Table of Contents

QEZSNDMG API

```
     **-- Send message:
     D SndMsg          Pr                  ExtPgm( 'QEZSNDMG' )
     D  MsgTyp                       10a   Const                                 Msg type
     D  DlvMod                       10a   Const                                 Delivery mode
     D  MsgTxt                      494a   Const  Options( *VarSize )            Message text
     D  MsgTxtLen                    10i 0 Const                                 MsgTxtLen
     D  MsgRcv                       10a   Const  Options( *VarSize ) Dim( 299 )User/Wrkstn list
     D  MsgRcvNbr                    10i 0 Const                                 Number of usr/wrk
     D  MsgSntInd                    10i 0                                       Msg sent Ind
     D  FncRqs                       10i 0                                       Function request
     D  Error                     32767a          Options( *VarSize )            Error DS
     D  ShwSndMsgDsp                  1a   Const  Options( *NoPass )             display Y/N
     D  MsgQueNam                    20a   Const  Options( *NoPass )             Message queue
     D  NamTypInd                     4a   Const  Options( *NoPass )             Message Key
     D  CcsId                        10i 0 Const  Options( *NoPass )             Character set ID
```

Figure 1

The procedure prototype looks complex at first glance, but is fairly simple to use. In this example, (figure 2) the API will be used to send an inquiry message to the system operator (QSYSOPR). Based on the code below, an inquiry break message will be sent to the system operator.

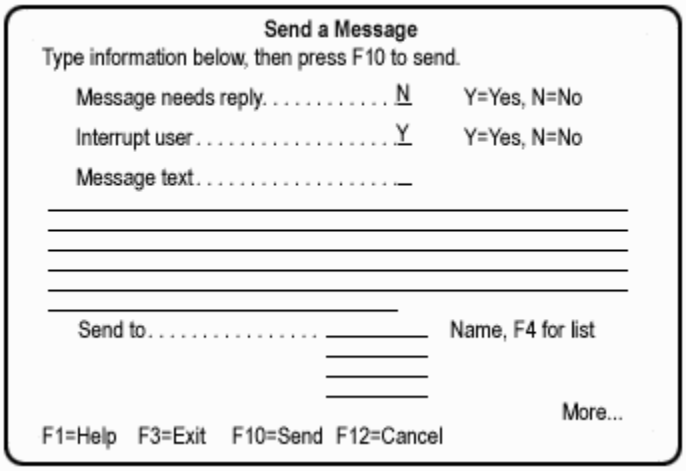
```
sm_MsgTyp    = '*INQ';          // Msg type
sm_DlvMod    = '*BREAK';        // Delivery mode
sm_MsgTxt    = textToSend;      // Message text
sm_MsgTxtLen = %len(sm_MsgTxt); // MsgTxtLen
sm_MsgRcv    = '*SYSOPR';       // User/Wrkstn list
sm_MsgRcvNbr = 1;               // Number of usr/wrk
sm_MsgSntInd = 0;               // Msg sent indicator
sm_FncRqs    = 0;               // Function request
sm_ShwSndMsg = 'N';             // display Y/N
sm_MsgQNam   = *blanks;         // Message queue
sm_MsgKey    = '*USR';          // Message type

SndMsg( sm_MsgTyp
        : sm_DlvMod
        : sm_msgTxt
        : sm_msgTxtLen
        : sm_msgRcv
        : sm_msgRcvNbr
        : sm_MsgSntInd
        : sm_FncRqs
        : ERRC0100
        : sm_shwSndMsg
        : sm_MsgQNam
        : sm_msgKey
        );
```

Figure 2

Since the intent is to send an inquiry message (*INQ) type, one that will require a reply, the delivery mode will be set to break, so the message will interrupt the system operator job activity. Since the actual text to send can vary in length, (up to 494 characters can be specified for the message), the length must be passed. This message will be sent to a single message queue (note: this is a variable length parameter and multiple queues, or *ALL queues may be sent a message.) The number of queues to receive the message is set to 1. The message sent indicator and function requests variables are set to zero. The display message is set to ‘N’.

The display message parameter should be set to 'N' for any batch application. However, in an interactive application, pre-defined text might give way to a spontaneous message sent be the application user. The API can accommodate prompting for message text in the interactive environment by setting the display message parameter to 'Y', which will cause the Send a Message panel to be displayed prior to an message being sent.



(If the message needs reply is set to N, the message will be sent as *INFO, not *INQ.)

The message queue name is blank in this example. Leaving the message queue name blank will cause the message to be sent to the default message queue assigned to the *USR in receipt of the message. (*SYSOPR in general would be assigned to QSYSOPR message queue.)

In this instance no message key is being used—no attempt will be made to retrieve a message by key so the parameter is left blank.

The sample code in Fig. 2 references *QSYSOPR, but the API isn't limited to a single user (or workstation). Up to 299 user profile or display station names may be entered to specify where the message may be sent.

The name type indicator parameter will define whether the names in the list are user profile names or display station names; the default is user profile names. At least one name must be specified if the API is used in a batch job or if the Send a Message display is not to be used.

The message is sent to the user profile or display station message queue. To specify other user message queues, use one of the following special values:

*ALL The message queues of all users. When you use this value, it must be the only item in the list. This value cannot be used if *DSP is specified for the name type indicator parameter.

*ALLACT The message queues of all active users or display stations. This value can be used in combination with specific user profile names or display station names and with *SYSOPR.

*SYSOPR The system operator's message queue, QSYSOPR. This value can be used in combination with specific user profile names and with *ALLACT. It cannot be used if *DSP is specified for the name type indicator parameter.

If the list specifies display station names, the library list will be used to find the work station message queues.

The API error data structure is very generic. In fact, it could serve as a general API error DS, since many of IBM's system API's use the ERRC0100 format data structure. ERRC0200 could be used as well—consult IBM's documentation to determine which best fits for individual application requirements.

```
     **-- Api error data structure:
     D ERRC0100        Ds                  Qualified
     D  BytPro                       10i 0 Inz( %Size( ERRC0100 ))
     D  BytAvl                       10i 0 Inz
     D  MsgId                         7a
     D                                1a
     D  MsgDta                      128a
```

Figure 3

In format ERRC0100, one field is an INPUT field; it controls whether an exception is returned to the application or the error code structure is filled in with the exception information. When the bytes provided field is greater than or equal to 8, the rest of the error code structure is filled in with the OUTPUT exception information associated with the error. When the bytes provided INPUT field is zero, all other fields are ignored and an exception is returned.

QMHRCVM API

```
     D rcvPgmMsg       Pr                  ExtPgm('QMHRCVM')
     D  MsgInfo                    32766A  options(*varsize)                 Message info
     D  pm_Length                     10i 0                                  Len of Msg Info
     D  pm_Format                      8a                                    Format
     D  pm_MQname                     20a                                    Qualified MSQ
     D  pm_MType                      10a                                    Message type
     D  pm_Mkey                        4a                                    Message key
     D  pm_Wait                       10i 0                                  Wait time
     D  pm_Action                     10a                                    Action
     D  pm_Error                   32767a          Options( *VarSize )
```

Figure 4

Since an inquiry message was sent to the system operator, a reply should be forthcoming. The message was not sent to a program message queue, so in order to retrieve a reply, the QMHRCVM API must be used—the API to retrieve non-program queue messages. (Program message queues use a similar, but not interchangeable API.)

```
          * Type definition for the RCVM0100 format

     D MsgInfo         DS                  qualified
     D  BytesRtn                      10I 0
     D  BytesAvail                    10I 0
     D  Severity                      10I 0
     D  MsgID                          7A
     D  Type                           2A
     D  Key                            4A
     D                                 7A
     D  CCSID_st                      10I 0
     D  CCSID                         10I 0
     D  DataLen                       10I 0
     D  DataAvail                     10I 0
     D  Data                        1024A

     **-- Api error data structure:
     D ERRC0100        Ds                  Qualified
     D  BytPro                        10i 0 Inz( %Size( ERRC0100 ))
     D  BytAvl                        10i 0 Inz
     D  MsgId                          7a
     D                                 1a
     D  MsgDta                       128a
```

Figure 5

Two data structures (DS) are going to be used for this process. The first DS (figure 4) will be used to gather the reply information. It corresponds to the layout of the format name RCVM0100, which should be returned to the requesting application. The second DS is the generic error DS—the same DS used for the send message API.

```
D Rm_Format       S              8a   Inz('RCVM0100')
D Rm_Wait         S             10i 0 Inz(-1)
D Rm_Action       S             10a   Inz('*REMOVE')


         ERRC0100.bytPro  = 16    ;          // set error code DS not to use exceptions
          CLEAR msgInfo            ;          // clear return data structure
          Rm_length = %len(MsgInfo);          // message length
          Rm_MQName = *blanks      ;          // message queue
          Rm_MQName = pSDSjobUser  ;          // Message queue to receive
          %subst(Rm_MQName:11)  = '*LIBL';    // qualifier
          Rm_Mtype = '*RPY'        ;          // reply from inquiry message

          RcvPgmMsg( MsgInfo
                     : Rm Length
                     : Rm_format
                     : Rm MQname
                     : Rm_Mtype
                     : Rm_Mkey
                     : Rm_Wait
                     : Rm Action
                     : ERRC0100
                   ) ;
```

Figure 6

Invoking the API is requires the message info DS, (and the length). The format of the DS (in this case RCVM0100) to be returned from the API is supplied. Since a reply is expected (eventually) from the system operator, the message type is set to *RPY.

Note: In the example above the wait-time parameter has been set to -1. This will cause the process to wait indefinitely for a reply to the message. Since an inquiry message has been sent, it is logical to wait for a reply. When using a message key to retrieve a reply or if the message is not a reply type message this parameter should be set to zero. For a *RPY, it could be set to a number of seconds—300 for example to specifically instruct the application to wait up to five minutes for a reply.

A qualified message queue name is specified. The job user (from the program status data structure) is expected to receive the reply, but in this case, *LIBL is used as a qualifier.

The message key is blank, since a message key was not associated with the sent message. The action was *REMOVE.

The generic error DS completes the parameters needed for this API.

```
              if MsgInfo.DataLen < 1;
                 msgReply = *blank;
              else;
                 msgReply = MsgInfo.Data;
              endif;
```

Figure 7

Once the API is called, the message information DS should indicate if a reply was retrieved. In the example above (figure 7), the field MSGREPLY will either be blank (no message returned) or contain the response entered by the system operator.

A messaging procedure, something that could easily reside in an application, or more efficiently in a service program, might make program communication far more flexible than simply showing a CPF message when an error occurs in the application.

```
Select;
   When pStatus = 'U';
      textMessage = 'C, D, update of control record failed.';
      MsgReply =  SndEZbreak(textMessage);
      If MsgReply = 'D';
         Dump(A);
      EndIf;
   When pStatus = 'N';
      textMessage = 'C, R, control record not found.';
      MsgReply =  SndEZbreak(textMessage);
      pStatus = MsgReply;
   Other             ;
      textMessage = %trim(psdsProgName) + ' completed normally ' +
      'for ' + pSType + ' loan type.';
      logmessage(textMessage);
EndSL;
```

Figure 8

The sub-procedure call in the example is a simple example, formatting a text message, up to 132-bytes in length, and receiving a single byte in return. The procedure, SNDEZBREAK, is built over the QEZSNDMG, and QMHRCVM API's . In one case a reply of 'D' will cause the application to execute a dump of the program receiving the reply. In another case, the status is reset to 'R', perhaps to allow the procedure to be retried.

The example is fairly simplistic. Message replies are not limited to a single byte. Up to 1024 characters can be used in a reply. The actions taken based on the reply can be as simple as in the illustration or as complex as needed to satisfy the application requirements. For example, the message reply could be an email address, one used to send an email to a designated account to alert an individual, or a group that some action is required. It could be designed to forward a message from the operator to some other user profile or workstation. The reply could be an instruction to call another procedure, or another application. Using the API moves the possible program actions beyond a simple cancel, or dump of the program.

SNDEZBREAK: Sample Procedure

```
 //*-------------------------------------------------------------------------
 //* SndEZBreak - Send break message to QSYSOPR
 //*
 //*  Parameter       ATR      I/O       Description
 //*  ----------      ---      ---       -----------
 //*  textToSend      132a     I         Text message to send to QSYSOPR
 //*  MsgReply          1a       O       Single character message reply
 //*-------------------------------------------------------------------------

p SndEZBreak...
p                 b
d SndEZBreak...
d                 pi             1a
d TextToSend                   132a   CONST

 **-- variables & constants:
D MsgReply        s               1a
D MsgQueue        s              20a

    * Type definition for the RCVM0100 format

D MsgInfo         DS                   qualified
D  BytesRtn                      10I 0
D  BytesAvail                    10I 0
D  Severity                      10I 0
D  MsgID                          7A
D  Type                           2A
D  Key                            4A
D                                 7A
D  CCSID_st                      10I 0
D  CCSID                         10I 0
D  DataLen                       10I 0
D  DataAvail                     10I 0
D  Data                        1024A

**-- Api error data structure:
D ERRC0100        Ds                   Qualified
D  BytPro                        10i 0 Inz( %Size( ERRC0100 ))
D  BytAvl                        10i 0 Inz
D  MsgId                          7a
D                                 1a
D  MsgDta                       128a

**-- Send message:
D SndMsg          Pr                   ExtPgm( 'QEZSNDMG' )
D  MsgTyp                        10a   Const                                 Msg type
D  DlvMod                        10a   Const                                 Delivery mode
D  MsgTxt                       494a   Const  Options( *VarSize )            Message text
D  MsgTxtLen                     10i 0 Const                                 MsgTxtLen
D  MsgRcv                        10a   Const  Options( *VarSize ) Dim( 299 )User/Wrkstn list
D  MsgRcvNbr                     10i 0 Const                                 Number of usr/wrk
D  MsgSntInd                     10i 0                                       Sent indicator
D  FncRqs                        10i 0                                       Function
D  Error                      32767a          Options( *VarSize )           Error DS
D  ShwSndMsgDsp                  1a   Const  Options( *NoPass )             display Y/N
D  MsgQueNam                     20a   Const  Options( *NoPass )             Message queue
D  NamTypInd                      4a   Const  Options( *NoPass )             Message Key
D  CcsId                         10i 0 Const  Options( *NoPass )             Character set ID

D rcvPgmMsg       Pr                   ExtPgm('QMHRCVM')
D  MsgInfo                    32766A   options(*varsize)                     Message info
```

```
D  pm_Length                    10i 0                                   Len of Msg Info
D  pm_Format                     8a                                     Format
D  pm_MQname                    20a                                     Qualified MSQ
D  pm_MType                     10a                                     Message type
D  pm_Mkey                       4a                                     Message key
D  pm_Wait                      10i 0                                   Wait time
D  pm_Action                    10a                                     Action
D  pm_Error                  32767a          Options( *VarSize )
 *
 *Program message parameters
 *
D Rm_MQname       S             20a
D Rm_Length       S             10i 0
D Rm_MType        S             10a
D Rm_MKey         S              4a   Inz(*Blanks)
D Rm_CSEntry      S             10a
D Rm_Counter      S             10i 0
D Rm_Format       S              8a   Inz('RCVM0100')
D Rm_Wait         S             10i 0 Inz(-1)
D Rm_Action       S             10a   Inz('*REMOVE')
D sm_MsgTyp       S             10a                                     Msg type
D sm_DlvMod       S             10a                                     Delivery mode
D sm_MsgTxt       S            494a                                     Message text
D sm_MsgTxtLen    S             10i 0                                   MsgTxtLen
D sm_MsgRcv       S             10a                                     User list
D sm_MsgRcvNbr    S             10i 0                                   Number of usr/wrk
D sm_MsgSntInd    S             10i 0                                   Sent indicator
D sm_FncRqs       S             10i 0                                   Function request
D sm_ShwSndMsg    S              1a                                     display Y/N
D sm_MsgQNam      S             20a                                     Message queue
D sm_MsgKey       S              4a                                     Message Key
D sm_CcsId        S             10i 0                                   Character set ID

 /Free

   sm_MsgTyp    = '*INQ';          // Msg type
   sm_DlvMod    = '*BREAK';        // Delivery mode
   sm_MsgTxt    = textToSend;      // Message text
   sm_MsgTxtLen = %len(sm_MsgTxt); // MsgTxtLen
   sm_MsgRcv    = '*SYSOPR';       // User/Wrkstn list
   sm_MsgRcvNbr = 1;               // Number of usr/wrk
   sm_MsgSntInd = 0;               // Msg sent indicator
   sm_FncRqs    = 0;               // Function request
   sm_ShwSndMsg = 'N';             // display Y/N
   sm_MsgQNam   = *blanks;         // Message queue
   sm_MsgKey    = '*USR';          // Message type

   SndMsg( sm_MsgTyp
           : sm_DlvMod
           : sm_msgTxt
           : sm_msgTxtLen
           : sm_msgRcv
           : sm_msgRcvNbr
           : sm_MsgSntInd
           : sm_FncRqs
           : ERRC0100
           : sm_shwSndMsg
           : sm_MsgQNam
           : sm_msgKey
           );

    ERRC0100.bytPro  = 16    ;          // set error code DS not to use exceptions
      CLEAR msgInfo            ;          // clear return data structure
      Rm_length = %len(MsgInfo);          // message length
      Rm_MQName = *blanks      ;          // message queue
      Rm_MQName = pSDSjobUser  ;          // Message queue to receive
      %subst(Rm_MQName:11)  = '*LIBL';    // qualifier
      Rm_Mtype = '*RPY'        ;          // reply from inquiry message

    RcvPgmMsg( MsgInfo
               : Rm_Length
```

```
                  : Rm_format
                  : Rm_MQname
                  : Rm_Mtype
                  : Rm_Mkey
                  : Rm_Wait
                  : Rm_Action
                  : ERRC0100
                ) ;

            if MsgInfo.DataLen < 1;
               msgReply = *blank;
            else;
               msgReply = MsgInfo.Data;
            endif;

            msgReply = %subst(msgInfo.Data:1:1);

            Return msgReply;

 /end-free
P
```