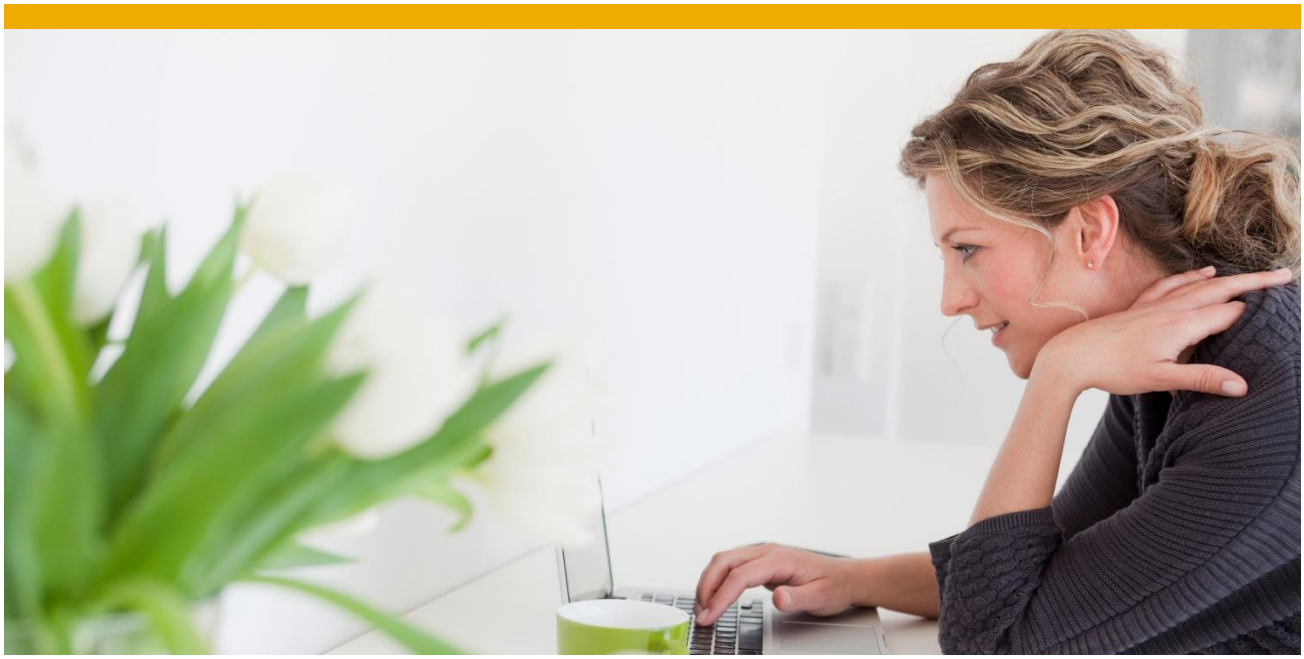


TM Logging for Error Analysis

TABLE OF CONTENTS

PROBLEM SITUATION

You have a customer problem in which unexpected situations arise and a normal debugging analysis is difficult or impossible. A typical situation is a data inconsistency which happens in irregular points in time and which is not reproducible. You want to do a detailed analysis based on some contextual data available during the running transaction.

PROBLEM SOLUTION

A possible solution is to do a logging of the transaction context in cases an exceptional situation arises. The standard way to monitor exceptional situations in coding is to use checkpoint groups and write a log using LOG-POINT or ASSERT commands in ABAP coding. This idea was implemented in January 2013 by TM2 (contact: Bernhard Hauser).

EXAMPLE: INITIAL ITEM IDS (NESTLE CUSTOMER MESSAGE)

The problem in the customer system (TM 8.0) was that TOR item IDs were sometimes initial. However, the problem was not reproducible.

IMPLEMENTING AN OWN LOGGING

The following parts are needed to do one's own logging.

To provide a solution a logging class was implemented:

Class Interface /SCMTMS/CL_LOGGING Implemented / Active

Properties | Interfaces | Friends | Attributes | Methods | Events | Types

Parameter | Exception

Method	Level	Vis...	M...	Description
DELETE_LOG	Insta...	Pub...		Delete log from DB
DISPLAY_LOG	Insta...	Pub...		Display log on screen
GET_INSTANCE	Stati...	Pub...		Get singleton instance
GET_LOG_GUID	Stati...	Pub...		Get Log GUID
LOG_CALLSTACK	Insta...	Pub...		Log the current callstack
LOG_CURRENT_CONTEXT	Stati...	Pub...		Log current system context
LOG_INTERNAL_TABLE	Insta...	Pub...		Log contents of an internal table
LOG_ITAB_COLNAME	Insta...	Pri...		Logging of column name
LOG_ITAB_VALUE	Insta...	Pri...		Logging of column value
LOG_STRING	Insta...	Pub...		Add to log buffer
READ_LOG	Insta...	Pub...		Reads buffer from DB
SET_MAX_LOGS	Insta...	Pub...		Set the maximum number of allowed logs
WRITE_LOG	Insta...	Pub...		Writes buffer to DB

The logging is based the 2 DB tables `/scmtms/d_log` (log header) and `/scmtms/d_logitm` (log items) for temporary data. There are methods to read data from the log tables (READ_LOG), to write data to an internal log buffer (LOG_* methods) and to write the log buffer to the log tables (WRITE_LOG). The log can be deleted explicitly (DELETE_LOG).
Specifically methods are provided to log:

- The callstack (LOG_CALLSTACK)

- Any internal table and its contents (LOG_INTERNAL_TABLE)
- Log the whole context (consisting of callstack and optionally a table) and write the log to DB.

CODING IMPLEMENTATION

Consider the following implementation in class /SCMTMS/CL_TOR_D_ITEM_AC, method PROCESS_DATA (in B1D):
The following coding counts the number of initial item IDs:

```
LOOP AT ct_tor_item_all INTO ls_item.
  IF ls_item-item_id IS INITIAL.
    lv_num_initial_id = lv_num_initial_id + 1.
  ENDIF.
  CLEAR lt_chg_fields.
```

The actual coding to create a log for a defined checkpoint group and to provide a detailed logging is then done like this:

```
IF lv_num_initial_id > 0. " invalid state: do logging of intial Item IDs
  lv_log_guid = /scmtms/cl_logging=>get_log_guid( ).
  WRITE: lv_log_guid TO lv_log_guid_subkey.
  ASSERT ID /scmtms/tor_log
    SUBKEY lv_log_guid_subkey
    FIELDS lv_log_guid
    CONDITION /scmtms/cl_logging=>log_current_context(
              it_table = ct_tor_item_all iv_log_guid = lv_log_guid ) = abap_false.
ENDIF.
```

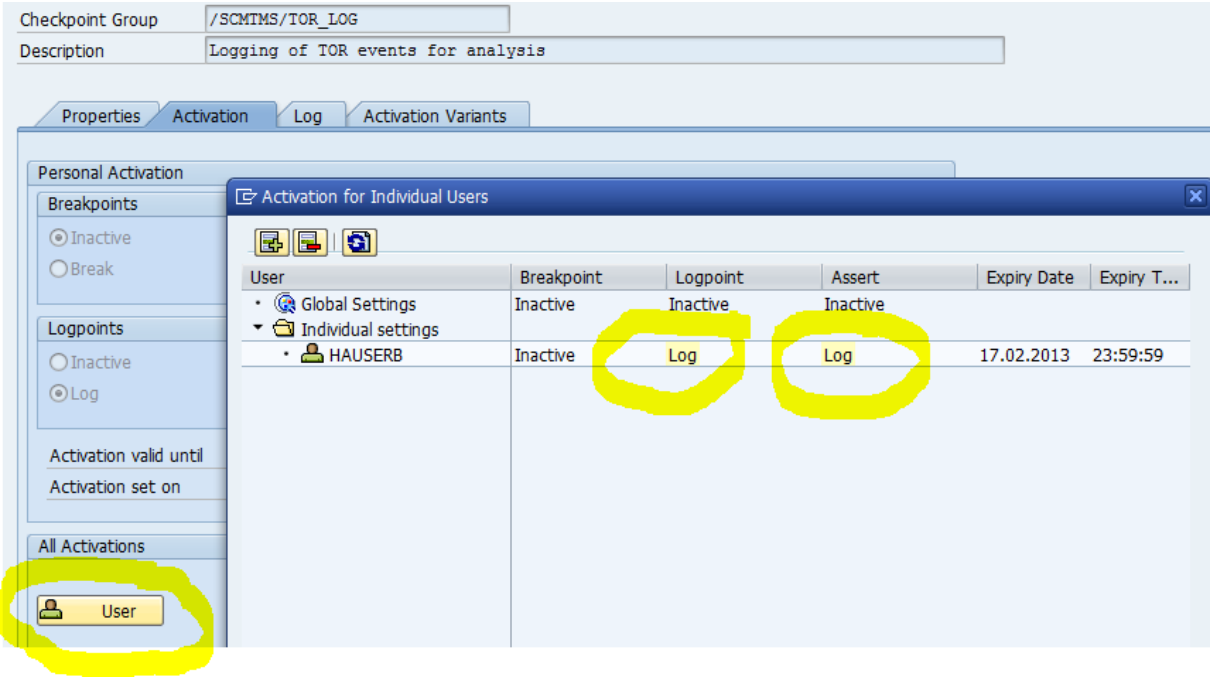
The coding is therefore only executed in exception situations (i.e. number of initial Item IDs > 0). Then a GUID is generated that identifies the log. With the ASSERT command, the log_current_context method of the logging class is called. By this, 2 things are done at the same time:

- A log entry is written for checkpoint group /SCMTMS/TOR_LOG (note that the checkpoint group needs to be activated, see below for details).
- A detailed log is written into the log tables, including the callstack and the internal table of all TOR items which is relevant in this context.

That's all! So it's just about 5-10 lines of coding!

CHECKPOINT GROUP ACTIVATION

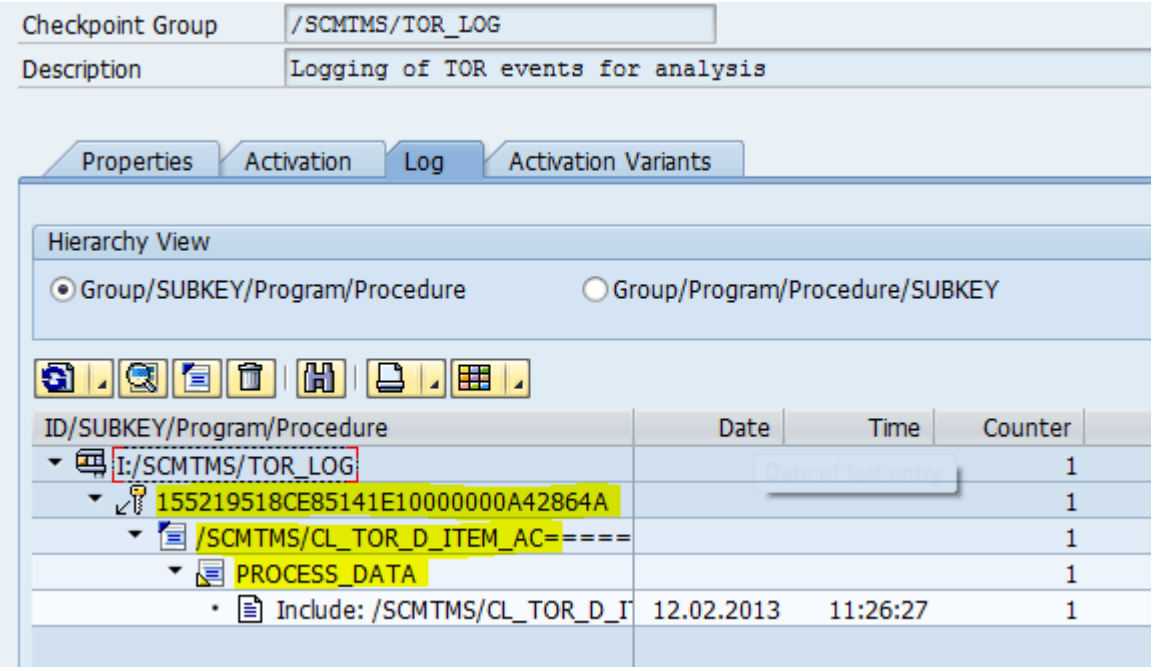
To enable logging for the checkpoint group, use transaction SAAB (select activate button):



In this case the logging is active only for user HAUSERB. Logging is done for any ASSERT or LOG-POINT commands with this checkpoint group.

TEST RUN

After a “positive case” of the problem happens, a log entry is created in the checkpoint group which looks like this:

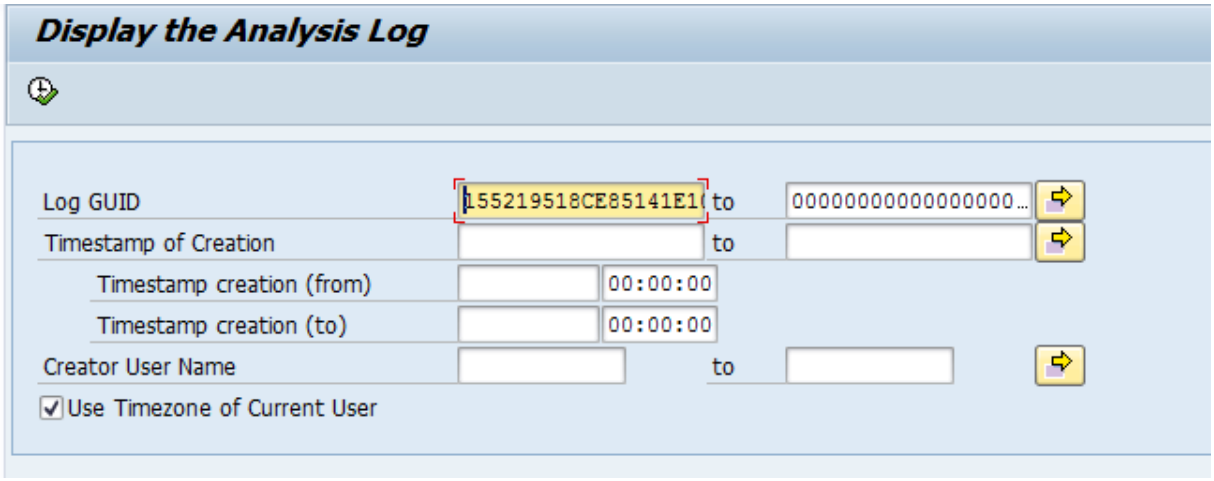


As can be seen, the GUID is used as a (sub)key. This has 2 advantages:

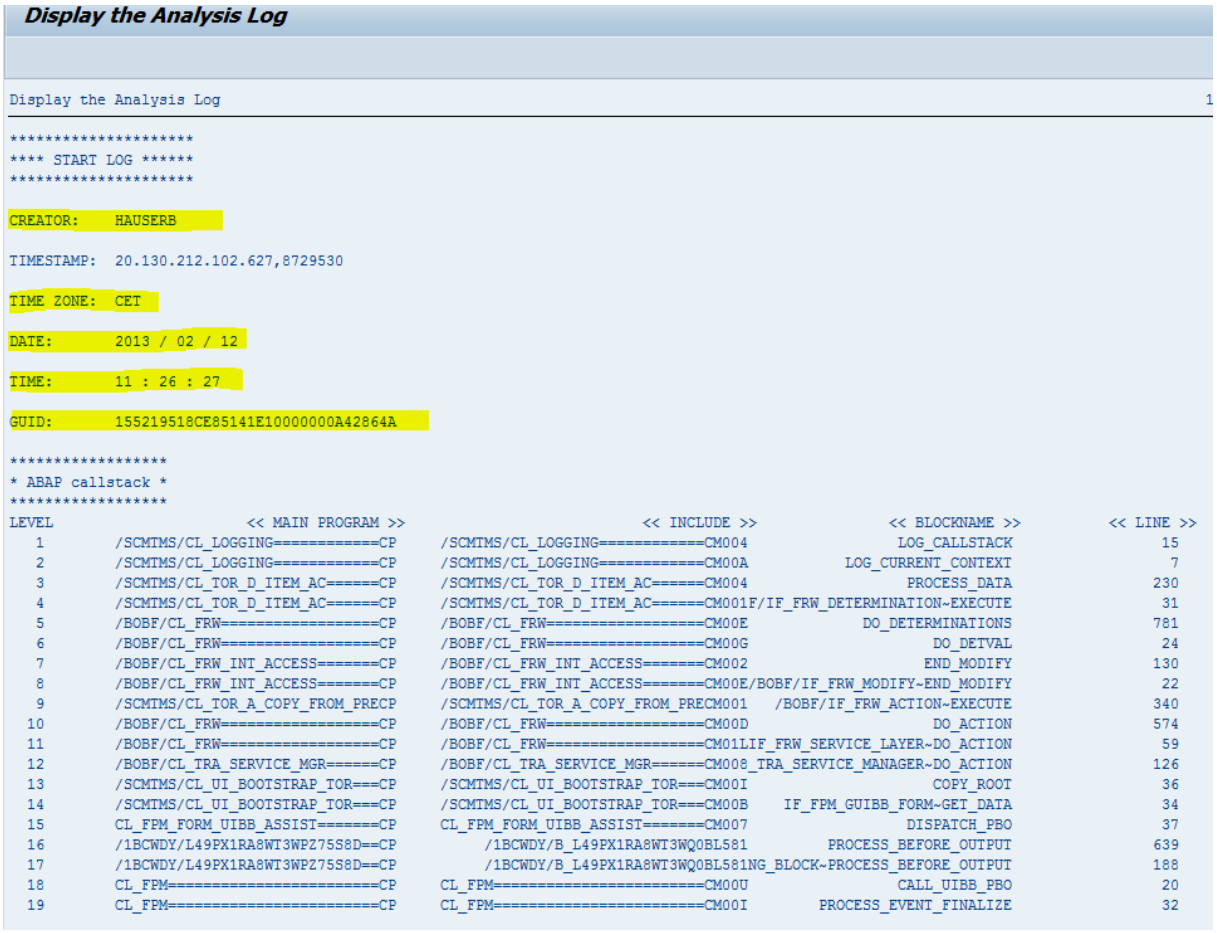
- The GUID uniquely identifies the log so taking the GUID, you can select just the log you’re interested in.
- If the same problem occurs in this coding, separate log entries are written in the checkpoint group (without the usage of the GUID, the counter on the right would be increased only).

DISPLAY THE LOG

Run report /SCMTMS/LOG_DISPLAY:



The GUID can be entered here, then only the relevant log is displayed.
Instead, based on a time window, logs can be selected, or based on user criteria. To show the log depending on user time zone, use the indicator.
The following snapshot shows the display of the log:



Display the Analysis Log

Display the Analysis Log 1

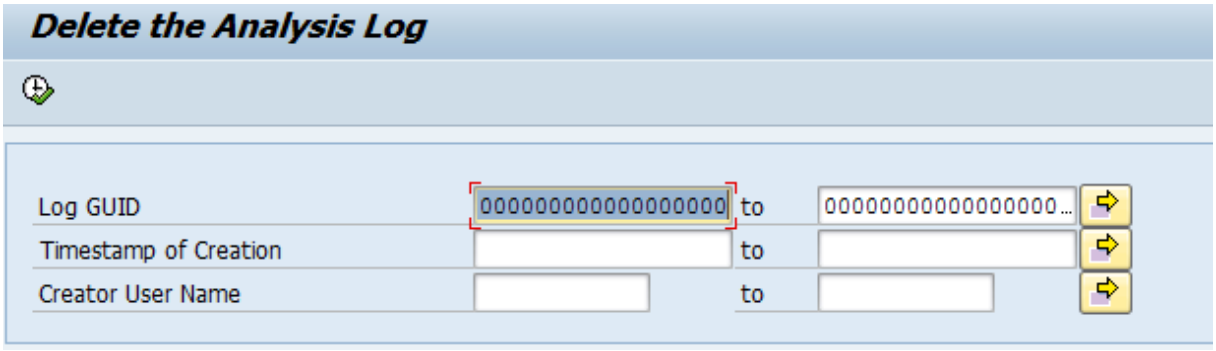
```
*********************
**** START LOG ******
*********************

CREATOR:    HAUSERB

TIMESTAMP:  20.130.212.102.627,8729530

TIME ZONE:  CET

DATE:       2013 / 02 / 12

TIME:       11 : 26 : 27

GUID:       155219518CE85141E10000000A42864A

******************
* ABAP callstack *
******************
```

LEVEL	<< MAIN PROGRAM >>	<< INCLUDE >>	<< BLOCKNAME >>	<< LINE >>
1	/SCMTMS/CL_LOGGING==========CP	/SCMTMS/CL_LOGGING==========CM004	LOG_CALLSTACK	15
2	/SCMTMS/CL_LOGGING==========CP	/SCMTMS/CL_LOGGING==========CM00A	LOG_CURRENT_CONTEXT	7
3	/SCMTMS/CL_TOR_D_ITEM_AC======CP	/SCMTMS/CL_TOR_D_ITEM_AC======CM004	PROCESS_DATA	230
4	/SCMTMS/CL_TOR_D_ITEM_AC======CP	/SCMTMS/CL_TOR_D_ITEM_AC======CM001F/IF_FRW_DETERMINATION~EXECUTE		31
5	/BOBF/CL_FRW==================CP	/BOBF/CL_FRW==================CM00E	DO_DETERMINATIONS	781
6	/BOBF/CL_FRW==================CP	/BOBF/CL_FRW==================CM00G	DO_DETVAL	24
7	/BOBF/CL_FRW_INT_ACCESS=======CP	/BOBF/CL_FRW_INT_ACCESS=======CM002	END_MODIFY	130
8	/BOBF/CL_FRW_INT_ACCESS=======CP	/BOBF/CL_FRW_INT_ACCESS=======CM00E/BOBF/IF_FRW_MODIFY~END_MODIFY		22
9	/SCMTMS/CL_TOR_A_COPY_FROM_PRECP	/SCMTMS/CL_TOR_A_COPY_FROM_PRECM001 /BOBF/IF_FRW_ACTION~EXECUTE		340
10	/BOBF/CL_FRW==================CP	/BOBF/CL_FRW==================CM00D	DO_ACTION	574
11	/BOBF/CL_FRW==================CP	/BOBF/CL_FRW==================CM01LIF_FRW_SERVICE_LAYER~DO_ACTION		59
12	/BOBF/CL_TRA_SERVICE_MGR======CP	/BOBF/CL_TRA_SERVICE_MGR======CM008_TRA_SERVICE_MANAGER~DO_ACTION		126
13	/SCMTMS/CL_UI_BOOTSTRAP_TOR===CP	/SCMTMS/CL_UI_BOOTSTRAP_TOR===CM00I	COPY_ROOT	36
14	/SCMTMS/CL_UI_BOOTSTRAP_TOR===CP	/SCMTMS/CL_UI_BOOTSTRAP_TOR===CM00B	IF_FPM_GUIBB_FORM~GET_DATA	34
15	CL_FPM_FORM_UIBB_ASSIST=======CP	CL_FPM_FORM_UIBB_ASSIST=======CM007	DISPATCH_PBO	37
16	/1BCWDY/L49PX1RA8WT3WPZ75S8D==CP	/1BCWDY/B_L49PX1RA8WT3WQ0BL581	PROCESS_BEFORE_OUTPUT	639
17	/1BCWDY/L49PX1RA8WT3WPZ75S8D==CP	/1BCWDY/B_L49PX1RA8WT3WQ0BL581NG_BLOCK~PROCESS_BEFORE_OUTPUT		188
18	CL_FPM========================CP	CL_FPM========================CM00U	CALL_UIBB_PBO	20
19	CL_FPM========================CP	CL_FPM========================CM00I	PROCESS_EVENT_FINALIZE	32

The most important attributes are written at the beginning of the log. Following is the callstack and some internal table data.

LOG DELETION

Use report /SCMTMS/delete_log to delete the log using selection criteria.



The successful deletion is shown at the end of the report.

PERFORMANCE ASPECTS

Note that when the checkpoint group is deactivated, there is nearly no impact on performance (the only small penalty is that in case of an exceptional situation, a GUID is created). As exceptional situation occur rather rarely, this can be ignored.

RESTRICTING THE NUMBER OF LOGS

In order to restrict the number of logs written to the DB (so the data volume of the customer database does not grow too much), the logging class provides the method SET_MAX_LOGS that sets an upper limit to the number of logs written to the DB. In case this number is exceeded, no further logging is done. The standard default value is 20.

www.sap.com