Application Log: Technical Documentation

The 'Application Log' is a tool for collecting messages, saving, reading and deleting logs in the database, and displaying logs. It is introduced and described in the following documentation.

CONTENTS

Introduction

Situations can arise at runtime in an application program which must be brought to the user's attention. These are usually errors, but it may also be useful to report a successful procedure (although the user should not be inundated with unimportant information).

We will not distinguish between exceptions, errors, messages, etc. here. These are all situations in which a particular piece of information (usually a T100 message) arises, which is displayed in a log either immediately or later. This information is called a message here.

These messages are not output individually (ABAP command MESSAGE), they are collected and displayed together later.

This set of messages is a log. Logs usually also contain general header information (log type, creator, creation time, etc.) Several logs can be created in a transaction.

The Application Log provides a comprehensive infrastructure for collecting messages, saving them in the database and displaying them as logs. This infrastructure and some conventions are described below.

Technical core and simplified shell

The Application Log has two levels: a technical core of small, flexible function modules and a simplified shell, which uses core function modules for particular scenarios.

Example:
The technical core contains function modules to search the database (BAL_DB_SEARCH), load logs into memory (BAL_DB_LOAD) and output logs which are in memory (BAL_DSP_LOG_DISPLAY). The simplified shell contains the function module APPL_LOG_DISPLAY which uses the core function modules in this order to display logs which are in the database.

Both core and simplified shell function modules can be used.

The simplified shell function modules are generally the old Release 3.0 Application Log function modules (beginning with APPL_LOG_…) which internally now call the function modules of the new technical core (beginning with BAL_…).

All function groups are in the development class SZAL, the simplified shell function groups begin with SLG…, those of the technical core with SBAL…

If you already use the old Application Log, you need not change, but only some of the new Application Log functionality is available in the simplified old shell.

Example:
The (old) function module to display logs which are in the database, APPL_LOG_DISPLAY, has the new Importing parameter I_S_DISPLAY_PROFILE, which describes the log display format (see section Display log). This parameter is passed on to the core.

Naming conventions

= =

New Application Log function module naming conventions:
Function module names: begin with BAL for Basis Application Log
Importing parameters: begin with I_
Exporting parameters: begin with E_
Changing parameters: begin with C_

If the function module parameter is a structure, an S_ follows, for tables a T_.

Example:
I_S_LOG_HEADER is a function module Importing parameter based on a DDIC structure.

Similar conventions apply for DDIC types: they usually begin with BAL_, followed by S_ for structures or T_ for table types.

Introduction

Situations can arise at runtime in an application program which must be brought to the user's attention. These are usually errors, but it may also be useful to report a successful procedure (although the user should not be inundated with unimportant information).

We will not distinguish between exceptions, errors, messages, etc. here. These are all situations in which a particular piece of information (usually a T100 message) arises, which is displayed in a log either immediately or later. This information is called a message here.

These messages are not output individually (ABAP command MESSAGE), they are collected and displayed together later.

This set of messages is a log. Logs usually also contain general header information (log type, creator, creation time, etc.) Several logs can be created in a transaction.

The Application Log provides a comprehensive infrastructure for collecting messages, saving them in the database and displaying them as logs. This infrastructure and some conventions are described below.

Technical core and simplified shell

The Application Log has two levels: a technical core of small, flexible function modules and a simplified shell, which uses core function modules for particular scenarios.

Example:
The technical core contains function modules to search the database (BAL_DB_SEARCH), load logs into memory (BAL_DB_LOAD) and output logs which are in memory (BAL_DSP_LOG_DISPLAY). The simplified shell contains the function module APPL_LOG_DISPLAY which uses the core function modules in this order to display logs which are in the database.

Both core and simplified shell function modules can be used.

The simplified shell function modules are generally the old Release 3.0 Application Log function modules (beginning with APPL_LOG_…) which internally now call the function modules of the new technical core (beginning with BAL_…).

All function groups are in the development class SZAL, the simplified shell function groups begin with SLG…, those of the technical core with SBAL…

If you already use the old Application Log, you need not change, but only some of the new Application Log functionality is available in the simplified old shell.

Example:
The (old) function module to display logs which are in the database, APPL_LOG_DISPLAY, has the new Importing parameter I_S_DISPLAY_PROFILE, which describes the log display format (see section Display log). This parameter is passed on to the core.

Naming conventions

= =

New Application Log function module naming conventions:
Function module names: begin with BAL for Basis Application Log
Importing parameters: begin with I_
Exporting parameters: begin with E_
Changing parameters: begin with C_

If the function module parameter is a structure, an S_ follows, for tables a T_.

Example:
I_S_LOG_HEADER is a function module Importing parameter based on a DDIC structure.

Similar conventions apply for DDIC types: they usually begin with BAL_, followed by S_ for structures or T_ for table types.

Simple call

Overview

How to collect messages and display them as a log most simply.

Function modules:
BAL_LOG_CREATE Create log with header data
BAL_LOG_MSG_ADD Add a message to a log
BAL_DSP_LOG_DISPLAY Display message in memory

Types:
BAL_S_LOG Contains log header data
BAL_S_MSG Contains message data
BALLOGHNDL Log handle
BALMSGHNDL Message handle

Example program
SBAL_DEMO_01 simulates a flight check and outputs a result log.
= = > SBAL_DEMO_01 = = > SBAL_DEMO_01 coding

Open log

The function module **BAL_LOG_CREATE** opens the Application Log whose header data is in the Importing parameter **I_S_LOG_HEADER**, which has the structure BAL_S_LOG.

The function module **BAL_LOG_CREATE** returns the log handle (LOG_HANDLE, CHAR22). The **LOG_HANDLE** is a GUID (globally unique identifier) which uniquely identifies a log. You can access this log with this handle, e.g. to subsequently change the header data (BAL_LOG_HDR_CHANGE) or to put a message in the log (BAL_LOG_MSG_ADD). The LOG_HANDLE has its permanent value straight awys, so it remains valid after saving.

= = > **Note:**
Logs in memory and in the database are referred to in the new Application Log by the log handle (LOG_HANDLE), but the previous LOGNUMBER, which is assigned from number range interval 01 of number range object APPL_LOG when you save, still exists. A lot of applications have a reference to this LOGNUMBER in their structures, so it is still supported. The LOGNUMBER is also more understandable for users than the LOG_HANDLE. There is a 1:1 relationship between LOG_HANDLE and LOGNUMBER.

Add a message to a log

= =

Functionality

A message is added to the log with the (log handle) I_LOG_HANDLE

The message data is passed to the function module BAL_LOG_MSG_ADD in the IMPORTING parameter **I_S_MSG** (structure BAL_S_MSG).

A message handle which uniquely identifies this message is returned in **E_S_MSG_HANDLE**.

This data is mostly the T100 information (message type, work area, message number, the 4 message variables), but can be other information such as application-specific data (context) or extended long text or callback routine parameters.

The function modules BAL_LOG_MSG_ADD, BAL_LOG_MSG_CUMULATE, etc. return the message handle (E_S_MSG_HANDLE).
The message handle comprises the log handle of the log to which the message belongs and an internally-assigned sequential number (MSGNUMBER). The handle uniquely identifies a message and you can access a message with it, e.g. to change (BAL_LOG_MSG_CHANGE) or read (BAL_LOG_MSG_READ) it.

Display log

= =
= =

BAL_DSP_LOG_DISPLAY displays the collected messages. It can be called without parameters, in which case all messages in memory are displayed in a standard format (this standard format is also used in the transaction SLG1).

= = > **Note**
The log handle is optional for function modules such as BAL_LOG_MSG_ADD, BAL_LOG_MSG_CUMULATE, BAL_LOG_MSG_ADD_FREE_TEXT, etc.
If it is not specified, the default log, which can be set, with other default data, with BAL_GLB_MSG_DEFAULTS_SET is used. If no default log is defined, it is set automatically by BAL_LOG_CREATE (see here).

Which data can be logged?

Overview

The Application Log logs message data as described below.

Function modules
BAL_LOG_CREATE Create log with header data
BAL_LOG_MSG_ADD Log a message

Types:
BAL_S_LOG Log header data
BAL_S_MSG Message data
BAL_S_CONT Message/log header context
BAL_S_PARM Message/log header parameters

Example program
Report SBAL_DEMO_02 simulates a flight check and outputs a result log.
= = > SBAL_DEMO_02 = = > SBAL_DEMO_02 coding

Log header

Application Log opens a log with BAL_LOG_CREATE. The header data is in the structure **BAL_S_LOG** as follows:

- **OBJECT, SUBOBJECT**
  The Application Log is used by various applications. Every log has the attributes OBJECT and SUBOBJECT to help applications to find their logs efficiently.
  These are normed (CHAR20) application or subapplication codes which you can set with the transaction SLG0 (example: OBJECT = "FREIGHT_COSTS" (freight costs), SUBOBJECT = "SETTLEMENT" (settlement)).
  These are optional in the log header at runtime, but they **must** be present when you save (with the function module BAL_DB_SAVE).

- **EXTNUMBER**
  The external ID in the log header (CHAR100) is a free text description of the log by the application.
  It could be used to link an application object to a log, by putting the application object document number in the external log ID.
  An external ID can also combine several logs into one logical log (logical logs can be displayed like one log).
  The database contains an index on the fields OBJECT/SUBOBJECT/EXTNUMBER. If these fields are specified, a log can be read from the database efficiently (no "Full Table Scan").

- **ALDATE, ALTIME, ALUSER , ALTCODE, ALPROG, ALMODE**

Further log header log creation information: date, time, user (ALDATE, ALTIME, ALUSER), the transaction or program which created the log (ALTCODE, ALPROG), and the processing mode in which the log was created (online, background, etc.)(ALMODE).

- **ALCHDATE, ALCHTIME, ALCHUSER**
  If an existing log in the database is changed later, the user, date and time are recorded in ALCHDATE, ALCHTIME and ALCHUSER.
- **DATE_DEL, DEL_BEFORE**
  Logs have an expiry date (DATE_DEL) after which they can be deleted, and a flag (DEL_BEFORE) which explicitly forbids deletion before this date. See here for more about deleting logs.
- **ALSTATE**
  Logs also have a status which specifies whether a log is finished or not. It is only for information and is not used.
- **CONTEXT: CONTEXT-TABNAME, CONTEXT-VALUE**
  Log header context information
- **PARAMS: PARAMS-T_PAR, PARAMS-ALTEXT, PARAMS-CALLBACK**
  Log header parameters

= = > Note 1
When you read a log header with BAL_LOG_HDR_READ you get additional information which is not in BAL_S_LOG, because it is generated internally, e.g. the internal LOGNUMBER, the number of A, E, W, I and S messages, and the highest problem class which occurred.

= = > Note 2
Application Log used to open logs at runtime by specifying the Application Log object and subobject, so logs were identified at runtime by OBJECT/SUBOBJECT and there could only be one log for one OBJECT/SUBOBJECT.
This restriction no longer applies. A log is identified by a handle and OBJECT/SUBOBJECT are only log attributes.

Message

Messages which Application Log can log with the function module BAL_LOG_MSG_ADD have the structure BAL_S_MSG, which has the following forms:

- **MSGTY, MSGID, MSGNO, MSGV1, MSGV2, MSGV3, MSGV4**
  T100 message data.
  The fields message type (MSGTY), work area (MSGID), and error number (MSGNO) are required, the fields for the four message variables MSGV1 to MSGV4 are optional.
- **PROBCLASS, DETLEVEL, ALSORT, TIME_STMP**
  These are T100 message attributes such as problem class (PROBCLASS, e.g. "very serious"), level of detail (DETLEVEL, between 1 and 9), sort criterion (ALSORT, any) and time stamp (TIME_STMP). These fields (except TIME_STMP) can be displayed.
- **MSG_COUNT**
  If a message is cumulated, the cumulation value can be put in MSG_COUNT, which is

incremented when BAL_LOG_MSG_CUMULATE adds more messages to it.

- **CONTEXT: CONTEXT-TABNAME, CONTEXT-VALUE**
  Message context information
- **PARAMS: PARAMS-T_PAR, PARAMS-ALTEXT, PARAMS-CALLBACK**
  Message parameters

Context

A message or log header is often only meaningful in context.
The Aplication Log provides a context.

Example:
The message 'Credit limit exceeded for customer ABC' is meaningful in dialog because it appears while a particular document is being processed, but the log should also contain the document number. This information may be in the message variables, but this can cause problems in detailed context information (e.g. order number, item number, schedule line number, etc.).

This context information can be passed with a message (or log header) to the Application Log in a DDIC structure (maximum length 256 bytes). You pass the name of the DDIC structure in **CONTEXT-TABNAME** and its contents in **CONTEXT-VALUE** for later display.

Example:

```
DATA:
  l_s_msg              TYPE  bal_s_msg,
  l_s_my_context     type  my_ddic_structure.

* Message 123(XY): 'Credit limit exceeded for customer &1'.
  l_s_msg-msgty =  'E'.
  l_s_msg-msgid =  'XY'.
  l_s_msg-msgno =  '123'.
  l_s_msg-msgv1 =  'ABC'.
* Add document number to message as context:
  l_s_my_context-document =  '3000012345'.
  l_s_msg-context-tabname =  'MY_DDIC_STRUCTURE'.
  l_s_msg-context-value    =  l_s_my_context.
* Log  message
  CALL  FUNCTION 'BAL_LOG_MSG_ADD'
       EXPORTING
          i_s_msg    =  l_s_msg
       EXCEPTIONS
          others      =  1.
```

Parameters

The Application Log can contain message header and message detail display parameter information, which can be used in two ways:

- As "extended long text"
  If the T100 message long text is not sufficient because more than the 4 message variables are needed, you can enter an additional 'Text in Dialog' containing any number of place holders, which are passed in the table **PARAMS-T_PAR**, in the field **PARAMS-ALTEXT**, with transaction SE61.
  The form routine MSG_ADD_WITH_EXTENDED_LONGTEXT in the program SBAL_DEMO_02 contains an example.

- As callback routine
  If you specify a callback routine to display your own detail information, in **PARAMS-CALLBACK**, it is called in the detail display.
  An Application Log callback routine can be realized in two ways:
  as a FORM routine or as a function module
  The following fields must be specified to setup a callback routine:
  USEREXITT: Routine type (' ' = FORM, 'F' = function module)
  USEREXITP: Program containing the routine (only for FORM)
  USEREXITF: Routine name (form routine or function module name)
  A function module must be parameterized like a form routine (USING is replaced by IMPORTING). The same parameter names must be used.

Message sets

Overview

Sets of logs can be created with the following methods.

Function modules:
BAL_LOG_CREATE Create log with header data
BAL_LOG_MSG_ADD Log a message
BAL_LOG_MSG_CUMULATE Add a message cumulatively to the log
BAL_LOG_MSG_REPLACE Replace the last message
BAL_GLB_MSG_CURRENT_HANDLE_GET Get the current message handle
BAL_LOG_MSG_DELETE Delete message
BAL_LOG_MSG_CHANGE Change message
BAL_GLB_MSG_DEFAULTS_GET Get message data defaults
BAL_GLB_MSG_DEFAULTS_SET Set message data defaults

Types
BAL_S_MDEF Message defaults

Example program
Program SBAL_DEMO_02 simulates a flight check and outputs a check result log.
= = > SBAL_DEMO_02 = = > SBAL_DEMO_02 coding

Add message to log

This is the 'classical' way of logging messages with BAL_LOG_MSG_ADD.

= = > **Note**
The log handle is optional for function modules such as BAL_LOG_MSG_ADD, BAL_LOG_MSG_CUMULATE, BAL_LOG_MSG_ADD_FREE_TEXT, etc.
If it is not specified, the default log, which can be set, with other default data, with BAL_GLB_MSG_DEFAULTS_SET is used. If no default log is defined, it is set automatically by BAL_LOG_CREATE (see here).

Add message cumulatively

Functionality

A message is added to the log with (log handle) I_LOG_HANDLE cumulatively.

The message data is in the IMPORTING parameter **I_S_MSG** (structure BAL_S_MSG).

A message handle **E_S_MSG_HANDLE**, which uniquely identifies this message, is returned.

'Cumulative'

Some messages are sent several times by a program, without providing new information each time. Such messages can be cumulated with BAL_LOG_MSG_CUMULATE to save memory. When the same message is repeated, no new message is added, the counter MSG_COUNT for the old message is incremented.
You can specify when messages are the same in the function module interface. The T100 data must be identical, and you can specify that other data must also be the same:

- o **I_COMPARE_ATTRIBUTES = 'X'**
  Message attributes (problem class PROBCLASS, level of detail DETLEVEL and sort field ALSORT) must be identical
- o **I_COMPARE_CONTEXT = 'X'**
  The context must be the same
- o **I_COMPARE_PARAMETERS = 'X'**
  The message parameters must be the same.

To find identical messages quickly for cumulation, the Application Log constructs a small index table containing as unambiguous a signature of a message as possible, at runtime. This index is only constructed if cumulation is used.

Replace last message

= =
= =

Functionality

The most recent Application Log message is deleted and replaced by a new message.

The new message data is in the IMPORTING parameter **I_S_MSG** (structure BAL_S_MSG). A message handle **E_S_MSG_HANDLE**, which uniquely identifies the message, is returned.

In which log is the new message put?

- o If a log handle is passed in **I_LOG_HANDLE**, the message is put in that log.
- o Otherwise it is put in the same log as the deleted message.
- o If there is no old message and no log is specified in I_LOG_HANDLE, the message is put in the default log (see here).

Why replace the last message?

The function module BAL_LOG_MSG_REPLACE can overwrite a message sent to the Application Log by an external program, with your own message.

Example
A generic scheduling module is called to calculate a flight schedule. If scheduling fails, the

function module may send a relatively technical message: "Scheduling of procedure 0006 unsuccessful". As messages should always be logged where they occur, this module sends a message to the Application Log. The message "The flight from Hamburg to New York could not be scheduled" would be much more meaningful to the user.

= = > Note
You can also get the handle of the last message sent with BAL_GLB_MSG_CURRENT_HANDLE_GET . This can be useful if you want to delete or change the last message, not replace it (with BAL_LOG_MSG_DELETE) or (BAL_LOG_MSG_CHANGE) respectively.

Message as free text

= =
= =

Functionality

A free text message is added to the log with (log handle) I_LOG_HANDLE.

The message text is passed to the function module BAL_LOG_MSG_ADD_FREE_TEXT in the IMPORTING parameter **I_TEXT** (maximum length 200 characters).

The error severity (**I_MSGTY**) and (optionally) the problem class (**I_PROBCLASS**) can also be specified.

A message handle **E_S_MSG_HANDLE**, which uniquely identifies this message, is returned.

Set message defaults

= =
= =

Some information which is required to make a message meaningful is only available at a higher program level, not where the message is sent.

Example
The destination of a road transport is checked in a low-level routine, which knows neither the transport number nor the route involved.

The context information can be set as defaults using **BAL_GLB_MSG_DEFAULTS_SET**, before this routine is called, and put in the messages which it sends.
The data type BAL_S_MDEF, which contains other data (such as message attributes, parameters, the default log, etc.) as well as the context, is passed to this function module.
You can also get the current defaults with **BAL_GLB_MSG_DEFAULTS_GET** . This is useful when you want to change some, but not all, defaults (e.g. the item number but not the order number).

- **= = > Note**
  You should use the function modules BAL_GLB_MSG_DEFAULTS_GET and BAL_GLB_MSG_DEFAULTS_SET together, to be sure of the current defaults.

The defaults affect the following function modules:

BAL_LOG_MSG_ADD Put a message in a log
BAL_LOG_MSG_CUMULATE Add message cumulatively
BAL_LOG_MSG_REPLACE Replace last message
BAL_LOG_MSG_ADD_FREE_TEXT Add message as free text

Message with complex context

= =

You may want to put more complex information in a message (or log header) than you can put in the context or parameter described above.
You can use the Application Log INDX table with the ABAP commands
`EXPORT TO DATABASE and IMPORT FROM DATABASE`
Program SBAL_DEMO_06 shows how you can save and read complex contexts, as follows:

- **Collect messages:**
  Define a CALLBACK routine (…-PARAMS-CALLBACK-…) for a log header or message. Collect the complex context information in internal user tables.

- **Save logs:**
  Write internal tables at this event with
  `EXPORT my_data TO DATABASE bal_indx(al) ID lognumber.`
  The internal log number LOGNUMBER is returned by the function module BAL_DB_SAVE.

- **Display log:**
  If message or log header detail is selected in the log display, the CALLBACK routine setup when collecting, is called
  You can read and display the data here using
  `EXPORT my_data FROM DATABASE bal_indx(al) ID lognumber.`
  The internal log number LOGNUMBER is in the internal table passed to this callback routine (under PARAM = "%LOGNUMBER").

- **Delete logs**
  The application does nothing. The data is deleted automatically.

= = > **Note**
Use complex context information with care. Problems may arise, e.g. if the structure of the complex context MY_DATA has changed in a Release change. You may not be able to read the data. There is currently no guarantee that the complex context can be archived automatically when the archiving function is realized (it does not yet exist).

Find and read messages

Overview

How to find and read messages in memory (not in the database).

Function modules:
BAL_GLB_SEARCH_LOG Find logs in memory
BAL_GLB_SEARCH_MSG Find messages in memory
BAL_LOG_HDR_READ Read log header and data
BAL_LOG_MSG_READ Read message and data

Types
BAL_S_LFIL Log header data filter criteria
BAL_S_MFIL Message data filter criteria
BAL_T_CFIL Context data filter criteria
BAL_T_LOGH Log handle table
BAL_T_MSGH Message handle table

Example program
SBAL_DEMO_03 creates several logs in memory, searches for messages and logs, and reads data.
= = > SBAL_DEMO_03 = = > SBAL_DEMO_03 coding

Find and read logs in memory

If the application program has no handle to access a log in the Application Log, it can get one with BAL_GLB_SEARCH_LOG, which searches for logs in memory.
This function module gets various log header data search criteria as Importing parameters:

- **I_S_LOG_FILTER**
  Log header filter criteria (structure BAL_S_LFIL)
- **I_S_LOG_CONTEXT_FILTER**
  Log header context data filter criteria (type BAL_T_CFIL)
- **I_T_LOG_HANDLE**
  Log handle set to be searched (type BAL_T_LOGH)

If several parameters are specified, they are related by a logical AND.
The result **E_T_LOG_HANDLE** is a table of log handles.

You can read the header data of a log using BAL_LOG_HDR_READ with a log handle. This function module returns the log header data in **E_S_LOG** and other data such as:

- **E_EXISTS_ON_DB E_IS_MODIFIED**

Does the log exist in the database?
If so, has it been changed?

- **E_LOGNUMBER**
  Internal log number (if the log has not yet been saved, this is only a temporary number beginning with '$')
- **E_STATISTICS**
  Statistical log data (120 messages, of which 13 errors, 4 warnings, etc.)
- **E_TXT_OBJECT, E_TXT_SUBOBJECT, E_TXT_ALMODE, etc.**
  Texts for various log header fields (e.g. E_TXT_ALMODE = 'Batch Input', if ALMODE = 'I')

Find and read messages in memory

If the application program has no handle to access an Application Log message, it can get one with BAL_GLB_SEARCH_MSG, which searches memory for logs.
This function module gets various message or log header data search criteria as Importing parameters:

- **I_S_LOG_FILTER**
  Log header filter criteria (structure BAL_S_LFIL)
- **I_S_LOG_CONTEXT_FILTER**
  Log header context data filter criteria (type BAL_T_CFIL)
- **I_T_LOG_HANDLE**
  Set of log handles to be searched (type BAL_T_LOGH)
- **I_S_MSG_FILTER**
  Message data filter criteria (structure BAL_S_MFIL)
- **I_S_MSG_CONTEXT_FILTER**
  Log header context data filter criteria (type BAL_T_CFIL)
- **I_T_MSG_HANDLE**
  Set of message handles to be searched (type BAL_T_MSGH)

If several parameters are specified, they are related by a logical AND.
The result **E_T_MSG_HANDLE** is a table of the set of message handles of the messages found.

You can read message data with BAL_LOG_MSG_READ using a message handle. This function module returns the message data in **E_S_MSG** and additional data such as:

- **E_EXISTS_ON_DB**
  Does the message already exist in the database?
- **E_TXT_MSGTY, E_TXT_MSGID, etc.**
  Texts for various message fields (e.g. E_TXT_MSGTY = 'Error', if MSGTY = 'E')

Log header filter criteria

= =
= =

This structure contains log header filter criteria, mainly field RANGES such as:
- **LOG_HANDLE**
- **EXTNUMBER**
- **OBJECT**
- **SUBOBJECT**
- **ALDATE**
- **ALTIME**
- **ALPROG**
- **ALTCODE**
- **ALUSER**
- **ALMODE**
- **PROBCLASS**

You can also search for the internal log number **LOGNUMBER**.

For a time interval, use **DATE_TIME** which contains the:
- from time (**DATE_TIME-DATE_FROM DATE_TIME-TIME_FROM** and the
- to time (**DATE_TIME-DATE_TO DATE_TIME-TIME_TO**)

= = > **Note**
If you specify several criteria, they are related by a logical AND.

o **Example**
Search for all logs of object 'BCT1' with external number '12345' or '67890' which were created by transaction 'XY01' this morning

```
= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = = = = = = = = = = = = = = = = =
DATA:
  l_s_log_filter   TYPE bal_s_lfil,
  l_r_object       TYPE bal_s_obj,
  l_r_extnumber    TYPE bal_s_extn,
  l_r_altcode      TYPE bal_s_tcde.

* define object
  l_r_object-option    = 'EQ'.
  l_r_object-sign      = 'I'.
  l_r_object-low       = 'BCT1'.
  append l_r_object to l_s_log_filter-object.
* define external numbers
  l_r_extnumber-option = 'EQ'.
  l_r_extnumber-sign   = 'I'.
  l_r_extnumber-low    = '12345'.
  append l_r_extnumber to l_s_log_filter-extnumber.
  l_r_extnumber-low    = '67890'.
  append l_r_extnumber to l_s_log_filter-extnumber.
```

```
* transaction code
  l_r_altcode-option     =  'EQ'.
  l_r_altcode-sign       =  'I'.
  l_r_altcode-low        =  'XY01'.
  append l_r_altcode to l_s_log_filter-altcode.
* time interval
  l_s_log_filter-date_time-date_from =  sy-datum.
  l_s_log_filter-date_time-time_from =  '000000'.
  l_s_log_filter-date_time-date_to   =  sy-datum.
  l_s_log_filter-date_time-time_to   =  '120000'.
==============================================
=================
```

Message filter criteria

==
===================================

This structure contains mesage filter criteria, mainly message field RANGES such as:
- **MSGNUMBER** Message number in Application Log
- **MSGID** Message class (or work area)
- **MSGNO** Message number in message class
- **MSGTY** Message type
- **DETLEVEL** Level of detail
- **PROBCLASS** Problem class

You can also specify a combination of message class and message number (field **MSGIDMSGNO**).

= = > Note
If several criteria are specified, they are related by a logical AND.

o **Example**
Search for all serious and very serious errors.

```
==============================================
==================
DATA:
  l_s_msg_filter   TYPE bal_s_mfil,
  l_r_msgty        TYPE bal_s_msty,
  l_r_probclass    TYPE bal_s_prcl.

* define message type
  l_r_msgty-option     =  'EQ'.
  l_r_msgty-sign       =  'I'.
  l_r_msgty-low        =  'E'.  "Error
  append l_r_msgty to l_s_msg_filter-msgty.
* define problem class
  l_r_probclass-option     =  'EQ'.
  l_r_probclass-sign       =  'I'.
  l_r_probclass-low        =  '1'.  "very serious messages
```

```
  append l_r_probclass to l_s_msg_filter-probclass.
  l_r_probclass-low        = '2'. "serious messages
  append l_r_probclass to l_s_msg_filter-probclass.
= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = = = = = = = = = = = = = = = =
```

Context filter criteria

Context filter criteria; context field RANGES.

Internal table with the structure:
- **TABNAME** context DDIC structure name
- **FIELDNAME** field whose RANGE follows
- **T_RANGE** RANGE table with SIGN, OPTION, LOW and HIGH

o **Example**
Search for airlines 'SF' and 'AB':

```
= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = = = = = = = = = = = = = = = =
DATA:
  l_t_context_filter   TYPE bal_t_cfil,
  l_s_context_filter   TYPE bal_s_cfil,
  l_s_range            TYPE bal_rfield.


* define field
  l_s_context_filter-tabname     = 'BAL_S_EX01'.
  l_s_context_filter-fieldname = 'CARRID'.
* define airlines
  l_s_range-option     = 'EQ'.
  l_s_range-sign       = 'I'.
  l_s_range-low        = 'SF'.
  append l_s_range to l_s_context_filter-t_range.
  l_s_range-low        = 'AB'.
  append l_s_range to l_s_context_filter-t_range.
* put result in filter table
  append l_s_context_filter to l_t_context_filter.
…
= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = = = = = = = = = = = = = = = =
```

Log and message handle tables

Log handle table.

= = > **Note**
Sorted table. Make entries with INSERT … INTO TABLE, not APPEND.

Message handle table.

= = > **Note**
Sorted table. Make entries with INSERT … INTO TABLE, not APPEND.

Log display: Functional principle

Which information can be displayed?

You could imagine the set of messages in memory as an extremely wide table with a large number of fields (the data is not saved in this form in memory). The possible fields in this table are:

- Message line (MSGTY, MSGID, MSGNO, MSGV1, etc.)
- Message attributes (PROBCLASS, DETLEVEL, etc.)
- Message context fields
- Message texts:
  - Formatted message line
  - Field long texts ("Very serious" for problem class 1, etc.)
- Data of the log header to which this message belongs:
  - Log header data (EXTNUMBER, USER, DATUM, etc.)
  - Log header context fields
  - Log header texts (field long texts)
- External data inserted by the caller by a callback routine, e.g. the material short text

The displayable fields are listed in the structure BAL_S_SHOW, which does not contain the context fields or external data, which the Application Log cannot know.

How is the data formatted?

This large dataset must be presented appropriately to the user. The data formatting can be controlled by specifying a profile, which is a caller-defined complex data type (structure BAL_S_PROF), which is passed to the output module BAL_DSP_LOG_DISPLAY, not a user-defined display variant.
The display is based on certain basic assumptions:

- The messages are presented in a list which contains a subset of the displayable fields, which can be specified in a field catalog (analogously to the ABAP List Viewer) which is in the display profile BAL_S_PROF.
- Detail information can be called for each message:
  - Message long text

  - Eetended long text or CALLBACK routine
  - Technical information about a message (message type, work area, message number, etc.)

- You can search for and filter the message set with ABAP List Viewer functions. You can also conveniently restrict the dataset by message type (A, E; W; I/S) in the list header. You can show or hide the I and S messages by clicking on an icon.
- You can add a hierarchy tree for navigation in what can be a long and confusing list. The tree provides a table of contents for the message set. You can display the messages in a chapter in the list by clicking on a node or pushbutton. You can specify the tree structure in the display profile.

Log display: Function module BAL_DSP_LOG_DISPLAY

Overview

Function modules:
BAL_DSP_LOG_DISPLAY Display logs

Types
BAL_S_PROF Display profile
BAL_S_LFIL Filter criteria for log header data
BAL_S_MFIL Filter criteria for message data
BAL_T_CFIL Filter criteria for context data
BAL_T_LOGH Log handle
BAL_T_MSGH Message handle table

Function module BAL_DSP_LOG_DISPLAY

Functionality

The transaction SLG1 displays database Application Logs in a standard format.

Logs must often be output in a different, application-dependent format, and logs which have not been saved must also be displayed.

Assume that a set of logs containing messages, which were either collected or loaded from the database (BAL_DB_LOAD), is in memory.

This data can be displayed by calling the function module **BAL_DSP_LOG_DISPLAY**, passing:

- **I_S_LOG_FILTER, … I_T_MSG_HANDLE**
  What is to be displayed (via filter criteria)
- **I_S_DISPLAY_PROFILE**
  How the data is to be displayed (via a display profile)
- **I_AMODAL**
  Whether the display is in another session.

= = > Note
You lose program control of displays in a new session and cannot refresh the log display.

Example

Prgram SBAL_DEMO_04 shows various ways of displaying logs (= = > Run = = > Coding).

WHAT is to be displayed?

The IMPORTING parameters determine the dataset to be displayed by specifying the:

- log filter criteria
  - I_S_LOG_FILTER Log header filter criteria
  - I_S_LOG_CONTEXT_FILTER Log header context filter criteria
  - I_T_LOG_HANDLE Log handle table
- message filter criteria
  - I_S_MSG_FILTER Message filter criteria
  - I_S_MSG_CONTEXT_FILTER Message context filter criteria
  - I_T_MSG_HANDLEMessage handle table

The filters are the same data types as are used to search for messages and logs.
If specifying filters is not sufficient, you can specify the dataset to be displayed by specifying a set of log and message handles which you have collected by your own criteria.

- If you specify several parameters, they are linked by a logical AND, so only those messages which satisfy all conditions are displayed.
- If you only specify log filter criteria, all messages in memory whose log headers satisfy the specified criteria are displayed (logs containing no messages can also be displayed).
- If there are only message criteria, all messages in memory which satisfy these criteria are displayed.
- If none of these parameters are specified, all messages in memory are displayed.

HOW is the data to be displayed?

The display profile (structure BAL_S_PROF) specifies how the log is to appear. It contains the field catalog, which describes which fields are in the list and in the various chapter levels of the navigation tree.

The Application Log provides predefined display profiles which you can get with function modules, but you can also construct your own display profile. If no display profile is specified, the standard display profile of transaction SLG1 is chosen.
The display profile contains the following fields (all fields are optional except **MESS_FCAT**):

- **General parameters**
  - **LANGU**
    Log output language
  - **TITEL**
    Screen title

- **USE_GRID**
  Messages are to be displayed with the ALV Grid Control, not the standard ALV (ignored if the display uses Grid Control by default).

- **START_COL, START_ROW, END_COL, END_ROW**
  Coordinates if the log is displayed in a popup.
  POP_ADJST
  If this parameter is 'X', the system adjusts the dialog box height
  (if the log is displayed in a dialog box) to the data to be displayed. The values entered above are then upper limits.

- **NOT_EMPTY**
  If this parameter is 'X', branches which are just one entry and all data (in the field catalog) are initial, are not displayed.
  Example:
  As document 1000013 initially has no position-dependent messages, the tree is initially (by document and item numbers):
  Document 1000012
  -- Item 0010
  -- Item 0020
  Document 1000013
  -- Item 0000
  Document 1000014
  -- Item 0010
  If NOT_EMPTY = 'X' the entry "Item 0000" is omitted because all fields after it are initial (including invisible ones).

- **COLORS**
  Problem class display colors. COLORS-PROBCLASS1 = "3" e.g. highlights all problem class 3 messages in yellow (standard).
  Message list parameter

- **MESS_FCAT**
  Message list field catalog (table type BAL_T_FCAT with structure BAL_S_FCAT).

- **MESS_SORT**
  Message sort sequence table (table type BAL_T_SORT with structure BAL_S_SORT). Contains table and field name, serial number and sort ascending or descending flag. Fields mentioned here must have been previously mentioned in MESS_FCAT.

- **SHOW_ALL**
  "X": all messages are immediately visible in the list and need not be selected in the tree first.

- **MESS_MARK**
  "X": messages selectable via checkbox

- **CWIDTH_OPT**
  Optimize message list column width

- **DISVARIANT**
  Structure DISVARIANT: List viewer display variant data (Report name, etc.)

o **Navigation tree parameters**

- **LEV1_FCAT, …, LEV9_FCAT**
  Field catalogs for chapter levels 1 to 9 (table type BAL_T_FCAT with structure BAL_S_FCAT).
- **LEV1_SORT, …, LEV9_SORT**
  Chapter level sort sequence table (table type BAL_T_SORT with structure BAL_S_SORT).
- **HEAD_TEXT, HEAD_SIZE**
  CHAR/INTEGER: Contents and width of the tree header
- **ROOT_TEXT**
  CHAR: Root node text (only to be used in exceptional cases. The root node should not be a header, which should be in HEAD_TEXT).
- **TREE_SIZE**
  INTEGER Tree Control size (in CHARACTER). This value is only approximate because of proportional font.
- **TREE_ONTOP**
  "X": tree control is displayed above the message list
- **TREE_ADJST**
  "X": if the tree is above the messages (TREE_ONTOP = 'X'), try to adjust the hight of the tree to the number of rows to be displayed. TREE_SIZE is the maximum hight of the tree.
- **EXP_LEVEL**
  1,.., 9 the level to which the tree is to be expanded
- **BYDETLEVEL**
  "X": construct navigation tree by message field DETLEVEL.
- **TREE_NOMSG**
  "X": the tree contains no message data. It consists exclusively of log header data.

o **Callback routines**
The callback routines are defined according to the DDIC structure BAL_S_CLBK.

- **CLBK_READ**
  Read external display data (e.g. material short text)
  (see here).
- **CLBK_UCOM**
  Perform user commands
  (see here).
- **CLBK_UCBF**
  Called BEFORE performing a user command
  (see here).
- **CLBK_UCAF**
  Called AFTER performing a user command
  (see here).
- **CLBK_PBO**
  Display PBO (e.g. to set a user status)

(see here).

- **User pushbuttons**
  - **EXT_PUSH1, …, EXT_PUSH4**
    These components put user pushbutton in the menu, without having to define a GUI status. Choosing one of these pushbuttons at PAI calls the user command "%EXT_PUSH1", … "%EXT_PUSH4", to which you can react in the corresponding callback routine. A pushbutton definition has the following elements:
    EXT_PUSH1-ACTIVE = "X": pushbutton is active
    EXT_PUSH1-DEF-TEXT pushbutton text
    EXT_PUSH1-DEF-ICON_ID pushbutton icon
    EXT_PUSH1-DEF-ICON_TEXT icon text
    EXT_PUSH1-DEF-QUICKINFO quick info
    EXT_PUSH1-DEF-PATH fastpath

Field catalog BAL_T_FCAT

The field catalog BAL_T_FCAT defines the fields in the message list and in the various levels of the navigation tree in the Application Log display.
A field catalog entry (structure BAL_S_FCAT) contains the following information:

- **REF_TABLE, REF_FIELD**
  Table field name of the field to be displayed (e.g. BAL_S_SHOW-PROBCLASS).
  These two fields are required, all others are optional.
- **COL_POS**
  Position at which this field is to be displayed.
  Note: the Application Log puts the error seriousness icon in the first column by default. This column is fixed and is in all outputs (for recognizability). The delivered standard profile starts with column 2 (e.g. for the field T_MSG, message text). The message long text and detail icons are also automatically added.
  If you change a standard profile and insert a column before the message text (in position 2), the column positions of the other fields must be incremented by 1.
- **OUTPUTLEN**
  Output length
- **COLTXT_ADD, COL_SEP**
  Each navigation tree level should normally only display a small number of fields (ideally only one) to avoid overloading the user with unnecessary information. The name of a field cannot be a navigation tree column header, because it can have up to 9 levels, but you can put the field name in front of the field contents by setting the COLTXT_ADD flag to #X# in the field catalog. You can also put a separator (e.g. a colon) between the field name and the value (COL_SEP = #:#).
  Example: #Problem class: 1# instead of only #1#.
- **COLTEXT, COLDDICTXT, CLTXT_LEN**
  You can specify the text before a field (or above a column) in COLTEXT. It is normally got

from DDIC as specified in COLDDICTXT (COLDDICTXT = "L", "M", "S", "R" specifies whether the long, medium or short field name or the header is to be taken). CLTXT_LEN specifies the length.

- **IS_TREECOL**
  You can only put fields with a column header next to the tree at level 1, by setting the field IS_TREECOL to #X#.
- **IS_EXTERN**
  Some fields are not in a message or its context, but they can be derived from it. For example, if the material number is in the context, you can get the material short text in the callback routine BAL_CALLBACK_READ, which handles all fields which are flagged IS_EXTERN = "X" in the field catalog.
- **NO_OUT**
  Some fields are technical and should not be displayed (z.B. LOG_HANDLE). Flag them as NO_OUT.

Sort catalog BAL_T_SORT

The sort catalog BAL_T_SORT defines the message or navigation tree entry sort sequence in the Application Log display.
A sort catalog entry (structure BAL_S_SORT) contains the following information:

- **REF_TABLE, REF_FIELD**
  Table field name to be displayed (e.g. BAL_S_SHOW-PROBCLASS)
  = = > **Note**
  This field MUST be in the message or chapter level field catalog.
- **SPOS**
  Sort sequence
- **UP, DOWN**
  Sort in ascending or descending order

Log display: Standard display profiles

Overview

You pass a Display profil, which describes the log display format, to the function module BAL_DSP_LOG_DISPLAY.

The Application Log provides various pre-defined display profiles which you can get with function modules (and change if necessary).

Function modules:
BAL_DSP_PROFILE_STANDARD_GET Standard profile (SLG1) for a lot of logs
BAL_DSP_PROFILE_SINGLE_LOG_GET Standard profile (SLG1) for one log
BAL_DSP_PROFILE_NO_TREE_GET Display without tree (fullscreen)
BAL_DSP_PROFILE_POPUP_GET Display without tree (popup)
BAL_DSP_PROFILE_DETLEVEL_GET Hierarchy by message DETLEVEL

Example program
Program SBAL_DEMO_04 shows various ways of displaying logs.
= = > SBAL_DEMO_04 = = > SBAL_DEMO_04 coding

Standard profile (SLG1) for a lot of logs

Functionality

BAL_DSP_PROFILE_STANDARD_GET returns a display profile which is used in the standard Application Log display transaction (SLG1) to display several logs at once.
In this format, navigation tree chapter level 1 contains the log header and its data, level 2 is classified by problem class. The message text is in the message list.
The messages are not displayed until the user selects a log (or subset of a log).

Example

Program SBAL_DEMO_04_STANDARD (= = > Run = = > Coding)

Standardprofil (SLG1) for one log

Functionality

BAL_DSP_PROFILE_SINGLE_LOG_GET returns a display profile which is used in the standard

Application Log display transaction (SLG1) when only one log is to be displayed.

In this format the navigation tree chapter 1 level contains the log header and its data, level 2 is classified by problem class. The message text is in the message list.

The format is designed to show one log:
- the problem class classification is expanded
- the message list is displayed

Example

Program SBAL_DEMO_04_SINGLE_LOG (= = > Run = = > Coding)

Display without tree (fullscreen)

= =
= =

Functionality

BAL_DSP_PROFILE_NO_TREE_GET returns a display profile which lists the messages in a full screen.

You see the message text in the message list.

There is no navigation tree to navigate in the set of messages in this format.

Example

Program SBAL_DEMO_04_NO_TREE (= = > Run = = > Coding)

Display without tree (popup)

= =
= =

Functionality

BAL_DSP_PROFILE_POPUP_GET returns a display profile which lists the messages in a popup.

You see the message text in the message list. There is no navigation tree to navigate in the set of messages in this format.

Example

Report SBAL_DEMO_04_POPUP (= = > Run = = > Coding)

Hierarchy by message DETLEVEL

= =
= =

Functionality

BAL_DSP_PROFILE_DETLEVEL_GET returns a display profile which the navigation tree creates from the message texts.

The field DETLEVEL, which you can specify in the structure BAL_S_MSG when you create a message (e.g. with function module BAL_LOG_MSG_ADD), specifies the level in the tree.

Example

A log contains the following data:

Typ DETLEVEL Meldungstext
S 1 Settlement of Airline SAP Flights
S 2 Flight 007 from Hamburg to Toronto
S 3 Invoice 17003115 created
S 3 Invoice 17003116 created
E 3 Invoice 17003117 error
E 4 Customer 1234 in document 17003117: Address data incomplete
S 3 Invoice 17003118 created
S 3 Invoice 17003119 created
S 3 Invoice 17003120 created
…

It is displayed as follows:

@5C@Settlement of Airline SAP Flights
| @5C@Flight 007 from Hamburg to Toronto
| @5B@Invoice 17003115 created
| @5B@Invoice 17003116 created
| @5C@Invoice 17003117 error
| @5C@Customer 1234 in document 17003117: Address data…
| @5B@Invoice 17003118 created
| @5B@Invoice 17003119 created
| @5B@Invoice 17003120 created
….

The error seriousness is passed upwards in the tree, i.e. if the message " Customer 1234 in document 17003117: Address data incomplete " is an error (red icon), it is passed to the higher levels.

See also program SBAL_DEMO_04_DETLEVEL (= = > Run = = > Coding)

Display log in subscreen

Functionality

The Application Logs can be displayed as a subscreen.

Example

Program SBAL_DEMO_04_SUBSCREEN shows how to display logs in a subscreen (= = > Run = = > Coding).

Related function modules

BAL_DSP_OUTPUT_INIT Initialize output
BAL_DSP_OUTPUT_SET_DATA Define dataset to be displayed
BAL_DSP_OUTPUT_PAI Process PAI function codes
BAL_DSP_OUTPUT_FREE End output

Procedure

- **Define subscreen area in user screen**
  e.g. in a Tabstrip (subscreen area name MY_SUBSCREEN).

  CALL SUBSCREEN MY_SUBSCREEN INCLUDING 'SAPLSBAL_DISPLAY' '0101' must be called at PBO.
  At PAI only:
  CALL SUBSCREEN MY_SUBSCREEN.

- **Before calling screen or only once at PBO**
  Initialize the display with function module **BAL_DSP_OUTPUT_INIT**, which has the IMPORTING parameter Display profile, which controls how the data are to be displayed.

  - = = > **CAUTION**
    USE_GRID = 'X' must be in this display profile and is not necessarily set by the standard function modules BAL_DSP_PROFILE_…

- **After calling BAL_DSP_OUTPUT_INIT**
  Call function module **BAL_DSP_OUTPUT_SET_DATA**, which defines the dataset to be displayed.
  This function module can be called several times if, e.g. the dataset to be displayed has changed.
  It has similar parameters to the function module BAL_DSP_LOG_DISPLAY and gets Filter criteria which determine which of the data in memory are to be displayed.

- **PAI**
  Call function module **BAL_DSP_OUTPUT_PAI** to process the subscreen comands (e.g. Go to long text).

- **End display**

Call function module **BAL_DSP_OUTPUT_FREE** to close Controls and release resources.

= = > Note
The function modules BAL_DSP_OUTPUT_INIT and BAL_DSP_OUTPUT_FREE are always called as a pair; they increment and decrement the program-internal Application Log stack respectively. This logic lets you display another log (e.g. in a popup) in log display.

Save and load logs

Overview

Logs which have been collected in memory can be saved in the database. Saved logs can be reloaded into memory and changed or displayed.

Function modules
BAL_DB_SAVE Saves logs in the database
BAL_DB_SAVE_PREPARE Prepare save
BAL_DB_SEARCH Find logs in the database
BAL_DB_LOAD Load logs from the database
BAL_LOG_REFRESH Remove logs from memory
BAL_GLB_MEMORY_REFRESH (Partially) reset global memory

Example

Report SBAL_DEMO_05 (= = > Run = = > Coding) simulates a settlement run for all flights on a specified date. You can choose:

- Simulate settlement.
  The documents are only collected in memory with temporary numbers, which are logged.
- Perform settlement.
  A log is saved in the database after the temporary document numbers have been replaced by permanent ones in the log.
- Display logs.

Save logs

Functionality

You can save logs in memory in the database with the function module BAL_DB_SAVE. You can save all data in memory (Importing parameter I_SAVE_ALL = 'X') or a subset specified by a set of log handles (Importing parameter I_T_LOG_HANDLE).

You can get the log handle table by calling the function module BAL_GLB_SEARCH_LOG which searches for logs in memory by specified filter criteria.

When logs are saved, an internal log number is issued (field LOGNUMBER). At runtime this field has a temporary value (e.g. $00001).

The function module BAL_DB_SAVE returns a table (Exporting parameter E_NEW_LOGNUMBERS) which relates LOG_HANDLE, external number EXTNUMBER, temporary LOGNUMBER and permanent LOGNUMBER, so you can find out which number was

assigned to a log after saving.

You can also save IN UPDATE TASK (Importing parameter I_IN_UPDATE_TASK = 'X').

You can also save the logs in another client (parameter I_CLIENT). If you do not specify I_CLIENT, you save in the current client.

Notes

After logs have been saved they are still in memory in a state as though they had just been loaded from the database. To delete saved logs from memory, use either the function module BAL_LOG_REFRESH (for one log) or BAL_GLB_MEMORY_REFRESH (for several or all logs).

The field LOGNUMBER is still visible to the caller for reasons of compatibility, but it only has a temporary value at runtime and only becomes permanent after saving, so all application tables which point to a log with the LOGNUMBER must be updated when saving.
If you use the LOG_HANDLE field, this is not necessary. LOG_HANDLE has its permanent value as soon as a log is created (with BAL_LOG_CREATE).

Prepare save

= =
= =

Functionality

Application Log message variables or contexts can sometimes still contain temporary data.

For example document numbers. When a document is created, it has a temporary number (e.g. #$0001#). A permanent number is only issued from a number range interval when the document is saved.

If messages are created for such a document, the message variables could contain temporary numbers. These temporary values should be replaced by permanent ones when you save (e.g. $0001 by 0000123456), otherwise the log is of no value.

The function module BAL_DB_SAVE_PREPARE performs this substitution.
You pass a replacement pattern (table type **BAL_T_RPLV**) which specifies, for example that message variables with old contents #$0001# (field OLD_VALUE) are to be replaced by the new value #12345# (field NEW_VALUE), #$0002# by #67890#, etc.

You can also define replacements for context information (table type **BAL_T_RPLC**). For example, the data in the field #VBELN# (FIELDNAME) is to be replaced in all contexts which have the DDIC structure #MY_STRUC# (TABNAME); #$0001# (OLD_VALUE) by #1234 (NEW_VALUE), etc.

I_REPLACE_IN_ALL_LOGS = 'X' specifies that the replacement be made in all logs in memory. If you only want to replace in certain logs, set I_REPLACE_IN_ALL_LOGS = ' ' and put the log handle in I_T_REPLACE_IN_THESE_LOGS.

- **Note**
  When replacing message variables you cannot be completely sure that e.g. the message variable MSGV1 in a particular message really contains an order number. It could also be a (coincidentally identical) temporary number of a different document which was created in the background and for which messages were also created.

You can avoid such ambiguities by specifying the source of a message variable when a message is sent, in the fields MSGV1_SRC, …, MSGV4_SRC in the structure BAL_S_MSG. You can refer to these values (field MSGV_SRC) when you replace the message variables with BAL_DB_SAVE_PREPARE.

Find logs in the database

Functionality

The function module BAL_DB_SEARCH finds logs in the database.

You pass log header filter criteria (structure BAL_S_LFIL), and a sorted table of log headers (structure BALHDR) which satisfy the criteria is returned.
You can pass this to the module BAL_DB_LOAD BAL_DB_LOAD, which loads these logs into memory.

- **Notes**
  Avoid a FULL TABLE SCAN when you create the filter structure BAL_S_LFIL by specifying the following fields or field combinations:
  - LOGNUMBER (primary index of the log header table)
  - LOG_HANDLE (has an index)
  - OBJECT/SUBOBJECT/EXTNUMBER (has an index)

  For an application object to efficiently access a log, it must have either LOGNUMBER or LOG_HANDLE in its structures, or the field EXTNUMBER should contain a unique key derived from the application object data (e.g. document number). Together with the OBJECT/SUBOBJECT (the application which wrote the log), the access should be unique. Other criteria such as time restrictions or transaction which created the log, can also be specified in the filter structure.
  You can also search in another client. The client in E_T_LOG_HEADER is taken into account automatically. If I_CLIENT is not specified, the current client is used.

Load logs from the database

Functionality

The function module BAL_DB_LOAD loads logs from the database.
Which logs are to be loaded into memory can be specified in one of several ways:

- **I_T_LOG_HANDLE**
  A table of log handles
- **I_T_LOGNUMBER**
  A table of internal log numbers
- **I_T_LOG_HEADER**
  A table of log headers (returned by function module BAL_DB_SEARCH)

The result of loading can be a table of log handles (Exporting parameter E_T_LOG_HANDLE) or message handles (Exporting parameter E_T_MSG_HANDLE).

This function module is cross-client:

- If you specify I_T_LOG_HANDLE it searches in all clients (this is not critical because the log handle is globally unique)
- If you specify I_T_LOGNUMBER the client in the parameter I_CLIENT is taken into account. If it is not specified, the current client is used.
- If you specify I_T_LOG_HEADER the client in the table field MANDANT is taken into account (it is filled automatically by the function module BAL_DB_SEARCH).

Other parameters:

You can specify that only the log headers are to be loaded in memory with the Importing parameter I_DO_NOT_LOAD_MESSAGES. See Read log messages as required.

You can specify that the exception LOG_ALREADY_LOADED be raised if one of the logs to be loaded is already in memory, with the Importing parameter
I_EXCEPTION_IF_ALREADY_LOADED = 'X'. In this case no logs are loaded.
I_EXCEPTION_IF_ALREADY_LOADED = ' ' (default) ignores a log to be loaded if it is already in memory. All other logs are loaded correctly.

- **Note**
  To load the database status, use the function module BAL_DB_RELOAD, which first deletes a log from memory if necessary before loading it.

Read log messages as required

The parameter I_DO_NOT_LOAD_MESSAGES = "X" tells the function module BAL_DB_LOAD to read only the log headers into memory.
The messages in a log are only read into memory at certain events:

- read access to the messages (e.g. to display the log, or by the function module BAL_LOG_MSG_READ)
- change access to the log (e.g. when changing the header data with function module BAL_LOG_HDER_CHANGE, or when adding messages with BAL_LOG_MSG_ADD)
- when the function module BAL_DB_RELOAD is called for this log.

when the messages in a log are all reloaded once. Messages are not reloaded individually, so either only the header, or all of a log is in memory.

I_DO_NOT_LOAD_MESSAGES = "X" has no effect if you have defined statistics with the function module BAL_STATISTICS_GLB_SET or BAL_STATISTICS_LOG_SET (Application Log statistics tell you how many errors, warnings, etc. there were at a particular time for a specified criterion, e.g. "3 errors for material ABC").
These statistics are based on message data, so the messages must always be read.

I_DO_NOT_LOAD_MESSAGES = "X" is useful if you want to see the log header data first.

It can also be used to display logs:

if the display profile only uses log header data in the tree, you only need to read the log headers into memory. When a log is selected in the tree, the log messages are read for display.

The function module BAL_DSP_LOG_DISPLAY does not automatically know whether the tree only contains log header data, because it can also contain context information, which can be in a message or the header, so the display profile I_S_DISPLAY_PROFILE (structure BAL_S_PROF) must explicitly state that the tree contains no message data, with I_S_DISPLAY_PROFILE-TREE_NOMSG = #X#.

- **Note**
  I_DO_NOT_LOAD_MESSAGES = #X#
  and the display profile option
  I_S_DISPLAY_PROFILE-TREE_NOMSG = #X#
  are only meaningful when
  I_S_DISPLAY_PROFILE-SHOW_ALL = # #
  in the display profile, otherwise the messages are displayed immediately.

Delete log from the database

Overview

Logs must be deleted to prevent the Application Log database tables from overflowing. There are two ways of doing so:

- with the standard log deletion transaction SLG2, whose log deletion logic is described below.
- by calling Application Log function modules from the application (e.g. when deleting an application object or in a user application transaction)

Function modules
BAL_DB_DELETE Delete logs from the database

Transaction SLG2: Delete logs

This transaction is a report on whose selection screen you can specify which logs are to be deleted. It can run online (Program -> Execute (F8)) or in the background (Program-> Execute in Background (F9)), where it can be scheduled regularly.

- **Options**
  - **Get number only**
    The report reads no database data, it just determines how many logs can be deleted. This is quick, and is the default option.
  - **Create list**
    No logs are deleted, those which can be deleted are listed, and the user can select those to be deleted.
    If more than 100 logs can be deleted, the first 100 are displayed, followed by the next 100 on demand, etc.
  - **Delete immediately**
    All deletable logs are deleted immediately from the database (useful in Batch).
- **Selection conditions**
  The set of logs to be deleted can be specified by selection conditions for the Application Log object/subobject, external number, log number, problem class and creation date/time.
- **Expiry date**
  Not all logs which satisfy the selection conditions can be deleted.
  A log can only be deleted when it has expired, i.e. its expiry date has been reached or passed. You specify the expiry date in the log header data (structure BAL_S_LOG) field **ALDATE_DEL** when you create a log with the function module BAL_LOG_CREATE.
  The field ALDATE_DEL is rarely filled and is set to 31.12.2098 by the Application Log by default. Such logs are in practice never deleted from the system, so the report has the

following "Expiry date" options:

- **Only delete logs whose expiry data has been reached**
  This is the standard option which does not delete logs for which no expiry date was specified.
- **Delete logs which can be deleted before expiry**
  This option also deletes logs whose expiry date has not been reached, but whose DEL_BEFORE flag is initial. DEL_BEFORE can be passed when you open a log, like ALDATE_DEL. It's default value is initial, i.e. "Delete before expiry" is allowed.

The user can thus delete logs in two steps:

o delete expired logs

o delete logs which can be deleted before expiry, with object/subobject selection conditions

The application developer has various ways of using the expiry date ALDATE_DEL and the DEL_BEFORE flag when the function module BAL_LOG_CREATE is called. Examples:

o **Delete log as soon as possible**
  ALDATE_DEL = sy-datum.
  DEL_BEFORE = space.
  The log can be deleted on the day it is created.

o **Delete log in 100 days at the earliest**
  ALDATE_DEL = sy-datum + 100.
  DEL_BEFORE = "X".
  The residence time of 100 days must be specified by the application. Application Log residence times (e.g. dependent on Application Log object/subobject) cannot currently be set in customizing.

o **Retain log for as long as possible**
  ALDATE_DEL = "20981231". (or initial)
  DEL_BEFORE = space.
  This log is not deleted when the delete transaction SLG2 is called with the standard option "Only expired logs". Only the option "Delete logs for which delete before expiry is allowed" deletes the log.

o **Log must not be deleted by the standard transaction**
  ALDATE_DEL = "20981231". (or initial)
  DEL_BEFORE = "X".
  This log can only be explicitly deleted by the application (see next chapter).

= = > **Note**
Transaction SLG2 did not exist before Release 4.6A. Logs used to be deleted by the program RSSLG200 which was not parameterized and deleted all expired logs. It could be scheduled as a job with RSSLG210.
To delete logs before expiry, you had to use the program RSSLGK90 which had selection conditions for Application Log object/subobject, etc.

Delete logs with function modules

= =

Functionality

The function module BAL_DB_DELETE deletes logs from the application.
You can pass the logs to be deleted to the function module in one of three ways.

- I_T_LOG_HANDLE: table with log handles.
  This method is useful if you have kept the LOG_HANDLE in your application.
- I_T_LOGNUMBER: table with log numbers.
  You can use this table if you have the log number LOGNUMBER as reference to the log in the application table and not the LOG_HANDLE (perhaps from older releases)
- I_T_LOGS_TO_DELETE: table with log headers.
  This table is returned by the function module BAL_DB_SEARCH which you use when you have no reference to the log in your application table and the link is established via the field EXTNUMBER in the log header. In this case specify the application log object/subobject and the external number in the BAL_DB_SEARCH filter.

BAL_DB_DELETE is cross-client:

- If you specify I_T_LOG_HANDLE, logs in other clients are also deleted (this is not critically because the log handle is unique)
- If you specify I_T_LOGNUMBER it deletes in the client I_CLIENT. If you do not specify I_CLIENT, it deletes in the current client.
- If you specify I_T_LOG_HEADER the client in the field MANDANT is taken into account (filled by the function module BAL_DB_SEARCH automatically).

Other parameters:

The parameter I_IN_UPDATE_TASK in the function module BAL_DB_DELETE specifies whether the deletion is to be performed in the update task.

The parameter I_WITH_COMMIT_WORK specifies whether the function module BAL_DB_DELETE should COMMIT WORK. This is advantageous if you want to delete a lot of logs with a lot of data. Databases usually restrict the Rollback segment or the number of DB locks for table entries to be deleted. BAL_DB_DELETE works blockwise when I_WITH_COMMIT_WORK = "X" to avoid exceeding this limit.

Note:

The function module BAL_DB_DELETE does not check whether a log can be deleted (expiry date, etc.). These checks must be made in the application.

There are various possible application log deletion scenarios:

- The log must be deleted because the object which points to the log is deleted.
  In this case you can pass LOG_HANDLE, LOGNUMBER, etc. directly to the delete module.
- You must check whether the log can be deleted.
  The most important log header data can be read directly with BAL_DB_SEARCH. The table E_T_LOG_HEADER can be checked and the deletable logs passed to BAL_DB_DELETE.
- The BAL_DB_SEARCH data are not sufficient.
  This procedure is more complicated and should be an exception:

- The log is loaded into main memory with BAL_DB_LOAD (with the option I_DO_NOT_LOAD_MESSAGES = "X" only all log header data is loaded).
- The log header data can now be read with BAL_LOG_HDR_READ, checked, and the deletable logs found.
- The logs loaded are removed from main memory with BAL_LOG_REFRESH …
- … and the deletable logs are finally deleted from the database with BAL_DB_DELETE.

= = > **Note**
The function module BAL_DB_DELETE did not exist before Release 46A. The function modules APPL_LOG_DELETE and APPL_LOG_DELETE_WITH_LOGNUMBER existed instead.
These function modules also deleted logs, but only those which had expired, or which could be deleted before expiry.

= = > **Note**
When logs are deleted, a callback routine, in which you can delete your own log tables, is called (see here).

Change log

Overview

Application Log can create and change logs, whether they are still in memory or have already been saved in the database. These functions are provided by the function modules described below.

Function modules
BAL_DB_ENQUEUE Lock log
BAL_DB_LOAD Load log(s)
BAL_DB_SAVE Save log(s)
BAL_DB_DEQUEUE Unlock log
BAL_LOG_MSG_CHANGE Change message
BAL_LOG_MSG_DELETE Delete message
BAL_LOG_HDR_CHANGE Change log header
BAL_LOG_DELETE Delete log (incl. in DB if saved)
BAL_LOG_REFRESH Delete log from memory

Lock and unlock logs

To change a saved log, you must load it into memory with the function module BAL_DB_LOAD and save it after changing with BAL_DB_SAVE.

You should lock the log with BAL_DB_ENQUEUE, specifying the log handle I_LOG_HANDLE, before loading, to prevent its being changed by two programs at the same time.

You can also use the lock SCOPE parameter to specify when the lock is to be automaticaly reset (see SAP lock concept).

You can unlock a log with BAL_DB_DEQUEUE after saving.

Change log

You can change logs as follows:

- Change message with BAL_LOG_MSG_CHANGE
  Message data (I_S_MSG) can be completely changed. You specify the message handle I_S_MSG_HANDLE.
- Delete message with BAL_LOG_MSG_DELETE
  The message with the message handle I_S_MSG_HANDLE is deleted.
- Change log header data with BAL_LOG_HDR_CHANGE

Log header data (I_S_LOG) can be completely changed. You specify the log handle I_LOG_HANDLE

- Delete log with BAL_LOG_DELETE
  The log in memory with the log handle I_LOG_HANDLE is deleted from memory, and is deleted from the database when you save with BAL_DB_SAVE.

= = > Note
To delete logs only in the database without first loading them into memory, use the function module BAL_DB_DELETE.

= = > Note
To delete a log from memory only, without deleting it physically from the database when saving, use the function module BAL_LOG_REFRESH, e.g. when you have saved a log with BAL_DB_SAVE and then want to delete it from memory.

| Transaction call |

Overview

Application function modules are called in various contexts:

- Dialog
- Bulk processing
- EDI incoming processing
- …

Messages are handled differently in different contexts.

- In dialog, messages may have to be output immediately.
- In bulk processing, messages are first collected and output as a log at the end.
- There are possible mixed forms: if 100 order items are changed in dialog a message is not output for each item, a popup appears at the end with the messages.
- The function module caller may want to decide how messages are handled: do not collect the messages of the called function module because they are too technical, or only collect important messages.

The called function module should ideally not know how its messages are handled. It sends the messenges to the Application Log and the CALLER decides how to handle them (e.g. directly output or collect).

The Application Log can therefore be configured at the start of the transaction. The configuration can be protected from overwriting during the program.

Function modules

BAL_GLB_CONFIG_SET Configure
BAL_GLB_CONFIG_GET Read configuration
BAL_GLB_AUTHORIZATION_GET Authorize
BAL_GLB_AUTHORIZATION_RESET Reset authorization
BAL_GLB_MEMORY_REFRESH (Partially) initialize memory
BAL_MSG_DISPLAY_ABAP Output message as ABAP-MESSAGE

Types

BAL_S_CONF Configuration data
BALAUTH Authorization

Set and read configuration

You can configure the Application Log at the start of the transaction with the BAL_GLB_CONFIG_SET function module.

You must pass the import parameter I_S_CONFIGURATION, with the structure BAL_S_CONF, to the function module.

You can get existing configurations with BAL_GLB_CONFIG_GET.

BAL_S_CONF specifies the messages which Application Log is to collect (component COLLECT) and which are to be output as soon as they are sent (component DISPLAY):

- **COLLECT-INACTIVE**
  COLLECT-INACTIVE = 'X' completely deactivates message collection.
  COLLECT-ACTIVE = ' ' activates message collection.
- **COLLECT-MSG_FILTER, COLLECT-CON_FILTER**
  Specifies which messages are to be collected by Application Log. These filter criteria refer to the message data, where COLLECT-MSG_FILTER filters by message attributes and COLLECT-CON_FILTER by message context.
  These filters have no effect if COLLECT-INACTIVE = 'X' is set.
- **DISPLAY-INACTIVE**
  DISPLAY-INACTIVE = 'X' prevents any messages being displayed when they are sent to the Application Log.
  DISPLAY-INACTIVE = ' ' displays the messages with the callback routine specified in DISPLAY-CALLBACK. If DISPLAY-CALLBACK is empty, DISPLAY-INACTIVE = ' ' has no effect.
- **DISPLAY-MSG_FILTER, DISPLAY-CON_FILTER**
  Specifies, analogously to collect, which messages are to be displayed (if DISPLAY-INACTIVE = ' ' and a callback routine was specified).
- **DISPLAY-CALLBACK**
  This is the message display callback routine which is called when DISPLAY-INACTIVE = ' ' and a message satisfies the specified filter (see also here).

Example 1
The parent program calls BAL_GLB_CONFIG_SET to ensure that Application Log only collects error messages and warnings. All other message types are ignored.

Example 2
Important error messages should be displayed as an ABAP-MESSAGE as soon as they are sent (as well as normal log output at the end of the transaction).

Only one display routine is currently delivered in the Standard (function module BAL_DSP_MSG_DISPLAY_ABAP). This could be useful if a check module is called in background and dialog. Background messages are collected, dialog E messages are displayed, for example in a screen PAI

Other output routines may be developed in future Releases, e.g. to display a message list in an amodal window.

= = > Note
Application Log performance can deteriorate appreciably if you use complex filters, because each message sent has to be checked for whether it should be collected or displayed. Filters should be as simple as possible.
Filters should also not be used to implement customer-defined controls (controllable error messages: the customer can specify conditions in Customizing which determine whether a message is to be collected, and as which message type). Such conditions need complex filters.

= = > **Note**
The message data, including defaults is used to check whether a message is collected or displayed.

The configuration affects the following function modules:
BAL_LOG_MSG_ADD Put message in a log
BAL_LOG_MSG_CUMULATE Add message cumulatively
BAL_LOG_MSG_REPLACE Replace last message
BAL_LOG_MSG_ADD_FREE_TEXT Add message as free text

Authorization

= =
= =

Critical functions such as configuration (function module BAL_GLB_CONFIG_SET) and initialization (function module BAL_GLB_MEMORY_REFRESH) should normally only be performed by the mother program. Lower level routines should not perform these global activities.

Problems can occur when a lower-level routine calls the initialization module, for example because this routine was not originally intended to be called in this context.

You can avoid such effects with authorizations:
The mother program (the first program to have control) can get an authorization at the start with the function module **BAL_GLB_AUTHORIZATION_GET**, which returns a unique key in **E_AUTHORIZATION**.

The critical functions can only be performed by specifying the key **I_AUTHORIZATION**. If **I_AUTHORIZATION** is not specified, or has the wrong value, the action (e.g. initialize memory) is refused. **BAL_GLB_AUTHORIZATION_GET** can not be repeated, so you cannot get a second key.

You can return the key with **BAL_GLB_AUTHORIZATION_RESET** (specifying the key).
All the above function modules can then be called without authorization.

The following function modules require authorization:
BAL_GLB_AUTHORIZATION_GET
BAL_GLB_AUTHORIZATION_RESET
BAL_GLB_CONFIG_SET
BAL_GLB_MEMORY_REFRESH
BAL_GLB_MSG_DEFAULTS_SET
BAL_STATISTICS_GLB_SET

= = > **Note**
To remove the data of a single log from memory use the function module BAL_LOG_REFRESH, which does not require authorization as it only affects one log and not the entire function group memory.

= = > **Note**
Your programs should allow for the fact that an action can be refused and not assume successful performance.

- = = > **Example**
  It is common to initialize memory, call a function and then look for errors in the log. This shifts program exception handling to the log tool, which is not its purpose.

You should not do this because the messages collected by the Application Log and memory reset can be controlled externally, so you cannot control which messages are in the log.

Other function modules

Overview

Other function modules which have not been mentioned previously.

Roll area-independent processing

BAL_GLB_MEMORY_EXPORT puts the function group memory in ABAP-MEMORY.
This data can be fetched again with BAL_GLB_MEMORY_IMPORT. If logs already exist, the imported logs are added to the existing ones.

Data and existence checks

Certain checks are made on Application Log data. The Application Log object in the log header must exist. If you pass a message context, you must also specify the name of the underlying DDIC structure.

These checks are in BAL_LOG_HDR_CHECK and BAL_LOG_MSG_CHECK. They are made automatically when a message or log is created, but they are described here for reasons of modularity (for example if you want to use these checks in your own message collector).

You can check whether a log or message is still in memory (specifying the log or message handles) with the function modules BAL_LOG_EXIST and BAL_LOG_MSG_EXIST.

Read or check Application Log object and subobject

If you specify an object and subobject in a log header, the Application Log checks whether they exist and whether they belong together.

These functions are modular and autonomous and can be accessed externally:
BAL_OBJECT_SELECT reads an Application Log object table record
BAL_SUBOBJECT_SELECT reads a subobject table record
BAL_OBJECT_SUBOBJECT checks whether object and subobject exist and the combination is allowed.

Log display: Detail screens

= =

You can get various detail information about a message and the log header, in the log display, using modular function modules which can also be called independently of the log display. You pass the log or message handle and the language as import parameters.

- o Message detail screens:
  - BAL_DSP_MSG_LONGTEXT:
    Displays message long text.
  - BAL_DSP_MSG_PARAMETERS
    Either outputs the extended long text or calls a CALLBACK routine (depending on BAL_S_MSG-PARAMS)
  - BAL_DSP_MSG_TECHNICAL_DATA
    Outputs the message technical data such as work area, error number, etc.
- o Log header detail screen:
  - BAL_DSP_LOG_PARAMETERS
    Either outputs the extended long text or calls a CALLBACK routine (depending on BAL_S_LOG-PARAMS)
  - BAL_DSP_LOG_TECHNICAL_DATA
    Outputs all log header data

= = > **Note**
These function modules output data like F1 help, i.e. the long text, extended long text, etc. can also be displayed amodally, depending on the user settings (Help -> Settings).

|---| Application Log Callback Routine Overview | |---|

= =

The following information is listed:

- **Purpose and Event**
  What does the callback routine do and when is it called?
- **Definition**
  How is the callback routine set?
- **Parameters**
  - **= = > Note**
    An Application Log callback routine can be realized in two ways:
    as a FORM routine or as a function module
    The following fields must be specified to setup a callback routine:
    USEREXITT: Routine type (' ' = FORM, 'F' = function module)
    USEREXITP: Program containing the routine (only for FORM)
    USEREXITF: Routine name (form routine or function module name)
    A function module must be parameterized like a form routine (USING is replaced by IMPORTING). The same parameter names must be used.

Example program and template
SBAL_CALLBACK
= = > SBAL_CALLBACK = = > SBAL_CALLBACK Coding

BAL_CALLBACK_DISPLAY

= =

Purpose and event
You can specify the appearance of a message when it is created.
For example, all messages (or some) are in an amodal window, to provide constant information about the progress of the program (this is not yet possible).

Definition
In the I_S_CONFIGURATION parameter of the function module BAL_GLB_CONFIG_SET in the I_S_CONFIGURATION-DISPLAY-CALLBACK field.

Parameterization

```
FORM bal_callback_display
       USING
         i_s_msg TYPE bal_s_msg.
...
ENDFORM.
```

Example program

The program SBAL_CALLBACK is an example and template.
You can e.g. select BAL_CALLBACK_DISPLAY in the selection screen of this program. You go to the debugger if:
- this callback routine is defined
- this callback routine is processed
You can also search for the string "BAL_CALLBACK_DISPLAY" in the program coding.

BAL_CALLBACK_DETAIL_LOG

Purpose and event
This callback routine can display user log header detail information. It is called when the cursor is positioned on a log header row and #Detail# is chosen, in the log display.

Definition
The callback routine is set for each log header when a log is created by BAL_LOG_CREATE .
The I_S_LOG-PARAMS-CALLBACK field must be set in the transfer structure I_S_LOG (structure BAL_S_LOG).

Parameterization

```
FORM bal_callback_detail_log
       TABLES
         i_t_params  STRUCTURE spar.
...
ENDFORM.
```

The internal table I_t_params contains the fields:
PARAM (CHAR10) Parameter name
VALUE (CHAR75) Parameter contents.
I_t_params contains the parameters created under BAL_S_LOG-PARAMS-T_PAR for a log.
The table also contains the log number under the name '%LOGNUMBER'.

If this information is insufficient, you can get the data which describes the currently selected objects in the log display, with the function module BAL_DSP_USER_COMMAND_DATA_GET. This data includes the handle of the current log (E_S_USER_COMMAND_DATA-TREE_LOGH). You can use this value to get more log data (e.g. with the function module BAL_LOG_HDR_READ).

Example program
The program SBAL_CALLBACK is an example and template.
You can e.g. select BAL_CALLBACK_DETAIL_LOG in the selection screen of this program. You go to the debugger if:
- this callback routine is defined
- this callback routine is processed
You can also search for the string "BAL_CALLBACK_DETAIL_LOG" in the program coding.

BAL_CALLBACK_DETAIL_MSG

Purpose and event
This callback routine can display user message detail information. It is called when the cursor is positioned on a message row and #Detail# is chosen, or the detail icon next to the message is chosen.

Definition
The callback routine is set for each message when it is sent by BAL_LOG_MSG_ADD. The I_S_MSG-PARAMS-CALLBACK field must be set in the importing parameter I_S_MSG (structure BAL_S_MSG).

Parameterization

```
FORM bal_callback_detail_msg
       TABLES
         i_t_params  STRUCTURE spar.
...
ENDFORM.
```

The internal table I_t_params contains the fields:
PARAM (CHAR10) Parameter name
VALUE (CHAR75) Parameter contents.
I_t_params contains the parameters created under BAL_S_MSG-PARAMS-T_PAR for a message (e.g. using BAL_LOG_MSG_ADD).
The table also contains the log number under the name '%LOGNUMBER', and the four message variables ('V1' to 'V4').

If this information is insufficient, you can get the data which describes the currently selected objects in the log display, with the function module BAL_DSP_USER_COMMAND_DATA_GET. This data includes the handle of the current message (E_S_USER_COMMAND_DATA-LIST_MSGH).
You can use this value to get more log data (e.g. with the function module BAL_LOG_MSG_READ).

Example program
The program SBAL_CALLBACK is an example and template.
You can e.g. select BAL_CALLBACK_DETAIL_MSG in the selection screen of this program. You go to the debugger if:
- this callback routine is defined
- this callback routine is processed
You can also search for the string "BAL_CALLBACK_DETAIL_MSG" in the program coding.

BAL_CALLBACK_READ

Purpose and event
This callback routine reads log display data, e.g. material short text. The routine is called for each message and field defined as external in the field catalog. Read the data buffered to avoid performance problems. You cannot prefetch or read the table of data to be read in one go because it is dynamic.

Definition
The display profile I_S_DISPLAY_PROFILE (structure BAL_S_PROF) is passed in the log display

(e.g. called with BAL_DSP_LOG_DISPLAY). The callback routine is defined in the field I_S_DISPLAY_PROFILE-CLBK_READ. It is called for all fields which have the attribute IS_EXTERN = #X# in the field catalogs LEV1_FCAT, …, LEV9_FCAT or MESS_FCAT.

Parameterization

```
FORM bal_callback_read
       USING
         i_s_info           TYPE bal_s_cbrd
       CHANGING
         c_display_data     TYPE bal_s_show
         c_context_header   TYPE bal_s_cont
         c_context_message  TYPE bal_s_cont
         c_field            TYPE any.
...
ENDFORM.
The structure i_s_info specifies the field for which the callback routine
was called (REF_TABLE and REF_FIELD). Put the contents of the field in
c_field.

You need the other message data (e.g. material number to get material
short text ), to fill c_field.
It is in c_display_data (contains displayable message and log header
data), c_context_header (log header context) and c_context_message
(message context).
```

= = > **Note**

This CALLBACK routine is called at two events, which of them is in the field I_S_INFO-IS_MESSAGE:

1. I_S_INFO-IS_MESSAGE = ' ' = = > at tree creation
2. I_S_INFO-IS_MESSAGE = 'X' = = > when creating message list

The events are (normally) chronologically distinct: the tree is created when the log display appears, the message list when the user selects a set of messages in the tree.

This fact is used to optimize performance: only those fields in the structure c_display_data are filled which are needed at this event.

o = = > **Example**

When the tree is created, you do not need the message text. This would waste time. The message text is fetched when the user has selected e.g. 100 of perhaps 1.000 messages from the tree.

This affects the data in the structure c_display_data:

_S_INFO-IS_MESSAGE = ' '

When called from the tree, only those fields in c_display_data are sure to be filled which are in the field catalogs LEV1_FCAT to LEV9_FCAT.

I_S_INFO-IS_MESSAGE = 'X'

When called for the list, only those fields in c_display_data are sure to be filled which are in MESS_FCAT.

Bear this in mind when you use this callback routine.

Example program

The program SBAL_CALLBACK is an example and template.

You can e.g. select BAL_CALLBACK_READ in the selection screen of this program. You go to the debugger if:
- this callback routine is defined
- this callback routine is processed
You can also search for the string "BAL_CALLBACK_READ" in the program coding.

BAL_CALLBACK_PBO

Purpose and event
This routine sets a user log display menu to integrate other application-specific elements in the log display. It is called at log display PBO.

Definition
In the Display profile CLBK_PBO field.
Parameterization

```
FORM bal_callback_pbo
       USING
         i_t_extab TYPE slis_t_extab.
...
ENDFORM.
```

I_t_extab contains the inactive Fcodes. Pass this table if you want to setup a user menu in this routine:

SET PF-STATUS 'MY_STATUS' EXCLUDING i_t_extab.

= = > Note
You normally create a user menu by copying and modifying an Application Log menu. This has the disadvantage that you are cutoff from future Application Log standard menu changes.
If you only want to put some pushbuttons in the log display, use the component EXT_PUSH1 bis EXT_PUSH4 in the Display profile.

Example program
The program SBAL_CALLBACK is an example and template.
You can e.g. select BAL_CALLBACK_PBO in the selection screen of this program. You go to the debugger if:
- this callback routine is defined
- this callback routine is processed
You can also search for the string "BAL_CALLBACK_PBO" in the program coding.

BAL_CALLBACK_UCOMM, BAL_CALLBACK_BEFORE_UCOMM, BAL_CALLBACK_AFTER_UCOMM

Purpose and event
o **BAL_CALLBACK_UCOMM** is called when a non-Application Log command is issued at PAI.
o **BAL_CALLBACK_BEFORE_UCOMM** is called for such commands and before performing some standard commands.

o **BAL_CALLBACK_AFTER_UCOMM**: is called for such commands and after performing some standard commands.

…_BEFORE_… and …_AFTER_… are performed for the following standard commands:
o **%LONGTEXT** Long text
o **%DETAIL** Detailed message/log header information
o **%TECHDET** Message/log header technical details
o **&IC1** Double-click on message or tree element
o **%EXT_PUSH1** Externally-defined pushbutton 1
o **%EXT_PUSH2** Externally-defined pushbutton 2
o **%EXT_PUSH3** Externally-defined pushbutton 3
o **%EXT_PUSH4** Externally-defined pushbutton 4

Definition
o **BAL_CALLBACK_UCOMM**: in the Display profile, field CLBK_UCOM
o **BAL_CALLBACK_BEFORE_UCOMM**: in the display profile, field CLBK_UCBF
o **BAL_CALLBACK_AFTER_UCOMM**: in the display profile, field CLBK_UCAF

Parameterization
```
FORM bal_callback_ucomm
       CHANGING
         c_s_user_command_data TYPE bal_s_cbuc.
...
ENDFORM.
```

Example program
The program SBAL_CALLBACK is an example and template.
You can e.g. select BAL_CALLBACK_UCOMM in the selection screen of this program. You go to the debugger if:
- this callback routine is defined
- this callback routine is processed
You can also search for the string "BAL_CALLBACK_UCOMM" in the program coding.

Analogously for the other two CALLBACKs.

Data in the callback routines
= =

BAL_S_CBUC parameterizes callback routines which are called by pressing a button in the Application Log log display.

The structure contains current display status information (what has been selected, the cursor position, etc.), and some fields which can be changed in the callback routine (refresh or end display)

The fields are:

o **General fields**

- **UCOMM**

Fcode selected

- **Fields which can be changed in the callback routine**
  - **UCOMM_EXEC**
    'X': command successfully processed.
    ' ': command not processed.
    UCOMM_EXEC can be used when BAL_CALLBACK_CBBF is used and you want to react to a standard command here and not perform the standard.
    CALLBACK_AFTER_UCOMM is always called whether a command was processed or not.
  - **EXIT**
    Leave log display.
  - **REFRESH**
    Refresh log display.
    This can be useful when the underlying messages in the memory have changed (e.g. by BAL_LOG_MSG_CHANGE) or been deleted (BAL_LOG_MSG_DELETE).
    The refresh displays the messages which satisfy the filter criteria specified in the original call (e.g. all messages in a log = > new messages will now also be displayed).
    You can display a different set of messages (e.g. another log) with the function module BAL_DSP_OUTPUT_SET_DATA (the REFRESH flag should not be setz or the display will be constructed twice).
    Caution: Refresh gets all message data (e.g. their texts) again, so it takes as long as the original display and should be used sparingly for large numbers of messages.
  - **MARKS_DEL**
    Delete message selections. This flag is only meaningful if the message selection option (I_S_DISPLAY_PROFILE-MESS_MARK = 'X') was chosen in the log display.
  - **MSGTY, MSGID, MSGNO, MSGV1, MSGV2, MSGV3, MSGV4**
    Message to be output. This can be useful for example to tell the user to select a message for this function.
    Messages (e.g: 'Select a message') are not normally output directly in Application Log processing routines, they are put in the structure c_s_user_command_data, because other processing routines can overwrite them.
- **Navigation tree information**
  - **TREE_CLICK**
    The user double-clicked on the tree.
  - **TREE_LEVEL**
    Tree level selected
  - TREE_TABLE, TREE_FIELD, TREE_VALUE
    Table name, field name and field contents selected (if only one field was selected)
  - **TREE_SELF**
    Table selected with field names and contents selected (if several fields were selected, e.g. 'User/Date/Time').
  - **TREE_LOGH**
    Handle of log selected in the tree (if one log was selected). This is e.g. the case if one

log was selected in the standard log display (transaction SLG1) at the highest tree level.

- **TREE_MSGH**
  Handle of the message selected in the tree.
  This field is only filled if messages are displayed in the tree.
  This is the case if the display was called with I_S_DISPLAY_PROFILE-BYDETLEVEL = 'X', e.g. with the standard profile from function module BAL_DSP_PROFILE_DETLEVEL_GET.

o **Message list information**

- **LIST_MSGH**
  Message selected in the list (by positioning the cursor)

- **LIST_TMSGH**
  Set of messages selected.
  This field is only filled if the select several messages option (I_S_DISPLAY_PROFILE-MESS_MARK = 'X') was chosen in the display profile.

- **LIST_TABLE, LIST_FIELD, LIST_VALUE**
  Table name, field name and field contents selected in the display.

o **Internal fields**

- **LIST_SEL, LIST_COL, LIST_ROW, LIST_TABIX, TREE_NODE, TREE_ITEM, LIST_REFR**

CALLBACK_DB_DELETE

Purpose and event
This routine is called when logs are deleted from the database.
It can delete data which you put in your own database tables for the log (e.g. index tables).

Definition
The definition is a little unusual in that the callback routine can only be a function module which obeys the naming convention:
If **ABC** is the name of the Application Log object defined in the transaction SLG0, the function module **BAL_DBDEL_ABC** is called when a log which has the object **ABC** in its log header is deleted in the database. The subobject is not relevant.

Parameterization

```
FUNCTION BAL_DBDEL_...
*"----------------------------------------------------------------------
*"*"Local interface:
*"       IMPORTING
*"             REFERENCE(I_T_LOGS_TO_DELETE)   TYPE  BALHDR_T
*"             REFERENCE(I_IN_UPDATE_TASK)     TYPE  BOOLEAN
*"----------------------------------------------------------------------
...
ENDFUNCTION.
```

I_T_LOGS_TO_DELETE is the table of log headers to be deleted

If I_IN_UPDATE_TASK = 'X' the deletion is performed in the update task