9 Messages – Definition, setting, output

Error handling and issuing appropriate messages to the operator play an important role in the programming online applications. The ISPF provides for the fulfillment of this task very good tools. It provides a very good facility for the feedback of errors that occur during the execution of functions. On the other hand, you can very easily use the message output facility of ISPF to inform the user about everything that happens.

The following functions of the ISPF can print out messages:

- From a REXX program using the ISPF command SETMSG.
- From panel by specifying a message ID in plausibility test instructions.
- From the ISPF service commands DISPLAY and TBDISPL.
- Into the ISPF LOG data sets through the ISPF LOG command.

9.1 ERROR HANDLING IN ISPF

Having already discussed the issue of all kinds of reports in the previous chapters, I want to familiarize you with the techniques with which you see errors when running ISPF commands and how you can output error messages. The ISPF provides very sophisticated procedures for reporting errors that occur during the execution of ISPF functions. However, there is a small catch:

Note:

You have to explicitly say at the beginning of each REXX procedure that ISPF should report the errors back to the program after an error occurs. However, this also has a disadvantage: If you forget to check the RC behind any ISPF command, then errors having occurred during its execution, will not inform you about these errors. The program works just wrong. Moreover, you do not know why. For this reason, you must place a query for errors behind every ISPF command in which an error could happen. In addition, you should never forget to insert the command **CONTROL ERRORS RETURN** at the beginning of each program that uses ISPF services.

When an error occurs?:

You should query the RC after each call to an ISPF function. Depending on the function an RC <> 0 may be a bug or not. For example, when the LMGET function reads a sequential data set or the records of a member a RC = 8 returns when the end of data set is reached. In this case of course a RC > 0 is not an error. For other functions, however, such as the LMINIT function, a RC <> 0 always indicates an error.

Returning error messages

The ISPF command CONTROL is used to control the behavior of ISPF errors during execution of an ISPF function. The CONTROL command is used for a variety of purposes. I would like to only explain the specific function to return the error messages at this point. This statement looks like this:

"ISPEXEC CONTROL ERRORS RETURN"

Once you have executed this command, from now on the errors occurring during

The ISPF error messages are reported as follows:

Each ISPF function automatically generates error information when an error occurs (and only then). This information is stored in the ISPF system in two variables. These are the variables:

ZERRSM Short error message text, 24 characters.

ZERRLM Long error message text, 512 characters.

If an error occurs, these variables are written in the function pool. They are usually only present when an error with an RC > 8 occurred during the execution of an ISPF function. The two shaded characters in variable names are very important: S Stands for SHORT. That is, in ZERRSM a short error message with **24** characters will be returned. L stands for LONG. In ZERRLM, extensive error messages with 512 characters will be returned. These variables can be queried in a program and used appropriately for own message output.

Tip:

The variables are always returned in the above-mentioned length. Before using these two variables, you should remove the superfluous BLANKS using the REXX STRIP function.

Output of error messages

The own and the ISPF error messages can be displayed in several ways:

– Using the REXX SAY statement

– Using the ISPF function SETMSG

Method 1: Display error messages with the SAY statement

The method with SAY statements has the disadvantage on an online panel that the currently displayed ISPF panel is completely removed from the screen and then the message is written from top to bottom. As a result, you can often not see which situation caused the error. In addition, the disappearance of the ISPF panels is annoying. An advantage of this method is that the error message also issues when the program runs in a batch job. Moreover, there is no restriction for the length of the message.

Method 2: Output error messages with SETMSG

With SETMSG, you can output error messages and other information so that they will appear in the display of the next panel. This has the distinct advantage that the message will still appear when the creating program (e.g. because of error occurred) was long over.

Comparison of both methods

If we look at the pros and cons of both methods, we soon come to the conclusion that when working online in ISPF, also called in **foreground,** we should always use the SETMSG method. However, since the SETMSG method cannot be used in batch jobs, in this case you must use the SAY method.. In this situation, it is very useful to be able to determine whether an ISPF program runs online in the foreground or in the background as batch job. A very useful REXX function gives us the information in which environment a REXX/ISPF program runs:

if sysvar("SYSENV") = "FORE" then fore = 1; else fore = 0

If you use this function, you always know in which environment a program runs and then you can use the correct method for outputting your error messages. See the following short program:

Program 9.1: TEST2 – Output of messages when a program runs online or in batch

```
/* DOC: TEST2     REXX MAIN                                          */
/* DOC: Output of messages when a program runs online or in batch.   */
/* © FRANZ LANZ 2015                                                 */
/*********************************************************************/
arg text
if text = "" then do  /* Check availability of a parameter */
   zedlmsg = "Required parameter is missing. Program exits!"
   zedsmsg = "Parameter missing"
   zispfrc = 12 /* The RC of the job will be 12 */
end
else do
   zedlmsg = "Entered parameter is: "text
   zedsmsg = "Parameter found"
   zispfrc = 0  /* The RC of the job will be 0  */
end
if sysvar("SYSENV") = "FORE" then "ISPEXEC SETMSG MSG(ISRZ001)"
else do
   "ISPEXEC VPUT (ZISPFRC) SHARED" /* Set the job-RC when run in ISPF*/
   say zedlmsg
end
exit zispfrc  /* Set the job-RC when run in TSO */
```

9.1.3 SETMSG – Set next message :

Function:

When a program is running **online** in ISPF, generally all messages can be displayed with the following command.

MSG	Here the message ID of a programmer-defined message must be specified. There are also pre-defined message IDs, which can be used for the issuance of own messages. See the next chapter.
COND	This parameter indicates that this message is only issued when an earlier SETMSG message is not waiting for output.

9.1.4 Definition of ISPF messages

The message members are stored in the ISPF library **ISPMLIB** .

The definition of ISPF messages is somewhat complicated. Messages are defined in message members, which are used for the output of the messages. This is related; on the one hand, the messages should easily be found by the names of their ID, and on the other hand, not every message should form a member. This results in the need to combine multiple messages in one member. To give you an idea about the structure of message members, I have inserted a member that contains the messages beginning with ISRZ00 here.

Program 9.2: Messages definition member ISRZ00

```
BROWSE    ISP.SISPMENU(ISRZ00)                          Line 00000000 Col 001 080
 Command ===>                                                 Scroll ===> CSR
******************************** Top of Data ********************************
ISRZ000  '&ZEDSMSG                ' .ALARM = NO .HELP = ISR2MACR NOKANA
'&ZEDLMSG'
ISRZ001  '&ZEDSMSG' .ALARM = YES .HELP = ISR2MACR NOKANA
'&ZEDLMSG'
ISRZ002  '&ZERRSM' .ALARM = &ZERRALRM .HELP = &ZERRHM NOKANA
'&ZERRLM'
ISRZ003  '&ZERRSM' .A=&ZERRALRM .H=&ZERRHM .T=&ZERRTP .W=&ZERRWN NOKANA
'&ZERRLM'
/* 5645-001, 5655-042 (C) COPYRIGHT IBM CORP 1980, 1996                    */
```

As you can see, variables can be used in the definitions of the message line. These are the names that begin with an ampersand (&). These variables are replaced through their actual content before outputting the message. In REXX programs, these variables are naturally without specifying & defined.

9.1.5 Naming convention of the ISPF message IDs

When printing a message, the name of the message ID is searched in the members of the data sets of the ISPMLIB chain. This raises the question: What is the name of the member in which a message ID is defined?

Rules:

The name of the member is found by cutting the name of the message ID after the second numeric characters. Examples are shown in the following table:

Rules for building of message members

Message ID	Member names
G015	G01
ISPE241	ISPE24
XYZ123A	XYZ12

Thus all message IDs always start with the name of the member in which they are defined. In addition, the message ID and the associated member name must contain at least two numeric characters.

Searching of message members:

When the following message display command occurs in a program:

"ISPEXEC SETMSG MSG(XYZ123A)"

Then the member **XYZ12** is searched in the allocation chain of the ISPF library ISPMLIB and the message XYZ123A is displayed.

9.1.6 Definition of messages

The chapter for the definition of messages fills in the brochure **ISPF Dialog Developer's Guide and Reference** 10 pages. Since I am not able to demonstrate all the information contained in these pages here, I would like to show an example of the message definition. The message member SSC01 contains the following definition:

```
SSC013B .ALARM=YES .TYPE=ACTION .WINDOW=LNORESP
'As an Expiration Date only PERM, NONE, blank or a date in the ' +
'form YYYY/MM/DD can be entered.'
```

message text	
Long message text	As an Expiration Date only PERM, NONE, blank or a date in the form YYYY/MM/DD can be entered.
Alarm	Yes

This message appears when in the panel SSCPP01 in the field Exp.Date a wrong value is entered. To display the message, the following command is used:

"ISPEXEC SETMSG MSG(SSC013B)"

If a wrong value is entered in Exp.Date, the following message appears on the screen:

```
*-------------------------------------------------------------------------*
| As an Expiration Date only PERM, NONE, blank or a date in the form      |
| YYYY/MM/DD can be entered.                                              |
*-------------------------------------------------------------------------*
```

Format of the message definition:

The definition of a message in a message member is composed of two or more lines.

Coding 9.1: Format of message definition

```
Line 1:
msgid ['short message'][.HELP=panel|*][.ALARM=YES|NO]
[NOKANA|KANA][.WINDOW=RESP|NORESP|LRESP|LNORESP]
[.TYPE=NOTIFY|WARNING|ACTION|CRITICAL]
Line 2:
'long message' [+]

Additional long message text lines are optional

Line 3:
['long message' [+] ]
Line 4:
['long message' [+]  ]
Line n:
['long message'     ]
```

Note:

As you see, you can use a plethora of options here, but do not have to! I have grayed the mandatory parts. If I define a message, I always let the short message text away. This has the advantage that the long message text is displayed immediately without pressing the PF1 key before. If you have defined the short message also, then, when issuing the message, only the short message is displayed initially at top right of the screen, the long message only after pressing the PF1 key.

You will see in the following design, the basic structure of a message member. The name of the member is TT00. There are four messages defined in the member:

Coding 9.2: Member TT00 – Example for the definition of a message member

```
/* DOC: MSGS TT00     ************************************************/
/* DOC: Example for the definition of a message member               */
/* © FRANZ LANZ 2015                                                 */
/*********************************************************************/
TT001
'Message: TT001, Display color: White, Alarm: NO'

TT002 .ALARM=YES
'Message: TT002, Display color: Yellow,  Alarm: YES'

TT003 .ALARM=YES .TYPE=ACTION
'Message: TT003, Display color: Red,   Alarm: YES, ' +
'Continuation line one. ' +
'Continuation line two. ' +
'Continuation line three. '

TT004 'Short message' .ALARM=YES .TYPE=ACTION
'Message: TT004, Display color: Red,   Alarm: YES, Short message: YES'
```

I have outputted all the messages once each with the command ISPEXEC SETMSG MSG (TT00x). See the results below:

```
Message: TT001, Display color: White, Alarm: NO
```

```
Message: TT002, Display color: Yellow,  Alarm: YES
```

```
Message: TT003, Display color: Red, Alarm: YES, Continuation line one.
Continuation line two. Continuation line three.
```

Program 9.3: MSGTEST – Test of messages outputting

```
EDIT       PROX.BOOK.REXX(MSGTEST) - 01.04                  Short message
Command ===>                                              Scroll ===> CSR
****** ***************************** Top of Data ********************************
000001 /* DOC: REXX MSGTEST                                                */
000002 /* DOC: Test of Messages                                           */
000003 /* © FRANZ LANZ 2015                                               */
000004 /*********************************************************************/
000005 "ISPEXEC CONTROL ERRORS RETURN"
000006 "ISPEXEC SETMSG MSG(TT004)"
000007 exit
****** **************************** Bottom of Data ***************************
```

After the **short message** has appeared, I pressed the PF1 key. Then the **Long Message** was displayed at the bottom of the screen.

Tip:

You can move the framed message box with the cursor within the total available display. This is especially helpful if the message field obscures such parts of the surrounding panels that you want to see together with the message text. Moving goes like this: You position the cursor on a point of the frame of the message field and then press ENTER. Then the text **Window move pending** appears in the upper right corner of the screen. Now move the cursor to any position on the screen and press ENTER, the entire message frame moves to the cursor position. You can use this method to make invisible places of the surrounding screen again visible.

9.1.7 The standard message member ISRZ00

Now that you have received an overview of the technology of the message output in ISPF, I will explain how you can easily put out own messages from a REXX program, without having to define an own message member in the ISPMLIB library. In order to offer this possibility, a standard message member is included in ISPF. This member is contained in the ISPF standard message library ISP.SISPMENU and its name is **ISRZ00.** If we remember the rules for the name structure of messages, messages will be stored in the member ISRZ00 that begin with ISRZ00. Let us have a look into this member:

Program 9.4: Standard message member ISRZ00

```
  Menu  Utilities  Compilers  Help
 ------------------------------------------------------------------------------
 ISRBROBA  ISP.SISPMENU(ISRZ00)                          Line 00000000 Col 001 080
 Command ===>                                              Scroll ===> CSR
******************************** Top of Data ********************************
ISRZ000  '&ZEDSMSG                ' .ALARM = NO .HELP = ISR2MACR NOKANA
'&ZEDLMSG'
ISRZ001  '&ZEDSMSG' .ALARM = YES .HELP = ISR2MACR NOKANA
'&ZEDLMSG'
ISRZ002  '&ZERRSM' .ALARM = &ZERRALRM .HELP = &ZERRHM NOKANA
'&ZERRLM'
ISRZ003  '&ZERRSM' .A=&ZERRALRM .H=&ZERRHM .T=&ZERRTP .W=&ZERRWN NOKANA
'&ZERRLM'
/* 5694-A01 (C) COPYRIGHT IBM CORP 1980, 2004                                */
******************************** Bottom of Data ********************************
```

You can use the message IDs contained in this member to print out any message texts as you like. As you can see, four message IDs are defined there. In all these definitions, the message issued texts are specified as a variable. If these variables are filled in the calling program with texts accordingly, then the texts are displayed on the screen when the message is outputted. The use of variables in REXX programs is naturally done without the ampersand.

Remark:

When the variable of the **short message** is not defined or contains a null string, only the **long message** will be outputted immediately.

Description of the individual message IDs:

ISRZ000

Names of variables: ZEDSMSG and ZEDLMSG.

This Message ID is used when you want to spend only a hint. The issue of the long message appears in white.

ISRZ001

Names of variables: ZEDSMSG and ZEDLMSG.

This Message ID is most commonly used to issue error messages. There is an alarm output and the message will appear in yellow.

ISRZ002

Names of the variables:

ZERRSM, ZERRLM, ZERRALRM, ZERRHM, ZERRTP, ZERRWN

Allows you to output the error message data directly, which are returned from ISPF because the ISPF error messages will be stored exactly in the variables which are used in this error message definition. The message appears in yellow. Depending on the content of the variable ZERRALRM, a beep will be outputted. The name of a HELP panel can be stored in the variable ZERRHM. This HELP panel is called when the PF1 key is pressed after the message appears.

ISRZ003

Names of the variables:

ZERRSM, ZERRLM, ZERRALRM, ZERRHM, ZERRTP, ZERRWN

Here you have the same options as with the ID ISRZ002. Additionally, you can

you are interested, I suggest you perform some tests with these variables. I have never needed these variables previously.

Examples from practice programs for using the message member ISRZ001:

Example 1: Excerpt from a REXX program.

```
14 if sysvar("SYSENV") = "FORE" then do
15    zedlmsg = "ISPF error in program >"exname"< RC="rc,
16              "from statement:"copies(" ",60) srcline copies(" ",60)
17    if zerrlm <> "ZERRLM" then,
18       zedlmsg = zedlmsg" ISPF error message:" strip(zerrlm)
19    address "ISPEXEC" "SETMSG MSG(ISRZ001)"
20 end
```

Lines	Explanation
14:	This query is determined that the message is only outputted if the REXX program runs in FOREGROUND online under ISPF.
15+16	The text of the message is stored in the variable ZERRLM.
17+18	If ZERRLM contains an ISPF message, then the ISPF error message will be appended to the existing message text.
19	The message will be outputted and is displayed on the next screen.

Example 2: Excerpt from an edit macro.

```
67 if sysdsn(dsn) = "OK" then do
68    address "ISPEXEC" "EDIT DATASET("dsn")"
69    if rc > 4 then do
70       address "ISPEXEC" "SETMSG MSG(ISRZ002)"
71       "CURSOR = "cl cc
72       exit
73    end
74 end
75 else do
76    zedsmsg = "Data set not found"
77    zedlmsg = "The data set "dsn" could not be found."
78    address "ISPEXEC" "SETMSG MSG(ISRZ001)"
79 end
```

Lines	Explanation
70	Because the ISPF EDIT command stores its messages when an error occurs in the variables ZERRSM and ZERRLM, then the message ID ISRZ002 can be used here.
78	Here, the variables ZEDSMSG and ZEDLMSG are filled with private texts. Therefore, ISRZ001 will be used here.

Remark:

The program from which the above excerpt is derived, is an edit macro. Therefore, the ISPF commands must be called using the form **address "ISPEXEC"**. You cannot use the shape by prefixing the command with ISPEXEC here because the ISPF edit processor would not accept this. The shape with ISPEXEC before an ISPF command only understands the TSO command processor!

Recommended / Playlists / History / Topics / Settings / Get the App / Sign Out

PREV
8 Data set processing using ISPF

NEXT
10 Panels – create and use