

APIs by Example: Exit Points, APIs, and Environment Variables

[Carsten Flensburg](#)

Thu, 07/18/2013 - 8:30am

Control inquiry and reply messages in an interactive job—here's how!



From release to release, IBM adds more [exit points](#) to the IBM i operating system. These exit points let you create exit programs that the system will call every time the event associated with the exit point occurs. Sometimes, however, you need to further granulate control of when to execute an exit program. One option is to employ environment variables at the individual job level. Environment variables provide a global setting to a job so that any program running in the job can access that setting, independent of parameters and call levels.

Although system components create and use some environment variables, you can also define and add environment variables if you need to.

To add and retrieve [environment variables](#) for a job, you use the appropriate environment variables CL commands and APIs. In this article, I'll show you how to do this by demonstrating both methods. (The articles listed in the "Find Out More" section below, provide additional details about environment variables and associated programming techniques.) For the exit points involved, I take advantage of the inquiry handling exit point (QIBM_QMH_HDL_INQEXT) and the reply handling exit point (QIBM_QMH_REPLY_INQ). For instructions on how to create the exit points, see the "How to Compile" section below.

Controlling Inquiry Message Replies

While developing a new utility to extract journal entries and restore those records back into their original file, I encountered the CPA32B2 inquiry message ("Change of file &1 may cause data to be lost"), which raised my attention to the inquiry handling and reply handling exit points. This development effort entailed the use of the new Run SQL Statements (RUNSQL) command in a CL program to execute the SQL statement ALTER TABLE DROP COLUMN, which was necessary to remove one of the fields in the table.

When a job is run interactively, the CPA32B2 message is sent to the job's external program queue; the default inquiry message handling ([INQMGRPY](#)) job attribute of *RQD forces the user to reply to the message. For a batch job, the message is automatically replied to by default. Setting the job's inquiry message handling attribute to *SYSRPLY returns the reply specified in the system reply list entry for the CPA32B2 message (if any is defined).

My objective was to control this behavior in an interactive job by using the reply value "I" for the inquiry message reply, which would drop the column and not prompt the user for a response. To handle inquiry messages in a batch job, I wanted to intercept a default value reply of "C" and replace it with the reply value "I," and this is exactly what the inquiry handling and reply handling exit points can do.

Inquiry Handling Exit Point Parameters

The inquiry handling exit program is called in an interactive job when an inquiry message has been sent to the job's *EXT message queue and the Display Program Messages screen is about to interrupt the job to obtain a reply. The exit program can then send a response to the inquiry message, freeing the interactive user from having to provide a reply. The inquiry handling exit point defines the interface in Figure 1 to its exit program.

Figure 1: Parameters defined by the inquiry handling exit point

Required Parameter Group:

1	Type of call	Input	Binary(4)
2	Qualified message queue name	Input	Char(20)
3	Message key	Input	Char(4)
4	Message identifier	Input	Char(7)

Here, the *Type of call* parameter will always have the value 1—*Inquiry needs reply*. Similarly, the parameter *Qualified message queue name* should always contain the value *EXT; for the current implementation of the exit point, there are no other options. The *Message key* parameter provides the unique identifier of the message that's held in the external message queue. And finally, the *Message identifier* parameter lets you easily evaluate whether the inquiry message being intercepted is relevant to your exit program by interrogating the message ID value. If you decide to intercept the inquiry message, you must then handle it by sending a reply to the inquiry message. To do this, use the Send Reply Message (QMHSNDRM) API to send the reply value of your choice. Doing so prevents the inquiry message from displaying and requiring an answer from the user.

If the exit program successfully intercepts and replies to the inquiry message, an informational message (in this case, CPI2516, "Reply sent by inquiry handling exit program") is placed in the job log of the job in which the inquiry message was issued. To further document the event, CPI2516's second-level message text contains full details about the inquiry message, the exit point, and the exit program.

If the CPA32B2 inquiry message is issued in a batch job, the above technique won't work. To that end, you must use the QIBM_QMH_REPLY_INQ exit point, which calls the registered exit program at the point where the inquiry message received a reply. As I mentioned previously, the message system component's default reply mechanism typically provides the inquiry reply. The CPA32B2 message, for example, results in the reply value C, causing the ALTER TABLE SQL statement to fail. To ensure that the reply value I is returned, you can register an exit program for exit point QIBM_QMH_REPLY_INQ and have the exit program replace the reply value with the default reply.

Reply Handling Exit Point Parameters

Figure 2 shows the parameters that the reply handling exit point employs. (For full details, see the exit point documentation in the "Find Out More" section below.)

Figure 2: Parameters used by the reply handling exit point

Required Parameter Group:			
1	Type of call	Input	Binary(4)
2	Qualified message queue name	Input	Char(20)
3	Message key	Input	Char(4)
4	Message identifier	Input	Char(7)
5	Reply	I/O	Char(*)
6	Length of reply	I/O	Bin(4)
7	CCSID of reply	I/O	Bin(4)
8	Reply action return code	Output	Bin(4)

The *Type of call* parameter defines the reason for calling the exit program. You can see the supported values in Figure 3.

Figure 3: QIBM_QMH_REPLY_INQ's Type of call parameter's supported values

- 0 Reply notification-no action allowed
- 1 Reply validation requested
- 2 Default reply validation requested
- 3 Default reply notification
- 4 Reply rejected notification
- 5 Replaced reply not valid
- 6 Reply replaced notification
- 7 Reply cannot be sent notification

For values 0, 4, 5, and 6, parameter 8 (Reply action return code) is ignored.
For values 4 and 7, parameter 5 (Reply) is blank, and parameters 6 and 7 (Length of reply and CCSID of reply, respectively) are zero.

The *Qualified message queue name* parameter specifies the qualified name of the message queue containing the inquiry message, and the *Message key* parameter identifies the specific inquiry message in that message queue that needs a reply. *Message identifier* holds the seven characters of the message ID if the inquiry message is sent as a predefined message stored in a message file, and it contains blanks if the message is impromptu (e.g., as is the case with the Send Break Message—SNDBRMSG—command).

The *Reply* parameter stipulates the reply value for the processed inquiry message. The exit program uses this parameter to override the provided reply value by replacing the input value with the new value. The next two parameters—*Length of reply* and *CCSID of reply*—are also I/O. Finally, the exit program defines the *Reply action return code* parameter to indicate whether to reject, accept, or replace the reply. Again, please refer to the online exit point documentation for all the details.

When the QIBM_QMH_REPLY_INQ exit program requests to replace the reply value, a CPD2479 diagnostic message (“Reply handling exit program requested to replace a reply value”) is logged in the job that’s generating the inquiry message. Following a successful replacement of the reply value, the CPF2458 diagnostic message (“Reply replaced by reply handling exit program”) is added to the job log. Both messages provide additional information about the event in their second-level message text.

A Clear-Cut Implementation

The exit program implementation for both exit points is straightforward. As I mentioned previously, overriding the CPA32B2 inquiry message should happen only if a certain condition is met—namely, that you’ve defined a specific environment variable for the job in which the inquiry message is issued. For this purpose, I’ve defined the environment variable SQL_VFY_ALTER_IGNORE. If this variable is present and has the value Y, both exit programs will override and change the reply value of the CPA32B2 inquiry message to I so that the ALTER TABLE DROP COLUMN SQL statement can process and complete successfully.

The QIBM_QMH_HDL_INQEXT exit program CBX2611 performs the following steps:

1. Checks whether the type of call parameter is *Inquiry needs reply*.
2. Checks whether the message ID is CPA32B2.
3. Checks whether the environment variable SQL_VFY_ALTER_IGNORE is defined and the value is Y.
4. Checks whether the message was sent by system module QDBCHGFI and procedure VFYALTER.

If these checks are met, it sends the reply message with the value I.

Figure 4 shows the RPG IV code snippets that employ this outline.

Figure 4: The QIBM_QMH_HDL_INQEXT exit program code snippets

```

/Free
If PxTypCall = INQ_NEED_RPY;
  If PxMsgId = 'CPA32B2';
    If GetStrVal( getenv( 'SQL_VFY_ALTER_IGNORE' ) ) = 'Y';
      RcvPgmMsg( RCV0300
                : %Size( RCV0300 )
                : 'RCV0300'
                : '*EXT'
                : *Zero
                : '*ANY'
                : PxMsgKey
                : *Zero
                : '*SAME'
                : ERRC0100
                );

      If ERRC0100.BytAvl = *Zero;
        pSndInf = %Addr( RCV0300.VarDta ) + RCV0300.DtaLenRtn +
                  RCV0300.MsgLenRtn +
                  RCV0300.HlpLenRtn;

        If SndInf.SndPgmNam = 'QDBCHGFI' And
           SndInf.SndModNam = 'QDBCHGFI' And
           SndInf.SndPrcNam = 'VFYALTER';

          SndRpyVal( PxMsgQue_q: PxMsgKey: 'I' );
        EndIf;
      EndIf;
    EndIf;
  EndIf;
/End-Free

```

The QIBM_QMH_REPLY_INQ exit program CBX2612 performs these five steps:

1. Checks whether the type of call parameter is *Default reply notification*.
2. Checks whether the message ID is CPA32B2.
3. Checks whether the reply length is greater than zero and the reply value is C.
4. Checks whether the environment variable SQL_VFY_ALTER_IGNORE is defined and the value is Y.
5. If these checks are met, it replaces the reply value with the value I.

You can see the RPG IV code snippets that implement this outline in Figure 5.

Figure 5: The QIBM_QMH_REPLY_INQ exit program code snippets

```
/Free

If  PxTypCall = DFT_RPY_NTF;

    If  PxMsgId = 'CPA32B2';

        If  PxRpyLen > *Zero And
            %Subst( PxMsgRpy: 1: PxRpyLen ) = SQL_VFY_CNL;

            If  GetStrVal( getenv( 'SQL_VFY_ALTER_IGNORE' ) ) = 'Y';

                PxMsgRpy      = SQL_VFY_IGN;
                PxRpyLen      = %Len( SQL_VFY_IGN );
                PxRpyCcsId    = JOB_CCSID;
                PxRtnActCod    = RPY_ACT_RPL;
            EndIf;
        EndIf;
    EndIf;
EndIf;

/End-Free
```

After successfully creating the exit programs (into library QGPL, in this example), you can add them to their respective exit points by using the following two commands:

```
AddExitPgm  ExitPnt( QIBM_QMH_HDL_INQEXT )
             Format( INQE0100 )
             PgmNbr( *LOW )
             Pgm( QGPL/CBX2611 )

AddExitPgm  ExitPnt( QIBM_QMH_REPLY_INQ )
             Format( RPYI0100 )
             PgmNbr( *LOW )
             Pgm( QGPL/CBX2612 )
```

As indicated by the exit program outlines above, the exit programs will handle the CPA32B2 inquiry message only if you've added the environment variable SQL_VFY_ALTER_IGNORE to the current job and specified a value of Y. To activate the new exit programs for the current job, run the following command just before executing the RUNSQL statement that's generating the CPA32B2 inquiry message:

```
AddEnvVar  EnvVar( SQL_VFY_ALTER_IGNORE )
             Value( Y )
             Level( *JOB )
```

When the RUNSQL statement has completed, immediately deactivate the new exit programs for the current job by using the following command:

```
RmvEnvVar  EnvVar( SQL_VFY_ALTER_IGNORE )
             Level( *JOB )
```

Figure 6 shows an example of how to suppress CPA32B2 in a CL program.

Figure 6: CL program suppressing the CPA32B2 inquiry message

```

Pgm      ( &DtaFil_q )

Dcl      &DtaFil_q   *Char      20
Dcl      &DtaFil     *Char      10  Stg( *Defined )  Defvar( &DtaFil_q   1 )
Dcl      &DtaLib     *Char      10  Stg( *Defined )  Defvar( &DtaFil_q  11 )
Dcl      &SqlStm     *Char     1024

AddEnvVar  EnvVar( SQL_VFY_ALTER_IGNORE )          +
           Value( Y )                               +
           Level( *JOB )                             +
           Replace( *YES )
RcvMsg     MsgType( *LAST )  Rmv( *YES )

ChgVar     &SqlStm      ( 'ALTER TABLE ' *Cat      +
                        &DtaLib      *Tcat '/' *Cat +
                        &DtaFil      *Bcat      +
                        'DROP COLUMN JOESD' )

RunSql     Sql( &SqlStm )  Commit( *NONE )  Naming( *SYS )

RmvEnvVar  EnvVar( SQL_VFY_ALTER_IGNORE )          +
           Level( *JOB )
RcvMsg     MsgType( *LAST )  Rmv( *YES )

EndPgm

```

In addition to the RUNSQL scenario described previously, the CPA32B2 inquiry message might result from a DDS-defined physical file that you've altered via the CHGPF command. CPA32B2 is also issued if the CHGPF command specifies the SRCFILE and SRCMBR parameters, and the source member pointed to modifies the physical file to an extent that's incompatible with the current record format in terms of fields omitted or field attributes changed.

More to Come

The RUNSQL scenario will form the starting point for an upcoming APIs by Example column. In that article, I'll present a new Extract Journal Data (EXTJRNDTA) command to demonstrate the practical use of the techniques discussed here. Until then!

How to Compile

Below, you'll find instructions for creating the exit programs. The following sources are included with the code download associated with this article:

CBX2611: RPGLE—Inquiry Handling Exit Program—Ignore CPA32B2

CBX2612: RPGLE—Reply Handling Exit Program—Ignore CPA32B2

CBX261M: CLPA—Exit programs—Build and Configure

To create the above exit programs, compile and run the CBX261M CL program, following the instructions in the source header. You'll also find compilation instructions in the respective source headers of the individual sources.

Find Out More

[“The CL Corner: New Support for CL Commands Lets You Know When a Command Ends”](#)

i Can blog column: [“IBM i Security Never Sleeps”](#)

IBM i 7.1 Information Center documentation

[Environment Variable APIs](#)

[getenv\(\)—Get Value of Environment Variable](#)

[Inquiry Handling Exit Program](#)

[Job External Message Queue *EXT](#)

[Receive Program Message \(QMHRCPM\) API](#)

[Reply Handling Exit Program](#)

[Send Reply Message \(QMHSNDRM\) API](#)

[Using the System Reply List for i5/OS](#)

Articles at iProDeveloper.com:

“[APIs by Example: Change and Retrieve Command Exit Points—and Command Exit CL Commands](#)” (March 2010)

“[Carsten’s Corner: Analyzing Registered Exit Programs: PRTREGEXIT Revisited](#)” (October 2011)

“[Carsten’s Corner: Customizing Spooled File Panels with Spooled File Actions](#)” (September 2010)

“[Controlling the Action of the System Request Key and the ATTN Key](#)” (September 2005)

“[Query and Change your V5R4 Job Interrupt Status Attribute](#)” (March 2008)

“[Retrieve Environment Variables in CL](#)” (February 2010)

“[Taking Control of the System Request Menu](#)” (October 2005)

“[Use Environment Variables for Special Settings](#)” (April 2005)

Source URL: <http://iprodeveloper.com/application-development/apis-example-exit-points-apis-and-environment-variables>