ABAP Message Error Handling Standards

ABAP Message/Error Handling Standards

Purpose

The message handling process varies depending on the type of programming, functional requirements, run time environment and modularization. ABAP errors occur for the same reasons that errors occur in other programming languages (e.g., record not found, type mismatch, record locked, etc.)

The following sections detail the proper way to handle certain types of messages in certain situations. The list is not comprehensive and the programmer is responsible for using good professional judgment for the cases that are not covered below.

Messages

- ABAP programs should use the MESSAGE statement to output messages to inform of necessary runtime activity.
- For executable program, the message-id should be defined in the REPORT statement
- Custom messages should be created and maintained as a message class in transaction SE91 by related applications (SD, FI, Interface, Report), or, specific to a program if it does not fit into a general custom application specific message class.
- The message text should be defined as a concise text describing the runtime message. Substrings (&) should be used to identify the variable that is specific for the message.

Example

REPORT z.... MESSAGE-ID z1.

......

IF sy-subrc EQ 0.

    MESSAGE S001 WITH mara-matnr.    "Material & has been updated

ENDIF.

- For large development projects a common message class ‘ZCA’ (cross-application) should be created. This message class should contain all common messages used by add-on programs. Messages within this class can be used with the format:

MESSAGE Ennn(ZCA)


nnn = Message number within ZCOMMON

- The long text option in the message should be used to define the action.

SAP Message Types (format = a)

<table>
<thead>
<tr>
<th>Code</th>
<th>Type</th>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>Informational</td>
<td>Press ENTER to continue</td>
<td>It contains information about operations already performed and can be safely ignored without any consequences.</td>
</tr>
<tr>
<td>W</td>
<td>Warning</td>
<td>Correction possible</td>
<td>It provides information about the consequences of certain actions. These messages cannot be ignored but the user can choose whether or not to make a correction or bypass the message.</td>
</tr>
<tr>
<td>E</td>
<td>Error</td>
<td>Correction required</td>
<td>It contains information about processing errors. The system interrupts the current processing so that the errors can be corrected. Only then can processing continue.</td>
</tr>
<tr>
<td>A</td>
<td>Abend</td>
<td>Transaction terminated</td>
<td>It provides information about processing errors but the processing cannot be resumed.</td>
</tr>
<tr>
<td>X</td>
<td>Exit</td>
<td>Transaction terminated with short dump</td>
<td>It provides no processing information, but rather, a stack dump for the state of the system.</td>
</tr>
<tr>
<td>S</td>
<td>Success</td>
<td>Message on next screen</td>
<td>It contains normal information which is show in screen.</td>
</tr>
</tbody>
</table>
Return Codes

ABAP System Field SY-SUBRC

Messages in ABAP are handled, in most cases, by the system field SY-SUBRC which retains the value of the return code after specific operations such as select, read, translate, etc. Whenever it is possible for a statement to set a return code value, which must be handled to insure proper continuation of the program, SY-SUBRC should be explicitly checked and appropriate action taken.

SY-SUBRC EQ 0 : indicates a successful completion of the statement
SY-SUBRC NE 0 : indicates an error condition for the statement

If no action needs to be taken when SY-SUBRC returns an error condition a comment must be included stating such and the SY-SUBRC check is not being done.

Exception Codes

Exception codes are used to communicate errors from a function module to the calling program. The calling program must handle all possible errors generated by a function by the use of the EXCEPTIONS clause of the CALL FUNCTION statement in conjunction with SY-SUBRC. The use of OTHERS is acceptable to handle errors that do not have specific handling required. Letting error codes "fall through" is not acceptable. This makes it unnecessary to use of the MESSAGE ... RAISING EXCEPTION construct, which should, therefore, be avoided.

When writing a function module, RAISE EXCEPTION should be used to terminate the processing of the function and return an error code to the calling program unless one or more of the EXPORT parameters contains valid information that the caller will require. If RAISE EXCEPTION is invoked in a function module the EXPORT parameters are not filled when control is returned (immediately) to the calling program.

Example:

CALL FUNCTION 'STRING_SPLIT'
  EXPORTING
    DELIMITER = ':'
    STRING = FELD
  IMPORTING
    HEAD = HEAD
    TAIL = TAIL
  EXCEPTIONS
    NOT_FOUND = 1
    OTHERS = 2.
CASE SY-SUBRC.
  WHEN 1 ...
  WHEN 2 ...
  WHEN OTHER.
ENDCASE.

Closing the Spool

E and A level error messages overwrite the print spool and destroy its contents. All write statements are erased and overwritten by the ABEND. To maintain "spool integrity", if desired, insert the following code before each A or E message.

Example:

NEW-PAGE PRINT OFF.
COMMIT WORK. Commit spool data.
Catch System Exceptions

- When possible runtime exceptions should be captured and processed as error using the CATCH ... ENDCATCH or TRY ... CATCH ...ENDTRY statements instead of allowing a program to dump.
- When a runtime exception is trapped a detailed message should be presented to the user indicating the severity of the problem, the cause of the problem, any remedies required to fix the problem, and how to reprocess the ended task

Classical Exceptions

- SYSTEM-EXCEPTION
  - Use an Error Class in the CATCH STATEMENT whenever possible
  - Only use a specific Error ID if the error situation can be very precisely defined
    - Each Error ID or Error Class should be placed on separate lines of the CATCH Statement
    - The CATCH ... ENDCATCH return code check should be done immediately after the ENDCATCH statement and proper message should be issued

Example:

CATCH SYSTEM-EXCEPTIONS ARITHMETIC_ERRORS = 1
    CONVERSION_ERRORS = 2
    COMPUTE_INT_ZERODIVIDE = 3

V_RESULT = 10 / 0.
ENDCATCH.
CASE SY-SUBRC.
  WHEN 1. MESSAGE Ennn(ZCA).
  WHEN 2. MESSAGE Ennn(ZCA).
  WHEN OTHERS. ......
ENDCASE.

Class-Based Exceptions

- The exceptions of all exception classes visible in a program can be triggered with statement RAISE EXCEPTION.
- Class-based exceptions can be declared in the interface of procedures. For local procedures, you use the addition RAISING of the statements METHODS and FORM for it. In the Class and Function Builder, you select exception classes when defining exceptions in the interface. The declared exceptions can occur at the call position of a procedure if the exception is not treated in the procedure.
- If the exception occurs during execution of a TRY block of a TRY control structure, an appropriate CATCH block is searched as a handler. Execution of each TRY block opens a context, also called a protected area, into which the execution of other TRY blocks can be embedded. Starting at the position where the exception occurs, the system scans the TRY control structures of the participating TRY blocks from the inside to the outside for the first CATCH block, in which the exception class or one of its superclasses appears. If it finds a CATCH block, it executes it and continues processing after its TRY control structure.
- If no handlers are found in any of the participating TRY control structures of a protected area, or if the exception does not occur during processing of a TRY block of a TRY control structure, a runtime error occurs at the point where the exception occurred. The short dump of the runtime error contains the name of the exception class and the exception text.
- Statement CLEANUP introduces a statement block of a TRY control structure in which you can carry out cleanup tasks.

Example:

TRY.
  V_RESULT = 10 / 0.
CATCH CX_SY_ZERODIVIDE INTO O_REF.
  MESSAGE Ennn(ZCA).
CLEANUP.
  CLEAR V_RESULT.
ENDTRY.