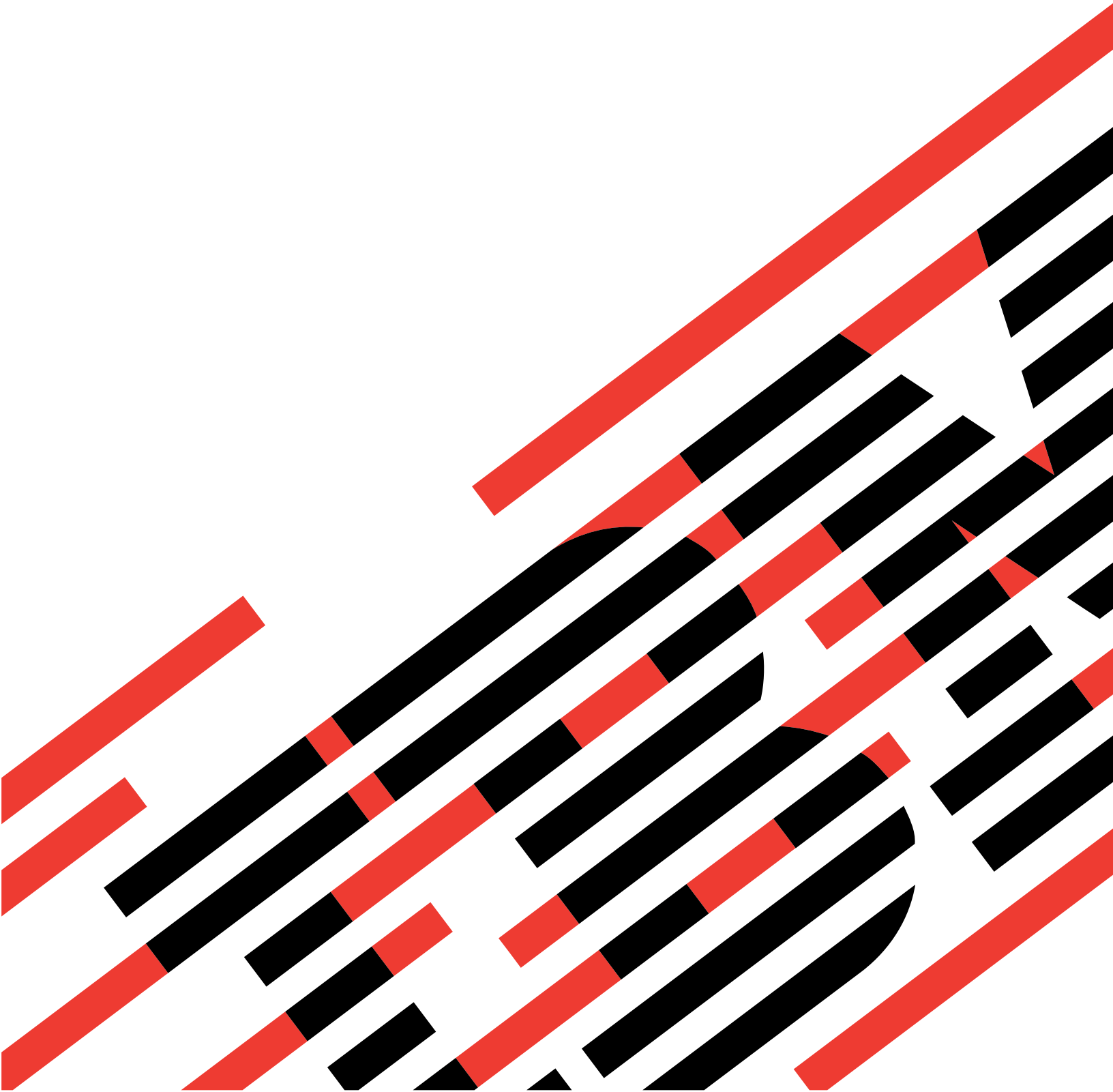
IBM Systems - iSeries

Performance Management APIs

Version 5 Release 4



IBM Systems - iSeries

Performance Management APIs

Version 5 Release 4

Note

Before using this information and the product it supports, be sure to read the information in "Notices," on page 109.

Fifth Edition (May 2004)

This edition applies to version 5, release 4, modification 0 of IBM i5/OS (product number 5722-SS1) and to all subsequent releases and modifications until otherwise indicated in new editions. This version does not run on all reduced instruction set computer (RISC) models nor does it run on CISC models.

© Copyright International Business Machines Corporation 1998, 2006. All rights reserved.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

© Copyright IBM Corp. 1998, 2006

Performance Management APIs

The performance management APIs allow you to collect and manage performance data using Collection Services, performance collector, performance explorer (PEX), and IBM(R) Performance Management @server iSeries(TM) APIs.

The performance management APIs include:

- "Collection Services APIs"
- "Performance Collector APIs" on page 44
- "Performance Explorer (PEX) APIs" on page 91
- "IBM Performance Management eServer iSeries APIs" on page 100

For additional information, see the Performance topic.

APIs by category

APIs

These are the APIs for this category.

Collection Services APIs

For information about Collection Services, see Collection Services.

The Collection Services APIs include:

- "Collector APIs"
- "Management Collection Object APIs" on page 22
- "User-Defined Transaction APIs" on page 38

The Collection Services exit program is:

- "Collection Services Data Collection Exit Program" on page 104 is called by Collection Services to collect performance data for a user-defined performance category.

Top | "Performance Management APIs" | APIs by category

Collector APIs

The collector APIs provide services to manage collections. These APIs:

- Start, end, and cycle collections
- Change and retrieve system parameters for the data collected
- Register and deregister a user-defined data category
- Add and remove collector notification

The collector APIs include:

- "Add Collector Notification (QypsAddCollectorNotification) API" on page 2 (QypsAddCollectorNotification) registers with a collector to provide notifications to a specified data queue for a collection event.

© Copyright IBM Corp. 1998, 2006

- "Change System Collector Attributes (QYPSCSCA, QypsChgSysCollectorAttributes) API" on page 6 (QYPSCSCA, QypsChgSysCollectorAttributes) changes system collection attributes. System attributes provide the default values for each collector. These include the collection interval in seconds, the library where the data is to be stored, the retention period for data, the cycle time, the cycle interval, the companion job flag, and the name of the default collection definition.
- "Cycle Collector (QYPSCYCC, QypsCycleCollector) API" on page 8 (QYPSCYCC, QypsCycleCollector) closes current collection objects and opens new collection objects.
- "Deregister Collector Data Category (QypsDeregCollectorDataCategory) API" on page 9 (QypsDeregCollectorDataCategory) removes a user-defined data category from the Collection Services function of Management Central.
- "End Collector (QYPSENDC, QypsEndCollector) API" on page 10 (QYPSENDC, QypsEndCollector) ends a specified collector.
- "Register Collector Data Category (QypsRegCollectorDataCategory) API" on page 11 (QypsRegCollectorDataCategory) adds a user-defined data category to one or more collector definitions of the Collection Services function of Management Central.
- "Remove Collector Notification (QypsRmvCollectorNotification) API" on page 17 (QypsRmvCollectorNotification) removes a notification registration from a collector.
- "Retrieve System Collector Attributes (QYPSRSCA, QypsRtvSysCollectorAttributes) API" on page 19 (QYPSRSCA, QypsRtvSysCollectorAttributes) retrieves system collection attributes. These include the collection interval in seconds, the library where the data is to be stored, the retention period for data, the cycle time, the cycle interval, the companion job flag, the name of the default collection definition, and the currently running collection definition, if any.
- "Start Collector (QYPSSTRC, QypsStartCollector) API" on page 21 (QYPSSTRC, QypsStartCollector) starts a specified collector.

Top | "Performance Management APIs," on page 1 | APIs by category

Add Collector Notification (QypsAddCollectorNotification) API

Required Parameter Group:

1	Collector name	I	Char(10)
2	Qualified data queue name	I	Char(20)
3	Notification type	I	Binary(4)
4	Category list	I	Array of Char(10)
5	Category count	I	Binary(4)
6	Error Code	I/O	Char(*)

Service Program Name: QYPSCOLL
Default Public Authority: *EXCLUDE
Threadsafe: Yes

The Add Collector Notification (QypsAddCollectorNotification) API registers with a collector to provide notifications to a specified data queue for a collection event. A collection event occurs when:

- The collector cycle interval is reached.
- The collector is ended.
- The default data collection interval is reached.

When a collector is ended, notifications are removed. When a collector is started, no notifications are registered.

Authorities and Locks

API Public Authority
*EXCLUDE

Data Queue Authority
*CHANGE

Library Authority
*EXECUTE

Required Parameter Group

Collector name
INPUT; CHAR(10)

The name of the collector that is adding a notification. One of these special values must be used:

**PFR* Performance Collector

Qualified data queue name
INPUT; CHAR(20)

The data queue used to send the event notification. The first ten characters contain the data queue name, and the second ten characters contain the data queue library name. The data queue must already exist, and the user profile running the API must have *CHANGE authority to it. You can use these special values for the library name:

**CURLIB* The job's current library.
**LIBL* The library list.

Notification type
INPUT; BINARY(4)

Notification is to be sent to the specified data queue when one of these events occur:

0 Collector - notify when a cycle, end, or interval event occurs.
1 Cycle - notify when the collection cycle interval occurs.
2 End - notify when the collection is ended.
3 Interval - notify when the collection interval occurs.
4 Category - notify when the category interval occurs.

For more information on the format of the notification record, see "Notification Record Format" on page 4.

Category list
INPUT; ARRAY OF CHAR(10)

List of category names, for which notification is to be sent. This field is only applicable when Notification type is set to category notification (4). Category name can be a system-defined category name or a user-defined category name.

System-defined category name
A 10 character name of a system-defined category. For the *PFR collector system-defined categories are:

- *APPN
- *CMNBASE
- *CMNSAP

- *CMNSTN
- *DISK
- *HDWCFG
- *IOPBASE
- *IPCS
- *JOBMI
- *JOBOS
- *LCLRSP
- *POOL
- *POOLTUNE
- *SNA
- *SNADS
- *SUBSYSTEM
- *SYSBUS
- *SYSCPU
- *SYSLVL
- *TCPBASE
- *TCPIFC
- *USRTNS

Registered user-defined category name
A 10 character name of a user-defined category registered by the Register Collector Data Category (QypsRegCollectorDataCategory) API.

Category count
INPUT; BINARY(4)

The number of categories entered in input field Category list. This field is only applicable when Notification type is set to category notification (4). Category count must have a value of '0' when Notification type is not a category notification (4).

Error code
I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

Usage Notes

An application uses the Add Collector Notification (QypsAddCollectorNotification) API to start receiving notifications about collector events. The collector sends notifications using the data queue. The application program has to create a data queue and pass the qualified name of this data queue to the Add Collector Notification (QypsAddCollectorNotification) API. The collector sends notification messages or records to the specified data queue. The format of the notification records is shown below. The application program is responsible for reading and removing entries from the data queue.

When the application no longer needs to receive notifications from the collector, it uses the Remove Collector Notification (QypsRmvCollectorNotification) API.

The application is responsible for cleaning up the data queue.

Notification Record Format

For detailed descriptions of the fields in this table, see "Notification Record Field Descriptions" on page 5.

Offset		Type	Field
Dec	Hex		
0	0	Char(10)	Entry type
10	A	Char(2)	Entry identifier
12	C	Char(10)	Collection object name
22	16	Char(10)	Library name
32	20	Char(8)	Sequence identifier
40	28	Char(10)	Category
50	32	Char(40)	Reserved

Notification Record Field Descriptions

Category The category associated with the notification. This field is only applicable when the notification event specified in Entry identifier is set to category notification ('04').

Collection object name The name of the collection object where any data collected was placed. This name is generated when the collector starts , and is of the form QDDDHHMMSS.

Entry identifier The notification event that occurred. Values are:

'01' Cycle - a collection cycle interval occurred.
'02' End - a collection has ended.
'03' Interval - a collection interval occurred.
'04' Category - a category collection interval occurred.

Entry type The type of this data queue entry. Set to value '*COLNOT '.

Library name The name of the library containing the collection object for which the event occurred.

Reserved This space is reserved for possible future use.

Sequence identifier A unique identifier assigned to this collection event. Its format is based on the time since the collector was started in the format DDHHMMSS , where 00000000 represents the time the collector was started.

Error Messages

Message ID	Error Message Text
CPF3C1E E	Required parameter &1 omitted.
CPF3C3C E	Value for parameter &1 not valid.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9801 E	Object &2 in library &3 not found.
CPF9802 E	Not authorized to object &2 in &3.
CPF9810 E	Library &1 not found.
CPF9820 E	Not authorized to use library &1.
CPFB94A E	Collector communications error. Reason code &1.

Introduced: V5R2

Top | "Performance Management APIs," on page 1 | APIs by category

Change System Collector Attributes (QYPSCSCA, QypsChgSysCollectorAttributes) API

Required Parameter Group:

1	Collector name	Input	Char(10)
2	Default collection interval	Input	Binary(4)
3	Library	Input	Char(10)
4	Retention period	Input	Binary(4)
5	Cycle time	Input	Binary(4)
6	Cycle interval	Input	Binary(4)
7	Companion user job flag	Input	Binary(4)
8	Default collector definition	Input	Char(10)
9	Error code	I/O	Char(*)

Default Public Authority: *USE
Service program: QYPSCOLL
Threadsafe: No

The Change System Collector Attributes (QYPSCSCA, QypsChgSysCollectorAttributes) API changes system or global collection attributes. Attributes consist of the default collection interval in seconds, the library used to store the collection data, the retention period for the data, the time the initial cycle is to occur, the interval between cycles, whether a companion job is to be started, and the default collector definition. If appropriate, system collector attributes changed while a collector is running will take effect immediately.

Authorities and Locks

API Public Authority
*EXCLUDE

Job Authority
*JOBCTL

Library Authority
*EXECUTE

Required Parameter Group

Collector name
INPUT; CHAR(10)

The name of the collector whose default values are to be altered. The special value is:

**PFR* Performance collector

Default collection interval
INPUT; BINARY(4)

The default interval to use when collecting data for a category in seconds. This may be specified as 15, 30, 60, 300, 900, 1800, or 3600 seconds. Changes take effect immediately. The following special values are allowed:

0 Do not collect on interval
-2 No change

Library
INPUT; CHAR(10)

The name of the library used to store the collection data. Changes take effect when the collector starts or cycles. The following special values are allowed:

**CURLIB* Current library of the job calling the API
**SAME* No change

Retention period
INPUT; BINARY(4)

The retention period is used to determine how long collection data is to exist. Collection data older than the retention period is deleted. The retention period is specified in hours. Changes take effect immediately. The value specified must be between 1 and 720 hours, or one of the following special values:

0 Permanent
-2 No change

Cycle time
INPUT; BINARY(4)

The time at which the first cycle is to occur. The cycle time is specified in minutes past midnight. The maximum allowed value is 1439 minutes, which is one minute less than 24 hours. Changes take effect immediately. The following special value is allowed:

-2 No change

Cycle interval
INPUT; BINARY(4)

The lapse time between cycles. The cycle time is specified in hours, and can range from a minimum value of one hour to a maximum value of 24 hours. Changes take effect immediately. The following special value is allowed:

-2 No change

Companion user job flag
INPUT; BINARY(4)

Whether to start a job to run in concert with the collector. Changes take effect when the collector is started. The possible special values are:

-2 No change.
0 No companion user job is started.
1 A companion user job is started. (For the *PFR collector, this is the database transfer job CRTPFRDTA.)

Default collector definition
INPUT; CHAR(10)

The name of the collector definition to run. Changes take effect when the collector is started or cycled. The possible special values are:

**CURRENT*
**CUSTOM*
**MINIMUM*
**ENHCPCPLN*
**SAME*
**STANDARD*
**STANDARDP*

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
CPF222E E	&1 special authority is required.
CPF3C1E E	Required parameter &1 omitted.
CPF3C3C E	Value for parameter &1 is not valid.
CPF3C36 E	Number of parameters, &1, entered for this API was not valid.
CPF3CF2 E	Errors occurred during running of &1 API.
CPF9810 E	Library &1 not found.
CPF9820 E	Not authorized to use library &1.

API introduced: V4R4

"Change System Collector Attributes (QYPSCSCA, QypsChgSysCollectorAttributes) API" on page 6 | "Performance Management APIs," on page 1 | APIs by category

---

Cycle Collector (QYPSCYCC, QypsCycleCollector) API

Required Parameter Group:

1	Collector name	Input	Char(10)
2	Error code	I/O	Char(*)

Default Public Authority: *USE
Service program: QYPSCOLL
Threadsafe: No

The Cycle Collector (QYPSCYCC, QypsCycleCollector) API closes current collection objects and begins writing collection data to new collection objects.

Authorities and Locks

API Public Authority
*EXCLUDE

Job Authority
*JOBCTL

Required Parameter Group

Collector name
INPUT; CHAR(10)

The name of the collector to be cycled. The special value is:

**PFR* Performance collector

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
CPF222E E	&1 special authority is required.
CPF3C1E E	Required parameter &1 omitted.
CPF3C3C E	Value for parameter &1 is not valid.
CPF3C36 E	Number of parameters, &1, entered for this API was not valid.
CPF3CF2 E	Errors occurred during running of &1 API.
CPFB94A E	Collector communications error. Reason code &1.

API introduced: V4R4

"Change System Collector Attributes (QYPSCSCA, QypsChgSysCollectorAttributes) API" on page 6 | "Performance Management APIs," on page 1 | APIs by category

---

Deregister Collector Data Category (QypsDeregCollectorDataCategory) API

Required Parameter Group:

1	Collector name	I	Char(10)
2	Category name	I	Char(10)
3	Error Code	I/O	Char(*)

Service Program Name: QYPSCOLL
Default Public Authority: *EXCLUDE
Threadsafe: Yes

The Deregister Collector Data Category (QypsDeregCollectorDataCategory) API removes a user-defined data category from the Collection Services function of Management Central.

Authorities and Locks

API Public Authority
*EXCLUDE

Required Parameter Group

Collector name
INPUT; CHAR(10)

The name of the collector where the category will be removed. The only currently supported special value is:

**PFR* Performance Collector

Category name
INPUT; CHAR(10)

The unique name of the user-defined data category. Category name must be a valid *NAME (basic name) and all uppercase. See ELEM (Element) Statement in CL Reference for more information about *NAME. Names of user-defined data categories registered by IBM products start with ″Q″. Non-IBM applications are discouraged from prefixing names of user-defined categories with ″Q″.

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
CPF24B4 E	Severe error while addressing parameter list.
CPF222E E	&1 special authority is required.
CPF3C1E E	Required parameter &1 omitted.
CPF3C3C E	Value for parameter &1 not valid.
CPF3CF1 E	Error code parameter not valid.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPFB94E E	Category name &1 does not exist.

Introduced: V5R2

Top | "Performance Management APIs," on page 1 | APIs by category

End Collector (QYPSENDC, QypsEndCollector) API

Required Parameter Group:

1	Collector name	Input	Char(10)
2	Error code	I/O	Char(*)

Default Public Authority: *EXCLUDE
Service program: QYPSCOLL
Threadsafe: No

The End Collector (QYPSENDC,QypsEndCollector) API ends the data collection associated with the current collection definition of the specified collector. This may also end the collector job if there are no other data requests to process.

The *PFR collector can also process data requests from the Management Central monitors and applications that are collecting performance data with the performance collector APIs (QPMWKCOL and QPMLPFRD).

Note: The QPMASERV and QPMACLCT jobs will be active if the performance collector APIs are in use.

Authorities and Locks

API Public Authority
*EXCLUDE

Job Authority
*JOBCTL

Required Parameter Group

Collector name
INPUT; CHAR(10)

The name of the collector to be ended. The special value is:

**PFR* Performance collector job QYPSPFRCOL

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
CPF222E E	&1 special authority is required.
CPF3C1E E	Required parameter &1 omitted.
CPF3C3C E	Value for parameter &1 is not valid.
CPF3C36 E	Number of parameters, &1, entered for this API was not valid.
CPF3CF2 E	Errors occurred during running of &1 API.
CPFB94A E	Collector communications error. Reason code &1.

API introduced: V4R4

"Change System Collector Attributes (QYPSCSCA, QypsChgSysCollectorAttributes) API" on page 6 | "Performance Management APIs," on page 1 | APIs by category

---

Register Collector Data Category (QypsRegCollectorDataCategory) API

Required Parameter Group:

1	Collector name	I	Char(10)
2	Category name	I	Char(10)
3	Collector definition	I	Char(10)
4	CCSID	I	Binary(4)
5	Data collection program attributes	I	Char(*)
6	Category attributes	I	Char(*)
7	Error Code	I/O	Char(*)

Service Program Name: QYPSCOLL
Default Public Authority: *EXCLUDE
Threadsafe: Yes

The Register Collector Data Category (QypsRegCollectorDataCategory) API adds a user-defined data category to one or more collector definitions of the Collection Services function of Management Central.

Authorities and Locks

API Public Authority
*EXCLUDE

API caller must have at least *USE authority to the user profile specified in the Data collection program attributes parameter.

The user profile specified in the Data collection program attributes parameter must have at least *USE authority to the specified job description.

Required Parameter Group

Collector name
INPUT; CHAR(10)

The name of the collector where the user-defined data category will be added. The only currently supported value is:

**PFR* Performance Collector

Category name
INPUT; CHAR(10)

The unique name of the user-defined data category. The category name must be a valid *NAME (basic name) and all uppercase. See ELEM (Element) Statement in CL Reference for more information about *NAME. Names of user-defined data categories registered by IBM products start with ″Q″. Non-IBM applications are discouraged from prefixing names of user-defined categories with ″Q″.

Collector definition
INPUT; CHAR(10)

The collector definition that the user-defined data category will be added to. Only one collector definition may be specified. Specifying *STANDARD registers the category to the *STANDARD, *STANDARDP and *CUSTOM definitions. Specifying *STANDARDP registers the category to the *STANDARDP and *CUSTOM definitions. Specifying *CUSTOM registers the category to the *CUSTOM definition only. The possible values are:

- *CUSTOM
- *STANDARD
- *STANDARDP

CCSID
INPUT; BINARY(4)

The coded character set identifier (CCSID) for the user-defined data category. Refer to specific field descriptions to determine where the CCSID is applicable. The CCSID will be validated by the API. The default value is 0.

0 Use the current job default CCSID.
CCSID A valid CCSID number. The valid range for this parameter is 1 through 65533.

Data collection program attributes
INPUT; CHAR(*)

The attributes of the data collection program associated with the category. For more information on the format of the attributes, see “Format of Data Collection Program Attributes.”

Category attributes
INPUT; CHAR(*)

Additional attributes associated with the category. For more information on the format of the category attributes, see “Format of Category attributes” on page 15.

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 Data Collection Program Attributes

For detailed descriptions of the fields in this table, see “Data Collection Program Attributes Field Descriptions” on page 13.

Offset		Type	Field
Dec	Hex		
0	0	Binary(4)	Size of fixed portion of attributes
4	4	Char(10)	Program type
14	E	Char(8)	Parameter format
22	16	Char(10)	User profile
32	20	Char(20)	Qualified job description name
52	34	Char(20)	Qualified (service) program name
72	48	Binary(4)	Size of work area
76	4C	Binary(4)	Offset to service program entry point name
80	50	Binary(4)	Length of service program entry point name
84	54	Binary(4)	Offset to Java class name
88	58	Binary(4)	Length of Java class name
92	5C	Binary(4)	Offset to Java class path
96	60	Binary(4)	Length of Java class path
100	64	Binary(4)	Offset to category parameter string
104	68	Binary(4)	Length of category parameter string
108	6C	Binary(4)	Offset to Java options array
112	70	Binary(4)	Number of entries in Java options array
116	74	Binary(4)	Reserved

Data Collection Program Attributes Field Descriptions

Overview of Offset/Length usage of variable length character data The variable length character data will follow the fixed portion of the Data collection program attributes structure and reside with the same address space. The offset to the variable length character data specified in the fixed portion of the structure is the offset in bytes from the beginning of the attribute structure to the first byte of the character data.

Length of category parameter string The length of the string passed in for the category parameter string. If 0 is specified then the null parameter string will be passed to the data collection program.

Length of Java class name The length of the string passed in for the Java class name. This parameter must be set to 0 if Program type is not *JVAPGM.

Length of Java class path The length of the string passed in for the Java class path. This parameter must be set to 0 if Program type is not *JVAPGM.

Length of service program entry point name The length of the string passed in for the service program entry point name. This parameter must be set to 0 if Program type is not *SRVPGM.

Number of entries in Java Options Array The number of entries in the Java Options Array (see Format of Java Options Array below). If set to 0, no options will be passed to the Java Virtual Machine (JVM). This parameter must be set to 0 if Program type is not *JVAPGM.

Offset to category parameter string The offset of the string passed in for the category parameter string. The category parameter string is a character string which is passed to the data collection program to

customize its processing. The category parameter string must contain character data. The CCSID is applicable to the category parameter string. If 0 is specified then a null parameter string will be passed to the data collection program.

Offset to Java class name The offset in bytes to the string passed in for the Java class name. This parameter must be set to 0 if Program type is not *JVAPGM. The Java class name is the name of a Java class which implements the data collection program interface for this category. Refer to the Java section of the iSeries Information Center for more information on Java class name format. The CCSID is applicable to the Java class name.

Offset to Java class path The offset in bytes to the string passed in for the Java class path. This parameter must be set to 0 if Program type is not *JVAPGM. Refer to the Java section of the iSeries Information Center for more information on the Java class path format. The CCSID is applicable to the Java class path.

Offset to Java options array The offset of the Java Options Array (see Format of Java Options Array below). The Java Options Array is an array of options passed to the JVM. If the offset is set to 0, no options will be passed to the JVM. This parameter must be set to 0 if Program type is not *JVAPGM. All options are character strings and the CCSID is applicable to them.

Offset to service program entry point name The offset in bytes to the string passed in for the service program entry point name. This parameter must be set to 0 if Program type is not *SRVPGM. The service program entry point name is the name of an entry point in a service program which implements the data collection program for this category.

Parameter format This field defines the format of the parameters passed to the data collection program when it is called by Collection Services to collect data for the category. The only format currently supported is PMDC0100.

Qualified job description name The job description which will be used by the Collection Services secondary job to run the data collection program. The first 10 characters contain the job description name and the next 10 characters contain the library name. The following special values can be used for the job description name:

**JOB* Use the job description associated with the current job. The specified library parameter is ignored and must be filled with blank spaces or hex zeros.

**USER* Use the job description associated with the current user. The specified library parameter is ignored and must be filled with blank spaces or hex zeros.

The following special values can be used for the library name:

**CURLIB* The current library of the job executing this API.

**LIBL* Search the library list to find the specified job description.

Qualified (service) program name The qualified name of a program object if Program type is *PGM or the qualified name of a service program object if Program type is *SRVPGM. The first 10 characters contain the program or service program name and the next 10 characters contain the library name. This parameter must be set to 0 if Program type is *JVAPGM.

Size of fixed portion of attributes The size in bytes of the fixed portion of the Data collection program attribute structure.

Size of work area The size in bytes of a work area Collection Services will provide to the data collection program to save state information between the calls. This parameter must be set to 0 if Program type is *JVAPGM.

User profile User profile which will be used by Collection Services to run the data collection program. The API caller must have at least *USE authority to this user profile.

Format of Java Options Array

For detailed descriptions of the fields in this table, see "Java Options Array Field Descriptions."

Offset		Type	Field
Dec	Hex		
0	0	Binary(4)	Offset to Java option 1
4	4	Binary(4)	Length of Java option 1
8	8	Binary(4)	Offset to Java option 2
12	C	Binary(4)	Length of Java option 2
		...	
		Char(*)	Java option 1
		Char(*)	Java option 2

The Java options array contains an array of option settings which will be passed to the Java Virtual Machine (JVM) at Java initialization time. The number of elements in this array is determined by the Number of entries in Java options array field in the Data collection program attributes structure. This array is optional and is ignored when Program type is not *JVAPGM.

Java options are not validated and are passed to the JVM exactly as specified for the Registration API.

Java Options Array Field Descriptions

Length of Java option N

The length in bytes of the string passed in for the Nth Java option.

Offset to Java option N

The offset in bytes from the beginning of the Java Options Array to the string passed in for the Nth Java option. All Java options are character strings and the CCSID applies to them.

Format of Category attributes

For detailed descriptions of the fields in this table, see "Category attributes Field Descriptions" on page 16.

Offset		Type	Field
Dec	Hex		
0	0	Binary(4)	Size of attribute structure
4	4	Binary(4)	Minimum collection interval
8	8	Binary(4)	Maximum collection interval
12	C	Binary(4)	Default collection interval
16	10	Char(27)	Qualified message file and message identifier
43	2B	Char(50)	Text description
93	5D	Char(3)	Reserved

Category attributes Field Descriptions

Default collection interval

The default interval to use when collecting data for a category in seconds. This may be specified as one of 15, 30, 60, 300, 900, 1800, or 3600 seconds. The following special value is allowed:

0 Use the collector definition of the collector that the category is registered to.

Minimum collection interval

The minimum interval this user-defined data category should be collected at. In other words, this represents the smallest interval of data collection. This may be specified as one of 15, 30, 60, 300, 900, 1800, or 3600 seconds. Specifying 0 represents no restriction on the minimum collection interval.

Maximum collection interval

The maximum interval this user-defined data category should be collected at. In other words, this represents the largest interval of data collection. This may be specified as one of 15, 30, 60, 300, 900, 1800, or 3600 seconds. Specifying 0 represents no restriction on the maximum collection interval.

Qualified message file and message identifier

The qualified message file and message identifier of the text description of the category. The first 10 characters contain the message file name, the next 10 characters contain the library name of the message file, and the final 7 characters contain the message identifier. If the text description is specified as a character string (in Text description field), this field should be set to all blanks or hex zeros. The possible values for the library are:

**LIBL* Search the library list for the first occurrence of the message file.
Library name The name of the library the message file resides in.

Size of attribute structure

The size in bytes of the category attribute structure.

Text description

The text description associated with the category. The supplied CCSID will be applied to the text description. This parameter is ignored and must be filled with blank spaces or hex zeros if a qualified message file and message identifier has been specified.

Error Messages

Message ID	Error Message Text
CPF24B4 E	Severe error while addressing parameter list.
CPF222E E	&1 special authority is required.
CPF3C1E E	Required parameter &1 omitted.
CPF3C3C E	Value for parameter &1 not valid.
CPF3CF1 E	Error code parameter not valid.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9802 E	Not authorized to object &2 in &3.
CPF9810 E	Library &1 not found.
CPF9820 E	Not authorized to use library &1.
CPFB537 E	Error found in parameter &1 at offset &2.

Message ID	Error Message Text
CPFB538 E	Error found in parameter &1 at offset &2.
CPFB94C E	Collection interval value must be one of 15, 30, 60, 300, 900, 1800, or 3600 seconds.
CPFB94D E	Category name &1 aleady exists.

Introduced: V5R2

Top | "Performance Management APIs," on page 1 | APIs by category

Remove Collector Notification (QypsRmvCollectorNotification) API

Required Parameter Group:

1	Collector name	I	Char(10)
2	Qualified data queue name	I	Char(20)
3	Notification type	I	Binary(4)
4	Category list	I	Array of Char(10)
5	Category count	I	Binary(4)
6	Error Code	I/O	Char(*)

Service Program Name: QYPSCOLL
Default Public Authority: *EXCLUDE
Threadsafe: Yes

The Remove Collector Notification (QypsRmvCollectorNotification) API removes a notification registration from a collector for a specified data queue and collection event. A collection event occurs when:

- The collector cycle interval is reached.
- The collector is ended or stopped.
- The default data collection interval is reached.
- The category data collection interval is reached.

Authorities and Locks

API Public Authority
*EXCLUDE

Data Queue Authority
*CHANGE

Library Authority
*EXECUTE

Required Parameter Group

Collector name
INPUT; CHAR(10)

The name of the collector that is removing notification. One of these special values must be used:

**PFR* Performance Collector

Qualified data queue name
INPUT; CHAR(20)

The data queue from which the notification is to be removed. The first ten characters contain the data queue name, and the second ten characters contain the data queue library name. The data queue must already exist. You can use these special values for the library name:

**CURLIB* The job's current library.
**LIBL* The library list.

Notification type
INPUT; BINARY(4)

The type of event notification to remove:

0 Collector - remove the cycle, end, and interval notifications.
1 Cycle - remove the collection cycle event notification.
2 End - remove notification of the end event.
3 Interval - remove notification of the default collection interval event.
4 Category - remove notification for the category event.

Category list
INPUT; ARRAY OF CHAR(10)

List of category names, for which notification is to be removed. This field is only applicable when Notification type is set to category notification (4). Category name can be a system-defined category name or a user-defined category name.

System-defined category name
A 10 character name of a system-defined category. For the *PFR collector system-defined categories are:

- *APPN
- *CMNBASE
- *CMNSAP
- *CMNSTN
- *DISK
- *HDWCFG
- *IOPBASE
- *IPCS
- *JOBMI
- *JOBOS
- *LCLRSP
- *POOL
- *POOLTUNE
- *SNA
- *SNADS
- *SUBSYSTEM
- *SYSBUS
- *SYSCPU
- *SYSLVL
- *TCPBASE
- *TCPIFC
- *USRTNS

Registered user-defined category name
A 10 character name of a user-defined category registered by the Register Collector Data Category (QypsRegCollectorDataCategory) API.

Category count
INPUT; BINARY(4)

The number of categories entered in input field Category list. This field is only applicable when Notification type is set to category notification (4). Category count must have a value of '0' when Notification type is not a category notification (4).

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
CPF3C1E E	Required parameter &1 omitted.
CPF3C3C E	Value for parameter &1 not valid.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9802 E	Not authorized to object &2 in &3.
CPF9820 E	Not authorized to use library &1.
CPFB94A E	Collector communications error. Reason code &1.

API introduced: V5R2

Top | "Performance Management APIs," on page 1 | APIs by category

---

Retrieve System Collector Attributes (QYPSRSCA, QypsRtvSysCollectorAttributes) API

Required Parameter Group:

1	Collector name	Input	Char(10)
2	Default collection interval	Output	Binary(4)
3	Library	Output	Char(10)
4	Retention period	Output	Binary(4)
5	Cycle time	Output	Binary(4)
6	Cycle interval	Output	Binary(4)
7	Companion user job	Output	Binary(4)
8	Default collector definition	Output	Char(10)
9	Current collector definition	Output	Char(10)
10	Error code	I/O	Char(*)

Default Public Authority: *EXCLUDE
Service program: QYPSCOLL
Threadsafe: No

The Retrieve System Collector Attributes (QYPSRSCA, QypsRtvSysCollectorAttributes) API retrieves system or global collection attributes. Attributes consist of the default collector state, the default collection interval in seconds, the library used to store the collection data, the retention period for the data, the time

the initial cycle is to occur, the interval between cycles, the companion user job flag, the default collection definition, and the currently running collection definition.

Authorities and Locks

API Public Authority
*EXCLUDE

Required Parameter Group

Collector name
INPUT; CHAR(10)

The name of the collector whose default values are to be retrieved. The special value is:

**PFR* Performance collector

Default collection interval
OUTPUT; BINARY(4)

The default interval used when collecting data for a category in seconds. The interval is 15, 30, 60, 300, 900, 1800, or 3600 seconds. The following special value may be returned:

0 Do not collect on interval

Library
OUTPUT; CHAR(10)

The name of the library used to store the collection data.

Retention period
OUTPUT; BINARY(4)

The retention period indicates how long collection data is to exist. Collection data older than the retention period is deleted. The retention period is specified in hours. The maximum value that will be returned is 720 hours, or 30 days. The following special value may be returned:

0 Permanent

Cycle time
OUTPUT; BINARY(4)

The time at which the first cycle is to occur. The cycle time is specified in minutes past midnight. The maximum allowed value is 1439 minutes, which is one minute less than 24 hours.

Cycle interval
OUTPUT; BINARY(4)

The lapse time between cycles. The cycle time is specified in hours, and can range from a minimum value of one hour to a maximum value of 24 hours.

Companion user job flag
OUTPUT; BINARY(4)

Whether a job is started to run in concert with the collector. One of the following values will be returned:

0 No companion user job is started.
1 A companion user job is started. (For the *PFR collector, this is the database transfer job CVTPFRDTA.)

Default collector definition
OUTPUT; CHAR(10)

The name of the collector definition to run. The possible special values are:

**CURRENT*
**CUSTOM*
**ENHCPCPLN*
**MINIMUM*
**STANDARD*
**STANDARDP*

Current collector definition
OUTPUT; CHAR(10)

The name of the currently running collector definition. The possible special values are:

**CUSTOM*
**ENHCPCPLN*
**MINIMUM*
**NONE*
**STANDARD*
**STANDARDP*

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
CPF3C36 E	Number of parameters, &1, entered for this API was not valid.
CPF3C3C E	Value for parameter &1 is not valid.
CPF3C1E E	Required parameter &1 omitted.
CPF3CF2 E	Errors occurred during running of &1 API.

API introduced: V4R4

"Change System Collector Attributes (QYPSCSCA, QypsChgSysCollectorAttributes) API" on page 6 | "Performance Management APIs," on page 1 | APIs by category

---

Start Collector (QYPSSTRC, QypsStartCollector) API

Required Parameter Group:

1	Collector name	Input	Char(10)
2	Default collector definition	Input	Char(10)
3	Error code	I/O	Char(*)

Default Public Authority: *EXCLUDE
Service program: QYPSCOLL
Threadsafe: No

The Start Collector (QYPSSTRC, QypsStartCollector) API starts a collector. When the collector job is not running, the job is submitted to the QSYSNOMAX job queue and a start request passes to it. If no default collection definition is provided, the default provided by the system value is used. When the collector job is running and a new default collection definition is provided, the collector changes to use that definition. If the collector job is running and no new default collector definition is provided, no action is taken.

Authorities and Locks

API Public Authority
*EXCLUDE

Job Authority
*JOBCTL

Required Parameter Group

Collector name
INPUT; CHAR(10)

The name of the collector to start. The special value is:

**PFR* Performance collector job QYPSPFRCOL

Default collector definition
OUTPUT; CHAR(10)

The name of the collector definition to run. The possible special values are:

**CURRENT*
**CUSTOM*
**ENHCPCPLN*
**MINIMUM*
**SAME*
**STANDARD*
**STANDARDP*

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
CPF222E E	&1 special authority is required.
CPF3C1E E	Required parameter &1 omitted.
CPF3C3C E	Value for parameter &1 is not valid.
CPF3C36 E	Number of parameters, &1, entered for this API was not valid.
CPF3CF2 E	Errors occurred during running of &1 API.
CPFB94A E	Collector communications error. Reason code &1.

API introduced: V4R4

"Change System Collector Attributes (QYPSCSCA, QypsChgSysCollectorAttributes) API" on page 6 | "Performance Management APIs," on page 1 | APIs by category

---

Management Collection Object APIs

The management collection object APIs support working with management collection objects (attribute *PFR). These APIs:

- Retrieve the active management collection object name
- Retrieve the attributes of a management collection object
- Open and close a management collection object

- Open and close a repository of a management collection object
- Read data from a repository of a management collection object

The management collection object APIs are:

- "Close Management Collection Object (QpmCloseMgtcol) API" (QpmCloseMgtcol) closes a management collection object.
- "Close Management Collection Object Repository (QpmCloseMgtcolRepo) API" on page 24 (QpmCloseMgtcolRepo) closes a repository of a management collection object.
- "Open Management Collection Object (QpmOpenMgtcol) API" on page 25 (QpmOpenMgtcol) opens a specified management collection object for processing.
- "Open Management Collection Object Repository (QpmOpenMgtcolRepo) API" on page 26 (QpmOpenMgtcolRepo) opens a specified repository of a management collection object for processing.
- "Read Management Collection Object Data (QpmReadMgtcolData) API" on page 28 (QpmReadMgtcolData) positions to a specific record in a repository of a management collection object, returns information about the record, and optionally reads specified bytes of data from the record.
- "Retrieve Active Management Collection Object Name (QpmRtvActiveMgtcolName) API" on page 32 (QpmRtvActiveMgtcolName) returns the object name and library name of an active management collection object.
- "Retrieve Management Collection Object Attributes (QpmRtvMgtcolAttrs) API" on page 33 (QpmRtvMgtcolAttrs) returns information about attributes of a management collection object and repositories of a management collection object.

Top | "Performance Management APIs," on page 1 | APIs by category

---

Close Management Collection Object (QpmCloseMgtcol) API

Required Parameter Group:

1	Management collection object handle	Input	Binary(4)
2	Error code	I/O	Char(*)

Service Program Name: QPMAAPI
Default Public Authority: *EXCLUDE
Threadsafe: Yes

The Close Management Collection Object (QpmCloseMgtcol) API closes a management collection object that was previously opened by the "Open Management Collection Object (QpmOpenMgtcol) API" on page 25. All repositories that were opened for this management collection object by the "Open Management Collection Object Repository (QpmOpenMgtcolRepo) API" on page 26 are implicitly closed. After the management collection object is closed, its handle is no longer valid.

Authorities and Locks

API Public Authority
*EXCLUDE

When the management collection object is closed, the *SHRRD lock placed on the object by the Open Management Collection Object (QpmOpenMgtcol) API is released.

Required Parameter Group

Management collection object handle
INPUT; BINARY(4)

A handle to an open management collection object. This handle was created by the Open Management Collection Object (QpmOpenMgtcol) API.

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
CPF0AA4 E	Lock request was not satisfied in a specified time.
CPF24B4 E	Severe error while addressing parameter list.
CPF3CF1 E	Error code parameter not valid.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF3C3C E	Value for parameter &1 not valid.

API introduced: V5R2

Top | "Performance Management APIs," on page 1 | APIs by category

---

Close Management Collection Object Repository (QpmCloseMgtcolRepo) API

Required Parameter Group:

1	Management collection object repository handle	Input	Binary(4)
2	Error code	I/O	Char(*)

Service Program Name: QPMAAPI
Default Public Authority: *EXCLUDE
Threadsafe: Yes

The Close Management Collection Object Repository (QpmCloseMgtcolRepo) API closes a repository of a management collection object that was previously opened by the "Open Management Collection Object Repository (QpmOpenMgtcolRepo) API" on page 26. After the repository is closed, its handle is no longer valid.

Authorities and Locks

API Public Authority
*EXCLUDE

Required Parameter Group

Management collection object repository handle
INPUT; BINARY(4)

A handle to an open repository of a management collection object. This handle was created by the Open Management Collection Object Repository (QpmOpenMgtcolRepo) API.

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
CPF0AA4 E	Lock request was not satisfied in a specified time.
CPF24B4 E	Severe error while addressing parameter list.
CPF3C3C E	Value for parameter &1 not valid.
CPF3CF1 E	Error code parameter not valid.
CPF3CF2 E	Error(s) occurred during running of &1 API.

API introduced: V5R2

Top | "Performance Management APIs," on page 1 | APIs by category

---

Open Management Collection Object (QpmOpenMgtcol) API

Required Parameter Group:

1	Qualified object name	Input	Char(20)
2	Management collection object handle	Output	Binary(4)
3	Error code	I/O	Char(*)

Service Program Name: QPMAAPI
Default Public Authority: *EXCLUDE
Threadsafe: Yes

The Open Management Collection Object (QpmOpenMgtcol) API opens a specified management collection object for processing and returns a handle to the open management collection object. This handle uniquely identifies the open management collection object and is used by the following APIs:

- "Close Management Collection Object (QpmCloseMgtcol) API" on page 23
- "Open Management Collection Object Repository (QpmOpenMgtcolRepo) API" on page 26

The management collection object handle is valid until the management collection object is closed by the Close Management Collection Object (QpmCloseMgtcol) API. The handle is scoped to a job so that a management collection object opened in one thread can be used by another thread provided the handle is known.

Authorities and Locks

API Public Authority
*EXCLUDE

Authority to library containing collection object
*EXECUTE

If the open operation was successful, a *SHRRD lock is placed on the management collection object.

Required Parameter Group

Qualified object name
INPUT; CHAR(20)

Name of a management collection object and the library in which it is located. The first 10 characters contain the object name and the second 10 characters contain the library name.

The system supports management collection objects with different attributes; they contain different information. The Management Collection Object APIs support only collection objects which are created by the Collection Services collector. These collection objects have the attribute *PFR.

Management collection object handle
OUTPUT; BINARY(4)

A handle to the open management collection object. This handle is used by the Close Management Collection Object (QpmCloseMgtcol) API and the Open Management Collection Object Repository (QpmOpenMgtcolRepo) API to uniquely identify the open management collection object.

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
CPF0A2B E	Not able to process management collection object &1 in library &2.
CPF0AA4 E	Lock request was not satisfied in a specified time.
CPF2105 E	Object &1 in &2 type *&3 not found.
CPF2110 E	Library &1 not found.
CPF2114 E	Cannot allocate object &1 in &2 type *&3.
CPF2207 E	Not authorized to use object &1 in library &3 type *&2.
CPF24B4 E	Severe error while addressing parameter list.
CPF3CF1 E	Error code parameter not valid.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9810 E	Library &1 not found.

API introduced: V5R2

Top | "Performance Management APIs," on page 1 | APIs by category

---

Open Management Collection Object Repository (QpmOpenMgtcolRepo) API

Required Parameter Group:

1	Management collection object handle	Input	Binary(4)
2	Management collection object repository name	Input	Char(10)
3	Format name	Input	Char(8)
4	Management collection object repository handle	Output	Binary(4)
5	Error code	I/O	Char(*)

Service Program Name: QPMAAPI
Default Public Authority: *EXCLUDE
Threadsafe: Yes

The Open Management Collection Object Repository (QpmOpenMgtcolRepo) API opens a specified repository of a management collection object for processing. The management collection object is identified by a handle which was created by the "Open Management Collection Object (QpmOpenMgtcol) API" on page 25. If the open operation is successful, a handle to the open repository is returned. This handle uniquely identifies the open repository and is used by these APIs:

- "Close Management Collection Object Repository (QpmCloseMgtcolRepo) API" on page 24
- "Read Management Collection Object Data (QpmReadMgtcolData) API" on page 28

The management collection object repository handle is valid until the repository is closed by the Close Management Collection Object Repository (QpmCloseMgtcolRepo) API. The repository handle is scoped to a job so that a repository opened in one thread can be used by another thread provided the handle is known.

The API caller must specify a format name which identifies the kind of processing to be performed on the repository data. This format name also defines the format of the input and output parameters of the Read Management Collection Object Data (QpmReadMgtcolData) API when this API is used with this repository.

Authorities and Locks

API Public Authority
*EXCLUDE

Required Parameter Group

Management collection object handle
INPUT; BINARY(4)

A handle to an open management collection object. This handle was created by the Open Management Collection Object (QpmOpenMgtcol) API.

Management collection object repository name
INPUT; CHAR(10)

Name of a repository of a management collection object. Currently, the API supports repositories created by user-defined performance collection categories only.

Format name
INPUT; CHAR(8)

Name of the format that defines the kind of processing to be performed on the data in this repository. Currently, the Management Collection Object APIs support format MCOD0100 only.

When this format is specified, the Read Management Collection Object Data (QpmReadMgtcolData) API will return raw data from the repository of the management collection object. No additional processing will be performed and the data will be treated as an unstructured sequence of bytes.

The format name also defines the format of the input and output parameters of the Read Management Collection Object Data (QpmReadMgtcolData) API when this API is called for this repository. See description of "Read Management Collection Object Data (QpmReadMgtcolData) API" on page 28 for more details.

Management collection object repository handle
OUTPUT; BINARY(4)

A handle to the open repository of the management collection object. This handle is used by other APIs to uniquely identify the open repository of the management collection object.

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
CPF0AA2 E	Repository &1 is not found in a collection object.
CPF0AA3 E	Attempt to access unsupported repository.
CPF0AA4 E	Lock request was not satisfied in a specified time.
CPF24B4 E	Severe error while addressing parameter list.
CPF3C21 E	Format name &1 is not valid.
CPF3C3C E	Value for parameter &1 not valid.
CPF3CF1 E	Error code parameter not valid.
CPF3CF2 E	Error(s) occurred during running of &1 API.

API introduced: V5R2

Top | "Performance Management APIs," on page 1 | APIs by category

---

Read Management Collection Object Data (QpmReadMgtcolData) API

Required Parameter Group:

1	Management collection object repository handle	Input	Binary(4)
2	Read options	Input	Char(*)
3	Record information	Output	Char(*)
4	Record data	Output	Char(*)
5	Error code	I/O	Char(*)

Service Program Name: QPMAAPI
Default Public Authority: *EXCLUDE
Threadsafe: Yes

The Read Management Collection Object Data (QpmReadMgtcolData) API performs the following actions:

- Positions to a specific record in a repository of a management collection object.
- Returns information about the record.
- Optionally reads specified bytes of data from the record.

The repository is identified by a handle which was previously created by the "Open Management Collection Object Repository (QpmOpenMgtcolRepo) API" on page 26.

Record processing options are specified in the read options parameter.

Information about the repository record is returned in the record information parameter.

Data from a record is returned in the record data parameter.

The formats of the read options, record information and record data parameters are determined by the format name that was passed to the Open Management Collection Object Repository (QpmOpenMgtcolRepo) API at the time the repository was opened for processing.

Authorities and Locks

API Public Authority
*EXCLUDE

Required Parameter Group

Management collection object repository handle
INPUT; BINARY(4)

A handle to an open repository of a management collection object. This handle was created by the Open Management Collection Object Repository (QpmOpenMgtcolRepo) API.

Read options
INPUT; CHAR(*)

Contains control information that determines how the API will process the record. See "Format of Read Options Parameter."

Record information
OUTPUT; CHAR(*)

Information about the current repository record. See "Format of Record Information Parameter" on page 30. This parameter should be large enough to accommodate the entire record information structure. Otherwise, results are unpredictable.

Record data
OUTPUT; CHAR(*)

If requested in the read options parameter, data from the current repository record is returned in this parameter. The format of the data returned in this parameter is determined by the format name passed to the Open Management Collection Object Repository (QpmOpenMgtcolRepo) API at the time the repository was opened for processing.

The only format supported in this release is MCOD0100. For the MCOD0100 format, the API returns an unformatted sequence of bytes from the current repository record.

This parameter should be large enough to accommodate all data requested in the read options parameter. Otherwise, results are unpredictable

See "Format of Read Options Parameter" for more details.

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 Read Options Parameter

The format of the read options parameter is determined by the format name passed to the Open Management Collection Object Repository (QpmOpenMgtcolRepo) API at the time the repository was opened for processing.

The only format supported in this release is MCOD0100. The table below shows the structure of the read options parameter for the MCOD0100 format. For detailed descriptions of the fields in the table, see "Field Descriptions" on page 30 below.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bytes provided by API caller
4	4	BINARY(4)	Record positioning option
8	8	BINARY(8)	Offset in record data
16	10	BINARY(8)	Number of bytes to read
24	18	CHAR(8)	Record key

Format of Record Information Parameter

The format of the record information parameter is determined by the format name passed to the Open Management Collection Object Repository (QpmOpenMgtcolRepo) API at the time the repository was opened for processing.

The only format supported in this release is MCOD0100. The table below shows the structure of the record information parameter for MCOD0100 format. For detailed descriptions of the fields in the table, see "Field Descriptions" below.

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Record status
4	4	BINARY(4)	Record type
8	8	BINARY(8)	Number of bytes returned
16	10	CHAR(8)	Record key
24	18	CHAR(8)	Record timestamp
32	20	BINARY(8)	Total record data length

Field Descriptions

Bytes provided by API caller. The number of bytes of read options provided. For the MCOD0100 format, this length should be at least 32 bytes.

Number of bytes returned. The number of bytes of record data returned by the API in the record data parameter.

Number of bytes to read. The number of bytes of record data that should be returned in the record data parameter. If this field is set to zero, no record data will be returned. If the repository record contains less data than requested by this field, the API returns only the available data. The number of bytes returned field in the record information parameter will be set to the actual number of bytes returned.

Offset in record data. Byte offset into the record data identifying the first byte of the record data to be returned in the record data parameter. This field and the number of bytes to read field together define which part of the record data will be returned.

Record key. For the read options parameter, this field is used together with the record positioning option field to specify the record key used in the record search.

For the record information parameter, this field returns the key of the record actually found.

Format of this field is DDHHMMSS, where:

DD Number of days from the beginning of collection to this collection object. Day numbering starts from 0.
HHMMSS Time in hours, minutes and seconds when a particular collection sample was scheduled.

Record keys of repository records, with the possible exception of the first record in the collection period, are normalized at the collection interval boundary. For example, for a 15-minute collection interval, valid record keys will be 00124500 or 01223000, but not 00131014.

Record positioning option. The record that is the target of this call to the API. Supported positioning options are:

0 *Read next record.* For this option, the API returns the next repository record in relation to the one processed by the previous call to the API. If no records have been read from the repository, the very first record is returned. If the previous record was the last one in the repository or if the repository is empty, the API returns record-not-found record status.

1 *Read current record.* For this option, the API returns the same record that was processed by the previous call to the API. This option is used to read different parts of the same record. If no records have been read from repository, the API returns a record-not-found record status.

2 *Read first record.* For this option, the API returns the very first record in the repository. This option is used to start reading the repository from the beginning. If the repository is empty, the API returns record-not-found record status.

3 *Read record by key equal.* For this option, the API returns the record with the key specified in the record key field of the read options parameter. If no record is found, the API returns record-not-found record status.

4 *Read record by key less than or equal.* For this option, the API returns the record with the largest key that is less than or equal to the key specified in the record key field of the read options parameter. If no record is found, the API returns record-not-found record status.

5 *Read record by key greater than or equal.* For this option, the API returns the record with the smallest key that is greater than or equal to the key specified in the record key field of the read options parameter. If no record is found, the API returns record-not-found record status.

Record status. The result of record positioning. Valid values are:

0 Record was successfully found and processed.

1 Record-not-found status. Possible causes for this status are listed in the description of the record positioning option field.

Record timestamp. The exact time when data collection started for the current repository record. Time is represented in the system timestamp format. See Convert Date and Time Format (QWCCVTDT) API for details about time formats. Unlike the time represented by the record key field, this time is not normalized. Note that data collection can be a time-consuming process. The record timestamp field contains the time when data collection started for the current record, not necessarily the time when the collection was completed and the last piece of data was written into this record.

Record type. The type of the current repository record. The following record types can be returned:

0 Interval record

1 Collection control record

2 Stop record

3 Unexpected record type

Total record data length. Length in bytes of the record data in the current repository record.

Usage Notes

To understand how this API works, it is important to know how data is stored in the management collection object.

Collection Services stores performance data collected for a performance collection category in a repository of a management collection object. Data is stored as a sequence of repository records of different types. The following record types are defined:

- *Collection control record.* This type of record can be used by the performance collection category to store some kind of control information necessary for the correct interpretation of the collected data. This type of record is normally written as the first record of the collection session, but can also be written as the last record before the stop record.

- *Interval record.* This type of record contains actual performance data. One record of this type is produced for every collection interval.
- *Stop record.* This type of record is the last one in a series of records pertaining to one collection session. If data collection for the performance collection category was restarted without cycling the collector, the stop record will be followed by an (optional) collection control record, then interval records for the new session and so on.

The repository records contain control information such as record type, record key, record timestamp, and so on, and a variable amount of record data (between 0 and 4GB).

Error Messages

Message ID	Error Message Text
CPF24B4 E	Severe error while addressing parameter list.
CPF3CF1 E	Error code parameter not valid.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF3C3C E	Value for parameter &1 not valid.
CPF0AA4 E	Lock request was not satisfied in a specified time.

API introduced: V5R2

Top | "Performance Management APIs," on page 1 | APIs by category

---

Retrieve Active Management Collection Object Name (QpmRtvActiveMgtcolName) API

Required Parameter Group:

1	Qualified object name	Output	Char(20)
2	Error code	I/O	Char(*)

Service Program Name: QPMAAPI
Default Public Authority: *EXCLUDE
Threadsafe: Yes

The Retrieve Active Management Collection Object Name (QpmRtvActiveMgtcolName) API returns the object name and library name of an active management collection object. This is an object that is currently used by the Collection Services collector to collect performance data.

Authorities and Locks

API Public Authority
*EXCLUDE

Required Parameter Group

Qualified object name
OUTPUT; CHAR(20)

The name of an active management collection object and the library in which it is located. On successful return from the API, the first 10 characters contain the object name and the second 10 characters contain 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.

Error Messages

Message ID	Error Message Text
CPF0A1A E	No active collection.
CPF24B4 E	Severe error while addressing parameter list.
CPF3CF1 E	Error code parameter not valid.
CPF3CF2 E	Error(s) occurred during running of &1 API.

API introduced: V5R2

Top | "Performance Management APIs," on page 1 | APIs by category

---

Retrieve Management Collection Object Attributes (QpmRtvMgtcolAttrs) API

Required Parameter Group:

1	Receiver variable	Output	Char(*)
2	Length of receiver variable	Input	Binary(4)
3	Format name	Input	Char(8)
4	Qualified object name	Input	Char(20)
5	Error code	I/O	Char(*)

Service Program Name: QPMAAPI
Default Public Authority: *EXCLUDE
Threadsafe: Yes

The Retrieve Management Collection Object Attributes (QpmRtvMgtcolAttrs) API returns information about attributes of a management collection object and repositories of a management collection object.

Authorities and Locks

API Public Authority
*EXCLUDE

Authority to library containing collection object
*EXECUTE

While retrieving attributes, this API places a *SHRRD lock on the management collection object.

Required Parameter Group

Receiver variable
OUTPUT; CHAR(*)

The variable that receives the requested information. It can be smaller than the format requested as long as the next parameter, length of receiver variable, specifies the length correctly. When this variable is smaller than the format, the API returns only the data that the variable can hold.

Length of receiver variable
INPUT; BINARY(4)

The length of the receiver variable. The minimum length is 8 bytes. Do not specify a length that is longer than the receiver variable; the results are unpredictable.

Format name
INPUT; CHAR(8)

The content and format of the information returned in the receiver variable for a specified management collection object. The possible format names are:

"MCOA0100 Format"	Retrieve attributes of a management collection object only
"MCOA0200 Format" on page 35	Retrieve attributes of a management collection object and attributes of repositories of an object

Qualified object name
INPUT; CHAR(20)

Name of a management collection object for which you want to retrieve information and the library in which it is located. The first 10 characters contain the object name and the second 10 characters contain the library name.

The system supports management collection objects with different attributes, or objects that contain different information. The Management Collection Object APIs support collection objects that are created by the Collection Services collector only. These collection objects have attribute *PFR.

Error code
I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error code parameter.

MCOA0100 Format

The following information is returned for the MCOA0100 format. For detailed descriptions of the fields in the table, see "Field Descriptions" on page 36.

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	BINARY(8)	Object size
16	10	BINARY(4)	Object retention period
20	14	BINARY(4)	Default collection interval
24	18	BINARY(4)	Number of repositories
28	1C	CHAR(14)	Date and time when object was created
42	2A	CHAR(14)	Date and time of last update to the object
56	38	CHAR(10)	Partition serial number
66	42	CHAR(1)	Object is active
67	43	CHAR(1)	Object was repaired
68	44	CHAR(1)	Summarization status
69	45	CHAR(3)	Reserved

MCOA0200 Format

When the MCOA0200 format is requested, this API will return information about the management collection object (MCOA0100 format) plus information about all of the repositories found in this management collection object. This format returns zero or more "Repository entry," described later. The number of repository entries returned in this format is specified in the number of repository entries returned field. For detailed descriptions of the fields in the table, see "Field Descriptions" on page 36.

Offset		Type	Field
Dec	Hex		
0	0		Everything from MCOA0100 format
72	48	BINARY(4)	Number of repository entries returned
76	4C	BINARY(4)	Offset to repository information
80	50	CHAR(*)	Repository information
80	50	BINARY(4)	Offset to repository entry 1
84	54	BINARY(4)	Length of repository entry 1
88	58	BINARY(4)	Offset to repository entry 2
92	5C	BINARY(4)	Length of repository entry 2
		...	
		CHAR(*)	Repository entry 1
		CHAR(*)	Repository entry 2
		...	

Repository entry

Each repository entry contains attributes of one repository of a management collection object. It includes one or more "Collection period entry," described later.

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	Repository name
10	A	CHAR(10)	Category name
20	14	BINARY(4)	Number of collection period entries
24	18	BINARY(8)	Repository size
32	20	CHAR(*)	Collection period entry 1

Collection period entry

A new collection period entry is created each time a collection is started or a collection interval is changed for the performance category associated with a repository. Each repository has at least one collection period.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Length of collection period entry
4	4	CHAR(14)	Date and time of collection period start

Offset		Type	Field
Dec	Hex		
18	12	CHAR(14)	Date and time of collection period end
32	20	BINARY(4)	Collection interval
36	24	CHAR(4)	Reserved

Field Descriptions

Bytes available. The length of all the data available to return. All available data is returned if the receiver variable has sufficient length.

Bytes returned. The length of the data actually returned.

Category name. The name of the performance collection category that created this repository.

Collection interval. The collection interval in seconds for the performance category that created this repository. Each category can have its own collection interval. A performance category can have different collection intervals at different times during the collection (for different collection periods). If the collection interval is negative, this category was not configured to perform interval collections. Such categories perform one-time-only collections; in the beginning of the collection (collection interval is set to minus 1) or at the end of the collection (collection interval is set to minus 2).

Date and time of collection period end. The date and time when either the collection was ended in this repository or the collection interval was changed for the performance category associated with this repository. This is represented in a format YYYYMMDDHHMMSS, where:

YYYY	Year
MM	Month
DD	Day of the month
HH	Hour
MM	Minute
SS	Second

This field reports blanks for collection periods that are in progress.

Date and time of collection period start. The date and time when either the collection was started into this repository or the collection interval was changed for the performance category associated with this repository. For a description of the format of this field, see the date and time of collection period end field.

Date and time when object was created. The date and time the collection object was created. For a description of the format of this field, see the date and time of collection period end field.

Date and time of last update to the object. The date and time when the last update to collection object data occurred. For a collection object that is not active, this is the time when the collection ended to this collection object. For a description of the format of this field, see the date and time of collection period end field.

Default collection interval. The default collection interval in seconds for this collection object. Individual performance categories may have different collection intervals.

Length of collection period entry. The length of a collection period entry, in bytes.

Length of repository entry. The length of a repository entry, in bytes. A repository entry contains information about one particular repository and includes one or more collection period entries.

Number of collection period entries. The number of collection period entries reported for a repository. Each repository has at least one collection period entry. A new collection period starts when the collection is started for the performance category associated with this repository or a new collection interval is set for this category.

Number of repositories. The number of repositories found in this management collection object.

Number of repository entries returned. The number of repositories for which information is returned in a receiver variable. This can be different than the value in the number of repositories field if the receiver variable is not big enough to hold the entire result.

Object is active. Whether collection is currently in progress for this collection object. Possible values are:

0 Collection into this collection object has ended
1 Collection into this collection object is in progress

Object retention period. The number of hours the collection object should be kept on the system before it is deleted automatically. The retention period starts when the collection is ended for this collection object. When the collection object is set for permanent retention, the object retention period field is set to minus 1.

Object size. The size of the management collection object in Kbytes (K = 1024).

Object was repaired. Whether the collection object was repaired. When the collection object is not correctly closed, for example, during abrupt system termination, it is repaired when it is touched the first time after that. Such an object may have corrupted data inside. Using such an object may cause unpredictable results. Possible values are:

0 Collection object did not require repair
1 Collection object was repaired

Offset to repository information. The offset in bytes from the beginning of a receiver variable to the beginning of a repository information structure.

Offset to repository entry. The offset in bytes from the beginning of a receiver variable to the beginning of a particular repository entry.

Partition serial number. The logical serial number of a system partition where the collection object was created.

Repository information. A variable-size field that contains information about repositories in a management collection object. This field contains an array of offset and length pairs (see offset to repository entry field and length of repository entry field), followed by a series of corresponding repository entry structures.

Repository name. The 10-character name of a repository of a management collection object.

Repository size. The size of a repository in Kbytes (K = 1024).

Reserved. A reserved field.

Summarization status. When the Collection Services collector cycles the management collection object, a process is started for this collection object that will extract performance summary information to be used for historical data analysis. The summarization status field indicates the status of this process:

0 Summarization was not performed for this collection object
1 Summarization is complete
2 Summarization is in progress
3 Summarization was attempted but failed

Error Messages

Message ID	Error Message Text
CPF0A2B E	Not able to process management collection object &1 in library &2.
CPF2105 E	Object &1 in &2 type &3 not found.
CPF2110 E	Library &1 not found.
CPF2114 E	Cannot allocate object &1 in &2 type *&3.
CPF2207 E	Not authorized to use object &1 in library &3 type *&2.
CPF24B4 E	Severe error while addressing parameter list.
CPF3C21 E	Format name &1 is not valid.
CPF3C24 E	Length of the receiver variable is not valid.
CPF3CF1 E	Error code parameter not valid.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9810 E	Library &1 not found.

API introduced: V5R2

Top | "Performance Management APIs," on page 1 | APIs by category

User-Defined Transaction APIs

The user-defined transaction APIs are used by an application to gather time interval performance data for application-defined transactions. These APIs indicate the start and end of a user-defined transaction.

The user-defined transaction APIs are:

- "End Transaction (QYPEENDT, qypeEndTransaction) API" (QYPEENDT, qypeEndTransaction) indicates the end of a user-defined transaction.
- "Start Transaction (QYPESTRT, qypeStartTransaction) API" on page 41 (QYPESTRT, qypeStartTransaction) is called at the start of a user-defined transaction.

Top | "Performance Management APIs," on page 1 | APIs by category

End Transaction (QYPEENDT, qypeEndTransaction) API

Required Parameter Group:

1	Application identifier	Input	Char(20)
2	Transaction identifier	Input	Binary(4) Unsigned
3	Application trace data	Input	Char(*)
4	Length of application trace data	Input	Binary(4) Unsigned
5	Transaction start time	Input	Char(8)
6	Application performance counters	Input	Char(*)
7	Length of application performance counters	Input	Binary(4) Unsigned
8	Error code	I/O	Char(*)

Service Program Name: QYPESVPG
Default Public Authority: *USE
Threadsafe: Yes

The End Transaction (OPM, QYPEENDT; ILE, qypeEndTransaction) API is used together with the "Start Transaction (QYPESTRT, qypeStartTransaction) API" on page 41 (QYPESTRT, qypeStartTransaction) API and the "Log Transaction (QYPELOGT, qypeLogTransaction) API" on page 94 (QYPELOGT, qypeLogTransaction) API to collect performance data for user-defined transactions. The End Transaction API is called by an application at the end of a user-defined transaction.

This API can be used to provide both trace type of performance data - collected by Performance Explorer (PEX) - and interval type of performance data - collected by Collection Services.

If the Performance Explorer (PEX) is running, this API generates an end of transaction trace record. In addition to the data supplied by the application in the application trace data parameter, PEX will capture the current values of performance counters associated with the current thread such as CPU time used, I/O activity and seize/lock activity. After the End Performance Explorer (ENDPEX) command is run, the application-supplied data for this record is written to the QMUDTA field in the QAYPEMIUSR file. The performance counters are written to individual fields in the QAYPEMIUSR and QAYPETIDX files.

If Collection Services is collecting data for the user-defined transaction (*USRTNS) category, this API will save transaction performance data for the current transaction. This data includes transaction response time as well as optional performance counters provided by the application in the application performance counters parameter.

See "Usage Notes" on page 42 for the Start Transaction (QYPESTRT, qypeStartTransaction) API for more information.

Authorities and Locks

API Public Authority
*USE

Required Parameter Group

Application identifier
INPUT; CHAR(20)

The name of the application. Given that many applications could use this API, the name should be chosen so that it is unique. Application identifiers starting with "QIBM_Qccc_", where ccc is a component identifier, are reserved for IBM use.

The application identifier is used as the transaction type by Collection Services. The application identifier should be chosen carefully, because Collection Services will only report information about the first 15 unique transaction types for every job which uses user-defined transaction APIs. All other transaction types for each job will be combined in a single type *OTHER.

Transaction identifier
INPUT; BINARY(4) UNSIGNED

Any sort of unique transaction identifier, such as a sequential number. In order to collect meaningful data, the identifier passed to the End Transaction API should be the same as the identifier used in the call to the Start Transaction API for the same transaction.

The transaction identifier is not used by Collection Services.

Application trace data
INPUT; CHAR(*)

Application-defined trace data to be saved by PEX. This can be any data that the user wants to associate with this transaction - for example, the user ID of the client performing the transaction, the name of the file being updated by the transaction, or the account ID being accessed by the transaction. The data can be up to 3032 bytes long. This data is reported by PEX in the QAYPEMIUSR file. Application trace data is not processed by Collection Services.

Length of application trace data
INPUT; BINARY(4) UNSIGNED

The length (in bytes) of application-defined trace data to be saved by PEX. The value must be between 0 and 3032.

Transaction start time
INPUT; CHAR(8)

The time (in MI timestamp format) that the transaction started. The user should provide the transaction start time that was previously returned from the call to the corresponding Start Transaction API. If a null pointer is passed for this parameter, Collection Services will ignore this request. Transaction start time is not used by PEX.

Application performance counters
INPUT; CHAR(*)

Application-provided counter data to be collected by Collection Services. The application can define from 0 to 16 BINARY(8) UNSIGNED counters that Collection Services will collect. These counters may contain any kind of information the application wants to associate with this transaction; for exampel, the number of SQL statements processed to serve the transaction, the number of pages printed for the transaction, and so on. The user should reset these counters just before calling the Start Transaction API and provide these counters when calling the corresponding End Transaction API. If the application trace data is suitably organized - if it is a sequence of BINARY(8) UNSIGNED counters - the application performance counters parameter can be a subset of the application trace data parameter .

Application performance counters are not processed by PEX.

Length of application performance counters
INPUT; BINARY(4) UNSIGNED

The length (in bytes) of the application-provided counter data to be collected by Collection Services. This length can range from 0 (no counters) to 128 (16 BINARY(8) UNSIGNED counters) and must be a multiple of 8.

Error code
I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

Usage Notes

For the description of how Performance Explorer (PEX) and Collection Services save and report performance data for this API, see “Usage Notes” on page 42 for the Start Transaction API.

Error Messages

Message ID	Error Message Text
CPF3C36 E	Number of parameters, &1, entered for this API was not valid.
CPF3C3C E	Value for parameter &1 is not valid.
CPF3CF2 E	Error(s) occurred during running of &1 API.

API introduced: V5R2

Start Transaction (QYPESTRT, qypeStartTransaction) API

Required Parameter Group:

1	Application identifier	Input	Char(20)
2	Transaction identifier	Input	Binary(4) Unsigned
3	Application trace data	Input	Char(*)
4	Length of application trace data	Input	Binary(4) Unsigned
5	Transaction start time	Output	Char(8)
6	Error code	I/O	Char(*)

Service Program Name: QYPESVPG
Default Public Authority: *USE
Threadsafe: Yes

The Start Transaction (OPM, QYPESTRT; ILE, qypeStartTransaction) API is used together with the "End Transaction (QYPEENDT, qypeEndTransaction) API" on page 38 (QYPEENDT, qypeEndTransaction) API and the "Log Transaction (QYPELOGT, qypeLogTransaction) API" on page 94 (QYPELOGT, qypeLogTransaction) API to collect performance data for user-defined transactions. The Start Transaction API is called by an application at the beginning of a user-defined transaction.

This API can be used to provide both trace type of performance data - collected by Performance Explorer (PEX) - and interval type of performance data - collected by Collection Services.

If the Performance Explorer (PEX) is running, this API generates a start of transaction trace record. In addition to the data supplied by the application in the application trace data parameter, PEX will capture the current values of performance counters associated with the current thread such as CPU time used, I/O activity and seize/lock activity. After the End Performance Explorer (ENDPEX) command is run, the application-supplied data for this record is written to the QMUDTA field in the QAYPEMIUSR file (see "Usage Notes" on page 42). The performance counters are written to individual fields in the QAYPEMIUSR and QAYPETIDX files.

If Collection Services is collecting data for the user-defined transaction (*USRTNS) category, this API provides a reference point for the End Transaction API to calculate transaction response time. (See the "End Transaction (QYPEENDT, qypeEndTransaction) API" on page 38 (QYPEENDT, qypeEndTransaction) API).

Authorities and Locks

API Public Authority
*USE

Required Parameter Group

Application identifier
INPUT; CHAR(20)

The name of the application. Given that many applications could use this API, the name should be chosen so that it is unique. Application identifiers starting with "QIBM_Qccc_", where ccc is a component identifier, are reserved for IBM use.

The application identifier is used as the transaction type by Collection Services. The application identifier should be chosen carefully, because Collection Services will only report information

about the first 15 unique transaction types for every job which uses user-defined transaction APIs. All other transaction types for each job will be combined in a single type *OTHER.

Transaction identifier
INPUT; BINARY(4) UNSIGNED

Any sort of unique transaction identifier, such as a sequential number. The transaction identifier is not used by Collection Services.

Application trace data
INPUT; CHAR(*)

Application-defined trace data to be saved by PEX. This can be any data that the user wants to associate with this transaction - for example, the user ID of the client performing the transaction, the name of the file being updated by the transaction, or the account ID being accessed by the transaction. The data can be up to 3032 bytes long. This data is reported by PEX in the QAYPEMIUSR file. Application trace data is not processed by Collection Services. See "Usage Notes" for more information.

Length of application trace data
INPUT; BINARY(4) UNSIGNED

The length (in bytes) of user-defined data to be captured by PEX. The value must be between 0 and 3032.

Transaction start time
OUTPUT; CHAR(8)

The time (in MI timestamp format) that the transaction started. The user should save this value and pass it unchanged to the corresponding End Transaction API. If a null pointer is passed for this parameter, Collection Services will ignore this request. Transaction start time is not used by PEX.

Error code
I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

Usage Notes

How the data is collected

Performance data provided by an application to the user-defined transaction APIs (the Start Transaction API, the End Transaction API and the Log Transaction API) is collected by both Performance Explorer (PEX) and Collection Services.

To capture the trace data provided by the APIs, a Performance Explorer session muct be active for the current thread. To configure PEX to collect data for user-defined transaction APIs, use *USRTNS event on the operating system events (OSEVT) parameter of the Add PEX Definition (ADDPEXDFN) command.

When a PEX session is active for the current thread, a call to the Start Transaction API, the End Transaction API or the Log Transaction API will produce a trace record. In addition to the trace data passed by the application in the application trace data parameter, PEX will capture current values of system performance counters associated with the current thread.

The application trace data is reported by PEX in the QMUDTA field of the QAYPEMIUSR file (see below). The performance counters are reported in individual fields in the QAYPEMIUSR and QAYPETIDX files. This data can be used later to calculate resource consumption for the specific transaction.

Collection Services collects transaction-related statistics in a management collection (*MGTCOL) object when the collection is enabled for the *USRTNS performance category. The Create Performance Data (CRTPFRDTA) command exports the user transaction data collected by Collection Services from the *MGTCOL object to the user-defined transaction (QAPMUSRTNS) file. This file contains one record per transaction type per job per Collection Services collection interval. The data reported in the file includes standard data such as the total transaction time and total number of transactions, as well as optional application-provided counters (see “End Transaction (QYPEENDT, qypeEndTransaction) API” on page 38 (QYPEENDT, qypeEndTransaction) API).

Collection Services summarizes transaction data within a job by transaction type. Collection Services uses the API application identifier parameter as the transaction type. Collection Services stores transaction data for the first 15 transaction types it encounters for a job. If more than 15 transaction types are encountered in a job, it still collects the data; however, the data is combined and reported under a transaction type of *OTHER. (See Collection Services for more information on Collection Services and performance database files.)

How to use collected data

It is the application’s responsibility to decide what constitutes a transaction and to use the user-defined transation APIs in a consistent way; that is, for every call to the Start Transaction API, there should be a call to the End Transaction API with the same transaction identifier.

Performance Explorer collects application-provided trace data as well as current values for a set of system performance counters. These performance counters are a snapshot of the current thread activity when the event was recorded - this means that all numeric values are total numbers for the entire life of the thread (in other words, ″raw″ values as opposed to ″delta″ values).

Note: The performance counters are defined as 8 byte unsigned binary values in the QAYPEMIUSR file. However the data is coming from 4 byte unsigned binary values. For this reason, code which does computations with the values of the performance counters must check for counter wrap - counters will increment from 4,294,967,295 (or FFFFFFFF hexadecimal) to 0. The counters were defined as 8 byte fields to allow expansion in the future.

If a transaction ends in the same thread it was started in, values captured by the Start Transaction API can be subtracted from the values captured by the End Transaction API to get the amount consumed by this transaction:

```
value per transaction =
value from the End Transaction API - value from the Start Transaction API
```

However, if a transaction ends in a different thread than it was started in, this simple subtraction cannot be done. Rather, it is necessary to use the Log Transaction API to record two additional events - one when the transaction in the first thread ends and the other one when the transaction in the second thread begins. Then values per transaction can be be calculated in the following way:

```
value per transaction =
( value from the Log Transaction API in thread 1 - value from the Start Transaction API )
+ ( value from the End Transaction API - value from the Log Transaction API in thread 2 )
```

Collection Services does not collect additional system performance counters for the *USRTNS category. However, Collection Services collects many types of performance data for other performance categories. By joining records from the QAPMUSRTNS file with records from other performance database files produced for the same collection interval, one can calculate average resource consumption for transactions of different types.

The format of the QMUDTA field of the QAYPEMIUSR file

The QMUDTA field has a common header. The following APIs use this header:

- Start Transaction (QYPESTRT, qypeStartTransaction)
- "End Transaction (QYPEENDT, qypeEndTransaction) API" on page 38
- "Log Transaction (QYPELOGT, qypeLogTransaction) API" on page 94
- "Add Trace Point (QYPEADDT, qypeAddTracePoint) API" on page 92

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(4)	″API ″ eye catcher
4	4	CHAR(20)	Application identifier
24	18	CHAR(1)	Type of data: *0* Generic trace point *1* Start of transaction *2* End of transaction *3* Log transaction

After the common header, the QMUDTA field has the following format for the Start Transaction, End Transaction, and Log Transaction APIs:

Offset			
Dec	**Hex**	**Type**	**Field**
25	19	CHAR(10)	Transaction identifier
35	23	CHAR(1)	Reserved
36	24	BINARY(4) UNSIGNED	Length of application trace data
40	28	CHAR(*)	Application trace data

Error Messages

Message ID	Error Message Text
CPF3C36 E	Number of parameters, &1, entered for this API was not valid.
CPF3C3C E	Value for parameter &1 is not valid.
CPF3CF2 E	Error(s) occurred during running of &1 API.

API introduced: V5R2

Top | "Performance Collector APIs" | APIs by category

Performance Collector APIs

The performance collector APIs allow applications to be developed that provide real-time performance monitoring capabilities. The performance collector APIs are:

- "List Performance Data (QPMLPFRD) API" on page 45 (QPMLPFRD) returns data from the data collector to the user space specified in theWork with Collector API.
- "Work with Collector (QPMWKCOL) API" on page 88 (QPMWKCOL) controls the starting and stopping of collections of information for certain types of resources. This API allows you to change the way data about a certain resource is collected.

Note: The Work with Collector API must be used before using the List Performance Data API.

The Performance Collector exit program is:

- "Performance Monitor Exit Program" on page 108 processes the performance data that is collected by the performance monitor as the monitor ends.

Note: Starting in Version 5 Release 1, the Start Performance Monitor (STRPFRMON) command is no longer supported.

Top | "Performance Management APIs," on page 1 | APIs by category

List Performance Data (QPMLPFRD) API

Required Parameter Group:

1	Type of resource	Output	Char(10)
2	Sequence number of collection	Output	Binary(4)
3	Error Code	I/O	Char(*)

Default Public Authority: *EXCLUDE
Threadsafe: Yes

The List Performance Data (QPMLPFRD) API returns the latest collection of performance data in the user space specified for the resource on the Work with Collector (QPMWKCOL) API. When QPMLPFRD is called, it will return data immediately if any data is available; otherwise, it will wait until data becomes available and then return the data. QPMLPFRD only returns data for one type of a resource at a time. The user cannot specify the type of data that is returned by QPMLPFRD. It returns whatever resource data is available at the time. The type of resource must be tested to determine its type if more than one type is collected. The call to QPMLPFRD returns the type of resource data and the sequence number of the collection. The sequence number is increased by the user-specified interval time and can be used to see if a collection was missed. For example, if job data is being collected at 15-second intervals and the previous call to QPMLPFRD returned a sequence number of 265, the sequence number for the next collection retrieved should be 280 or else a collection was missed. This API should be called once per interval for each type of resource data being collected.

The data returned by QPMLPFRD is in a raw format and the user needs to perform delta calculations on the data before it can be used (just as in the sequence number example above). Deltas are the difference between the latest data numbers and the previous data numbers. For example, a user requests job data every 15 seconds using the Work with Collector (QPMWKCOL) API. The user then calls QPMLPFRD to copy the latest collection of data into the resource's user space. The transaction count for JOB1 is 256. In the previous collection, the transaction count for JOB1 was 251. By subtracting the previous data from the current data (256 - 251), the number of transactions performed by JOB1 in that 15 seconds is found. Thus, the user must retain the previous interval's data to calculate deltas from. After the deltas are calculated, the current data becomes the previous data in preparation for the next interval. Because QPMLPFRD replaces the data in the resource's user space with the new data, the user must either save the previous interval's data of interest or use QPMWKCOL to change the resource's user space before calling QPMLPFRD again. This also means that after starting a collection, a user must call QPMLPFRD twice before he can do any calculations.

Using deltas, the impact of missed collections is lessened by the delta calculations. For example, if you were collecting data at 15-second intervals and the sequence numbers for your last two collections were 315 and 345, a collection was missed. You can still perform delta calculations using these two collections and get data for the time period you missed. Of course, the data represents an interval of 30 seconds instead of 15. However, you should never calculate deltas for a period of greater than 4 minutes (the longest collection interval). When the time between collections exceeds 4 minutes, there is a risk that

counters will wrap twice. Because there is no way to tell if counters wrapped twice, the user would perform delta calculations as normal and end up with inaccurate data.

There are three situations to consider when calculating deltas. First, the data between the two collections must be matched by item. When doing this, be aware that the number of items reported and their order could change from one collection to another due to jobs starting and ending, communications lines varying on and off, disks and IOPs being replaced and added, as well as the order that the devices report in with their data.

The second situation is when the raw-data counters wrap. For example, assume counter A can hold up to 99 and currently it is set at 96. If the system adds 10 to the count, the value of counter A becomes 6 because when it reaches 100 it starts over again at 0. When a counter wraps, it makes the delta calculation result in a negative number. To compensate for this, a wrap factor equal to the largest number the field can hold plus 1 should be added to the negative delta. The following excerpt of code (written in C) shows how a subroutine can be defined to calculate deltas.

```
int CalculateDelta(int CurrentData, int PrevData)
{
  #define MAXBIN4 2147483647
  int Delta;

  Delta = CurrentData - PrevData;
  if (Delta < 0)
    Delta = (Delta + MAXBIN4) + 1;
  return Delta;
}
```

Note: Some of the fields wrap at smaller values. See the format tables for any differences.

The third situation involves communications lines varying off and on. When a communication line goes from inactive to active, its counters might get reset. If you had been calculating deltas for such a line, you should restart the calculations (save the current data but skip the delta calculation until the next collection is processed). To determine whether you need to restart delta calculations due to a line activation, you can check for a mismatch between the current and previous value of the Number of Vary On Operations field. Also be aware that when a communications line goes inactive, it will be reported for at least one collection in the inactive state and then disappear from the list of lines reported.

QPMLPFRD relies on the Work with Collector (QPMWKCOL) API. First, QPMWKCOL must be used to start a collection before QPMLPFRD can be called. Second, the user space specified for a resource on the QPMWKCOL call is the user space that QPMLPFRD will copy the data into.

The data in the user spaces is replaced only if QPMLPFRD is called. However, internal data spaces get updated with every collection. It is these internal data spaces that are copied to the user spaces when QPMLPFRD is called. Therefore, if you do not call QPMLPFRD for every collection every interval, data will be missed. Although, as mentioned above, deltas can help compensate for missed collections, it is not recommended to make a regular practice of skipping collections. Data can also be missed if the performance collector takes a long time to retrieve it or is unable to retrieve it due to system problems.

The QPMLPFRD and QPMWKCOL APIs allow multiple users to be able to share the same data collection. This sharing minimizes the system overhead when multiple users are collecting data and ensures that each user is getting data consistent and synchronized with other users.

Authorities and Locks

None.

Required Parameter Group

Type of resource
OUTPUT; CHAR(10)

The type of resource the collected data is for. It will be set to one of the following values:

**JOB*	Job-related information ("Job Format" on page 49)
**POOL*	Pool-related information ("Pool Format" on page 54)
**DISK*	Disk-related information ("Disk Format" on page 55)
**IOP*	IOP-related information ("IOP Format" on page 60)
**COMM*	Communications-related information ("Communications Data Formats" on page 63)

Sequence number of collection
OUTPUT; BINARY(4)

The sequence number of this collection of data. This value increases by one for every second.

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 the Generated List

The list of performance data that the QPMLPFRD API returns into the user space consists of:

- A user area
- A generic header
- An input parameter section
- A header section
- A list data section

The user area and generic header are described in User Space Format for List APIs. For details about the remaining items, see the following sections.

When you retrieve list entry information from a user space, you must use the entry size returned in the generic header. The size of each entry may be padded at the end. If you do not use the entry size, the result may not be valid. For examples of how to process lists, see the API Examples.

Except where noted, delta calculations need to be performed on all numeric (BINARY(4)) fields.

Input Parameter Section

For a description of the fields in this format, see "Field Descriptions" on page 48.

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(10)	Type of resource
10	A	CHAR(2)	Reserved
12	C	BINARY(4)	Sequence number of collection

Header Section

For a description of the fields in this format, see "Field Descriptions" on page 48.

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	Type of resource
10	A	CHAR(2)	Reserved
12	C	BINARY(4)	Interval time
16	10	BINARY(4)	Total CPU seconds used
20	14	BINARY(4)	Number of CPUs
24	18	BINARY(4)	Sequence number of collection
28	1C	CHAR(10)	System name
38	26	CHAR(6)	Release level
44	2C	CHAR(12)	Date and time of collection
56	38	BINARY(4)	Total available CPU seconds

Field Descriptions

Date and time of collection. The date and time the data collection interval ended. This will be in the format YYMMDDHHMMSS, where:

YY	Year
MM	Month
DD	Day
HH	Hour
MM	Minute
SS	Second

The date and time reported here are the same as those reported by the generic header, but the generic header also reports the century digit.

Interval time. The number of seconds since the initial data was collected. This value is used in calculations of utilization, counts per second, and other time-based calculations.

Number of CPUs. The number of CPUs configured on the system at the time the data sample was taken. No delta calculation should be performed on this field.

Release level. The version, release, and modification level of the operating system the data was collected on.

Reserved. An ignored field.

Sequence number of collection. The sequence number of the data collection. Each time data is collected, this number is increased by the user-specified interval time. This number might not accurately represent the elapsed time and should not be used in place of interval time for time-related calculations. This number can be used to detect missed intervals.

System name. The name of the system the data was collected on.

Total available CPU seconds. The total amount of CPU time that was available to the partition based on its configured processor capacity. This value should be used to calculate all CPU utilizations; however, it must be used for shared processor partitions and partitions where the configured capacities may change.

For uncapped partitions, the CPU seconds used may exceed the available CPU seconds which indicates the amount of additional uncapped capacity that was used.

Total CPU seconds used. The total number of CPU seconds that are used during the collection interval for all processors.

Type of resource. The type of resource the collected data is for. The possible values for this field are:

**JOB*	Job-related information
**POOL*	Pool-related information
**DISK*	Disk-related information
**IOP*	IOP-related information
**COMM*	Communications-related information

Job Format

For a description of the fields in this format, see "Job Field Descriptions" on page 51.

This format returns data for each job and task that are active in the system when data is collected for the reported interval. Data is not returned for jobs and tasks that end during an interval.

The amount of job and task data that this API returns is limited to the amount of data that can fit in the user and internal job spaces. This results in a limit of approximately 18,000 jobs and tasks reported. While the spaces support thousands of jobs and tasks, the API might take longer than the collection interval to collect data on this many jobs and tasks. Collecting data for a large number of jobs and tasks might also lead to undesirable system performance impacts.

Every job (threaded or not) has a primary thread. The primary thread is the first thread started for any job and it remains active for the duration of the job. A multi-threaded job has additional threads that may start and end at any point during the life of the job.

Only limited support is provided for multi-threaded jobs. When a job has multiple threads, some of the data that is reported is incomplete. Only one entry is returned for a job regardless of how many threads are active within the job. Within this entry some fields are a total of all thread activity while other fields contain data for the primary thread only. Fields that apply only to the primary thread are identified in the table below.

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(10)	Job name
10	A	CHAR(2)	User name
20	14	CHAR(6)	Job number
26	1A	CHAR(1)	Job type
27	1B	CHAR(1)	Job subtype
28	1C	CHAR(1)	Pass-through source job flag
29	1D	CHAR(1)	Pass-through target job flag
30	1E	CHAR(1)	Emulation job flag
31	1F	CHAR(1)	Client Access application job flag
32	20	CHAR(1)	Target DDM job flag
33	21	CHAR(1)	MRT job flag

Offset		Type	Field
Dec	Hex		
34	22	CHAR(1)	System/36 environment job flag
35	23	CHAR(2)	Job priority
37	25	CHAR(2)	Job pool
39	27	CHAR(1)	Machine interactive flag
40	28	CHAR(8)	Reserved
48	30	BINARY(4)	Database CPU time
52	34	BINARY(4)	Time slice
56	38	BINARY(4)	CPU time
60	3C	BINARY(4)	Transaction count
64	40	BINARY(4)	Transaction time
68	44	BINARY(4)	Synchronous database reads[2]
72	48	BINARY(4)	Synchronous database writes[2]
76	4C	BINARY(4)	Synchronous nondatabase reads[2]
80	50	BINARY(4)	Synchronous nondatabase writes[2]
84	54	BINARY(4)	Asynchronous database reads[2]
88	58	BINARY(4)	Asynchronous database writes[2]
92	5C	BINARY(4)	Asynchronous nondatabase reads[2]
96	60	BINARY(4)	Asynchronous nondatabase writes[2]
100	64	BINARY(4)	Communications puts
104	68	BINARY(4)	Communications gets
108	6C	BINARY(4)	Reserved
112	70	BINARY(4)	Binary overflows[2]
116	74	BINARY(4)	Decimal overflows[2]
120	78	BINARY(4)	Floating-point overflows[2]
124	7C	BINARY(4)	Logical database reads
128	80	BINARY(4)	Logical database writes
132	84	BINARY(4)	Miscellaneous database operations
136	88	BINARY(4)	Permanent writes[2]
140	8C	BINARY(4)	Reserved
144	90	BINARY(4)	PAG faults[2]
148	94	BINARY(4)	Number of print lines
152	98	BINARY(4)	Number of print pages
156	9C	BINARY(4)	Active-to-wait transitions[1,2]
160	A0	BINARY(4)	Wait-to-ineligible transitions[1,2]
164	A4	BINARY(4)	Active-to-ineligible transitions[1,2]
168	A8	CHAR(10)	Line description
178	B2	CHAR(10)	Secondary line description
188	BC	CHAR(2)	Task type
190	BE	CHAR(2)	Task type extender
192	C0	BINARY(4)	Threads currently active

Offset		Type	Field
Dec	Hex		
196	C4	BINARY(4)	Thread count
200	C8	BINARY(4)	Pages allocated
204	CC	BINARY(4)	Pages deallocated

Note:

[1] This counter does not wrap at the standard wrap value. Instead, it increments up to 65535, then wraps to 0.

[2] This counter is supported for primary threads only. For multithreaded jobs, the value reported may not reflect the total job activity.

Job Field Descriptions

Active-to-ineligible transitions. The total number of transitions from active state to ineligible state.

Active-to-wait transitions. The total number of transitions from active state to wait state.

Asynchronous database reads. The total number of physical asynchronous read operations for database functions.

Asynchronous database writes. The total number of physical asynchronous write operations for database functions.

Asynchronous nondatabase reads. The total number of physical asynchronous read operations for nondatabase functions.

Asynchronous nondatabase writes. The total number of physical asynchronous write operations for nondatabase functions.

Binary overflows. The number of binary overflows.

Client Access application job flag. This field will be set to 1 if this is a Client Access application job. Otherwise, it will be set to 0.

CPU time. The processing unit time (in milliseconds) used by this job.

Communications gets. The number of communications read (logical) operations. These do not include remote work station activity. They include only activity related to intersystem communication function (ICF) files when the I/O is for an ICF device.

Communications puts. The number of communications writes. These do not include remote work station activity. They include only activity related to ICF files when the I/O is for an ICF device.

Database CPU time. The processing unit time (in milliseconds) used by this job for database processing. For any particular job, this field contains the total database CPU time for all threads of the job.

Decimal overflows. The number of decimal overflows.

Emulation job flag. This field will be set to 1 if this is a emulation job. Otherwise, it will be set to 0.

Floating-point overflows. The number of floating point overflows.

Job name. The name of the job or task. For an interactive job, the system assigns the job the name of the work station where the job started. For a batch job, the name is specified in the command when the job is submitted. For a task name, this field will contain the first 10 characters of the 16-character task name.

Job number. The number of the job (in decimal) or task (in hexadecimal). Note that the numbering system for jobs and tasks is different such that a job and a task could have the same number.

Job pool. The pool that the job ran in.

Job priority. The job's priority over other jobs.

Job subtype. The subtype of the job. The possible values for this file are:

blank	The job has no special subtype
D	Immediate
E	Evoke job (communications batch)
J	Prestart job
P	Print driver job
T	Multiple requester terminal (MRT) job (System/36 environment only)
U	Alternate spool user

Job type. The type of job or task. The possible values for this field are:

A	Autostart job
B	Batch job
I	Interactive job
M	Subsystem monitor job
R	Spooled reader job
S	System job
V	Vertical Licensed Internal Code (VLIC) (tasks only)
W	Spooled writer job
X	Start-control-program-function (SCPF) system job

Line description. The name of the communications line this work station and its controller are attached to. This is only available for remote work stations.

Logical database reads. The number of times the database module was called. This does not include I/O operations to readers/writers, or I/O operations caused by the Copy Spooled File (CPYSPLF) or Display Spooled File (DSPSPLF) command. If SEQONLY(*YES) is specified on the Override with Database File (OVRDBF) command, these numbers show each block of records read, not the number of individual numbers read.

Logical database writes. The number of times the internal database write function was called. This does not include I/O operations to readers/writers, or I/O operations caused by the CPYSPLF or DSPSPLF commands. If SEQONLY(*YES) is specified on OVRDBF command, these numbers show each block of records written, not the number of individual records written.

Machine interactive flag. This field is set to 1 if the machine is counting the processor resources consumed by this job or task against the interactive workload. Otherwise, it is set to 0.

Miscellaneous database operations. The number of update, delete, force-end-of-data, and release operations.

MRT job flag. This field will be set to 1 if this is a multiple requester terminal (MRT) job. Otherwise, it will be set to 0.

Number of print lines. The number of lines written by the program. This does not reflect what is actually printed. Spooled files can be ended or printed with multiple copies.

Number of print pages. The number of pages printed by the program.

PAG faults. The total number of times the process access group (PAG) was referred to, but was not in main storage.

Pages allocated. Total number of pages of temporary and permanent storage which have been allocated by the job since the job started.

Pages deallocated. Total number of pages of temporary and permanent storage which have been deallocated by the job since the job started.

Permanent writes. The number of permanent writes.

Pass-through source job flag. This field will be set to 1 if this is a pass-through source job. Otherwise, it will be set to 0.

Pass-through target job flag. This field will be set to 1 if this is a pass-through target job. Otherwise, it will be set to 0.

Reserved. An ignored field.

Secondary line description. The name of the communications line this work station and its controller are attached to. This is only available for pass-through and emulation.

Synchronous database reads. The total number of physical synchronous read operations for database functions.

Synchronous database writes. The total number of physical synchronous write operations for database functions.

Synchronous nondatabase reads. The total number of physical synchronous read operations for nondatabase functions.

Synchronous nondatabase writes. The total number of physical synchronous write operations for nondatabase functions.

System/36 environment job flag. This field will be set to 1 if this is a System/36 environment job. Otherwise, it will be set to 0.

Target DDM job flag. This field will be set to 1 if this is a target DDM job. Otherwise, it will be set to 0.

Task type. Type of task. Possible values are:

01 Resident task
02 Supervisor task
03 MI process task
04 System/36 environment task

Task type extender. A task type extender identifies the area of functional support provided by the task. See the task type field description for types of functions supported.

For detailed information on task type extender values, see Performance data files: Task type extender.

Thread count The count of the number of threads initiated within the job.

Threads currently active The count of the current number of active threads in the process when the data was sampled. An active thread may be actively running, suspended, or waiting for a resource. No delta calculation should be done on this field.

Time slice. The time slice value in seconds. No delta calculation should be performed on this field.

Transaction count. The number of transactions performed by the job.

Transaction time. The total transaction time (in seconds).

User name. The user profile under which the job is run. The user name is the same as the user profile name and can come from several different sources depending on the type of job. For a task, this field will be blank.

Wait-to-ineligible transitions. Total number of transitions from wait state to ineligible state.

Pool Format

For a description of the fields in this format, see "Pool Field Descriptions."

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Pool number
4	4	BINARY(4)	Activity level
8	8	BINARY(4)	Pool size
12	C	BINARY(4)	Machine-reserved portion
16	10	BINARY(4)	Database faults
20	14	BINARY(4)	Nondatabase faults
24	18	BINARY(4)	Database pages
28	1C	BINARY(4)	Nondatabase pages
32	20	BINARY(4)	Active-to-wait transitions[1]
36	24	BINARY(4)	Wait-to-ineligible transitions[1]
40	28	BINARY(4)	Active-to-ineligible transitions[1]
44	2C	BINARY(4)	Active-to-wait transitions[2]
48	30	BINARY(4)	Wait-to-ineligible transitions[2]
52	34	BINARY(4)	Active-to-ineligible transitions[2]

Note:

1 This counter does not wrap at the standard wrap value. Instead, it increments up to 32767, then wraps to 0.

2 This counter wraps at the standard wrap value. It will increment up to 2,147,483,647 (2^{31} - 1), then wrap to 0.

Pool Field Descriptions

Active-to-ineligible transitions. The total number of transitions by processes assigned to this pool from active state to ineligible state. There are two versions of this counter—one that wraps at the standard wrap value and one that wraps at a nonstandard wrap value.

Active-to-wait transitions. The total number of transitions by processes assigned to this pool from active state to wait state. There are two versions of this counter—one that wraps at the standard wrap value and one that wraps at a nonstandard wrap value.

Activity level. The maximum number of processes that can be active in the machine at the same time. No delta calculation should be performed on this field.

Database faults. The total number of interruptions to processes (not necessarily assigned to this pool) that were required to transfer data into the pool to permit the MI instruction to process the database function.

Database pages. The total number of pages of database data transferred from auxiliary storage to the pool to permit the instruction to run as a consequence of set access state, implicit access group movement, and internal machine actions.

Machine-reserved portion. The amount of storage (in kilobytes) from the pool that is dedicated to machine functions. No delta calculation should be performed on this field.

Nondatabase faults. The total number of interruptions to processes (not necessarily assigned to this pool) that were required to transfer data into the pool to permit the MI instruction to process the nondatabase function.

Nondatabase pages. The total number of pages of nondatabase data transferred from auxiliary storage to the pool to permit the instruction to run as: a consequence of set access state, implicit access group movement, and internal machine actions.

Pool number. The unique identifier of this pool. The value is from 1 to 64. No delta calculation should be performed on this field.

Pool size. The amount of main storage (in kilobytes) assigned to the pool. Note that if a pool is reported with a pool size of zero, then the pool does not exist. No delta calculation should be performed on this field.

Wait-to-ineligible transitions. Total number of transitions by processes assigned to this pool from wait state to ineligible state. There are two versions of this counter—one that wraps at the standard wrap value and one that wraps at a nonstandard wrap value.

Disk Format

One entry will be reported for each disk resource. Typically, there will be one disk resource per disk unit except for a multipath disk unit which has multiple disk resources associated with it.

For a description of the fields in this format, see "Disk Field Descriptions" on page 57.

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Reserved
4	4	BINARY(4)	Reserved
8	8	CHAR(4)	Disk arm number
12	C	CHAR(4)	Disk drive type
16	10	CHAR(1)	Mirror flag
17	11	CHAR(1)	Mirror status
18	12	CHAR(1)	Managed by IOP

Offset		Type	Field
Dec	Hex		
19	13	CHAR(1)	Disk unit category «
20	14	BINARY(4)	Times the arm not busy
24	18	BINARY(4)	Samples taken
28	1C	BINARY(4)	Drive capacity
32	20	BINARY(4)	Drive available space
36	24	BINARY(4)	Blocks read
40	28	BINARY(4)	Blocks written
44	2C	BINARY(4)	Read commands
48	30	BINARY(4)	Write commands
52	34	BINARY(4)	Processor idle loop count
56	38	BINARY(4)	Processor idle loop time
60	3C	BINARY(4)	Seeks > 2/3 of disk
64	40	BINARY(4)	Seeks > 1/3 and < 2/3 of disk
68	44	BINARY(4)	Seeks > 1/6 and < 1/3 of disk
72	48	BINARY(4)	Seeks > 1/12 and < 1/6 of disk
76	4C	BINARY(4)	Seeks < 1/12 of disk
80	50	BINARY(4)	Zero seeks
84	54	BINARY(4)	Buffer overruns
88	58	BINARY(4)	Buffer underruns
92	5C	BINARY(4)	Total queue elements
96	60	CHAR(2)	ASP number
98	62	CHAR(2)	Reserved
100	64	BINARY(4)	Reserved
104	68	BINARY(4)	Reserved
108	6C	CHAR(10)	IOP resource name
118	76	CHAR(10)	Unit resource name
128	80	CHAR(1)	Compressed unit
129	81	CHAR(1)	Reserved
130	82	CHAR(10)	ASP resource name
140	8C	BINARY(4)	ASP number - extended
144	90	CHAR(1)	Multipath unit
145	91	CHAR(1)	Initial path of multipath unit
146	92	CHAR(1)	Production copy of remotely mirrored independent ASP
147	93	CHAR(1)	Mirror copy of remotely mirrored independent ASP
148	94	CHAR(4)	Reserved
» 152	98	BINARY(8)	Drive capacity - extended
160	A0	BINARY(8)	Drive available space - extended «

Disk Field Descriptions

ASP resource name. The resource name of the ASP to which this unit currently is allocated. A value of blanks specifies the system ASP or a basic ASP.

ASP number. The auxiliary storage pool (ASP) to which this unit is currently allocated. The values are:

-1	The ASP number is greater than 99 and cannot be reported in this field. Use the value in the ASP number - extended field instead.
00	This unit currently is not allocated.
01	The system ASP.
02-32	A basic ASP.
33-99	An independent ASP.

ASP number - extended. The field is defined the same as the ASP number field except that it also can report ASP numbers greater than 99. The values reported are:

0	This unit currently is not allocated.
1	The system ASP.
2-32	A basic ASP.
33-255	An independent ASP.

No delta calculation should be done on this field.

Blocks read. The number of blocks read. Block is one sector on the disk drive.

Blocks written. The number of blocks written. Block is one sector on the disk drive.

Buffer overruns. The number of times that data was available to be read into the disk controller buffer from the disk, but the disk controller buffer still contained valid data that was not retrieved by the storage device controller. Consequently, the disk had to take an additional revolution until the buffer was available to accept data.

Buffer underruns. The number of times that the disk controller was ready to transfer data to the disk on a write operation, but the disk controller buffer was empty. The data was not transferred in time by the disk IOP to the disk controller buffer. The disk was forced to take an extra revolution awaiting the data.

Compressed unit. A compressed unit indicator. The values are:

'0'	This unit is not compressed.
'1'	This unit is compressed.

Disk arm number. The identifier of the unit. Each actuator arm on the disk drives that are available to the machine represents a unit of auxiliary storage. The value of the unit number is assigned by the system when the unit is allocated to an ASP. If the disk arm number is 0000, then the arm is not configured. Both arms of a mirrored-arm pair have the same disk arm number and can be differentiated by the mirror flag value.

Disk drive type. The type of disk drive, such as 9332, 9335, or 6100.

» Disk unit category. This field indicates if this disk unit has some special characteristics, which may require a special interpretation of its performance data. The values are:

X'00'	No special category applies
X'01'	This disk unit is located in Enterprise Storage Server (ESS)

Drive available space. The total number of kilobytes of auxiliary storage space that is not currently assigned to objects or to internal machine functions. When this field is set to the largest positive value it can hold (2147483647), the drive capacity is too large for this field and the field Drive available space - extended should be used instead. No delta calculations should be done on this field.

Drive available space - extended. The total number of kilobytes of auxiliary storage space that is not currently assigned to objects or to internal machine functions. No delta calculations should be done on this field.

Drive capacity. The total number of kilobytes of auxiliary storage provided on the unit for the storage of objects and internal machine functions. When this field is set to the largest positive value it can hold (2147483647), the drive capacity is too large for this field and the field Drive capacity - extended should be used instead. No delta calculations should be done on this field.

Drive capacity - extended. The total number of kilobytes of auxiliary storage provided on the unit for the storage of objects and internal machine functions. No delta calculations should be done on this field. «

Initial path of multipath unit. The value of this field is '1' when the disk resource represents the initial path of a multipath disk unit; otherwise, it is '0'. The initial path is the first path observed by the system. As such, it can change after an IPL. The resource name of the initial path can be used for reporting a multipath disk unit under a single resource name.

IOP resource name. System-unique name to identify the IOP.

» Managed by IOP. The flag indicating whether this disk unit is managed by IOP. The values are:

'0' This disk unit is not managed by IOP
'1' This disk unit is managed by IOP «

Mirror copy of remotely mirrored independent ASP. The value of this field is '1' when the disk unit is in a mirror copy of a remotely mirrored independent ASP; otherwise, it is '0'.

Mirror flag. The flag indicating whether this disk arm is locally mirrored. The values are:

blank The arm is not locally mirrored.
'A' The first arm of a locally mirrored pair.
'B' The second arm of a locally mirrored pair.

Mirror status. The status of local mirroring. The values are:

- X'00' - Not mirrored
- X'01' - Active
- X'02' - Resuming
- X'03' - Suspended

Multipath unit. The value of this field is '1' when the disk resource represents a multipath disk unit; otherwise, it is '0'. Performance data will be reported for each disk resource associated with a multipath disk unit. For a multipath disk unit, the following counters come from the device so their values are duplicated for each disk resource reported:

- Processor idle loop count
- Processor idle loop time
- Seeks (6 counters)

- Buffer overruns
- Buffer underruns

Other field values that are duplicated include drive capacity and drive available space.

The arm number and mirror flag of a particular multipath disk unit can be used to identify the entries associated with that unit.

Processor idle loop count. The number of times the disk controller passed through the idle loop. The count is increased differently for the 9332 and 9335 disk drives. For the 9332, this counter is increased only if the disk controller is totally idle (no I/O operations are active). For the 9335, even though the disk controller may be idle and the counter gets increased, an I/O operation can be active (for example, seek is being performed). This field is nonzero for drive types that support a dedicated disk processor and is zero for other drive types.

Processor idle loop time. The time (in hundredths of microseconds) to make one pass through the idle loop. The value reported could be a multiple of the actual processor idle loop time. In that case, the value reported for the processor idle loop count is reduced by the same multiple so that the calculated disk processor utilization is correct. This field applies to drive types that support a dedicated disk processor and is zero for other drive types. No delta calculation should be done on this field.

Production copy of remotely mirrored independent ASP. The value of this field is '1' when the disk unit is in a production copy of a remotely mirrored independent ASP; otherwise, it is '0'.

Read commands. The number of read data commands.

Reserved. An ignored field.

Samples taken. The number of samples taken at approximately two per second.

Seeks < 1/12 of disk. The number of times the arm traveled from its current position to less than 1/12 of the disk on a seek request.

Seeks > 1/12 and < 1/6 of disk. The number of times the arm traveled more than 1/12 but less than 1/6 of the disk on a seek request.

Seeks > 1/6 and < 1/3 of disk. The number of times the arm traveled more than 1/6 but less than 1/3 of the disk on a seek request.

Seeks > 1/3 and < 2/3 of disk. The number of times the arm traveled more than 1/3 but less than 2/3 of the disk on a seek request.

Seeks > 2/3 of disk. The number of times the arm traveled more than 2/3 of the disk on a seek request.

Times the arm not busy. The number of times there were no outstanding I/O operations active at sample time.

Total queue elements. The number of I/O operations waiting service at sample time. This number includes the I/O operation that is in progress. Divide this by the number of samples taken to get the average queue length.

Unit resource name. Typically, there will be one disk (unit) resource per disk unit except for a multipath disk unit which has multiple disk resources associated with it (see *multipath unit*).

Write commands. The number of write data commands.

Zero seeks. The number of times the access arm did not physically move on a seek request. The operation may have resulted in a head switch.

IOP Format

For a description of the fields in this format, see "IOP Field Descriptions" on page 61.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Reserved
4	4	BINARY(4)	Reserved
8	8	CHAR(1)	IOP type
9	9	CHAR(4)	Resource type
13	D	CHAR(3)	Reserved
16	10	BINARY(4)	Idle loop count
20	14	BINARY(4)	Idle loop time
24	18	BINARY(4)	RAM utilization
28	1C	BINARY(4)	IOP system function time
32	20	BINARY(4)	All protocols communications time
36	24	BINARY(4)	SDLC communications time
40	28	BINARY(4)	Asynchronous communications time
44	2C	BINARY(4)	Bisynchronous communications time
48	30	BINARY(4)	X.25 LLC communications time
52	34	BINARY(4)	X.25 PLC communications time
56	38	BINARY(4)	X.25 DLC communications time
60	3C	BINARY(4)	LAN communications time
64	40	BINARY(4)	SDLC short-hold mode time
68	44	BINARY(4)	ISDN LAPE and LAPD time
72	48	BINARY(4)	ISDN Q931 communications time
76	4C	BINARY(4)	Disk time
80	50	CHAR(1)	Function 1 identifier
81	51	CHAR(1)	Function 2 identifier
82	52	CHAR(1)	Function 3 identifier
83	53	CHAR(1)	Function 4 identifier
84	54	CHAR(1)	Function 5 identifier
85	55	CHAR(3)	Reserved
88	58	BINARY(4)	Function 1 time
92	5C	BINARY(4)	Function 2 time
96	60	BINARY(4)	Function 3 time
100	64	BINARY(4)	Function 4 time
104	68	BINARY(4)	Function 5 time
108	6C	BINARY(4)	Processor 2 time
112	70	CHAR(10)	IOP resource name
122	7A	CHAR(2)	Reserved

Offset		Type	Field
Dec	Hex		
124	7C	BINARY(4)	Reserved
128	80	BINARY(4)	Twinaxial time
132	84	BINARY(4)	Other function time
136	88	BINARY(4)	Interrupt level time
140	8C	BINARY(4)	Remote access time

IOP Field Descriptions

All protocols communications time. The total processing unit time (in milliseconds) used by all of the communication protocol tasks that are running in the primary IOP processor. This field only applies to communications and multifunction IOPs. Otherwise, it will be set to 0.

Asynchronous communications time. The total processing unit time (in milliseconds) used by asynchronous communications tasks that are running in the primary IOP processor. This field only applies to communications and multifunction IOPs. Otherwise, it will be set to 0.

Bisynchronous communications time. The total processing unit time (in milliseconds) used by bisynchronous communications tasks that are running in the primary IOP processor. This field only applies to communications and multifunction IOPs. Otherwise, it will be set to 0.

Disk time. The total processing unit time (in milliseconds) used by disk tasks that are running in the primary IOP processor. This field only applies to multifunction IOPs. Otherwise, it will be set to 0.

Function 1-5 identifier. The identifier for additional functions that may be running in the primary IOP processor. Each function identifier has an associated function time value. Function identifier may have the following values:

- X'00' - No time value supplied
- X'11' - Integrated xSeries Server for iSeries pipe task (Integrated xSeries Server for iSeries is also known as file server I/O processor and FSIOP.)
- X'20' - Storage subsystem task
- X'22' - Tape task
- X'23' - Diskette task
- X'24' - Optical task
- X'30' - Communications subsystem task
- X'42' - Localtalk task
- X'43' - Wireless task
- X'50' - Service processor task
- X'60' - Cryptography task

This field only applies to communications and multifunction IOPs. Otherwise, it will be set to X'00'.

Function 1-5 time. The total processing unit time (in milliseconds) used by the IOP function that is running in the primary IOP processor. This field only applies to communications and multifunction IOPs. Otherwise, it will be set to 0.

Idle loop count. The number of times the primary IOP processor ran an idle loop. This is done when the IOP has no work to perform. This count is used with idle loop time to calculate the primary IOP processor utilization in seconds:

```
U = IT - (ILC * ILT / 100,000,000)
```

where:

U is the processor utilization in seconds for the interval
IT is the change in the interval time during the interval
ILC is the change in the idle loop count during the interval
ILT is the idle loop time

Idle loop time. The time (in hundredths of microseconds) for the primary IOP processor to run the idle loop once. The value reported could be a multiple of the actual idle loop time. In that case, the value reported for the idle loop count is reduced by the same multiple so that the calculated IOP processor utilitzation is correct. No delta calculation should be done on this field.

Interrupt level time. The total processing unit time (in milliseconds) used by interrupt level processing that is running in the primary IOP processor. This does not include interrupt level processing time that can be associated with a particular task. This field only applies to multifunction IOPs. Otherwise, it is set to 0.

IOP resource name. System-unique name to identify the IOP.

IOP system function time. The total time (in milliseconds) used by the IOP for basic system function that is running in the primary IOP processor. This field only applies to communications and multifunction IOPs. Otherwise, it will be set to 0.

IOP type. The type of IOP. The possible values for this field are:

C Communications IOP
D Disk IOP
L Local work station IOP
M Multifunction IOP

Note that QPMLPFRD will report I/O processor (IOP) statistics differently starting with Version 3 Release 7. Performance statistics for IOPs introduced in Version 3 Release 7 or later will be reported as multifunction IOPs even if the IOP supports only one of the three IOP functions (communications, disk, or local workstation). There will be no change in the reporting of performance statistics for IOPs introduced before Version 3 Release 7, which will still be reported under the appropriate IOP type (communications, disk, local workstation, or multifunction).

ISDN LAPE and LAPD time. The total processing unit time (in milliseconds) used by integrated services digital network (ISDN) communications tasks that are running in the primary IOP processor. This field only applies to communications and multifunction IOPs. Otherwise, it will be set to 0. The ISDN communications tasks are:

LAPD Link access procedure D-channel
LAPE Enhanced version of LAPD

ISDN Q.931 communications time. The total processing unit time (in milliseconds) used by ISDN Q.931 communications tasks that are running in the primary IOP processor. This field only applies to communications and multifunction IOPs. Otherwise, it will be set to 0.

LAN communications time. The total processing unit time (in milliseconds) used by the token-ring network, Ethernet, frame relay, fiber distributed data interface (FDDI), and asynchronous transfer mode

(ATM) communications tasks that are running in the primary IOP processor. This includes processing time due to token-ring and Ethernet LAN emulation. This field only applies to communications and multifunction IOPs. Otherwise, it will be set to 0.

Other function time. The total processing unit time (in milliseconds) used by other IOP functions that are running in the primary IOP processor. Other functions include those that cannot be reported in the function 1-5 identifier fields because all of the function 1-5 identifier fields are in use. This field applies to communications and multifunction IOPs only. Otherwise, it is set to 0.

Processor 2 time. The utilization (in milliseconds) of the second IOP processor, which handles specialized functions. This field applies to wireless IOPs, and is zero for other IOPs.

RAM utilization. Available local storage (in bytes). The number of bytes of free local storage in the IOP. The free local storage will probably be noncontiguous because of fragmentation. This field only applies to communications and multifunction IOPs. Otherwise, it will be set to 0. No delta calculations should be done on this field.

Remote access time. The total processing unit time (in milliseconds) used by remote access tasks that are running in the primary IOP processor. This field applies to multifunction IOPs only. Otherwise, it is set to 0.

Reserved. An ignored field.

Resource type. The model number of the IOP.

SDLC communications time. The total processing unit time (in milliseconds) used by SDLC communications tasks that are running in the primary IOP processor. This field only applies to communications and multifunction IOPs. Otherwise, it will be set to 0.

SDLC short-hold mode time. The total processing unit time (in milliseconds) that is used by SDLC short-hold mode tasks that are running in the primary IOP processor. This field only applies to communications and multifunction IOPs. Otherwise, it will be set to 0.

Twinaxial time. The total processing unit time (in milliseconds) used by workstation and local twinaxial tasks that are running in the primary IOP processor. This field only applies to multifunction IOPs. Otherwise, it will be set to 0.

X.25 DLC communications time. The total processing unit time (in milliseconds) used by X.25 data link control (DLC) and Point-to-Point Protocol (PPP) communications tasks that are running in the primary IOP processor. This field only applies to communications and multifunction IOPs. Otherwise, it will be set to 0.

X.25 LLC communications time. The total processing unit time (in milliseconds) used by X.25 logical link control (LLC) communications tasks that are running in the primary IOP processor. This field only applies to communications and multifunction IOPs. Otherwise, it will be set to 0.

X.25 PLC communications time. The total processing unit time (in milliseconds) used by X.25 packet layer communications (PLC) tasks that are running in the primary IOP processor. This field only applies to communications and multifunction IOPs. Otherwise, it will be set to 0.

Communications Data Formats

The formats for communications data are handled differently from the formats for other types of resources. All communications protocols are kept in the same space, but, because each protocol has unique data fields, each individual field in the space will have a different meaning depending on the protocol. Therefore, different formats are presented for each protocol. Also, because the protocols vary in

the number of data fields, some protocol formats will not use all the space provided (each record in the space has the same length). The communications data formats are:

- Asynchronous Format ("Asynchronous Format")
- Bisynchronous Format ("Bisynchronous Format" on page 65)
- Token-Ring Format ("Token-Ring Format" on page 67)
- Ethernet Format ("Ethernet Format" on page 73)
- IDLC Format ("IDLC Format" on page 78)
- LAPD Format ("LAPD Format" on page 80)
- SDLC Format ("SDLC Format" on page 82)
- X.25 Format ("X.25 Format" on page 84)
- PPP Format ("PPP Format" on page 86)

Asynchronous Format

For a description of the fields in this format, see "Asynchronous Field Descriptions."

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Reserved
4	4	BINARY(4)	Reserved
8	8	CHAR(1)	Protocol
9	9	CHAR(10)	Line description
19	13	CHAR(1)	Line active
20	14	CHAR(12)	Reserved
32	20	BINARY(4)	Line speed
36	24	BINARY(4)	Number of vary on operations
40	28	BINARY(4)	Active time
44	2C	BINARY(4)	Bytes transmitted
48	30	BINARY(4)	Bytes received
52	34	BINARY(4)	Protocol data units transmitted
56	38	BINARY(4)	Protocol data units received
60	3C	BINARY(4)	Protocol data units received in error
64	40	CHAR(10)	IOP resource name

Asynchronous Field Descriptions

Active time. The amount of time in seconds that the line was active (varied on). This field should be used instead of interval time for all time-dependent fields calculated (for example, line utilization) to get accurate statistics.

Bytes received. The number of bytes received (data and control characters), including characters received in error.

Bytes transmitted. The number of bytes transmitted (data and control characters) including bytes transmitted again because of errors.

IOP resource name. System-unique name to identify the IOP.

Line active. The state of the line when the collection interval ended. The values are:

0 The line is not active.
1 The line is active.

Line description. The name of the description for this line.

Line speed. The speed of this line in bits per second (bps). No delta calculation should be performed on this field.

Number of vary on operations. The total number of vary on operations.

Protocol. Protocol type. This will be set to A for asynchronous.

Protocol data units received. The total number of protocol data units received.

Protocol data units received in error. The total number of protocol data units received with parity and stop bit errors.

Protocol data units transmitted. The total number of protocol data units successfully transmitted and the data circuit-terminating equipment (DCE) acknowledged.

Bisynchronous Format

For a description of the fields in this format, see "Bisynchronous Field Descriptions" on page 66.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Reserved
4	4	BINARY(4)	Reserved
8	8	CHAR(1)	Protocol
9	9	CHAR(10)	Line description
19	13	CHAR(1)	Line active
20	14	CHAR(12)	Reserved
32	20	BINARY(4)	Line speed
36	24	BINARY(4)	Number of vary on operations
40	28	BINARY(4)	Active time
44	2C	BINARY(4)	Bytes transmitted
48	30	BINARY(4)	Bytes received
52	34	BINARY(4)	Data characters transmitted
56	38	BINARY(4)	Data characters received
60	3C	BINARY(4)	Data characters retransmitted
64	40	BINARY(4)	Data characters received in error
68	44	BINARY(4)	Characters received in error
72	48	BINARY(4)	NAK received to text sent
76	4C	BINARY(4)	Wrong ACK to text sent
80	50	BINARY(4)	Enquiry to text sent
84	54	BINARY(4)	Invalid (unrecognized) format

Offset		Type	Field
Dec	Hex		
88	58	BINARY(4)	Enquiry to ACK
92	5C	BINARY(4)	Disconnect received (abort)
96	60	BINARY(4)	EOT received (abort)
100	64	BINARY(4)	Disconnect received (forward abort)
104	68	BINARY(4)	EOT received (forward abort)
108	6C	BINARY(4)	Data blocks transmitted
112	70	BINARY(4)	Data blocks received
116	74	CHAR(10)	IOP Resource name

Bisynchronous Field Descriptions

Active time. The amount of time in seconds that the line was active (varied on). This field should be used instead of interval time for all time-dependent fields calculated (for example, line utilization) to get accurate statistics.

Bytes received. The number of bytes received (data and control characters), including bytes received in error.

Bytes transmitted. The number of bytes transmitted (data and control characters) including bytes transmitted again because of errors.

Characters received in error. The number of characters received with a block-check character error.

Data blocks received. The number of data blocks received.

Data blocks transmitted. The number of data blocks transmitted.

Data characters received. The number of data characters received successfully (excluding synchronous characters) while in data mode.

Data characters received in error. The number of data characters received with a block-check character error while in data mode.

Data characters retransmitted. The number of data characters transmitted again.

Data characters transmitted. The number of data characters transmitted successfully while in data mode.

Disconnect received (abort). The number of times the remote station issued a disconnect with abnormal end. This could occur when error recovery did not succeed or the binary synchronous job was ended.

Disconnect received (forward abort). The number of times the host station issued a disconnect with abnormal end. This could occur when error recovery did not succeed or the binary synchronous job was ended.

EOT received (abort). The end of transmission was received (abnormal end). This is similar to a disconnect.

EOT received (forward abort). The end of transmission was received (forward abnormal end). This is similar to a disconnect.

Enquiry to ACK. An enquiry to acknowledged character. The remote station returned an acknowledgment (for example, ACK0), and the host system sent an ENQ character. This indicates that the host station did not recognize the acknowledgment as a valid acknowledgment.

Enquiry to text sent. The number of times text was sent by a station and an ENQ character was returned. The receiving station expected some form of acknowledgment, such as an ACK0, ACK1, or NAK.

IOP resource name. System-unique name to identify the IOP.

Invalid (unrecognized) format. The number of times one of the delimiter characters that encloses the data in brackets being sent or received is not valid.

Line active. The state of the line when the collection interval ended. The values are:

0 The line is not active.
1 The line is active.

Line description. The name of the description for this line.

Line speed. The speed of this line in bits per second (bps). No delta calculation should be performed on this field.

NAK received to text sent. Negative acknowledgment character received to text sent. The number of times the remote station did not understand the command sent from the host system.

Number of vary on operations. The total number of vary on operations.

Protocol. The protocol type. This will be set to B for bisynchronous.

Reserved. An ignored field.

Wrong ACK to text sent. Wrong acknowledgment character to text sent. The host system received an acknowledgment from the remote device that was not expected. For example, the host system expected an ACK0 and received an ACK1.

Token-Ring Format

Token-ring format was formerly known as establishment communications link (ECL) format.

This format reports token-ring LAN protocol statistics for asynchronous transfer mode (ATM) ports that support token-ring LAN emulation and for token-ring ports.a

For a description of the fields in this format, see "Token-Ring Field Descriptions" on page 69.

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Reserved
4	4	BINARY(4)	Reserved
8	8	CHAR(1)	Protocol
9	9	CHAR(10)	Line description
19	13	CHAR(1)	Line active
20	14	CHAR(12)	Reserved
32	20	BINARY(4)	Line speed

Offset		Type	Field
Dec	Hex		
36	24	BINARY(4)	Number of vary on operations
40	28	BINARY(4)	Active time
44	2C	BINARY(4)	I-frame characters transmitted
48	30	BINARY(4)	I-frame characters received
52	34	BINARY(4)	RNR frames transmitted
56	38	BINARY(4)	RNR frames received
60	3C	BINARY(4)	Reject frames transmitted
64	40	BINARY(4)	Reject frames received
68	44	BINARY(4)	I-frames transmitted
72	48	BINARY(4)	I-frames received
76	4C	BINARY(4)	SABME frames transmitted
80	50	BINARY(4)	SABME frames received
84	54	BINARY(4)	N2 retries expiration count
88	58	BINARY(4)	T1 timer expiration count
92	5C	BINARY(4)	Frames transmitted[2]
96	60	BINARY(4)	Frames received[2]
100	64	BINARY(4)	Routing I-frames transmitted[1,2]
104	68	BINARY(4)	Routing I-frames received[1,2]
108	6C	BINARY(4)	Line errors[2]
112	70	BINARY(4)	Internal errors[1,2]
116	74	BINARY(4)	Burst error[2]
120	78	BINARY(4)	ARI/FCI error[2]
124	7C	BINARY(4)	Abort delimiter[2]
128	80	BINARY(4)	Lost frame[1,2]
132	84	BINARY(4)	Receive congestion[2]
136	88	BINARY(4)	Frame-copied error[2]
140	8C	BINARY(4)	Frequency error[1,2]
144	90	BINARY(4)	Token error[1,2]
148	94	BINARY(4)	Direct memory access bus error[1,2]
152	98	BINARY(4)	Direct memory access parity error[1,2]
156	9C	BINARY(4)	Address not recognized[1,2]
160	A0	BINARY(4)	Frame-not-copied error[1,2]
164	A4	BINARY(4)	Transmit strip error[1,2]
168	A8	BINARY(4)	Unauthorized AP[1,2]
172	AC	BINARY(4)	Unauthorized MAC frame[1,2]
176	B0	BINARY(4)	Soft error[1,2]
180	B4	BINARY(4)	Transmit beacon[1,2]
184	B8	BINARY(4)	IOA status overrun[1,2]
188	BC	BINARY(4)	Frames discarded[1,2]
192	C0	BINARY(4)	Spurious interrupts[1,2]

Offset		Type	Field
Dec	Hex		
196	C4	BINARY(4)	Total MAC bytes received OK
200	C8	BINARY(4)	Total MAC bytes transmitted OK
204	CC	BINARY(4)	Total frames not transmitted - hardware error[2]
208	D0	BINARY(4)	Ring use count[2]
212	D4	BINARY(4)	Ring sample count[2]
216	D8	BINARY(4)	FCS/code violations in repeated frames[2]
220	DC	BINARY(4)	Frames transmitted and failed to return[2]
224	E0	BINARY(4)	Number of underruns[2]
228	E4	CHAR(10)	IOP resource name
238	EE	CHAR(1)	Duplex
239	EF	CHAR(1)	Reserved
240	F0	BINARY(8)	Line speed - long
248	F8	BINARY(8)	I-frame characters transmitted - long
256	100	BINARY(8)	I-frame characters received - long
264	108	BINARY(8)	I-frames transmitted - long
272	110	BINARY(8)	I-frames received - long
280	118	BINARY(8)	Frames transmitted - long[2]
288	120	BINARY(8)	Frames received - long[2]
296	128	BINARY(8)	Routing I-frames transmitted - long[1,2]
304	130	BINARY(8)	Routing I-frames received - long[1,2]
312	138	BINARY(8)	Total MAC bytes received OK - long
320	140	BINARY(8)	Total MAC bytes transmitted OK - long
328	148	BINARY(4)	Unsupported protocol frames[1,2]
332	14C	BINARY(4)	Reserved

Notes:

1 Not applicable for file server I/O processor.

2 Not applicable for token-ring LAN emulation over ATM.

Token-Ring Field Descriptions

ARI/FCI error. Address-recognized indicator or frame-copied indicator error. This is a physical control field-extension field error.

Abort delimiter. The number of times an abnormal ending delimiter was transmitted because of an internal error.

Active time. The amount of time in seconds that the line was active (varied on). This field should be used instead of interval time for all time-dependent fields calculated (for example, line utilization) to get accurate statistics.

Address not recognized. Total number of frames with address-not-recognized error.

Burst error. The number of burst errors. Burst of same polarity is detected by the physical unit after the starting delimiter of a frame or token.

Direct memory access bus error. Direct memory access (DMA) error for the IOP/IOA bus.

Direct memory access parity error. DMA parity error for the IOP/IOA.

Duplex. The duplex state of the line. For some lines, this value might change over time. This field can have the following values:

blank	The duplex state is not known.
F	Full duplex: The line can simultaneously transmit and receive data.
H	Half duplex: The line can either transmit data or receive data, but the line cannot simultaneously transmit and receive data.

FCS/code violations in repeated frames. This counter is incremented for every frame that has a code violation or fails the frame check sequence (FCS) cyclic redundancy check.

Frame-copied error. The number of times a frame with a specific destination address was copied by another adapter.

Frames discarded. The total number of frames discarded.

Frame-not-copied error. Total number of frames with frame not copied error.

Frames received. The total number of frames (logical link control (LLC) and medium access control (MAC)) received. For high-speed lines, this counter might wrap multiple times during the interval, resulting in a calculated delta value that is incorrect. To avoid this problem, use the frames received - long field.

Frames received - long. The same as the frames received field, but larger.

Frames transmitted. The total number of frames (LLC and MAC) transmitted. For high-speed lines, this counter might wrap multiple times during the interval, resulting in a calculated delta value that is incorrect. To avoid this problem, use the frames transmitted - long field.

Frames transmitted - long. The same as the frames transmitted field, but larger.

Frames transmitted and failed to return. This counter is incremented when a transmitted frame fails to return.

Frequency error. The number of frequency errors on the adapter.

I-frame characters received. The total number of characters received in all I-frames. For high-speed lines, this counter might wrap multiple times during the interval, resulting in a calculated delta value that is incorrect. To avoid this problem, use the I-frame characters received - long field.

I-frame characters received - long. The same as the I-frame characters received field, but larger.

I-frame characters transmitted. The total number of characters transmitted in all I-frames. For high-speed lines, this counter might wrap multiple times during the interval, resulting in a calculated delta value that is incorrect. To avoid this problem, use the I-frame characters transmitted - long field.

I-frame characters transmitted - long. The same as the I-frame characters transmitted field, but larger.

I-frames received. The number of I-frames received. For high-speed lines, this counter might wrap multiple times during the interval, resulting in a calculated delta value that is incorrect. To avoid this problem, use the I-frames received - long field.

I-frames received - long. The same as the I-frames received field, but larger.

I-frames transmitted. The number of I-frames transmitted, excluding I-frames transmitted again. For high-speed lines, this counter might wrap multiple times during the interval, resulting in a calculated delta value that is incorrect. To avoid this problem, use the I-frames transmitted - long field.

I-frames transmitted - long. The same as the I-frames transmitted field, but larger.

IOA status overrun. The number of adapter interrupt status queue overruns. The earliest results are discarded.

IOP resource name. System-unique name to identify the IOP.

Internal errors. The number of adapter internal errors.

0 The line is not active.
1 The line is active.

Line description. The name of the description for this line.

Line errors. The number of code violations of frame-check sequence errors.

Line speed. The speed of this line in bits per second (bps). No delta calculation should be performed on this field. A value of -1 is reported if the size of this field is too small to report the actual value. When -1 is reported, the actual value must be obtained from the line speed - long field.

Line speed - long. The same as the line speed field, but larger.

Lost frame. The number of times the adapter could not remove its own frame from the ring.

N2 retries expiration count. This count is updated when the host has attempted to contact a station *n* times, and the T1 timer ended *n* times before the station responded.

Number of underruns. This counter is incremented each time a DMA underrun is detected.

Number of vary on operations. The total number of vary on operations.

Protocol. Protocol type. This will be set to E for establishment communications link (ECL).

Receive congestion. The number of times a frame was not copied because no buffer was available for the IOA to receive.

Reject frames received. The number of reject frames received.

Reject frames transmitted. The number of reject frames transmitted.

Reserved. An ignored field.

Ring sample count. The number of times the ring use count was sampled or accumulated.

```
Ring utilization % = Ring use count/Ring sample count
```

Ring use count. The number of times the ring was in use.

Routing I-frames received. Total number of frames (LLC and MAC) with a routing-information field received. For high-speed lines, this counter might wrap multiple times during the interval, resulting in a calculated delta value that is incorrect. To avoid this problem, use the routing I-frames received - long field.

Routing I-frames received - long. The same as the routing I-frames received field, but larger.

Routing I-frames transmitted. Total number of frames (LLC and MAC) with a routing-information field transmitted. For high-speed lines, this counter might wrap multiple times during the interval, resulting in a calculated delta value that is incorrect. To avoid this problem, use the routing I-frames transmitted - long field.

Routing I-frames transmitted - long. The same as the routing I-frames transmitted field, but larger.

RNR frames received. The number of receive-not-ready frames received.

RNR frames transmitted. The number of receive-not-ready frames transmitted.

SABME frames received. The number of set-asynchronous-balanced-mode-extended frames received.

SABME frames transmitted. The number of set-asynchronous-balanced-mode-extended frames transmitted.

Soft error. The total number of soft errors as reported by the adapter. A soft error is an intermittent error on a network that requires retransmission.

Spurious interrupts. The total number of interrupts that medium access control (MAC) could not decode.

T1 timer expiration count. The number of times the T1 timer ended.

Token error. When this adapter serves as ring monitor, the number of times the adapter token timer ended without detecting any frames or tokens on the ring.

Total frames not transmitted - hardware error. A count of frames that could not be transmitted due to the hardware not signaling transmission completion for an excessive period of time.

Total MAC bytes received OK. The count of bytes in frames that were successfully received. It includes bytes from received multicast and broadcast frames. This number includes everything, starting from destination address up to but excluding FCS. For high-speed lines, this counter might wrap multiple times during the interval, resulting in a calculated delta value that is incorrect. To avoid this problem, use the total MAC bytes received OK - long field.

Total MAC bytes received OK - long. The same as the total MAC bytes received OK field, but larger.

Total MAC bytes transmitted OK. The total number of bytes transmitted successfully. This number includes everything, starting from destination address up to but excluding FCS. For high-speed lines, this counter might wrap multiple times during the interval, resulting in a calculated delta value that is incorrect. To avoid this problem, use the total MAC bytes transmitted OK - long field.

Total MAC bytes transmitted OK - long. The same as the total MAC bytes transmitted OK field, but larger.

Transmit beacon. The total number of beacon frames transmitted.

Transmit strip error. The total number of adapter-frame-transmit or frame-stripping-process errors.

Unauthorized AP. Unauthorized access priority. The number of times the access priority request is not authorized.

Unauthorized MAC frame. The number of unauthorized MAC frames. The adapter is not authorized to send a MAC frame if:

- A source class is specified.
- The MAC frame has a source class of zero.
- MAC frame physical-control-attention field is greater than 1.

Unsupported protocol frames. Number of frames that were discarded because they specified an unsupported protocol. This count is included in the frames discarded counter.

Ethernet Format

This format reports Ethernet LAN protocol statistics for asynchronous transfer mode (ATM) ports that support Ethernet LAN emulation and for Ethernet ports.

For a description of the fields in this format, see "Ethernet Field Descriptions" on page 75.

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Reserved
4	4	BINARY(4)	Reserved
8	8	CHAR(1)	Protocol
9	9	CHAR(10)	Line description
19	13	CHAR(1)	Line active
20	14	CHAR(12)	Reserved
32	20	BINARY(4)	Line speed
36	24	BINARY(4)	Number of vary on operations
40	28	BINARY(4)	Active time
44	2C	BINARY(4)	I-frame characters transmitted
48	30	BINARY(4)	I-frame characters received
52	34	BINARY(4)	RNR frames transmitted
56	38	BINARY(4)	RNR frames received
60	3C	BINARY(4)	Reject frames transmitted
64	40	BINARY(4)	Reject frames received
68	44	BINARY(4)	I-frames transmitted
72	48	BINARY(4)	I-frames received
76	4C	BINARY(4)	SABME frames transmitted
80	50	BINARY(4)	SABME frames received
84	54	BINARY(4)	N2 retries expiration count
88	58	BINARY(4)	T1 timer expiration count
92	5C	BINARY(4)	Total frames transmitted[4]
96	60	BINARY(4)	Total frames received[4]
100	64	BINARY(4)	Inbound frames missed[1,4]

Offset		Type	Field
Dec	Hex		
104	68	BINARY(4)	CRC error[4]
108	6C	BINARY(4)	More than 16 retries[4]
112	70	BINARY(4)	Out-of-window collisions[1,2,4]
116	74	BINARY(4)	Alignment error[4]
120	78	BINARY(4)	Carrier loss[2,4]
124	7C	BINARY(4)	Time domain reflectometry[1,2,4]
128	80	BINARY(4)	Receive buffer errors[1,4]
132	84	BINARY(4)	Spurious interrupts[1,2,4]
136	88	BINARY(4)	Discarded inbound frames[1,4]
140	8C	BINARY(4)	Receive overruns[4]
144	90	BINARY(4)	Memory error[1,2,4]
148	94	BINARY(4)	Interrupt overrun[1,4]
152	98	BINARY(4)	Transmit underflow[4]
156	9C	BINARY(4)	Babble errors[1,2,4]
160	A0	BINARY(4)	Signal quality error[1,2,4]
164	A4	BINARY(4)	More than one retry to transmit[4]
168	A8	BINARY(4)	Exactly one retry to transmit[2,4]
172	AC	BINARY(4)	Deferred conditions[4]
176	B0	BINARY(4)	Transmit frames discarded[3,4]
180	B4	BINARY(4)	Total MAC bytes received OK
184	B8	BINARY(4)	Total MAC bytes transmitted OK
188	BC	BINARY(4)	Total frames not transmitted - hardware error[4]
192	C0	BINARY(4)	Total mail frames discarded[3,4]
196	C4	CHAR(10)	IOP resource name
206	CE	CHAR(1)	Duplex
207	CF	CHAR(1)	Reserved
208	D0	BINARY(8)	Line speed - long
216	D8	BINARY(8)	I-frame characters transmitted - long
224	E0	BINARY(8)	I-frame characters received - long
232	E8	BINARY(8)	I-frames transmitted - long
240	F0	BINARY(8)	I-frames received - long
248	F8	BINARY(8)	Total frames transmitted - long[4]
256	100	BINARY(8)	Total frames received - long[4]
264	108	BINARY(8)	Total MAC bytes reeived OK - long
272	110	BINARY(8)	Total MAC bytes transmitted OK - long
280	118	BINARY(4)	Unsupported protocol frames[1,2,4]
284	11C	BINARY(4)	Reserved

Offset		Type	Field
Dec	Hex		
Notes:			
[1] Not applicable for file server I/O processor.			
[2] Not applicable for wireless LAN support.			
[3] Wireless LAN support only.			
[4] Not applicable for Ethernet LAN emulation over ATM.			

Ethernet Field Descriptions

Active time. The amount of time in seconds that the line was active (varied on). This field should be used instead of interval time for all time-dependent fields calculated (for example, line utilization) to get accurate statistics.

Alignment error. The number of times an inbound frame contained a noninteger number of bytes and a cyclic-redundancy-check (CRC) error.

Babble errors. The number of times the transmitter exceeded the maximum allowable time on the channel.

Carrier loss. Access to the network has been disconnected.

CRC error. The number of cyclic-redundancy-check (CRC) errors detected by the receiver.

Deferred conditions. The number of times the chip set on the IOAs deferred transmission due to a busy channel.

Discarded inbound frames. The number of receiver-discarded frames due to the lack of queue entries.

Duplex. The duplex state of the line. For some lines, this value might change over time. This field can have the following values:

blank	The duplex state is not known.
F	Full duplex: The line can simultaneously transmit and receive data.
H	Half duplex: The line can either transmit data or receive data, but the line cannot simultaneously transmit and receive data.

Exactly one retry to transmit. The number of frames that required one retry for successful transmission.

I-frame characters received. The total number of characters received in all I-frames. For high-speed lines, this counter might wrap multiple times during the interval, resulting in a calculated delta value that is incorrect. To avoid this problem, use the I-frame characters received - long field.

I-frame characters received - long. The same as the I-frame characters received field, but larger.

I-frame characters transmitted. The total number of characters transmitted in all I-frames. For high-speed lines, this counter might wrap multiple times during the interval, resulting in a calculated delta value that is incorrect. To avoid this problem, use the I-frame characters transmitted - long field.

I-frame characters transmitted - long. The same as the I-frame characters transmitted field, but larger.

I-frames received. The number of I-frames received. For high-speed lines, this counter might wrap multiple times during the interval, resulting in a calculated delta value that is incorrect. To avoid this problem, use the I-frames received - long field.

I-frames received - long. The same as the I-frames received field, but larger.

I-frames transmitted. The number of I-frames transmitted, excluding I-frames transmitted again. For high-speed lines, this counter might wrap multiple times during the interval, resulting in a calculated delta value that is incorrect. To avoid this problem, use the I-frames transmitted - long field.

I-frames transmitted - long. The same as the I-frames transmitted field, but larger.

IOP resource name. System-unique name to identify the IOP.

Inbound frames missed. The number of times a receive buffer error or missed frame was detected by the IOA.

Interrupt overrun. The number of interrupts not processed due to lack of status queue entries.

Line active. The state of the line when the collection interval ended. The values are:

0 The line is not active.
1 The line is active.

Line description. The name of the description for this line.

Line speed. The speed of this line in bits per second (bps). For some lines, this value might change over time. No delta calculation should be performed on this field. A value of -1 is reported if the size of this field is too small to report the actual value. When -1 is reported, the actual value must be obtained from the line speed - long field.

Line speed - long. The same as the line speed field, but larger.

Memory error. The number of times the IOA did not receive a ready signal within 25.6 microseconds of asserting the address on the data or address lines.

More than one retry to transmit. The number of frames that required more than one retry for successful transmission.

More than 16 retries. The number of frames unsuccessfully transmitted due to excessive retries.

N2 retries expiration count. This count is updated when the host has attempted to contact a station *n* times, and the T1 timer ended *n* times before the station responded.

Number of vary on operations. The total number of vary on operations.

Out-of-window collisions. The number of collisions that occurred after the allotted time interval for a collision to occur and after the transmission attempt to be retried.

Protocol. Protocol type. This will be set to T for Ethernet.

Reserved. An ignored field.

RNR frames received. The number of receive-not-ready frames received.

RNR frames transmitted. The number of receive-not-ready frames transmitted.

Receive buffer errors. The number of hardware buffer overflows that occurred upon receiving a frame.

Receive overruns. The number of times the receiver has lost all or part of an incoming frame due to buffer shortage.

Reject frames received. The number of reject frames received.

Reject frames transmitted. The number of reject frames transmitted.

SABME frames received. The number of set-asynchronous-balanced-mode-extended frames received.

SABME frames transmitted. The number of set-asynchronous-balanced-mode-extended frames transmitted.

Signal quality error. The number of times a signal indicating the transmit is successfully complete did not arrive within 2 microseconds of successful transmission.

Spurious interrupts. The number of times an interrupt was received but could not be decoded into a recognizable interrupt.

T1 timer expiration count. The number of times the T1 timer ended.

Time domain reflectometry. Counter used to approximate distance to a cable fault. This value is associated with the last occurrence of more than 16 retries.

Total frames not transmitted - hardware error. A count of frames that could not be transmitted due to the hardware not signaling transmission completion for an excessive period of time.

Total frames received. The total number of type II frames received. For high-speed lines, this counter might wrap multiple times during the interval, resulting in a calculated delta value that is incorrect. To avoid this problem, use the total frames received - long field.

Total frames received - long. The same as the total frames received field, but larger.

Total frames transmitted. The total number of type II frames transmitted. For high-speed lines, this counter might wrap multiple times during the interval, resulting in a calculated delta value that is incorrect. To avoid this problem, use the total frames transmitted - long field.

Total frames transmitted - long. The same as the total frames transmitted field, but larger.

Total MAC bytes received OK. The count of bytes in frames that were successfully received. It includes bytes from received multicast and broadcast frames. This number includes everything, starting from destination address up to but excluding FCS. For high-speed lines, this counter might wrap multiple times during the interval, resulting in a calculated delta value that is incorrect. To avoid this problem, use the total MAC bytes received OK - long field.

Total MAC bytes received OK - long. The same as the total MAC bytes received OK field, but larger.

Total MAC bytes transmitted OK. The total number of bytes transmitted successfully. This number includes everything, starting from destination address up to but excluding FCS. For high-speed lines, this counter might wrap multiple times during the interval, resulting in a calculated delta value that is incorrect. To avoid this problem, use the total MAC bytes transmitted OK - long field.

Total MAC bytes transmitted OK - long. The same as the total MAC bytes transmitted OK field, but larger.

Total mail frames discarded. The number of stored and forward mail products dropped.

Transmitted frames discarded. The number of outbound frames discarded by input/output adapter (IOA).

Transmit underflow. The number of times the transmitter has truncated a message due to the late data received from main storage.

Unsupported protocol frames. Number of frames that were discarded because they specified an unsupported protocol. This count is included in the discarded inbound frames counter.

IDLC Format

For a description of the fields in this format, see "IDLC Field Descriptions" on page 79.

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Reserved
4	4	BINARY(4)	Reserved
8	8	CHAR(1)	Protocol
9	9	CHAR(10)	Line description
19	13	CHAR(10)	Network interface description
29	1D	CHAR(1)	Line active
30	1E	CHAR(2)	Reserved
32	20	BINARY(4)	Line speed
36	24	BINARY(4)	Number of vary on operations
40	28	BINARY(4)	Active time
44	2C	BINARY(4)	Bytes transmitted
48	30	BINARY(4)	Bytes received
52	34	BINARY(4)	Receive CRC errors
56	38	BINARY(4)	Short frame errors
60	3C	BINARY(4)	Aborts received
64	40	BINARY(4)	Sequence errors
68	44	BINARY(4)	Frames transmitted
72	48	BINARY(4)	Frames retransmitted
76	4C	BINARY(4)	Frames received
80	50	BINARY(4)	Frames received in error
84	54	CHAR(1)	B1 channel
85	55	CHAR(1)	B2 channel
86	56	CHAR(10)	IOP resource name
96	60	CHAR(4)	B channel used

IDLC Field Descriptions

Aborts received. The number of frames received that contained high-level data link control (HDLC) abort indicators. This indicates that the remote equipment ended frames before they were complete.

Active time. The amount of time in seconds that the line was active (varied on). This field should be used instead of interval time for all time-dependent fields calculated (for example, line utilization) to get accurate statistics.

B channel used. The B channel used is associated with a bit in this field being set to 1. Bit 0 (most significant bit) and 31 (least significant bit) are reserved. Bits 1 to 30 are associated with B channels 30 to 1, respectively.

B1 channel. The user can send data on this channel. This is set to 1 if the B1 channel was used.

B2 channel. The user can send data on this channel. This is set to 1 if the B2 channel was used.

Bytes received. The total number of bytes received from the remote link station. This includes no errors.

Bytes transmitted. The total number of bytes transmitted to a remote link station. This includes bytes retransmitted and bytes sent on transmissions stopped by transmit underrun, in addition to successful transmissions.

Frames received. Total number of information (I), unnumbered information (UI), and supervisory (S) frames received from the remote link station. This includes no errors.

Frames received in error. The sum of receive CRC errors, short frame errors, overrun, underrun, aborts received, and frame sequence errors.

Frames retransmitted. The number of frames that required retransmission due to errors. Errors can be caused by a remote device that is failing or by not receiving data fast enough.

Frames transmitted. Total number of information (I), unnumbered information (UI), and supervisory (S) frames sent to a remote link station. This includes frames retransmitted and frames sent on transmissions stopped by transmit underruns, in addition to successful transmissions.

IOP resource name. System-unique name to identify the IOP.

Line active. The state of the line when the collection interval ended. The values are:

0 The line is not active.
1 The line is active.

Line description. The name of the description for this line.

Line speed. The speed of this line in bits per second (bps). No delta calculation should be performed on this field.

Network interface description. The name of the network interface description.

Number of vary on operations. The total number of vary on operations.

Protocol. Protocol type. This will be set to I for IDLC.

Reserved. An ignored field.

Receive CRC errors. The number of received frames that contain a cyclic-redundancy-check (CRC) error. This indicates that the data was not received error-free.

Sequence errors. The number of frames received during the time interval that contained sequence numbers indicating that frames were lost.

Short frame errors. The number of short frames received. A short frame is a frame that has fewer octets between its start flag and end flag than are permitted.

LAPD Format

For a description of the fields in this format, see "LAPD Field Descriptions" on page 81.

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Reserved
4	4	BINARY(4)	Reserved
8	8	CHAR(1)	Protocol
9	9	CHAR(10)	Network interface description
19	13	CHAR(1)	Line active
20	14	CHAR(12)	Reserved
32	20	BINARY(4)	Line speed
36	24	BINARY(4)	Number of vary on operations
40	28	BINARY(4)	Active time
44	2C	BINARY(4)	Bytes transmitted
48	30	BINARY(4)	Bytes received
52	34	BINARY(4)	Loss of frame alignment
56	38	BINARY(4)	Reserved
60	3C	BINARY(4)	Reserved
64	40	BINARY(4)	Reserved
68	44	BINARY(4)	Reserved
72	48	BINARY(4)	Errored Seconds
76	4C	BINARY(4)	Severely Errored Seconds
80	50	BINARY(4)	Collision detect
84	54	BINARY(4)	Receive CRC errors
88	58	BINARY(4)	Short frame errors
92	5C	BINARY(4)	Aborts received
96	60	BINARY(4)	Sequence errors
100	64	BINARY(4)	Frames transmitted
104	68	BINARY(4)	Frames retransmitted
108	6C	BINARY(4)	Frames received
112	70	BINARY(4)	Frames received in error
116	74	BINARY(4)	Total outgoing calls
120	78	BINARY(4)	Retry for outgoing calls
124	7C	BINARY(4)	Total incoming calls
128	80	BINARY(4)	Retry for incoming calls

Offset		Type	Field
Dec	Hex		
132	84	CHAR(1)	S1 maintenance channel
133	85	CHAR(10)	IOP resource name

LAPD Field Descriptions

Aborts received. The number of frames received that contained high-level data link control (HDLC) abort indicators. This indicates that the remote equipment ended frames before they were complete.

Active time. The amount of time in seconds that the line was active (varied on). This field should be used instead of interval time for all time-dependent fields calculated (for example, line utilization) to get accurate statistics.

Bytes received. The total number of bytes received from the remote link station. This includes no errors.

Bytes transmitted. The total number of bytes transmitted to a remote link station. This includes bytes retransmitted and bytes sent on transmissions stopped by transmit underrun, in addition to successful transmissions.

Collision detect. The number of times the terminal equipment (TE) detected that its transmitted frame has been corrupted by another TE attempting to use the same bus.

Errored Seconds. The number of seconds that had one or more Path Coding Violations, one or more Out of Frame defects, one or more Controlled Slip events, or a detected Alarm Indication Signal defect.

Frames received. The total number of information (I), unnumbered information (UI), and supervisory (S) frames received from the remote link station. This includes no errors.

Frames received in error. The sum of receive CRC errors, short frame errors, overrun, underrun, aborts received, and frame sequence errors.

Frames retransmitted. The number of frames requiring retransmission due to errors. Errors can be caused by a remote device that is failing or cannot receive data fast enough.

Frames transmitted. The total number of information (I), unnumbered information (UI), and supervisory (S) frames sent to a remote link station. This includes frames retransmitted and frames sent on transmissions stopped by transmit underrun, in addition to successful transmissions.

IOP resource name. System-unique name to identify the IOP.

Line active. The state of the line when the collection interval ended. The values are:

0 The line is not active.
1 The line is active.

Line speed. The speed of this line in bits per second (bps). No delta calculation should be performed on this field.

Loss of frame alignment. The total number of times when a time period equivalent to two 48-bit frames has elapsed without having detected valid pairs of line code violations.

Network interface description. The name of the network interface description.

Number of vary on operations. The total number of vary on operations.

Protocol. Protocol type. This will be set to D for LAPD.

Receive CRC errors. The number of frames received that contain a cyclic-redundancy-check (CRC) error.

Reserved. An ignored field.

Retry for incoming calls. The number of incoming calls that were rejected by the network.

Retry for outgoing calls. The number of outgoing calls that were rejected by the network.

S1 maintenance channel. This field will be set to one if this ISDN had maintenance channel active.

Sequence errors. The number of received frames that contained sequence numbers that indicated frames were lost.

Severely Errored Seconds.

- For ESF signals, the number of seconds that had 320 or more Path Coding Violation error events, one or more Out of Frame defects, or a detected Alarm Indication Signal defect.
- For E1-CRC signals, the number of seconds that had 832 or more Path Coding Violation error events, or one or more Out of Frame defects.
- For E1-noCRC signals, the number of seconds that had 2048 or more Line Coding Violations.
- For D4 signals, the number of seconds that had Framing Error events, an Out of Frame defect, or 1544 or more Line Coding Violations.

Short frame errors. The number of short frames received. A short frame is a frame that has fewer octets between its start flag and end flag than are permitted.

Total incoming calls. The total number of incoming call attempts.

Total outgoing calls. The total number of outgoing call attempts.

SDLC Format

For a description of the fields in this format, see "SDLC Field Descriptions" on page 83.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Reserved
4	4	BINARY(4)	Reserved
8	8	CHAR(1)	Protocol
9	9	CHAR(10)	Line description
19	13	CHAR(1)	Line active
20	14	CHAR(12)	Reserved
32	20	BINARY(4)	Line speed
36	24	BINARY(4)	Number of vary on operations
40	28	BINARY(4)	Active time
44	2C	BINARY(4)	Bytes transmitted
48	30	BINARY(4)	Bytes received
52	34	BINARY(4)	I-frames retransmitted

Offset		Type	Field
Dec	Hex		
56	38	BINARY(4)	Error-free frames received
60	3C	BINARY(4)	Frames received in error
64	40	BINARY(4)	Invalid frames received
68	44	BINARY(4)	Link resets
72	48	BINARY(4)	I-frames transmitted
76	4C	BINARY(4)	Frames retransmitted
80	50	BINARY(4)	RR frames transmitted
84	54	BINARY(4)	RR frames received
88	58	BINARY(4)	RNR frames transmitted
92	5C	BINARY(4)	RNR frames received
96	60	BINARY(4)	Polling wait time
100	64	CHAR(10)	IOP resource name

SDLC Field Descriptions

Active time. The amount of time in seconds that the line was active (varied on). This field should be used instead of interval time for all time-dependent fields calculated (for example, line utilization) to get accurate statistics.

Bytes received. The number of bytes received (data and control characters), including bytes received in error.

Bytes transmitted. The number of bytes transmitted (data and control characters) including bytes transmitted again because of errors.

Error-free frames received. The number of I-frames, supervisory frames, and frames not numbered that were received without error (whether or not they were transmitted again from the remote side).

Frames received in error. The number of I-frames, supervisory frames, and frames not numbered that were received in error. The following are the error possibilities:

- A supervisory frame or I-frame was received with an Nr count that is requesting retransmission of a frame.
- An I-frame was received with an Ns count that indicates that frames were missed.
- A frame is received with a frame-check-sequence error, an abnormal end, a receive overrun, or a frame-truncated error.

Frames retransmitted. The number of I-frames, supervisory frames, and frames not numbered that were transmitted again.

I-frames retransmitted. The number of I-frames transmitted again.

I-frames transmitted. The number of I-frames transmitted.

IOP resource name. System-unique name to identify the IOP.

Invalid frames received. The number of invalid frames received. These are frames received with either a short frame error (frame is less than 32 bits) or a residue error (frame is not on a byte boundary).

Line active. The state of the line when the collection interval ended. The values are:

0 The line is not active.
1 The line is active.

Line description. The name of the description for this line.

Line speed. The speed of this line in bits per second (bps). No delta calculation should be performed on this field.

Link resets. The number of times a set normal response mode (SNRM) was received when the station was already in normal response mode.

Number of vary on operations. The total number of vary on operations.

Polling wait time. The length of time (in tenths of seconds) that the system waits for the response to a poll while in normal disconnect mode before polling the next station. No delta calculation should be done on this field.

Protocol. Protocol type. This will be set to S for SDLC.

RNR frames received. The number of receive-not-ready supervisory frames received.

RNR frames transmitted. The number of receive-not-ready supervisory frames transmitted.

RR frames received. The number of receive-ready supervisory frames received.

RR frames transmitted. The number of receive-ready supervisory frames transmitted.

X.25 Format

For a description of the fields in this format, see "X.25 Field Descriptions" on page 85.

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Reserved
4	4	BINARY(4)	Reserved
8	8	CHAR(1)	Protocol
9	9	CHAR(10)	Line description
19	13	CHAR(1)	Line active
20	14	CHAR(12)	Reserved
32	20	BINARY(4)	Line speed
36	24	BINARY(4)	Number of vary on operations
40	28	BINARY(4)	Active time
44	2C	BINARY(4)	Bytes transmitted
48	30	BINARY(4)	Bytes received
52	34	BINARY(4)	I-frames retransmitted
56	38	BINARY(4)	Frames received in error
60	3C	BINARY(4)	Invalid frames received
64	40	BINARY(4)	Link resets

Offset		Type	Field
Dec	Hex		
68	44	BINARY(4)	I-frames transmitted
72	48	BINARY(4)	Error-free frames received
76	4C	BINARY(4)	RR frames transmitted
80	50	BINARY(4)	RR frames received
84	54	BINARY(4)	RNR frames transmitted
88	58	BINARY(4)	RNR frames received
92	5C	BINARY(4)	Reset packets transmitted
96	60	BINARY(4)	Reset packets received
100	64	CHAR(10)	IOP resource name

X.25 Field Descriptions

Active time. The amount of time in seconds that the line was active (varied on). This field should be used instead of interval time for all time-dependent fields calculated (for example, line utilization) to get accurate statistics.

Bytes received. The number of bytes received (data and control characters), including bytes received in error.

Bytes transmitted. The number of bytes transmitted (data and control characters) including bytes transmitted again because of errors.

Error-free frames received. The number of I-frames, supervisory frames, and frames not numbered that were received without error (whether or not they were transmitted again from the remote side).

Frames received in error. The number of I-frames, supervisory frames, and frames not numbered that were received in error. The following are the error possibilities:

- A supervisory frame or I-frame was received with an Nr count that is requesting retransmission of a frame.
- An I-frame was received with an Ns count that indicates that frames were missed.
- A frame is received with a frame-check-sequence error, an abnormal end, a receive overrun, or a frame-truncated error.

I-frames retransmitted. The number of I-frames transmitted again.

I-frames transmitted. The number of I-frames transmitted excluding I-frames transmitted again.

IOP resource name. System-unique name to identify the IOP.

Invalid frames received. The number of invalid frames received. These are frames received with either a short frame error (frame is less than 32 bits) or a residue error (frame is not on a byte boundary).

Line active. The state of the line when the collection interval ended. The values are:

0 The line is not active.
1 The line is active.

Line description. The name of the description for this line.

Line speed. The speed of this line in bits per second (bps). No delta calculation should be performed on this field.

Link resets. The number of times a set normal response mode (SNRM) was received when the station was already in normal response mode.

Number of vary on operations. The total number of vary on operations.

Protocol. Protocol type. This will be set to X for X.25.

Reserved. An ignored field.

Reset packets received. The number of reset packets received.

Reset packets transmitted. The number of reset packets transmitted.

RNR frames received. The number of receive-not-ready supervisory frames received.

RNR frames transmitted. The number of receive-not-ready supervisory frames transmitted.

RR frames received. The number of receive-ready supervisory frames received.

RR frames transmitted. The number of receive-ready supervisory frames transmitted.

PPP Format

For a description of the fields in this format, see "PPP Field Descriptions" on page 87.

PPP lines are full duplex.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Reserved
4	4	BINARY(4)	Reserved
8	8	CHAR(1)	Protocol
9	9	CHAR(10)	Line description
19	13	CHAR(1)	Line active
20	14	CHAR(8)	Reserved
28	1C	BINARY(8)	Line speed
36	24	BINARY(4)	Number of vary on operations
40	28	BINARY(4)	Active time
44	2C	CHAR(10)	IOP resource name
54	36	CHAR(2)	Reserved
56	38	BINARY(8)	Bytes transmitted
64	40	BINARY(8)	Bytes received
72	48	BINARY(8)	Frames transmitted
80	50	BINARY(8)	Error-free frames received
88	58	BINARY(4)	Frames received in error
92	5C	BINARY(4)	Invalid frames received

PPP Field Descriptions

Active time. The amount of time in seconds that the line was active (varied on). This field should be used instead of interval time for all time-dependent fields calculated (for example, line utilization) to get accurate statistics.

Bytes received. The number of bytes received, including all bytes in frames that had any kind of error.

Bytes transmitted. The number of bytes transmitted, including bytes transmitted again.

Error-free frames received. The number of frames received without error.

Frames received in error. The number of frames received with one of the following errors: a frame check sequence error, an abnormal end, a receive overrun, or a frame truncated error.

Frames transmitted. The number of frames transmitted.

Invalid frames received. The number of frames received with a residue error (frame is not on a byte boundary).

IOP resource name. System-unique name to identify the IOP.

Line active. The state of the line when the collection interval ended. The values are:

0 The line is not active.
1 The line is active.

Line description. The name of the description for this line.

Line speed. The speed of this line in bits per second (bps). No delta calculation should be performed on this field.

Number of vary on operations. The total number of vary on operations.

Protocol. Protocol type. This is set to P for PPP.

Reserved. An ignored field.

Error Messages

Message ID	Error Message Text
CPF0A42 E	Collector ended abnormally.
CPF0A43 E	Data not available.
CPF0A44 E	Collection not active for user.
CPF0A45 E	Cannot copy data to user space &1.
CPF0A47 E	User space &1 in lib &2 not large enough.
CPF24B4 E	Severe error while addressing parameter list.
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: V2R3

"Change System Collector Attributes (QYPSCSCA, QypsChgSysCollectorAttributes) API" on page 6 | "Performance Management APIs," on page 1 | APIs by category

Work with Collector (QPMWKCOL) API

Required Parameter Group:

1	Type of action to perform	Input	Char(10)
2	Type of resource	Input	Char(10)
3	Time between collections	Input	Binary(4)
4	Qualified user space name	Input	Char(20)
5	First sequence number	Output	Binary(4)
6	Error Code	I/O	Char(*)

Default Public Authority: *EXCLUDE
Threadsafe: Conditional; see "Usage Notes" on page 91.

The Work with Collector (QPMWKCOL) API starts, ends, or changes the collection of performance data for a particular resource on your system. The performance collector used by the API has the following attributes:

- It collects data for user-specified resource types that can include job, pool, disk, input/output processor (IOP), and communications.
- It collects data at time intervals ranging from 15 seconds to four minutes.
- It deposits the collected data into user spaces.
- Multiple users can collect data at the same time.
- The same resource type data can be collected by different users with the same or different interval lengths.
- Different resources can be collected at different interval times.
- It will collect data until all collections have been explicitly ended or all of the user's jobs have ended.
- It does not calculate deltas on the data collected. For more information on deltas, see the "List Performance Data (QPMLPFRD) API" on page 45 (QPMLPFRD) API.
- It will report job data for a job or task that is active when the data is collected. A job or task that terminates during an interval will no longer be reported.
- Its intent is to provide performance data useful for real-time monitoring of system performance. Thus, it will not support all of the counters that collection services supports. It is not intended to be used in place of collection services for detailed performance analysis, capacity planning, and other such functions.

When the first user of the collector issues a call to the QPMWKCOL API, two jobs (QPMASERV and QPMACLCT) are submitted to batch. QPMASERV acts as a server, communicating between the APIs (QPMWKCOL and QPMLPFRD) and the QPMACLCT job, which does the actual data collection. These jobs run at priority 0 in subsystem QSYSWRK. No matter how many users are collecting data, there will only be one instance of each job running. The programs will continue to run until all users have ended all of their collections. They will also end if none of the users' jobs are still active.

To start a data collection for a resource, call the QPMWKCOL API using the following:

- The value *START for the type of action to perform parameter
- The type of resource data to collect (job, pool, disk, input/output processor (IOP), or communications)
- The length of time between collections (15, 30, 60, 120, or 240 seconds)
- The name and library of the user space the data should be copied into. The user space must be created in the system ASP or in a basic ASP and not in an independent ASP. This ensures that the server job QPMASERV, which processes the API request, can access the user space.
- The error code parameter

A separate request must be made for each resource desired. When the request is valid, the sequence number of the first collection will be passed back to the user. The sequence number is increased by the

user-specified interval time and can be used to see if an interval collection was missed. For example, if 15 is the first sequence number received back from the QPMWKCOL API but 30 is the sequence number received back from the List Performance Data (QPMLPFRD) API, you missed the collection of data with a sequence number of 15.

By using *CHANGE for the type of action to perform parameter, the interval time or the user space for a resource can be changed.

To end a collection, use *END for the type of action to perform parameter. Because a collection will continue to be active until it is ended, it is important that any collections that are no longer needed be ended. If an *END request is from the last user of a resource, data collection of the resource stops. If not, the resource will still be collected, but this user will no longer have access to data until a *START request is issued again. After the last user of the collector ends all of his collections, the collector jobs (QPMASERV and QPMACLCT) end.

Because QPMWKCOL works with the QPMLPFRD API, the parameters selected for QPMWKCOL will affect QPMLPFRD. For example, the interval time selected determines how often QPMLPFRD should be called. Because the new data replaces old data, if QPMLPFRD is not called before the next interval is collected, the data from the previous interval will be lost, although the deltas may still be calculated for the longer interval. The qualified user space is also an important parameter to QPMLPFRD because this is the space that QPMLPFRD will copy the performance data into. If the space is not large enough to hold all the data or if the space is locked, an error message will be issued to the user.

Starting in Version 5 Release 1, performance data is collected by the performance data collector used by collection services. This has the following implications:

- The collection services performance data collector (QYPSPFRCOL) job runs in addition to the QPMASERV and QPMACLCT jobs.
- Performance data is stored in the management collection (*MGTCOL) object, allowing it to be processed by the create performance data (CRTPFRDTA) command.
- The collection services performance data collector supports data collection at one-minute intervals, but not at two- or four-minute intervals. Therefore, when the API user requests data at two- or four-minute intervals, the data will be collected at one-minute intervals, but reported back to the user every two or four minutes.

Authorities and Locks

User Space Authority
*CHANGE

Library Authority
*EXECUTE

User Space Lock
*EXCLRD

Required Parameter Group

Type of action to perform
INPUT; CHAR(10)

Whether you want to start, end, or change the collection of a resource. The following values may be specified:

**START*	Start the collection of the specified resource.
**END*	End the collection of the specified resource.
**CHANGE*	Change the collection of the specified resource.

Type of resource
INPUT; CHAR(10)

The type of resource to start, end, or change. The following values may be specified.

**JOB*	Job-related information
**POOL*	Pool-related information
**DISK*	Disk-related information
**IOP*	IOP-related information
**COMM*	Communications-related information

Time between collections
INPUT; BINARY(4)

The number of seconds between each new collection of data. The following values may be specified.

15	Collect every 15 seconds.
30	Collect every 30 seconds.
60	Collect every 60 seconds.
120	Collect every 120 seconds (2 minutes).
240	Collect every 240 seconds (4 minutes).

Notes:

1. The disk- and IOP-related data require a minimum of 30 seconds between collections.
2. The communication-related data requires a minimum of 60 seconds between collections.
3. The jobs-related data should be collected as infrequently as possible to minimize the impact on system performance.

Qualified user space name
INPUT; CHAR(20)

The name of the user space that is to receive the data for this type of resource. The first 10 characters contain the user space name, and the second 10 characters contain the name of the library where the user space is located. The user space must be created in the system ASP or in a basic ASP and not in an independent ASP. This ensures that the server job QPMASERV, which processes the API request, can access the user space. The special values for the library name are:

**LIBL*	Library list is used.
**CURLIB*	Current library is used.

The library name value is resolved when this API is called. If no library is specified as the current library for the job, QGPL is used. Both entries are left-justified.

First sequence number
OUTPUT; BINARY(4)

The sequence number of the first data collection that will be available for the user.

Error code
I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

Usage Notes

The QPMWKCOL API has been classified as conditionally threadsafe. This classification is the result of a dependency on the Submit Job (SBMJOB) command, which has been classified conditionally threadsafe. Refer to the SBMJOB command in the Control Language information for restrictions.

Error Messages

Message ID	Error Message Text
CPF0A37 E	Request type &1 not valid.
CPF0A38 E	Resource type &1 not valid.
CPF0A39 E	Interval time of &1 seconds not valid.
CPF0A40 E	Interval time of &1 seconds for IOP data not valid.
CPF0A41 E	&1 seconds for communications data not valid.
CPF0A42 E	Collector ended abnormally.
CPF0A44 E	Collection not active for user.
CPF0A45 E	Cannot copy data to user space &1.
CPF0A46 E	Interval time of &1 seconds for disk data not valid.
CPF0A47 E	User space &1 in lib &2 not large enough.
CPF24B4 E	Severe error while addressing parameter list.
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: V2R3

Top | "Performance Management APIs," on page 1 | APIs by category

---

Performance Explorer (PEX) APIs

For information about performance explorer (PEX), see Performance explorer.

The PEX APIs are used to collect trace performance data for user-defined (application-defined) transactions and to record application-defined trace data. The user-defined transaction APIs indicate the start and end of a transaction and allow logging during a transaction.

The PEX APIs are:

- "Add Trace Point (QYPEADDT, qypeAddTracePoint) API" on page 92 (QYPEADDT, qypeAddTracePoint) records application-defined trace data.
- "End Transaction (QYPEENDT, qypeEndTransaction) API" on page 38 (QYPEENDT, qypeEndTransaction) indicates the end of a user-defined transaction.
- "Log Transaction (QYPELOGT, qypeLogTransaction) API" on page 94 (QYPELOGT, qypeLogTransaction) generates a transaction log record in the PEX trace data.
- "Retrieve PEX Information (QYPERPEX, qypeRetrievePexInfo) API" on page 95 (QYPERPEX, qypeRetrievePexInfo) returns a list of active Performance Explorer collections.
- "Start Transaction (QYPESTRT, qypeStartTransaction) API" on page 41 (QYPESTRT, qypeStartTransaction) is called at the start of a user-defined transaction.

Top | "Performance Management APIs," on page 1 | APIs by category

Add Trace Point (QYPEADDT, qypeAddTracePoint) API

Required Parameter Group:

1	Application identifier	Input	Char(20)
2	Event subtype identifier	Input	Char(10)
3	Application trace data	Input	Char(*)
4	Length of application trace data	Input	Binary(4) Unsigned
5	Error code	I/O	Char(*)

Service Program Name: QYPESVPG
Default Public Authority: *USE
Threadsafe: Yes

The Add Trace Point (OPM, QYPEADDT; ILE, qypeAddTracePoint) API is used to record application-defined trace data.

If Performance Explorer (PEX) is running, this API generates a trace record of the type specified in the event subtype identifier parameter. In addition to the data supplied by the application in the application trace data parameter, PEX will capture the current values of performance counters associated with the current thread such as CPU time used, I/O activity and seize/lock activity. After the End Performance Explorer (ENDPEX) command is run, the application trace data is written to the QMUDTA field in the QAYPEMIUSR file (see "Usage Notes" on page 93). The performance counters are written to individual fields in the QAYPEMIUSR and QAYPETIDX files.

Authorities and Locks

API Public Authority
*USE

Required Parameter Group

Application identifier
INPUT; CHAR(20)

The name of the application. Given that many applications could use this API, the name should be chosen so that it is unique. Application identifiers starting with "QIBM_Qccc_", where ccc is a component identifier, are reserved for IBM use.

Event subtype identifier
INPUT; CHAR(10)

The Performance Explorer (PEX) event subtype to be used for the trace record. Allowed values for this parameter are:

- **APPEVT1*
- **APPEVT2*
- **APPEVT3*
- **APPEVT4*

To configure PEX to collect data generated by this API, use the same event subtype identifier on the application events (APPEVT) parameter of the Add PEX Definition (ADDPEXDFN) command.

Application trace data
INPUT; CHAR(*)

Application-defined trace data to be saved by PEX. This can be any data that the user wants to associate with this trace record. The data can be up to 3042 bytes long. This data is reported by PEX in the QAYPEMIUSR file.

Length of application trace data
INPUT; BINARY(4) UNSIGNED

The length (in bytes) of application-defined trace data to be saved by PEX. The value must be between 0 and 3042.

Error code
I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

Usage Notes

Application-defined trace data is reported in the QMUDTA field of the QAYPEMIUSR file.

The format of the QMUDTA field of the QAYPEMIUSR file is described below.

The QMUDTA field has a common header. The following APIs use this header:

- "Start Transaction (QYPESTRT, qypeStartTransaction) API" on page 41
- "End Transaction (QYPEENDT, qypeEndTransaction) API" on page 38
- "Log Transaction (QYPELOGT, qypeLogTransaction) API" on page 94
- Add Trace point (QYPEADDT, qypeAddTracePoint)

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(4)	″API ″ eye catcher
4	4	CHAR(20)	Application identifier
24	18	CHAR(1)	Type of data: *0* Generic trace point *1* Start of transaction *2* End of transaction *3* Log transaction

After the common header, the QMUDTA field has the following format for the Add Trace Point API:

Offset			
Dec	**Hex**	**Type**	**Field**
25	19	CHAR(1)	Reserved
26	1A	BINARY(4) UNSIGNED	Length of application trace data
30	1E	CHAR(*)	Application trace data

Error Messages

Message ID	Error Message Text
CPF3C36 E	Number of parameters, &1, entered for this API was not valid.
CPF3C3C E	Value for parameter &1 is not valid.

API introduced: V5R2

Log Transaction (QYPELOGT, qypeLogTransaction) API

Required Parameter Group:

1	Application identifier	Input	Char(20)
2	Transaction identifier	Input	Binary(4) Unsigned
3	Application trace data	Input	Char(*)
4	Length of application trace data	Input	Binary(4) Unsigned
5	Error code	I/O	Char(*)

Service Program Name: QYPESVPG
Default Public Authority: *USE
Threadsafe: Yes

The Log Transaction (OPM, QYPELOGT; ILE, qypeLogTransaction) API is used together with the "Start Transaction (QYPESTRT, qypeStartTransaction) API" on page 41 (QYPESTRT, qypeStartTransaction) API and the "End Transaction (QYPEENDT, qypeEndTransaction) API" on page 38 (QYPEENDT, qypeEndTransaction) API to collect performance data for user-defined transactions. The Log Transaction API is called by an application any time between the calls to the Start Transaction API and the End Transaction API to trace the progress of a user-defined transaction.

This API can be used to provide trace type of performance data - collected by Performance Explorer (PEX). Collection Services ignores this API.

If the Performance Explorer (PEX) is running, this API generates a log transaction trace record. In addition to the data supplied by the application in the application trace data parameter, PEX will capture the current values of performance counters associated with the current thread such as CPU time used, I/O activity and seize/lock activity. After the End Performance Explorer (ENDPEX) command is run, the application-supplied data for this record is written to the QMUDTA field in the QAYPEMIUSR file. The performance counters are written to individual fields in the QAYPEMIUSR and QAYPETIDX files.

See "Usage Notes" on page 42 for the Start Transaction (QYPESTRT, qypeStartTransaction) API for more information.

Authorities and Locks

API Public Authority
*USE

Required Parameter Group

Application identifier
INPUT; CHAR(20)

The name of the application. Given that many applications could use this API, the name should be chosen so that it is unique. Application identifiers starting with "QIBM_Qccc_", where ccc is a component identifier, are reserved for IBM use.

Transaction identifier
INPUT; BINARY(4) UNSIGNED

Any sort of unique transaction identifier, such as a sequential number. In order to collect meaningful data, the identifier passed to the Log Transaction API should be the same as the identifier used in the call to the Start Transaction API for the same transaction.

Application trace data
INPUT; CHAR(*)

Application-defined trace data to be saved by PEX. This can be any data that the user wants to associate with this transaction - for example, the user ID of the client performing the transaction, the name of the file being updated by the transaction, or the account ID being accessed by the transaction. The data can be up to 3032 bytes long. This data is reported by PEX in the QAYPEMIUSR file.

Length of application trace data
INPUT; BINARY(4) UNSIGNED

The length (in bytes) of application-defined trace data to be saved by PEX. The value must be between 0 and 3032.

Error code
I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

Usage Notes

For the description of how Performance Explorer (PEX) saves and reports performance data for this API, see "Usage Notes" on page 42 for the Start Transaction API.

Error Messages

Message ID	Error Message Text
CPF3C36 E	Number of parameters, &1, entered for this API was not valid.
CPF3C3C E	Value for parameter &1 is not valid.

API introduced: V5R2

"Change System Collector Attributes (QYPSCSCA, QypsChgSysCollectorAttributes) API" on page 6 | "Performance Management APIs," on page 1 | APIs by category

---

Retrieve PEX Information (QYPERPEX, qypeRetrievePexInfo) API

Required Parameter Group:

1	Receiver variable	Output	Char(*)
2	Length of receiver variable	Input	Binary(4)
3	Format name	Input	Char(8)
4	PEX session name	Input	Char(10)
5	Error Code	I/O	Char(*)

Service Program Name: QYPESVPG
Default Public Authority: *USE
Threadsafe: Yes

The Retrieve PEX Information (QypeRetrievePexInformation) API retrieves information about a PEX session. It can be used to retrieve the following:

- Session name
- Collection type
- State
- State qualifier
- Event count

- Filtered event count
- Start complete time
- Resume time
- Storage used
- Qualified job name of the job that created the session
- Definition name
- Filter name
- Sampling interval

Authorities and Locks

None

Required Parameter Group

Receiver variable
OUTPUT; CHAR(*)

The variable that is to receive the information requested. The minimum size for this area is 8 bytes. You can specify the size of this area to be smaller than the format requested if you specify the length of receiver variable parameter correctly. As a result, the API returns only the data that the area can hold.

Length of receiver variable
INPUT; BINARY(4)

The length of the receiver variable. If this value is larger than the actual size of the receiver variable, the results may not be predictable. The minimum value is 8.

Format name
INPUT; CHAR(8)

The content and format of the information returned for the PEX session.

The possible format names are:

"PEXI0100 Format" on page 97	Basic information about the PEX session.
"PEXI0200 Format" on page 97	Detailed information about the PEX session.

PEX session name
INPUT; CHAR(10)

The name of a PEX session. Special values supported:

**ALL*	All PEX sessions.

Error code
I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

Header section

For detailed descriptions of the fields in this table, see Field Descriptions

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	BINARY(4)	Offset to first entry
12	C	BINARY(4)	Number of entries returned
16	10	BINARY(4)	Size of each entry
20	14	BINARY(4)	Number of entries available

PEXI0100 Format

The following information is returned for one entry with the PEXI0100 format. For detailed descriptions of the fields in the table, see Field Descriptions.

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	Session name
10	A	CHAR(2)	Reserved
12	C	BINARY(4)	Collection type
16	10	BINARY(4)	State
20	14	BINARY(4)	State qualifier
24	18	BINARY(4)	Event count
28	1C	BINARY(4)	Filtered event count
32	20	CHAR(8)	Start complete time
40	28	CHAR(8)	Resume time
48	30	BINARY(4)	Storage used
52	34	CHAR(10)	Job name of the job that created the session
62	3E	CHAR(10)	User name of the job that created the session
72	48	CHAR(6)	Job number of the job that created the session

PEXI0200 Format

The following information is returned for one entry with the PEXI0200 format. For detailed descriptions of the fields in the table, see Field Descriptions.

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	Session name
10	A	CHAR(2)	Reserved
12	C	BINARY(4)	Collection type
16	10	BINARY(4)	State

Offset		Type	Field
Dec	Hex		
20	14	BINARY(4)	State qualifier
24	18	BINARY(4)	Event count
28	1C	BINARY(4)	Filtered event count
32	20	CHAR(8)	Start complete time
40	28	CHAR(8)	Resume time
48	30	BINARY(4)	Storage used
52	34	CHAR(10)	Job name of the job that created the session
62	3E	CHAR(10)	User name of the job that created the session
72	48	CHAR(6)	Job number of the job that created the session
82	52	CHAR(10)	Definition name
92	5C	CHAR(10)	Filter name
102	5C	CHAR(2)	Reserved
104	5E	BINARY(4)	Sampling interval

Field Descriptions

Bytes available. Number of bytes available to be returned by the API. If this value is larger than the bytes returned it means that the API had more information to return than the receiver variable could hold.

Bytes returned. Number of bytes returned by the API and placed in the receiver variable. If this value is smaller than the bytes available, it means that the API had more information to return than the receiver variable could hold.

Collection type. Indicates type of data being collected in the PEX session.

Possible values are:

1 Trace.
2 Stats.
3 Stats hierarchical.
6 Profile.

Definition name. Name of definition used by the session.

Event count. Number of events in the session.

Filter name. Name of filter used by the session. This field will contain a value if a valid filter name was specified with the FTR keyword of the STRPEX command. If no filter name was specified in STRPEX, this field will be all blanks.

Filtered event count. Number of filtered events in the session. A filtered event is one that is not collected. It is filtered out based on specifications supplied in the Add Pex Filter (ADDPEXFTR) command.

Job name of the job that created the session. The name of the job that did the STRPEX command to start the session.

Job number of the job that created the session. The system-assigned number of the job that did the STRPEX command to start the session.

Number of entries available. Number of fixed-length entries that were available to be returned to the caller of the API, given sufficient space in the receiver variable. If this value is larger than the number of entries returned, it means that the API had more information to return than the receiver variable could hold.

Number of entries returned. The number of fixed-length entries following the header section. If this value is smaller than the number of entries available, it means that the API had more information to return than the receiver variable could hold.

Offset to first entry. The offset, in bytes, from the start of the header section to the beginning of the actual data returned by the API.

Reserved. An ignored field.

Resume time. Time (in 8 byte time of day format) that the session was resumed, if STRPEX OPTION(*RESUME) was used. If STRPEX (*RESUME) was not used, this field will be set to x'00' (nulls).

Sampling interval. Size of sampling interval in milliseconds. This field will contain a value if a sampling interval was specified with the INTERVAL keyword of the ADDPEXDFN command for the PEX definition used by this session.

Session name. The name of the PEX session.

Size of each entry. The size in bytes of each fixed-length entry following the header section.

Start complete time. Time (in 8 byte time of day format) that the session completed its start activity.

State. Specifies the internal state of the session.

Possible values are:

0 Created
1 Active
2 Suspended
3 Deleted
4 Stopped
5 Stopping (stop pending)
6 Starting

State qualifier. Specifies additional details about the state of the session.

Possible values are:

0 Normal(normal)
2 Wrapped (normal)
4096 Storage limit reached (limit)
4098 ASP storage limit reached (limit)
4099 User profile storage limit reached (limit)
4100 Byte stream file storage limit reached (limit)
8193 Internal error (error)
8194 Damage encountered (error)

Storage used. Number of bytes of storage used by the session.

User name of the job that created the session. The user name of the job that did the STRPEX command to start the session.

Error messages:

Message	Error Message Text
CPFAF04 E	Session &1 is not active.
CPF3C1E E	Required parameter &1 omitted.
CPF3C3C E	Value for parameter &1 not valid.
CPF3CF1 E	Error code parameter not valid.
CPF3CF2 E	Error(s) occurred during running of &1 API.

API introduced: V5R3

"Retrieve PEX Information (QYPERPEX, qypeRetrievePexInfo) API" on page 95 | "Performance Management APIs," on page 1 | APIs by category

IBM Performance Management eServer iSeries APIs

For information about IBM(R) Performance Management (PM) @server iSeries(TM), see PM eServer iSeries concepts.

The PM iSeries APIs can start PM iSeries, end PM iSeries, and retransmit PM iSeries data.

The PM iSeries APIs follow:

- "End PM eServer iSeries (Q1PENDPM) API" (Q1PENDPM) ends IBM Performance Management for eServer iSeries jobs. PM eServer iSeries jobs will not run again until the Start API (Q1PSTRPM) is issued or the product is configured using the CFGPM400 command.
- "Retransmit PM eServer iSeries Data (Q1PRTRN) API" on page 102 (Q1PRTRN) marks previously transmitted data as untransmitted data, thus allowing the data to be retransmitted.
- "Start PM eServer iSeries (Q1PSTRPM) API" on page 102 (Q1PSTRPM) configures IBM Performance Management for eServer iSeries to start sending performance data to IBM.

Top | "Performance Management APIs," on page 1 | APIs by category

End PM eServer iSeries (Q1PENDPM) API

Required Parameter Group:

1	End type	Input	Char(10)
2	Delay time	Input	Binary(4)
3	Error Code	I/O	Char(*)

Default Public Authority: *EXCLUDE
Threadsafe: No

The End PM @server iSeries(TM) (Q1PENDPM) API ends IBM Performance Management for eServer iSeries jobs. PM eServer iSeries jobs will not run again until the Start API (Q1PSTRPM) is issued or the product is configured using the CFGPM400 command.

Authorities and Locks

Public Authority
*EXCLUDE

Special authorities
You must have *JOBCTL special authority to use this API.

Required Parameter Group

End type
Input; Char(10)

The PM iSeries(TM) jobs to have controlled endings or the jobs that are to be ended immediately. The possible values are:

**CNTRLD*	The PM eServer iSeries jobs end in a controlled manner.
**IMMED*	The PM eServer iSeries jobs end immediately. The programs that are running do not get time to perform cleanup. This option may cause undesirable results if data has been partially updated. Therefore, this option should be used only if a controlled end was unsuccessful.

Delay time
Input; Binary(4)

The delay time when ″End type″ is *CNTRLD. This parameter is ignored if ″End type″ is *IMMED. The possible values are:

-1	The PM eServer iSeries jobs continue processing until the activity processing currently is complete.
Delay time (seconds)	The PM eServer iSeries jobs end immediately after the delay time.

Error code
I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter. If this parameter is omitted, diagnostic and escape messages are issued to the application.

Error Messages

Message ID	Error Message Text
CPFB03A E	Range of parameter &2 does not include &4.
CPFB03D E	PM eServer iSeries not active.
CPFB03E E	PM eServer iSeries already ending with *IMMED option.
CPFB03F E	PM eServer iSeries already ending with *CNTRLD option.
CPF24B4 E	Severe error addressing parameter list.
CPF3C1E E	Required parameter &1 omitted.
CPF3CF1 E	Error code parameter not valid.
CPF3CF2 E	Error(s) occurred during running of &1 API.

API introduced: V5R1

“Change System Collector Attributes (QYPSCSCA, QypsChgSysCollectorAttributes) API” on page 6 | “Performance Management APIs,” on page 1 | APIs by category

Retransmit PM eServer iSeries Data (Q1PRTRN) API

Required Parameter Group:

1	Start Date	Input	Char(7)

Default Public Authority: *EXCLUDE
Threadsafe: No

The Retransmit PM @server iSeries(TM) Data (Q1PRTRN) API marks previously transmitted data as untransmitted data. This allows the data to be retransmitted. This API should be used only with the assistance of IBM(R) service.

The date entered is in CYYMMDD format.

This API does not validate that the date entered is a valid date. For example, you could enter a date of 1000230 even though there are not 30 days in February. The API begins marking data for retransmission with the records that are greater than or equal to the passed date parameter. The API does some date validation; for example, 1000232 would not be accepted. If successful, you would expect to see message CPC0B01 Program Q1PRTRN completed successfully. If no message is received, display the job log for details.

Authorities and Locks

API Public Authority
*EXCLUDE

Special authorities
User must have *ALLOBJ authority to use this API.

Required Parameter Group

Start date
INPUT; Char(7)

Records with this date or greater are marked for retransmission. The date should be entered in CYYMMDD format.

Error Messages

Message ID	Error Message Text
CPF9802 E	Not authorized to object QA1PONE in QUSRSYS.
CPF0555 E	Date not in specified format or date not valid.
CPC0B01 C	Program Q1PRTRN completed successfully.

API introduced: V5R1

"Change System Collector Attributes (QYPSCSCA, QypsChgSysCollectorAttributes) API" on page 6 | "Performance Management APIs," on page 1 | APIs by category

Start PM eServer iSeries (Q1PSTRPM) API

Required Parameter Group:

1	Error code	I/O	Char(*)

Default Public Authority: *EXCLUDE
Threadsafe: No

The Start PM @server iSeries(TM) (Q1PSTRPM) API configures IBM(R) Performance Management for eServer iSeries to start sending performance data to IBM. This allows customers to receive periodic performance reports of their systems. See PM eServer iSeries for more information on this topic.

This API sets the following configurations:

- The server is configured to send performance data to IBM.
- The server is configured so that it does not receive performance data from remote systems to which it is attached.
- The Q1PLIN line description is created through the CRTLINSDLC command or reuses the line description if it already exists.
- The Q1PCTL controller description is created through the CRTCTLAPPC command or reuses the controller description if it already exists.
- The Q1PDEV device description is created through the CRTDEVAPPC command or reuses the device description if it already exists.
- The Q1PMOD mode description is created through the CRTMODD command or reuses the mode description if it already exists.
- Contact person information is taken from the "Work with Contact Information" (WRKCNTINF) interface.
- The PM eServer iSeries scheduler job, Q1PSCH, is submitted.

Authorities and Locks

Public authority
*EXCLUDE

Special authorities
You must have *JOBCTL special authority to use this API.

Required Parameter Group

Error code
I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter. If this parameter is omitted, diagnostic and escape messages are issued to the application.

Error Messages

Message ID	Error Message Text
CPFB03B E	PM eServer iSeries not started.
CPFB03C E	PM eServer iSeries already started.
CPF24B4 E	Severe error addressing parameter list.
CPF3C1E E	Required parameter &1 omitted.
CPF3CF1 E	Error code parameter not valid.
CPF3CF2 E	Error(s) occurred during running of &1 API.

API introduced: V5R1

"Change System Collector Attributes (QYPSCSCA, QypsChgSysCollectorAttributes) API" on page 6 | "Performance Management APIs," on page 1 | APIs by category

Exit Programs

These are the Exit Programs for this category.

Collection Services Data Collection Exit Program

Required Parameter Group:

1	Collection request parameters	I/O	Char(*)
2	Data buffer	Output	Char(*)
3	Work area	I/O	Char(*)
4	Return code	Output	Binary(4)

QSYSINC member name: QPMDCPRM

A Collection Services Data Collection exit program is called by Collection Services to collect performance data for a user-defined performance category. The Collection Services collector will store this data in the management collection object. For the exit program to be called by the collector, it must first be registered using the "Register Collector Data Category (QypsRegCollectorDataCategory) API" on page 11. User-defined performance data that is stored in the management collection object can be read using the "Read Management Collection Object Data (QpmReadMgtcolData) API" on page 28.

At the time a user-defined category is registered, a single entry point is specified for the data collection program. This entry point can be a program object (*PGM) or an entry point in a service program (*SRVPGM).

Collection Services calls this entry point with parameters which specify the type of request to perform and the control information necessary for the data collection program to complete the request.

Collection Services provides a data buffer where the data collection program will return collected data.

Collection Services also provides a work area which can be used by the data collection program to save its state between the calls.

The parameter structure passed to the data collection program is mapped by the QPMDCPRM header file shipped with the system in the QSYSINC library.

Authorities and Locks

None.

Required Parameter Group

Collection request parameters

I/O; CHAR(*)

This is a structure in which Collection Services passes control information about the data collection request to the data collection program. The data collection program uses this structure to return information about actions performed. The format of the data in this structure is determined by the parameter format specified when the user-defined category was registered using the Register Collector Data Category (QypsRegCollectorDataCategory) API. Currently, PMDC0100 is the only format supported. The layout of this structure is described in "Layout of Collection Request Structure" on page 105 below.

Data buffer

OUTPUT; CHAR(*)

This is a data buffer in which the data collection program returns data to be stored in the management collection object. Collection Services will store this data in a record of the repository of the management collection object.

For start collection and end collection requests, this is control data and will be stored in a collection control record. For interval collection request, this is collected performance data and will be stored in an interval record.

The length of this buffer is defined in the data buffer bytes available field of the collection request parameters structure. Writing to the data buffer beyond this length will have unpredictable results.

Work area

I/O; CHAR(*)

The work area is a storage area created by Collection Services for the sole use of the data collection program. Initially, the work area is set to zeroes. The same work area is passed in every call to the data collection program. This work area can be used by the data collection program to save some state information between the calls. Collection Services will preserve the work area contents between the calls. The length of the work area is set at category registration time and is defined in the length of work area field of the collection request.

Return code

OUTPUT; BINARY(4)

Return code set by the data collection program. Collection Services will take an action based on the code returned from the data collection program. The return code is ignored for the cleanup and terminate request. Possible return codes and Collection Services actions:

0 Data collection request was performed successfully.

> 0 Data collection request encountered recoverable errors. A non-zero return code will be logged in the job log. Action taken by Collection Services depends on the collection request:

- For start collection and end collection requests, Collection Services will assume that no data can be collected for this category and data collection for this category will be stopped.
- For interval collection request, Collection Services will assume that future requests may still be successful. No data will be stored for this request. However, Collection Services will continue collecting data for this category.

< 0 Data collection request encountered unrecoverable errors. Collection Services will assume that no further data collection is possible for this category and stop collecting data for this category. The data collection program may use a negative return code to cause immediate termination of data collection for the user-defined category.

Layout of Collection Request Structure

The layout of the collection request structure is determined by the parameter format specified when the user-defined category was registered using the Register Collector Data Category (QypsRegCollectorDataCategory) API.

Currently, PMDC0100 is the only format supported. The table below shows the layout of the collection request structure for the PMDC0100 format. For detailed descriptions of the fields in the table, see "Field Descriptions" on page 106 below.

Offset				
Dec	**Hex**	**Input/Output**	**Type**	**Field**
0	0	INPUT	CHAR(8)	Format name
8	8	INPUT	CHAR(10)	Category name
18	12	—	CHAR(2)	Reserved
20	14	INPUT	BINARY(4)	Request type
24	18	INPUT	BINARY(4)	Request type modifier
28	1C	INPUT	BINARY(4)	Data buffer bytes available

Offset		Input/Output	Type	Field
Dec	Hex			
32	20	INPUT	BINARY(4)	Offset to category parameter string
36	24	INPUT	BINARY(4)	Length of category parameter string
40	28	INPUT	BINARY(4)	Length of work area
44	2C	—	CHAR(4)	Reserved
48	30	INPUT	CHAR(8)	Interval key
56	38	INPUT	CHAR(8)	Interval time
64	40	OUTPUT	BINARY(4)	Data buffer bytes provided
68	44	OUTPUT	BINARY(4)	More data indicator
72	48	—	CHAR(8)	Reserved

Field Descriptions

Category name. Name of the user-defined category for which data collection is performed. This name was registered previously uisng the Register Collector Data Category (QypsRegCollectorDataCategory) API.

Data buffer bytes available. Size of the data buffer passed in the data buffer parameter.

Data buffer bytes provided. The data collection program indicates in this field how many bytes of data in the data buffer should be stored by Collection Services in the repository of the management collection object. If this field is set to 0 for start collection or end collection requests, a collection control record will not be created. If this field is set to 0 for an interval collection request, Collection Services will still create an interval record but will not store any data in it. This field is set to zero at entry to the data collection program.

Format name. Name of the collection request structure format passed to the data collection program. This is the parameter format specified when the user-defined category was registered using the Register Collector Data Category (QypsRegCollectorDataCategory) API. Currently, PMDC0100 is the only format supported.

Interval key. Record key of the repository record about to be written into the repository of the management collection object. Format of this field is DDHHMMSS, where:

DD Number of days from the beginning of collection to this collection object. Day numbering starts from 0.
HHMMSS Time in hours, minutes and seconds when a particular collection sample was scheduled.

Record keys of interval records, with the possible exception of the key of the first interval record in a collection period, are normalized to the collection interval boundary. For example, for a 15 minute collection interval, valid record keys will be 00124500 or 00223000, but not 00131014.

Interval time. Exact time in system timestamp format when the Collection Services collector initiated a particular request. This time is provided for reference only. Normally this time is close to the time implied by the interval key field but, unlike the interval key, this time is not normalized. See Convert Date and Time Format (QWCCVTDT) API for details about time formats.

Length of category parameter string. Length of the category parameter string. If the parameter string was not registered for this category, this field is set to 0.

Length of work area. Length of the work area passed in the work area parameter. The size of the work area is set during category registration.

More data indicator. By setting this field to a non-zero value, the data collection program may indicate to Collection Services that it has more data to store in the repository of the management collection object for this request. When this field is returned with a non-zero value, Collection Services will transfer the returned data to the current repository record then will call the data collection program again with all parameters identical to the previous call except that the request type modifier field will be set to 20. The data collection program may use the more data indicator field as many times as needed to transfer all collected data to Collection Services. This field is set to zero at entry to the data collection program.

Offset to category parameter string. Offset in bytes to the category parameter string starting from the beginning of the collection request. This parameter string was specified when the user-defined category was registered using the Register Collector Data Category (QypsRegCollectorDataCategory) API. The parameter string can be used to pass customization information to the data collection program. The length of this string is defined in the length of category parameter string field. If a parameter string was not registered for this category, this field is set to 0. The parameter string is only passed when the request type is set to 10 - start collection request.

Request type. Type of action requested from the data collection program. The data collection program must support the following request types:

10 *Start collection request.* This is the first request that the data collection program will receive in a data collection session. When receiving this request, the data collection program is expected to initialize whatever interfaces it uses to collect the data. Optionally, in the provided data buffer, the data collection program may return collection control information to be stored in a collection control record in the repository of the management collection object.

20 *End collection request.* This is the last request the data collection program will receive in a data collection session. When receiving this request, the data collection program is expected to close whatever interfaces it uses to collect the data, release resources, and so on. Optionally, in the provided data buffer, the data collection program may return collection control information to be stored in a collection control record in the repository of the management collection object.

30 *Interval collection request.* The data collection program will receive a request of this type each time the interval collection for this user-defined category is scheduled. The time between interval collection requests is specified at category registration time. When receiving this request, the data collection program is expected to perform its regular collection of performance data and return collected data in the provided data buffer. This data will be stored in an interval record in the repository of the management collection object.

40 *Cleanup and terminate (shutdown) request.* This request is sent to the data collection program when Collection Services cannot continue data collection. An example of such a problem is the loss of contact with the Collection Services collector job. When receiving this request, the data collection program is expected to perform necessary cleanup, release resources, and so on. The data collection program cannot return any data for this request. Any data placed in the data buffer will be ignored.

Request type modifier. This field modifies the meaning of the request type field. This field can have two values:

10 Normal request to collect data.

20 Continuation of the previous request. This value is used to indicate to the data collection program that this is the continuation of the same collection request. When the data collection program returns with the more data indicator set to a non-zero value, Collection Services will transfer the returned data to the current repository record then will call the data collection program again with the same parameters except that the request type modifier field will be set to 20. In this way, the data collection program may store more collected data in the management collection object than fits into the data buffer.

API introduced: V5R2

"Change System Collector Attributes (QYPSCSCA, QypsChgSysCollectorAttributes) API" on page 6 | "Performance Management APIs," on page 1 | APIs by category

Performance Monitor Exit Program

Required Parameter Group:

1	Member name	Input	Char(10)
2	Library name	Input	Char(10)

The Performance Monitor exit program is called to process the performance data just collected by the performance monitor. You would write an exit program for the performance monitor if you wanted to be sure that the performance data being collected was processed as soon as the monitor was done collecting it.

Note: The Performance Monitor exit program pertains to the Start Performance Monitor (STRPFRMON) command, not the Performance Monitor Collector APIs. Starting in Version 5 Release 1, the Start Performance Monitor (STRPFRMON) command is no longer supported.

Authorities and Locks

None.

Required Parameter Group

Member name
INPUT; CHAR(10)

The performance data member name.

Library name
INPUT; CHAR(10)

The library that contains the performance data.

Error Messages

The performance monitor handles any error that could occur in the exit program. Not only does the performance monitor contain generic MCH (machine), CPF (i5/OS), and PFR (performance) error messages, it also contains a function-check handler.

Exit program introduced: V2R3

Top | "Performance Management APIs," on page 1 | APIs by category

Appendix. Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

```
IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY  10504-1785
U.S.A.
```

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

```
IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan
```

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

© Copyright IBM Corp. 1998, 2006

IBM Corporation
Software Interoperability Coordinator, Department YBWA
3605 Highway 52 N
Rochester, MN 55901
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, IBM License Agreement for Machine Code, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

(C) IBM 2006. Portions of this code are derived from IBM Corp. Sample Programs. (C) Copyright IBM Corp. 1998, 2006. All rights reserved.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Programming Interface Information

This Application Programming Interfaces (API) publication documents intended Programming Interfaces that allow the customer to write programs to obtain the services of IBM i5/OS.

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:
Advanced 36
Advanced Function Printing
Advanced Peer-to-Peer Networking
AFP
AIX
AS/400
COBOL/400
CUA
DB2
DB2 Universal Database
Distributed Relational Database Architecture
Domino
DPI
DRDA
eServer
GDDM
IBM
Integrated Language Environment
Intelligent Printer Data Stream
IPDS
i5/OS
iSeries
Lotus Notes
MVS
Netfinity
Net.Data
NetView
Notes
OfficeVision
Operating System/2
Operating System/400
OS/2
OS/400
PartnerWorld
PowerPC
PrintManager
Print Services Facility
RISC System/6000
RPG/400
RS/6000
SAA
SecureWay
System/36
System/370
System/38
System/390
VisualAge
WebSphere
xSeries

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, and service names may be trademarks or service marks of others.

Terms and Conditions

Permissions for the use of these Publications is granted subject to the following terms and conditions.

Personal Use: You may reproduce these Publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative works of these Publications, or any portion thereof, without the express consent of IBM.

Commercial Use: You may reproduce, distribute and display these Publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these Publications, or reproduce, distribute or display these Publications or any portion thereof outside your enterprise, without the express consent of IBM.

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the Publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the Publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations. IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE

Printed in USA