means that if there are a number of exit programs to call or if an exit program runs for a long time, other users or jobs trying to use the message queue can see CPF2477 (message queue in use) more often if the exit programs could not complete during the internal lock wait time built into message handling functions.
3. Error situations:
  - If an error occurs while retrieving the list of exit programs for the reply handling exit point, the error will be left in the joblog and the reply will be accepted as if no exit programs were specified for the exit point.
  - A reply will be accepted if a reply handling exit program:
    - cannot be found
    - specifies a return code that is not valid

- sends any error messages
  - tries to replace a reply value with a reply that is not valid
  - Any message resulting from calling the exit program (such as program not found) or any message issued by the exit program will be left in the joblog. No new or additional message will be sent to indicate that an error occurred or that the return code was not valid.
4. The send reply function is threadsafe. A reply handling exit program can be called from within a multithreaded job. When a reply handling exit program is added to the registration facility, it can specify a multithreaded job action. One of the following actions will be taken in a multithreaded job, depending on the value indicated by the registration facility:
- 1 The exit program will be called.
  - 2 The exit program will be called and then CPI3C80 (an exit program was called in a multithreaded job) will be issued as an informational message to the caller of the send reply function for each exit program with this value. If an exit program is called again for a reply being rejected by another exit program, the CPI3C80 will not be resent.
  - 3 The exit program will NOT be called and CPF3C80 (an exit program was not called in a multithreaded job) will be issued as an informational message to the caller of the send reply function for each exit program with this value. For exit programs with this value, the exit point will act as if the reply was accepted by the exit program and continue the function of calling the next exit program or sending the reply.



---

Exit program introduced: V5R2

---

[Top](#) | [Message Handling APIs](#) | [APIs by category](#)