IBM Systems - iSeries

Program and CL Command APIs

Version 5 Release 4

IBM Systems - iSeries

Program and CL Command 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 203.

Sixth Edition (February 2006)

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.

Contents

Program and CL Command APIs

The Program and CL Command APIs create programs, retrieve program information, list and retrieve module information, activate bound programs, resolve pointers to exports, and retrieve command information.

Before using the Create Program API, you should have some MI programming experience and understand the concepts in the iSeries Machine Interface instructions, which provides detailed descriptions of the iSeries(TM) machine interface instruction fields and the formats of those fields.

The Program and CL Command APIs are:

- "Activate Bound Program (QleActBndPgm) API" on page 3 (QleActBndPgm) activates the specified bound program or service program and all dependent service programs, and then initalizes the newly activated service programs.
- "Activate Bound Program Long (QleActBndPgmLong) API" on page 5 (QleActBndPgmLong) activates the specified bound program or service program and all dependent service programs, and then initalizes the newly activated service programs (64 bit version of QleActBndPgm).
- "Add Associated Space Entry (QbnAddAssociatedSpaceEntry) API" on page 7 (QbnAddAssociatedSpaceEntry) is used by a compiler preprocessor to put data into the associated space of the created module.
- "Add Bindtime Exit (QbnAddBindtimeExit) API" on page 9 (QbnAddBindtimeExit) is used by a compiler preprocessor to define an exit program that can be called when a created module is bound into an ILE program.
- "Add Extended Attribute Data (QbnAddExtendedAttributeData) API" on page 10 (QbnAddExtendedAttributeData) is used by a preprocessor to set the extended attribute field of a created module.
- "Add Preprocessor Level Data (QbnAddPreProcessorLevelData) API" on page 11 (QbnAddPreProcessorLevelData) is used to set the level of the preprocessor used to create a module.
- "Call Service Program Procedure (QZRUCLSP) API" on page 12 (QZRUCLSP) allows an unbound call to an ILE procedure exported by a service program.
- "Check Command Syntax (QCMDCHK) API" on page 16 (QCMDCHK) performs syntax checking for a single command, and optionally prompts for the command.
- "Create Program (QPRCRTPG) API" on page 18 (QPRCRTPG) converts the symbolic representation of a machine interface (MI) program into a program object.
- "End Preprocessor (QbnEndPreProcessor) API" on page 58 (QbnEndPreProcessor) must be called by every preprocessor after the output source file and preprocessor information is created. It records that a preprocessor was called.
- "Execute Command (QCMDEXC) API" on page 60 (QCMDEXC) runs a single CL command or can be used to run a command from within a high-level language or CL program.
- "Get Export (QleGetExp) API" on page 62 (QleGetExp) allows the caller to resolve a pointer to an export (either data or procedure) either by name or export number. The pointer is materialized for the specified activation.
- "Get Export Long (QleGetExpLong) API" on page 64 (QleGetExpLong) allows the caller to resolve a pointer to an export (either data or procedure) either by name or export number. The pointer is materialized for the specified activation (64 bit version of QleGetExp).
- "List ILE Program Information (QBNLPGMI) API" on page 66 (QBNLPGMI) gives information about Integrated Language Environment * (ILE *) programs, similar to the Display Program (DSPPGM) command.

- “List Module Information (QBNLMODI) API” on page 82 (QBNLMODI) API lists information about modules similar to the Display Module (DSPMOD) command. The information is placed in a user space specified by you.
- “List Service Program Information (QBNLSPGM) API” on page 89 (QBNLSPGM) gives information about service programs, similar to the Display Service Program (DSPSRVPGM) command.
- “Process Commands (QCAPCMD) API” on page 108 (QCAPCMD) performs command analyzer processing on command strings.
- “Replace Command Exit Program (QCARPLCM) API” on page 113 (QCARPLCM) may be used as the exit program for the QIBM_QCA_CHG_COMMAND for any command.
- “Retrieve Associated Space (QbnRetrieveAssociatedSpace) API” on page 114 (QbnRetrieveAssociatedSpace) retrieves data stored with the QbnAddAssociatedSpaceEntry API.
- “Retrieve Command Definition (QCDRCMDD) API” on page 116 (QCDRCMDD) retrieves information from a CL command (*CMD) object and generates XML (Extensible Markup Language) source statements that describe the command.
- “Retrieve Command Information (QCDRCMDI) API” on page 120 (QCDRCMDI) retrieves information from a command definition object.
- “Retrieve Module Information (QBNRMODI) API” on page 130 (QBNRMODI) retrieves module information and place it into a single variable in the calling program similar to the information returned using the Display Module (DSPMOD) command. The amount of information returned is limited to the size of the variable.
- “Retrieve Program Associated Space (QCLRPGAS) API” on page 149 (QCLRPGAS) retrieves information from the associated space of a user-state, user-domain program.
- “Retrieve Program Information (QCLRPGMI) API” on page 151 (QCLRPGMI) retrieves program information similar to the Display Program (DSPPGM) command.
- » “Retrieve Program Interface Information (QBNRPII) API” on page 172 (QBNRPII) retrieves program interface information from modules bound into programs or service programs. «
- “Retrieve Prompt Override (QPTRTVPO) API” on page 176 (QPTRTVPO) calls the prompt override program for a specified command and returns the prompt override command string from the prompt override program.
- “Retrieve Service Program Information (QBNRSPGM) API” on page 178 (QBNRSPGM) retrieves service program information similar to the information returned using the Display Service Program (DSPSRVPGM) command.
- “Scan for String Pattern (QCLSCAN) API” on page 188 (QCLSCAN) is used to scan a string of characters to see if the string contains a pattern.
- “Start Preprocessor (QbnStartPreProcessor) API” on page 191 (QbnStartPreProcessor) must be called first by every compiler preprocessor that has data to be processed during module creation.
- “Store Program Associated Space (QCLSPGAS) API” on page 191 (QCLSPGAS) stores information in the associated space of a user-state, user-domain program.

The Program and CL Command exit programs are:

- “Command Analyzer Change Exit Program” on page 193 (QIBM_QCA_CHG_COMMAND) is called when the command for which it is registered is processed.
- “Command Analyzer Retrieve Exit Program” on page 198 (QIBM_QCA_RTV_COMMAND) is called when the command for which it is registered is processed.

Top | APIs by category

APIs

These are the APIs for this category.

Activate Bound Program (QleActBndPgm) API

Required Parameter:

1	Pointer to bound program	Input	PTR(SYP)

Omissible Parameter Group:

2	Activation mark	Output	Binary(4)
3	Activation information	Output	Char(*)
4	Length of activation information	Input	Binary(4)
5	Error code	I/O	Char(*)

Returned Value:

	Activation mark	Output	Binary(4)

Default Public Authority: *USE
Service Program: QLEAWI
Threadsafe: Yes

The Activate Bound Program (QleActBndPgm) API activates the bound program or service program specified by the pointer to bound program parameter if it is not already active. All dependent service programs are activated, and all initialization of the newly activated service programs is done.

Initialization may consist of calling user procedures. All unhandled exceptions that occur during initialization will be percolated to the caller of QleActBndPgm; they will never be returned in the error code parameter. QleActBndPgm will never move the resume cursor for the exception. Therefore, it is possible for the caller of QleActBndPgm to handle an exception and resume initialization.

This API is identical to the "Activate Bound Program Long (QleActBndPgmLong) API" on page 5 (QleActBndPgmLong) API except that all mark values produced or consumed by the API are Binary(4) mark values. The QleActBndPgmLong API produces and consumes Binary(8) mark values. This API is normally used in conjunction with the Get Export (QleGetExp) API. First, you activate a service program using QleActBndPgm, and then later you use QleGetExp to retrieve pointers to procedures and data from that activation.

The QleActBndPgm API may not be called in a secondary thread of a Common Platform Architecture (CPA) application.

Authorities and Locks

None.

Required Parameter

Pointer to bound program
INPUT; PTR(SYP)

A system pointer to the program or service program to be activated.

Omissible Parameter Group

Activation mark
OUTPUT; BINARY(4)

The entity uniquely identifying the activation within the current job.

Activation information
OUTPUT; CHAR(*)

The structure that contains additional activation information. The format of this structure is shown in "Format of Activation Information." If activation information is specified, the length of activation information parameter must also be specified. The storage for activation information must be on a 16-byte boundary.

Length of activation information
INPUT; BINARY(4)

The length of the activation information. If the length is larger than the size of the storage for activation information, the results may not be predictable. The minimum length is 8 bytes.

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.

Returned Value

Activation mark
OUTPUT; BINARY(4)

This API returns the value for the activation mark parameter.

Format of Activation Information

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	CHAR(8)	Reserved
16	10	BINARY(4)	Activation group mark
20	14	BINARY(4)	Activation mark
24	18	CHAR(7)	Reserved
31	1F	CHAR(1)	Flags
32	20	CHAR(16)	Reserved

Field Descriptions

Activation group mark. An entity uniquely identifying the activation group within the current job.

Activation mark. An entity uniquely identifying the activation within the current job.

Flags	**Bit values**	
Bit 0	*0*	The program or service program was not already activated.
	1	The program or service program was already activated.
Bit 1-7	Reserved	

Bytes available. The length of all available information that could be returned. Bytes available can be greater than the length of activation information parameter. If it is greater, the information returned is truncated to the length specified.

Bytes returned. The length of all information returned. The value of the bytes returned field is always less than or equal to the length returned in the bytes available field.

Reserved. An area of reserved storage.

Error Messages

Message ID	Error Message Text
CPF24B4 E	Severe error while addressing parameter list.
CPF3C1E E	Required parameter &1 omitted.
CPF3C24 E	Length of the receiver variable is not valid.
CPF3C3A E	Value for parameter &2 for API &1 not valid.
CPF3C90 E	Literal value cannot be changed.
CPF3CF1 E	Error code parameter not valid.
CPF3CF2 E	Error(s) occurred during running of &1 API.
MCH3402 E	The argument being tested is not an address.
MCH4421 E	At least one field in the allocation strategy is not valid.
MCH4430 E	The exit priority value provided for &1 is not valid.

API introduced: V3R6

Top | "Program and CL Command APIs," on page 1 | APIs by category

---

Activate Bound Program Long (QleActBndPgmLong) API

Required Parameter:

1	Pointer to bound program	Input	PTR(SYP)

Omissible Parameter Group:

2	Activation mark	Output	Binary(8)
3	Activation information	Output	Char(*)
4	Length of activation information	Input	Binary(4)
5	Error code	I/O	Char(*)

Returned Value:

	Activation mark	Output	Binary(8)

Default Public Authority: *USE
Service Program: QLEAWI
Threadsafe: Yes

The Activate Bound Program Long (QleActBndPgmLong) API activates the bound program or service program specified by the pointer to bound program parameter if it is not already active. All dependent service programs are activated, and all initialization of the newly activated service programs is done.

Initialization may consist of calling user procedures. All unhandled exceptions that occur during initialization will be percolated to the caller of QleActBndPgmLong; they will never be returned in the error code parameter. QleActBndPgmLong will never move the resume cursor for the exception. Therefore, it is possible for the caller of QleActBndPgmLong to handle an exception and resume initialization.

This API is identical to the "Activate Bound Program (QleActBndPgm) API" on page 3 (QleActBndPgm) API except that all mark values produced or consumed by the API are Binary(8) mark values. The QleActBndPgm API produces and consumes Binary(4) mark values. The QleActBndPgmLong API is

normally used in conjunction with the Get Export Long (QleGetExpLong) API. First, you activate a service program using the QleActBndPgmLong API, and then later you use the QleGetExpLong API to retrieve pointers to procedures and data from that activation.

The QleActBndPgmLong API may not be called in a secondary thread of a Common Platform Architecture (CPA) application.

Authorities and Locks

None.

Required Parameter

Pointer to bound program
INPUT; PTR(SYP)

A system pointer to the program or service program to be activated.

Omissible Parameter Group

Activation mark
OUTPUT; BINARY(8)

The entity uniquely identifying the activation within the current job.

Activation information
OUTPUT; CHAR(*)

The structure that contains additional activation information. The format of this structure is shown in "Format of Activation Information." If activation information is specified, the length of activation information parameter must also be specified. The storage for activation information must be on a 16-byte boundary.

Length of activation information
INPUT; BINARY(4)

The length of the activation information. If the length is larger than the size of the storage for activation information, the results may not be predictable. The minimum length is 8 bytes.

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.

Returned Value

Activation mark
OUTPUT; BINARY(8)

This API returns the value for the activation mark parameter.

Format of Activation Information

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	CHAR(8)	Reserved

Offset			
Dec	Hex	Type	Field
16	10	BINARY(8)	Activation group mark
24	18	BINARY(8)	Activation mark
32	20	CHAR(7)	Reserved
39	27	CHAR(1)	Flags
40	28	CHAR(8)	Reserved

Field Descriptions

Activation group mark. An entity uniquely identifying the activation group within the current job.

Activation mark. An entity uniquely identifying the activation within the current job.

Flags	Bit values	
Bit 0		
	0	The program or service program was not already activated.
	1	The program or service program was already activated.
Bit 1-7	Reserved	

Bytes available. The length of all available information that could be returned. Bytes available can be greater than the length of activation information parameter. If it is greater, the information returned is truncated to the length specified.

Bytes returned. The length of all information returned. The value of the bytes returned field is always less than or equal to the length returned in the bytes available field.

Reserved. An area of reserved storage.

Error Messages

Message ID	Error Message Text
CPF24B4 E	Severe error while addressing parameter list.
CPF3C1E E	Required parameter &1 omitted.
CPF3C24 E	Length of the receiver variable is not valid.
CPF3C3A E	Value for parameter &2 for API &1 not valid.
CPF3C90 E	Literal value cannot be changed.
CPF3CF1 E	Error code parameter not valid.
CPF3CF2 E	Error(s) occurred during running of &1 API.
MCH3402 E	The argument being tested is not an address.
MCH4421 E	At least one field in the allocation strategy is not valid.
MCH4430 E	The exit priority value provided for &1 is not valid.

API introduced: V5R3

Top | "Program and CL Command APIs," on page 1 | APIs by category

Add Associated Space Entry (QbnAddAssociatedSpaceEntry) API

Required Parameter Group:

1	Associated space identifier	Input	Char(10)

2	Associated space entry data	Input	Char(*)
3	Length of associated space entry data	Input	Binary(4)
4	Options	Input	Char(1)
5	Error code	I/O	Char(*)

Default Public Authority: *USE
Service Program: QBNPREPR
Threadsafe: No

The Add Associated Space Entry (QbnAddAssociatedSpaceEntry) API may be used by a compiler preprocessor to put data in the associated space of the created module.

Authorities and Locks

None

Required Parameter Group

Associated space identifier
INPUT; CHAR(10)

The associated space identifier has the following special value:

**PREPROC* Identifies the type of data being stored in the created module. The special value must be left-justified and padded with blanks.

Associated space entry data
INPUT; CHAR(*)

The data to be placed into the associated space of the created module. The format of this data is specified by the user. This data will be copied into an ILE bound program or ILE service program and is available for use when the program is running. The QbnRetrieveAssociatedSpace API is used to retrieve this data from an ILE program or service program.

Length of associated space entry data
INPUT; BINARY(4)

The length of the data contained in the associated space entry data parameter.

Options
INPUT; CHAR(1)

You must specify one of the following special values. If more than one associated space entry is defined as extendable during preprocessing, the module will not be created. If DB2 UDB for iSeries SQL statements are contained in the input source file, option 2 cannot be specified.

1 The associated space entry will not be extendable.
2 The associated space entry can be extended. The associated space expands as data is stored in the associated space using QbnAddAssociatedSpaceEntry 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
CPF24B4 E	Severe error while addressing parameter list.

Message ID	Error Message Text
CPF3C90 E	Literal value cannot be changed.
CPF3CF1 E	Error code parameter not valid.
CPF5CA2 E	&1 is not a valid associated space identifier parameter.
CPF5CA3 E	Option &1 is not valid.
CPF5CA4 E	Error occurred while addressing API Parameter.
CPF5D22 E	Not able to locate internal data.
CPF5D24 E	Unexpected error occurred during preprocessor processing.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

API introduced: V3R1

Top | "Program and CL Command APIs," on page 1 | APIs by category

---

Add Bindtime Exit (QbnAddBindtimeExit) API

Required Parameter Group:

1	Qualified exit program name	Input	Char(20)
2	Exit program data	Input	Char(*)
3	Length of exit program data	Input	Binary(4)
4	Error code	I/O	Char(*)

Default Public Authority: *USE
Service Program: QBNPREPR
Threadsafe: No

The Add Bindtime Exit (QbnAddBindtimeExit) API may be used by a compiler preprocessor to define an exit program that is called when the created module is bound into a ILE program.

Authorities and Locks

None

Required Parameter Group

Qualified exit program name
INPUT; CHAR(20)

The qualified name of the exit program to be called when the created module is bound into an ILE program. The first 10 characters contain the program name, which is left-justified and padded with blanks. The second 10 characters contain the name of the library where the exit program is located and is left-justified and padded with blanks.

The library name can be specified with the following special value:

**LIBL* The library list.

The exit program is passed five parameters when called. The first two parameters are the exit program data and the exit program data length. The third parameter is a reserved CHAR(10). The fourth and fifth parameters are both reserved BINARY(4).

The exit program data being used in this API is defined by the user.

Exit program data
INPUT; CHAR(*)

This data is copied into the output source file member by the QbnEndPreProcessor API. When the exit program is called at ILE program creation time, a copy of the data is passed. The format

of this data is specified by the supplier of the exit program. This data is ignored if the length of data to be passed to the length of exit program data parameter value is 0.

Length of exit program data
INPUT; BINARY(4)

The length of the data contained in the exit program data parameter.

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.
CPF3C90 E	Literal value cannot be changed.
CPF3CF1 E	Error code parameter not valid.
CPF5CA4 E	Error occurred while addressing API Parameter.
CPF5CEA E	Library value &1 is not valid.
CPF5D22 E	Not able to locate internal data.
CPF5D24 E	Unexpected error occurred during preprocessor processing.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

API introduced: V3R1

Top | "Program and CL Command APIs," on page 1 | APIs by category

---

Add Extended Attribute Data (QbnAddExtendedAttributeData) API

Required Parameter Group:

1	Extended attribute data	Input	Char(20)
2	Error code	I/O	Char(*)

Default Public Authority: *USE
Service Program: QBNPREPR
Threadsafe: No

The Add Extended Attribute Data (QbnAddExtendedAttributeData) API may be used by a preprocessor to set the extended attribute field of the created module. The attribute field is part of the service related attributes of a module object. Because multiple preprocessors may be called in the path of module creation, and because each module only has a single attribute, only the initial preprocessor is allowed to set the extended attribute.

Authorities and Locks

None

Required Parameter Group

Extended attribute data
INPUT; CHAR(20)

The extended attribute data to be set in the created module. This data further describes the object, such as CLP for a CL program or PF for a file.

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.
CPF3C90 E	Literal value cannot be changed.
CPF3CF1 E	Error code parameter not valid.
CPF5CA4 E	Error occurred while addressing API Parameter.
CPF5D22 E	Not able to locate internal data.
CPF5D24 E	Unexpected error occurred during preprocessor processing.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

API introduced: V3R1

Top | "Program and CL Command APIs," on page 1 | APIs by category

---

Add Preprocessor Level Data (QbnAddPreProcessorLevelData) API

Required Parameter Group:

1	Preprocessor level data	Input	Char(6)
2	Error code	I/O	Char(*)

Service Program: QBNPREPR
Threadsafe: No

The Add Preprocessor Level Data (QbnAddPreProcessorLevelData) API may be used to set the level of the preprocessor used to create the module.

Authorities and Locks

None

Required Parameter Group

Preprocessor level data
INPUT; CHAR(6)

A description of the current preprocessor environment. The preprocessor-level data is specified with version, release, and modification information. This string must contain 6 characters of the form VxRyMz where x, y, and z are single digits. The valid values are between 0 and 9. The preprocessor-level data is an extension of the product number. This preprocessor-level data can then be used to show a different environment from that of the compiler. The preprocessor-level data must be less than or equal to the VxRyMz string of characters specified by the compiler, or module creation will fail.

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.
CPF3C90 E	Literal value cannot be changed.
CPF3CF1 E	Error code parameter not valid.
CPF5CA4 E	Error occurred while addressing API Parameter.
CPF5D22 E	Not able to locate internal data.
CPF5D24 E	Unexpected error occurred during preprocessor processing.
CPF5D29 E	Level data specified is not valid.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

API introduced: V3R1

Top | "Program and CL Command APIs," on page 1 | APIs by category

---

Call Service Program Procedure (QZRUCLSP) API

Required Parameter Group:

1	Qualified service program name	Input	Char(20)
2	Export Name	Input	Char(*)
3	Return Value Format	Input	Binary(4)
4	Parameter Formats	Input	Array(*) of Binary(4)
5	Number of Parameters	Input	Binary(4)
6	Error code	I/O	CHAR(*)

Optional Parameters:

7	Return Value	Output	Char(*)
8	Parameter 1	I/O	Char(*)
9	Parameter 2	I/O	Char(*)
10	Parameter 3	I/O	Char(*)
11	Parameter 4	I/O	Char(*)
12	Parameter 5	I/O	Char(*)
13	Parameter 6	I/O	Char(*)
14	Parameter 7	I/O	Char(*)

Default Public Authority: *USE
Threadsafe: No

The Call Service Program Procedure (QZRUCLSP) API allows an unbound call to an ILE procedure exported by a service program.

The name of the service program, and the name of the exported procedure are passed in as parameters. This API runs in the callers activation group and if the specified service program also specifies that it run in the caller's activation group, it too will run in the same group. All dependent service programs are activated, and all initialization of the newly activated service programs is done.

Since the QZRUCLSP API has no way of determining the parameters that the called procedure expects, the layout of those parameters must be described by the caller in a ″Parameter Format″ array.

All of the parameter values given to this API to be subsequently passed to the procedure are passed by reference.

This API does not support calling procedures that have been defined using ″#pragma argopt″.

Authorities and Locks

Service Program Authority
*EXECUTE

Service Program Library Authority
*EXECUTE

Service Program Lock
*SHRRD

Required Parameter Group

Qualified service program name
INPUT; CHAR(20)

The name of the service program for which the information is to be listed. The first 10 characters contain the service program name. The second 10 characters contain the name of the library where the service program is located.

The library name can be these special values:

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

Export name
INPUT; CHAR(*)

A null terminated string containing the name of the exported identifier. The name is matched exactly, without CCSID conversion or folding to uppercase.

Return Value Format
INPUT; BINARY(4)

The format of the returned data.

This value must be one of the following:

0 The procedure does not return a value. The "Return Value" parameter is ignored.
1 The procedure returns an integer. The "Return Value" parameter should address a location to receive a BINARY(4) value.
2 The procedure returns a pointer. The "Return Value" parameter should address a location to receive a 16 byte pointer.
3 The procedure returns an integer and the "errno" return value set by many program calls. The "Return Value" parameter should address a location to receive two BINARY(4) values. The first is the four byte return value, and the second is the four byte errno value.

Parameter Formats
INPUT; ARRAY(*) of BINARY(4)

The format of the parameters. This length of this array is specified in the "Number of parameters" value.

Each array entry should be one of the following:

1 The parameter is a BINARY(4) argument to be passed to the procedure by value.
2 The parameter is a pointer value.

Number of parameters
INPUT; BINARY(4)

The number of parameters that will be passed to the procedure. Up to seven parameters are supported.

Error code
I/O; CHAR(*)

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

Optional Parameters

Return value
OUTPUT; CHAR(*)

For procedures that return a value, this parameter points to the space to receive the data.

If this parameter is not passed, or is passed using a null pointer, no value is returned regardless of the value of the ″Return value format″ parameter.

Parameter 1
I/O; CHAR(*)

The first parameter passed to the procedure. If the corresponding entry in the parameter format array is a 1, this parameter should address a BINARY(4) value. If the corresponding entry in the parameter format array is a 2, this parameter should address the storage being referenced.

If the parameter format array indicates that a parameter should be passed to the exported procedure in this position, but Parameter 1 is not passed to the QZRUCLSP API, then zero or a null pointer, depending on the parameter format array entry, is passed to the procedure.

Parameter 2
I/O; CHAR(*)

The second parameter passed to the procedure. If the corresponding entry in the parameter format array is a 1, this parameter should address a BINARY(4) value. If the corresponding entry in the parameter format array is a 2, this parameter should address the storage being referenced.

If the parameter format array indicates that a parameter should be passed to the exported procedure in this position, but Parameter 2 is not passed to the QZRUCLSP API, then zero or a null pointer, depending on the parameter format array entry, is passed to the procedure.

Parameter 3
I/O; CHAR(*)

The third parameter passed to the procedure. If the corresponding entry in the parameter format array is a 1, this parameter should address a BINARY(4) value. If the corresponding entry in the parameter format array is a 2, this parameter should address the storage being referenced.

If the parameter format array indicates that a parameter should be passed to the exported procedure in this position, but Parameter 3 is not passed to the QZRUCLSP API, then zero or a null pointer, depending on the parameter format array entry, is passed to the procedure.

Parameter 4
I/O; CHAR(*)

The fourth parameter passed to the procedure. If the corresponding entry in the parameter format array is a 1, this parameter should address a BINARY(4) value. If the corresponding entry in the parameter format array is a 2, this parameter should address the storage being referenced.

If the parameter format array indicates that a parameter should be passed to the exported procedure in this position, but Parameter 4 is not passed to the QZRUCLSP API, then zero or a null pointer, depending on the parameter format array entry, is passed to the procedure.

Parameter 5
I/O; CHAR(*)

The fifth parameter passed to the procedure. If the corresponding entry in the parameter format array is a 1, this parameter should address a BINARY(4) value. If the corresponding entry in the parameter format array is a 2, this parameter should address the storage being referenced.

If the parameter format array indicates that a parameter should be passed to the exported procedure in this position, but Parameter 5 is not passed to the QZRUCLSP API, then zero or a null pointer, depending on the parameter format array entry, is passed to the procedure.

Parameter 6

I/O; CHAR(*)

The sixth parameter passed to the procedure. If the corresponding entry in the parameter format array is a 1, this parameter should address a BINARY(4) value. If the corresponding entry in the parameter format array is a 2, this parameter should address the storage being referenced.

If the parameter format array indicates that a parameter should be passed to the exported procedure in this position, but Parameter 6 is not passed to the QZRUCLSP API, then zero or a null pointer, depending on the parameter format array entry, is passed to the procedure.

Parameter 7

I/O; CHAR(*)

The seventh parameter passed to the procedure. If the corresponding entry in the parameter format array is a 1, this parameter should address a BINARY(4) value. If the corresponding entry in the parameter format array is a 2, this parameter should address the storage being referenced.

If the parameter format array indicates that a parameter should be passed to the exported procedure in this position, but Parameter 7 is not passed to the QZRUCLSP API, then zero or a null pointer, depending on the parameter format array entry, is passed to the procedure.

Usage Notes

Since this API is implemented as a program, it adds an additional control boundary between the caller and the service program procedure.

Any exceptions generated by the service program procedure are either returned in the error code structure, if provided, or resignalled to the caller.

Error Messages

Message ID	Error Message Text
CPF24B4 E	Severe error while addressing parameter list.
CPF3C1D E	Length specified in parameter &1 not valid.
CPF3C1E E	Required parameter &1 omitted.
CPF3C24 E	Length of the receiver variable is not valid.
CPF3C3A E	Value for parameter &2 for API &1 not valid.
CPF3C90 E	Literal value cannot be changed.
CPF3CF1 E	Error code parameter not valid.
CPF3CF2 E	Error(s) occurred during running of &1 API.
MCH3402 E	The argument being tested is not an address.
MCH4421 E	At least one field in the allocation strategy is not valid.
MCH4422 E	&1 cannot be called in the default activation group.
MCH4430 E	The exit priority value provided for &1 is not valid.

Example

The following is an example of a program calling the **Qp0lGetAttr()** API. That API takes the following parameters:

```
int Qp0lGetAttr
  (Qlg_Path_Name_T         *Path_Name,
   Qp0l_AttrTypes_List_t   *Attr_Array_ptr,
   char                    *Buffer_ptr,
   uint                     Buffer_Size_Provided,
   uint                    *Buffer_Size_Needed_ptr,
   uint                    *Num_Bytes_Returned_ptr,
   uint                     Follow_Symlnk, ...);
```

The procedure returns an integer, and its parameters are pointer, pointer, pointer, integer, pointer, pointer, integer. The parameter format array for calling this procedure is 2, 2, 2, 1, 2, 2, 1.

```
#include <QZRUCLSP.H>

int main(int argc, char **argv)
{
   int rc;          /* return code               */
   struct {
            Qlg_Path_Name_T lg;
            char *path;
          } lname;  /* the path name parameter   */

   int attrreq[2];  /* the attributes requested  */
   char buffer[32]; /* returned information      */
   int needed;      /* bytes needed              */
   int returned;    /* bytes returned            */
   int parm_format[7] = {2, 2, 2, 1, 2, 2, 1};



   ...

   QZRUCLSP("QP0LLIB2  QSYS      ", /* SRVPGM         */
            "Qp0lGetAttr",          /* Procedure      */
            1,                      /* Return integer */
            parm_format,            /* parm formats   */
            7,                      /* Seven parms    */
            NULL,                   /* error code     */
            &rc,                    /* return value   */
            &lpath,                 /* pointer        */
            attrreq,                /* pointer        */
            buffer,                 /* pointer        */
            sizeof(buffer),         /* integer        */
            &needed,                /* pointer        */
            &returned,              /* pointer        */
            0);                     /* integer        */

   ...
}
```

API introduced: V4R4

Top | "Program and CL Command APIs," on page 1 | APIs by category

Check Command Syntax (QCMDCHK) API

Required Parameter Group:

1	Command string	I/O	Char(*)
2	Length of command string	Input	Packed(15,5)

Optional Parameter:

3	IGC process control	Input	Char(3)

Default Public Authority: *USE
Threadsafe: Yes.
See “Usage Notes” on page 18 for command considerations.

The Check Command Syntax (QCMDCHK) API performs syntax checking for a single command, and optionally prompts for the command. The command is not run. If prompting is requested, the command string is returned to the calling program with the updated values as entered through prompting. The QCMDCHK API can be called from an HLL program.

Typical uses of QCMDCHK are:

- Prompt the user for a command and then store the command for later processing.
- Determine the parameter values specified by the user.
- Log the processed command. First, prompt with the QCMDCHK API, run with the Execute Command (QCMDEXC) API, and then log the processed command.

Notes:

1. Command strings in System/38 syntax can use the QCACHECK API. The QCACHECK API accepts the same parameters as QCMDEXC and QCMDCHK.
2. The Process Commands (QCAPCMD) API also provides similar functions. »
3. If the command to be checked is a proxy command, the QCMDCHK API will resolve to the target command. If the target command is also a proxy, the process repeats until either a non-proxy command is found, or the proxy chain becomes greater than the allowed maximum. Once a non-proxy command is found, the resolved command will replace the proxy command in the command string to be checked. The returned command string will contain the original proxy command name.
4. Proxy commands will be resolved before the command exit point QIBM_QCA_CHG_COMMAND is called. «

Authorities and Locks

Any Command
*USE

Required Parameter Group

Command string
I/O;CHAR(*)

The command you want to check is entered as a character string. If the command contains blanks, it must be enclosed in apostrophes. The maximum length of the character string is 32,702 characters; delimiters (the apostrophes enclosing the string) are not counted as part of the string.

Length of command string
INPUT;PACKED(15,5)

The length of the command string being passed. If the command string is passed as a quoted string, the command length is exactly the length of the quoted string. If the command string is passed in a variable, the command length is the length of the variable.

Optional Parameter Group

IGC process control
INPUT;CHAR(*)

The IGC process control instructs the system to accept double-byte data. The only value supported is IGC. IGC must be entered using all uppercase letters.

Usage Notes

While this API is threadsafe, it should not be used to run a command that is not threadsafe in a job that has multiple threads. Use the Display Command (DSPCMD) command to determine whether a command is threadsafe.

Error Messages

Message ID	Error Message Text
CPF0005 E	Returned command string exceeds variable provided length.
CPF0006 E	Errors occurred in command.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.
xxxnnnn E	Any escape message issued by any command may be returned. The messages listed previously are those issued by this API. Once the API has called the command analyzer, any message issued as an escape message may appear.

API in existance prior to V1R3

Top | "Program and CL Command APIs," on page 1 | APIs by category

---

Create Program (QPRCRTPG) API

Required Parameter Group:

1	Intermediate representation of the program	Input	Char(*)
2	Length of intermediate representation of program	Input	Binary(4)
3	Qualified program name	Input	Char(20)
4	Program text	Input	Char(50)
5	Qualified source file name	Input	Char(20)
6	Source file member information	Input	Char(10)
7	Source file last changed date and time information	Input	Char(13)
8	Qualified printer file name	Input	Char(20)
9	Starting page number	Input	Binary(4)
10	Public authority	Input	Char(10)
11	Option template	Input	Char(*)
12	Number of option template entries	Input	Binary(4)

Optional Parameter:

13	Error code	I/O	Char(*)

Default Public Authority: *USE
Threadsafe: No

The Create Program (QPRCRTPG) API converts the symbolic representation of a machine interface (MI) program into an OPM program object. This symbolic representation is known as the intermediate representation of a program.

The QPRCRTPG API creates a program object that resides in the *USER domain and runs in the *USER state. If you want the program object to be temporary, you must do one of the following:

- Delete the object when you no longer need it.
- Create the object in the QTEMP library, and let the system delete the object automatically when the job ends.

You can specify program objects created with the QPRCRTPG API in CL commands that process objects of type *PGM. For example, you can:

- Save and restore program objects using the Save Object (SAVOBJ) and Restore Object (RSTOBJ) commands.
- Delete program objects using the Delete Program (DLTPGM) command.
- Run program objects using the Call (CALL) command.
- Rename program objects using the Rename Object (RNMOBJ) command.
- Move program objects to a different library using the Move Object (MOVOBJ) command.

Note: MI instructions that reference system-domain or write-protected objects fail at security levels 40 and 50. At those levels, you must use APIs to work with the objects.

Authorities and Locks

Program Authority
*ALL. Required only if the program already exists and the option value *REPLACE is specified.

Program Library Authority
*CHANGE

Printer File Authority
*USE

Printer File Library Authority
*USE

Source File Authority
*USE

Source File Library Authority
*USE

Required Parameter Group

Intermediate representation of the program
INPUT; CHAR(*)

A string containing the intermediate representation of the program to be processed by the QPRCRTPG API. See “Program Syntax” on page 27.

Length of intermediate representation of program
INPUT; BINARY(4)

The size, in bytes, of the intermediate representation of the program.

Qualified program name
INPUT; CHAR(20)

The name and library of the program to be created or replaced. The first 10 characters contain the program name, and the second 10 characters contain the name of the library where the program is located. The special value *CURLIB may be used for the library name.

Program text
INPUT; CHAR(50)

Text that briefly describes the program.

Qualified source file name
INPUT; CHAR(20)

The name and library containing the source program. The first 10 characters contain the source file name, and the second 10 characters contain the name of the library where the file is located.

This places the value in the program object's service description. The special value *NONE may be used for the source file name. If you specify *NONE, no source file information is placed in the program object's service description. A special value, such as *LIBL, is not valid for the source file library.

Source file member information
INPUT; CHAR(10)

The file member containing the source program. This places the value in the program object's service description.

This value must be blanks if you specify *NONE as the **source file name**.

Source file last changed date and time information
INPUT; CHAR(13)

The date and time the member of the source file was last updated. The format of this field is in the CYYMMDDHHMMSS format, where:

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

This places the value in the program object's service description.

This value must be blank if you specify *NONE for the source file name parameter.

Qualified printer file name
INPUT; CHAR(20)

The name and library containing the printer file used to generate listings. The first 10 characters contain the printer file name, and the second 10 characters contain the name of the library where the file is located. The only special values supported for the library name are *LIBL and *CURLIB.

This value is ignored if you specify *NOLIST for the generate listing option (see "Values for the Option Template Parameter" on page 21).

Starting page number
INPUT; BINARY(4)

The first page number to be used on listings. This value should be between 1 and 9999; otherwise, the API uses 1.

This value is ignored if you specify *NOLIST for the generate listing option (see "Values for the Option Template Parameter" on page 21).

Public authority
INPUT; CHAR(10)

The authority you give the users who do not have specific private authorities to the object, and where the user's group has no specific authority to the object.

The values allowed are:

**CHANGE*
**ALL*
**USE*
**EXCLUDE*
The name of an authorization list

Option template
INPUT; CHAR(*)

This is an array of options. You can specify between 0 and 17 values. Each entry contains a CHAR(11) value as described in "Values for the Option Template Parameter."

Number of option template entries
INPUT; BINARY(4)

The number of option template entries.

The value must be between 0 and 17.

Optional Parameter

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.

Values for the Option Template Parameter

When you are using the QPRCRTPG API, you can specify a value in the option template. Only one value per option should be specified. If you specify more than one, the system only uses the first one. If you specify no value for a given option, the system uses the default value (underlined).

Create program object
Creates a program object.

The values allowed are:

**GEN*	Generates a program and places the program in the appropriate library.
**NOGEN*	No program is generated. The syntax of the intermediate representation of the program is checked, and if the generate listing option is *LIST, a listing is produced.

Replace program
Replaces the existing program if a program by the same name already exists in the specified library.

The values allowed are:

**NOREPLACE*	Does not replace an existing program by the same name in the specified library.
**REPLACE*	Replaces the existing program by moving it to the QRPLOBJ library.

Generate listing
Generates an output listing.

The values allowed are:

**NOLIST*	Does not generate a listing.
**LIST*	Generates a listing.

You must specify the following parameters:

- Printer file name and library
- Starting page number

Create cross-reference listing
Whether the listing is to contain a cross-reference list of variable and data item references.

The values allowed are:

**NOXREF*	Does not create cross-reference listing.
**XREF*	Creates a cross-reference listing of references to variables, labels, or both.

Create summary listing

Whether the listing is to contain a list of program attributes.

The values allowed are:

**NOATR*	Does not create a summary listing section.
**ATR*	Creates a summary listing section.

User profile

The values allowed are:

**USER*	The user profile of the user running the program is used as a source of authority when this program runs.
**ADOPT*	When the program runs, the object authority of both the program's owner and user are used.
**OWNER*	The system uses the user profile of the owner of the program as a source of authority when this program runs. Programs called by this program adopt this authority.

Use adopted authority

Whether the system uses the program-adopted authority from the calling programs as a source of authority when this program is running. The user must be authorized to create programs with adopted authority for the *ADPAUT option to take effect.

The values allowed are:

**ADPAUT*	The system uses program-adopted authority from the calling program.
**NOADPAUT*	The system does not use program-adopted authority from the calling program.

Note: Authorization to create programs which can adopt authority is controlled by the QUSEADPAUT system value. For more information, refer to the description of this system value in the Work Management topic.

Constrain arrays

The values allowed are:

**SUBSCR*	Constrains arrays. This requests additional run-time checks to ensure that references to array elements are not outside the bounds of the declare statement. This option causes the resulting program to run slower.
**NOSUBSCR*	Does not constrain arrays. The results of references to array elements outside the bounds of the declare statement are not defined.
**UNCON*	Allows fully unconstrained arrays. This ensures that references to array elements outside the bounds of the declare statement act as if the array element actually exists.

Note: This program attribute may be changed at run-time using the Override Program Attributes (OVRPGATR) MI instruction.

Constrain strings

The values allowed are:

**SUBSTR*	Constrains strings. This requests additional run-time checks to ensure that references to character strings are not outside the bounds of the declare statement. This option causes the resulting program to run slower.
**NOSUBSTR*	Does not constrain strings. The results of substring references outside the bounds of the declare statement are not defined.

Note: You can change this program attribute at run-time using the Override Program Attributes (OVRPGATR) MI instruction.

Initialize static storage

Static storage is allocated the first time a program is called. It remains allocated until explicitly deallocated.

The values allowed are:

**CLRPSSA* Initializes static storage. This code clears the program static storage area (PSSA) on entry using the Call External (CALLX) MI instruction.

**NOCLRPSSA* Does not initialize the PSSA.

Initialize automatic storage

Automatic storage is allocated each time a program runs and automatically deallocated when no longer needed.

The values allowed are:

**CLRPASA* Initializes automatic storage. This code clears the program automatic storage area (PASA) on entry using the Call External (CALLX) MI instruction.

**NOCLRPASA* Does not initialize the PASA.

Ignore decimal data errors

Whether errors found in decimal data result in exceptions.

The values allowed are:

**NOIGNDEC* Does not ignore decimal data errors.

When you specify *NOIGNDEC, decimal values used in numeric operations are checked for valid decimal digits and sign codes. If the system finds an error, it signals an exception.

**IGNDEC* Ignores data decimal errors.

When you specify *IGNDEC, decimal values used in numeric operations ensure they contain valid decimal digit and sign codes. However, the system treats digits that are not valid as zeros and signs that are not valid as positive signs. There is no exception signaled.

This option applies to only a subset of the numeric operations you specify.

Note: In all cases, the system signals decimal data errors if you use data pointers to address any of the instruction's operands.

The following list contains the MI instructions this option affects:

MI Instruction	Packed Source Operands Supported	Zoned Source Operands Supported	Notes
ADDN		X	
CMPNV		X	
CVTCN	X	X	You must specify operand 3 (the numeric view to be used for operand 2) as a constant and no data-pointer-defined operands.
CVTDFFP		X	

MI Instruction	Packed Source Operands Supported	Zoned Source Operands Supported	Notes
CVTNC	X	X	You must specify operand 3 (the numeric view to be used for operand 1) as a constant and no data-pointer-defined operands.
CPYNV	X	X	You must specify no data-pointer-defined operands.
DIV		X	
DIVREM		X	
EDIT		X	You must specify no data-pointer-defined operands.
EXTRMAG		X	
MULT		X	
NEG		X	
REM		X	
SCALE		X	
SUBN		X	

When you specify *IGNDEC, the system may still signal the decimal data exception. That is, other MI instructions and instruction combinations not listed above may signal the decimal data exception when the system finds decimal data that is bad.

Ignore binary data size errors

The values allowed are:

**NOIGNBIN* The system handles binary data size errors normally. When a binary size error occurs, an exception is signaled and the receiver contains the left-truncated result.

**IGNBIN* The system ignores binary data size errors. This is used when an overflow or underflow occurs on a computation and when a control MI instruction has a receiver that is a binary field. The receiver contains the left-truncated result.

Support coincident operands

The system overlaps coincident operands between the source and receiver operands in one or more program instructions. **Coincident operands** are operands that overlap physically, in storage.

The values allowed are:

**NOOVERLAP* Does not support coincident operands. If you specify *NOOVERLAP, you guarantee that coincident operand overlap will not occur while running the instruction. Therefore, the system can use the receiver on an instruction as a work area during operations performed to produce the final result. Using the receiver as a work area does not use as much processor resource as would be required to move the final result from an internal work area to the receiver.

**OVERLAP* Supports coincident operands. If you specify *OVERLAP, the operands on an MI instruction may overlap. Therefore, the system cannot use the receiver on an instruction as a work area during operations that produce the final result. This can require more processor resource for running the instruction but it ensures valid results if an overlap occurs.

The following is a list of instructions this option affects:

- Add logical character (ADDLC)

- Add numeric (ADDN)
- And (AND)
- Compute math function using one input value (CMF1)
- Concatenate (CAT)
- Convert character to numeric (CVTCN)
- Convert decimal form to floating-point (CVTDFFP)
- Convert external form to numeric value (CVTEFN)
- Convert floating-point to decimal form (CVTFPDF)
- Convert numeric to character (CVTNC)
- Copy bytes left adjusted with pad (CPYBLAP)
- Copy bytes right adjusted with pad (CPYBRAP)
- Divide (DIV)
- Divide with remainder (DIVREM)
- Exclusive or (XOR)
- Multiply (MULT)
- Or (OR)
- Remainder (REM)
- Scale (SCALE)
- Subtract logical character (SUBLC)
- Subtract numeric (SUBN)
- Trim length (TRIML)

Allow duplicate declares

The values allowed are:

**NODUP* This does not allow a program object to be declared more than once. This requests that duplicate declare (DCL) statements be diagnosed as errors.

**DUP* This allows a program object to be declared more than once. This requests that program objects declared more than once be pooled and not be diagnosed as errors.

Optimize

The values allowed are:

**OPT* This optimizes the program. In most instances, this produces the smallest and best running program. Occasionally, the source program may signal a MCH2802 escape message during processing. If this occurs, you should not optimize the program.

**NOOPT* This does not optimize the program. This requests the normal level code optimization when you create the program.

Target release

The release of the operating system on which you intend to use the object being created. You can use the object on a system with the specified release or with any subsequent release of the operating system installed.

The values allowed are:

**CURRENT* The object is to be used on the release of the operating system currently running on your system.

**PRV* The object is to be used on the previous release with modification level 0 of the operating system.

target-release Specify the release in the format VxRxMx.

Error Messages

Message ID	Error Message Text
CPD0078 D	Value &3 for parameter not a valid name.
CPF2143 E	Cannot allocate object &1 in &2 type *&3.
CPF2144 E	Not authorized to &1 in &2 type *&3.
CPF2146 E	Owner of object &1 and object being replaced not the same.
CPF2283 E	Authorization list &1 does not exist.
CPF223B D	&1 in &2 type *&3 adopted authority from previous call levels was set to *NO.
CPF223E E	Authority check for use adopted authority attribute failed.
CPF3CF1 E	Error code parameter not valid.
CPF3C35 E	Value &3 for parameter &2 not a valid name.
CPF3C5A E	Number of option template entries is not valid.
CPF3C5B E	Option template entry is not valid.
CPF3C5C E	Source file name and library is not valid.
CPF3C5D E	Source file member is not valid.
CPF3C5F E	Internal Representation of Program (IRP) string length parameter is not valid.
CPF3C50 E	Program &1 not created.
CPD0078 D	Value &3 for parameter not a valid name.
CPD3C50 D	Value &1 for the IRG string length parameter was not valid.
CPD3C52 D	Number of option template entries is not valid.
CPD3C53 D	Option template entry is not valid.
CPD3C54 D	Source file name and library is not valid.
CPD3C55 D	Source file member is not valid.
CPD3C56 D	Source file last changed date and time is not valid.
CPF3C56 E	Source file last changed date and time is not valid.
CPF3C60 E	Program name and library is not valid.
CPF3C61 E	Authority is not valid.
CPF3C62 E	Source file library specified.
CPF3C63 E	Source file member specified.
CPF3C64 E	Source file last changed date and time specified.
CPF3C90 E	Literal value cannot be changed.
CPF6301 E	Intermediate representation of program (IRP) contains &1 errors. Probable compiler error.
CPF6303 E	Message &1, &2 received while running create program command.
CPF6304 E	Library &1 not found.
CPF6306 E	Program &1 in library &2 already exists.
CPF6307 E	Program template value at offset &1, bit &2, length &3 not valid.
CPF6308 E	Not authorized to create program.
CPF6309 E	Not authorized to library &1.
CPF6455 E	Member &2 file &1 in library &3 not found.
CPF6457 E	Cannot allocate library &1 for program insertion.
CPF6551 E	Work space &2 cannot be extended. Probable compiler error.
CPF6552 E	Space &2 type &3 subtype &4 not PRM workspace.
CPF6553 E	PRM permanent table resolution failed. Probable compiler error.
CPF6554 E	Type of IST object &4 at offset &3 not valid. Probable PRM error.
CPF6555 E	Addressability field type not valid for IST number &4 at offset &3. Probable PRM error.
CPF6557 E	Error condition for IST &4 at &3 of IST space not valid. Probable PRM error.
CPF6560 E	Operation code &5 in MI instruction &3 at offset &6 not found in QPRODT.
CPF6561 E	Operand &4 in &3 at offset &5 in program template not valid.
CPF6563 E	Program was too large to be created.
CPF6564 E	Machine storage limit violation.
CPF6565 E	User profile storage limit exceeded.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

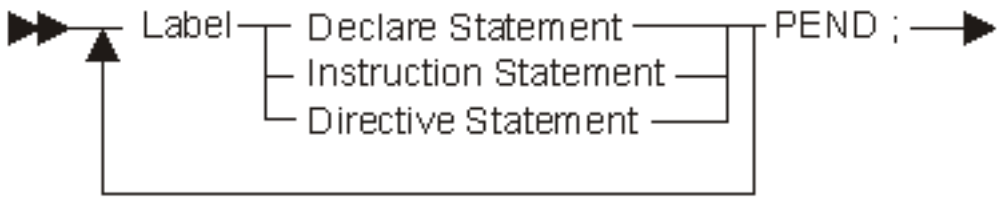
Program Attributes

The QPRCRTPG API creates programs that have the following attributes:

- An associated space of 480 bytes initialized to hexadecimal 00. You can use the QCLSPGAS API to store information in the program's associated space.
- Observability.
- A blank extended attribute.

Program Syntax

A program object consists of an **instruction stream** and an **object definition table (ODT)**. The intermediate representation of a program defines both of these components. It consists of one or more statements:



Instruction statements define MI instructions placed in the instruction stream. Declare statements define program objects placed in the ODT. Directive statements:

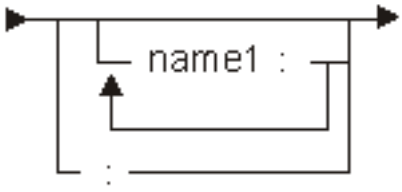
- Control the formatting of the output listing.
- Define entry-point program objects.
- Define symbolic breakpoints.
- Specify the end of the program.

The following sections explain how to define these statements.

Note: In the diagrams below, names that begin with an uppercase letter identify values specified in another diagram. Names that begin with a lowercase letter identify values defined in the table below the diagram.

Label

The following diagram and table show the possible labels:



Each name specified in the label generates a branch-point program object corresponding to the next MI instruction.

Constant	Range	Description
name1	Any	Label name for next instruction

Declare Statement

Declare statements define program data objects. All the declare statements in a program build the object definition table (ODT).

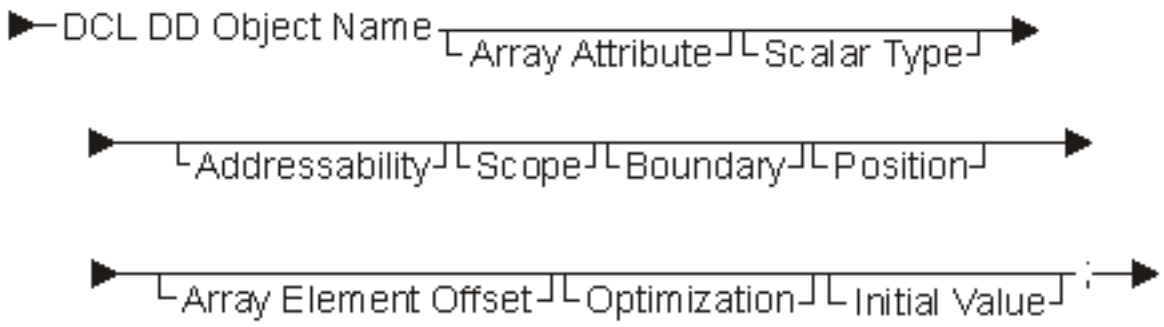
You cannot specifically declare branch and entry-point program objects. However, you can declare branch-point program objects using labels. You can also declare entry-point program objects using the entry directive statement.

The types of declare statements follow:

- Scalar Data Object Declare Statement
- Pointer Data Object Declare Statement
- Space Pointer Machine Object Declare Statement
- Operand List Declare Statement
- Instruction Definition List Declare Statement
- Exception Description Declare Statement
- Space Object Declare Statement
- Constant Object Declare Statement

Scalar-Data-Object Declare Statement

The following diagram and table show the scalar-data-object declare statement:



Only certain combinations of attributes are allowed based on the data object's addressability. The table below shows these combinations.

Address-ability	Array Attribute	Array Element Offset	Position	Boundary	Initial Value
STAT	X		X		X
	X			X	X
AUTO	X		X		X
	X			X	X
DEF	X		X	X	X
	X	X	X	X	
DIR	X		X	X	X
	X	X	X	X	
BAS	X		X	X	

Address-ability	Array Attribute	Array Element Offset	Position	Boundary	Initial Value
BASPCO	X		X	X	
PARM	X			X	

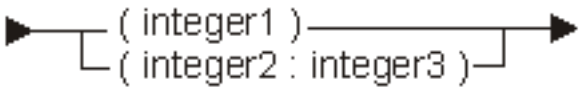
Object Name

The following diagram and table show the possible object names:

Constant	Range	Description
name1	Any	Program object name to be declared

Array Attribute

The following diagram and table show the possible array attributes:



Constant	Range	Description
integer1	1 to 16 776 191	Dimension of the data object with an implied lower bound of 1.
integer2	-2 147 483 648 to 2 147 483 647	Lower bound of the array.
integer3	integer2 to 2 147 483 647	Upper bound of the array. The dimension (integer3 - integer2) cannot exceed 16 776 191.

Example

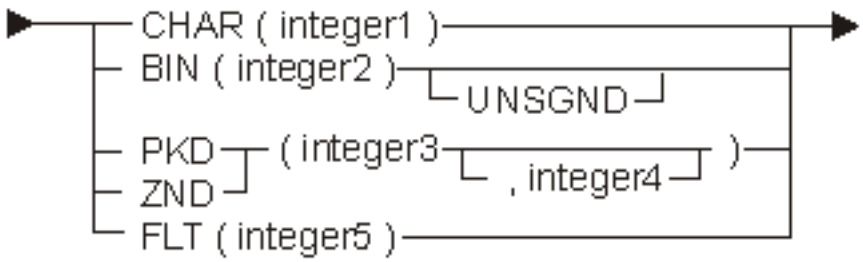
See Code disclaimer information for information pertaining to code examples.

The following declare statements each define an array of 50 elements. The elements of ARRAY1 are numbered 1 to 50. The elements of ARRAY2 are numbered 0 to 49. Each element of the array is a BIN(2) field. The addressability of the arrays is static.

```
DCL DD ARRAY1(50) BIN(2);
DCL DD ARRAY2(0:49) BIN(2);
```

Scalar Type

The following diagram and tables show the possible data types of scalar items:



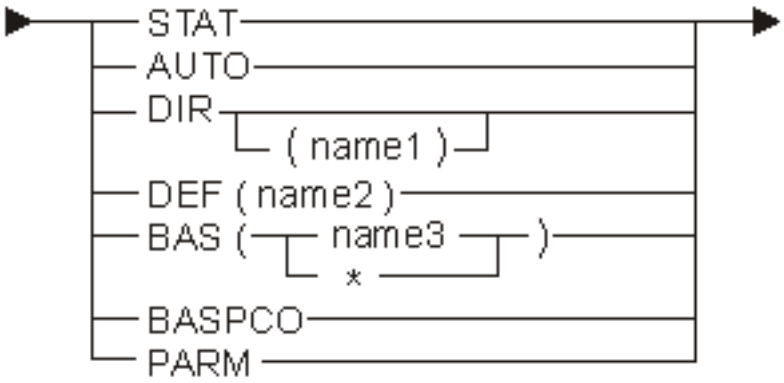
If you specify no value, the system uses BIN(2).

Keyword	Description
CHAR	Scalar type is a character string.
BIN	Scalar type is binary.
UNSGND	Scalar type is unsigned binary. If you do not specify this value, the scalar type is signed binary.
PKD	Scalar type is packed decimal.
ZND	Scalar type is zoned decimal.
FLT	Scalar type is floating-point.

Constant	Range	Description
integer1	See description.	Length in bytes of the character data object. If the data object is an array, the range is 1 to 32 767. Otherwise, the range is 1 to 16 776 191.
integer2	2 or 4	Length in bytes of the binary data object.
integer3	1 to 31	Total digits in the data object.
integer4	0 to integer3	Number of digits to the right of the assumed decimal point in the data object.
integer5	4 or 8	Precision in bytes of the data object.

Addressability

The following diagram and tables show the possible addressabilities:



Keyword	Description
STAT	Addressability type is direct static.
AUTO	Addressability type is direct automatic.
DIR	Addressability type is defined. See “Using Space Objects” on page 53 for more information.
DEF	Addressability type is direct on the previous space.
BAS	Addressability type is based.
*	Object does not have explicit basing object.
BASPCO	Addressability type is based on process communication object space pointer.
PARM	Addressability type is a parameter.

Constant	Range	Description
name1	Any	Space object name
name2	Any	Scalar data object name or pointer data object name
name3	Any	Pointer data object name or space pointer object name

If you specify no value, the system uses STAT.

Scope

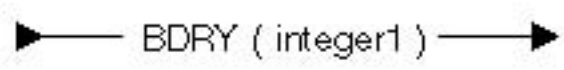
Scope refers to the ability to export a variable so that other programs can access it. The following diagram and table show the possible scopes:

Keyword	Description
INT	Data object is not externally accessible
EXT	Data object is externally accessible

If you specify no value, the system uses INT.

Boundary

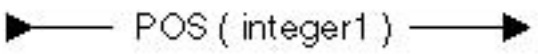
The following diagram and table show the possible boundaries:



Constant	Range	Description
integer1	1, 2, 4, 8, 16	Data object boundary

Position

The following diagram and table show the possible positions:



Constant	Range	Description
integer1	1 to 16 776 191	Data object position

Example

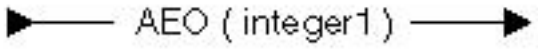
The following declare statements show how POS can be used along with DEF to access the same storage space in different ways:

```
DCL DD DATETIME CHAR(12);
DCL DD DATE CHAR(6) DEF(DATETIME);
DCL DD TIME CHAR(6) DEF(DATETIME) POS(7);
```

DATETIME represents a 12 character time and date stamp. The first 6 characters contain the date and the second 6 characters contain the time.

Array Element Offset

The following diagram and table show the possible array element offsets:



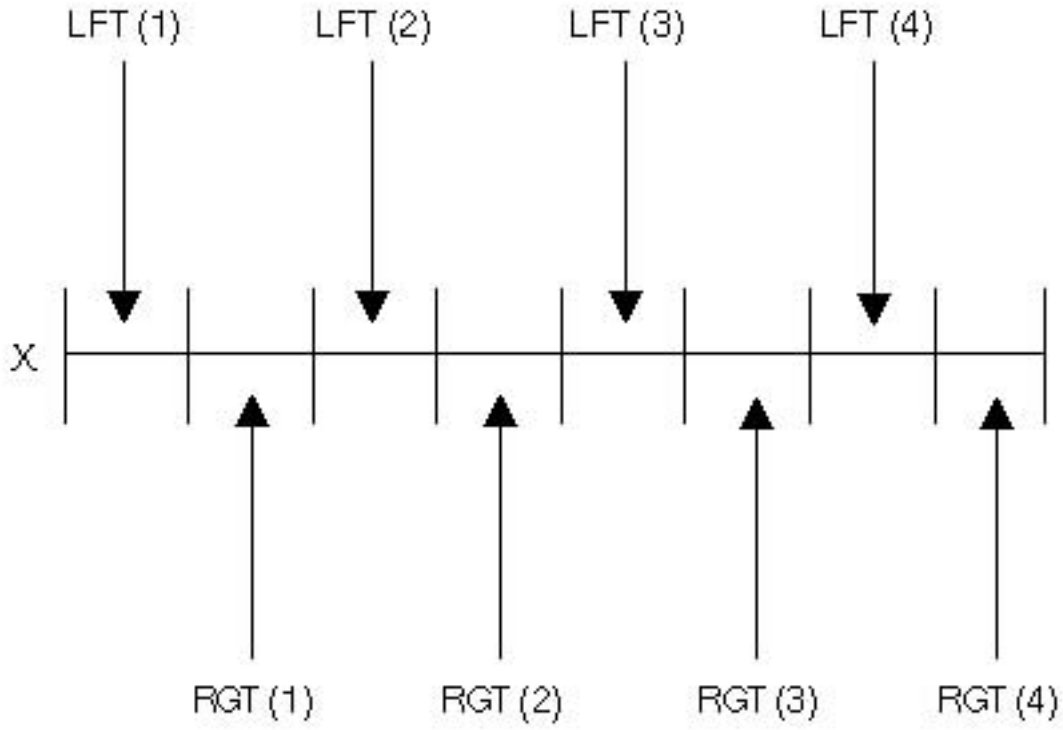
Constant	Range	Description
integer1	1 to 32767	Array element offset

Example

The following example shows AEO used in conjunction with DEF and POS:

```
DCL DD X CHAR(16);
DCL DD LFT(4) BIN(2) DEF(X) AEO(4) POS(1);
DCL DD RGT(4) BIN(2) DEF(X) AEO(4) POS(3);
```

Both LFT and RGT redefine the storage declared by X. Because the size of each array element is smaller than the array element offset, there are 2-byte gaps between each array element:



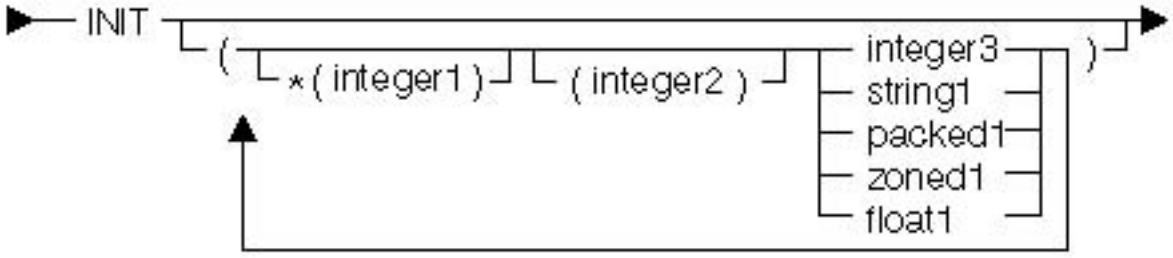
Optimization

Optimization determines whether or not an item can be moved to a register and stored there over time. The following diagram shows the possible optimization:

This value indicates that the data object contains an abnormal value. You cannot optimize the value for more than a single reference because the value may be changed in a manner that the QPRCRTPG API cannot detect.

Initial Value

The following diagram and table show each possible initial value:



Constant	Range	Description
integer1	1 to 16 776 191	Position of elements in a character string
integer1	-2 147 483 648 to 2 147 483 647	Position of elements in an array
integer2	1 to 16 776 191	Replication factor in a character string or array
integer3	Any	Initial value for signed and unsigned binary data objects
string1	Any	Initial value for character string data objects
packed1	Any	Initial value for packed decimal data objects
zoned1	Any	Initial value for zoned decimal data objects
float1	Any	Initial value for floating-point data objects

Example

The following declare statement declares and initializes a 10-element array:

```
DCL DD IV(10) BIN(2) STAT INIT((1)10,*(2)(2)11,*(4)(3)12,*(7)(4)13);
```

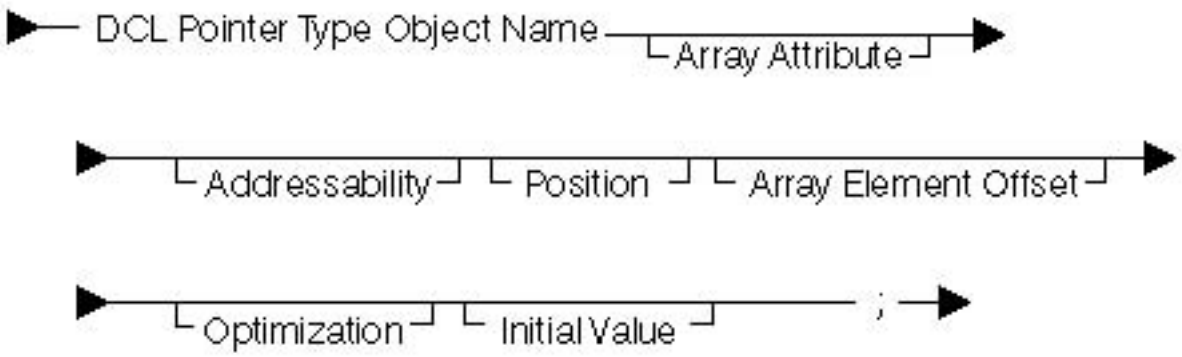
There are four initial value elements. The following table describes this function:

Initial Value Element	Result	Position	Replication Factor	Initial value
(1)10	IV(1)=10	1 (default)	1	10
*(2)(2)11	IV(2)=11 IV(3)=11	2	2	11

Initial Value Element	Result	Position	Replication Factor	Initial value
*(4)(3)12	IV(4)=12 IV(5)=12 IV(6)=12	4	3	12
*(7)(4)13	IV(7)=13 IV(8)=13 IV(9)=13 IV(10)=13	7	4	13

Pointer-Data-Object Declare Statement

The following diagram and table show the pointer-data-object declare statement:



The system only allows certain combinations of attributes based on the data object's addressability. These combinations are listed as follows:

Address-ability	Array Attribute	Array Element Offset Attribute	Position Attribute	Initial Value Attribute
STAT	X		X	X
AUTO	X		X	X
DEF	X	X	X	
			X	X
DIR	X	X	X	
			X	X
BAS	X		X	
BASPCO	X		X	
PARM	X			

Pointer Type

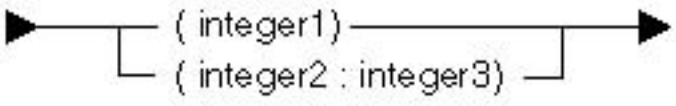
The following diagram and table show the possible pointer types:

Keyword	Description
PTR	Pointer type is not specified.
INSPTR	Pointer type is the instruction pointer.
SPCPTR	Pointer type is the space pointer.
DTAPTR	Pointer type is the data pointer.
SYSPTR	Pointer type is the system pointer.

If you specify an initial value, you must specify INSPTR, SPCPTR, DTAPTR or SYSPTR.

Array Attribute

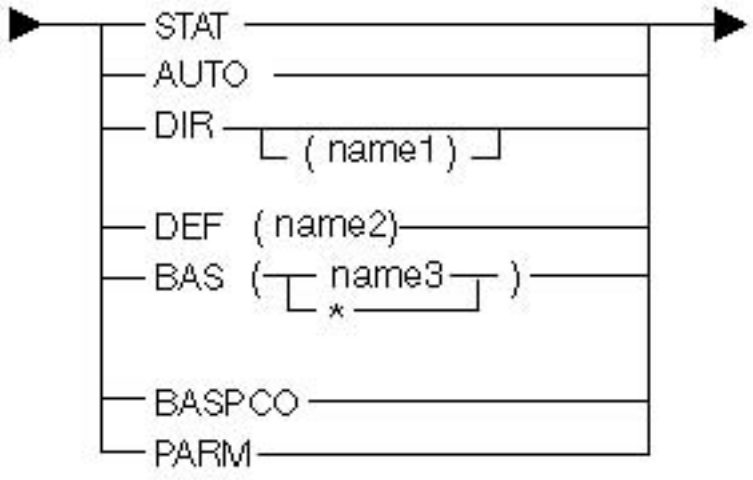
The following diagram and table show the possible array attributes:



Constant	Range	Description
integer1	1 to 1 000 000	Dimension of the data object with an implied lower bound of 1.
integer2	-2 147 483 648 to 2 147 483 647	Lower bound of the array.
integer3	integer2 to 2 147 483 647	Upper bound of the array. The dimension (integer3 - integer2) should not exceed 1 000 000.

Addressability

The following diagram and tables show the possible addressabilities:

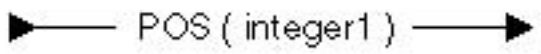


Keyword	Description
STAT	Addressability type is direct static.
AUTO	Addressability type is direct automatic.
DIR	Addressability type is defined. See "Using Space Objects" on page 53 for more information.
DEF	Addressability type is defined.
BAS	Addressability type is based.
*	Object does not have explicit basing object.
BASPCO	Addressability type is based on the process communication object space pointer.
PARM	Addressability type is parameter.

Constant	Range	Description
name1	Any	Space object name
name2	Any	Scalar data object name or the pointer data object name
name3	Any	Pointer data object name or the space pointer machine object name

Position

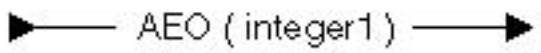
The following diagram and table show the possible positions:



Constant	Range	Description
integer1	1 to 16 776 191	Data object position

Array Element Offset Value

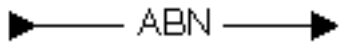
The following diagram and table show the possible array element offset values:



Constant	Range	Description
integer1	1 to 32 767	Array element offset

Optimization

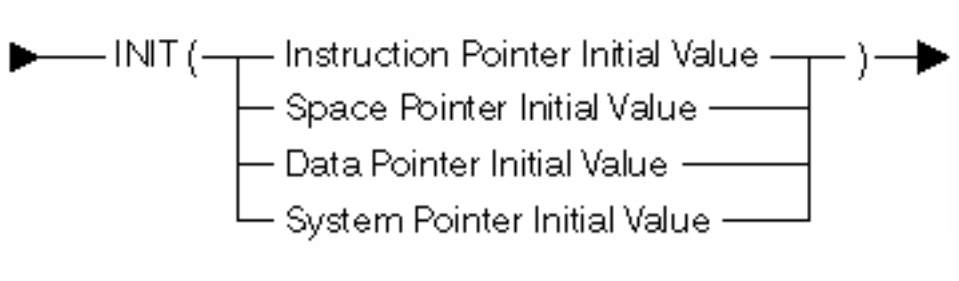
The following diagram shows the possible optimizations:



This value indicates that the data object contains an abnormal value. The system cannot optimize a value for more than a single reference because the value may be changed in a manner the QPRCRTPG API cannot find.

Initial Value

The following diagram shows each possible initial value:



An initial value can only be specified if a pointer-type value other than PTR is specified. The syntax of the initial value is based on the pointer-type value that was used.

Instruction Pointer Initial Value

The following diagram and table show the possible initial value for the instruction pointer:

Constant	Range	Description
name1	Any	Label name

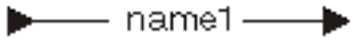
Example

The following statement declares and initializes an instruction pointer:

```
LABELI:

  :
  :

  DCL INSPTR INSTRUCTION_PTR INIT(LABELI);
```

Space Pointer Initial Value

The following diagram and table show the initial value for the space pointer:



Constant	Range	Description
name1	Any	Scalar data object name or pointer data object name

Example

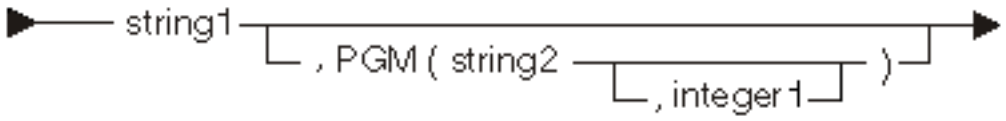
The following statement declares and initializes a space pointer:

```
DCL PTR ANY_POINTER;
DCL SPCPTR SPACE_PTR INIT(ANY_POINTER);
```

The pointer SPACE_PTR is initialized to point to the space location containing ANY_POINTER. It does *not* contain the value of ANY_POINTER.

Data Pointer Initial Value

The following diagram and table show the initial value for the data pointer:



Constant	Range	Description
string1	32 bytes	External data object name
string2	30 bytes	Program containing external data object
integer1	0 to 255	Subtype of program

Example

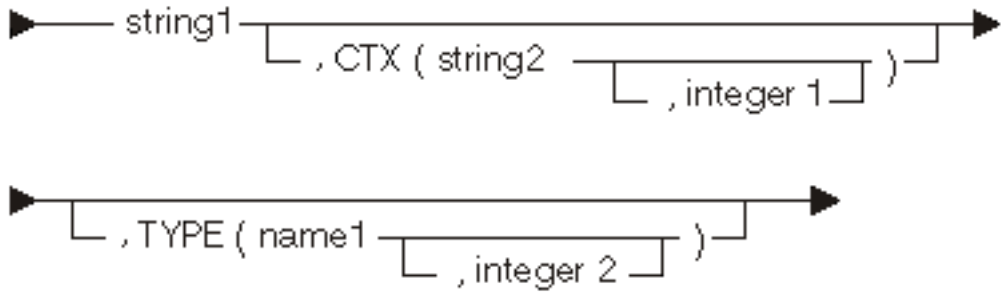
The following statement declares and initializes a data pointer:

```
DCL DTAPTR DVALUE INIT("DBINARY",PGM("DPGM"));
```

The pointer DTAPTR refers to the externally defined program object DBINARY contained in program DPGM.

System Pointer Initial Value

The following diagram and tables show the initial value for the system pointer:



Constant	Range	Description
string1	1 to 30 bytes	System object
string2	1 to 30 bytes	Context where the system object is located

Constant	Range	Description
integer1	0 to 255	Subtype of the context
name1	See table below.	Symbolic type of the system object
integer2	0 to 255	Subtype of the system object

The following system object types are supported:

Type	Description
PGM	Program
CTX	Context
Q	Queue
SPC	Space
PCS	Process control space

Example

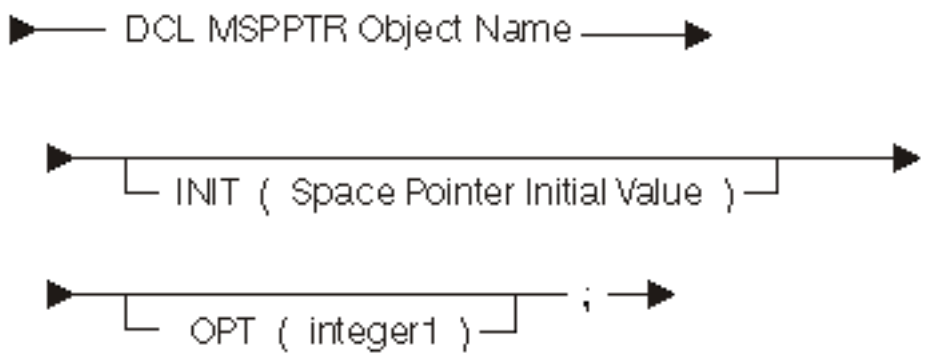
The following statement declares and initializes a system pointer:

```
DCL SYSPTR SYSTEM_PTR INIT("MYPGM",CTX("PGMLIB"),TYPE(PGM));
```

The pointer SYSTEM_PTR refers to the *PGM object MYPGM in the PGMLIB library.

Space-Pointer-Machine-Object Declare Statement

The following diagram and table show the space-pointer-machine-object declare statement:



Constant	Range	Description
integer1	0 to 255	Optimization priority value, where 255 is the highest priority

The iSeries system provides two types of pointers that can access data:

- Space Pointers (SPCPTR)
- Machine Space Pointers (MSPPTR)

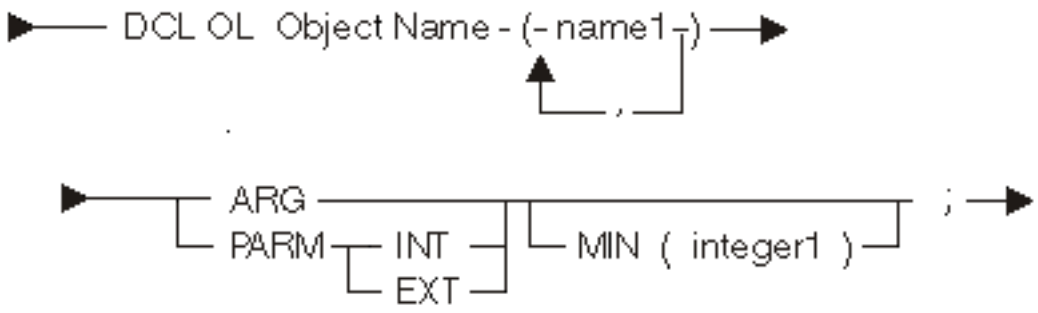
  The MSPPTR has the following restrictions:
  - It cannot be passed as a parameter
  - It cannot be part of a structure (SPC)
  - It cannot be based (BAS(on_some_pointer)) pointer

– It is logically only automatic (AUTO) in storage scope

Because the MSPPTR has the above restrictions, the translator often assigns the MSPPTR to a hardware register for the life of the entire program unit. What this means is that loads may be eliminated from the generated code.

Operand-List Declare Statement

The following diagram and tables show the operand-list declare statement:



Keyword	Description
ARG	Defines the argument list
PARM	Defines the parameter list
INT	An internal parameter list
EXT	An external parameter list

Constant	Range	Description
name1	Any	Scalar data object or a pointer data object name. Up to 255 names can be specified.
integer1	0 to 255	Minimum number of elements that the list can contain. This implicitly defines a variable-length operand list. If you do not specify the operand list, the system defines a fixed-length operand list. Up to 255 names can be specified.

Example

The following statements declare both argument and parameter operand lists along with the associated argument and parameter data objects:

```
DCL DD ARG1 BIN(2);
DCL DD ARG2 CHAR(3);
DCL OL ARGUMENT_LIST (ARG1, ARG2) ARG;

DCL DD PARM1 BIN(2) PARM;
DCL DD PARM2 CHAR(3) PARM;
DCL OL PARAMETER_LIST (PARM1, PARM2) PARM EXT;
```

A parameter operand list that refers to the data objects has parameter (PARM) addressability.

Instruction-Definition-List Declare Statement

The following diagram and table show the instruction-definition-list declare statement:

Constant	Range	Description
name1	Any	Label name. Up to 255 names can be specified.

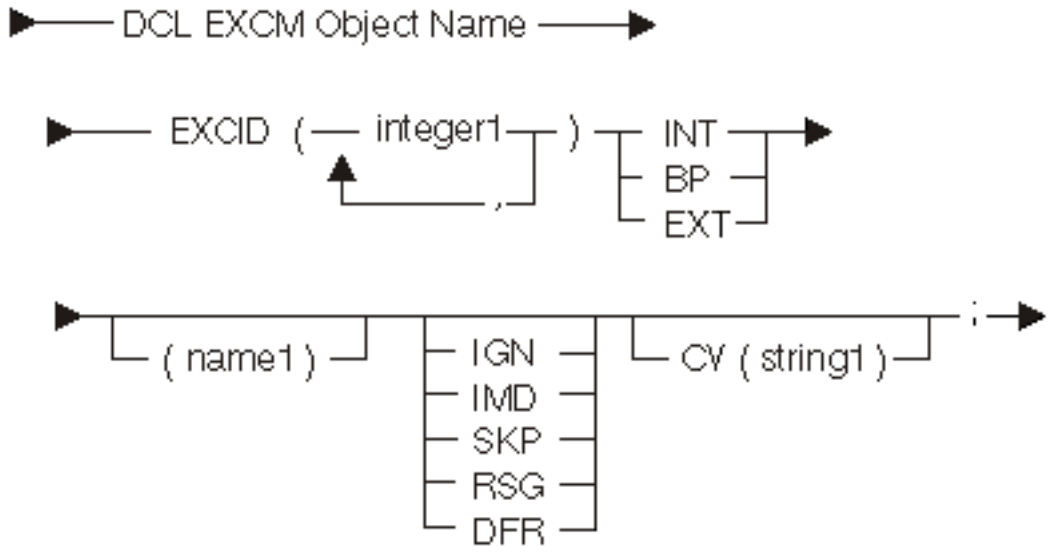
Example

The following statements declare and use an instruction definition list:

```
LABEL1:

  :
  :

  DCL IDL INSTRUCTION_LIST (LABEL1,LABEL2,LABEL3);

  :
  :

LABEL2:
  B INSTRUCTION_LIST(3); /* Branch to LABEL3 */

  :
  :

LABEL3:
```

Exception-Description Declare Statement

The following diagram and tables show the exception-description declare statement:

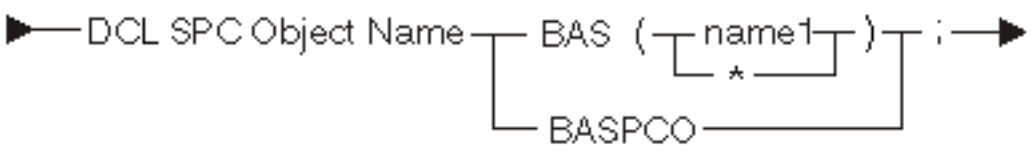


Keyword	Description
INT	Exception handler type is the internal entry point.
BP	Exception handler type is the internal branch point.
EXT	Exception handler type is the external entry point.
IGN	Exception handling action ignores any exceptions and continues processing.
IMD	Exception handling action passes control to the specified exception handler. This is the default.
SKP	Exception handling action is to continue to search for another exception description to handle the exception.
RSG	Exception handling action continues to search for an exception description by signaling the exception again to the previous call.
DFR	Exception handling action postpones handling and saves exception data for later exception handling.

Constant	Range	Description
integer1	0 to 65535	Exception identifier
name1	Any	Name of the label for branch point exception handlers, name of the entry point for the internal exception handlers, and the name of the system pointer for the external exception handlers
string1	1 to 32 bytes	Compare value

Space-Object Declare Statement

The following diagram and tables show the space-object declare statement:



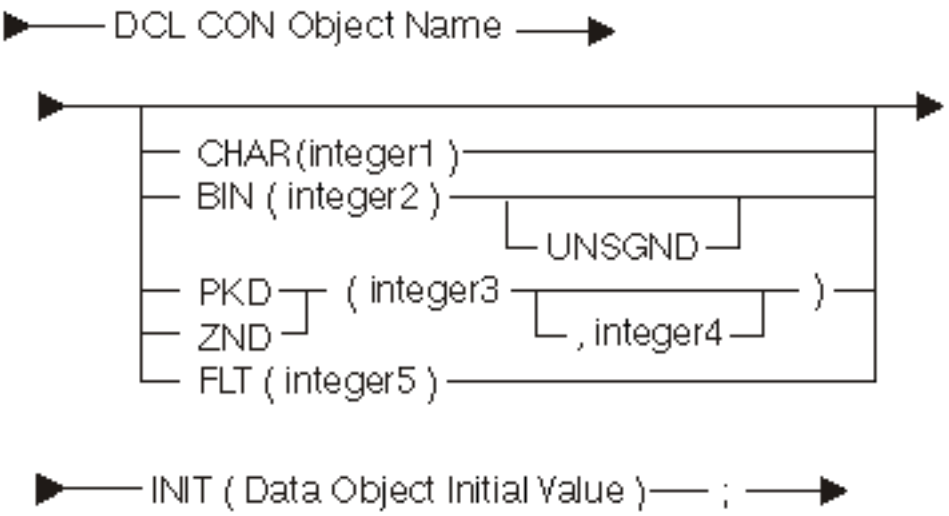
Keyword	Description
BAS	Addressability type is based.
*	Object does not have explicit basing object.
BASPCO	Addressability type is based on process communication object space pointer.

Constant	Range	Description
name1	Any	Basing pointer name for the space

For information on using space objects, refer to "Using Space Objects" on page 53.

Constant-Object Declare Statement

The following diagram and tables show the constant-object declare statement:



Keyword	Description
CHAR	Scalar type is character string.
BIN	Scalar type is binary.
UNSGND	Scalar type is unsigned binary.
PKD	Scalar type is packed decimal.
ZND	Scalar type is zoned decimal.

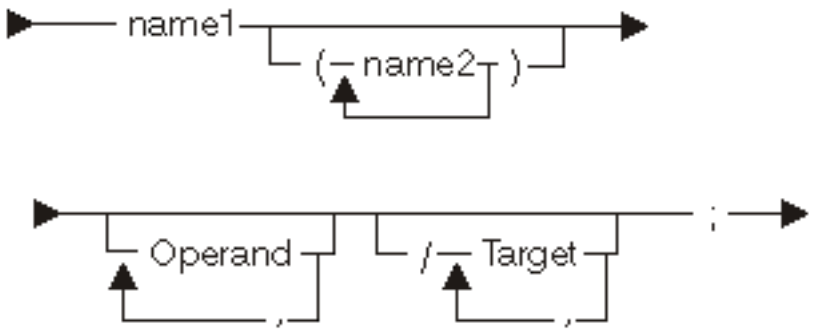
Keyword	Description
FLT	Scalar type is floating-point.

Constant	Range	Description
integer1	1 to 32 767	Length in bytes of the character data object
integer2	2 or 4	Length in bytes of the binary data object
integer3	1 to 31	Number of decimal digits
integer4	0 to integer3	Number of fractional digits
integer5	4 or 8	Number of bytes in floating-point constant

If you do not specify a scalar type, the system uses BIN(2).

Instruction Statement

An instruction statement defines an MI instruction. The instruction stream used to create the program is made up of all the instruction statements in the intermediate representation of the program.



Constant	Range	Description
name1	See description.	Opcode for this instruction, as defined in the iSeries Machine Interface Instructions .
name2	S, R, B, I	This is the form of the instruction. *S* Short *R* Round *B* Branch *I* Indicator

For the semantic meanings and the syntax restrictions (number and types of operands, optional forms, and so on) for individual MI instructions, see the iSeries Machine Interface Instructions.

Following the abbreviated instruction name, you can specify the optional forms of certain MI instructions using a string of characters enclosed in parentheses. The following is an example of some of the various combinations possible for a single MI instruction, ADD NUMERIC:

```
ADDN      A,B,C;              Add numeric (A=B+C)
ADDN(S)   A,B;                Add numeric short (A=A+B)
ADDN(SR)  A,B;                Add numeric short and round (A=A+B)
```

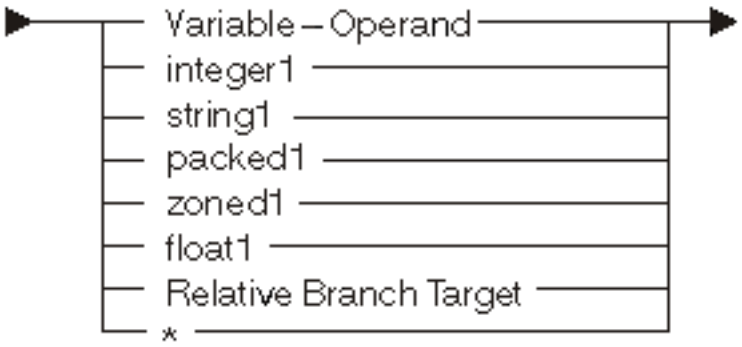
```
ADDN(SB)  A,B/POS(X),NEG(Y);   Add numeric short and branch  (A=A+B,
                               branch to X if A>0, branch to Y if A<0)
ADDN(RI)  A,B,C/POS(I),NEG(J); Add numeric round and indicator (A=B+C;
                               I='on' if A>0; j='on' if A<0 )
```

Also note that the order of characters in the optional form string is not significant. Thus, all of the following instructions are both valid and equivalent:

```
ADDN(SRB)A,B/POS(X);     Add numeric short, round and branch
ADDN(SBR)A,B/POS(X);     Add numeric short, round and branch
ADDN(RSB)A,B/POS(X);     Add numeric short, round and branch
```

Operand

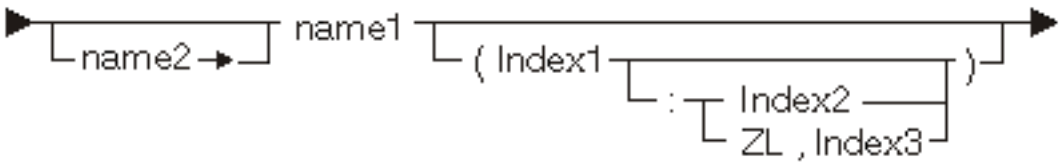
The following diagram and table show the possible operands:



Constant	Range	Description
integer1	Any	Numeric binary scalar operand
string1	Any	Character scalar operand
packed1	Any	Numeric packed decimal scalar operand
zoned1	Any	Numeric zoned decimal scalar operand
float1	Any	Numeric floating-point scalar operand (4 or 8 bytes)
*		Null operand

Variable Operand

The following diagram and table show the possible variable operands:

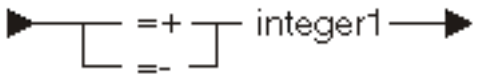


Constant	Range	Description
name1	Any	Data object name to be used as a primary operand.
name2	Any	Pointer data object to be used as the basing pointer.

Constant	Range	Description
Index1	See description.	Subscript or substring start position. The range for array subscripts is between the lower bound of the array and the upper bound of the array. The range for substrings is between 1 and 16 776 191.
Index2	1 to 32 767	Length of the substring.
Index3	0 to 32 767	Length of the substring (zero allowed).

Relative Branch Target

The following diagram and table show the possible relative branch targets:



Constant	Range	Description
integer1	1 to 4095	Branch target instruction number *relative* to the current instruction. You must label the target (named or null label).

Note: You cannot use blanks between either the ’

```
=+
```

’ symbol set and integer1 or the ’

```
=-
```

’ symbol set and integer1. However, a blank must precede the symbol sets.

Example

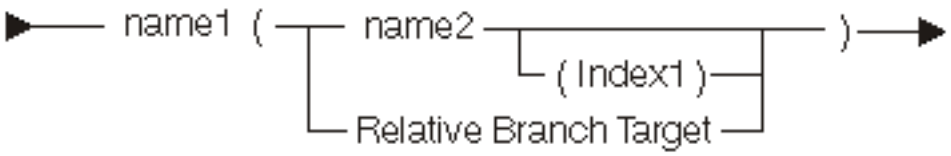
The following instructions illustrate the use of relative branch targets:

```
  CPYNV X,0;
  CMPBLA(B) A,'1'/EQ( =+2);
  CPYNV X,1;
: CPYNV Y,X;        /* Destination of relative branch */
```

Note: A null label is placed in the destination instruction of the relative branch.

Target

The following diagram and table show the possible targets:



Constant	Range	Description
name1	See keyword table.	Keyword for branch or indicator forms. You can use an N before keywords to negate the condition except for IGN and DFR. See ″Resultant Conditions″, under each MI instruction for the valid values.
name2	Any	Label name, instruction pointer name, or instruction definition list name for the branch form. The name of character variable is for the indicator form.
Index1	1 to 255	Instruction definition list index. You can only specify this value when name2 is the name of an instruction definition list.

The following table shows the branch and indicator keywords:

Keyword	Description
Group 1	
HI MXD NOR POS TR ZC	High Mixed Normalized Positive Truncated record Zero and carry
Group 2	
CR DEN IGN LO NEG NTZNTC RO	Complete record Denormalized Exception ignored Low Negative Not-zero and no carry Receiver overrun
Group 3	
AUTH DFR DQ EQ INF SE SGN ZER ZNTC	Authorized Exception postponed Dequeued Equal Infinity Source all used Signaled Zero Zero and no carry
Group 4	
EC NAN NTZC UNEQ UNOR	Escape code encountered Not a number (NaN) Not-zero and carry Unequal Unordered

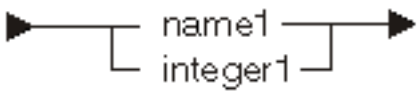
By adding N to the beginning of the appropriate keyword you can form a not condition. For example, the code for ″not equal″ is NEQ.

All conditions coded on a particular instruction must be mutually exclusive. All conditions within a group are equivalent, and therefore, only one may be specified. For example, POS (positive) and HI (high) cannot be coded on the same instruction.

The not form of a condition is satisfied by any condition from another group. For example, NEQ (not equal) is satisfied by HI (high), LO (low), or UNOR (unordered). Therefore, you cannot specify NEQ with any of the other three. However, you can use NEQ and EQ (or any other keyword in group 3) together because they are mutually exclusive.

Index

The following diagram and table show the possible indexes:



Constant	Range	Description
name1	See description below.	Binary variable to use as the index
integer1	See description below.	Integer value to use as the index

An index is a numeric value that qualifies an array or substring reference. The context in which the index is used determines the range. For more information, refer to the preceding tables.

Directive Statements

The directive statements are as follows:

- Title Directive Statement
- Space Directive Statement
- Eject Directive Statement
- Break Directive Statement
- Entry Directive Statement
- Reset Directive Statement
- Program End Directive Statement

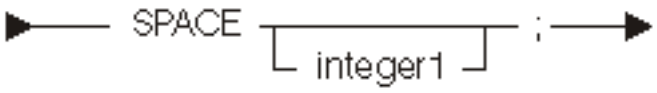
Title Directive Statement

The title directive statement causes a heading to appear on the listings. Only one title directive statement may be specified in a program. The following diagram and table show the title directive statement:

Constant	Range	Description
string1	Any	Text of the title

Space Directive Statement

The space directive statement causes a blank line to appear in the listing. The following diagram and table show the space directive statement:



Constant	Range	Description
integer1	Any	Number of lines to skip

Eject Directive Statement

The eject directive statement causes the next line to appear on a new page. The following diagram shows the eject directive statement:

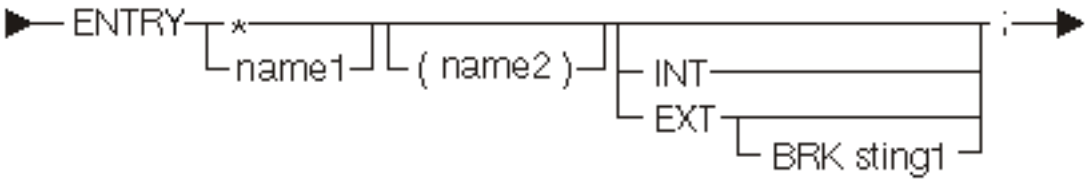
Break Directive Statement

The break directive statement allows symbolic breakpoints to be defined. The following diagram and table show the break directive statement:

Constant	Range	Description
string1	Any	Breakpoint name

Entry Directive Statement

The following diagram and tables show the entry directive statement:



Keyword	Description
INT	Internal entry point.
EXT	External entry point.

Keyword	Description
BRK	Symbolic breakpoint is associated with the entry point.
*	Entry point defined has no name or is associated with the next MI instruction.

Constant	Range	Description
name1	Any	Entry point name being defined
name2	Any	Parameter list name for this entry point
string1	1-10 bytes	Breakpoint name

The default scope is internal (INT).

The entry statement defines entry point program objects. The next instruction number is associated with this entry point. The entry statement is to be the definition point for this object, so the ODT number assigned to this object is the next available ODT number.

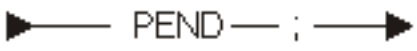
Reset Directive Statement

The following diagram shows the reset directive statement:

The specified name is a previously declared space object. The reset statement causes subsequent data object declarations containing the DIR attribute to use the specified space object. The system maintains next byte counts for each space object; these counts are not affected by the reset statement. For more information, see “Using Space Objects” on page 53.

Program End Directive Statement

The following diagram shows the program end directive statement:



This must be the last statement in the program. To ensure comments and strings end before processing the PEND statement, use the following statement:

```
/*'/*'/*"/*"*/; PEND;;;
```

Coding Techniques

This section contains additional information for coding the intermediate representation of a program.

Using Declare Statements

Use the following guidelines when using declare statements:

- A declare statement for data objects defined on another data object must occur after the declare statement for the data object on which it is defined.

  Example: The following sets of declare statements are valid:

  ```
  DCL DD A CHAR(5);
  DCL DD B CHAR(1) DEF(A);
  ```

  ```
  DCL DD A CHAR(5);
  DCL DD X BIN(2);
  DCL PTR P1 AUTO;
  DCL DD B CHAR(1) DEF(A);
  ```

  Example: The following declare statements are not valid because B is defined on A but is declared before A:

  ```
  DCL DD B CHAR(1) DEF(A);
  DCL DD A CHAR(5);
  ```

  This restriction also applies when there is a chain of dependencies.

  Example: In the figure below, B is defined on A and C is defined on B:

  ```
  DCL DD A CHAR(5);
  DCL DD B CHAR(3) DEF(A);
  DCL DD C CHAR(1) DEF(B);
  ```

  If any object in a chain of definitions, as shown in the previous examples, has an initial value specified, then the following restrictions apply:

  – No object in that chain can have the BAS (based) addressability attribute.
  – The highest level data object in the chain must be either static or automatic.
  – When you initialize the same area twice, the system uses the last value.

  Example: The following declare statements are valid because:

  – The BAS addressability attribute is not used.
  – Data object A (implicitly) has the static addressability attribute.

  ```
  DCL DD A CHAR(5);
  DCL DD B CHAR(3) DEF(A) INIT(C'YES');
  DCL DD C CHAR(1) DEF(B);
  ```

- All declare statements for the objects that make up the elements of an operand list must precede the declare statement for the operand list.
- When a declare statement for an exception description refers to a system pointer, the declare statement for the system pointer must precede the DCL for the exception description.

Using Space Objects

Space objects, when used in conjunction with program objects declared with the DIR attribute, provide a convenient way of declaring structures.

Note: Space objects, as used here, do not refer to i5/OS space objects.

When you declare a space object, a scalar data object with a scalar type of CHAR(32767) is created. This object contains the structure to be defined. Associated with this object is a ″next byte″ count. This value is initially 1 and represents the position where the next structure element will be placed.

Example: Simple Space Objects

After you declare a space object, you can declare one or more scalar or pointer data objects with an addressability attribute of DIR. As a result, the system automatically declares each object with the DEF and POS attributes. The name associated with the DEF attribute is the most recently declared space

object. The value associated with the POS attribute is the space object's next byte count. After you declare the object, the system sets the next byte count associated with the space object to the next available position within the structure.

The group of declare statements on the left is equivalent to the group on the right:

```
DCL SPC X BAS(PTR);        DCL DD X CHAR(32767) BAS(PTR);
DCL DD A CHAR(2) DIR;      DCL DD A CHAR(2) DEF(X) POS(1);
DCL DD B ZND(5,2) DIR;     DCL DD B ZND(5,2) DEF(X) POS(3);
DCL DD C FLT(4) DIR;       DCL DD C FLT(4) DEF(X) POS(8);
```

Example: Explicit Position Values

Data objects declared with DIR may also have an explicit POS value. The object is defined on the appropriate space object and uses the specified POS value. However, the next byte count is changed only if the POS value causes the count to increase.

The group of declare statements on the left is equivalent to the group on the right:

```
DCL SPC X BAS(PTR);             DCL DD X CHAR(32767) BAS(PTR);
DCL DD A CHAR(4) DIR;           DCL DD A CHAR(4) DEF(X) POS(1);
DCL DD B CHAR(4) POS(20) DIR;   DCL DD B CHAR(4) DEF(X) POS(20);
DCL DD C CHAR(4) DIR;           DCL DD C CHAR(4) DEF(X) POS(24);
DCL DD D CHAR(4) POS(10) DIR;   DCL DD D CHAR(4) DEF(X) POS(10);
DCL DD E CHAR(4) DIR;           DCL DD E CHAR(4) DEF(X) POS(28);
```

Example: Explicit Boundary Alignment

When you declare objects with an explicit boundary other than 1, the object is positioned on the next available byte with that boundary. The position of any data object with the direct attribute is the next available byte in the space if no boundary or position is specified. The position of any pointer object with the direct attribute is the next available byte in the space if no position is specified. Space objects are assumed to begin on a 16-byte boundary. You must ensure this condition exists at run-time.

The group of declare statements on the left is equivalent to the group on the right:

```
DCL SPC X BAS(PTR);           DCL DD X CHAR(32767) BAS(PTR);
DCL DD A CHAR(1) DIR;         DCL DD A CHAR(1) DEF(X) POS(1);
DCL DD B FLT(4) DIR; BDRY(4); DCL DD B FLT(4) DEF(X) POS(5);
DCL PTR C DIR; POS(17);       DCL PTR C DEF(X) POS(17);
```

Example: Reset Directive Statement

You can use the reset directive statement to change the name of the space object to be used by subsequent declare statements.

The group of declare statements on the left is equivalent to the group on the right:

```
DCL SPCPTR PTR1;           DCL SPCPTR PTR1;
DCL SPCPTR PTR2;           DCL SPCPTR PTR2;

DCL SPC X BAS(PTR1);       DCL DD X CHAR(32767) BAS(PTR1);
DCL DD A CHAR(2) DIR;      DCL DD A CHAR(2) DEF(X) POS(1);
DCL DD B ZND(5,2) DIR;     DCL DD B ZND(5,2) DEF(X) POS(3);

DCL SPC Y BAS(PTR2);       DCL DD Y CHAR(32767) BAS(PTR2);
DCL DD C CHAR(5) DIR;      DCL DD C CHAR(5) DEF(Y) POS(1);
DCL DD D CHAR(7) DIR;      DCL DD D CHAR(7) DEF(Y) POS(6);

RESET X;
DCL DD E CHAR(3) DIR;      DCL DD E CHAR(3) DEF(X) POS(8);
```

Constants

This section describes the syntax of constant values.

Integer

Integers define signed and unsigned binary scalar data values. The two forms of integers are decimal and hexadecimal. The decimal form is a sequence of digits optionally preceded by a sign. The hexadecimal form is a string of hexadecimal digits delimited with apostrophes and preceded by an H. Neither form may exceed the 4-byte limit on binary numbers. When the value of the integer is between -4095 and +8191, the QPRCRTPG API converts the integer to an immediate operand where it can.

Example

```
+123
-1
54788

H'0F0D'
H'0123'
H'5E2D1AB4'
```

String

Strings define scalar character string data values. The three types of string constants are character form, hexadecimal form, and Hollerith form.

The character form is a delimited string optionally preceded by a C. Apostrophes or double quotation marks may be used for this form. The hexadecimal form is a delimited string of hexadecimal digits preceded by an X. The Hollerith form is a string of bytes preceded by the count of the number of bytes in the string. The syntax is:

```
< count | string >
```

The count in the preceding syntax is the number of characters in the string. The QPRCRTPG API ensures that the string contains the right number of characters by checking for the

```
>
```

character. No blanks are allowed between

```
<
```

and

```
>
```

unless they are part of the string. The QPRCRTPG API simply flags the constant as in error if the right corner bracket does not appear in the correct position.

Example

The following groups of strings are equivalent:

```
'ABCDE'
C'ABCDE'
X'C1C2C3C4C5'
<5|ABCDE>

'TE''ST'
"TE'ST"
X'E3C57DE2E3'
<5|TE'ST>

'/*'
X'615C'
<2|/*>
```

Packed

Packed constants define packed decimal scalar data values. Packed constants are a string of decimal digits delimited with apostrophes. They can have an embedded decimal point and can be preceded by a sign. P must precede the delimited string. Packed constants have a maximum of 31 significant digits.

Note: You must specify at least one numeric digit.

Example

```
P'+123.456'
P'1'
P'-1'
P'-123.345345345345'
P'+.00000000000001'
```

Zoned

Zoned constants define zoned decimal scalar data values. The external representation of zoned constants is the same as that for packed constants except that the preceding character is a Z.

Note: You must specify at least one numeric digit.

Example

```
Z'+123.456'
Z'1'
Z'-1'
Z'-123.345345345345'
Z'+.00000000000001'
```

Floating-Point Constants

Floating-point constants define floating-point scalar data values. You must specify whether the constant is a 4-byte (short floating-point) or an 8-byte (long floating-point) value.

There are two ways to represent floating-point values. First, you can specify floating-point constants as a delimited string of decimal digits possibly with an embedded decimal point and optionally preceded by a sign. An F for short floating-point values or an E for long floating-point values must precede the delimited string. An E in the string determines the start of the base 10 exponent. You specify the exponent as signed.

Second, you can specify floating-point constants as a string of hexadecimal digits. The delimited string must be preceded by an XF for short floating-point values or an XE for long floating-point values.

Note: You must specify at least one numeric digit.

Example

```
Short Floating-Point    Long Floating-Point
Values                  Values

F'0'                    E'0'
F'+12'                  E'+12'
F'-12.21'               E'-12.21'
F'12.34E2'              E'12.34E2'
F'+3.2345678E-02'       E'+3.2345678E-02'

XF'449A4000'            XE'46CE6F37FFBE8722'
XF'40490FD0'            XE'400921F9F01B866E'
```

Several special values are allowed:

```
Short Floating-Point   Long Floating-Point
Values                 Values

F'MNAN'                E'MNAN'     Masked Not A Number
F'UNAN'                E'UNAN'     Unmasked Not A Number
F'+INF'                E'+INF'     Plus Infinity
F'-INF'                E'-INF'     Minus Infinity
```

Note: You must use floating-point constants to initialize floating-point data objects.

Name

Names specified in the intermediate representation of a program are a sequence of characters of up to 48 characters in length. You cannot use the following characters as the first character of the name:

```
blank /,;():<+'%-0123456789
```

You cannot use the following characters in subsequent characters of the name:

```
blank /,;():<+'%
```

Example

```
.NAME
NAME
THIS_IS_A_NAME
THIS_IS_A_NAME_2
&NAME
!NAME
?NAME
.0001
```

Note: Symbols that begin with a period (.) are not inserted into the program's symbol table and may not be referred to by the i5/OS debug function.

Comments

Comments, in the intermediate representation of a program, may appear anywhere in the text. Comments are treated as blanks so they are significant in finding tokens. Comments are a string of characters starting with

```
/*
```

and ending with

```
*/
```

. If a comment occurs immediately following a semicolon, it prints as a separate line (or a multiple line as required) on the listing. If a comment is embedded in a statement, then it appears as a part of that statement, such as a remark.

Example

The following statements are equivalent:

```
CPYBLA A,B;
CPYBLA A, /* C-> */ B ;
CPYBLA A,B;  /* B is based on C */
```

Blanks

You can use strings of blanks of any length in the intermediate representation of a program. Blanks act as delimiters in finding tokens and in some places are necessary as in separating the opcode and operand in an instruction statement.

Example

The following statements are equivalent:

```
ADDN A,B,C;
ADDN       A    ,     B            , C           ;
```

API introduced: V1R3

Top | "Program and CL Command APIs," on page 1 | APIs by category

---

End Preprocessor (QbnEndPreProcessor) API

Required Parameter Group:

1	Qualified input source file name	Input	Char(20)
2	Input source member name	Input	Char(10)
3	Qualified output source file name	Input	Char(20)
4	Output source member name	Input	Char(10)
5	Qualified exit program name	Input	Char(20)
6	Exit program data	Input	Char(*)
7	Length of exit program data	Input	Binary(4)
8	Error code	I/O	Char(*)

Service Program: QBNPREPR
Default Public Authority: *USE
Threadsafe: No

The End Preprocessor (QbnEndPreProcessor) API must be called by every preprocessor after the output source file and preprocessor information is created. It records the fact that a preprocessor was called and may be used to pass information used during module creation. This information can be classified as follows:

- Associated space data
- Extended attribute data
- Preprocessor level data
- Name of an exit program to call at ILE program creation time

The End Preprocessor API then moves the above information in the output source file member where it is used at module creation time. The initial preprocessor may get input from inline data, but all subsequent preprocessors must get their input from the output file member of the previous preprocessor.

The output source file created by a previous preprocessor must not be changed. If the output file has been changed, module creation fails.

Authorities and Locks

Input Source File Authority
*READ and *OBJOPR

Input Source Library Authority
*EXECUTE

Output Source File Authority
*CHANGE and *OBJOPR

Output Source File Member Lock
*EXCL

Output Source Library Authority
*EXECUTE

Required Parameter Group

Qualified input source file name
INPUT; CHAR(20)

The qualified name of the input source file to the preprocessor. The first 10 characters contain the input source file name, which is left-justified and padded with blanks. The second 10 characters contain the input source file library, which is left-justified and padded with blanks. The input source file name can be specified with the following special value:

**INLINE* The input source data is specified as an inline data file.

Input source member name
INPUT; CHAR(10)

The name of the member within the input source file, which is left-justified and padded with blanks. This parameter is ignored if the qualified input source file name parameter is *INLINE.

Qualified output source file name
INPUT; CHAR(20)

The qualified name of the output source file to the preprocessor. The first 10 characters contain the output source file name. The file is left-justified and padded with blanks. The second 10 characters contain the output source file library name. The file is left-justified and padded with blanks.

Output source member name
INPUT; CHAR(10)

The name of the member within the output source file. The file is left-justified and padded with blanks.

Qualified exit program name
INPUT; CHAR(20)

The qualified name of the exit program to be called during module creation. The first 10 characters contain the exit program name, which is left-justified and padded with blanks. The second 10 characters contain the exit program library where the exit program is located, which is left-justified and padded with blanks. You can use this special value for the exit program name:

**NONE* This indicates that there is no exit program.

You can use this special value for the exit program library:

**LIBL* The library list.

The exit program is passed five parameters when called. The first two parameters are the exit program data and the exit program data length. The third parameter is reserved CHAR(10). The fourth and fifth parameters are both reserved BINARY(4).

The exit program data being used in this API is defined by the user.

Exit program data
INPUT; CHAR(*)

Data that is stored with the output source file member. When module creation calls the exit program, a copy of the data is passed. The format of this data is specified by the preprocessor. This value is ignored if *NONE is specified for the qualified exit program name parameter.

Length of exit program data
INPUT; BINARY(4)

The length of the data contained in the exit program data parameter. This value is ignored if *NONE is specified for the qualified exit program name parameter.

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
CPF2207 E	Not authorized to use object &1 in library &3 type *&2.
CPF24B4 E	Severe error while addressing parameter list.
CPF3C90 E	Literal value cannot be changed.
CPF3CF1 E	Error code parameter not valid.
CPF5CA0 E	Input source file name &1 is not valid.
CPF5CA1 E	Exit program name &1 is not valid.
CPF5CEA E	Library value &1 is not valid.
CPF5CA4 E	Error occurred while addressing API Parameter.
CPF5D20 E	Not able to open source file member &3.
CPF5D21 E	Not able to open source file member &3.
CPF5D22 E	Not able to locate internal data.
CPF5D23 E	Source file member has been changed.
CPF5D24 E	Unexpected error occurred during preprocessor processing.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

API introduced: V3R1

Top | "Program and CL Command APIs," on page 1 | APIs by category

Execute Command (QCMDEXC) API

Required Parameter Group:

1	Command string	Input	Char(*)
2	Length of command string	Input	Packed (15,5)

Optional Parameter:

3	IGC process control	INPUT	Char(3)

Default Public Authority: *USE
Threadsafe: Yes.
See "Usage Notes" on page 61 for command considerations.

The Execute Command (QCMDEXC) API runs a single command. It is used to run a command from within a high-level language (HLL) program or from within a CL program where it is not known at compile time what command is to be run or what parameters are to be used.

QCMDEXC is called from within your HLL program and the command it runs is passed to it as a parameter on the CALL command.

After the command runs, control returns to your HLL program.

Notes:

1. Command strings in System/38 syntax can use the QCAEXEC API. The QCAEXEC API accepts the same parameters as QCMDEXC.
2. The Process Commands (QCAPCMD) API also provides similar functions. »
3. If the command to be executed is a proxy command, the QCMDEXC API will resolve to the target command. If the target command is also a proxy, the process repeats until either a non-proxy command is found, or the proxy chain becomes greater than the allowed maximum. Once a non-proxy command is found, the resolved command will replace the proxy command in the command string to be executed.
4. Proxy commands will be resolved before the command exit points QIBM_QCA_CHG_COMMAND and QIBM_QCA_RTV_COMMAND are called. «

Authorities and Locks

Any Command
*USE

Required Parameter Group

Command string
INPUT;CHAR(*)

The command you want to run entered as a character string. If the command contains blanks, it must be enclosed in apostrophes. The maximum length of the character string is 32,702 characters; delimiters (the apostrophes enclosing the string) are not counted as part of the string.

Length of command string
INPUT;PACKED(15,5)

The maximum length being passed. If the command string is passed as a quoted string, the command length is exactly the length of the quoted string. If the command string is passed in a variable, the command length is the length of the variable. It is not necessary to reduce the command length to the actual length of the command string in the variable, although it is permissible to do so.

Optional Parameter Group

IGC process control
INPUT;CHAR(3)

The IGC process control instructs the system to accept double-byte data. The only value supported is IGC. IGC must be entered using all uppercase letters.

Usage Notes

While this API is threadsafe, it should not be used to run a command that is not threadsafe in a job that has multiple threads. Use the Display Command (DSPCMD) command to determine whether a command is threadsafe.

Error Messages

Message ID	Error Message Text
CPF0005 E	Returned command string exceeds variable provided length

Message ID	Error Message Text
CPF0006 E	Errors occurred in command.
CPF3C90 E	Literal value cannot be changed.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.
xxxnnnn E	Any escape message issued by any command may be returned. The messages listed previously are those issued by this API. Once the API has called the command analyzer, any message issued as an escape message may appear.

API in existance prior to V1R3

Top | "Program and CL Command APIs," on page 1 | APIs by category

---

Get Export (QleGetExp) API

Omissible Parameter Group:

1	Activation mark	Input	Binary(4)
2	Export number	Input	Binary(4)
3	Export name length	Input	Binary(4)
4	Export name	Input	Char(*)
5	Exported item	Output	PTR(OPN)
6	Type of export item	Output	Binary(4)
7	Error code	I/O	Char(*)

Returned Value:

Exported item	Output	PTR(OPN)

Service Program: QLEAWI
Default Public Authority: *USE
Threadsafe: Yes

The Get Export (QleGetExp) API allows the caller to resolve a pointer to an export (either data or procedure) either by name or export number. The pointer is materialized for the specified activation. If the activation mark given is zero, then all activations in the activation group are searched (no guaranteed search order). The QleGetExp API is identical to the "Get Export Long (QleGetExpLong) API" on page 64 (QleGetExpLong) API, except that the QleGetExpLong API takes a BINARY(8) activation mark as the first parameter instead of a BINARY(4) activation mark.

Authorities and Locks

None.

Omissible Parameter Group

Activation mark

INPUT; BINARY(4)

The activation containing the export. If this parameter is omitted, then it is treated as if a 0 was specified. This parameter may not be omitted if the search is done by export number.

The following special value is supported for this parameter:

0 All of the activations in the caller's activation group are searched. If more than one activation contains the specified export, it is undefined as to which of those activations the export is taken from.

Export number

INPUT; BINARY(4)

Materialize the nth exported identifier in the service program. The order is defined by the binding service language with the first exported identifier being 1. If this parameter is omitted, then it is treated as if a 0 was specified.

The following special value is supported for this parameter:

0 Materialize the item named in the export name parameter.

Export name length
INPUT; BINARY(4)

The length of the export name. If this parameter is omitted, then it is treated as if a 0 was specified. This parameter is ignored if the export number parameter is not zero.

The following special value is supported for this parameter:

0 The export name is a null-terminated string.

Export name
INPUT; CHAR(*)

A string containing the name of the exported identifier. The name is matched exactly, without CCSID conversion or folding to uppercase. This parameter is ignored if the export number parameter is not zero. The export name cannot be omitted if the export number is omitted.

Exported item
OUTPUT; PTR(OPN)

The procedure pointer or space pointer to the exported item. If the identifier could not be exported, this value is null.

Type of export item
OUTPUT; BINARY(4)

The type of the exported item. The possible types follow:

0 Export was not found
1 Export is a procedure
2 Export is data
3 Export not accessible

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.

Returned Value

Pointer to exported item
OUTPUT; PTR(OPN)

This API returns the value for the pointer to the exported item parameter.

Error Messages

Message ID	Error Message Text
CPF24B4 E	Severe error while addressing parameter list.
CPF3C1D E	Length specified in parameter &1 not valid.

Message ID	Error Message Text
CPF3C1E E	Required parameter &1 omitted.
CPF3C90 E	Literal value cannot be changed.
CPF3CF1 E	Error code parameter not valid.
CPF3CF2 E	Error(s) occurred during running of &1 API.
MCH4421 E	At least one field in the allocation strategy is not valid.
MCH4422 E	&1 cannot be called in the default activation group.

API introduced: V3R6

Top | "Program and CL Command APIs," on page 1 | APIs by category

---

Get Export Long (QleGetExpLong) API

Omissible Parameter Group:

1	Activation mark	Input	Binary(8)
2	Export number	Input	Binary(4)
3	Export name length	Input	Binary(4)
4	Export name	Input	Char(*)
5	Exported item	Output	PTR(OPN)
6	Type of export item	Output	Binary(4)
7	Error code	I/O	Char(*)

Returned Value:

Exported item	Output	PTR(OPN)

Service Program: QLEAWI
Default Public Authority: *USE
Threadsafe: Yes

The Get Export Long (QleGetExpLong) API allows the caller to resolve a pointer to an export (either data or procedure) either by name or export number. The pointer is materialized for the specified activation. If the activation mark given is zero, then all activations in the activation group are searched (no guaranteed search order). The QleGetExpLong API is identical to the "Get Export (QleGetExp) API" on page 62 (QleGetExp) API, except that the QleGetExp API takes a BINARY(4) activation mark as the first parameter instead of a BINARY(8) activation mark.

Authorities and Locks

None.

Omissible Parameter Group

Activation mark

INPUT; BINARY(8)

The activation containing the export. If this parameter is omitted, then it is treated as if a 0 was specified. This parameter may not be omitted if the search is done by export number.

The following special value is supported for this parameter:

0 All of the activations in the caller's activation group are searched. If more than one activation contains the specified export, it is undefined as to which of those activations the export is taken from.

Export number

INPUT; BINARY(4)

Materialize the nth exported identifier in the service program. The order is defined by the binding service language with the first exported identifier being 1. If this parameter is omitted, then it is treated as if a 0 was specified.

The following special value is supported for this parameter:

0 Materialize the item named in the export name parameter.

Export name length
INPUT; BINARY(4)

The length of the export name. If this parameter is omitted, then it is treated as if a 0 was specified. This parameter is ignored if the export number parameter is not zero.

The following special value is supported for this parameter:

0 The export name is a null-terminated string.

Export name
INPUT; CHAR(*)

A string containing the name of the exported identifier. The name is matched exactly, without CCSID conversion or folding to uppercase. This parameter is ignored if the export number parameter is not zero. The export name cannot be omitted if the export number is omitted.

Exported item
OUTPUT; PTR(OPN)

The procedure pointer or space pointer to the exported item. If the identifier could not be exported, this value is null.

Type of export item
OUTPUT; BINARY(4)

The type of the exported item. The possible types follow:

0 Export was not found
1 Export is a procedure
2 Export is data
3 Export not accessible

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.

Returned Value

Pointer to exported item
OUTPUT; PTR(OPN)

This API returns the value for the pointer to the exported item parameter.

Error Messages

Message ID	Error Message Text
CPF24B4 E	Severe error while addressing parameter list.
CPF3C1D E	Length specified in parameter &1 not valid.

Message ID	Error Message Text
CPF3C1E E	Required parameter &1 omitted.
CPF3C90 E	Literal value cannot be changed.
CPF3CF1 E	Error code parameter not valid.
CPF3CF2 E	Error(s) occurred during running of &1 API.
MCH4421 E	At least one field in the allocation strategy is not valid.
MCH4422 E	&1 cannot be called in the default activation group.

API introduced: V5R3

Top | "Program and CL Command APIs," on page 1 | APIs by category

---

List ILE Program Information (QBNLPGMI) API

Required Parameter Group:

1	Qualified user space name	Input	Char(20)
2	Format name	Input	Char(8)
3	Qualified ILE program name	Input	Char(20)
4	Error Code	I/O	Char(*)

Default Public Authority: *USE
Threadsafe: No

The List ILE Program Information (QBNLPGMI) API gives information about ILE programs, similar to the Display Program (DSPPGM) command. The information is placed in a user space specified by you.

If an original program model (OPM) program is specified for the qualified ILE program name, an error is returned and the user space is not changed.

You can use the QBNLPGMI API to:

- List modules bound into an ILE program
- List service programs bound to an ILE program
- List data items exported to the activation group
- List data item imports that are resolved by weak exports that were exported to the activation group
- List copyrights of an ILE program

Authorities and Locks

User Space Authority
*CHANGE

User Space Library Authority
*EXECUTE

User Space Lock
*EXCLRD

Program Authority for PGML0100 and PGML0110 Formats
*USE

Program Authority for other Formats
*READ

Program Library Authority
*EXECUTE

Program Lock
*SHRRD

Required Parameter Group

Qualified user space name
INPUT; CHAR(20)

The user space that is to receive the ILE program information. 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 library name can be a specific library name or one of these special values:

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

Format name
INPUT; CHAR(8)

The content and format of the information to be returned about the specified programs. One of the following format names may be used:

"PGML0100 Format" on page 68	ILE program module (*MODULE) information.
"PGML0110 Format" on page 70	Variable length ILE program module (*MODULE) information. **Note:** Do not use the generic header entry size for format PGML0110. Use the Size of this entry field returned in this format for the size of each entry.
"PGML0200 Format" on page 72	ILE service program (*SRVPGM) information.
"PGML0300 Format" on page 72	Data items exported to the activation group (*ACTGRPEXP).
"PGML0400 Format" on page 73	Data item imports resolved by weak exports that were exported to the activation group (*ACTGRPIMP).
"PGML0500 Format" on page 73	ILE program copyright (*COPYRIGHT) information.

Qualified ILE program name
INPUT; CHAR(20)

The name of the ILE program for which the information is to be listed. The first 10 characters contain the ILE program name, and the second 10 characters contain the name of the library where the ILE program is located.

The ILE program name can be a specific ILE program name or one of the following special values:

**ALL*	All ILE programs
*generic**	All ILE programs that begin with this generic prefix. For example, WRK* would include all ILE programs that begin with WRK.

The library name can be a specific library name or one of these special values:

**ALL*	All libraries in the system
**ALLUSR*	All non-system libraries. For information on the libraries included, see *ALLUSR in Generic library names.
**CURLIB*	The job's current library
**LIBL*	The library list
**USRLIBL*	Libraries listed in the user portion of the library list

Error code

I/O; CHAR(*)

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

Format of the Generated List

The user space contains:

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

For details about the user area and generic header, see User Space Format for List APIs. For descriptions of each field in the list returned, see "Field Descriptions" on page 73.

Input Parameter Section

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	User space name specified
10	A	CHAR(10)	User space library name specified
20	14	CHAR(8)	Format name specified
28	1C	CHAR(10)	Program name specified
38	26	CHAR(10)	Program library name specified

Header Section

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	User space name used
10	A	CHAR(10)	User space library name used

PGML0100 Format

The PGML0100 format includes information on all the modules that are bound into the programs specified. The modules will be listed in the user space in the order the modules are bound into the program. You must have a program authority of *USE to use this format. The following table shows how this information for each module is organized. For detailed descriptions of the fields in the list, see "Field Descriptions" on page 73.

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	Program name
10	A	CHAR(10)	Program library name
20	14	CHAR(10)	Bound module name

Offset		Type	Field
Dec	Hex		
30	1E	CHAR(10)	Bound module library name
40	28	CHAR(10)	Source file name
50	32	CHAR(10)	Source file library name
60	3C	CHAR(10)	Source file member name
70	46	CHAR(10)	Module attribute
80	50	CHAR(13)	Module creation date and time
93	5D	CHAR(13)	Source file updated date and time
106	6A	CHAR(10)	Sort sequence table name
116	74	CHAR(10)	Sort sequence table library name
126	7E	CHAR(10)	Language identifier
136	88	BINARY(4)	Optimization level
140	8C	BINARY(4)	Maximum optimization level
144	90	CHAR(10)	Debug data
154	9A	CHAR(6)	Release module created on
160	A0	CHAR(6)	Release module created for
166	A6	CHAR(20)	Reserved
186	BA	CHAR(1)	User-modified
187	BB	CHAR(13)	Licensed program
200	C8	CHAR(5)	PTF number
205	CD	CHAR(6)	APAR ID
211	D3	CHAR(1)	Creation data
212	D4	BINARY(4)	Module CCSID
216	D8	CHAR(8)	Object control level
224	E0	CHAR(1)	Enable performance collection
225	E1	CHAR(10)	Profiling data
235	EB	CHAR(1)	Reserved
236	EC	BINARY(4)	Number of procedures
240	F0	BINARY(4)	Number of procedures block reordered
244	F4	BINARY(4)	Number of procedures block-order measured
248	F8	CHAR(1)	Teraspace storage enabled
249	F9	CHAR(1)	Storage model
250	FA	CHAR(74)	Reserved
324	144	BINARY(4)	Number of SQL statements
328	148	CHAR(18)	Relational database
346	15A	CHAR(10)	Commitment control
356	164	CHAR(10)	Allow copy of data
366	16E	CHAR(10)	Close SQL cursors
376	178	CHAR(10)	Naming convention
386	182	CHAR(10)	Date format
396	18C	CHAR(1)	Date separator

Offset		Type	Field
Dec	Hex		
397	18D	CHAR(10)	Time format
407	197	CHAR(1)	Time separator
408	198	CHAR(10)	Delay PREPARE
418	1A2	CHAR(10)	Allow blocking
428	1AC	CHAR(10)	Default collection name
438	1B6	CHAR(10)	SQL package name
448	1C0	CHAR(10)	SQL package library name
458	1CA	CHAR(10)	Dynamic user profile
468	1D4	CHAR(10)	SQL sort sequence table name
478	1DE	CHAR(10)	SQL sort sequence table library name
488	1E8	CHAR(10)	SQL language identifier
498	1F2	CHAR(10)	Connection method
508	1FC	BINARY(4)	SQL path length
512	200	CHAR(3483)	SQL path

PGML0110 Format

The PGML0110 format includes information on all the modules that are bound into the programs specified. The modules will be listed in the user space in the order the modules are bound into the program. You must have a program authority of *USE to use this format. The following table shows how this information for each module is organized. For detailed descriptions of the fields in the list, see "Field Descriptions" on page 73.

Note: Do not use the generic header entry size for this format. Use the Size of this entry field returned in this format for the size of each entry.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Size of this entry
4	4	CHAR(10)	Program name
14	E	CHAR(10)	Program library name
24	18	CHAR(10)	Bound module name
34	22	CHAR(10)	Bound module library name
44	2C	CHAR(10)	Source file name
54	36	CHAR(10)	Source file library name
64	40	CHAR(10)	Source file member name
74	4A	CHAR(10)	Module attribute
84	54	CHAR(13)	Module creation date and time
97	61	CHAR(13)	Source file updated date and time
110	6E	CHAR(10)	Sort sequence table name
120	78	CHAR(10)	Sort sequence table library name
130	82	CHAR(10)	Language identifier

Offset		Type	Field
Dec	Hex		
140	8C	BINARY(4)	Optimization level
144	90	BINARY(4)	Maximum optimization level
148	94	CHAR(10)	Debug data
158	9E	CHAR(6)	Release module created on
164	A4	CHAR(6)	Release module created for
170	AA	CHAR(20)	Reserved
190	BE	CHAR(1)	User-modified
191	BF	CHAR(13)	Licensed program
204	CC	CHAR(5)	PTF number
209	D1	CHAR(6)	APAR ID
215	D7	CHAR(1)	Creation data
216	D8	BINARY(4)	Module CCSID
220	DC	CHAR(8)	Object control level
228	E4	CHAR(1)	Enable performance collection
229	E5	CHAR(10)	Profiling data
239	EF	CHAR(1)	Reserved
240	F0	BINARY(4)	Number of procedures
244	F4	BINARY(4)	Number of procedures block reordered
248	F8	BINARY(4)	Number of procedures block-order measured
252	FC	CHAR(1)	Teraspace storage enabled
253	FD	CHAR(1)	Storage model
254	FE	CHAR(2)	Reserved
256	100	BINARY(4)	Offset to Licensed Internal Code options
260	104	BINARY(4)	Length of Licensed Internal Code options
264	108	CHAR(64)	Reserved
328	148	BINARY(4)	Number of SQL statements
332	14C	CHAR(18)	Relational database
350	15E	CHAR(10)	Commitment control
360	168	CHAR(10)	Allow copy of data
370	172	CHAR(10)	Close SQL cursors
380	17C	CHAR(10)	Naming convention
390	186	CHAR(10)	Date format
400	190	CHAR(1)	Date separator
401	191	CHAR(10)	Time format
411	19B	CHAR(1)	Time separator
412	19C	CHAR(10)	Delay PREPARE
422	1A6	CHAR(10)	Allow blocking
432	1B0	CHAR(10)	Default collection name
442	1BA	CHAR(10)	SQL package name
452	1C4	CHAR(10)	SQL package library name

Offset		Type	Field
Dec	Hex		
462	1CE	CHAR(10)	Dynamic user profile
472	1D8	CHAR(10)	SQL sort sequence table name
482	1E2	CHAR(10)	SQL sort sequence table library name
492	1EC	CHAR(10)	SQL language identifier
502	1F6	CHAR(10)	Connection method
512	200	BINARY(4)	SQL path length
516	204	CHAR(3483)	SQL path
1074	432	CHAR(*)	Reserved
Bound module information through offsets			
		CHAR(*)	Licensed Internal Code options

PGML0200 Format

The PGML0200 format includes information on all the service programs that are bound to the programs specified. The following table shows how this information is organized. For detailed descriptions of the fields in the list, see "Field Descriptions" on page 73.

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	Program name
10	A	CHAR(10)	Program library name
20	14	CHAR(10)	Bound service program name
30	1E	CHAR(10)	Bound service program library name
40	28	CHAR(16)	Bound service program signature

PGML0300 Format

The PGML0300 format lists data items exported to the activation group. The list data items are specified in the data export entry in the binding specifications component when the module was created. The following table shows how this information is organized. For detailed descriptions of the fields in the list, see "Field Descriptions" on page 73.

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	Program name
10	A	CHAR(10)	Program library name
20	14	BINARY(4)	Size of data item export
24	18	BINARY(4)	Data item export name CCSID
28	1C	BINARY(4)	Data item export name length
32	20	CHAR(256)	Data item export name

PGML0400 Format

The PGML0400 format lists data item imports that were resolved by weak exports that had been exported to an activation group. The following table shows how this information is organized. For detailed descriptions of the fields in the list, see "Field Descriptions."

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	Program name
10	A	CHAR(10)	Program library name
20	14	BINARY(4)	Data item import name CCSID
24	18	BINARY(4)	Data item import name length
28	1C	CHAR(256)	Data item import name

PGML0500 Format

The PGML0500 format includes copyright information for the ILE programs specified. The following table shows how this information is organized. For detailed descriptions of the fields in the list, see "Field Descriptions."

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	Program name
10	A	CHAR(10)	Program library name
20	14	CHAR(4)	Reserved
24	18	BINARY(4)	Copyright length
28	1C	CHAR(256)	Copyright

Field Descriptions

Allow blocking. Whether blocking will be used to improve the performance of certain SQL statements. The possible values are:

**NONE*	Blocking is not used.
**READ*	Blocking is used for read-only cursors when running COMMIT(*NONE) and there are no EXECUTE or EXECUTE IMMEDIATE statements.
**ALLREAD*	Blocking is used for all read-only cursors when running COMMIT(*NONE) or COMMIT(*CHG).
Blank	The module does not contain SQL statements.

Allow copy of data. Whether a copy of the data can be used in the implementation of an SQL query. The possible values are:

**NO*	A copy of the data cannot be used.
**YES*	A copy of the data can be used when needed.
**OPTIMIZE*	The system determines whether a copy of the data is used for optimal performance.
Blank	The module does not contain SQL statements.

APAR ID. The module was changed as the result of the authorized program analysis report (APAR) with this identification number. This is blank if the module was not changed at bind time.

Bound module library name. The name of the library containing the module bound into this program at bind time.

Bound module name. The name of the module bound into this program. This is a copy of the module that was bound into this program. It is not the *MODULE object on the system.

Bound service program library name. The name of the library containing the service program bound to the program at bind time. This is the library name in which the activation expects to find the service program at run time. Hexadecimal zeros indicate the library list is used at the time the service program is needed.

Bound service program name. The name of the service program bound to the program.

Bound service program signature. The current signature of the service program at the time it was bound to the program.

Close SQL cursors. Specifies when SQL cursors are implicitly closed and SQL-prepared statements are implicitly discarded. The possible values are:

**ENDMOD*	When the module ends.
**ENDACTGRP*	When the activation group is deleted.
Blank	The module does not contain SQL statements.

Commitment control. The level of commitment control that was specified on the SQL precompile. The possible values are:

**NONE*	No commitment control was specified on the SQL precompile. Uncommitted changes in other jobs can be seen.
**CHG*	Objects referred to in SQL COMMENT ON, CREATE, DROP, GRANT, LABEL ON, and REVOKE statements, and updated, deleted, or inserted rows (records) are locked until the end of the unit of work (transaction). Uncommitted changes in other jobs can be seen.
**CS*	Objects referred to in SQL COMMENT ON, CREATE, DROP, GRANT, LABEL ON, and REVOKE statements, and updated, deleted, and inserted rows (records) are locked until the end of the unit of work (transaction). A row (record) that is selected but not updated is locked until the next row (record) is selected. Uncommitted changes in other jobs cannot be seen.
**ALL*	Objects referred to in SQL COMMENT ON, CREATE, DROP, GRANT, LABEL ON, and REVOKE statements, and all rows selected, updated, deleted, and inserted are locked until the end of the unit of work (transaction). Uncommitted changes in other jobs cannot be seen.
Blank	The module does not contain SQL statements.

Connection method. The method used for establishing remote connections when running distributed programs.

Special values that can be returned are:

**RUW*	Only one connection to a relational database is allowed. Consecutive CONNECT statements result in the previous connection being disconnected before a new connection is established.
**DUW*	Connections to several relational databases are allowed. Consecutive CONNECT statements to additional relational databases does not result in disconnection of previous connects. SET CONNECTION can be used to switch between connections. Read-only connections may result.
Blank	The module does not contain SQL statements.

Copyright. The copyright string included in this program.

Copyright length. The length of the copyright string.

Creation data. Whether the bound module has all creation data and if that data is observable or unobservable.

0 *NO. Not all the creation data is present in the bound module.
1 *YES. The creation data is present in the bound module and all of that data is observable.
2 *UNOBS. The creation data is present in the bound module but not all of that data is observable.

Data item export name. Data items that are exported to an activation group. These data items can be used outside of the module or program that they are defined in.

Data item export name CCSID. The coded character set identifier (CCSID) for the name of this data item export.

Data item export name length. The length of the name of the data export item.

Data item import name. The name of the data item imports that were resolved by weak exports that had been exported to the activation group.

Data item import name CCSID. The coded character set identifier (CCSID) for the name of this data item import.

Data item import name length. The length of the name of the data import item.

Date format. The format used when accessing date-result columns through SQL. All output date fields are returned in this format. For input date strings, the value you specify is used to determine whether the date is a valid format. The values returned are:

**USA*	USA format (mm/dd/yyyy).
**ISO*	International Standards Organization format (yyyy-mm-dd).
**EUR*	European format (dd.mm.yyyy).
**JIS*	Japanese Industrial Standard Christian Era (yyyy-mm-dd).
**MDY*	Month/day/year format (mm/dd/yy).
**DMY*	Day/month/year format (dd/mm/yy).
**YMD*	Year/month/day format (yy/mm/dd).
**JUL*	Julian format (a numeric value from 1 to 365).
Blank	The module does not contain SQL statements.

Date separator. The separator used when accessing date-result columns. This information is blank if the module does not contain SQL statements; however, the number of SQL statements field should be checked to determine if the module contains SQL statements. This is because a blank may be specified as a separator value.

Debug data. Whether debug data was generated when this module was created. If debug data exists, the module may be debugged using the source debugger. The possible values are:

**YES*	Debug data was generated.
**NO*	Debug data was not generated.

Default collection name. The collection name used for the unqualified names of tables, views, indexes, and SQL packages. The possible values are:

**NONE*	There is no default collection name.
Blank	The module does not contain SQL statements.

Delay PREPARE. Whether SQL prepare processing can be delayed until the statement is actually used. The possible values are:

**YES*	Prepare processing can be delayed.
**NO*	Prepare processing cannot be delayed.
Blank	The module does not contain SQL statements.

Dynamic user profile. The user profile used for dynamic SQL statements. The following special values can be returned:

**USER*	Local dynamic SQL statements are run under the profile of the module's user. Distributed dynamic SQL statements are run under the profile of the SQL package's user.
**OWNER*	Local dynamic SQL statements are run under the profile of the module's owner. Distributed dynamic SQL statements are run under the profile of the SQL package's owner.
Blank	The module does not contain SQL statements.

Enable performance collection. The level of performance collection enabled for this module. The following values can be returned:

*'00'X *NONE or '10'X *PEP*	This gives the entry/exit information for the PEP only. No entry/exit hooks in the module's internal procedures and no precall or postcall hooks around calls to other procedures are included. **Note:** If *NONE is shown and the module was created or re-created on an iSeries server running Version 3 Release 6 Modification 0 prior to the installation of PTF MF11968, the module will not have any performance collection enabled. To enable performance collection, use one of the following commands and specify ENBFPRCOL(*PEP): • Change Module (CHGMOD) • Change Program (CHGPGM) • Change Service Program (CHGSRVPGM)
*'50'X *ENTRYEXIT *NONLEAF*	This gives the entry/exit information on all of the non-leaf procedures in the module. This includes the PEP routine. This is useful to capture information on most routines but not at the expense of destroying the 'leaf-ness' of the leaf procedures.
*'70'X *ENTRYEXIT *ALLPRC*	This gives the entry/exit information on all the procedures of the module (including those that were leaf procedures). This includes the PEP routine. This is useful to capture information on all procedures.
*'D0'X *FULL *NONLEAF*	This gives the entry/exit information on all the procedures of the module that are not leaf procedures. This includes the PEP routine. Precall and postcall hooks around calls to external procedures are also included.
*'F0'X *FULL *ALLPRC*	This gives the entry/exit information on all procedures of the module (including those that were leaf procedures). This includes the PEP routine. Precall and postcall hooks around calls to external procedures are also included. This is useful to capture information on all procedures.

Format name specified. The format used to return the ILE program information to the user space.

Language identifier. Returns the 3-character language identifier used when the module was compiled. The following special values can also be returned:

**JOBRUN*	The language identifier associated with the job at the time the program that the module is bound into runs.
Blank	The module does not contain any language identification information.

Length of Licensed Internal Code options. The size, in two-byte characters, of the Licensed Internal Code options string. This will be 0 if no Licensed Internal Code options were used for this module.

Licensed Internal Code options. The Licensed Internal Code options that are in use by the module. This field is specified in UCS-2 (CCSID 13488).

Licensed program. If the module was part of a licensed program at bind time, this field shows the product number and the level of the licensed program. This is blank if the module is not part of a licensed program at bind time.

Maximum optimization level. The highest level of optimization this module could have at bind time. If observability has been removed from the module, this maximum optimization level value might not be the same as the one specified when the module was created. Possible values are:

65535	The module is not restricted to a maximum optimization level. It can be retranslated to any of the supported optimization levels. 65535 is also known as *NOMAX.
40	Maximum level of optimization. This level includes all the optimizations performed at optimization level 30. In addition, it includes optimization that disables call and instruction tracing. Thus, tracing of modules created at this optimization level cannot be done.
30	More optimization is performed in addition to those performed at optimization level 20. Variables cannot be changed but can be displayed while the program is being debugged. However, the displayed value of the variable during debugging may not be its actual value. 30 is also known as *FULL.
20	Some optimization is performed on the generated code. When the module optimized at this level is being debugged, the variables can be displayed but not changed. This level improves the performance of the module slightly over level 10. 20 is also known as *BASIC.
10	No additional optimization is performed on the generated code. Variables can be displayed and changed when the program is being debugged. With no optimization of the code, this value provides the lowest level of module performance. 10 is also known as *NONE.

Module attribute. The language used in the module. This field can be blank (for example, when a module is created by a compilation process internal to IBM).

Module CCSID. The coded character set identifier (CCSID) for this module.

Module creation date and time. The date and time the module was created. The creation date and time field is in the CYYMMDDHHMMSS format as follows:

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

Naming convention. The convention used for naming objects in SQL statements. The possible values are:

**SQL*	The SQL naming convention is used.
**SYS*	The system naming convention is used.
Blank	The module does not contain SQL statements.

Number of procedures. The number of procedures defined in the module. This number includes the program entry procedure (PEP), if one was generated by the compiler for this module.

Number of procedures block-order measured. The number of procedures defined in the module that had block-order profiling data collected at the time block-order profiling data was applied. If the module does not have block-order profiling data applied, this value will be zero.

Number of procedures block reordered. The number of procedures defined in the module that are block reordered. If the module does not have block-order profiling data applied, this value will be zero. This value can decrease if the program that this bound module is contained in is retranslated.

Number of SQL statements. The number of SQL statements contained in the module. This value is zero if the module does not contain SQL statements.

Object control level. The object control level for the module at the time it was bound into this program. You can compare the object control level of a module to the object control level of a listing to make sure the listing matches the module.

Offset to Licensed Internal Code options. The offset from the beginning of the user space where the Licensed Internal Code options begin for this bound module. This will be 0 if no Licensed Internal Code options were used for this module.

Optimization level. Optimization levels cause the translator to produce machine code that reduces the amount of system resources necessary to run the program. The more optimization, the more efficiently the module runs on the system. Also, with more optimization you may not be able to change or view variables that have been optimized. The possible values are:

65535	The module is not restricted to a maximum optimization level. It can be retranslated to any of the supported optimization levels. 65535 is also known as *NOMAX.
40	Maximum level of optimization. This level includes all the optimizations performed at optimization level 30. In addition, it includes optimization that disables call and instruction tracing. Thus, tracing of modules created at this optimization level cannot be done.
30	More optimization is performed in addition to those performed at optimization level 20. Variables cannot be changed but can be displayed while the program is being debugged. However, the displayed value of the variable during debugging may not be its actual value. 30 is also known as *FULL.
20	Some optimization is performed on the generated code. When the module optimized at this level is being debugged, the variables can be displayed but not changed. This level improves the performance of the module slightly over level 10. 20 is also known as *BASIC.
10	No additional optimization is performed on the generated code. Variables can be displayed and changed when the program is being debugged. With no optimization of the code, this value provides the lowest level of module performance. 10 is also known as *NONE.

Profiling data. The profiling data attribute for the module bound into this program. Possible values are:

**NOCOL*	The collection of profiling data is not enabled and block-order profiling data is not applied to the module bound into this program.
**COL*	The collection of profiling data is enabled. Any applied block-order profiling data has been removed for the module bound into this program.
**APYBLKORD*	Block-order profiling data is applied to the module bound into this program. See the number of procedures block reordered field for the current number of procedures in this module that are block reordered.

Program library name. The name of the library containing the program.

Program library name specified. The program library name that was passed to this API on the call in the qualified ILE program name and library parameter.

Program name. The name of the program.

Program name specified. The program name that was passed to this API on the call in the qualified ILE program name and library parameter.

PTF number. The program temporary fix (PTF) that resulted in the creation of the module. This field is blank for user-created modules.

Relational database. The default relational database that was specified on the SQL precompile. A nonblank value other than *LOCAL specifies the name of the relational database to be resolved through the relational database directory. The following special values can be returned:

**LOCAL*	The module can only access data on the local system.
Blank	The module does not contain SQL statements.

Release module created for. The version, release, and modification level of the operating system for which the module was created. The field has a VvRrMm format, where:

Vv	The character V is followed by a 1-character version number.
Rr	The character R is followed by a 1-character release level.
Mm	The character M is followed by a 1-character modification level.

Release module created on. The version, release, and modification level of the operating system on which the module was created. The field has a VvRrMm format, where:

Vv	The character V is followed by a 1-character version number.
Rr	The character R is followed by a 1-character release level.
Mm	The character M is followed by a 1-character modification level.

Reserved. An ignored field.

Size of data item export. The size, in bytes, of the data item export.

Size of this entry. The size, in bytes, of this entry.

Sort sequence table library name. The name of the library that is used to locate the sort sequence table. This information is blank if the module does not contain any sort sequence information or a special value was returned for the sort sequence table name. The following special values can be returned:

**LIBL*	The sort sequence table is found in the library list when the ILE program runs this module.
**CURLIB*	The sort sequence table is found in the current library when the ILE program runs this module.

Sort sequence table name. The name of the sort sequence table and the library used when the module was compiled. This does not apply to SQL statements in the module. The following special values can be returned:

**HEX*	No sort sequence is used.
**JOBRUN*	The sort sequence is the sort sequence value associated with the job at the time the ILE program runs this module.
**LANGIDSHR*	The shared sort sequence for the language identifier is used.
**LANGIDUNQ*	The unique sort sequence for the language identifier is used.

Note: This sort sequence table does not apply to SQL statements.

Source file library name. The name of the library that contains the source file used to create the module. The field is blank if no source file was used to create the module.

Source file member name. The name of the member in the source file. The field is blank if no source file was used to create the module.

Source file name. The name of the source file used to create the module. The field is blank if no source file was used to create the module.

Source file updated date and time. The date and time the member in the source file was last updated. The field is in the same format as the module creation date and time field. The field is blank if no source file was used to create the module.

SQL language identifier. Returns the 3-character language identifier used when the module was compiled. This information is blank if the module does not contain any language identification information. The following possible special value can also be returned:

**JOBRUN*	The language identifier is the LANGID associated with the job at the time the module is run.

SQL package library name. The name of the library the SQL package is in.

SQL package name. The name of the SQL package created on the relational database specified on the RDB parameter of the command that created this module. The possible values are:

**NONE*	There is no default collection name.
Blank	The module does not contain SQL statements.

SQL path. The list of libraries used during resolution of functions and data types within SQL statements. The list is in the form of repeating library names, each surrounded by double quotes and separated by commas. Even though 3843 bytes are reserved, the path's length is determined by the SQL path length entry.

SQL path length. The length, in bytes, of the SQL path.

SQL sort sequence table library name. The name of the library that is used to locate the SQL sort sequence table. This information is blank if the module does not contain any SQL sort sequence information or a special value was returned for the SQL sort sequence table name. The following special values can be returned:

**LIBL*	The SQL sort sequence table is found by looking in the library list.
**CURLIB*	The SQL sort sequence table is found by looking in the current library.

SQL sort sequence table name. The name of the table name used when the module was compiled. This information is blank if the module does not contain any SQL sort sequence information. The following special values can be returned:

**HEX*	No SQL sort sequence is used for the SQL statements.
**JOBRUN*	The SQL sort sequence is the SRTSEQ value associated with the job at the time the SQL statements within the module are run.
**LANGIDSHR*	The shared SQL sort sequence for the language identifier (LANGID) is used for the SQL statements.
**LANGIDUNQ*	The unique SQL sort sequence for the language identifier (LANGID) is used for the SQL statements.

Storage model. Where the automatic and static storage for this bound module is allocated at run time. The following values can be returned:

0 *SNGLVL — Automatic and static storage are allocated from single-level storage.
1 *TERASPACE — Automatic and static storage are allocated from teraspace.
2 *INHERIT — Automatic and static storage are allocated from either single-level storage or teraspace, depending on the activation.

Teraspace storage enabled. The teraspace storage capability for this bound module. Possible values are:

*'00'X *NO* — The module bound to this program is not teraspace storage enabled.
*'80'X *YES* — The module bound to this program is teraspace storage enabled.

Time format. The format used when accessing time-result columns through SQL. All output time fields are returned in this format. The values returned are:

**USA* — USA format (hh:mm a.m. or p.m.).
**ISO* — International Standards Organization format (hh.mm.ss).
**EUR* — European format (hh.mm.ss).
**JIS* — Japanese Industrial Standard Christian Era (hh.mm.ss).
**HMS* — Hours/minutes/seconds format (hh:mm:ss).
Blank — The module does not contain SQL statements.

Time separator. The separator used when accessing time-result columns. This information is blank if the module does not contain SQL statements; however, the number of SQL statements field should be checked to determine if the module contains SQL statements. This is because a blank may be specified as a separator value.

User-modified. Whether the module was changed by the user at bind time. The possible values are:

0 — The user did not change the module.
1 — The user changed the module.

User space library name specified. The user space library name that was passed to this API on the call in the qualified user space name parameter.

User space library name used. The name of the library that contains the user space that receives the ILE program information requested.

User space name specified. The user space name that was passed to this API on the call in the qualified user space name parameter.

User space name used. The name of the user space that receives the ILE program information requested.

Error Messages

Message ID	Error Message Text
CPF24B4 E	Severe error while addressing parameter list.
CPF3C20 E	Error found by program &1.
CPF3C21 E	Format name &1 is not valid.
CPF3C90 E	Literal value cannot be changed.
CPF3CAA E	List is too large for user space &1.
CPF3CF1 E	Error code parameter not valid.
CPF5CF5 E	&1 in library &2 not bound program.
CPF5CF6 E	Program name &1 not valid special value.
CPF811A E	User space &4 in &9 damaged.

Message ID	Error Message Text
CPF9570 E	Error occurred creating or accessing debug data.
CPF9801 E	Object &2 in library &3 not found.
CPF9802 E	Not authorized to object &2 in &3.
CPF9803 E	Cannot allocate object &2 in library &3.
CPF9804 E	Object &2 in library &3 damaged.
CPF9806 E	Cannot perform function for object &2 in library &3.
CPF9807 E	One or more libraries in library list deleted.
CPF9808 E	Cannot allocate one or more libraries on library list.
CPF9810 E	Library &1 not found.
CPF9811 E	Program &1 in library &2 not found.
CPF9820 E	Not authorized to use library &1.
CPF9821 E	Not authorized to program &1 in library &2.
CPF9830 E	Cannot assign library &1.
CPF9838 E	User profile storage limit exceeded.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

API introduced: V2R3

Top | "Program and CL Command APIs," on page 1 | APIs by category

---

List Module Information (QBNLMODI) API

Required Parameter Group:

1	Qualified user space name	Input	Char(20)
2	Format name	Input	Char(8)
3	Qualified module name	Input	Char(20)
4	Error code	I/O	Char(*)

Default Public Authority: *USE
Threadsafe: No

The List Module Information (QBNLMODI) API lists information about modules. The information is placed in a user space specified by you. This API is similar to the Display Module (DSPMOD) command.

You can use the QBNLMODI API to:

- List the symbols defined that can be exported to other modules
- List the symbols that are defined external to the module
- List procedure names and their type
- List objects that are referenced when the module is bound into an ILE program or service program
- List copyright information

Authorities and Locks

User Space Authority
*CHANGE

User Space Library Authority
*EXECUTE

User Space Lock
*EXCLRD

Module Authority
*USE

Module Library Authority
*EXECUTE

Module Lock
*SHRRD

Required Parameter Group

Qualified user space name
INPUT; CHAR(20)

The user space that is to receive the module information. The first 10 characters contain the user space name. The second 10 characters contain the name of the library where the user space is located. The library name can be a specific library name or one of these special values:

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

Format name
INPUT; CHAR(8)

The content and format of the information to be returned about the specified modules. One of the following format names may be used:

"MODL0100 Format" on page 84	Module export (*EXPORT) information.
"MODL0200 Format" on page 85	Module import (*IMPORT) information.
"MODL0300 Format" on page 85	Module procedures (*PROCLIST) information.
"MODL0400 Format" on page 86	Referenced system objects (*REFSYSOBJ) information.
"MODL0500 Format" on page 86	Module copyright (*COPYRIGHT) information.

Note: Do not use the generic header entry size for formats returned by this API. Use the Size of this entry field returned in each format for the size of each entry.

Qualified module name
INPUT; CHAR(20)

The name of the module for which the information is to be listed. The first 10 characters contain the module name. The second 10 characters contain the name of the library where the module is located.

The module name can be a specific module name or one of the following special values:

**ALL*	All modules
*generic**	All modules that begin with this generic prefix. For example, WRK* lists information for all modules that begin with WRK to which you are authorized.

The library name can be a specific library name or one of these special values:

**ALL*	All libraries in the system
**ALLUSR*	All non-system libraries. For information on the libraries included, see *ALLUSR in Generic library names.
**CURLIB*	The job's current library
**LIBL*	The library list
**USRLIBL*	Libraries listed in the user portion of the library list

Error code

I/O; CHAR(*)

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

Format of the Generated List

The user space contains:

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

For details about the user area and generic header, see User Space Format for List APIs. For descriptions of each field in the list returned, see "Field Descriptions" on page 87.

Note: Do not use the generic header entry size for formats returned by this API. Use the Size of this entry field returned in each format for the size of each entry.

Input Parameter Section

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	User space name specified
10	A	CHAR(10)	User space library name specified
20	14	CHAR(8)	Format name specified
28	1C	CHAR(10)	Module name specified
38	26	CHAR(10)	Module library name specified

Header Section

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	User space name used
10	A	CHAR(10)	User space library name used

MODL0100 Format

The MODL0100 format lists the symbols defined in the module and that are exported to other modules. The following table shows how this information for each module is organized. For detailed descriptions of the fields in the list, see "Field Descriptions" on page 87.

Note: Do not use the generic header entry size for this format. Use the Size of this entry field returned in this format for the size of each entry.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Size of this entry
4	4	CHAR(10)	Module name
14	E	CHAR(10)	Module library name
24	18	CHAR(1)	Exported defined symbol type
25	19	CHAR(3)	Reserved
28	1C	BINARY(4)	Offset to exported defined symbol name
32	20	BINARY(4)	Length of exported defined symbol name
36	24	CHAR(10)	Uses argument optimization (ARGOPT)
46	2E	CHAR(*)	Reserved
Module information through offsets			
		CHAR(*)	Exported defined symbol name

MODL0200 Format

The MODL0200 format lists symbols defined external to the module. The following table shows how this information for each module is organized. For detailed descriptions of the fields in the list, see "Field Descriptions" on page 87.

Note: Do not use the generic header entry size for this format. Use the Size of this entry field returned in this format for the size of each entry.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Size of this entry
4	4	CHAR(10)	Module name
14	E	CHAR(10)	Module library name
24	18	CHAR(1)	Imported (unresolved) symbol type
25	19	CHAR(3)	Reserved
28	1C	BINARY(4)	Offset to imported (unresolved) symbol name
32	20	BINARY(4)	Length of imported (unresolved) symbol name
36	24	CHAR(10)	Uses argument optimization (ARGOPT)
46	2E	CHAR(*)	Reserved
Module information through offsets			
		CHAR(*)	Imported (unresolved) symbol name

MODL0300 Format

The MODL0300 format lists procedure names and their types. The following table shows how this information for each module is organized. For detailed descriptions of the fields in the list, see "Field Descriptions" on page 87.

Note: Do not use the generic header entry size for this format. Use the Size of this entry field returned in this format for the size of each entry.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Size of this entry
4	4	CHAR(10)	Module name
14	E	CHAR(10)	Module library name
24	18	CHAR(1)	Procedure type
25	19	CHAR(3)	Reserved
28	1C	BINARY(4)	Offset to procedure name
32	20	BINARY(4)	Length of procedure name
36	24	CHAR(10)	Uses argument optimization (ARGOPT)
46	2E	CHAR(*)	Reserved
Module information through offsets			
		CHAR(*)	Procedure name

MODL0400 Format

The MODL0400 format lists the objects that are referenced by the module when the module is bound to an ILE program or service program. The following table shows how this information for each module is organized. For detailed descriptions of the fields in the list, see "Field Descriptions" on page 87.

Note: Do not use the generic header entry size for this format. Use the Size of this entry field returned in this format for the size of each entry.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Size of this entry
4	4	CHAR(10)	Module name
14	E	CHAR(10)	Module library name
24	18	CHAR(10)	Object type
34	22	CHAR(10)	Object library name
44	2C	BINARY(4)	Offset to object name
48	30	BINARY(4)	Length of object name
52	34	CHAR(*)	Reserved
Module information through offsets			
		CHAR(*)	Object name

MODL0500 Format

The MODL0500 format lists the copyrights contained in the module. The following table shows how this information for each module is organized. For detailed descriptions of the fields in the list, see "Field Descriptions" on page 87.

Note: Do not use the generic header entry size for this format. Use the Size of this entry field returned in this format for the size of each entry.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Size of this entry
4	4	CHAR(10)	Module name
14	E	CHAR(10)	Module library name
24	18	BINARY(4)	Offset to copyright
28	1C	BINARY(4)	Length of copyright
32	20	CHAR(*)	Reserved
Module information through offsets			
		CHAR(*)	Copyright

Field Descriptions

Copyright. Copyright information for the module.

Exported defined symbol name. An exported procedure or variable in this module. Other modules may use these symbols.

Exported defined symbol type. Indicates whether the exported symbol is a procedure or a data symbol. The possible values are:

'00'X The exported symbol is a procedure.
'01'X The exported symbol is a data item.

Format name specified. The format name that was passed to this API on the call in the format parameter.

Imported (unresolved) symbol name. An imported procedure or variable in this module. This is typically a reference to a procedure exported from another module.

Imported (unresolved) symbol type. Indicates whether the imported symbol is a procedure or a data symbol. The possible values are:

'00'X The imported symbol is a procedure.
'01'X The imported symbol is a data item.

Length of copyright. The length of the copyright.

Length of exported defined symbol name. The length of the exported defined symbol name.

Length of imported (unresolved) symbol name. The length of the imported (unresolved) symbol name.

Length of object name. The length of the object name.

Length of procedure name. The length of the procedure name.

Module library name. The name of the library containing the module.

Module library name specified. The module library name that was passed to this API in the qualified module name parameter.

Module name. The name of the module.

Module name specified. The module name that was passed to this API in the qualified module name parameter.

Object library name. The name of the library where the object exists. If the object library name is blank, the object is in the integrated file system.

Object name. A system object that is referenced at bind time. This object (modules and/or service programs and/or binding directories) is used by CRTPGM or CRTSRVPGM when this module is listed on the MODULE parameter on CRTPGM or CRTSRVPGM.

Object type. The object type of the system object that is referenced at bind time. The possible special values are:

**MODULE*	The object is a module.
**SRVPGM*	The object is a service program.
**BNDDIR*	The object is a binding directory.

Offset to copyright. Offset from the beginning of the user space to the copyright for this entry.

Offset to exported defined symbol name. Offset from the beginning of the user space to the exported defined symbol name for this entry.

Offset to imported (unresolved) symbol name. Offset from the beginning of the user space to the imported (unresolved) symbol name for this entry.

Offset to object name. Offset from the beginning of the user space to the object name for this entry.

Offset to procedure name. Offset from the beginning of the user space to the procedure name for this entry.

Procedure name. A procedure defined in this module.

Procedure type. The type of procedure. The possible values are:

'00'X	Regular procedure. If the type is regular, this procedure cannot serve as a program entry procedure.
'01'X	Entry point procedure. If the type is entry point, this procedure can serve as a program entry procedure when this module is bound to a program.

Reserved. An ignored field.

Size of this entry. The size, in bytes, of this entry. Do not use the generic header entry size for formats with this field. Use this field for the size of each entry.

User space library name specified. The user space library name that was passed to this API on the call in the qualified user space name parameter.

User space library name used. The name of the library that contains the user space that receives the module information requested.

User space name specified. The user space name that was passed to this API on the call in the qualified user space name parameter.

User space name used. The name of the user space that receives the module information requested.

Uses argument optimization (ARGOPT). Whether or not the procedure import or export uses argument optimization. For data imports and exports, this value has no meaning and is always given as a blank. For some procedure imports, this value cannot be determined and is given as *UNKNOWN. Possible values follow:

**YES*	The procedure import or export uses argument optimization (ARGOPT)
**NO*	The procedure import or export does not use argument optimization (ARGOPT)
**UNKNOWN*	The procedure import is used only to construct procedure pointers and is never called directly.
Blank	The symbol is not a procedure import or export.

Error Messages

Message ID	Error Message Text
CPD5CFE E	Module &1 in library &2 is in error.
CPF24B4 E	Severe error while addressing parameter list.
CPF3C20 E	Error found by program &1.
CPF3C21 E	Format name &1 is not valid.
CPF3C90 E	Literal value cannot be changed.
CPF3CAA E	List is too large for user space &1.
CPF3CF1 E	Error code parameter not valid.
CPF5CFD E	Module name &1 not a valid special value.
CPF5CFE E	Module &1 in file &2 not changed.
CPF811A E	User space &4 in &9 damaged.
CPF8122 E	&8 damage on library &4.
CPF8123 E	Damage on object information for library &4.
CPF8130 E	Character in quoted name not valid.
CPF9801 E	Object &2 in library &3 not found.
CPF9802 E	Not authorized to object &2 in &3.
CPF9803 E	Cannot allocate object &2 in library &3.
CPF9804 E	Object &2 in library &3 damaged.
CPF9806 E	Cannot perform function for object &2 in library &3.
CPF9807 E	One or more libraries in library list deleted.
CPF9808 E	Cannot allocate one or more libraries on library list.
CPF9810 E	Library &1 not found.
CPF9811 E	Program &1 in library &2 not found.
CPF9820 E	Not authorized to use library &1.
CPF9821 E	Not authorized to program &1 in library &2.
CPF9830 E	Cannot assign library &1.
CPF9838 E	User profile storage limit exceeded.

API introduced: V3R7

Top | "Program and CL Command APIs," on page 1 | APIs by category

List Service Program Information (QBNLSPGM) API

Required Parameter Group:

1	Qualified user space name	Input	Char(20)
2	Format name	Input	Char(8)
3	Qualified service program name	Input	Char(20)
4	Error Code	I/O	Char(*)

Default Public Authority: *USE
Threadsafe: No

The List Service Program Information (QBNLSPGM) API gives information about service programs, similar to the Display Service Program (DSPSRVPGM) command. The information is placed in a user space specified by you.

You can use the QBNLSPGM API to:

- List modules bound into a service program
- List service programs bound to a service program
- List data items exported to the activation group
- List data item imports that are resolved by weak exports that were exported to the activation group
- List copyrights of a service program
- List procedure export information of a service program
- List data export information of a service program
- List signatures of a service program

Authorities and Locks

User Space Authority
*CHANGE

User Space Library Authority
*EXECUTE

User Space Lock
*EXCLRD

Service Program Authority for SPGL0100 and SPGL0110 Formats
*USE

Service Program Authority for other Formats
*READ

Service Program Library Authority
*EXECUTE

Service Program Lock
*SHRRD

Required Parameter Group

Qualified user space name
INPUT; CHAR(20)

The user space that is to receive the service program information. The first 10 characters contain the user space name. The second 10 characters contain the name of the library where the user space is located. The library name can be a specific library name or one of these special values:

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

Format name
INPUT; CHAR(8)

The content and format of the information to be returned about the specified service program(s). One of the following format names may be used:

"SPGL0100 Format" on page 92	Service program module (*MODULE) information.

"SPGL0110 Format" on page 94	Service program module (*MODULE) information. **Note:** Do not use the generic header entry size for format SPGL0110. Use the Size of this entry field returned in this format for the size of each entry.
"SPGL0200 Format" on page 96	Service program (*SRVPGM) information.
"SPGL0300 Format" on page 96	Data items exported to the activation group (*ACTGRPEXP).
"SPGL0400 Format" on page 96	Data item imports resolved by weak exports that were exported to the activation group (*ACTGRPIMP).
"SPGL0500 Format" on page 97	Service program copyright (*COPYRIGHT) information.
"SPGL0600 Format" on page 97	Service program procedure export (*PROCEXP) information.
"SPGL0610 Format" on page 97	Service program long procedure export name (*PROCEXP) information. **Note:** Do not use the generic header entry size for format SPGL0610. Use the Size of this entry field returned in this format for the size of each entry.
"SPGL0700 Format" on page 98	Service program data export (*DTAEXP) information.
"SPGL0800 Format" on page 98	Service program signature (*SIGNATURE) information.

Qualified service program name

INPUT; CHAR(20)

The name of the service program for which the information is to be listed. The first 10 characters contain the service program name. The second 10 characters contain the name of the library where the service program is located.

The service program name can be a specific service program name or one of the following special values:

**ALL*	All service programs
*generic**	All service programs that begin with this generic prefix. For example, WRK* lists information for all service programs that begin with WRK to which you are authorized.

The library name can be a specific library name or one of these special values:

**ALL*	All libraries in the system
**ALLUSR*	All non-system libraries. For information on the libraries included, see *ALLUSR in Generic library names.
**CURLIB*	The job's current library
**LIBL*	The library list
**USRLIBL*	Libraries listed in the user portion of the library list

Error code

I/O; CHAR(*)

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

Format of the Generated List

The user space contains:

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

- A header section
- A list data section

For details about the user area and generic header, see User Space Format for List APIs. For descriptions of each field in the list returned, see "Field Descriptions" on page 98.

Input Parameter Section

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	User space name specified
10	A	CHAR(10)	User space library name specified
20	14	CHAR(8)	Format name specified
28	1C	CHAR(10)	Service program name specified
38	26	CHAR(10)	Service program library name specified

Header Section

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	User space name used
10	A	CHAR(10)	User space library name used
20	14	BINARY(4)	Reason code

SPGL0100 Format

The SPGL0100 format includes information on all the modules that are bound into the programs specified. The modules are listed in the user space in the order the modules are bound into the program. You must have a service program authority of *USE to use this format. The following table shows how this information for each module is organized. For detailed descriptions of the fields in the list, see "Field Descriptions" on page 98.

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	Service program name
10	A	CHAR(10)	Service program library name
20	14	CHAR(10)	Bound module name
30	1E	CHAR(10)	Bound module library name
40	28	CHAR(10)	Source file name
50	32	CHAR(10)	Source file library name
60	3C	CHAR(10)	Source file member name
70	46	CHAR(10)	Module attribute
80	50	CHAR(13)	Module creation date and time
93	5D	CHAR(13)	Source file updated date and time
106	6A	CHAR(10)	Sort sequence table name
116	74	CHAR(10)	Sort sequence table library name

Offset		Type	Field
Dec	Hex		
126	7E	CHAR(10)	Language identifier
136	88	BINARY(4)	Optimization level
140	8C	BINARY(4)	Maximum optimization level
144	90	CHAR(10)	Debug data
154	9A	CHAR(6)	Release module created on
160	A0	CHAR(6)	Release module created for
166	A6	CHAR(20)	Reserved
186	BA	CHAR(1)	User-modified
187	BB	CHAR(13)	Licensed program
200	C8	CHAR(5)	PTF number
205	CD	CHAR(6)	APAR ID
211	D3	CHAR(1)	Creation data
212	D4	BINARY(4)	Module CCSID
216	D8	CHAR(8)	Object control level
224	E0	CHAR(1)	Enable performance collection
225	E1	CHAR(10)	Profiling data
235	EB	CHAR(1)	Reserved
236	EC	BINARY(4)	Number of procedures
240	F0	BINARY(4)	Number of procedures block reordered
244	F4	BINARY(4)	Number of procedures block-order measured
248	F8	CHAR(1)	Teraspace storage enabled
249	F9	CHAR(1)	Storage model
250	FA	CHAR(74)	Reserved
324	144	BINARY(4)	Number of SQL statements
328	148	CHAR(18)	Relational database
346	15A	CHAR(10)	Commitment control
356	164	CHAR(10)	Allow copy of data
366	16E	CHAR(10)	Close SQL cursor
376	178	CHAR(10)	Naming convention
386	182	CHAR(10)	Date format
396	18C	CHAR(1)	Date separator
397	18D	CHAR(10)	Time format
407	197	CHAR(1)	Time separator
408	198	CHAR(10)	Delay PREPARE
418	1A2	CHAR(10)	Allow blocking
428	1AC	CHAR(10)	Default collection name
438	1B6	CHAR(10)	SQL package name
448	1C0	CHAR(10)	SQL package library name
458	1CA	CHAR(10)	Dynamic user profile
468	1D4	CHAR(10)	SQL sort sequence table name

Offset		Type	Field
Dec	Hex		
478	1DE	CHAR(10)	SQL sort sequence table library name
488	1E8	CHAR(10)	SQL language identifier
498	1F2	CHAR(10)	Connection method
508	1FC	BINARY(4)	SQL path length
512	200	CHAR(3483)	SQL path

SPGL0110 Format

The SPGL0110 format includes information on all the modules that are bound into the programs specified. The modules are listed in the user space in the order the modules are bound into the program. You must have a service program authority of *USE to use this format. The following table shows how this information for each module is organized. For detailed descriptions of the fields in the list, see "Field Descriptions" on page 98.

Note: Do not use the generic header entry size for this format. Use the Size of this entry field returned in this format for the size of each entry.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Size of this entry
4	4	CHAR(10)	Service program name
14	E	CHAR(10)	Service program library name
24	18	CHAR(10)	Bound module name
34	22	CHAR(10)	Bound module library name
44	2C	CHAR(10)	Source file name
54	36	CHAR(10)	Source file library name
64	40	CHAR(10)	Source file member name
74	4A	CHAR(10)	Module attribute
84	54	CHAR(13)	Module creation date and time
97	61	CHAR(13)	Source file updated date and time
110	6E	CHAR(10)	Sort sequence table name
120	78	CHAR(10)	Sort sequence table library name
130	82	CHAR(10)	Language identifier
140	8C	BINARY(4)	Optimization level
144	90	BINARY(4)	Maximum optimization level
148	94	CHAR(10)	Debug data
158	9E	CHAR(6)	Release module created on
164	A4	CHAR(6)	Release module created for
170	AA	CHAR(20)	Reserved
190	BE	CHAR(1)	User-modified
191	BF	CHAR(13)	Licensed program
204	CC	CHAR(5)	PTF number

Offset		Type	Field
Dec	Hex		
209	D1	CHAR(6)	APAR ID
215	D7	CHAR(1)	Creation data
216	D8	BINARY(4)	Module CCSID
220	DC	CHAR(8)	Object control level
228	E4	CHAR(1)	Enable performance collection
229	E5	CHAR(10)	Profiling data
239	EF	CHAR(1)	Reserved
240	F0	BINARY(4)	Number of procedures
244	F4	BINARY(4)	Number of procedures block reordered
248	F8	BINARY(4)	Number of procedures block-order measured
252	FC	CHAR(1)	Teraspace storage enabled
253	FD	CHAR(1)	Storage model
254	FE	CHAR(2)	Reserved
256	100	BINARY(4)	Offset to Licensed Internal Code options
260	104	BINARY(4)	Length of Licensed Internal Code options
264	108	CHAR(64)	Reserved
328	148	BINARY(4)	Number of SQL statements
332	14C	CHAR(18)	Relational database
350	15E	CHAR(10)	Commitment control
360	168	CHAR(10)	Allow copy of data
370	172	CHAR(10)	Close SQL cursor
380	17C	CHAR(10)	Naming convention
390	186	CHAR(10)	Date format
400	190	CHAR(1)	Date separator
401	191	CHAR(10)	Time format
411	19B	CHAR(1)	Time separator
412	19C	CHAR(10)	Delay PREPARE
422	1A6	CHAR(10)	Allow blocking
432	1B0	CHAR(10)	Default collection name
442	1BA	CHAR(10)	SQL package name
452	1C4	CHAR(10)	SQL package library name
462	1CE	CHAR(10)	Dynamic user profile
472	1D8	CHAR(10)	SQL sort sequence table name
482	1E2	CHAR(10)	SQL sort sequence table library name
492	1EC	CHAR(10)	SQL language identifier
502	1F6	CHAR(10)	Connection method
512	200	BINARY(4)	SQL path length
516	204	CHAR(3483)	SQL path
1074	432	CHAR(*)	Reserved
Bound module information through offsets			

Offset		Type	Field
Dec	Hex		
		CHAR(*)	Licensed Internal Code options

SPGL0200 Format

The SPGL0200 format includes information on all the service programs that are bound to the programs specified. The following table shows how this information is organized. For detailed descriptions of the fields in the list, see "Field Descriptions" on page 98.

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	Service program name
10	A	CHAR(10)	Service program library name
20	14	CHAR(10)	Bound service program name
30	1E	CHAR(10)	Bound service program library name
40	28	CHAR(16)	Bound service program signature

SPGL0300 Format

The SPGL0300 format lists data items exported to the activation group. The list data items are specified in the data export entry in the binding specifications component when the module was created. The following table shows how this information is organized. For detailed descriptions of the fields in the list, see "Field Descriptions" on page 98.

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	Service program name
10	A	CHAR(10)	Service program library name
20	14	BINARY(4)	Size of data item export
24	18	BINARY(4)	Data item export name CCSID
28	1C	BINARY(4)	Data item export name length
32	20	CHAR(256)	Data item export name

SPGL0400 Format

The SPGL0400 format lists data item imports that were resolved by weak exports that had been exported to an activation group. The following table shows how this information is organized. For detailed descriptions of the fields in the list, see "Field Descriptions" on page 98.

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	Service program name
10	A	CHAR(10)	Service program library name
20	14	BINARY(4)	Data item import name CCSID

Offset		Type	Field
Dec	Hex		
24	18	BINARY(4)	Data item import name length
28	1C	CHAR(256)	Data item import name

SPGL0500 Format

The SPGL0500 format includes copyright information for the service programs specified. The following table shows how this information is organized. For detailed descriptions of the fields in the list, see "Field Descriptions" on page 98.

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	Service program name
10	A	CHAR(10)	Service program library name
20	14	CHAR(4)	Reserved
24	18	BINARY(4)	Copyright length
28	1C	CHAR(256)	Copyright

SPGL0600 Format

The SPGL0600 format includes procedure export information for the service programs specified. The following table shows how this information is organized.

Note: Check the subsetted list indicator in the generic header to determine if all the information that was available was returned. If the subsetted list indicator indicated there was data available that could not be returned, check the reason code in the header section for further details. For detailed descriptions of the fields in the list, see "Field Descriptions" on page 98.

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	Service program name
10	A	CHAR(10)	Service program library name
20	14	BINARY(4)	Procedure export CCSID
24	18	BINARY(4)	Procedure export name length
28	1C	CHAR(256)	Procedure export name
284	11C	CHAR(10)	Uses argument optimization (ARGOPT)

SPGL0610 Format

The following information is returned for the SPGL0610 format. All procedure export names available are returned in the SPGL0610 format, regardless of the size of the name. For detailed descriptions of the fields in the table, see .

Note: Do not use the generic header entry size for this format. Use the Size of this entry field returned in this format for the size of each entry.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Size of this entry
4	4	CHAR(10)	Service program name
14	E	CHAR(10)	Service program library name
24	18	BINARY(4)	Long procedure export CCSID
28	1C	BINARY(4)	Offset to long procedure export name
32	20	BINARY(4)	Length of long procedure export name
36	24	CHAR(10)	Uses argument optimization (ARGOPT)
46	2E	CHAR(*)	Reserved
Service program information through offsets			
		CHAR(*)	Long procedure export name

SPGL0700 Format

The SPGL0700 format includes data export information for the service programs specified. The following table shows how this information is organized. For detailed descriptions of the fields in the list, see "Field Descriptions."

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	Service program name
10	A	CHAR(10)	Service program library name
20	14	BINARY(4)	Data item CCSID
24	18	BINARY(4)	Data item name length
28	1C	CHAR(256)	Data item name

SPGL0800 Format

The SPGL0800 format includes signature information for the service programs specified. The following table shows how this information is organized. For detailed descriptions of the fields in the list, see "Field Descriptions."

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	Service program name
10	A	CHAR(10)	Service program library name
20	14	CHAR(16)	Signature

Field Descriptions

Allow blocking. Whether blocking is used to improve the performance of certain SQL statements. The possible values are:

**NONE* Blocking is not used.

**READ*	Blocking is used for read-only data cursors when running COMMIT(*NONE) and there are no EXECUTE or EXECUTE IMMEDIATE statements.
**ALLREAD*	Blocking is used for all read-only cursors when running COMMIT(*NONE) or COMMIT(*CHG).
Blank	The module does not contain SQL statements.

Allow copy of data. Whether a copy of the data can be used in the implementation of an SQL query. The possible values are:

**NO*	A copy of the data cannot be used.
**YES*	A copy of the data can be used when needed.
**OPTIMIZE*	The system determines whether a copy of the data is used for optimal performance.
Blank	The module does not contain SQL statements.

APAR ID. The module was changed as the result of the authorized program analysis report (APAR) with this identification number. This is blank if the module was not changed at bind time.

Bound module library name. The name of the library containing the module bound into this service program at bind time.

Bound module name. The name of the module bound into this service program. This is a copy of the module that was bound into this service program. It is not the *MODULE object on the system.

Bound service program library name. The name of the library containing the service program bound to this service program at bind time. This is the library name in which the activation expects to find the service program at run time. Hexadecimal zeros indicate the library list is used at the time the service program is needed.

Bound service program name. The name of the service program bound to this service program.

Bound service program signature. The current signature of the service program at the time the service program was bound to this service program.

Close SQL cursor. Specifies when SQL cursors are implicitly closed and SQL-prepared statements are implicitly discarded. The possible values are:

**ENDMOD*	When the module ends.
**ENDACTGRP*	When the activation group is deleted.
Blank	The module does not contain SQL statements.

Commitment control. The level of commitment control that was specified on the SQL precompile. The possible values are:

**NONE*	No commitment control was specified on the SQL precompile. Uncommitted changes in other jobs can be seen.
**CHG*	Objects referred to in SQL COMMENT ON, CREATE, DROP, GRANT, LABEL ON, and REVOKE statements are locked until the end of the unit of work (transaction). Updated, deleted, and inserted rows (records) are locked until the end of the unit of work. Uncommitted changes in other jobs can be seen.
**CS*	Objects referred to in SQL COMMENT ON, CREATE, DROP, GRANT, LABEL ON, and REVOKE statements are locked until the end of the unit of work (transaction). Updated, deleted, and inserted rows (records) are locked until the end of the unit of work. A row (record) that is selected but not updated is locked until the next row (record) is selected. Uncommitted changes in other jobs cannot be seen.

**ALL*	Objects referred to in SQL COMMENT ON, CREATE, DROP, GRANT, LABEL ON, and REVOKE statements are locked until the end of the unit of work (transaction). All rows selected, updated, deleted, and inserted are locked until the end of the unit of work. Uncommitted changes in other jobs cannot be seen.
Blank	The module does not contain SQL statements.

Connection method. The method used for establishing remote connections when running distributed service programs.

Special values that can be returned are:

**RUW*	Only one connection to a relational database is allowed. Consecutive CONNECT statements result in the previous connection being disconnected before a new connection is established.
**DUW*	Connections to several relational databases are allowed. Consecutive CONNECT statements to additional relational databases do not result in disconnection of previous connects. SET CONNECTION can be used to switch between connections. Read-only connections may result.
Blank	The module does not contain SQL statements.

Copyright. The copyright string included in this service program.

Creation data. Whether the bound module has all the creation data and if that data is observable or unobservable. The possible values are:

0	*NO. Not all the creation data is present in the bound module.
1	*YES. The creation data is present in the bound module and all of that data is observable.
2	*UNOBS. The creation data is present in the bound module but not all of that data is observable.

Copyright length. The length of the copyright string.

Data item CCSID. The coded character set identifier (CCSID) of this data item.

Data item export name. Data items that are exported to an activation group. These data items can be used outside of the module or service program that they are defined in.

Data item export name CCSID. The coded character set identifier (CCSID) for the name of this data item export.

Data item export name length. The length of the name of the data export item.

Data item import name. The name of the data item imports that were resolved by weak exports that had been exported to the activation group.

Data item import name CCSID. The coded character set identifier (CCSID) for the name of this data item import.

Data item import name length. The length of the name of the data import item.

Data item name. Service program data items that are allowed to be exported.

Data item name length. The length of the data item name.

Date format. The format used when accessing date-result columns through SQL. All output date fields are returned in this format. For input date strings, the value you specify is used to determine whether the date is a valid format. The values returned are:

**USA*	USA format (mm/dd/yyyy).
**ISO*	International Standards Organization format (yyyy-mm-dd).
**EUR*	European format (dd.mm.yyyy).
**JIS*	Japanese Industrial Standard Christian Era (yyyy-mm-dd).
**MDY*	Month/day/year format (mm/dd/yy).
**DMY*	Day/month/year format (dd/mm/yy).
**YMD*	Year/month/day format (yy/mm/dd).
**JUL*	Julian format (a numeric value from 1 to 365).
Blank	The module does not contain SQL statements.

Date separator. The separator used when accessing date-result columns. This information is blank if the module does not contain SQL statements. However, the number of SQL statements field should be checked to determine if the module contains SQL statements. This is because a blank may be specified as a separator value.

Debug data. Whether debug data was generated when this module was created. If debug data exists, the module may be debugged using the source debugger. The possible values are:

**YES*	Debug data was generated.
**NO*	Debug data was not generated.

Default collection name. The collection name used for the unqualified names of tables, views, indexes, and SQL packages. Possible values are:

**NONE*	There is no default collection name.
Blank	The module does not contain SQL statements.

Delay PREPARE. Whether SQL prepare processing can be delayed until the statement is actually used. The possible values are:

**YES*	Prepare processing can be delayed.
**NO*	Prepare processing cannot be delayed.
Blank	The module does not contain SQL statements.

Dynamic user profile. The user profile used for dynamic SQL statements. The following special values can be returned:

**USER*	Local dynamic SQL statements are run under the profile of the module's user. Distributed dynamic SQL statements are run under the profile of the SQL package's user.
**OWNER*	Local dynamic SQL statements are run under the profile of the module's owner. Distributed dynamic SQL statements are run under the profile of the SQL package's owner.
Blank	The module does not contain SQL statements.

Enable performance collection. The level of performance collection enabled for this module. The following values can be returned:

*'00'X *NONE or '10'X *PEP*	This gives the entry/exit information for the PEP only. No entry/exit hooks in the module's internal procedures and no precall or postcall hooks around calls to other procedures are included. **Note:** If *NONE is shown and the module was created or re-created on an iSeries server running Version 3 Release 6 Modification 0 prior to the installation of PTF MF11968, the module will not have any performance collection enabled. To enable performance collection, use one of the following commands and specify ENBFPRCOL(*PEP): • Change Module (CHGMOD) • Change Program (CHGPGM) • Change Service Program (CHGSRVPGM)
*'50'X *ENTRYEXIT *NONLEAF*	This gives the entry/exit information on all of the non-leaf procedures in the module. This includes the PEP routine. This is useful to capture information on most routines but not at the expense of destroying the 'leaf-ness' of the leaf procedures.
*'70'X *ENTRYEXIT *ALLPRC*	This gives the entry/exit information on all the procedures of the module (including those that were leaf procedures). This includes the PEP routine. This is useful to capture information on all procedures.
*'D0'X *FULL *NONLEAF*	This gives the entry/exit information on all the procedures of the module that are not leaf procedures. This includes the PEP routine. Precall and postcall hooks around calls to external procedures are also included.
*'F0'X *FULL *ALLPRC*	This gives the entry/exit information on all procedures of the module (including those that were leaf procedures). This includes the PEP routine. Precall and postcall hooks around calls to external procedures are also included. This is useful to capture information on all procedures.

Format name specified. The format name that was passed to this API on the call in the format parameter.

Language identifier. Returns the 3-character language identifier used when the module was compiled. The following special values can also be returned:

**JOBRUN*	The language identifier associated with the job at the time the service program into which the module is bound runs.
Blank	The module does not contain any language identification information.

Length of Licensed Internal Code options. The size, in two-byte characters, of the Licensed Internal Code options string. This will be 0 if no Licensed Internal Code options were used for this module.

Length of long procedure export name. The actual size, in bytes, of the long procedure export name for this entry.

Licensed Internal Code options. The Licensed Internal Code options that are in use by the module. This field is specified in UCS-2 (CCSID 13488).

Licensed program. If the module was part of a licensed program at bind time, this field shows the product number and the level of the licensed program. This is blank if the module is not part of a licensed program at bind time.

Long procedure export CCSID. The coded character set identifier (CCSID) of this procedure export name.

Long procedure export name. Service program procedures that are allowed to be exported.

Maximum optimization level. The highest level of optimization this module could have at bind time. If observability has been removed from the module, this maximum optimization level value might not be the same as the one specified at module creation. The possible values are:

65535	The module is not restricted to a maximum optimization level. It can be retranslated to any of the supported optimization levels. 65535 is also known as *NOMAX.
40	Maximum level of optimization. This level includes all the optimizations performed at optimization level 30. In addition, it includes optimization that disables call and instruction tracing. Thus, tracing of modules created at this optimization level cannot be done.
30	More optimization is performed in addition to those performed at optimization level 20. Variables cannot be changed but can be displayed while the program is being debugged. However, the displayed value of the variable during debugging may not be its actual value. 30 is also known as *FULL.
20	Some optimization is performed on the generated code. When the module optimized at this level is being debugged, the variables can be displayed but not changed. This level improves the performance of the module slightly over level 10. 20 is also known as *BASIC.
10	No additional optimization is performed on the generated code. Variables can be displayed and changed when the program is being debugged. With no optimization of the code, this value provides the lowest level of module performance. 10 is also known as *NONE.

Module attribute. The language in which the module is written. This field can be blank (for example, a module created by a compilation process internal to IBM).

Module CCSID. The coded character set identifier (CCSID) for this module.

Module creation date and time. The date and time the module was created. The creation date and time field is in the CYYMMDDHHMMSS format as follows:

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

Naming convention. The convention used for naming objects in SQL statements. The possible values are:

**SQL*	The SQL naming convention is used.
**SYS*	The system naming convention is used.
Blank	The module does not contain SQL statements.

Number of procedures. The number of procedures defined in the module. This number includes the program entry procedure (PEP), if one was generated by the compiler for this module.

Number of procedures block-order measured. The number of procedures defined in the module that had block-order profiling data collected at the time block-order profiling data was applied. If the module does not have block-order profiling data applied, this value will be zero.

Number of procedures block reordered. The number of procedures defined in the module that are block reordered. If the module does not have block-order profiling data applied, this value will be zero. This value can decrease if the service program in which this bound module is contained is retranslated.

Number of SQL statements. The number of SQL statements contained in the module. This value is zero if the module does not contain SQL statements.

Object control level. The object control level for the module at the time it was bound into this service program. You can compare the object control level of a module to the object control level of a listing to make sure they match.

Offset to Licensed Internal Code options. The offset from the beginning of the user space where the Licensed Internal Code options begin. This will be 0 if no Licensed Internal Code options were used for this module.

Offset to long procedure export name. The offset from the beginning of the user space where this procedure export name is stored.

Optimization level. Optimization levels cause the translator to produce machine code that reduces the amount of system resources necessary to run the program. The more optimization, the more efficiently the module runs on the system. Also, with more optimization you may not be able to change or view variables that have been optimized. The possible values are:

65535	The module is not restricted to a maximum optimization level. It can be retranslated to any of the supported optimization levels. 65535 is also known as *NOMAX.
40	Maximum level of optimization. This level includes all the optimizations performed at optimization level 30. In addition, it includes optimization that disables call and instruction tracing. Thus, tracing of modules created at this optimization level cannot be done.
30	More optimization is performed in addition to those performed at optimization level 20. Variables cannot be changed but can be displayed while the program is being debugged. However, the displayed value of the variable during debugging may not be its actual value. 30 is also known as *FULL.
20	Some optimization is performed on the generated code. When the module optimized at this level is being debugged, the variables can be displayed but not changed. This level improves the performance of the module slightly over level 10. 20 is also known as *BASIC.
10	No additional optimization is performed on the generated code. Variables can be displayed and changed when the program is being debugged. With no optimization of the code, this value provides the lowest level of module performance. 10 is also known as *NONE.

Procedure export name. Service program procedures that are allowed to be exported.

Procedure export CCSID. The coded character set identifier (CCSID) of this procedure name export.

Procedure export name length. The length of the procedure export name.

Profiling data. The profiling data attribute for the module that is bound into this service program. Possible values follow:

**NOCOL*	The collection of profiling data is not enabled and block-order profiling data is not applied to the module bound into this service program.
**COL*	The collection of profiling data is enabled. Any block-order profiling data that was applied has been removed for the module bound into this service program.
**APYBLKORD*	Block-order profiling data is applied to the module that is bound into this service program. See the number of procedures block reordered field for the current number of procedures in this module that are block reordered.

PTF number. The program temporary fix (PTF) that resulted in the creation of the module. This field is blank for user-created modules.

Reason code. The reason code describing why the returned list is only a subset. The following values can be returned:

0000	The list returned in the user space contains all information meeting the search criteria.
0001	Additional procedure exports were found that meet the search criteria but could not be included in the returned list. The requested format could not handle procedure names greater than 256 characters. Call the API again specifying the SPGL0610 format to get all the available information.

Relational database. The default relational database that was specified on the SQL precompile. A nonblank value other than *LOCAL specifies the name of the relational database to be resolved through the relational database directory. The following special values can be returned:

**LOCAL*	The module can only access data on the local system.
Blank	The module does not contain SQL statements.

Release module created for. The version, release, and modification level of the operating system for which the module was created. The field has a VvRrMm format, where:

Vv	The character V is followed by a 1-character version number.
Rr	The character R is followed by a 1-character release level.
Mm	The character M is followed by a 1-character modification level.

Release module created on. The version, release, and modification level of the operating system on which the module was created. The field has a VvRrMm format, where:

Vv	The character V is followed by a 1-character version number.
Rr	The character R is followed by a 1-character release level.
Mm	The character M is followed by a 1-character modification level.

Reserved. An ignored field.

Service program library name. The name of the library containing the service program.

Service program library name specified. The service program library name that was passed to this API on the call in the qualified service program name parameter.

Service program name. The name of the service program.

Service program name specified. The service program name that was passed to this API on the call in the qualified service program name parameter.

Signature. A valid signature of this service program.

Size of data item export. The size, in bytes, of the data item export.

Size of this entry. The size, in bytes, of this entry.

Sort sequence table library name. The name of the library that contained the sort sequence table used when the module was compiled. This does not apply to SQL statements in the module. This information is blank if the module does not contain any sort sequence information or a special value was returned for the sort sequence table name. The following special values can be returned:

**LIBL*	The sort sequence table is found in the library list when the service program runs this module.
**CURLIB*	The sort sequence table is found in the current library when the service program runs this module.

Sort sequence table name. The name of the sort sequence table used when the module was compiled. This does not apply to SQL statements in the module. The following special values can be returned:

**HEX*	No sort sequence is used.
**JOBRUN*	The sort sequence value associated with the job at the time the service program runs this module is used.

*LANGIDSHR The shared sort sequence for the language identifier is used.
*LANGIDUNQ The unique sort sequence for the language identifier is used.

Note: This sort sequence table does not apply to SQL statements.

Source file library name. The name of the library that contains the source file used to create the module. The field is blank if no source file was used to create the module.

Source file member name. The name of the member in the source file. The field is blank if no source file was used to create the module.

Source file name. The name of the source file used to create the module. The field is blank if no source file was used to create the module.

Source file updated date and time. The date and time the member in the source file was last updated. The field is in the same format as the module creation date and time field. The field is blank if no source file was used to create the module.

SQL language identifier. The 3-character language identifier used when the module was compiled. This information is blank if the module does not contain any language identification information. The following possible special value can be returned:

*JOBRUN The language identifier is the LANGID associated with the job at the time the module is run.

SQL package library name. The name of the library the SQL package is in.

SQL package name. The name of the SQL package created on the relational database specified on the RDB parameter of the command that created this module. Possible values are:

*NONE This is not a distributed module.
Blank The module does not contain SQL statements.

SQL path. The list of libraries used during resolution of functions and data types within SQL statements. The list is in the form of repeating library names, each surrounded by double quotes and separated by commas. Even though 3483 bytes are reserved, the path's length is determined by the SQL path length entry.

SQL path length. The length, in bytes, of the SQL path.

SQL sort sequence table library name. The name of the library that is used to locate the SQL sort sequence table. This information is blank if the module does not contain any SQL sort sequence information or a special value was returned for the SQL sort sequence table name. The following special values can be returned:

*LIBL The SQL sort sequence table is found by looking in the library list.
*CURLIB The SQL sort sequence table is found by looking in the current library.

SQL sort sequence table name. The sort sequence table name used when the module was compiled. This information is blank if the module does not contain any SQL sort sequence information. The following special values can be returned:

*HEX No SQL sort sequence is used for the SQL statements.

**JOBRUN*	The SQL sort sequence is the SRTSEQ value associated with the job at the time the SQL statements within the module are run.
**LANGIDSHR*	The shared SQL sort sequence for the language identifier (LANGID) is used for the SQL statements.
**LANGIDUNQ*	The unique SQL sort sequence for the language identifier (LANGID) is used for the SQL statements.

Storage model. Where the automatic and static storage for this bound module is allocated at run time. The following values can be returned:

0 *SNGLVL	Automatic and static storage are allocated from single-level storage.
1 *TERASPACE	Automatic and static storage are allocated from teraspace.
2 *INHERIT	Automatic and static storage are allocated from either single-level storage or teraspace, depending on the activation.

Teraspace storage enabled. The teraspace storage capability for this bound module. Possible values are:

*'00'X *NO*	The module bound to this service program is not teraspace storage enabled.
*'80'X *YES*	The module bound to this service program is teraspace storage enabled.

Time format. The format used when accessing time-result columns through SQL. All output time fields are returned in this format. The values returned are:

**USA*	USA format (hh:mm a.m. or p.m.).
**ISO*	International Standards Organization format (hh.mm.ss).
**EUR*	European format (hh.mm.ss).
**JIS*	Japanese Industrial Standard Christian Era (hh.mm.ss).
**HMS*	Hours/minutes/seconds format (hh:mm:ss).
Blank	The module does not contain SQL statements.

Time separator. The separator used when accessing time-result columns. This information is blank if the module does not contain SQL statements. However, the number of SQL statements field should be checked to determine if the module contains SQL statements. This is because a blank may be specified as a separator value.

User-modified. Whether the module was changed by the user. The possible values are:

0	The user did not change the module.
1	The user changed the module.

User space library name specified. The user space library name that was passed to this API on the call in the qualified user space name parameter.

User space library name used. The name of the library that contains the user space that receives the service program information requested.

User space name specified. The user space name that was passed to this API on the call in the qualified user space name parameter.

User space name used. The name of the user space that receives the service program information requested.

Uses argument optimization (ARGOPT). Whether or not the service program export uses argument optimization. The possible values are:

**YES* The service program export uses argument optimization.
**NO* The service program export does not use argument optimization.

Error Messages

Message ID	Error Message Text
CPF24B4 E	Severe error while addressing parameter list.
CPF3C20 E	Error found by program &1.
CPF3C21 E	Format name &1 is not valid.
CPF3C90 E	Literal value cannot be changed.
CPF3CAA E	List is too large for user space &1.
CPF3CF1 E	Error code parameter not valid.
CPF5CF6 E	Program name &1 not valid special value.
CPF811A E	User space &4 in &9 damaged.
CPF9570 E	Error occurred creating or accessing debug data.
CPF9801 E	Object &2 in library &3 not found.
CPF9802 E	Not authorized to object &2 in &3.
CPF9803 E	Cannot allocate object &2 in library &3.
CPF9804 E	Object &2 in library &3 damaged.
CPF9806 E	Cannot perform function for object &2 in library &3.
CPF9807 E	One or more libraries in library list deleted.
CPF9808 E	Cannot allocate one or more libraries on library list.
CPF9810 E	Library &1 not found.
CPF9820 E	Not authorized to use library &1.
CPF9830 E	Cannot assign library &1.
CPF9838 E	User profile storage limit exceeded.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

API introduced: V2R3

Top | "Program and CL Command APIs," on page 1 | APIs by category

---

Process Commands (QCAPCMD) API

Required Parameter Group:

1	Source command string	Input	Char(*)
2	Length of source command string	Input	Binary(4)
3	Options control block	Input	Char(*)
4	Options control block length	Input	Binary(4)
5	Options control block format	Input	Char(8)
6	Changed command string	Output	Char(*)
7	Length available for changed command string	Input	Binary(4)
8	Length of changed command string available to return	Output	Binary(4)
9	Error Code	I/O	Char(*)

Default Public Authority: *EXCLUDE
Threadsafe: Yes. See "Usage Notes" on page 112 for command considerations.

The Process Commands (QCAPCMD) API is used to perform command analyzer processing on command strings. You can check or run CL commands from HLLs as well as check syntax for specific source definition types.

You can use the QCAPCMD API to:

- Check the syntax of a command string prior to running it
- Prompt the command and receive the changed command string
- Run a command from an HLL

Authorities and Locks

Command
*USE

Command library
*EXCLUDE

Required Parameter Group

Source command string
INPUT; CHAR(*)

The command string to be prompted for or run.

Length of source command string
INPUT; BINARY(4)

The length of the source command string. Valid values are between 1 and 32 702. Message CPF3C1D will result for values outside this range. This length can include trailing blanks.

Options control block
INPUT; CHAR(*)

The options that control the handling of the command string. The layout of this parameter is the "CPOP0100 Format" on page 110.

Options control block length
INPUT; BINARY(4)

The length of the options control block. A minimum length of 20 is required for the CPOP0100 format.

Options control block format
INPUT; CHAR(8)

The format of the options control block. CPOP0100 is the only valid value. For more information, see "CPOP0100 Format" on page 110.

Changed command string
OUTPUT; CHAR(*)

The rebuilt command string. This is the updated command string, which includes changes from prompting, ordering of parameters, and the addition of keywords. This string may be substantially longer than the source command string. If an error occurs that prevents the command string from being rebuilt, this field is not changed. No padding is performed on the value returned. The length of changed command string available to return parameter should be used to determine how much data is returned.

Length available for changed command string
INPUT; BINARY(4)

The length available to return the updated command string. If an updated command string is not desired, a special value of 0 may be specified. This value must be a positive number or zero.

Length of changed command string available to return
OUTPUT; BINARY(4)

The length of the changed command string returned or available to return. If zero is specified for the length available for changed command string parameter, this value is not set. If an error occurs that prevents the command string from being rebuilt, this field is zero. If the changed command string parameter is not large enough to hold the entire rebuilt command string, this value is the total length available. The changed command string is truncated.

Error code

I/O; CHAR(*)

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

CPOP0100 Format

The CPOP0100 format includes information on the contents of the options control block parameter. The following table shows how this information is organized. For detailed descriptions of the fields in the list, see "Field Descriptions."

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Type of command processing
4	4	CHAR(1)	DBCS data handling
5	5	CHAR(1)	Prompter action
6	6	CHAR(1)	Command string syntax
7	7	CHAR(4)	Message retrieve key
» 11	4	BINARY(4)	CCSID of command string «
15	B	CHAR(5)	Reserved

Field Descriptions

»

CCSID of command string. CCSID of the command string. The possible values are:

0 Input is in the job CCSID.
1208 The command string is in UTF8.
1200 The command string is in UTF16.

«

Command string syntax. Whether command processing should be done in i5/OS mode or System/38 mode. The possible values are:

0 Use system syntax. The specification of qualified objects is in the format library/object.
1 Use System/38 syntax. The specification of qualified objects is in the format object.library. The system searches the QUSER38 library (if it exists) and the QSYS38 library for the command even though these libraries are not in the library list.

DBCS data handling. Whether the command analyzer should handle the SO/SI characters as DBCS delimiters. It is valid with all classes of command processing. This option is the equivalent of specifying the IGC process control parameter on the Execute Command (QCMDEXC) and Check Command Syntax (QCMDCHK) APIs. The possible values are:

0 Ignore DBCS data.

1 Handle DBCS data.

Message retrieve key. The message retrieve key identifies a request message. If prompting is requested, the request message identified by the message retrieve key is replaced by the updated command string. The updated command can then be logged in the job log. If the message key contains all hexadecimal zeros or all blanks, no request message is updated. The message key is valid for processing command types 0, 1, 2, and 3. The message key is ignored for processing command types 4, 5, 6, 7, 8, and 9. The message key is valid only during the current job.

Prompter action. Indicates whether the prompter should be called on a command string.

0 Never prompt the command. This will prevent a command prompt even if selective prompting characters are present in the command string.

Note: When the type of command processing field is 2 or 3 and there are missing required parameters, the command will be prompted, even when the prompter action is set to 0.

1 Always prompt the command. This forces a command prompt even if selective prompting characters are not present in the command string.

2 Prompt the command if selective prompting characters are present in the command string. A CPF0008 exception is sent if this value is specified with types of command processing values 4 through 8.

3 Show help. Provides help display for the command.

Reserved. This area must be set to hexadecimal zeros.

Type of command processing. The type of command processing to be performed by the system. The following processes can occur:

0 Command running. The processing for this type is the same as that performed by the QCMDEXC API. Commands processed must have a value of *EXEC on the ALLOW parameter of the Create Command (CRTCMD) or the Change Command (CHGCMD) command.

1 Command syntax check. The processing for this type is the same as that performed by the QCMDCHK API.

2 Command line running. This processing is like that provided by the QCMDEXC API but with the following additions:

- Limited user checking is performed.
- Prompting for missing required parameters is performed.
- If the System/36 environment is active and the commands are System/36 commands, the System/36 environment runs the commands.

This type processes commands with entry codes of `Job: I` (value of *INTERACT on the ALLOW parameter of the CRTCMD or CHGCMD command). While this type is meant to implement an interactive command line, it can be used in batch. When used in a batch job, the entry code for the command must be `Job: B`. Limited user checking and System/36 environment processing is done while prompting options are ignored.

3 Command line syntax check. This processing provides the check only complement of type 2 (command line running). The check option performs all checks against CL rules. The System/36 environment is not called.

4 CL program statement. The command string is checked according to the rules for CL programs (source entry utility (SEU) member type of CLP). Commands may not be run with this type. Command prompts include a prompt for a command label and comment. Variable names are allowed. Commands processed for this type must be defined with entry codes of `Pgm: B`, `Pgm: I`, or `Pgm: B,I`. They have values of *BPGM or *IPGM on the ALLOW parameter of the CRTCMD or CHGCMD command.

5 CL input stream. The command string is checked according to the rules for CL batch jobs (SEU member type of CL). Commands may not be run. Command prompts include a prompt for comment. Variable names are not allowed.

6 Command definition statements. The command string is checked according to the rules for command definition (SEU member type of CMD). Commands may not be run. The commands are restricted to CMD, PARM, ELEM, QUAL, DEP, and PMTCTL.

7 Binder definition statements. The command string is checked according to the rules for binder definition (SEU member type of BND). Commands may not be run. The commands are restricted to STRPGMEXP, ENDPGMEXP, and EXPORT.

8 User-defined option. This option allows a user to create user-defined option command strings similar to those used by the programming development manager (PDM). It allows checking and creating a command string for future use with types 0 through 3 except that variables are allowed. The command string produced may not be directly operable. That is, if CL variables were specified in the command string, the user must perform a substitution prior to using the API with types of 0 or 2.

9 ILE CL program source. The source is checked according to the rules for ILE CL programs (source entry utility (SEU) member type of CLLE). Commands may not be run with this type. Command prompts include a prompt for a command label and comment. Variable names are allowed. Commands processed for this type must be defined with entry codes of `CLLE: B`, `CLLE: I`, or `CLLE: B,I`. They have values of *IMOD or *BMOD on the ALLOW parameter of the CRTCMD or CHGCMD command.

10 Command prompt string. The command analyzer prepares the source command for prompting and returns the command string to use for the initial prompt display. If the command has an exit program registered for the QIBM_QCA_CHG_COMMAND exit point, the exit program is called. If the exit program replaces the original command, the changed command string returned by QCAPCMD is the replacement command from the exit program. The returned command string may not be syntactically correct because no syntax checking is done on the replacement command. The length of changed command string available to return is set to 0 and the changed command string parameter is not changed if any of these conditions are true:

1. The exit program is not called
2. The exit program ends in error.
3. The exit program does not replace the command.

Usage Notes

1. While this API is threadsafe, it should not be used to run a command that is not threadsafe in a job that has multiple threads. Use the Display Command (DSPCMD) command to determine whether a command is threadsafe.
2. The prompt actions controlled by the prompter option field in the option control block have the following considerations.
   - For commands with a value of 0, 1, 2, or 3 for the type of command processing field, a prompt occurs when 2 is specified for the prompter option field if:
     - Selective prompting is specified in the source command string parameter.
     - The job is running interactively.
   - If this API is called in a batch job with a valid prompt request, it is ignored. A valid prompt request is issued by specifying:
     - 1 for the prompter option field
     - 2 for the prompter option field with selective prompting characters in the command string
     - 3 for help
3. Calls of the API in batch jobs with values of 4, 5, 6, or 7 for the type of command processing field are processed. However, prompting requests are ignored.
4. The prompter option field in the options control block is ignored if 10 is specified for the type of command processing field. »
5. If the command to be executed is a proxy command, the QCAPCMD API will resolve to the target command. If the target command is also a proxy, the process repeats until either a non-proxy command is found, or the proxy chain becomes greater than the allowed maximum. Once a non-proxy command is found, the resolved command will replace the proxy command in the command string to be executed.
6. Proxy commands will be resolved before the command exit points QIBM_QCA_CHG_COMMAND and QIBM_QCA_RTV_COMMAND are called. «

Error Messages

Message ID		Error Message Text
CPF0008 E		Value in option control block not valid.
CPF24B4 E		Severe error while addressing parameter list.
CPF3C1D E		Length specified in parameter &1 not valid.
CPF3C20 E		Error found by program &1.
CPF3C90 E		Literal value cannot be changed.
CPF3CF1 E		Error code parameter not valid.
CPF3CF2 E		Error(s) occurred during running of &1 API.
CPF9872 E		Program or service program &1 in library &2 ended. Reason code &3.
xxxnnnn	E	Any escape message issued by any command may be returned. The messages listed previously are those issued by this API. Once the API has called the command analyzer, any message issued as an escape may appear.

API introduced: V2R3

Top | "Program and CL Command APIs," on page 1 | APIs by category

---

Replace Command Exit Program (QCARPLCM) API

Required Parameter Group:

1	Change command exit information	Input	Char(*)
2	Replacement command	Output	Char(*)
3	Length of replacement command string	Output	Binary(4)

Default Public Authority: *USE
Threadsafe: Yes

The Replace Command Exit Program (QCARPLCM) API may be used as the exit program for the QIBM_QCA_CHG_COMMAND for any command. If the original command was library-qualified with *SYSTEM or *NLVLIBL, the library qualifier will be replaced with *LIBL. None of the parameter values specified on the original command will be changed. See the Command Analyzer Change exit program for a description of the QIBM_QCA_CHG_COMMAND exit point.

Authorities and Locks

None

Required Parameter Group

Change command exit information
INPUT; CHAR(*)

Information about the command that the command analyzer was called to process. See "CHGC0100 Format" on page 196 in the Command Analyzer Change exit program.

Replacement command
OUTPUT; CHAR(*)

A string containing the command string that is to be substituted for the one that the command analyzer was called to process. If the original command was library-qualified with *SYSTEM or *NLVLIBL, the library qualifer will be replaced with *LIBL. None of the parameter values specified on the original command will be changed. The maximum length of the changed command string is 32000 bytes.

Length of replacement command string
OUTPUT; BINARY(4)

The length of the replacement command string (0 - 32000) in bytes.

Usage Notes

Registration Considerations

Use the Add Exit Program command (ADDEXITPGM) or API (QUSADDEP, QusAddExitProgram) to register this program as an exit program for a command. You must specify 20 bytes of exit program data. The first 10 characters specify the command name; the second 10 characters specify the library name. For example, to register QCARPLCM as the exit program to be called at the QIBM_QCA_CHG_COMMAND exit point for the Display Job (DSPJOB) command in library QSYS, specify:

```
ADDEXITPGM EXITPNT(QIBM_QCA_CHG_COMMAND)
           FORMAT(CHGC0100)
           PGMNBR(*LOW)
           PGMDTA(*JOB 20 'DSPJOB    QSYS      ')
```

If you register QCARPLCM as an exit program for a command in library QSYS, it also will be called for commands in the secondary language libraries. For example, if the exit program is registered for the DSPJOB command in library QSYS, it also will be called for the DSPJOB command in library QSYS2962.

If you rename the command or the library or move the command to another library, you also must have the exit program registered using the new command and library names.

If you register QCARPLCM as the exit program for a command, you cannot register another exit program for the command for the QIBM_QCA_CHG_COMMAND exit point.

Runtime Considerations

If two applications on the same system need to replace the same command with one of the same name but in different libraries, they can register QCARPLCM as the exit program for the QIBM_QCA_CHG_COMMAND exit point for the command. The applications must ensure that the correct application library is at the beginning of the system part of the library list.

If the original command was library-qualified with a specific library name, the exit program will not be allowed to change the command string, so the command analyzer will search only the specified library for the command.

API introduced: V4R5

Top | "Program and CL Command APIs," on page 1 | APIs by category

---

Retrieve Associated Space (QbnRetrieveAssociatedSpace) API

Required Parameter Group:

1	Qualified user space name	Input	Char(20)
2	Call level	Input	Binary(4)
3	Associated space identifier	Input	Char(10)
4	Error code	I/O	Char(*)

Default Public Authority: *USE
Service Program: QBNPREPR
Threadsafe: No

The Retrieve Associated Space (QbnRetrieveAssociatedSpace) API is used by a run-time routine to retrieve data stored with the QbnAddAssociatedSpaceEntry API. This data will be placed into the

specified user space. The format of the data is specified by the user when placed into the associated space of a module using the QbnAddAssociatedSpaceEntry API. This API should be called only by a preprocessor run-time routine.

Authorities and Locks

User Space Authority
*CHANGE

User Space Library Authority
*USE

ILE Program or Service Program Authority
*EXECUTE

Required Parameter Group

Qualified user space name
INPUT; CHAR(20)

The qualified name of the user space that is to receive the associated space data. The first 10 characters contain the user space name. It is left-justified and padded with blanks. The second 10 characters contain the name of the library where the user space is located. It is left-justified and padded with blanks. The library name can be specified with the following special values:

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

Call level
INPUT; BINARY(4)

The call level parameter identifies the location in the call stack of the ILE program or service program, which contains the associated space data.

0	The current program in the call stack identifies the associated space of the ILE program or service program.
n	The nth caller up the stack identifies the associated space of the ILE program or service program. This is a positive number.

Associated space identifier
INPUT; CHAR(10)

The associated space identifier has the following special value:

**PREPROC*	The type of data from within the associated space to copy into the user space. The special value must be left-justified and padded with blanks.

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
CPF2207 E	Not authorized to use object &1 in library &3 type *&2.
CPF24B4 E	Severe error while addressing parameter list.
CPF3C90 E	Literal value cannot be changed.
CPF3CF1 E	Error code parameter not valid.

Message ID	Error Message Text
CPF5CA2 E	&1 is not a valid associated space identifier parameter.
CPF5CA4 E	Error occurred while addressing API Parameter.
CPF5D24 E	Unexpected error occurred during preprocessor processing.
CPF811A E	User space &4 in &9 damaged.
CPF9801 E	Object &2 in library &3 not found.
CPF9802 E	Not authorized to object &2 in &3.
CPF9803 E	Cannot allocate object &2 in library &3.
CPF9807 E	One or more libraries in library list deleted.
CPF9808 E	Cannot allocate one or more libraries on library list.
CPF9810 E	Library &1 not found.
CPF9820 E	Not authorized to use library &1.
CPF9830 E	Cannot assign library &1.
CPF9838 E	User profile storage limit exceeded.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

API introduced: V3R1

Top | "Program and CL Command APIs," on page 1 | APIs by category

Retrieve Command Definition (QCDRCMDD) API

Required Parameter Group:

1	Qualified command name	Input	Char(20)
2	Destination information	Input	Char(*)
3	Destination format name	Input	Char(8)
4	Receiver variable	Output	Char(*)
5	Receiver format name	Input	Char(8)
6	Error Code	I/O	Char(*)

Default Public Authority: *USE
Threadsafe: Yes

The Retrieve Command Definition (QCDRCMDD) API retrieves information from a CL command (*CMD) object and generates XML (Extensible Markup Language) source statements which describe the command. The generated command information XML source is called Command Definition Markup Language or CDML. The CDML source can be stored in either a receiver variable or a stream file, depending on the destination format name specified.

The CDML source is stored in UTF-8. UTF-8 (CCSID 1208) is a Unicode format which resembles ASCII, but allows the data to be stored compactly and shared easily between iSeries systems and any other system which supports the UTF-8 format.

The CDML elements and attributes closely resemble the command definition statements used to create CL commands:

- CMD (Command) statement
- PARM (Parameter) statement
- ELEM (Element) statement
- QUAL (Qualifier) statement
- DEP (Dependency) statement
- PMTCTL (Prompt Control) statement

See the Document Type Definition (DTD) in /QIBM/XML/DTD/QcdCLCmd.dtd for the definition of the CDML tag language returned by this API.

If the default value for an optional command parameter has been changed using the Change Command Default (CHGCMDDFT) command, the returned command information will reflect the default currently in effect rather than the default specified when the command was created.

Additional object-level information for a command (*CMD) object can be retrieved by using the QCDRCMDI (Retrieve Command Information) API.

Authorities and Locks

API Public Authority
*USE

Command Library Authority
*EXECUTE

Command Authority
*USE

Command Lock
*SHRNUP

Output File Authority (if output stored in a stream file)
Authority to the path and file are determined by the open() API. For details, see the Authorities section of the open()—Open File API for files opened with an access mode of O_WRONLY and O_TRUNC.

Output File Lock
*SHRNUP

Required Parameter Group

Qualified command name
INPUT; CHAR(20)

The library-qualified command name for which to retrieve the command definition information. The first 10 characters contain the command name, while the second 10 characters identify the library name. The following special values are supported for the library name:

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

Destination information
INPUT; CHAR(*)

Provides information about the destination for the generated CDML source. If DEST0100 is specified for the destination format name, this parameter contains a 4-byte integer which is the size of the receiver variable (parameter 4). If DEST0200 is specified for the destination format name, this parameter contains a structure which gives the path name of the stream file where the generated CDML source is to be stored.

Destination format name
INPUT; CHAR(8)

The destination format to determine where the generated CDML source will be stored. Possible values are:

"DEST0100 Format" on page 118	Return the CDML source in the receiver variable.

"DEST0200 Format" Return the CDML source in a stream file using the path name coded in the destination information parameter.

Receiver variable
OUTPUT; CHAR(*)

The variable that is to receive the generated CDML source. The variable is used only when the destination format name is DEST0100. If the receiver variable is not large enough to hold all of the generated CDML source, no CDML source is returned.

Receiver format name
INPUT; CHAR(8)

The format of the command definition information to be returned. You must use one of the following format names:

CMDD0100 CDML source is returned that describes the CL command information needed to build a valid command string for the command.

CMDD0200 CDML source is returned that describes all of the command definition statements used to create the command. This is a superset of the information returned by receiver format CMDD0100.

Error code
I/O; CHAR(*)

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

DEST0100 Format

The following information needs to be supplied in the destination information parameter (parameter 2) for the DEST0100 format.

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Length of receiver variable

Field Descriptions

Length of receiver variable. The length of the receiver variable. If the length is larger than the size of the receiver variable, the results may not be predictable. The minimum length is 8 bytes.

DEST0200 Format

The destination information parameter (parameter 2) specifies the file path name where the generated CDML source is to be returned. See Path name format for information on specifying the output stream file path name.

Output Information Format (for CMDD0100 and CMDD0200 formats)

The following information is returned for both CMDD0100 and CMDD0200 formats.

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available

Offset		Type	Field
Dec	Hex		
8	8	CHAR(*)	Generated CDML source

Field Descriptions

Bytes available. The number of bytes of data available to be returned. All available data is returned if enough space is provided.

Bytes returned. The number of bytes of data returned.

Generated CDML source. The CDML source for the command. If the receiver variable is not large enough to hold the entire CDML source or if an unexpected error occurs while writing to the receiver variable, no data will be returned.

Usage Notes

The output file path name is represented by the 'Path name' field in the 'Path Name Format' structure when using the DEST0200 destination format. The output file path name is used to store the generated CDML source. The output stream file is opened for writing only, in text-only mode, in CCSID 1208, and allows sharing with readers only. If the output stream file exists, the file is truncated to zero length before writing any data. If the output stream file already exists, it should have been created with a CCSID of 1208; otherwise, the resulting XML output may not be usable. If the output file does not exist, it will be created with a CCSID of 1208 before attempting to write the CDML source to it. The output file is created so that the file owner has read and write permission to it. The output file can be replaced if the user has the authority to do so. For more information on authority requirements for stream files, see the open()—Open File API in the Integrated File System section of the i5/OS APIs in the Information Center.

If the CCSID of the command is 65535, the API uses the job default CCSID as the CCSID for the command.

Error Messages

Message ID	Error Message Text
CPE3006 E	Input/output error.
CPE3014 E	The object name is not correct.
CPE3021 E	The value specified for the argument is not correct.
CPE3025 E	No such path or directory.
CPE3027 E	Operation not permitted.
CPE3029 E	Resource busy.
CPE3401 E	Permission denied.
CPE3403 E	Not a directory.
CPE3404 E	No space available.
CPE3406 E	Operation would have caused the process to be suspended.
CPE3407 E	Interrupted function call.
CPE3408 E	The address used for an argument was not correct.
CPE3436 E	There is not enough buffer space for the requested operation.
CPE3440 E	Operation not supported.
CPE3450 E	Descriptor not valid.
CPE3452 E	Too many open files for this process.
CPE3453 E	Too many open files in the system.
CPE3460 E	Storage allocation request failed.
CPE3470 E	Function not implemented.
CPE3471 E	Specified target is a directory.

Message ID	Error Message Text
CPE3474 E	Unknown system state.
CPE3484 E	A damaged object was encountered.
CPE3485 E	A loop exists in the symbolic links.
CPE3486 E	A path name is too long.
CPE3489 E	System resources not available to complete request.
CPE3490 E	Conversion error.
CPE3499 E	Object is suspended.
CPE3500 E	Object is a read only object.
CPE3507 E	Object too large.
CPE3511 E	File ID conversion of a directory failed.
CPE3512 E	A File ID could not be assigned when linking an object to directory.
CPE3513 E	File handle rejected by server.
CPE3524 E	Function not allowed.
CPF24B4 E	Severe error while addressing parameter list.
CPF3C19 E	Error occurred with receiver variable specified.
CPF3C21 E	Format name &1 is not valid.
CPF3C24 E	Length of the receiver variable is not valid.
CPF3C90 E	Literal value cannot be changed.
CPF3CF1 E	Error code parameter 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.
CPF9803 E	Cannot allocate object &2 in library &3.
CPF9810 E	Library &1 not found.
CPF9820 E	Not authorized to use library &1.
CPF9830 E	Cannot assign library &1.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

API introduced: V5R1

Top | "Program and CL Command APIs," on page 1 | APIs by category

Retrieve Command Information (QCDRCMDI) 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 command name	Input	Char(20)
5	Error code	I/O	Char(*)

Default Public Authority: *USE
Threadsafe: Yes

The Retrieve Command Information (QCDRCMDI) API retrieves information from a command definition object and places it into a single variable in the calling program. The amount of information returned depends on the size of the variable. The information returned is the same information returned by the Display Command (DSPCMD) command.

You can use the QCDRCMDI API to retrieve any operable command. This includes both interactive (such as Display Program (DSPPGM) and Create Library (CRTLIB)) and non-interactive (such as DO, IF, and ELSE) commands. It does not include command definition statements that appear in command source, such as CMD, DEP, ELEM, PARM, PARMCTL, and QUAL.

Authorities and Locks

Command Definition Object Authority
*USE

Library Authority
*EXECUTE

Command Definition Object Lock
*SHRRD

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 as long as you specify the length 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 result may not be predictable. The minimum length is 8 bytes.

Format name
INPUT; CHAR(8)

The format of the command information to be returned. One of the following format names may be used:

"CMDI0100 Format"	Basic command information.
"CMDI0200 Format" on page 123	Complete command information.

Qualified command name
INPUT; CHAR(20)

The name of the command whose values are to be retrieved. The first 10 characters contain the name of the command. The second 10 characters contain the name of the library where the command is located.

You can use these special values for the library name:

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

Error code
I/O; CHAR(*)

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

CMDI0100 Format

The following table describes the information that is returned in the receiver variable for the CMDI0100 format. For detailed descriptions of the fields, see "Field Descriptions" on page 124.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	CHAR(10)	Command name
18	12	CHAR(10)	Command library name
28	1C	CHAR(10)	Command processing program » or proxy target command « name
38	26	CHAR(10)	Command processing program » or proxy target command « library name
48	30	CHAR(10)	Source file name
58	3A	CHAR(10)	Source file library name
68	44	CHAR(10)	Source file member name
78	4E	CHAR(10)	Validity check program name
88	58	CHAR(10)	Validity check program library name
98	62	CHAR(10)	Mode information
108	6C	CHAR(15)	Where allowed to run
123	7B	CHAR(1)	Allow limited user
124	7C	BINARY(4)	Maximum positional parameters
128	80	CHAR(10)	Prompt message file name
138	8A	CHAR(10)	Prompt message file library name
148	94	CHAR(10)	Message file name
158	9E	CHAR(10)	Message file library name
168	A8	CHAR(10)	Help panel group name
178	B2	CHAR(10)	Help panel group library name
188	BC	CHAR(10)	Help identifier
198	C6	CHAR(10)	Search index name
208	D0	CHAR(10)	Search index library name
218	DA	CHAR(10)	Current library
228	E4	CHAR(10)	Product library
238	EE	CHAR(10)	Prompt override program name
248	F8	CHAR(10)	Prompt override program library name
258	102	CHAR(6)	Restricted to target release
264	108	CHAR(50)	Text description
314	13A	CHAR(2)	Command processing program call state
316	13C	CHAR(2)	Validity check program call state
318	13E	CHAR(2)	Prompt override program call state
320	140	BINARY(4)	Offset to help bookshelf information
324	144	BINARY(4)	Length of help bookshelf information
328	148	BINARY(4)	Coded character set ID (CCSID)
332	14C	CHAR(1)	Enabled for GUI indicator
333	14D	CHAR(1)	Threadsafe indicator

Offset		Type	Field
Dec	Hex		
334	14E	CHAR(1)	Multithreaded job action
335	14F	CHAR(1)	Proxy command indicator
336	150	CHAR(14)	Reserved
The offsets to these fields are specified in previous offset variables.		CHAR(*)	Help bookshelf information

CMDI0200 Format

The following table describes the information that is returned in the receiver variable for the CMDI0200 format. For detailed descriptions of the fields, see "Field Descriptions" on page 124.

Offset		Type	Field
Dec	Hex		
0	0		Returns everything from format CMDI0100
350	15E	CHAR(10)	REXX source file name
360	168	CHAR(10)	REXX source file library name
370	172	CHAR(10)	REXX source file member name
380	17C	CHAR(10)	REXX command environment name
390	186	CHAR(10)	REXX command environment library name
400	190	CHAR(40)	Reserved
440	1B8	BINARY(4)	Number of REXX exit entries
444	1BC	BINARY(4)	Length of a REXX exit entry
Format of a REXX exit entry (repeated by the number of REXX exit entries)			
See note	See note	CHAR(10)	REXX exit program name
See note	See note	CHAR(10)	REXX exit program library name
See note	See note	BINARY(4)	REXX exit code
Note: The decimal and hexadecimal offsets to the above 3 fields depend on the number of REXX exit entries and the length of a REXX exit entry. The REXX exit entry fields (currently REXX exit program name, REXX exit program library name, and REXX exit code) repeat, in the order listed, by the number of REXX exit entries defined for this command.			
The offsets to these fields are specified in previous offset variables.		CHAR(*)	Help bookshelf information

Help Bookshelf Information

The following table describes the help bookshelf information that is returned for both the CMDI0100 and CMDI0200 formats.

Offset		Type	Field
Dec	Hex		
See note	See note	CHAR(71)	Reserved

Offset		Type	Field
Dec	Hex		
See note	See note	CHAR(8)	Help bookshelf
Note: The location and length of the help bookshelf information structure may be determined by using the offset to help bookshelf information field and the length of help bookshelf information field.			

Field Descriptions

For more information on the following fields, refer to the documentation for the Create Command (CRTCMD) command in the online help.

Allow limited user. Whether or not a user with limited authorities is allowed to run this command. The possible values are 0 (*NO) or 1 (*YES).

Bytes available. The length of all data available for the requested format. All available data is returned if enough space is provided.

Bytes returned. The length of all data actually returned. If the data is truncated because the receiver variable is not large enough to hold the data, this value is less than the bytes available.

Coded character set ID (CCSID). The value of the coded character set ID associated with this command. It is the value of the job coded character set ID when this command was created.

Command library name. The name of the library in which the command description resides.

Command name. The name of the command description about which information is being returned.

Command processing program call state. The state the command processing program is called from. The possible values are:

*S The command processing program is called from system state.
*U The command processing program is called from user state.

Command processing program » or proxy target command library name. The name of the library in which the command processing program or proxy target command resides. « This field is blank if the command processing program name contains the special value *REXX.

Command processing program name » or proxy target command name. The name of the program or target command that accepts parameters from the command and processes the command. Consult the proxy command indicator to determine if the value is a program or command name. « The possible values are:

*REXX	The REXX fields returned in the CMDI0200 format contain valid information about the command.
CPP-program-name	The command processing program name.
» Proxy target command name	The proxy target command name. «

Current library. The name of the library used as the current library during the processing of this command. The possible values are:

*NOCHG	The current library does not change for the processing of this command. If the current library is changed during processing of the command, the change remains in effect after command processing is complete.
*CRTDFT	No current library is active during processing of the command. The current library that was active before command processing began is restored when processing is completed.
library-name	The name of the library that is used as the current library. When command processing is completed, the current library is restored to its previous value.

Enabled for GUI indicator. Whether the command prompt panels are enabled for conversion to a graphical user interface. The possible values are:

0	The command prompt panels are not enabled for conversion to a graphical user interface.
1	The command prompt panels are enabled for conversion to a graphical user interface by including information about the panel content in the 5250 data stream.

Help bookshelf. The name of the help bookshelf for this command. The possible values are:

*NONE	No help bookshelf is specified.
*LIST	The list of bookshelves in the user's book path.
book-shelf	The help bookshelf name.

Help bookshelf information. Information describing the help bookshelf.

See "Help Bookshelf Information" on page 123 for more details about help bookshelf information.

Help identifier. The name of the general help module for the names of the help identifiers for this command. The possible values are:

*NONE	No help identifier is specified.
*CMD	The name of the command is used.
help-ID-name	A user-specified help module was used.

Help panel group library name. The name of the library in which the panel group resides.

Help panel group name. The name of the panel group in which the online help information exists for this command. If *NONE is returned, no help panel group is defined for this command.

Length of a REXX exit entry. The length of one REXX exit entry. This value is currently 24. There are 10 bytes for the REXX exit program name, 10 bytes for the REXX exit library name, and 4 bytes for the REXX exit code.

Length of help bookshelf information. The length of the help bookshelf.

Maximum positional parameters. The maximum number of parameters than can be coded in a positional manner for this command. The possible values are:

-1	No maximum positional coding limit was specified for this command.
0 through 75	The maximum number of parameters that can be coded in a positional manner for this command.

Message file library name. The name of the library in which the message file resides.

Message file name. The message file from which messages identified on the DEP statements used to define the command are retrieved.

Mode information. The mode of operating environment to which the command applies. The characters of this field are as follows, and they can have a value of 0 (does not apply) or 1 (does apply):

1 Production mode
2 Debug mode
3 Service mode
4-10 Reserved

Multithreaded job action. The action to take when a command that is not threadsafe is called in a multithreaded job. The possible values are:

0 Use the action specified in QMLTTHDACN system value.
1 Run the command. Do not send a message.
2 Send an informational message and run the command.
3 Send an escape message, and do not run the command.

If the threadsafe indicator is either threadsafe or conditionally threadsafe, the multithreaded job action value will be returned as 1.

Number of REXX exit entries. The number of times the REXX exit entries are repeated. These fields are REXX exit program name, REXX exit program library name, and REXX exit code.

Offset to help bookshelf information. The offset to the help bookshelf information.

Product library. The name of the product library that is in effect during the processing of the command. The possible values are:

*NOCHG	The product library does not change for the processing of this command.
*NONE	There is no product library in the job's library list.
library-name	The name of the library that is used as the product library during the processing of the command.

Prompt message file library name. The name of the library in which the prompt message file resides.

Prompt message file name. The name of the message file that contains the prompt text for this command. If *NONE is returned, no message file was specified for prompt text.

Prompt override program call state. The state the prompt override program is called from. The possible values are:

*S The prompt override program is called from the system state.
*U The prompt override program is called from the user state.

Prompt override program name. This is the name of the prompt override program that replaces default values (on the prompt display) with the current actual values for the parameter. If *NONE is returned, no prompt override program was specified for this command.

» **Proxy command indicator.** Whether the command processing program name and command processing program library name fields contain program or command information. The possible values are:

0	The values specified for the command processing program and library refer to a program.

1 The values specified for the command processing program and library refer to a proxy target command.

When true, the fields returned are:

- Command name
- Command library name
- Command processing program or proxy target command name
- Command processing program or proxy target command library name
- Text description

Reserved. An ignored field.

Restricted to target release. The version, release, and modification level to which this command is restricted. If this field is blank, the command can be used in the current release. This applies only to a command used in a CL program. It must match the contents of the target release parameter on the Create CL Program (CRTCLPGM) command. See the CRTCLPGM command for more information. This field has the format VvRrMm, where:

Vv	The character V is followed by a 1-character version number.
Rr	The character R is followed by a 1-character release level.
Mm	The character M is followed by a 1-character modification level.

REXX command environment library name. The name of the library in which the REXX command environment program resides.

REXX command environment name. The command environment program that is active when the REXX CPP starts to run. The REXX interpreter calls this program to process commands encountered in the REXX procedure. The possible values are:

*COMMAND	The i5/OS control language command environment is used.
*CPICOMM	The Common Programming Interface (CPI) for Communications command environment is used. CPICOMM is the command environment used for CL commands that are embedded within a REXX procedure.
program-name	The name of the program to process commands found in the REXX procedure.

REXX exit code. A value which controls the conditions in which the REXX exit program is called. The possible values are:

2	The exit program is called whenever an external function or subroutine has been called by the REXX program. The exit program is responsible for locating and calling the requested routine.
3	The exit program is called whenever the interpreter is going to call a command. The exit program is responsible for locating and calling the command.
4	The exit program is called whenever a REXX instruction or function attempts an operation on the REXX external data queue.
5	The exit program is called when session input or output operations are attempted.
7	The exit program is called after running each clause of the REXX procedure to determine whether it must be stopped.
8	The exit program is called after running each clause of the REXX program to check if tracing must be turned on or off.
9	The exit program is called before interpretation of the first instruction of a REXX procedure.
10	The exit program is called after interpretation of the last instruction of a REXX procedure.

REXX exit program library name. The name of the library in which the REXX exit program resides.

REXX exit program name. The exit program used when the REXX interpreter is started under the conditions specified by the REXX exit code for this program.

REXX source file library name. The name of the library in which the REXX source file resides.

REXX source file member name. The name of the source file member that contains the REXX procedure that is the command processing program.

REXX source file name. The name of the source file that contains the REXX procedure that is the command processing program. The possible values are:

QREXSRC	The IBM-supplied source file, QREXSRC, contains the source member that is used.
source-file-name	The name of the REXX source file that is used.

Search index library name. The name of the library in which the help search index resides.

Search index name. The name of the search index for this command. The possible values are:

*NONE	No help search index is specified.
search index name	The name of the help search index that is used.

Source file library name. The name of the library in which the source file resides.

Source file member name. The name of the source file member that contains the command definition statements used to create the command.

Source file name. The name of the source file that contains the source file member used to create the command.

Text description. The user text, if any, used to briefly describe the command and its function.

Threadsafe indicator. Whether the command can be used safely in

a multithreaded job.

The possible values are:

0 The command is not threadsafe and should not be used

in a multithreaded job. The value for the multithreaded job action field defines the action to be taken by the command analyzer when the command is used in a multithreaded job.

1 The command is threadsafe and can be used safely in a

multithreaded job.

2 The command is threadsafe under certain conditions. See the documentation for the command to determine the conditions under which the command can be used safely in a

multithreaded job.

Validity check program call state. The state the validity check program is called from. The possible values are:

*S The validity check program is called from the system state.
*U The validity check program is called from the user state.

Validity check program library name. The name of the library in which the validity check program resides.

Validity check program name. The name of a program that performs additional user-defined validity checking on the parameters in the command. If *NONE is returned, no separate user-defined validity checking is done for this command. All validity checking is done by the command analyzer and the command processing program.

Where allowed to run. The environments in which this command is allowed to run. The characters of this field are as follows, and they can have a value of 0 (does not apply) or 1 (does apply):

1 Batch program (*BPGM)
2 Interactive program (*IPGM)
3 Can be run using QCMDEXC, QCAEXEC, or QCAPCMD (*EXEC)
4 Interactive job (*INTERACT)
5 Batch job (*BATCH)
6 Batch REXX procedure (*BREXX)
7 Interactive REXX procedure (*IREXX)
8-15 Reserved

Error Messages

Message ID	Error Message Text
CPF2150 E	Object information function failed.
CPF2151 E	Operation failed for &2 in &1 type *&3.
CPF24B4 E	Severe error while addressing parameter list.
CPF3CF1 E	Error code parameter not valid.
CPF3C21 E	Format name &1 is not valid.
CPF3C24 E	Length of the receiver variable is not valid.
CPF3C90 E	Literal value cannot be changed.
CPF6250 E	Cannot display or retrieve command &1 in library &2.
CPF8103 E	Command &4 in &9 damaged.
CPF8122 E	&8 damage on library &4.
CPF8123 E	Damage on object information for library &4.
CPF9801 E	Object &2 in library &3 not found.
CPF9802 E	Not authorized to object &2 in &3.
CPF9803 E	Cannot allocate object &2 in library &3.
CPF9807 E	One or more libraries in library list deleted.
CPF9808 E	Cannot allocate one or more libraries on library list.
CPF9810 E	Library &1 not found.
CPF9820 E	Not authorized to use library &1.
CPF9830 E	Cannot assign library &1.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

API introduced: V2R2

Top | "Program and CL Command APIs," on page 1 | APIs by category

Retrieve Module Information (QBNRMODI) 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 module name	Input	Char(20)
5	Error Code	I/O	Char(*)

Default Public Authority: *USE
Threadsafe: No

The Retrieve Module Information (QBNRMODI) API lets you retrieve module information and place it into a single variable in the calling program. The amount of information returned is limited to the size of the variable. This information is similar to the information returned using the Display Module (DSPMOD) command.

You can use the QBNRMODI API to retrieve the following:

- Module creation information
- Module compatibility information
- Module SQL attributes
- Module size information

Authorities and Locks

Library Authority
*EXECUTE

Module Authority
*USE

Module Lock
*SHRRD

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 module.

The possible format names are:

"MODI0100 Format" on page 131 Basic module information.

"MODI0200 Format" on page 133 Module size information.

Qualified module name
INPUT; CHAR(20)

The first 10 characters contain the module name. The second 10 characters contain the name of the library where the module is located.

You can use these special values for the library name:

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

Error code
I/O; CHAR(*)

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

MODI0100 Format

The following information is returned for the MODI0100 format. For detailed descriptions of the fields in the table, see "Field Descriptions" on page 135.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	CHAR(10)	Module name
18	12	CHAR(10)	Module library name
28	1C	CHAR(10)	Module attribute
38	26	CHAR(13)	Module creation date and time
51	33	CHAR(10)	Source file name
61	3D	CHAR(10)	Source file library name
71	47	CHAR(10)	Source member name
81	51	CHAR(13)	Source file change date and time
94	5E	CHAR(10)	Reserved
104	68	CHAR(10)	Module owner
114	72	CHAR(2)	Reserved
116	74	BINARY(4)	Module CCSID
120	78	CHAR(50)	Text description
170	AA	CHAR(1)	Creation data
171	AB	CHAR(10)	Sort sequence table name
181	B5	CHAR(10)	Sort sequence table library name
191	BF	CHAR(10)	Language identifier
201	C9	CHAR(3)	Reserved
204	CC	BINARY(4)	Optimization level
208	D0	BINARY(4)	Maximum optimization level

Offset		Type	Field
Dec	Hex		
212	D4	CHAR(1)	Debug data
213	D5	CHAR(1)	Module compressed status
214	D6	CHAR(2)	Reserved
216	D8	BINARY(4)	Minimum number of parameters
220	DC	BINARY(4)	Maximum number of parameters
224	E0	CHAR(1)	Module state
225	E1	CHAR(1)	Module domain
226	E2	CHAR(2)	Reserved
228	E4	BINARY(4)	Number of exported defined symbols
232	E8	BINARY(4)	Number of imported (unresolved) symbols
236	EC	CHAR(6)	Release module created on
242	F2	CHAR(6)	Release module created for
248	F8	CHAR(6)	Earliest release module can be restored to
254	FE	CHAR(1)	Enable performance collection
255	FF	CHAR(1)	Conversion required
256	100	BINARY(4)	Offset to program entry procedure name
260	104	BINARY(4)	Length of program entry procedure name
264	108	CHAR(1)	Program entry procedure name indicator
265	109	CHAR(10)	Profile data
275	113	CHAR(1)	Intermediate language (IL) data
276	114	CHAR(1)	Teraspace storage enabled
277	115	CHAR(1)	Storage model
278	116	CHAR(2)	Reserved
280	118	BINARY(4)	Offset to Licensed Internal Code options
284	11C	BINARY(4)	Length of Licensed Internal Code options
288	128	CHAR(68)	Reserved
356	164	BINARY(4)	Number of SQL statements
360	168	CHAR(18)	Relational database
378	17A	CHAR(10)	Commitment control
388	184	CHAR(10)	Allow copy of data
398	18E	CHAR(10)	Close SQL cursor
408	198	CHAR(10)	Naming convention
418	1A2	CHAR(10)	Date format
428	1AC	CHAR(1)	Date separator
429	1AD	CHAR(10)	Time format
439	1B7	CHAR(1)	Time separator
440	1B8	CHAR(10)	Delay PREPARE
450	1C2	CHAR(10)	Allow blocking
460	1CC	CHAR(10)	Default collection name
470	1D6	CHAR(10)	SQL package name

Offset		Type	Field
Dec	Hex		
480	1E0	CHAR(10)	SQL package library name
490	1EA	CHAR(10)	Dynamic user profile
500	1F4	CHAR(10)	SQL sort sequence table name
510	1FE	CHAR(10)	SQL sort sequence table library name
520	208	CHAR(10)	SQL language identifier
530	212	CHAR(10)	Connection method
540	21C	BINARY(4)	SQL path offset
544	220	BINARY(4)	SQL path length
548	224	CHAR(*)	Reserved
Module information through offsets			
		CHAR(*)	Program entry procedure name
		CHAR(*)	SQL path
		CHAR(*)	Licensed Internal Code options

MODI0200 Format

The following information is returned for the MODI0200 format. For detailed descriptions of the fields in the table, see "Field Descriptions" on page 135.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	CHAR(10)	Module name
18	12	CHAR(10)	Module library name
Module size and limit information			
28	1C	BINARY(4)	Current total module size
32	20	BINARY(4)	Maximum module size
36	24	BINARY(4)	Current procedures and constants size
40	28	BINARY(4)	Maximum procedures and constants size
44	2C	BINARY(4)	Current debug space size
48	30	BINARY(4)	Maximum debug space size
52	34	BINARY(4)	Current associated space size
56	38	BINARY(4)	Maximum associated space size
60	3C	BINARY(4)	Current module constants size
64	40	BINARY(4)	Maximum module constants size
68	44	BINARY(4)	Current static storage size
72	48	BINARY(4)	Maximum static storage size
76	4C	BINARY(4)	Current dictionary mapping table size
80	50	BINARY(4)	Maximum dictionary mapping table size
84	54	BINARY(4)	Current exception mapping table size

Offset		Type	Field
Dec	Hex		
88	58	BINARY(4)	Maximum exception mapping table size
92	5C	BINARY(4)	Current exception mapping table list area size
96	60	BINARY(4)	Maximum exception mapping table list area size
100	64	BINARY(4)	Current binding specifications component size
104	68	BINARY(4)	Maximum binding specifications component size
108	6C	BINARY(4)	Current string directory component size
112	70	BINARY(4)	Maximum string directory component size
116	74	BINARY(4)	Current dictionary component size
120	78	BINARY(4)	Maximum dictionary component size
124	7C	BINARY(4)	Current instructions component size
128	80	BINARY(4)	Maximum instructions component size
132	84	BINARY(4)	Current initialization component size
136	88	BINARY(4)	Maximum initialization component size
140	8C	BINARY(4)	Current alias component size
144	90	BINARY(4)	Maximum alias component size
148	94	BINARY(4)	Current type information component size
152	98	BINARY(4)	Maximum type information component size
156	9C	BINARY(4)	Current literal pool component size
160	A0	BINARY(4)	Maximum literal pool component size
164	A4	BINARY(4)	Current static storage work area size
168	A8	BINARY(4)	Maximum static storage work area size
172	AC	BINARY(4)	Current binding work area size
176	B0	BINARY(4)	Maximum binding work area size
180	B4	BINARY(4)	Current number of auxiliary storage segments
184	B8	BINARY(4)	Maximum number of auxiliary storage segments
188	BC	BINARY(4)	Current number of static storage allocations
192	C0	BINARY(4)	Maximum number of static storage allocations
196	C4	BINARY(4)	Current number of procedures
200	C8	BINARY(4)	Maximum number of procedures
204	CC	BINARY(4)	Current number of copyrights
208	D0	BINARY(4)	Maximum number of copyrights
212	D4	CHAR(4)	Reserved
216	D8	BINARY(8)	Current static storage size - long
224	E0	BINARY(8)	Maximum static storage size - long
232	E8	CHAR(84)	Reserved
Procedure size and limit information			
316	13C	BINARY(4)	Current automatic storage allocation size
320	140	BINARY(4)	Maximum automatic storage allocation size
324	144	BINARY(4)	Offset to largest automatic storage allocation procedure
328	148	BINARY(4)	Length of largest automatic storage allocation procedure

Offset		Type	Field
Dec	Hex		
332	14C	BINARY(4)	Current Licensed Internal Code stack allocation size
336	150	BINARY(4)	Maximum Licensed Internal Code stack allocation size
340	154	BINARY(4)	Offset to largest Licensed Internal Code stack allocation procedure
344	158	BINARY(4)	Length of largest Licensed Internal Code stack allocation procedure
348	15C	BINARY(4)	Current debug statement mapping table size
352	160	BINARY(4)	Maximum debug statement mapping table size
356	164	BINARY(4)	Offset to largest debug statement mapping table procedure
360	168	BINARY(4)	Length of largest debug statement mapping table procedure
364	16C	BINARY(4)	Current exception statement mapping table size
368	170	BINARY(4)	Maximum exception statement mapping table size
372	174	BINARY(4)	Offset to largest exception statement mapping table procedure
376	178	BINARY(4)	Length of largest exception statement mapping table procedure
380	17C	BINARY(4)	Current machine instruction range mapping size
384	180	BINARY(4)	Maximum machine instruction range mapping size
388	184	BINARY(4)	Offset to largest machine instruction range mapping procedure
392	188	BINARY(4)	Length of largest machine instruction range mapping procedure
396	18C	BINARY(4)	Current largest procedure size
400	190	BINARY(4)	Maximum largest procedure size
404	194	BINARY(4)	Offset to largest procedure name
408	198	BINARY(4)	Length of largest procedure name
412	19C	CHAR(*)	Reserved
Procedure information through offsets			
		CHAR(*)	Largest automatic storage allocation procedure name
		CHAR(*)	Largest Licensed Internal Code stack allocation procedure name
		CHAR(*)	Largest debug statement mapping table procedure name
		CHAR(*)	Largest exception statement mapping table procedure name
		CHAR(*)	Largest machine instruction range mapping procedure name
		CHAR(*)	Largest procedure name

Field Descriptions

For more detailed information than that provided in the following field descriptions, refer to the Control Language (CL) information in the iSeries Information Center for the Create XXX Module (CRTxxxMOD) commands.

Allow blocking. Indicates whether blocking is used to improve the performance of certain SQL statements.

Possible values are:

**NONE* Blocking is not used.

**READ* Blocking is used for read-only cursors when running COMMIT(*NONE) and there are no EXECUTE or EXECUTE IMMEDIATE statements.

*ALLREAD Blocking is used for all read-only cursors when running COMMIT(*NONE) or COMMIT(*CHG).

Allow copy of data. Indicates whether a copy of the data can be used in the implementation of an SQL query.

Possible values are:

*NO A copy of the data cannot be used.
*YES A copy of the data can be used when needed.
*OPTIMIZE The system determines whether a copy of the data is used for optimal performance.

Bytes available. The number of bytes of data available to be returned. All available data is returned if enough space is provided.

Bytes returned. The number of bytes of data returned.

Close SQL cursor. Specifies when SQL cursors are implicitly closed and SQL prepared statements are implicitly discarded.

Possible values are:

*ENDMOD When the module ends.
*ENDACTGRP When the activation group is deleted.

Commitment control. The level of commitment control that was specified on the SQL precompile.

Possible values are:

*NONE No commitment control was specified on the SQL precompile. Uncommitted changes in other jobs can be seen.
*CHG Objects referred to in SQL COMMENT ON, CREATE, DROP, GRANT, LABEL ON, and REVOKE statements, and updated, deleted, or inserted rows (records) are locked until the end of the unit of work (transaction). Uncommitted changes in other jobs can be seen.
*CS Objects referred to in SQL COMMENT ON, CREATE, DROP, GRANT, LABEL ON, and REVOKE statements, and updated, deleted, or inserted rows (records) are locked until the end of the unit of work (transaction). A row (record) that is selected but not updated is locked until the next row (record) is selected. Uncommitted changes in other jobs cannot be seen.
*ALL Objects referred to in SQL COMMENT ON, CREATE, DROP, GRANT, LABEL ON, and REVOKE statements, and all rows selected, updated, deleted, or inserted are locked until the end of the unit of work (transaction). Uncommitted changes in other jobs cannot be seen.

Connection method. Indicates whether connections are allowed to one or more relational databases.

The following special values can be returned:

*RUW Only one connection to a relational database is allowed. Consecutive CONNECT statements result in the previous connection being disconnected before a new connection is established.
*DUW Connections to several relational databases are allowed. Consecutive CONNECT statements to additional relational databases do not result in disconnection of previous connections. The SET CONNECTION statement can be used to switch between connections. Read-only connections may result.

Conversion required. Indicator as to whether the module has been converted to reduced instruction set computer (RISC) format or if conversion is still required.

0 Conversion is not required. It has already been converted.
1 Conversion is required.

Creation data. Whether the module has all creation data and if that data is observable or unobservable. All observable creation data is needed to re-create the module using the Change Module (CHGMOD) command. All creation data (either observable or unobservable) is needed to convert the module during restore.

0 *NO. Not all the creation data is present.
1 *YES. All the creation data is present and observable.
2 *UNOBS. All the creation data is present but not all of that data is observable.

Current alias component size. The size, in bytes, of the alias component. The size is the decompressed size, even if the module is compressed. The current size increases as more data objects, particularly arrays and structures, are added to the module.

Current associated space size. The size, in bytes, of the associated space. The number of SQL statements in the module may affect the current size.

Current automatic storage allocation size. The size, in bytes, of the largest automatic storage allocation associated with the procedure referenced by the offset to the largest automatic storage allocation procedure name for this module. The current size increases as more automatic variables and bound procedure calls are added.

Current binding specifications component size. The size, in bytes, of the binding specifications component. The size is the decompressed size, even if the module is compressed. The current size increases as imported and exported procedures and data items are added.

Current binding work area size. The size, in bytes, of the binding work area. The size is the decompressed size, even if the module is compressed. The current size increases as imports and exports are added to the module.

Current debug space size. The size, in bytes, of the debug space. The size is the decompressed size, even if the module is compressed. See the debug view (DBGVIEW) parameter of the Create xxx Module (CRTxxxMOD) command for more information on the different debug view levels you can specify at create time. You can completely remove the debug view data from the module by using the remove observability (RMVOBS) parameter of the Change Module (CHGMOD) command.

Current debug statement mapping table size. The size, in bytes, of the debug statement mapping table associated with the procedure referenced by the offset to the largest debug statement mapping table procedure name for this module. The current size increases as statements are added or increase in complexity.

Current dictionary component size. The size, in bytes, of the dictionary component. The size is the decompressed size, even if the module is compressed. The current size increases as more data objects, exception handlers, constants, and procedures are added.

Current dictionary mapping table size. The amount of space, in bytes, for the dictionary mapping table. The size is the decompressed size, even if the module is compressed. The dictionary mapping table does not exist if the DBGVIEW(*NONE) option is used when the module is created. If it does exist, its current size depends on the number of different variables that are declared in the module.

Current exception mapping table list area size. The amount of space, in bytes, for the exception mapping table list area. The size is the decompressed size, even if the module is compressed. The current

size depends on the number of exception handlers that are enabled in the code. The handlers could be either defined by the user or defined by the compiler to handle exceptions in the generated code.

Current exception mapping table size. The amount of space, in bytes, for the exception mapping table. The size is the decompressed size, even if the module is compressed. The current size depends on the number of exception handlers that are either declared by the user or defined by the compiler to handle exceptions in the generated code.

Current exception statement mapping table size. The size, in bytes, of the exception statement mapping table associated with the procedure referenced by the offset to the largest exception statement mapping table procedure name for this module. This table is used to map high-level language statements to Licensed Internal Code instructions. The current size increases as statements are added or increase in complexity.

Current initialization component size. The size, in bytes, of the initialization component. The size is the decompressed size, even if the module is compressed. The current size increases as more statements to statically initialize data objects are added.

Current instructions component size. The size, in bytes, of the instructions component. The size is the decompressed size, even if the module is compressed.

Current largest procedure size. The size, in bytes, of the largest procedure in the module.

Current Licensed Internal Code stack allocation size. The size, in bytes, of the Licensed Internal Code stack allocation associated with the procedure referenced by the offset to the largest Licensed Internal Code stack allocation procedure name for this module. The current size increases at higher levels of optimization.

Current literal pool component size. The size, in bytes, of the literal pool component. The size is the decompressed size, even if the module is compressed. The current size increases as more literals or initializations are added to the module.

Current machine instruction range mapping size. The size, in bytes, of the machine instruction range mapping associated with the procedure referenced by the offset to the largest machine instruction range mapping procedure name for this module. The number of times exception handlers are enabled and disabled affects the current size. Also, some instructions, like packed decimal arithmetic, affect this size.

Current module constants size. The amount of space, in bytes, for module constants. The size is the decompressed size, even if the module is compressed. Changing the number of large aggregate constants or the number of smaller constants may affect the current size.

Current number auxiliary storage segments. The number of auxiliary storage segments used by this module.

Current number of copyrights. The number of copyrights defined in this module.

Current number of procedures. The number of procedures declared in this module.

Current number of static storage allocations. The number of static storage allocations used by this module. The current number increases as static data items (particularly imported data items) are added to the module.

Current procedures and constants size. The current amount of space, in kilobytes, for procedures and constants. The size is the decompressed size, even if the module is compressed. The current size increases

as more instructions are added. The number of literals and the size of the literal values may have an effect on the current size. This number could be zero in which case it means the size is less than 1 kilobyte.

Current static storage size. The amount of static storage, in bytes, required for the module. The size is the decompressed size, even if the module is compressed. As more and larger static and exported variables are declared, the current size increases. A value of 4294967295 will be given if 4 gigabytes (4294967296) or greater is needed. In this case, the *current static storage size - long* field should be used instead.

Current static storage size - long. The current amount of static storage, in bytes, required for the module.

Current static storage work area size. The size, in bytes, of the static storage work area. The size is the decompressed size, even if the module is compressed. The current size increases as static data items or initializations are added to the module.

Current string directory component size. The size, in bytes, of the string directory component. The size is the decompressed size, even if the module is compressed. The current size increases as imported and exported procedures and data items are added.

Current total module size. The size of the module, in kilobytes. The size is the decompressed size, even if the module is compressed.

Current type information component size. The size, in bytes, of the type information component. The size is the decompressed size, even if the module is compressed. The current size increases as more procedures are added to the module.

Date format. Specifies the format used when accessing date result columns through SQL. All output date fields are returned in this format. For input date strings, the value you specify is used to determine whether the date is a valid format.

Possible values are:

**USA*	USA format
*ISO	International Standards Organization format
*EUR	European format
*JIS	Japanese Industrial Standard Christian Era
*MDY	Month/day/year format
*DMY	Day/month/year format
*YMD	Year/month/day format
*JUL	Julian format

Date separator. Specifies the separator used when accessing date result columns. A blank value indicates either that there are no SQL statements or that the separator character is a blank. Assume the latter if the number of SQL statements parameter is not zero.

Debug data. Indicates whether debug data exists for this module. If debug data exists, the module may be debugged using the source debugger.

0	The module does not contain debug data.
1	The module does contain debug data.

Default collection name. Specifies the collection name used for the unqualified names of tables, views, indexes, and SQL packages. *NONE indicates there is no default collection.

Delay PREPARE. Indicates whether SQL prepare processing can be delayed until the statement is actually used.

Possible values are:

**YES*	Prepare processing can be delayed.
**NO*	Prepare processing cannot be delayed.

Dynamic user profile. Specifies the user profile used for dynamic SQL statements.

The following special values can be returned:

**USER*	Local dynamic SQL statements are run under the profile of the module's user. Distributed dynamic SQL statements are run under the profile of the SQL package's user.
**OWNER*	Local dynamic SQL statements are run under the profile of the module's owner. Distributed dynamic SQL statements are run under the profile of the SQL package's owner.

Earliest release module can be restored to. The earliest version, release, and modification level of the operating system to which the module may be restored.

The field has a VvRrMm format, where:

Vv	The character V is followed by a 1-character version number.
Rr	The character R is followed by a 1-character release level.
Mm	The character M is followed by a 1-character modification level.

Enable performance collection. The level of performance collection enabled for this module.

The following values can be returned:

*'00'X *NONE or '10'X *PEP*	This gives the entry/exit information for the PEP only. No entry/exit hooks in the module's internal procedures and no precall or postcall hooks around calls to other procedures are included. **Note:** If *NONE is shown and the module was created or re-created on an iSeries server running Version 3 Release 6 Modification 0 prior to the installation of PTF MF11968, the module will not have any performance collection enabled. To enable performance collection, use one of the following commands and specify ENBFPRCOL(*PEP): • Change Module (CHGMOD) • Change Program (CHGPGM) • Change Service Program (CHGSRVPGM)
*'50'X *ENTRYEXIT *NONLEAF*	This gives the entry/exit information on all of the non-leaf procedures in the module. This includes the PEP routine. This is useful to capture information on most routines but not at the expense of destroying the 'leaf-ness' of the leaf procedures.
*'70'X *ENTRYEXIT *ALLPRC*	This gives the entry/exit information on all the procedures of the module (including those that were leaf procedures). This includes the PEP routine. This is useful to capture information on all procedures.
*'D0'X *FULL *NONLEAF*	This gives the entry/exit information on all the procedures of the module that are not leaf procedures. This includes the PEP routine. Precall and postcall hooks around calls to external procedures are included.
*'F0'X *FULL *ALLPRC*	This gives the entry/exit information on all procedures of the module (including those that were leaf procedures). This includes the PEP routine. Precall and postcall hooks around calls to external procedures are also included. This is useful to capture information on all procedures.

Intermediate language (IL) data. Whether the module has intermediate language (IL) data.

1 The module contains IL data.
0 The module does not contain IL data.

Language identifier. The language identifier used when the module was compiled. A

possible special value is:

**JOBRUN* The language identifier associated with the job at the time the program (in which the module is bound) runs.

Note: This language identifier does not apply to DB2 for iSeries statements that may be contained in this module.

Largest automatic storage allocation procedure name. The name of the largest automatic storage allocation procedure in the module.

Largest debug statement mapping table procedure name. The name of the largest debug statement mapping table procedure in the module.

Largest exception statement mapping table procedure name. The name of the largest exception statement mapping table procedure in the module.

Largest Licensed Internal Code stack allocation procedure name. The name of the largest Licensed Internal Code stack allocation procedure in the module.

Largest machine instruction range mapping procedure name. The name of the largest machine instruction range mapping procedure in the module.

Largest procedure name. The name of the largest procedure in the module.

Length of largest automatic storage allocation procedure name. The size, in bytes, of the name of the largest automatic storage allocation procedure in this module.

Length of largest debug statement mapping table procedure name. The size, in bytes, of the name of the largest debug statement mapping table procedure.

Length of largest exception statement mapping table procedure name. The size, in bytes, of the name of the largest exception statement mapping table procedure.

Length of largest Licensed Internal Code stack allocation procedure name. The size, in bytes, of the name of the largest Licensed Internal Code stack allocation procedure.

Length of largest machine instruction range mapping procedure name. The size, in bytes, of the name of the largest machine instruction range mapping procedure.

Length of largest procedure name. The size, in bytes, of the name of the largest procedure.

Length of Licensed Internal Code options. The size, in two-byte characters, of the Licensed Internal Code options string. This is 0 if no Licensed Internal Code options were used for this module.

Length of program entry procedure name. The size, in bytes, of the program entry procedure name.

Licensed Internal Code options. The Licensed Internal Code options being used by the module. This field is specified in UCS-2 (CCSID 13488).

Maximum alias component size. The maximum possible size, in bytes, of the alias component.

Maximum associated space size. The maximum possible size, in bytes, of the associated space.

Maximum automatic storage allocation size. The maximum possible size, in bytes, of the largest automatic storage allocation associated with a procedure for a module.

Maximum binding specifications component size. The maximum possible size, in bytes, of the binding specifications component.

Maximum binding work area size. The maximum possible size, in bytes, of the binding work area.

Maximum debug space size. The maximum possible size, in bytes, of the debug space.

Maximum debug statement mapping table size. The maximum possible size, in bytes, of the debug statement mapping table associated with a procedure for a module.

Maximum dictionary component size. The maximum possible size, in bytes, of the dictionary component.

Maximum dictionary mapping table size. The maximum possible amount of space, in bytes, for the dictionary mapping table.

Maximum exception mapping table list area size. The maximum possible amount of space, in bytes, for the exception mapping table list area.

Maximum exception mapping table size. The maximum possible amount of space, in bytes, for the exception mapping table.

Maximum exception statement mapping table size. The maximum possible size, in bytes, of the exception statement mapping table associated with a procedure for a module.

Maximum initialization component size. The maximum possible size, in bytes, of the initialization component.

Maximum instructions component size. The maximum possible size, in bytes, of the instructions component.

Maximum largest procedure size. The maximum possible size, in bytes, of a procedure in a module.

Maximum Licensed Internal Code stack allocation size. The maximum possible size, in bytes, of the Licensed Internal Code stack allocation associated with a procedure for a module.

Maximum literal pool component size. The maximum possible size, in bytes, of the literal pool component.

Maximum machine instruction range mapping size. The maximum possible size, in bytes, of the machine instruction range mapping associated with a procedure for a module.

Maximum module constants size. The maximum possible amount of space, in bytes, for module constants.

Maximum module size. The largest size, in kilobytes, allowed for a module.

Maximum number auxiliary storage segments. The maximum possible number of auxiliary storage segments used by a module.

Maximum number of copyrights. The maximum possible number of copyrights defined in a module.

Maximum number of parameters. The maximum number of parameters that are to be received by the program entry procedure if one is present in the module.

Maximum number of procedures. The maximum possible number of procedures declared in a module.

Maximum number of static storage allocations. The maximum possible number of static storage allocations used by a module.

Maximum optimization level. The highest level of optimization you may request. If observability has been removed from the module, this maximum optimization level value might not be the same as the one specified when the module was created.

The following values can be returned:

65535	The module is not restricted to a maximum optimization level. It can be retranslated to any of the supported optimization levels. 65535 is also known as *NOMAX.
40	Maximum level of optimization. This level includes all the optimizations performed at optimization level 30. In addition, it includes optimization that disables call and instruction tracing. Thus, tracing of modules created at this optimization level cannot be done.
30	More optimization is performed in addition to those performed at optimization level 20. Variables cannot be changed but can be displayed while the program is being debugged. However, the displayed value of the variable during debugging may not be its actual value. 30 is also known as *FULL.
20	Some optimization is performed on the generated code. When the module optimized at this level is being debugged, the variables can be displayed but not changed. This level improves the performance of the module slightly over level 10. 20 is also known as *BASIC.
10	No additional optimization is performed on the generated code. Variables can be displayed and changed when the program is being debugged. With no optimization of the code, this value provides the lowest level of module performance. 10 is also known as *NONE.

Maximum procedures and constants size. The maximum possible amount of space, in kilobytes, for procedures and constants. This number could be zero, in which case it means the size is less than 1 kilobyte.

Maximum static storage size. The maximum possible amount of static storage, in kilobytes, required for a module.

Maximum static storage size - long. The maximum possible amount of static storage, in bytes, required for a module.

Maximum static storage work area size. The maximum possible size, in bytes, of the static storage work area.

Maximum string directory component size. The maximum possible size, in bytes, of the string directory component.

Maximum type information component size. The maximum possible size, in bytes, of the type information component.

Minimum number of parameters. The minimum number of parameters that are to be received by the program entry procedure if one is present in the module.

Module attribute. The programming language in which the module is written or the product that produced the module.

Module CCSID (Coded Character Set ID). The coded character set identifier (CCSID) for this module.

Module compressed status. Indicates whether the module is in compressed format.

The following values can be returned:

0 The module is not in a compressed status.
1 The module is in a compressed status.

Module creation date and time. The date and time when the module was created. The creation date and time field is in the CYYMMDDHHMMSS format as follows:

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

Module domain. The domain of the module.

The possible value is:

S The module is system domain.

Module library name. The library that contains the module.

Module name. The name of the module whose information is being given.

Module owner. The name of the user profile of the user who owns this module.

Module state. The state of the module.

The Possible values are:

I The module is inherit state and can be bound together with either system state modules or user state modules.
S The module is system state and must be bound with other system state modules or inherit state modules.
U The module is user state and must be bound with other user state modules or inherit state modules.

Naming convention. The naming convention used for naming objects in SQL statements.

Possible values are:

**SQL* The SQL naming convention is used.
**SYS* The system naming convention is used.

Number of exported defined symbols. The number of exported procedures and variables in this module.

Number of imported (unresolved) symbols. The number of imported procedures and variables in this module.

Number of SQL statements. The number of DB2 UDB for iSeries statements contained in the module.

Offset to largest automatic storage allocation procedure name. The offset from the beginning of the receiver variable where the largest automatic storage allocation procedure begins.

Offset to largest debug statement mapping table procedure name. The offset from the beginning of the receiver variable where the largest debug statement mapping table procedure name begins.

Offset to largest exception statement mapping table procedure name. The offset from the beginning of the receiver variable where the largest exception statement mapping table procedure name begins.

Offset to largest Licensed Internal Code stack allocation procedure name. The offset from the beginning of the receiver variable where the largest Licensed Internal Code stack allocation procedure name begins.

Offset to largest machine instruction range mapping procedure name. The offset from the beginning of the receiver variable where the largest machine instruction range mapping procedure name begins.

Offset to largest procedure name. The offset from the beginning of the receiver variable where the largest procedure name begins.

Offset to Licensed Internal Code options. The offset from the beginning of the receiver variable where the Licensed Internal Code options begin.

Offset to program entry procedure name. The offset from the beginning of the receiver variable where the program entry procedure name begins.

Optimization level. Optimization levels cause the translator to produce machine code that reduces the amount of system resources necessary to run the program. The more optimization, the more efficiently the module runs on the system. Also, with more optimization you may not be able to change or view variables that have been optimized.

The Possible values are:

65535	The module is not restricted to a maximum optimization level. It can be retranslated to any of the supported optimization levels. 65535 is also known as *NOMAX.
40	Maximum level of optimization. This level includes all the optimizations performed at optimization level 30. In addition, it includes optimization that disables call and instruction tracing. Thus, tracing of modules created at this optimization level cannot be done.
30	More optimization is performed in addition to those performed at optimization level 20. Variables cannot be changed but can be displayed while the program is being debugged. However, the displayed value of the variable during debugging may not be its actual value. 30 is also known as *FULL.
20	Some optimization is performed on the generated code. When the module optimized at this level is being debugged, the variables can be displayed but not changed. This level improves the performance of the module slightly over level 10. 20 is also known as *BASIC.
10	No additional optimization is performed on the generated code. Variables can be displayed and changed when the program is being debugged. With no optimization of the code, this value provides the lowest level of module performance. 10 is also known as *NONE.

Profile data. The profile data attribute for the module.

**NOCOL*	The collection of profile data is not enabled. The module will not collect profile data when it is included in a program or service program object.
**COL*	The collection of profile data is enabled. The module will collect profile data when it is included in a program or service program object.

Program entry procedure name. The name of the program entry procedure if one is present in the module.

Program entry procedure name indicator. Indicates if a program entry procedure is present in the module.

0 The module does not have a program entry procedure.
1 The module does have a program entry procedure.

Relational database. The default relational database that was specified on the SQL precompile.

Possible values are:

**LOCAL* The module can access data only on the local system.
*Non-blank value other than *LOCAL* The name associated with this module in the relational database index.

Release module created for. The version, release, and modification level of the operating system for which the module was created.

The field has a VvRrMm format, where:

Vv The character V is followed by a 1-character version number.
Rr The character R is followed by a 1-character release level.
Mm The character M is followed by a 1-character modification level.

Release module created on. The version, release, and modification level of the operating system that was running on the system when the module was created. The field has a VvRrMm format, where:

Vv The character V is followed by a 1-character version number.
Rr The character R is followed by a 1-character release level.
Mm The character M is followed by a 1-character modification level.

Reserved. An ignored field.

Sort sequence table library name. The name of the library that is used to locate the sort sequence table.

Possible special values are:

**LIBL* The library list is searched when the program (in which the module is bound) runs.
**CURLIB* The current library is searched when the program (in which the module is bound) runs.

Sort sequence table name. The name of the sort sequence table used when the module was compiled.

Possible special values are:

**HEX* No sort sequence is used.
**JOBRUN* The SRTSEQ value associated with the job at the time the program (in which the module is bound) runs.
**LANGIDSHR* The shared sort sequence for the language identifier (LANGID).
**LANGIDUNQ* The unique sort sequence for the language identifier.

Note: This sort sequence table does not apply to DB2 UDB for iSeries statements that may be contained in this module.

Source file change date and time. The date and time when the source member that was used to create this module was last changed. The source file change date and time field is in the CYYMMDDHHMMSS format as follows:

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

Source file library name. The library that contained the source file that was used to create this module.

Source file name. The source file that contained the source member that was used to create this module. If this field is blank, the module was created from an inline source file

Source member name. The source file member from which this module was created.

SQL language identifier. Returns the language identifier used when the module was compiled. This information is blank if the module does not contain any language identification information.

The special value that can be returned is:

**JOBRUN*	The language identifier is the LANGID associated with the job at the time the program (in which the module is bound) runs.

SQL package library name. Specifies the name of the library containing the SQL package. A blank indicates the module is not to be distributed.

The following possible special values can be returned:

**LIBL*	The SQL package is found by looking for it in the library list when the program (in which the module is bound) runs.
**CURLIB*	The SQL package is found in the current library when the program (in which the module is bound) runs.

SQL package name. Specifies the name of the SQL package created on the relational database specified on the RDB parameter of the command that created this module. *NONE indicates that this is not a distributed module.

SQL path. The list of libraries used during resolution of functions and data types within SQL statements. The list is in the form of repeating library names, each surrounded by double quotes and separated by commas.

SQL path length. The length, in bytes, of the SQL path.

SQL path offset. The offset, in bytes, from the beginning of this format definition to the beginning of the SQL path.

SQL sort sequence table library name. Returns the name of the library that is used to locate the SQL sort sequence table. This information is blank either if the module does not contain any SQL sort sequence information, or if a special value was returned for the SQL sort sequence table name.

The following possible special values that can be returned for the library are:

**LIBL*	The SQL sort sequence table is found by looking for it in the library list when the program (in which the module is bound) runs.
**CURLIB*	The SQL sort sequence table is found in the current library when the program (in which the module is bound) runs.

SQL sort sequence table name. Returns the SQL statement sort sequence table name used when the module was created. This information is blank if the module does not contain any SQL sort sequence information.

The following possible special values can also be returned:

**HEX*	No SQL sort sequence is used for the SQL statements.
*JOBRUN	The SQL sort sequence is the SRTSEQ value associated with the job at the time the SQL statements within the module are run.
*LANGIDSHR	The shared SQL sort sequence for the language identifier (LANGID) is used for the SQL statements.
*LANGIDUNQ	The unique SQL sort sequence for the language identifier (LANGID) is used for the SQL statements. **Note:** This sort sequence table does not apply to DB2 UDB for iSeries statements that may be contained in this module.

Storage model. Where the automatic and static storage for this bound module is allocated at run time.

The following values can be returned:

*0 *SNGLVL*	Automatic and static storage are allocated from single-level storage.
1 *TERASPACE	Automatic and static storage are allocated from teraspace.
2 *INHERIT	Automatic and static storage are allocated from either single-level storage or teraspace, depending on the activation.

Teraspace storage enabled. The teraspace storage capability for this module.

Possible values are:

*'00'X *NO*	The module is not teraspace storage enabled.
*'80'X *YES*	The module is teraspace storage enabled.

Text description. The text description that was provided for this module.

Time format. Specifies the format used when accessing time -result columns through SQL. All output time fields are returned in this format. For input time strings, the value you specify is used to determine whether the time is a valid format.

Possible values are:

**USA*	USA format
**ISO*	International Standards Organization format
**EUR*	European format

**JIS*	Japanese Industrial Standard Christian Era
**HMS*	Hours/minutes/seconds format

Time separator. Specifies the separator used when accessing time result columns. A blank value indicates that there are no SQL statements or that the separator character is a blank. Assume the latter if the number of SQL statements parameter is not zero.

Error Messages

Message ID	Error Message Text
CPF24B4 E	Severe error while addressing parameter list.
CPF3CF1 E	Error code parameter not valid.
CPF3C19 E	Error occurred with receiver variable specified.
CPF3C21 E	Format name &1 is not valid.
CPF3C24 E	Length of the receiver variable is not valid.
CPF3C90 E	Literal value cannot be changed.
CPF5CE7 E	Error occurred while retrieving *MODULE data.
CPF8122 E	&8 damage on library &4.
CPF8123 E	Damage on object information for library &4.
CPF813D E	Service program &4 in &9 damaged.
CPF9801 E	Object &2 in library &3 not found.
CPF9802 E	Not authorized to object &2 in &3.
CPF9803 E	Cannot allocate object &2 in library &3.
CPF9804 E	Object &2 in library &3 damaged.
CPF9806 E	Cannot perform function for object &2 in library &3.
CPF9807 E	One or more libraries in library list deleted.
CPF9808 E	Cannot allocate one or more libraries on library list.
CPF9810 E	Library &1 not found.
CPF9811 E	Program &1 in library &2 not found.
CPF9820 E	Not authorized to use library &1.
CPF9821 E	Not authorized to program &1 in library &2.
CPF9830 E	Cannot assign library &1.

API introduced: V3R6

Top | "Program and CL Command APIs," on page 1 | APIs by category

---

Retrieve Program Associated Space (QCLRPGAS) API

Required Parameter Group:

1	Output data buffer	Output	Char(*)
2	Length of output data buffer	Input	Binary(4)
3	Qualified program name	Input	Char(20)
4	Call stack counter	Input	Binary(4)
5	Data handle	Input	Char(16)
6	Length of data available	Output	Binary(4)
7	Error code	I/O	Char(*)

Default Public Authority: *EXCLUDE
Threadsafe: No

The Retrieve Program Associated Space (QCLRPGAS) API allows you to retrieve information from the associated space of an original program model (OPM), user-state program in the user domain. This API is intended to be used only on programs created by the Create Program (QPRCRTPG) API.

This API is used to retrieve information that was previously stored in the associated space. For example, it can be used by compilers at run time to retrieve information about the compilation process that was previously stored. The associated space of a program object is not a replacement for data areas or user spaces. It is recommended for static data only.

Authorities and Locks

Program Library Authority
*USE

Program Authority
*ALL

Program Lock
*EXCLRD

Required Parameter Group

Output data buffer
OUTPUT; CHAR(*)

The information retrieved from the associated space. This information contains scalar data only. If information that was previously stored contained pointer data, the pointer data was not preserved.

Length of output data buffer
INPUT; BINARY(4)

The length of the output data buffer, in bytes. If this value is larger than the actual size of the output data buffer, the results may not be predictable.

Qualified program name
INPUT; CHAR(20)

The program object for which you want to retrieve data from the associated space. This must be an OPM, user-state program in the user domain. An error is returned if the program is an Integrated Language Environment (ILE) program, or if it is a system-state or system-domain program. The first 10 characters contain the program name, and the second 10 characters contain the library name.

You can use the following special value for the program name:

* Use a program in the call stack. The call stack counter contains the number of programs up the stack from the calling program to look for the program from which to retrieve data.

You can use these special values for the library name:

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

Call stack counter
INPUT; BINARY(4)

A number greater than zero identifying the location in the call stack for the program if * is specified for the program name. This number is relative to the program that called this API. This parameter is ignored if the program name is not *.

Data handle
INPUT; CHAR(16)

The identifier to retrieve the information from in the associated space. Specify the data handle that was previously used on the Store Program Associated Space (QCLSPGAS) API to store

information in the associated space. If no data was previously stored at this data handle, no error is returned. The length of data available parameter will be set to 0 indicating no data was available.

Length of data available
OUTPUT; BINARY(4)

Either the length of the data actually returned or the length of data available.

The possible values follow:

0	No data was previously stored at the data handle specified.
From 1 to the value of the length of output data buffer parameter	The length of information available and successfully returned in the output data buffer.
Greater than the value of the length of output data buffer parameter	The total length of data available. If the output data buffer is too small to hold all of the information available, the information is truncated to fit the available space.

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
CPF0301 E	Length of data buffer is not valid.
CPF0302 E	Value for call stack counter not valid.
CPF0304 E	Operation not allowed for program &1 in library &2.
CPF24B4 E	Severe error while addressing parameter list.
CPF3C90 E	Literal value cannot be changed.
CPF3CF1 E	Error code parameter not valid.
CPF8100 E	All CPF81xx messages could be returned. xx is from 01 to FF.
CPF9803 E	Cannot allocate object &2 in library &3.
CPF9807 E	One or more libraries in library list deleted.
CPF9808 E	Cannot allocate one or more libraries on library list.
CPF9810 E	Library &1 not found.
CPF9811 E	Program &1 in library &2 not found.
CPF9820 E	Not authorized to use library &1.
CPF9821 E	Not authorized to program &1 in library &2.
CPF9830 E	Cannot assign library &1.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

API introduced: V2R3

Top | "Program and CL Command APIs," on page 1 | APIs by category

---

Retrieve Program Information (QCLRPGMI) 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 program name	Input	Char(20)
5	Error Code	I/O	Char(*)

Default Public Authority: *USE
Threadsafe: No

The Retrieve Program Information (QCLRPGMI) API lets you retrieve program information and place it into a single variable in the calling program. The amount of information returned is limited to the size of the variable. This information is the same as the information returned using the Display Program (DSPPGM) command.

You can use the QCLRPGMI API to retrieve the following:

- Program creation information
- Program statistics
- Program performance information
- SQL statement information
- ILE program size information

Authorities and Locks

Library Authority
*EXECUTE

Program Authority
*READ

Program Lock
*SHRRD

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 as long as you specify the length 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 program.

The possible format names are:

"PGMI0100 Format" on page 153	Basic program information for OPM and ILE programs.
"PGMI0200 Format" on page 155	Basic program information for OPM and ILE programs plus SQL statement information for OPM programs.

"PGMI0300 Format" on page 157 ILE program size information.

Qualified program name
INPUT; CHAR(20)

The first 10 characters contain the program name. The second 10 characters contain the name of the library where the program is located.

You can use these special values for the library name:

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

Error code
I/O; CHAR(*)

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

PGMI0100 Format

The following information is returned for the PGMI0100 format. Some of the fields returned are blank or zero if they do not apply to the type of program specified. For detailed descriptions of the fields in the table, see "Field Descriptions" on page 158.

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
Program creation information			
8	8	CHAR(10)	Program name
18	12	CHAR(10)	Program library name
28	1C	CHAR(10)	Program owner
38	26	CHAR(10)	Program attribute
48	30	CHAR(13)	Creation date and time
61	3D	CHAR(10)	Source file name
71	47	CHAR(10)	Source file library name
81	51	CHAR(10)	Source file member name
91	5B	CHAR(13)	Source file updated date and time
104	68	CHAR(1)	Observable information
105	69	CHAR(1)	User profile option
106	6A	CHAR(1)	Use adopted authority
107	6B	CHAR(1)	Log commands
108	6C	CHAR(1)	Allow RTVCLSRC
109	6D	CHAR(1)	Fix decimal data
110	6E	CHAR(50)	Text description
160	A0	CHAR(1)	Type of program
161	A1	CHAR(1)	Teraspace storage-enabled program
162	A2	CHAR(58)	Reserved

Offset		Type	Field
Dec	Hex		
Program statistics information			
220	DC	BINARY(4)	Minimum number of parameters
224	E0	BINARY(4)	Maximum number of parameters
228	E4	BINARY(4)	Program size
232	E8	BINARY(4)	Associated space size
236	EC	BINARY(4)	Static storage size
240	F0	BINARY(4)	Automatic storage size
244	F4	BINARY(4)	Number of MI instructions
248	F8	BINARY(4)	Number of MI ODT entries
252	FC	CHAR(1)	Program state
253	FD	CHAR(14)	Compiler identification
267	10B	CHAR(6)	Earliest release program can run
273	111	CHAR(10)	Sort sequence table name
283	11B	CHAR(10)	Sort sequence table library name
293	125	CHAR(10)	Language identifier
303	12F	CHAR(1)	Program domain
304	130	CHAR(1)	Conversion required
305	131	CHAR(20)	Reserved
Program performance information			
325	145	CHAR(1)	Optimization
326	146	CHAR(1)	Paging pool
327	147	CHAR(1)	Update program automatic storage area (PASA)
328	148	CHAR(1)	Clear program automatic storage area (PASA)
329	149	CHAR(1)	Paging amount
330	14A	CHAR(18)	Reserved
ILE information			
348	15C	CHAR(10)	Program entry procedure module
358	166	CHAR(10)	Program entry procedure module library
368	170	CHAR(30)	Activation group attribute
398	18E	CHAR(1)	Observable information compressed
399	18F	CHAR(1)	Run-time information compressed
400	190	CHAR(6)	Release program created on
406	196	CHAR(1)	Shared activation group
407	197	CHAR(1)	Allow update
408	198	BINARY(4)	Program CCSID
412	19C	BINARY(4)	Number of modules
416	1A0	BINARY(4)	Number of service programs
420	1A4	BINARY(4)	Number of copyrights
424	1A8	BINARY(4)	Number of unresolved references
428	1AC	CHAR(6)	Release program created for

Offset		Type	Field
Dec	Hex		
434	1B2	CHAR(1)	Allow static storage reinitialization
435	1B3	CHAR(1)	All creation data
436	1B4	CHAR(1)	Allow bound *SRVPGM library name update
437	1B5	CHAR(10)	Profiling data
447	1BF	CHAR(1)	Teraspace storage enabled modules
448	1C0	CHAR(1)	Storage model
449	1C1	CHAR(87)	Reserved

PGMI0200 Format

The following information is returned for the PGMI0200 format. Some of the fields returned are blank or zero if they do not apply to the type of program specified. For detailed descriptions of the fields in the table, see "Field Descriptions" on page 158.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
Program creation information			
8	8	CHAR(10)	Program name
18	12	CHAR(10)	Program library name
28	1C	CHAR(10)	Program owner
38	26	CHAR(10)	Program attribute
48	30	CHAR(13)	Creation date and time
61	3D	CHAR(10)	Source file name
71	47	CHAR(10)	Source file library name
81	51	CHAR(10)	Source file member name
91	5B	CHAR(13)	Source file updated date and time
104	68	CHAR(1)	Observable information
105	69	CHAR(1)	User profile option
106	6A	CHAR(1)	Use adopted authority
107	6B	CHAR(1)	Log commands
108	6C	CHAR(1)	Allow RTVCLSRC
109	6D	CHAR(1)	Fix decimal data
110	6E	CHAR(50)	Text description
160	A0	CHAR(1)	Type of program
161	A1	CHAR(1)	Teraspace storage-enabled program
162	A2	CHAR(58)	Reserved
Program statistics information			
220	DC	BINARY(4)	Minimum number of parameters
224	E0	BINARY(4)	Maximum number of parameters

Offset		Type	Field
Dec	Hex		
228	E4	BINARY(4)	Program size
232	E8	BINARY(4)	Associated space size
236	EC	BINARY(4)	Static storage size
240	F0	BINARY(4)	Automatic storage size
244	F4	BINARY(4)	Number of MI instructions
248	F8	BINARY(4)	Number of MI ODT entries
252	FC	CHAR(1)	Program state
253	FD	CHAR(14)	Compiler identification
267	10B	CHAR(6)	Earliest release program can run
273	111	CHAR(10)	Sort sequence table name
283	11B	CHAR(10)	Sort sequence table library name
293	125	CHAR(10)	Language identifier
303	12F	CHAR(1)	Program domain
304	130	CHAR(1)	Conversion required
305	131	CHAR(20)	Reserved
Program performance information			
325	145	CHAR(1)	Optimization
326	146	CHAR(1)	Paging pool
327	147	CHAR(1)	Update program automatic storage area (PASA)
328	148	CHAR(1)	Clear program automatic storage area (PASA)
329	149	CHAR(1)	Paging amount
330	14A	CHAR(18)	Reserved
Program SQL information			
348	15C	BINARY(4)	Number of SQL statements
352	160	CHAR(18)	Relational database
370	172	CHAR(10)	Commitment control
380	17C	CHAR(10)	Allow copy of data
390	186	CHAR(10)	Close SQL cursor
400	190	CHAR(10)	Naming convention
410	19A	CHAR(10)	Date format
420	1A4	CHAR(1)	Date separator
421	1A5	CHAR(10)	Time format
431	1AF	CHAR(1)	Time separator
432	1B0	CHAR(10)	Delay PREPARE
442	1BA	CHAR(10)	Allow blocking
ILE information			
452	1C4	CHAR(10)	Program entry procedure module
462	1CE	CHAR(10)	Program entry procedure module library
472	1D8	CHAR(30)	Activation group attribute
502	1F6	CHAR(1)	Observable information compressed

Offset		Type	Field
Dec	Hex		
503	1F7	CHAR(1)	Run-time information compressed
504	1F8	CHAR(6)	Release program created on
510	1FE	CHAR(1)	Shared activation group
511	1FF	CHAR(1)	Allow update
512	200	BINARY(4)	Program CCSID
516	204	BINARY(4)	Number of modules
520	208	BINARY(4)	Number of service programs
524	20C	BINARY(4)	Number of copyrights
528	210	BINARY(4)	Number of unresolved references
532	214	CHAR(6)	Release program created for
538	21A	CHAR(1)	Allow static storage reinitialization
Continuation of program SQL information			
539	21B	CHAR(10)	Default collection name
549	225	CHAR(10)	SQL package name
559	22F	CHAR(10)	SQL package library name
569	239	CHAR(10)	Dynamic user profile
579	243	CHAR(10)	SQL sort sequence table name
589	24D	CHAR(10)	SQL sort sequence table library name
599	257	CHAR(10)	SQL language identifier
609	261	CHAR(10)	Connection method
619	26B	CHAR(1)	Reserved
620	26C	BINARY(4)	SQL path offset
624	270	BINARY(4)	SQL path length
628	274	CHAR(91)	Reserved
Continuation of ILE information			
719	2CF	CHAR(1)	All creation data
720	2D0	CHAR(1)	Allow bound *SRVPGM library name update
721	2D1	CHAR(10)	Profiling data
731	2DB	CHAR(1)	Teraspace storage-enabled modules
732	2DC	CHAR(1)	Storage model
733	2DD	CHAR(87)	Reserved
Program information through offsets			
		CHAR(*)	SQL path

PGMI0300 Format

The following information is returned for the PGMI0300 format. This format is valid only for ILE programs. If an OPM program is specified, no data is returned and an error is returned. For detailed descriptions of the fields in the table, see "Field Descriptions" on page 158.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
ILE program size information			
8	8	CHAR(10)	Program name
18	12	CHAR(10)	Program library name
28	1C	BINARY(4)	Current total program size
32	20	BINARY(4)	Maximum program size
36	24	BINARY(4)	Current number of modules
40	28	BINARY(4)	Maximum number of modules
44	2C	BINARY(4)	Current number of service programs
48	30	BINARY(4)	Maximum number of service programs
52	34	BINARY(4)	Current string directory size
56	38	BINARY(4)	Maximum string directory size
60	3C	BINARY(4)	Current copyright string size
64	40	BINARY(4)	Maximum copyright string size
68	44	BINARY(4)	Current number of auxiliary storage segments
72	48	BINARY(4)	Maximum number of auxiliary storage segments
76	4C	BINARY(4)	Minimum static storage size
80	50	BINARY(4)	Maximum static storage size
84	54	CHAR(4)	Reserved
88	58	BINARY(8)	Minimum static storage size - long
96	60	BINARY(8)	Maximum static storage size - long

Field Descriptions

For more detailed information than that provided in the following field descriptions, refer to documentation for the command that was used to create the program. For information on non-SQL fields, this would normally be one of the following:

- One of the create program (CRTxxxPGM) commands described in the programmer's guide for the language, identified by the xxx in the command name.
- The Create Program (CRTPGM) command.

For information on SQL fields, (this would normally be a command of the form CRTSQLxxx) see DB2 Universal Database for iSeries SQL Programming Concepts. The xxx in the command name identifies the base language (RPG, COBOL, and so on) of the program.

Activation group attribute. The activation group attribute of this ILE program.

Possible values are:

**NEW* A new activation group with the same name as the program name is created when this program is called. The program runs in this activation group.

**DFTACTGRP*	The program uses one of two existing activation groups created when the process is started. One default activation group is reserved for system-state programs. The other default activation group is reserved for user-state programs.
**CALLER*	The program runs in the activation group of the program from which it is called.
activation group name	The name of the activation group in which this program runs. If the activation group already exists when the program is called, the program runs in the existing activation group. If the activation group does not exist when the program is called, a new activation group is created and the program runs in it.
Blank	This program is an OPM program.

All creation data. Whether the ILE program has all creation data and if that data is observable or unobservable. All observable creation data is needed to re-create the program using the Change Program (CHGPGM) command. All creation data (either observable or unobservable) is needed to convert the program during restore.

Possible values are:

0	*NO. Not all of the creation data is present. The creation template of the ILE program object could be missing or at least one of the modules in this program does not have creation data.
1	*YES. The ILE program has all creation data and all of that data is observable.
2	*UNOBS. The ILE program has all creation data but not all of that data is observable.

Allow blocking. Whether blocking is used to improve the performance of certain SQL statements.

Possible values are:

**NONE*	Blocking is not used.
**READ*	Blocking is used for read-only data cursors when running COMMIT(*NONE) and there are no EXECUTE or EXECUTE IMMEDIATE statements.
**ALLREAD*	Blocking is used for all read-only cursors when running COMMIT(*NONE) or COMMIT(*CHG).
Blank	The program does not contain SQL statements or it is an ILE program.

Allow copy of data. Whether a copy of the data can be used in the implementation of an SQL query.

Possible values are:

**NO*	A copy of the data cannot be used.
**YES*	A copy of the data can be used when needed.
**OPTIMIZE*	The system determines whether a copy of the data is used for optimal performance.
Blank	The program does not contain SQL statements or it is an ILE program.

Allow RTVCLSRC. The compiler allowed you to control this attribute through the ALWRTVSRC parameter if this program was created using the Create CL Program (CRTCLPGM) command.

Possible values are:

N	Source for the CL program is not saved with the program (*NO).
Y	Source is saved (*YES).

Source that is saved can be retrieved by using the Retrieve CL Source (RTVCLSRC) command. This information is blank if the program is not a CL program.

Allow static storage reinitialization. Whether program static storage can be reinitialized. The values are valid for ILE programs only.

Possible values are:

Y Program static storage can be reinitialized.
N Program static storage cannot be reinitialized.

Allow bound *SRVPGM library name update. Whether the Update Program (UPDPGM) command is allowed to change the bound *SRVPGM library names on this program. The values are valid for ILE programs only.

Possible values are:

Y The UPDPGM command can specify a library name for the SRVPGMLIB parameter.
N The UPDPGM command cannot specify a library name for the SRVPGMLIB parameter.

Allow update. Whether the Update Program (UPDPGM) command is allowed on this program. The values are valid for ILE programs only.

Possible values are:

Y The UPDPGM command can be run on this program.
N The UPDPGM command cannot be run on this program.

Associated space size. The size (in bytes) of the associated space used by this program.

Automatic storage size. The size (in bytes) of the automatic storage used by this program. This information is blank if the program is an ILE program.

Bytes available. The number of bytes of data available to be returned. All available data is returned if enough space is provided.

Bytes returned. The number of bytes of data returned.

Clear program automatic storage area (PASA). The compiler may have allowed you to control this attribute through the GENOPT parameter of the command used to create the program.

Possible values are:

N Do not clear PASA storage (*NOCLRPASA).
C Clear PASA storage (*CLRPASA).
Blank The program is an ILE program.

*NOCLRPASA reduces the time needed to call the program. However, if a program variable is referred to before it has been set, it may contain unpredictable data.

*CLRPASA increases the time needed to call the program. However, it ensures that if a program variable is referred to before it has been set, it will contain binary zeros instead of unpredictable data.

Close SQL cursor. Specifies when SQL cursors are implicitly closed and SQL-prepared statements are implicitly discarded.

Possible values are:

**ENDPGM*	When the program that contains the SQL statements ends. This value is valid for OPM programs only.
**ENDSQL*	When the last program containing SQL statements ends. This value is valid for OPM programs only.
**ENDJOB*	When the job ends. This value is valid for OPM programs only.
**ENDMOD*	When the module ends. This value is valid for ILE programs only.
**ENDACTGRP*	When the activation group is deleted.
Blank	The program does not contain SQL statements or it is an ILE program.

Commitment control. The level of commitment control that was specified on the SQL precompile.

Possible values are:

**NONE*	No commitment control was specified on the SQL precompile. Uncommitted changes in other jobs can be seen.
**CHG*	Objects referred to in SQL COMMENT ON, CREATE, DROP, GRANT, LABEL ON, and REVOKE statements are locked until the end of the unit of work (transaction). Updated, deleted, and inserted rows (records) are locked until the end of the unit of work. Uncommitted changes in other jobs can be seen.
**CS*	Objects referred to in SQL COMMENT ON, CREATE, DROP, GRANT, LABEL ON, and REVOKE statements are locked until the end of the unit of work (transaction). Updated, deleted, and inserted rows (records) are locked until the end of the unit of work. A row (record) that is selected but not updated is locked until the next row (record) is selected. Uncommitted changes in other jobs cannot be seen.
**ALL*	Objects referred to in SQL COMMENT ON, CREATE, DROP, GRANT, LABEL ON, and REVOKE statements are locked until the end of the unit of work (transaction). All rows selected, updated, deleted, and inserted are locked until the end of the unit of work. Uncommitted changes in other jobs cannot be seen.
Blank	The program does not contain SQL statements or it is an ILE program.

Compiler identification. The licensed program identifier, version, release, and modification level of the compiler. The field has a ppppppbVvRrMm format, where:

ppppppp	The licensed program identifier.
b	A blank character.
Vv	The character V is followed by a 1-character version number.
Rr	The character R is followed by a 1-character release level.
Mm	The character M is followed by a 1-character modification level.

For programs created by the Create Program (QPRCRTPG) API, this field identifies the version of the operating system that the program was created under.

The field may be blank if the program is created without going through a compilation process or if it is an ILE program.

Connection method. The method used for establishing remote connections when running distributed programs.

Special values that can be returned are:

*RUW	Only one connection to a relational database is allowed. Consecutive CONNECT statements result in the previous connection being disconnected before a new connection is established.
*DUW	Connections to several relational databases are allowed. Consecutive CONNECT statements to additional relational databases do not result in disconnection from previous connects. SET CONNECTION can be used to switch between connections. Read-only connections may result.

Blank The program does not contain SQL statements or is an ILE program.

Conversion required. Indicates whether the program has been converted to reduced instruction-set computer (RISC) format or if conversion is still required.

Possible values are:

0 Conversion is not required. The program has already been converted.
1 Conversion is required.

Creation date and time. The date and time the program was created. The creation date and time field is in the CYYMMDDHHMMSS format as follows:

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

Current copyright string size. The ILE program's copyright string size.

Current number of auxiliary storage segments. The number of auxiliary storage segments in this ILE program.

Current number of modules. The number of modules bound into this ILE program.

Current number of service programs. The number of service programs bound to this ILE program.

Current string directory size. The ILE program's string directory size.

Current total program size. The total size of the ILE program, in kilobytes.

Date format. The format used when accessing date-result columns through SQL. All output date fields are returned in this format. For input date strings, the value you specify is used to determine whether the date is a valid format. This information is blank if the program does not contain SQL statements or if it is an ILE program.

Possible values are:

*USA USA format (mm/dd/yyyy).
*ISO International Standards Organization format (yyyy-mm-dd).
*EUR European format (dd.mm.yyyy).
*JIS Japanese Industrial Standard Christian Era (yyyy-mm-dd).
*MDY Month/day/year format (mm/dd/yy).
*DMY Day/month/year format (dd/mm/yy).
*YMD Year/month/day format (yy/mm/dd).
*JUL Julian format (a numeric value from 1 to 365).
blank The program does not contain SQL statements, or it is an ILE program.

Date separator. The separator used when accessing date-result columns. This information is blank if the program does not contain SQL statements or if it is an ILE program. However, the number of SQL statements field should be checked to determine if the program contains SQL statements. This is because a blank may be specified as a separator value.

Default collection name. The collection name used for the unqualified names of tables, views, indexes, and SQL packages.

Possible values are:

**NONE*	There is no default collection name.
Blank	The program does not contain SQL statements or it is an ILE program.

Delay PREPARE. Whether SQL prepare processing can be delayed until the statement is actually used.

Possible values are:

**YES*	Prepare processing can be delayed.
**NO*	Prepare processing cannot be delayed.
Blank	The program does not contain SQL statements or it is an ILE program.

Dynamic user profile. The user profile used for dynamic SQL statements.

The following special values can be returned:

**USER*	Local dynamic SQL statements are run under the profile of the programs user. Distributed dynamic SQL statements are run under the profile of the SQL package's user.
**OWNER*	Local dynamic SQL statements are run under the profile of the programs owner. Distributed dynamic SQL statements are run under the profile of the SQL package's owner.
Blank	The program does not contain SQL statements or it is an ILE program.

Earliest release program can run. The version, release, and modification level of the earliest release the program is allowed to run on. The compiler may have allowed you to control this through the TGTRLS parameter of the command used to create the program. The field has a VvRrMm format, where:

Vv	The character V is followed by a 1-character version number.
Rr	The character R is followed by a 1-character release level.
Mm	The character M is followed by a 1-character modification level.

Fix decimal data. Whether decimal data that is not valid is corrected or an error is signaled. If the System/36 environment is loaded on your system, you can control this attribute through the fix decimal data (FIXDECDTA) parameter of the CRTS36CBL or CRTS36RPG command.

Possible values are:

N	An error is signaled to the program without correcting the data that is not valid (*NO).
Y	Decimal data that is not valid is corrected (*YES).
Blank	The program is an ILE program.

Language identifier. Returns the 3-character language identifier used when the program was compiled. This information is blank if the module does not contain any language identification information.

The following special value can also be returned:

**JOBRUN* The language identifier associated with the job at the time the program is run.

Log commands. The value specified for the LOG parameter of the CRTCLPGM command. This field is meaningful only if the program is a CL program. The possible values are N (*NO), Y (*YES), and J (*JOB). This information is blank if the program is not a CL program.

Maximum copyright string size. The maximum size of the copyright string in an ILE program.

Maximum number of auxiliary storage segments. The maximum number of auxiliary storage segments an ILE program can have.

Maximum number of modules. The maximum number of modules that can be bound into an ILE program.

Maximum number of parameters. The maximum number of parameters that may be received by the program when it is called. A value of -1 is returned if the information is not available.

Maximum number of service programs. The maximum number of service programs that can be bound to an ILE program.

Maximum program size. The maximum size that an ILE program can be, in kilobytes.

Maximum static storage size. The maximum static storage size (in bytes) that this program may need in order to run. A value of 4294967295 will be given if 4 gigabytes (4294967296) or greater is needed. In this case, the *maximum static storage size - long* field should be used instead.

Maximum static storage size - long. The maximum static storage size in bytes that this program may need in order to run.

Maximum string directory size. The maximum size that the string directory can be in an ILE program.

Minimum number of parameters. The minimum number of parameters that is to be received by the program when it is called. A value of -1 is returned if the information is not available.

Minimum static storage size. The minimum static storage size in bytes that this program needs in order to run. A value of 4294967295 will be given if 4 gigabytes (4294967296) or greater is needed. In this case, the *minimum static storage size - long* field should be used instead.

Minimum static storage size - long. The minimum static storage size in bytes that this program needs in order to run.

Naming convention. The convention used for naming objects in SQL statements.

Possible values are:

**SQL* The SQL naming convention is used.
**SYS* The system naming convention is used.
Blank The module does not contain SQL statements or it is an ILE program.

Number of copyrights. The number of copyrights in this ILE program. This field is zero if the program is an OPM program. Do not assume that a value of zero indicates that the program is an OPM program. ILE programs may not have any copyrights, so this value could be zero for an ILE program. Check the type of program field to determine whether the program is an OPM program or an ILE program.

Number of MI instructions. The number of machine interface (MI) instructions used by this program. A value of -1 is returned if the program is not observable. This information is zero if the program is an ILE program.

Number of MI ODT entries. The number of ODT (object definition table) entries for the program. This is the number of program objects declared by the compiler. Program objects include variables, constants, labels, operand lists, and exception descriptions. Typically, one or more ODT entries are used for each variable, constant, and label in your program. Some are used by the compiler for internal purposes. The number of internal ODT entries varies depending on the size and complexity of the program. There is a limit of 32 767 ODT entries in a program. A value of -1 is returned if the program is not observable. This information is zero if the program is an ILE program.

Number of modules. The number of modules bound into this program. This information is zero if the program is an OPM program.

Number of service programs. The number of service programs bound to this program. This information is zero if the program is an OPM program. Do not assume that a value of zero indicates that the program is an OPM program. ILE programs may not have any service programs bound to them, so this value could be zero for an ILE program. Check the type of program field to determine whether the program is an OPM program or an ILE program.

Number of SQL statements. The number of SQL statements contained in the program. This value is zero if the program does not contain SQL statements or if it is an ILE program.

Number of unresolved references. The number of symbols that could not be resolved at Create Program (CRTPGM) command time. This information is always zero if the program is an OPM program. If this is an ILE program, and if all references were resolved at the time the program was created, this value for this field could be zero.

Observable information. Whether the OPM program contains creation data and if that data is observable or unobservable. All observable creation data is needed to re-create the program using CHGPGM. All creation data (either observable or unobservable) is needed to convert the program during restore.

Possible values are:

A	*ALL. The program has all creation data and that data is observable.
N	*NONE. The program does not have all creation data.
U	*UNOBS. The program has all creation data but not all of that data is observable.
Blank	The program is an ILE program.

The observable information for most programs may be removed by using the remove observability (RMVOBS) parameter on the Change Program (CHGPGM) command.

Observable information compressed. Whether the observable information associated with the program is compressed.

Possible values are:

Y	The observable information is compressed.
N	The observable information is not compressed.
Blank	The program is an OPM program.

Optimization. Indicates what was specified on the OPTIMIZE parameter when the program was created or changed.

Possible values are:

N	*NOOPTIMIZE was specified.
O	*OPTIMIZE was specified.
Blank	The program is an ILE program.

Paging amount. The compiler may have allowed you to control this attribute through the GENOPT parameter of the command used to create the program.

Possible values are:

N	Page the program one page at a time (*NOBLOCK).
B	Page the program in eight-page blocks (*BLOCK).

*BLOCK gives better performance in most situations. It is likely that more than one page in the block is referred to before being replaced by some other paging occurring in the storage pool.

*NOBLOCK can give better performance if the other pages that would have been brought in as a block are unlikely to be referred to before being replaced by some other paging.

Paging pool. The paging pool used for the program object. The compiler may have allowed you to control this attribute through the GENOPT parameter of the command used to create the program.

The values returned are:

U	Use the user pool (*USER).
B	Use the base pool (*BASE).
M	Use the machine pool (*MACHINE).

*USER is used by most system programs and all user programs, unless GENOPT(*MACHINE) is specified.

*BASE is used by certain system programs to avoid disturbing the user pool when they need to be paged in. These programs are not used frequently enough to belong in the machine pool. This value will only appear for OPM programs.

*MACHINE is used by a small number of system programs that are so highly used that their pages should remain almost constantly in main storage. The machine pool is intended to be a stable, low paging pool. If user programs page in the machine pool, there may be contention for main storage between system and user programs. This may adversely affect system performance and response times. To prevent paging contention, increase the QMCHPOOL system value with the Change System Value (CHGSYSVAL) command.

Profiling data. Specifies the profiling data attribute for this ILE program.

Possible values are:

*NOCOL	The collection of profiling data is not enabled and profiling data is not applied.
*COL	The collection of profiling data is enabled for at least one module bound into this ILE program. Any applied profiling data has been removed. The QBNLPGMI API, format PGML0100, can be used to determine if a module bound into this ILE program is enabled to collect profiling data.
*APYBLKORD	Block order profiling data has been applied to at least one module bound into this ILE program. The QBNLPGMI API, format PGML0100, can be used to determine if a module bound into this ILE program has block order profiling data applied.

*APYPRCORD	Procedure order profiling data has been applied to this ILE program.
*APYALL	Block order and procedure order profiling data has been applied to this ILE program.
Blank	This program is an OPM program.

Program attribute. The language the program is written in. (For example, the value is CLP for a CL program, and the value is RPG for an RPG program). This field can be blank. (For example, the program was created by the Create Program (QPRCRTPG) API or the program is created by a compilation process internal to IBM).

Program CCSID. The coded character set identifier (CCSID) for this ILE program. This is 65535 for ILE programs. This information is zero if the program is an OPM program.

Program domain. The domain of the program.

Possible values are:

S	The program can be called by system-state programs.
U	The program can be called by user- or system-state programs.

Program entry procedure module. The module name that contains the program entry procedure for this program. This information is blank if the program is an OPM program.

Program entry procedure module library. The library name that contained the module that contained the program entry procedure for this program when the bind was done. This information is blank if the program is an OPM program.

Program library name. The name of the library containing the program.

Program name. The name of the program.

Program owner. The name of the program owner's user profile.

Program size. The size (in bytes) of this program.

Program state. The state of the program.

Possible values are:

I	The program runs under (inherits) the same state as its caller.
S	The program runs as a system-state program.
U	The program runs as a user-state program.

Relational database. The default relational database that was specified on the SQL precompile. A nonblank value other than *LOCAL specifies the name of the relational database to be resolved through the relational database directory.

The following special values can be returned:

**LOCAL*	The program can only access data on the local system.
Blank	The program does not contain SQL statements or it is an ILE program.

Release program created for. This is the release specified on the target release (TGTRLS) parameter of the Create Program (CRTPGM) command. The value specified for the TGTRLS parameter can affect the earliest release value on which the program can run.

Release program created on. The version, release, and modification level of the operating system on which the program was created.

Reserved. An ignored field.

Run-time information compressed. Whether the run-time information associated with the program is compressed.

Possible values are:

Y	The run-time information is compressed.
N	The run-time information is not compressed.
Blank	The program is an OPM program.

Shared activation group. Whether the program runs in a shared activation group.

Y	The activation group is shared.
N	The activation group is not shared.

Sort sequence table library name. The name of the library the sort sequence table is in.

This information is blank if the program does not contain any sort sequence information or a special value was returned for the sort sequence table name.

The following special values can be returned:

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

Sort sequence table name. The name of the sort sequence table used when the program was compiled. This does not apply to SQL statements in the program.

The following special values can be returned:

*HEX	No sort sequence is used.
*JOBRUN	The sort sequence value that is associated with the job at the time the program runs.
*LANGIDSHR	The shared sort sequence for the language identifier is used.
*LANGIDUNQ	The unique sort sequence for the language identifier is used.
Blank	The program does not contain SQL statements or it is an ILE program. **Note:** This sort sequence table does not apply to SQL statements.

Source file library name. The name of the library that contains the source file used to create the program. The field is blank if a source file was not used to create the program or if it is an ILE program.

Source file member name. The name of the member in the source file. The field is blank if a source file was not used to create the program or if it is an ILE program.

Source file name. The name of the source file used to create the program. The field is blank if a source file was not used to create the program or if it is an ILE program.

Source file updated date and time. The date and time the member in the source file was last updated. The field is in the same format as the creation time and date. The field is blank if a source file was not used to create the program or if it is an ILE program.

SQL package library name. The name of the library the SQL package is in.

SQL package name. The name of the SQL package created on the relational database specified on the RDB parameter of the command that created this program.

**NONE*	This is not a distributed program.
Blank	The program does not contain SQL statements or it is an ILE program.

SQL path. The list of libraries used during resolution of functions, procedures, and data types within SQL statements. The list is in the form of repeating library names, each surrounded by double quotes and separated by commas.

SQL path length. The length, in bytes, of the SQL path.

SQL path offset. The offset, in bytes, from the beginning of this format definition to the beginning of the SQL path.

SQL sort language identifier. The 3-character language identifier used when the program was compiled. This information is blank if the program does not contain any language identification information.

The following possible special value can be returned:

**JOBRUN*	The language identifier is the LANGID associated with the job at the time the program is run.

SQL sort sequence table name. The SQL sort sequence table returns the SQL sort sequence table name used when the program was compiled. This information is blank if the program does not contain any SQL sort sequence information or if this is an ILE program.

The following special values can be returned:

*HEX	No SQL sort sequence is used for the SQL statements.
*JOBRUN	The SQL sort sequence is the SRTSEQ value associated with the job at the time the SQL statements within the program are run.
*LANGIDSHR	The shared SQL sort sequence for the language identifier (LANGID) is used for the SQL statements.
*LANGIDUNQ	The unique SQL sort sequence for the language identifier (LANGID) is used for the SQL statements.

SQL sort sequence table library name. The name of the library the SQL sort sequence table is in. This information is blank if the program does not contain any SQL sort sequence information or a special value was returned for the SQL sort sequence table name.

The following special values can be returned:

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

Static storage size. For OPM programs this is the size (in bytes) of the static storage used by the program. For ILE programs this is the maximum amount of static storage (in bytes) that may be needed to run the program. A value of 4294967295 will be given if 4 gigabytes (4294967296) or greater is needed.

In this case, the *maximum static storage size - long* field should be used instead. The *maximum static storage size - long* field is available from the PGMI0300 format. OPM programs will always have less than 4 gigabytes of static storage.

Storage model. Where the automatic and static storage for this program is allocated at run time.

The following values can be returned:

0 *SNGLVL	Automatic and static storage are allocated from single-level storage.
1 *TERASPACE	Automatic and static storage are allocated from teraspace.
2 *INHERIT	Automatic and static storage are allocated from either single-level storage or teraspace, depending on the activation.

Teraspace storage enabled modules. The teraspace storage capability of the modules bound to this program.

Possible values are:

'00'X	No modules bound to this program are teraspace storage enabled.
'80'X	One or more modules bound to this program are teraspace storage enabled. The *PEP module, however, is not teraspace storage enabled.
'C0'X	The *PEP module is teraspace storage enabled, and there may be more modules bound to this program that are teraspace storage enabled.
'E0'X	All modules bound to this program are teraspace storage enabled.

Teraspace storage enabled program. The teraspace storage capability of an OPM program. A program must be teraspace storage enabled to access teraspace storage.

Possible values are:

0	This program is not teraspace storage enabled.
1	This program is teraspace storage enabled.

Text description. The user text, if any, used to briefly describe the program and its function.

Time format. The format used when accessing time-result columns through SQL. All output time fields are returned in this format.

Possible values are:

*USA	USA format (hh:mm a.m. or p.m.).
*ISO	International Standards Organization format (hh.mm.ss).
*EUR	European format (hh.mm.ss).
*JIS	Japanese Industrial Standard Christian Era (hh.mm.ss).
*HMS	Hours/minutes/seconds format (hh:mm:ss).
Blank	The program does not contain SQL statements or it is an ILE program.

Time separator. The separator used when accessing time-result columns. This information is blank if the program does not contain SQL statements or if it is an ILE program. However, the number of SQL statements is checked to determine if the program contains SQL statements. This is because a blank may be specified as a separator value.

Type of program. Whether the program is an ILE program or an OPM program.

Possible values are:

Blank	OPM program
B	ILE program

Update program automatic storage area (PASA). The compiler may have allowed you to control this attribute through the GENOPT parameter of the command used to create the program. This information is blank if the program is an ILE program.

Possible values are:

N	Do not update internal PASA stack information (*NOUPDPASA).
U	Update internal PASA stack information (*UPDPASA).

*NOUPDPASA reduces the time needed to call the program.

*UPDPASA increases the time needed to call the program but is used by some system programs that are dependent on updates of internal PASA stack information.

Use adopted authority. The value specified for the USEADPAUT option on the command used to change the program.

Possible values are:

Y	Uses program adopted authority from previous call levels when this program is running (*YES).
N	Does not use program adopted authority from previous call levels when this program is running (*NO).

User profile option. The value specified for the USRPRF option on the command used to create the program.

Possible values are:

U	The program runs under the current user's user profile (*USER).
O	The program runs under both the current user's and the owner's user profiles (*OWNER).

Error Messages

Message ID	Error Message Text
CPF2150 E	Object information function failed.
CPF2151 E	Operation failed for &2 in &1 type *&3.
CPF24B4 E	Severe error while addressing parameter list.
CPF3CF1 E	Error code parameter not valid.
CPF3C19 E	Error occurred with receiver variable specified.
CPF3C21 E	Format name &1 is not valid.
CPF3C24 E	Length of the receiver variable is not valid.
CPF3C90 E	Literal value cannot be changed.
CPF5CF5 E	&1 in library &2 not bound program.
CPF8122 E	&8 damage on library &4.
CPF8123 E	Damage on object information for library &4.
CPF8129 E	Program &4 in &9 damaged.
CPF9801 E	Object &2 in library &3 not found.
CPF9802 E	Not authorized to object &2 in &3.
CPF9803 E	Cannot allocate object &2 in library &3.

Message ID	Error Message Text
CPF9806 E	Cannot perform function for object &2 in library &3.
CPF9807 E	One or more libraries in library list deleted.
CPF9808 E	Cannot allocate one or more libraries on library list.
CPF9810 E	Library &1 not found.
CPF9811 E	Program &1 in library &2 not found.
CPF9820 E	Not authorized to use library &1.
CPF9821 E	Not authorized to program &1 in library &2.
CPF9830 E	Cannot assign library &1.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

API introduced: V2R2

Top | "Program and CL Command APIs," on page 1 | APIs by category

---

Retrieve Program Interface Information (QBNRPII) 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	Object type	Input	Char(10)
6	Qualified bound module name	Input	Char(20)
7	Error Code	I/O	Char(*)

Default Public Authority: *USE
Threadsafe: Yes

The Retrieve Program Interface Information (QBNRPII) API lets you retrieve program interface information from modules bound into programs or service programs and place it into a single variable in the calling program. The amount of information returned is limited to the size of the variable.

You can use the QBNRPII API to retrieve program interface information such as:

- exported procedures
- parameter information such as definition and usage
- return value information

Authorities and Locks

Library Authority
*EXECUTE

Program / Service Program Authority
*READ

Program / Service Program Lock
*SHRRD

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 program or service program.

The possible format names are:

"RPII0100 Format" on page 174	Program interface information.

Qualified object name
INPUT; CHAR(20)

The first 10 characters contain the name of the program or service program object. The second 10 characters contain the name of the library where the object is located.

You can use these special values for the library name:

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

Object type
INPUT; CHAR(10)

The object type of the object for which interface information is to be returned.

The possible object types are:

**PGM*	Retrieve interface information from a program object.
**SRVPGM*	Retrieve interface information from a service program object.

Qualified bound module name
INPUT; CHAR(20)

The first 10 characters contain the name of the bound module. The second 10 characters contain the name of the library where the module object was located at the time it was bound into the program or service program object.

You can use this special value for the bound module name:

**ALLBNDMOD*	Retrieve interface information for all bound modules for the program object. If **ALLBNDMOD* is specified for the module name, the library name must be blank.

You can use this special value for the library name:

**ANY*	Retrieve interface information for any bound modules matching the specified module name.

Error code
I/O; CHAR(*)

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

RPII0100 Format

The following information is returned for the RPII0100 format. For detailed descriptions of the fields in the table, see "Field Descriptions."

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	CHAR(10)	Object name
18	12	CHAR(10)	Object library
28	1C	CHAR(10)	Object type
38	26	CHAR(2)	Reserved
40	28	BINARY(4)	Offset to first interface entry
44	2C	BINARY(4)	Number of entries
48	30	CHAR(*)	Reserved
These fields repeat for each entry retrieved		BINARY(4)	Offset to next interface entry
		CHAR(10)	Module name
		CHAR(10)	Module library
		BINARY(4)	CCSID of interface information
		BINARY(4)	Type of interface information
		BINARY(4)	Offset to interface information
		BINARY(4)	Length of interface information returned
		BINARY(4)	Length of interface information available
		CHAR(*)	Reserved
		CHAR(*)	Interface information
		CHAR(*)	Reserved

Field Descriptions

Bytes available. The number of bytes of data available to be returned. All available data is returned if enough space is provided.

Bytes returned. The number of bytes of data returned.

CCSID (Coded Character Set ID) of interface information. The coded character set identifier (CCSID) of the interface data.

Interface information. The interface information for the bound module. See the type of interface information field for detailed information on the contents of this field.

Length of interface information available. The number of bytes available to be returned in the Interface information field. All available data will be returned if enough space is provided.

Length of interface information returned. The number of bytes returned in the Interface information field.

Module library. The name of the library where the module object was located at the time it was bound into the program or service program.

Module name. The name of the bound module.

Number of entries. The number of entries retrieved.

Object library. The name of the library where the object was found.

Object name. The name of the object for which interface information was retrieved.

Object type. The object type of the object.

Offset to first interface entry. The offset at which the first interface information entry begins. This value is relative to the start of the receiver variable. If this value is 0, there are no entries available.

Offset to interface information. The offset to the interface information retrieved for this entry. This value is relative to the start of the receiver variable.

Offset to next interface entry. The offset to the next interface information entry. This value is relative to the start of the receiver variable. If this value is 0, there are no more entries available.

Reserved. An unused field.

Type of interface information. The type of interface information returned for the module.

The possible value is:

1 The interface information is Program Call Markup Language (PCML) data. PCML is a tag language used to describe the input and output parameters for service programs and bound programs. See the Program Call Markup Language topic in the Information Center for more information on PCML data.

Error Messages

Message ID	Error Message Text
CPF2150 E	Object information function failed.
CPF2151 E	Operation failed for &2 in &1 type *&3.
CPF24B4 E	Severe error while addressing parameter list.
CPF3CF1 E	Error code parameter not valid.
CPF3C19 E	Error occurred with receiver variable specified.
CPF3C20 E	Error found by program &1.
CPF3C21 E	Format name &1 is not valid.
CPF3C24 E	Length of the receiver variable is not valid.
CPF3C3A E	Value for parameter &2 for API &1 not valid.
CPF8122 E	&8 damage on library &4.
CPF8123 E	Damage on object information for library &4.
CPF8129 E	Program &4 in &9 damaged.
CPF813D E	Service program &4 in &9 damaged.
CPF9801 E	Object &2 in library &3 not found.
CPF9802 E	Not authorized to object &2 in &3.
CPF9803 E	Cannot allocate object &2 in library &3.
CPF9806 E	Cannot perform function for object &2 in library &3.

Message ID	Error Message Text
CPF9807 E	One or more libraries in library list deleted.
CPF9808 E	Cannot allocate one or more libraries on library list.
CPF9810 E	Library &1 not found.
CPF9811 E	Program &1 in library &2 not found.
CPF9820 E	Not authorized to use library &1.
CPF9821 E	Not authorized to program &1 in library &2.
CPF9830 E	Cannot assign library &1.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

« API introduced: V5R4 with PTF

Top | "Program and CL Command APIs," on page 1 | APIs by category

Retrieve Prompt Override (QPTRTVPO) API

Required Parameter Group:

1	Receiver Variable	Output	Char(*)
2	Length of receiver variable	Input	Binary(4)
3	Receiver variable format name	Input	Char(8)
4	Command string	Input	Char(*)
5	Length of command string	Input	Binary(4)
6	Error code	I/O	Char(*)

Default Public Authority: *USE
Threadsafe: Yes. See "Usage Notes" on page 178 for prompt override program considerations.

The Retrieve Prompt Override (QPTRTVPO) API calls the prompt override program for a specified command and returns the prompt override command string from the prompt override program.

Authorities and Locks

Command
*USE

Prompt override program
*EXECUTE

Library *EXECUTE

Required Parameter Group

Receiver variable
OUTPUT; CHAR(*)

The variable that is to receive the information requested. The minimum size of this area is 8 bytes. You can specify the size of this area to be smaller than the format requested as long as you specify the length parameter correctly. The API will not attempt to return more data than the receiver 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 length is 8 bytes.

Receiver variable format name
INPUT; CHAR(8)

The format of the receiver variable. RTVP0100 is the only valid value. For more information, see "RTVP0100 Format."

Command string
INPUT; CHAR(*)

The command string containing the command name and the key parameter values. The command name may be library-qualified. The command string should be in the format in which it would be entered on a command line. Values must be specified for all key parameters.

Length of command string
INPUT; BINARY(4)

The length of the command string. Valid valids are between 1 and 32 702. The length can include trailing blanks but must not include trailing null characters.

Error code
I/O; CHAR(*)

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

RTVP0100 Format

The RTVP0100 format includes information on the contents of the receiver variable. The following table shows how this information is organized. For detailed descriptions of the fields in the list, see "Field Descriptions."

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	CHAR(10)	Prompt override program name
18	12	CHAR(10)	Prompt override program library
28	1C	BINARY(4)	Offset to prompt override command string
32	20	BINARY(4)	Length of prompt override command string
*	*	CHAR(*)	Prompt override command string

Field Descriptions

Bytes available. The length of all data available for the requested format. All available data is returned if enough space is provided.

Bytes returned. The length of all data actually returned. If the receiver variable is not large enough to hold the data, no other data is returned and this value is less than the bytes available.

Length of prompt override command string. The length of the prompt override command string returned by the API. If the prompt override program completed normally, but did not return a command string, this will be 0. If the prompt override program ended in error, this will be -1.

Offset to prompt override command string. Offset to the first byte of the prompt override command string.

Prompt override command string. The command string returned by the prompt override program. The command string will not include the command name, the key parameters, or any other parameters not specified by the prompt override program. If the receiver is not large enough to hold the entire command string, the command string will not be returned, and the Bytes available field will have the size of the receiver value required for the command string.

Prompt override program library name. The name of the library in which the prompt override program was found.

Prompt override program name. The name of the prompt override program that was called to supply the prompt override command string.

Usage Notes

While this API is threadsafe, it should not be used to call a prompt override program that is not threadsafe in a multithreaded job.

Error Messages

CPF0001 E	Error found on &1 command.
CPF24B4 E	Severe error while addressing parameter list.
CPF3C1D E	Length specified in parameter &1 not valid.
CPF3C19 E	Error occurred with receiver variable specified.
CPF3C20 E	Error found by program &1.
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.
CPF6802 E	Error calling prompt override program.
CPF6803 E	Required key parameter not specified.
CPF6804 E	No prompt override program for command.
CPF680A E	Current values could not be retrieved.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

API introduced: V5R1

Top | "Program and CL Command APIs," on page 1 | APIs by category

---

Retrieve Service Program Information (QBNRSPGM) 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 service program name	Input	Char(20)
5	Error Code	I/O	Char(*)

Default Public Authority: *USE
Threadsafe: No

The Retrieve Service Program Information (QBNRSPGM) API lets you retrieve service program information and place it into a single variable in the calling program. The amount of information returned is limited to the size of the variable. This information is similar to the information returned using the Display Service Program (DSPSRVPGM) command.

You can use the QBNRSPGM API to retrieve the following:

- Service program creation information
- Service program statistics
- Service program performance information
- Service program size information

Authorities and Locks

Library Authority
*EXECUTE

Service Program Authority (see note)
*READ

Service Program Lock
*SHRRD

Note: You must have *USE service program authority to retrieve the following fields in the SPGI0100 format:

- Export source file name
- Export source file library name
- Export source file member name

If you attempt to retrieve these fields with *READ service program authority, they are set to blanks. All other fields in the SPGI0100 format require *READ service program authority.

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 service program.

The possible format names are:

"SPGI0100 Format" on page 180	Basic service program information.
"SPGI0200 Format" on page 181	Service program size information.

Qualified service program name
INPUT; CHAR(20)

The first 10 characters contain the service program name. The second 10 characters contain the name of the library where the service program is located.

You can use these special values for the library name:

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

Error code

I/O; CHAR(*)

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

SPGI0100 Format

The following information is returned for the SPGI0100 format. See "Authorities and Locks" on page 179 for the authority needed regarding specific fields. For detailed descriptions of the fields in the table, see "Field Descriptions" on page 182.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
Service program creation information			
8	8	CHAR(10)	Service program name
18	12	CHAR(10)	Service program library name
28	1C	CHAR(10)	Service program owner
38	26	CHAR(10)	Service program attribute
48	30	CHAR(13)	Creation date and time
61	3D	CHAR(10)	Export source file name
71	47	CHAR(10)	Export source file library name
81	51	CHAR(10)	Export source file member name
91	5B	CHAR(30)	Activation group attribute
121	79	CHAR(16)	Current export signature
137	89	CHAR(1)	User profile
138	8A	CHAR(1)	Observable information compressed
139	8B	CHAR(1)	Run-time information compressed
140	8C	BINARY(4)	Service program CCSID
144	90	BINARY(4)	Number of modules
148	94	BINARY(4)	Number of service programs
152	98	BINARY(4)	Number of copyrights
156	9C	CHAR(50)	Text description
206	CE	CHAR(1)	Shared activation group
207	CF	CHAR(1)	Allow update
208	D0	BINARY(4)	Number of unresolved references
212	D4	CHAR(1)	Use adopted authority
213	D5	CHAR(1)	Allow bound *SRVPGM library name update
214	D6	CHAR(10)	Profiling data
224	E0	CHAR(1)	Teraspace storage enabled modules

Offset		Type	Field
Dec	Hex		
225	E1	CHAR(1)	Storage model
226	E2	CHAR(80)	Reserved '00'X
Service program statistics information			
306	132	CHAR(1)	Service program state
307	133	CHAR(1)	Service program domain
308	134	BINARY(4)	Associated space size
312	138	BINARY(4)	Static storage size
316	13C	BINARY(4)	Service program size
320	140	CHAR(6)	Release service program created on
326	146	CHAR(6)	Earliest release service program can run
332	14C	CHAR(6)	Release service program created for
338	152	CHAR(1)	Allow static storage reinitialization
339	153	CHAR(1)	Conversion required
340	154	CHAR(1)	All creation data
341	155	CHAR(91)	Reserved
Service program performance information			
432	1B0	CHAR(1)	Paging pool
433	1B1	CHAR(1)	Paging amount

SPGI0200 Format

The following information is returned for the SPGI0200 format. For detailed descriptions of the fields in the table, see "Field Descriptions" on page 182.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
Service program size information			
8	8	CHAR(10)	Service program name
18	12	CHAR(10)	Service program library name
28	1C	BINARY(4)	Current total service program size
32	20	BINARY(4)	Maximum service program size
36	24	BINARY(4)	Current number of modules
40	28	BINARY(4)	Maximum number of modules
44	2C	BINARY(4)	Current number of service programs
48	30	BINARY(4)	Maximum number of service programs
52	34	BINARY(4)	Current string directory size
56	38	BINARY(4)	Maximum string directory size
60	3C	BINARY(4)	Current copyright string size
64	40	BINARY(4)	Maximum copyright string size

Offset		Type	Field
Dec	Hex		
68	44	BINARY(4)	Current number of auxiliary storage segments
72	48	BINARY(4)	Maximum number of auxiliary storage segments
76	4C	BINARY(4)	Current number of program procedure exports
80	50	BINARY(4)	Maximum number of program procedure exports
84	54	BINARY(4)	Current number of program data exports
88	58	BINARY(4)	Maximum number of program data exports
92	5C	BINARY(4)	Current number of signatures
96	60	BINARY(4)	Maximum number of signatures
100	64	BINARY(4)	Minimum static storage size
104	68	BINARY(4)	Maximum static storage size
108	6C	CHAR(4)	Reserved
112	70	BINARY(8)	Minimum static storage size - long
120	78	BINARY(8)	Maximum static storage size - long

Field Descriptions

For more detailed information than that provided in the following field descriptions, refer to the online help for the Create Service Program (CRTSRVPGM) command.

Activation group attribute. The activation group attribute of this service program.

The possible values are:

**CALLER* — The service program runs in the activation group of the program from which it is called.

activation group name — The name of the activation group in which this service program runs. If the activation group already exists when the service program is called, the service program runs in the existing activation group. If the activation group does not exist when the service program is called, a new activation group is created in which the service program runs.

All creation data. Whether the ILE service program has all creation data and if that data is observable or unobservable. All observable creation data is needed to re-create the service program using the Change Service Program (CHGSRVPGM) command. All creation data (either observable or unobservable) is needed to convert the service program during restore.

Possible values are:

0 — *NO. Not all of the creation data is present. The creation template of the service program object could be missing or at least one of the modules in this service program does not have creation data. The Display Service Program (DSPSRVPGM) command with *MODULE specified as the value for the DETAIL parameter can be used to see whether a module has creation data.

1 — *YES. The ILE service program has all creation data and all of that data is observable.

2 — *UNOBS. The ILE service program has all creation data but not all of that data is observable.

Allow static storage reinitialization. Whether service program static storage can be reinitialized.

The possible values are:

Y Program static storage can be reinitialized.
N Program static storage cannot be reinitialized.

Allow bound *SRVPGM library name update. Whether the Update Service Program (UPDSRVPGM) command is allowed to change the bound *SRVPGM library names on this service program.

Possible values are:

Y The UPDSRVPGM command can specify a library name for the SRVPGMLIB parameter.
N The UPDSRVPGM command cannot specify a library name for the SRVPGMLIB parameter.

Allow update. Whether the Update Service Program (UPDSRVPGM) command is allowed on this service program.

Possible values are:

Y The UPDSRVPGM command can be run on this service program.
N The UPDSRVPGM command cannot be run on this service program.

Associated space size. The size (in bytes) of the associated space used by this service program.

Bytes available. The number of bytes of data available to be returned. All available data is returned if enough space is provided.

Bytes returned. The number of bytes of data returned.

Conversion required. Indicates whether the service program has been converted to reduced instruction-set computer (RISC) format or if conversion is still required.

Possible values are:

0 Conversion is not required. The service program has been converted.
1 Conversion is required.

Creation date and time. The date and time the service program was created. The creation date and time field is in the CYYMMDDHHMMSS format as follows:

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

Current copyright string size. The size of the service program's copyright string.

Current export signature. The current export signature of this service program.

Current number of auxiliary storage segments. The number of auxiliary storage segments in this service program.

Current number of modules. The number of modules bound into this service program.

Current number of program data exports. The number of data items exported from this service program.

Current number of program procedure exports. The number of procedures exported from this service program.

Current number of service programs. The number of service programs bound to this service program.

Current number of signatures. The number of signatures for this service program.

Current string directory size. The service program's string directory size.

Current total service program size. The total size of the service program, in kilobytes.

Earliest release service program can run. The version, release, and modification level of the earliest release on which the service program is allowed to run. The field has a VvRrMm format, where:

Vv The character V is followed by a 1-character version number.
Rr The character R is followed by a 1-character release level.
Mm The character M is followed by a 1-character modification level.

Export source file library name. The name of the library that contains the export source file. This may be blank if:

- No export source file was specified on the CRTSRVPGM command.
- You attempt to retrieve this field with a service program authority of *READ. A service program authority of *USE is required.

Export source file member name. The name of the member in the export source file that was used to create this service program. This may be blank if:

- No export source file was specified on the CRTSRVPGM command.
- You attempt to retrieve this field with a service program authority of *READ. A service program authority of *USE is required.

Export source file name. The name of the export source file that contains the export source file member. This may be blank if:

- No export source file was specified on the CRTSRVPGM command.
- You attempt to retrieve this field with a service program authority of *READ. A service program authority of *USE is required.

Maximum copyright string size. The maximum size of the copyright string in a service program.

Maximum number of auxiliary storage segments. The maximum number of auxiliary storage segments a service program can have.

Maximum number of modules. The maximum number of modules that can be bound into a service program.

Maximum number of program data exports. The maximum number of data items that can be exported by a service program.

Maximum number of program procedure exports. The maximum number of procedures that can be exported by a service program.

Maximum number of service programs. The maximum number of service programs that can be bound to a service program.

Maximum number of signatures. The maximum number of signatures that can be in a service program.

Maximum service program size. The maximum size of a service program, in kilobytes.

Maximum static storage size. The maximum amount of static storage in bytes that may be needed to run the service program. A value of 4294967295 will be given if 4 gigabytes (4294967296) or greater is needed. In this case, the *maximum static storage size - long* field should be used instead.

Maximum static storage size - long. The maximum amount of static storage in bytes that may be needed to run the service program.

Maximum string directory size. The maximum size of the string directory in a service program, in bytes.

Minimum static storage size. The minimum amount of static storage, in bytes, needed to run the service program. A value of 4294967295 will be given if 4 gigabytes (4294967296) or greater is needed. In this case, the *minimum static storage size - long* field should be used instead.

Minimum static storage size - long. The minimum amount of static storage in bytes needed to run the service program.

Number of copyrights. The number of copyrights in this service program.

Number of modules. The number of modules bound into this service program.

Number of service programs. The number of service programs bound to this program.

Number of unresolved references. The number of symbols that could not be resolved at Create Service Program (CRTSRVPGM) command time. This number could be zero if all references were resolved at the time the service program was created.

Observable information compressed. Whether the observable information associated with the service program is compressed.

The possible values are:

Y The observable information is compressed.
N The observable information is not compressed.

Paging amount. The compiler may have allowed you to control this attribute through the GENOPT parameter of the CRTSRVPGM command.

Possible values are:

N Page the program one page at a time (*NOBLOCK).
B Page the program in eight-page blocks (*BLOCK).

*BLOCK gives better performance in most situations. It is likely that more than one page in the block is referred to before being replaced by other paging occurring in the storage pool.

*NOBLOCK can give better performance if the other pages that would have been brought in as a block are unlikely to be referred to before being replaced by other paging.

Paging pool. The paging pool used for the service program object. The compiler may have allowed you to control this attribute through the GENOPT parameter of the CRTSRVPGM command.

The values returned are:

U Use the user pool (*USER).
M Use the machine pool (*MACHINE).

*USER is used by most system service programs and all user service programs unless GENOPT(*MACHINE) is specified.

*MACHINE is used by a few system service programs that are so highly used that their pages should remain almost constantly in main storage. The machine pool is intended to be a stable, low paging pool. If user service programs page in the machine pool, there may be contention for main storage between system and user service programs. This may adversely affect system performance and response times. To prevent paging contention, increase the QMCHPOOL system value with the Change System Value (CHGSYSVAL) command.

Profiling data. The profiling data attribute for this service program.

Possible values are:

**NOCOL* The collection of profiling data is not enabled and profiling data is not applied.
**COL* The collection of profiling data is enabled for at least one module bound into this service program. Any applied profiling data has been removed. The QBNLSPGM API, format SPGL0100, can be used to determine if a module bound into this service program is enabled to collect profiling data.
**APYBLKORD* Block-order profiling data has been applied to at least one module bound into this service program. The QBNLSPGM API, format SPGL0100, can be used to determine if a module bound into this service program has block-order profiling data applied.
**APYPRCORD* Procedure-order profiling data has been applied to this service program.
**APYALL* Block-order and procedure-order profiling data have been applied to this service program.

Release service program created for. This is the release specified on the target release (TGTRLS) parameter of the Create Service Program (CRTSRVPGM) command. The value specified for the TGTRLS parameter can affect the earliest release value on which the program can run.

Release service program created on. The version, release, and modification level of the operating system on which the service program was created. The field has a VvRrMm format, where:

Vv The character V is followed by a 1-character version number.
Rr The character R is followed by a 1-character release level.
Mm The character M is followed by a 1-character modification level.

Reserved. An ignored field.

Run-time information compressed. Whether the run-time information associated with the service program is compressed.

Possible values are:

Y The run-time information is compressed.
N The run-time information is not compressed.

Service program attribute. The language the program is written in (for example, RPG.)

Service program CCSID. The coded character set identifier (CCSID) for this service program. This is 65535 for service programs.

Service program domain. The domain of the service program.

Possible values are:

S The service program can be called by system-state programs.
U The service program can be called by user- or system-state programs.

Service program library name. The name of the library containing the service program.

Service program name. The name of the service program.

Service program owner. The name of the service program owner's user profile.

Service program size. The size (in bytes) of this service program.

Service program state. The state of the service program.

Possible values are:

I The service program runs under (inherits) the same state as its caller.
S The service program can call user- or system-state programs.
U The service program can call user-state programs.

Shared activation group. Whether the service program runs in a shared activation group.

Y The activation group is shared.
N The activation group is not shared.

Static storage size. The maximum amount of static storage in bytes that may be needed to run the service program. A value of 4294967295 will be given if 4 gigabytes (4294967296) or greater is needed. In this case, the *maximum static storage size - long* field should be used. The *maximum static storage size - long* field is available from the SPGI0200 format.

Storage model. Where the automatic and static storage for this service program is allocated at run time.

The following values can be returned:

*0 *SNGLVL* Automatic and static storage are allocated from single-level storage.
*1 *TERASPACE* Automatic and static storage are allocated from teraspace.
*2 *INHERIT* Automatic and static storage are allocated from either single-level storage or teraspace, depending on the activation.

Teraspace storage enabled modules. The teraspace storage capability of the modules bound to this service program.

Possible values are:

'00'X No modules bound to this service program are teraspace storage enabled.
'80'X One or more modules bound to this service program are teraspace storage enabled.
'A0'X All modules bound to this service program are teraspace storage enabled.

Text description. The user text, if any, used to briefly describe the service program and its function.

Use adopted authority. The value specified for the USEADPAUT option on the command used to change the service program.

Possible values are:

Y Uses program adopted authority from previous call levels when this service program is running (*YES).
N Does not use program adopted authority from previous call levels when this service program is running (*NO).

User profile. The value specified for the USRPRF option on the CRTSRVPGM command.

Possible values are:

U The service program runs under the current user's user profile (*USER).
O The service program runs under both the current user's and the owner's user profiles (*OWNER).

Error Messages

Message ID	Error Message Text
CPF2150 E	Object information function failed.
CPF2151 E	Operation failed for &2 in &1 type *&3.
CPF24B4 E	Severe error while addressing parameter list.
CPF3CF1 E	Error code parameter not valid.
CPF3C19 E	Error occurred with receiver variable specified.
CPF3C20 E	Error found by program &1.
CPF3C21 E	Format name &1 is not valid.
CPF3C24 E	Length of the receiver variable is not valid.
CPF3C90 E	Literal value cannot be changed.
CPF8122 E	&8 damage on library &4.
CPF8123 E	Damage on object information for library &4.
CPF813D E	Service program &4 in &9 damaged.
CPF9801 E	Object &2 in library &3 not found.
CPF9802 E	Not authorized to object &2 in &3.
CPF9803 E	Cannot allocate object &2 in library &3.
CPF9806 E	Cannot perform function for object &2 in library &3.
CPF9807 E	One or more libraries in library list deleted.
CPF9808 E	Cannot allocate one or more libraries on library list.
CPF9810 E	Library &1 not found.
CPF9811 E	Program &1 in library &2 not found.
CPF9820 E	Not authorized to use library &1.
CPF9821 E	Not authorized to program &1 in library &2.
CPF9830 E	Cannot assign library &1.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

API introduced: V2R3

Top | "Program and CL Command APIs," on page 1 | APIs by category

---

Scan for String Pattern (QCLSCAN) API

Required Parameter Group:

1	Character string	Input	Char(*)

2	Length of character string	Input	Packed(3,0)
3	Starting position	Input	Packed(3,0)
4	Character pattern	Input	Char(*)
5	Length of character pattern	Input	Packed(3,0)
6	Translate characters	Input	Char((1)
7	Trim trailing blanks	Input	Char((1)
8	Wildcard character	Input	Char(1)
9	Character string result	Output	Packed(3,0)

Default Public Authority: *USE
Threadsafe: No

The Scan for String Pattern (QCLSCAN) API is used to scan a string of characters to see if the string contains a pattern. This function is similar to the scan function supported within source entry utility (SEU) and on the display presented by the Display Spooled File (DSPSPLF) command. In addition, the QCLSCAN API also allows you to specify a 1-byte character in the pattern that matches with any character in the string to be searched, and a start position, which allows you to search the same string more than once.

A typical use of the QCLSCAN API is to allow the work station user to retrieve all records that contain a specified pattern. For example, if the database has records with book titles, the work station user may want to retrieve all those books with the pattern CHICAGO in the title. The work station user enters CHICAGO on the device display. The application program reads the database, calling the QCLSCAN API at least once for each record to test for the pattern. The application program only processes the records that pass the test for the pattern CHICAGO.

Another alternative for this task is using the Open Query File (OPNQRYF) command. If you are searching an entire database member, the OPNQRYF command normally produces faster results. If you are searching a small subset of a member or the file is already open, QCLSCAN normally produces faster results.

Scanning a field can require many lines of code in a high-level language and can cause a significant amount of overhead. Calling the QCLSCAN API and passing it a parameter list may be a simpler and faster way to do the scan.

Authorities and Locks

None.

Required Parameter Group

Character string
INPUT;CHAR(*)

A character field from 1 through 999 characters that contains the string to be scanned for the pattern.

Length of character string
INPUT;PACKED(3,0)

The length of the string to be scanned. If this length is greater than the actual length of the string, unexpected results can occur.

Starting position
INPUT;PACKED(3,0)

The position in the string where the scan is to start. The position must be greater than zero and not greater than the string length. Normally this value is 1. If the same string has multiple sets of patterns, this allows the string to remain the same while only the start position is varied to find the additional patterns.

Character pattern
INPUT;CHAR(*)

The pattern being scanned for.

Length of character pattern
INPUT;PACKED(3,0)

The length of the pattern. If this length is greater than the actual length of the pattern, unexpected results can occur.

Translate characters
INPUT;CHAR(1)

A variable that indicates to translate lowercase characters in the specified character string to uppercase characters. If this field contains a 1, the program translates lowercase characters of the string to uppercase before the scan using the coded character set identifier (CCSID) for the current job. If the translation cannot be done using the CCSID for the job, *CCSID37 is used. This does not change the user's data. Note that if 1 is specified and the pattern contains lowercase characters, a match never occurs. If 1 is specified, and the data to be searched contains noncharacter data (for example, packed or binary), unexpected results can occur.

Trim trailing blanks
INPUT;CHAR(1)

A fixed-length pattern field filled (left-justified) by a variable number of characters. If this variable contains a 1, trailing blanks are trimmed from the end of the pattern before the scan is started.

Wildcard character
INPUT;CHAR(1)

A variable that you can specify in the pattern, in positions that should not be tested when scanning for a match. When this character appears in the pattern, any character in the data is considered a match. A value of blank indicates that all characters of the pattern take part in the scan. If the wildcard character is the first character in the pattern, an error will occur.

Character string result
OUTPUT;PACKED(3,0)

The value is returned to the user program when the call completes.

If the value returned is positive, the result is the position of the first character of the pattern in the string.

If the value returned is zero, the pattern was not found.

If the value returned is negative, one of the following errors occurred:

-1 The pattern is longer than the string.
-2 The pattern length is less than 1.
-3 The first character of the pattern is a wildcard character.
-4 The pattern is blank and the trim trailing blanks parameter value is 1.
-5 The starting position within the string is not valid.

API existed prior to V3R1

Top | "Program and CL Command APIs," on page 1 | APIs by category

Start Preprocessor (QbnStartPreProcessor) API

Required Parameter:

1	Error code	I/O	Char(*)

Default Public Authority: *USE
Service Program: QBNPREPR
Threadsafe: No

The Start Preprocessor (QbnStartPreProcessor) API must be called first by every compiler preprocessor that has data to be processed during module creation. This API must be called prior to calling the QbnAddAssociatedSpaceEntry, QbnAddExtendedAttributeData, QbnAddPreProcessorLevelData, or QbnAddBindtimeExit API. A call to the QbnEndPreProcessor API is the last API to be called during a single preprocessor call.

Authorities and Locks

None

Required Parameter

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.
CPF3C90 E	Literal value cannot be changed.
CPF3CF1 E	Error code parameter not valid.
CPF5D28 E	Not able to start preprocessing.

API introduced: V3R1

Top | "Program and CL Command APIs," on page 1 | APIs by category

Store Program Associated Space (QCLSPGAS) API

Required Parameter Group:

1	Input data buffer	Input	Char(*)
2	Length of input data buffer	Input	Binary(4)
3	Qualified program name	Input	Char(20)
4	Call stack counter	Input	Binary(4)
5	Data handle	Input	Char(16)
6	Error code	I/O	Char(*)

Default Public Authority: *EXCLUDE
Threadsafe: No

The Store Program Associated Space (QCLSPGAS) API allows you to store information in the associated space of an original program model (OPM), user-state program in the user domain. This API is intended to be used only on programs created by the Create Program (QPRCRTPG) API.

This API's primary use is to store information about a program object for later use. For example, it can be used by a compiler to store information about the compilation process that is needed later at run time. The space associated with a program object is not a replacement for data areas or user spaces. It is recommended for static data only.

There is no capability to delete the information after it is stored. Multiple store requests using the same data handle will write over any information previously stored.

Authorities and Locks

Program Library Authority
*USE

Program Authority
*ALL

Program Lock
*EXCL

Required Parameter Group

Input data buffer
INPUT; CHAR(*)

The information to store in the associated space. This information can contain scalar data only. If it does contain pointer data, the pointer data is not kept.

Length of input data buffer
INPUT; BINARY(4)

The length of the data in the input data buffer, in bytes. The maximum size of an associated space is 16MB. An error is returned if the length of the input data and the data already stored in the associated space exceeds 16MB.

Qualified program name
INPUT; CHAR(20)

The program object for which you want to store data in the associated space. This must be an OPM, user-state program in the user domain. An error is returned if the program is an ILE program, or if it is a system-state or system-domain program. The first 10 characters contain the program name, and the second 10 characters contain the library name.

You can use the following special value for the program name:

* A program in the call stack. The call stack counter contains the number of programs up the stack from the calling program to look for the program where the data is to be stored.

You can use the following special values for the library name:

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

Call stack counter
INPUT; BINARY(4)

A number greater than zero identifying the location in the call stack for the program if * is specified for the program name. This number is relative to the program that called this API. This parameter is ignored if the program name is not *.

Data handle

INPUT; CHAR(16)

The identifier to store the information under in the associated space. This identifier can be any unique value you choose to store your information under. You must remember this value if you want to later retrieve this information. Data may have been stored under a data handle. If this API is called to store data under it, the existing data is overwritten with the new data.

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
CPF0301 E	Length of data buffer is not valid.
CPF0302 E	Value for call stack counter not valid.
CPF0303 E	Limit of associated space size exceeded.
CPF0304 E	Operation not allowed for program &1 in library &2.
CPF24B4 E	Severe error while addressing parameter list.
CPF3C90 E	Literal value cannot be changed.
CPF3CF1 E	Error code parameter not valid.
CPF8100 E	All CPF81xx messages could be returned. xx is from 01 to FF.
CPF9803 E	Cannot allocate object &2 in library &3.
CPF9807 E	One or more libraries in library list deleted.
CPF9808 E	Cannot allocate one or more libraries on library list.
CPF9810 E	Library &1 not found.
CPF9811 E	Program &1 in library &2 not found.
CPF9820 E	Not authorized to use library &1.
CPF9821 E	Not authorized to program &1 in library &2.
CPF9830 E	Cannot assign library &1.
CPF9838 E	User profile storage limit exceeded.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

API introduced: V2R3

Top | "Program and CL Command APIs," on page 1 | APIs by category

Exit Programs

These are the Exit Programs for this category.

Command Analyzer Change Exit Program

Required Parameter Group:

1	Change command exit information	Input	Char(*)
2	Replacement command	Output	Char(*)
3	Length of replacement command string	Output	Binary(4)

QSYSINC Member Name: ECACHCMD
Exit Point Name: QIBM_QCA_CHG_COMMAND
Exit Point Format Name: CHGC0100

The Command Analyzer Change exit program is called when the command for which it is registered is processed. The command analyzer calls the Change Command exit program for a command once through the registration facility before performing any of the following steps:

- Prompting the command,
- Performing interparameter checks,
- Calling the validity checking program, and
- Transferring control to the command processing program.

The exit program is not called if the command analyzer was called to syntax check the command, but not run it. The exit point supports one change command exit program for each command.

The exit program is called, but the command cannot be changed if any of the following occur:

- The command was qualified with a specific library name.
- The command has a parameter defined as RTNVAL(*YES), which allows the command processing program (CPP) to return a value to the calling program.
- The command has a parameter defined as either DSPINPUT(*NO) or DSPINPUT(*PROMPT), which does not allow the value to be shown in the prompt display or joblog.
- The command was used in a program running in system state.

If changing the command is allowed, the exit program can change the command string so that the command analyzer processes a command from a different library or a completely different command. It also can change any or all of the parameters specified on the command. When a changed command string is returned to the command analyzer, the command analyzer will not prompt the replaced command or call the validity checking program, any exit programs registered for the QIBM_QCA_RTV_COMMAND exit point, or the command processing program for the replaced command. The command analyzer will start over with the changed command instead of the original one. It will do a full validation of the new command, including prompting if prompting was requested for the replaced command. If the changed command string specifies a different command in a different library and there is an exit program registered for the QIBM_QCA_CHG_COMMAND exit point for the new command, it will be called, but it will not be allowed to change the command. Exit programs registered for the QIBM_QCA_RTV_COMMAND exit point for the new command will be called. If the command is not changed, the command analyzer will continue with normal processing of the original command, including calling any exit programs registered for the QIBM_CA_RTV_COMMAND exit point.

The replaced command string will be logged to the job log only when the original command was logged. For commands entered on a command line, the original command will be logged as a request message, and the substitute will be logged as a command message. This will allow users to retrieve their original command with the Retrieve function key.

If the exit program sends any escape messages to the command analyzer, the message will be left in the job log and ignored by the command analyzer. The command analyzer will continue processing the original command.

Exit programs may not be registered for the following system commands:

- CALLPRC
- » CALLSUBR «
- CHGVAR
- CNLRCV
- COPYRIGHT

- DCL
- DCLF
- DO
- DOFOR
- DOUNTIL
- DOWHILE
- ENDDO
- ENDPGM
- ENDRCV
- ENDSELECT
- » ENDSUBR «
- GOTO
- IF
- ITERATE
- LEAVE
- MONMSG
- OTHERWISE
- PGM
- RCVF
- RETURN
- » RTNSUBR «
- SELECT
- SNDF
- SNDRCVF
- » SUBR «
- TFRCTL
- WAIT
- WHEN

In addition, exit programs may not be registered for these commands:

- CALL command.
- Commands that are restricted to use by the CL compiler when compiling for a previous release.
- Commands in libraries QSYS38 and QUSER38.

If the exit program uses the registered CL command, a recursive loop may occur. Recursive loops may also occur if two or more exit programs use each other's CL commands. For example, if the exit program for CMDA uses CMDB and the exit program for CMDB uses CMDA, a recursive loop will occur.

Authorities and Locks

You must have *ALLOBJ and *SECADM special authorities to register an exit program for the QIBM_QCA_CHG_COMMAND exit point.

Required Parameter Group

Change command exit information
INPUT; CHAR(*)

Information about the command that the command analyzer was called to process.

Replacement command
OUTPUT; CHAR(*)

A string containing the command string that is to be substituted for the one the command analyzer was called to process. This will be ignored if the command is not allowed to be changed. The maximum length of the changed command string is 32000 bytes.

Commands that should not be specified as the replacement command by an exit program include:

- System commands that are used only in CL programs. See the list of system commands (page 194).
- Commands that have a parameter that is used as a return value (RTNVAL(*YES)).
- Commands that are restricted to use by the CL compiler when compiling for a previous release.

If any of these commands are used as replacement commands, the command analyzer will send a diagnostic message, followed by escape message CPF0001, and processing of the command will end.

Length of replacement command string
OUTPUT; BINARY(4)

The length of the replacement command string (0 - 32000) in bytes. If the length is 0, the original command string will be run, and the replacement command will be ignored.

CHGC0100 Format

The following table shows the format of the information supplied to a change command exit program. For a description of the fields in this format, see "Field Descriptions."

Offset		Type	Field
Dec	Hex		
0	0	CHAR(20)	Exit point name
20	14	CHAR(8)	Exit point format name
28	1C	CHAR(10)	Command name
38	26	CHAR(10)	Library name
48	30	CHAR(1)	Change allowed indicator
49	31	CHAR(1)	Prompt indicator
50	32	CHAR(2)	Reserved
52	34	BINARY(4)	Offset to command string
56	38	BINARY(4)	Length of command string
» 60	3C	BINARY(4)	Offset to proxy chain «
» 64	41	BINARY(4)	Number of entries in proxy chain «
		CHAR(*)	Command string
» Proxy commands and libraries. These fields repeat in the order listed.		CHAR(10)	Proxy command name «
		CHAR(10)	Proxy command library name «

Field Descriptions

Change allowed indicator. Whether or not the exit program is allowed to change the command string that will be run. See the conditions where change is not allowed (page 194). Possible values are:

1 The command may be changed.
0 The command may not be changed.

If a replacement command is returned to the command analyzer, it will be ignored.

Command name. The name of the command that is being processed.

Command string. The command string that the command analyzer is processing. The command name will be library qualified, and the parameter values will be enclosed in parentheses and preceded by the keyword names. Any parameter that was defined as DSPINPUT(*NO) or DSPINPUT(*PROMPT) when the command was created will be formatted without the parameter value (for example, "PASSWORD()") to prevent the exit programs from getting passwords and similar secure data. This command string may not be correct syntactically because required parameters may be missing, interparameter checking has not been done, and the validity checking program for the command has not been called.

Exit point name. The name of the exit point that calls the exit program.

Exit point format name. The format name for the Change Command exit program. The possible format name is:

CHGC0100 The format name that is used to supply the exit information.

Length of command string. The length of the command string being processed.

Library name. The name of the library for the command being processed.

» **Number of entries in proxy chain.** The number of proxy commands chained to this command. «

Offset to command string. The offset to the beginning of the command string.

» **Offset to proxy chain.** The offset to the beginning of the proxy chain information. «

Prompt indicator. Whether prompting was requested on the original command string. Possible values are:

0 Prompting was not requested for the original command.
1 Prompting was requested for the original command.

Proxy command name. Name of the proxy command.

Proxy command library name. Name of the proxy command library.

Reserved. Reserved area.

Usage Notes

Registration considerations.

Use the Add Exit Program command (ADDEXITPGM) or API (QUSADDEP, QusAddExitProgram) to register an exit program for a command. You must specify 20 bytes of exit program data. The first 10 characters specify the command name; the second 10 characters specify the library name.

Any exit programs registered for commands in library QSYS also will be called for commands in the secondary language libraries. For example, if an exit program is registered for the DSPJOB command in library QSYS, it will also be called for the DSPJOB command in library QSYS2962.

If you rename the command or the library or move the command to another library, you also must have an exit program registered using the new command and library names.

Only one exit program can be registered for each command for the QIBM_QCA_CHG_COMMAND exit point. If two applications on the same system, however, need to replace the same command, they can do it by registering a single exit program, which replaces the command with one that is qualified with *LIBL instead of replacing it with their own specific commands. The QCARPLCM program in library QSYS may be used to do this. Applications using QCARPLCM must ensure that their application library is at the beginning of the system part of the library list.

Any exit program registered for this exit point must be threadsafe if it will be called in a job that has multiple threads.

Runtime considerations.

This exit point is a command analyzer exit point that is called during the processing of individual commands. This does not imply that command usage by the system or by individual applications will not change or even be eliminated in the future. For example, if some system function, X, uses the CRTLIB CL command, you should not assume that X will always use the CRTLIB command. X's use of CL commands may change without any warning to use an API or some other interface. Therefore, you should not create any dependencies based on the assumption that a specific function is implemented using a CL command.

If a command is qualified with special value *SYSTEM, only library QSYS will be searched for the command. The change exit program will be allowed to change the command.

If a command is qualified with special value *NLVLIBL, only the national language version (NLV) libraries in the library list and QSYS will be searched for the command. The change exit program will be allowed to change the command.

Adopted authority from previous call levels will be used to determine authority to the exit program, but will not be propagated to the exit program. The exit program will have all of the authorities available to the user profile under which the job is currently running; this may be a profile which has been swapped to, rather than the user profile under which a job was started.

All users with at least *USE authority to the command also should have *OBJOPR and *EXECUTE authority to the exit program and *EXECUTE authority to the exit program's library. The command will fail if the user does not have sufficient authority to the exit program.

Exit program introduced: V4R5

Top | "Program and CL Command APIs," on page 1 | APIs by category

Command Analyzer Retrieve Exit Program

Required Parameter Group:

1	Retrieve command exit information	Input	Char(*)

QSYSINC Member Name: ECARTCMD
Exit Point Name: QIBM_QCA_RTV_COMMAND
Exit Point Format Name: RTVC0100

The Command Analyzer Retrieve exit program is called when the command for which it is registered is processed. This program is called by the command analyzer through the registration facility immediately before transferring control to the command processing program. Exit programs will not be called if the command analyzer was called to syntax-check the command without running it. The exit point supports a maximum of ten retrieve command exit programs for each command.

If the exit program sends any escape messages to the command analyzer, the message will be left in the job log and ignored by the command analyzer.

Exit programs may not be registered for the following system commands:

- CALLPRC
- » CALLSUBR «
- CHGVAR
- CNLRCV
- COPYRIGHT
- DCL
- DCLF
- DO
- DOFOR
- DOUNTIL
- DOWHILE
- ENDDO
- ENDPGM
- ENDRCV
- ENDSELECT
- » ENDSUBR «
- GOTO
- IF
- ITERATE
- LEAVE
- MONMSG
- OTHERWISE
- PGM
- RCVF
- RETURN
- » RTNSUBR «
- SELECT
- SNDF
- SNDRCVF
- » SUBR «
- TFRCTL
- WAIT
- WHEN

In addition, exit programs may not be registered for these commands:

- CALL command
- Commands that are restricted to use by the CL compiler when compiling for a previous release.

- Commands in libraries QSYS38 and QUSER38.

If the exit program uses the registered CL command, a recursive loop may occur. Recursive loops also may occur if two or more exit programs use each other's CL commands. For example, if the exit program for CMDA uses CMDB and the exit program for CMDB uses CMDA, a recursive loop will occur.

Authorities and Locks

You must have *ALLOBJ and *SECADM special authorities to register an exit program for the QIBM_QCA_RTV_COMMAND exit point.

Required Parameter Group

Retrieve command exit information
INPUT; CHAR(*)

Information about the command that the command analyzer was called to process.

RTVC0100 Format

The following table shows the format of the information supplied to a retrieve command exit program. For a description of the fields in this format, see "Field Descriptions."

Offset		Type	Field
Dec	Hex		
0	0	CHAR(20)	Exit point name
20	14	CHAR(8)	Exit point format name
28	1C	CHAR(10)	Command name
38	26	CHAR(10)	Library name
48	30	CHAR(4)	Reserved
52	34	BIN(4)	Offset to original command string
56	38	BIN(4)	Length of original command string
60	3C	BIN(4)	Offset to replacement command string
64	40	BIN(4)	Length of replacement command string
» 68	44	BINARY(4)	Offset to proxy chain «
» 72	48	BINARY(4)	Number of entries in proxy chain «
		CHAR(*)	Original command string
		CHAR(*)	Replacement command string
» Proxy commands and libraries. These fields repeat in the order listed.		CHAR(10)	Proxy command name
		CHAR(10)	Proxy command library name «

Field Descriptions

Command name. The name of the command that is being processed.

Exit point format name. The format name for the Retrieve Command exit program. The possible format name is:

RTVC0100 The format name that is used to supply the exit information.

Exit point name. The name of the exit point that calls the exit program (QIBM_QCA_RTV_COMMAND).

Length of original command string. The length of the original command string being processed.

Length of replacement command string. The length of the replacement command string from the user exit program that was called at exit point QIBM_QCA_CHG_COMMAND. This will be 0 if there is no exit program for exit point QIBM_QCA_CHG_COMMAND or if the exit program did not change the command.

Library name. The name of the library of the command being processed.

» **Number of entries in proxy chain.** The number of proxy commands chained to this command. «

Offset to original command string. The offset to the beginning of the original command string.

» **Offset to proxy chain.** The offset to the beginning of the proxy chain information. «

Offset to replacement command string. The offset to the beginning of the replacement command string. This will be 0 if there is no exit program for exit point QIBM_QCA_CHG_COMMAND or if the exit program did not change the command.

Original command string. The command string that was originally submitted to the command analyzer for processing. The command name will be library qualified and the parameter values will be enclosed in parentheses and preceded by the keyword names. Any parameter that has DSPINPUT(*NO) or DSPINPUT(*PROMPT) will be formatted without the parameter value (for example, ″PASSWORD()″) to prevent exit programs from getting passwords and similar secure data. If the original command string was replaced by an exit program called at the QIBM_QCA_CHG_COMMAND exit point, this may not be syntactically correct because required parameters may be missing, interparameter checks have not been done, and the validity checking program has not been called.

Replacement command string. The replacement command string from the user exit program that was called at exit point QIBM_QCA_CHG_COMMAND. The command name will be library qualified and the parameter values will be enclosed in parentheses and preceded by the keyword names.

Reserved. Reserved area.

Usage Notes

Registration considerations.

Use the Add Exit Program command (ADDEXITPGM) or the Add Exit Program (OPM, QUSADDEP; ILE, QusAddExitProgram) API to register an exit program for a command. You must specify 20 bytes of exit program data. The first 10 characters specify the command name; the second 10 characters specify the library name. Any exit programs registered for commands in QSYS also will be called for commands in the secondary language libraries. For example, if an exit program is registered for the DSPJOB command in library QSYS, it will also be called for the DSPJOB command in library QSYS2962.

If you rename the command or the library or move the command to another library, you must have an exit program registered for the new command and library names.

Any exit program registered for this exit point must be threadsafe if it will be called in a job that has multiple threads.

This exit point is a command analyzer exit point that is called during the processing of individual commands. This does not imply that command usage by the system or by individual applications will not change or even be eliminated in the future. For example, if some system function, X, uses the CRTLIB CL command, you should not assume that X will always use the CRTLIB command. X's use of CL

commands may change without any warning to use an API or some other interface. Therefore, you should not create any dependencies based on the assumption that a specific function is implemented using a CL command.

Runtime considerations.

If a command is qualified with special value *SYSTEM, only library QSYS will be searched for the command.

If a command is qualified with special value *NLVLIBL, only the national language version (NLV) libraries in the library list and QSYS will be searched for the command.

Adopted authority from previous call levels will be used to determine authority to the exit program, but will not be propagated to the exit program. The exit program will have all of the authorities available to the user profile under which the job is currently running; this may be a profile which has been swapped to, rather than the user profile under which a job was started.

All users with at least *USE authority to the command should also have *OBJOPR and *EXECUTE authority to the exit program and *EXECUTE authority to the exit program's library. The command will fail if the user does not have sufficient authority to the exit program.

Exit program introduced: V4R5

Top | "Program and CL Command 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:

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:

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