SAP
Case Management

Case Management

Developer Documentation

January 15, 2004

Contents

1 Einleitung

Case Management has been designed as a reuse component. Applications are provided with a range of options for integrating and customizing Case Management. Most of the activities required to customize Case Management are incorporated as Customizing settings in the Implementation Guide (IMG). Each of the activities is documented online.

The following documentation refers to those topics that require you to perform implementation tasks:

- System-driven execution of case-related activities; for this, you use Case Management API.
- Including an individual subcomponent.
- Implementation of Business Add-Ins (BAdIs)

2 Case Management API

Case Management API enables you to use the Case Management functions at the program level and not on the front end. To use the API, implement the IF_SCMG_CASE_API interface with the relevant methods.

2.1 Update

The update must be activated centrally using the methods *Create* or OPEN_CASE (with a parameter). All other methods evaluate the settings made here.

2.2 Connection to the Framework

To implement the API, you need to connect it to the framework. The methods can be called from an active framework. To make this connection, call the *set_root_object* method. This makes sure that the active framework is also used for the API methods.

If the program is independent, and is not provided with a framework, then each of the methods is checked implicitly to see whether the framework is activated. The framework is activated if it is inactive.

2.3 Methods of the IF_SCMG_CASE_API Interface

2.3.1 ATTRIBUTE_ALL_GET

Description: Get all attributes

Call sequence: OPEN_CASE -> ATTRIBUTE_ALL_GET …->CLOSE_CASE

Parameters:

IMPORTING:

IM_CHECK_AUTHORITY	TYPE	XFELD (optional)
	DEFAULT	SPACE
IM_RESELECT_DB	TYPE	SRMBOOLEAN (optional)
	DEFAULT	IF_SRM=>FALSE

EXPORTING:

EX_SKIPPED	TYPE	I
EX_VALUES	TYPE	TY_NAMEVALUEASSTRING

Exceptions:

FAILED

Example:

```
l_case = cl_scmg_case_api=>OPEN_CASE( IM_CASE_GUID = l_guid ).

Call Method lcl_case->attribute_all_get
     EXPORTING
           IM_CHECK_AUTHORITY  = 'X'
           IM_RESELECT_DB      = IF_SRM=>TRUE
     IMPORTING
           EX_Skipped          = MySkip
           EX_VALUES           = MyValues.
```

2.3.2 ATTRIBUTE_CHANGE

Description: Change a case attribute

Call sequence: OPEN_CASE -> ATTRIBUTE_GET -> ATTRIBUTE_CHANGE -> SAVE ->CLOSE_CASE

Parameters:

IMPORTING:

IM_VALUE	TYPE	STRING
IM_FIELDNAME	TYPE	STRING
IM_CHECK_AUTHORITY	TYPE	XFELD (optional)
	DEFAULT	SPACE

Exceptions:

FAILED

NO_AUTHORITY

Example:

```
l_case = cl_scmg_case_api=>OPEN_CASE( IM_CASE_GUID = l_guid ).

Call Method lcl_case->attribute_change
     EXPORTING
           IM_VALUE     = 'MyValue'
           IM_FIELDNAME = 'MyFieldname'.
```

2.3.3 ATTRIBUTE_GET

Description: Get a case attribute

Call sequence: OPEN_CASE -> ATTRIBUT_GET …->CLOSE_CASE

Parameters:

IMPORTING:

IM_FIELDNAME	TYPE	STRING
IM_CHECK_AUTHORITY	TYPE	XFELD
	DEFAULT	SPACE
IM_RESELECT_DB	TYPE	SRMBOOLEAN (optional)
	DEFAULT	IF_SRM=>FALSE

RETURNING:

RE_VALUE TYPE STRING

Exceptions:

FAILED

NO_AUTHORITY

Example:

```
l_case = cl_scmg_case_api=>OPEN_CASE( IM_CASE_GUID = l_guid ).

l_value = lcl_case->attribute_get( IM_FIELDNAME = 'MyFieldName'
                                   im_check_authority ='X'
                                   im_reselect_db = if_srm=>true).
```

2.3.4 AUTHORITY_CHECK

Description: Check the authorization for an activity

Call sequence: OPEN_CASE -> AUTHORITY_CHECK ...->CLOSE_CASE

Parameters:

IMPORTING:

IM_ACTIVITY TYPE SCMG_AUT_ACTIVITY

RETURNING:

RE_RESULT TYPE SY-SUBRC

Exceptions:

FAILED

Example:

```
l_case = cl_scmg_case_api=>OPEN_CASE( IM_CASE_GUID = l_guid ).

l_rc = lcl_case->authority_check( IM_ACTIVITY = 'DISP' ).

IF l_rc = 0.
*     Authorization exists
ELSE.
*      No authorization
ENDIF.
```

2.3.5 AUTHORITY_CHECK_FIELD

Description: Check authorization for accessing a field

Call sequence: OPEN_CASE ->AUTHORITY_CHECK_FIELD ...->CLOSE_CASE

Parameters:

IMPORTING:

IM_FIELDNAME TYPE FIELDNAME

IM_ACTIVITY_FIELD TYPE ACTIV_AUTH

RETURNING:

RE_RESULT TYPE SY-SUBRC

Exceptions:

FAILED

Example:

```
l_case = cl_scmg_case_api=>OPEN_CASE( IM_CASE_GUID = l_guid ).

l_rc = lcl_case->authority_check_field( IM_FIELDNAME = 'CASE_TITLE'
                                        IM_ACTIVITY_FILED = '03'  ).

IF l_rc = 0.
*     Authorization exists
ELSE.
*      No authorization
ENDIF.
```

2.3.6 BOR_OBJECTS_GET

Description: Get all BOR objects (for a specific anchor). The anchor must be defined in the record model.

Call sequence: OPEN_CASE -> BOR_OBJECTS_GET …->CLOSE_CASE

Parameters:

IMPORTING:

IM_ANCHOR	TYPE	STRING (optional)
IM_BOR_TYPE	TYPE	OJ_NAME (optional)
IM_BOR_KEY	TYPE	SWO_TYPEID (optional)

RETURNING:

RE_BOR_OBJECTS	TYPE	SCMG_TT_BOR_IN_RECORD

Exceptions:

FAILED

Example:

```
l_case = cl_scmg_case_api=>OPEN_CASE( IM_CASE_GUID = l_guid ).

CALL METHOD lcl_case->bor_objects_get
       EXPORTING
          IM_ANCHOR      = 'MyTEST'
          IM_BOR_TYPE    = MyBorTyp
          IM_BOR_KEY     = MyBorKey
       RECEIVING
          RE_BOR_OBJECTS = lt_bor_objects.
```

2.3.7 BOR_OBJECT_DELETE

Description: Delete a BOR object in the case record. You have two options: Either you specify the BOR object using the parameters IM_ANCHOR, IM_BOR_KEY, IM_BOR_TYPE, and IM_SPS_ID (in this case, all these parameters are mandatory), or you specify the BOR object using the node ID. You then need only to specify the IM_ID parameter. In the second variant, all objects below the record node are deleted.

Call sequence: OPEN_CASE -> BOR_OBJECT_DELETE -> SAVE …->CLOSE_CASE

Parameters:

IMPORTING:

IM_ID	TYPE	STRING (optional)

IM_ANCHOR	TYPE	STRING (optional)
IM_BOR_KEY	TYPE	SWO_TYPEID (optional)
IM_BOR_TYPE	TYPE	OJ_NAME (optional)
IM_SPS_ID	TYPE	STRING (optional)

Exceptions:

FAILED

INVALID_PARAMETERS

Example:

```
 l_case = cl_scmg_case_api=>OPEN_CASE( IM_CASE_GUID = l_guid ).

 CALL METHOD lcl_case->bor_object_delete
      EXPORTING
            IM_ANCHOR    = 'MyTEST'
            IM_BOR_TYPE  = MyBorTyp
            IM_BOR_KEY   = MyBorKey
            IM_SPS_ID    = MySPSid.

l_case->save( IM_DEQUEUE  = IF_SRM=>TRUE ).
l_case->close_case( ).
```

Or alternatively:

```
CALL METHOD lcl_case->bor_object_delete
      EXPORTING
            IM_ID        = '3'.
```

2.3.8 BOR_OBJECT_INSERT

Description: Insert a BOR object in the case record

Explanation: The anchor must be defined in the record model; BOR_KEY is the key of the fixed BOR object; BOR_TYPE is the BOR type; and SPS_ID is the element type for the BOR type in Case Management.

Call sequence: OPEN_CASE -> BOR_OBJECT_INSERT -> SAVE …->CLOSE_CASE

Parameters:

IMPORTING:

IM_ANCHOR	TYPE	STRING
IM_BOR_KEY	TYPE	SWO_TYPEID
IM_BOR_TYPE	TYPE	OJ_NAME
IM_DESCRIPTION	TYPE	STRING (optional)
IM_SPS_ID	TYPE	STRING
IM_RELATIONS	TYPE	SRM_XML_RELA_TAB (optional)
IM_PROPERTIES	TYPE	TY_NAMEVALUEASSTRING (optional)

Exceptions:

FAILED

NOT_ENQUEUED

Example:

```
 l_case = cl_scmg_case_api=>OPEN_CASE( IM_CASE_GUID = l_guid ).
```

```
CALL METHOD lcl_case->bor_object_insert
     EXPORTING
           IM_ANCHOR     = 'MyTEST'
           IM_BOR_TYPE   = MyBorTyp
           IM_BOR_KEY    = MyBorKey
           IM_SPS_ID     = MySPSid
           IM_PROPERTIES = lt_properties.

l_case->save( IM_DEQUEUE = IF_SRM=>TRUE ).
l_case->close_case( ).
```

2.3.9 CLOSE_CASE

Description: Close the case after it has been processed

Explanation: The case is unlocked and the connection to the back end is closed.

Example:

```
l_case = cl_scmg_case_api=>OPEN_CASE( IM_CASE_GUID = l_guid ).
...


l_case->close_case( ).
```

2.3.10 CREATE

Description: Create a case

Explanation: After an OPEN_CASE, a decision is made about whether the case is updated or not (IM_UPDATE_TASK = IF_SRM=>FALSE). By default, there is no update.

The IM_WITH_PATH parameter is obsolete and is no longer evaluated. The process route is now created in accordance with the configuration in the case type.

The IM_PATH_TEMPLATE parameter can be used to load a process route model into the current process route directly after the case is created.

Call sequence: CREATE -> GET_BACKEND_RECORD -> SAVE …->CLOSE_CASE

Parameters:

IMPORTING:			
	IM_CASE_TYPE	TYPE	SCMGCASE_TYPE
	IM_CHECK_AUTHORITY	TYPE	XFELD (optional)
		DEFAULT	SPACE
	IM_CASE_GUID	TYPE	SCMG_CASE_GUID (optional)
(	IM_WITH_PATH	TYPE	XFELD (optional)
		DEFAULT	SPACE) (obsolete)
	IM_PATH_TEMPLATE	TYPE	SRMWFPTHID (optional)
	IM_UPDATE_TASK	TYPE	SRMBOOLEAN (optional)
		DEFAULT IF_SRM=>FALSE	
RETURNING:			
	RE_CASE	TYPE REF TO	IF_SCMG_CASE_API

Exceptions:

FAILED

NO_AUTHORITY

Example:

```
CALL METHOD cl_scmg_case_api=>create
       EXPORTING
         IM_CASE_TYPE   = casetyp
         IM_UPDATE_TASK = IF_SRM=>TRUE
       RECEIVING
         re_case        = l_case
       EXCEPTIONS
         OTHERS         = 1.

l_case->save( IM_DEQUEUE = IF_SRM=>TRUE ).
l_case->close_case( ).
```

2.3.11 DELETE

Description: Delete the case

Explanation: An internal flag is set that prevents the API functions from being used to process this case any further. The case also cannot be saved.

Call sequence: OPEN_CASE -> DELETE

Parameters:

IMPORTING:

IM_CHECK_AUTHORITY	TYPE	XFELD (optional)
	DEFAULT	SPACE
IM_DELETE_FROM_CONTAINER	TYPE	SRMBOOLEAN (optional)
	DEFAULT	IF_SRM=>FALSE

Exceptions:

FAILED

NO_AUTHORITY

CONTAINER_ENQUEUED

Example:

```
l_case = cl_scmg_case_api=>OPEN_CASE( IM_CASE_GUID = l_guid ).

CALL METHOD lcl_case->delete
   EXPORTING
     IM_CHECK_AUTHORITY       = 'X'
     IM_DELETE_FROM_CONTAINER = 'X'.
```

2.3.12 DEQUEUE

Description: Unlock a case

Call sequence: OPEN_CASE -> … -> DEQUEUE …->CLOSE_CASE

Parameters:

RETURNING:

RE_OKAY	TYPE	XFELD

Exceptions:

FAILED

Example:

```
l_case = cl_scmg_case_api=>OPEN_CASE( IM_CASE_GUID = l_guid IM_ENQUEUE = 'X' ).

. . .
l_case->dequeue( ).
```

2.3.13 ELEMENT_DELETE

Description: Delete an object in the case record. You have two options: Either you specify the object using the parameters IM_ANCHOR, IM_SP_POID, and IM_SPS_ID (in this case, all these parameters are mandatory), or you specify the object using the node ID. You then need only to specify the IM_ID parameter. In the second variant, all objects below the specified record node in the hierarchy are deleted.

Call sequence: OPEN_CASE -> ELEMENT_DELETE -> SAVE …->CLOSE_CASE

Parameters:

IMPORTING:

IM_ANCHOR	TYPE	STRING (optional)
IM_SPS_ID	TYPE	STRING (optional)
IM_SP_POID	TYPE	SRM_LIST_POID (optional)
IM_ID	TYPE	STRING (optional)

Exceptions:

FAILED

INVALID_PARAMETERS

Example:

```
l_case = cl_scmg_case_api=>OPEN_CASE( IM_CASE_GUID = l_guid ).

CALL METHOD lcl_case->element_delete
      EXPORTING
            IM_ANCHOR    = 'MyTEST'
            IM_SPS_ID    = MySPSid
            IM_SP_POID   = lt_sp_poid.

l_case->save( IM_DEQUEUE  = IF_SRM=>TRUE ).
l_case->close_case( ).
```

Or alternatively:

```
CALL METHOD lcl_case->element_delete
      EXPORTING
            IM_ID        = '3'.
```

2.3.14 ELEMENT_INSERT

Description: Insert an object in the case record

Explanation: There are two options for inserting elements, either using the model ID (the node ID in the record model) or using the anchor. The anchor must then be defined in the record model. One of these two parameters must be specified. If you specify both parameters, then the object is inserted in the specified anchor. In both cases, you must specify the element type (SPS ID) and the corresponding parameters for finding the POID of the object.

Call sequence: OPEN_CASE -> ELEMENT_ADD -> SAVE …->CLOSE_CASE

Parameters:

IMPORTING:

IM_ANCHOR	TYPE	STRING (optional)
IM_SP_POID	TYPE	SRM_LIST_POID
IM_SPS_ID	TYPE	STRING
IM_MODEL_ID	TYPE	STRING (optional)
IM_DESCRIPTION	TYPE	STRING (optional)
IM_RELATIONS	TYPE	SRM_XML_RELA_TAB (optional)
IM_PROPERTIES	TYPE	TY_NAMEVALUEASSTRING (optional)

Exceptions:

FAILED

NOT_ENQUEUED

Example:

```
l_case = cl_scmg_case_api=>OPEN_CASE( IM_CASE_GUID = l_guid ).

 CALL METHOD lcl_case->element_insert
      EXPORTING
            IM_ANCHOR          = 'MyTEST'
            IM_SP_POID         = lt_sp_poid
            IM_SPS_ID          = MySPSid
            IM_DESCRIPTION     = MyDescription
            IM_PROPERTIES      = lt_properties.

 l_case->save( IM_DEQUEUE = IF_SRM=>TRUE ).
l_case->close_case( ).
```

Or alternatively:

```
 CALL METHOD lcl_case->element_insert
      EXPORTING
            IM_SP_POID         = lt_sp_poid
            IM_SPS_ID          = MySPSid
            IM_MODEL_ID        = ,2'
            IM_DESCRIPTION     = MyDescription
            IM_PROPERTIES      = lt_properties.
```

2.3.15 ENQUEUE

Description: Lock a case

Explanation: If OPEN_CASE is not called with the lock parameter, then you can use ENQUEUE to lock the case.

Call sequence: OPEN_CASE -> … -> ENQUEUE -> ... -> DEQUEUE …->CLOSE_CASE

Parameters:

IMPORTING:

IM_SCOPE	TYPE	STRING (optional)
	DEFAULT	IF_SRM_SP_ENQUEUE=>SCOPE_DIALOG
IM_MODE	TYPE	STRING (optional)
	DEFAULT	IF_SRM_SP_ENQUEUE=>MODE_EXCLUSIVE

RETURNING:

RE_OKAY TYPE XFELD

Exceptions:

FAILED

ENQUEUED

Example:

```
l_case = cl_scmg_case_api=>OPEN_CASE( IM_CASE_GUID = l_guid ).

CALL METHOD l_case->enqueue
     EXPORTING
            IM_MODE  = myMode
            IM_SCOPE = myScope
     RECEIVING
            RE_OKAY  = l_enqueued.
     EXCEPTIONS
            ENQUEUED = 1        “enqueue failed
            OTHERS   = 2.       “other failure

. . .

l_case->dequeue( ).
```

2.3.16 FIND_CASES_BY_ATTRIBUTE

Description: Find cases with the given attributes

Call sequence: FIND_CASES_BY_ATTRIBUTE

Parameters:

IMPORTING:

IM_SEARCH_FIELDS TYPE TY_NAMEVALUEASSTRING

CHANGING:

CH_CASE_GUIDS TYPE SCMG_TT_CASE_GUID (optional)

Exceptions:

FAILED

NOTHING_FOUND

Example:

```
CALL METHOD cl_scmg_case_api=>find_cases_by_attribute
     EXPORTING
            IM_SEARCH_FIELDS = l_search
     CHANGING
            CH_CASE_GUIDS    = l_guids.
```

2.3.17 FIND_CASES_BY_ATTR_RANGES

Description: Find cases with the specified search parameters

Explanation: You can use the following in the search criteria: groups (within a group: OR relationship; between different groups: AND relationship), ranges (upper and lower value of an attribute), and operators (**EQ** (equal), **NE** (not equal), **BT** (between), **NB** (not between), **GE** (greater than or equal), **GT** (greater than), **LE** (less than or equal), **LT** (less than), **CP** (contains pattern), **NP** (not pattern)). The search is conducted in the standard table and in the customer table (if specified).

Call sequence: FIND_CASES_BY_ATTR_RANGES

Parameters:

IMPORTING:

IM_ATTR_QUERY	TYPE	SDM_QUERY_DESC_TAB
IM_CUST_TABLE	TYPE	TABLENAME (optional)
IM_MAX_NUM_OF_CASES	TYPE	I (optional)

EXPORTING:

EX_CASE_GUIDS	TYPE	SCMG_TT_CASE_GUID

Exceptions:

NOTHING_FOUND

WRONG_MAX_NUM

WRONG_PROPERTIE

WRONG_OP

Example:

```
CALL METHOD cl_scmg_case_api=>find_cases_by_attr_ranges
     EXPORTING
           IM_ATTR_QUERY       = l_query
           IM_CUST_TABLE       = l_cust_table
           IM_MAX_NUM_OF_CASES = l_max_num
     IMPORTING
           EX_CASE_GUIDS       = l_guids.
```

2.3.18 FIND_CASES_BY_BOR_OBJECT

Description: Find cases with the BOR object

Call sequence: FIND_CASES_BY_BOR_OBJECT

Parameters:

IMPORTING:

IM_BOR_TYPE	TYPE	OJ_NAME
IM_BOR_KEY	TYPE	SWO_TYPEID
IM_SPS_ID	TYPE	STRING
IM_RMS_ID	TYPE	STRING
IM_RELATION	TYPE	STRING (optional)
	DEFAULT	‚CT'

EXPORTING:

EX_T_CASE_GUIDS	TYPE	SCMG_TT_CASE_GUID

Exceptions:

FAILED

Example:

```
CALL METHOD cl_scmg_case_api=>find_cases_by_bor_object
      EXPORTING
            IM_BOR_TYPE     = myBortyp
            IM_BOR_KEY      = myBorkey
            IM_SPS_ID       = mySpsid
            IM_RMS_ID       = myRmsid
      IMPORTING
            EX_T_CASE_GUIDS = l_guids.
```

2.3.19 GET_ALL_OBJECTS

Description: Get objects from the case record

Call sequence: OPEN_CASE -> GET_ALL_OBJECTS …->CLOSE_CASE

Parameters:

RETURNING:

RE_ELEMENTS	TYPE	SRM_REC_ELEM_TAB

Exceptions:

FAILED

Example:

```
l_case = cl_scmg_case_api=>OPEN_CASE( IM_CASE_GUID = l_guid ).

CALL METHOD lcl_case->get_all_objects
      RECEIVING
            RE_ELEMENTS = lt_rec_elems.
```

2.3.20 GET_ALL_SUBCOMPONENTS

Description: Gets all subcomponents of the case and fills G_SUB_INTERFACE

Call sequence: OPEN_CASE -> GET_ALL_SUBCOMPONENTS …->CLOSE_CASE

Parameters:

IMPORTING:

IM_CASE_TYPE	TYPE	STRING

Exceptions:

CX_SRM_SPCL_UNEXPECTED

Example:

```
l_case = cl_scmg_case_api=>OPEN_CASE( IM_CASE_GUID = l_guid ).
l_case -> get_all_subcomponents( im_case_type = 'DEMO' ).
```

2.3.21 GET_BACKEND_CASE

Description: Get the case back end

Parameters:

RETURNING:

RE_CASE_BACKEND	TYPE REF TO	IF_SCMG_CASE

Exceptions:

FAILED

Example:

```
l_case = cl_scmg_case_api=>OPEN_CASE( IM_CASE_GUID = l_guid ).

l_case_backend = l_case->get_case_backend( ).
```

2.3.22 GET_BACKEND_NOTES

Description: Get the notes back end

Parameters:

RETURNING:

RE_NOTES_BACKEND	TYPE REF TO	IF_SCMG_SP_CASE_NOTES_BACKEND

Exceptions:

FAILED

Example:

```
l_case = cl_scmg_case_api=>OPEN_CASE( IM_CASE_GUID = l_guid ).
l_notes = l_case->get_backend_notes( ).
```

2.3.23 GET_BACKEND_RECORD

Description: Get the records back end

Parameters:

RETURNING:

RE_RECORD_BACKEND	TYPE REF TO	IF_SRM_SP_RECORD

Exceptions:

FAILED

Example:

```
l_case = cl_scmg_case_api=>OPEN_CASE( IM_CASE_GUID = l_guid ).
l_record = l_case->get_backend_record( ).
```

2.3.24 GET_BACKEND_WF_PATH

Description: Get the process route API

Parameters:

IMPORTING:

IM_CREATE_IF_NONE	TYPE	XFELD (optional)
	DEFAULT	SPACE

RETURNING:

RE_WF_PATH_API	TYPE REF TO	CL_SRM_WF_PATH

Exceptions:

FAILED

NOT_ENQUEUED

Example:

```
l_case = cl_scmg_case_api=>OPEN_CASE( IM_CASE_GUID = l_guid ).

l_wf_path = l_case->get_backend_wf_path( ).
```

2.3.25 GET_BACKEND_WF_STATE

Description: Get the process route status

Parameters:

RETURNING:

RE_WF_STATE	TYPE	SRMWFPTHST

Example:

```
l_case = cl_scmg_case_api=>OPEN_CASE( IM_CASE_GUID = l_guid ).
l_wf_state = l_case->get_backend_wf_state( ).
```

2.3.26 GET_CASE (obsolete)

Description: Create a case API object

Explanation: For the subsequent modifying operations, call the method with the log flag (IM_ENQUEUE = X); otherwise other case actions will fail.

However, also avoid locking objects unnecessarily, since this generates unnecessary lock objects.

After a GET_CASE, a decision is made about whether the case is updated or not. IM_UPDATE_TASK = IF_SRM=>FALSE No update
IM_UPDATE_TASK = IF_SRM=>TRUE Update

Parameters:

IMPORTING:

IM_CASE_GUID	TYPE	SCMG_CASE_GUID
IM_ENQUEUE	TYPE	XFELD (optional)
	DEFAULT	SPACE
IM_MODE	TYPE	STRING (optional)
	DEFAULT	IF_SRM_SP_ENQUEUE=>MODE_EXCLUSIVE
IM_SCOPE	TYPE	STRING (optional)
	DEFAULT	IF_SRM_SP_ENQUEUE=>SCOPE_DIALOG
IM_UPDATE_TASK	TYPE	SRMBOOLEAN (optional)
	DEFAULT	IF_SRM=>FALSE

RETURNING:

RE_CASE	TYPE REF TO	IF_SCMG_CASE_API

Exceptions:

FAILED

ENQUEUE_FAILED

INVALID_GUID

Example:

```
l_case = cl_scmg_case_api=>get_case( IM_CASE_GUID = l_guid ).
```

2.3.27 GET_CASE_TEXT

Description: Get the terminology that is set in Customizing; by default, the text is "Case".

Call sequence: OPEN_CASE -> GET_CASE_TEXT …->CLOSE_CASE

Parameters:

IMPORTING:

IM_TERMID	TYPE	SCMGTERMID (optional)
	DEFAULT	‚CASE'

RETURNING:

RE_TERMID	TYPE	SCMGTERM

Exceptions:

Example:

```
l_case = cl_scmg_case_api=>OPEN_CASE( IM_CASE_GUID = l_guid ).
l_termid = l_case->get_case_text( ).
```

2.3.28 GET_INSTANCES_FOR_REC_ANCHOR

Description: Get all elements in the case record. If the anchor is specified as a parameter, then only this anchor's elements are found.

Call sequence: OPEN_CASE -> GET_INSTANCES_FOR_REC_ANCHOR …->CLOSE_CASE

Parameters:

IMPORTING:

IM_ANCHOR	TYPE	STRING (optional)

RETURNING:

RE_ELEMENT_TABLE	TYPE	SCMG_TT_INST_IN_RECORD

Exceptions:

FAILED

Example:

```
l_case = cl_scmg_case_api=>OPEN_CASE( IM_CASE_GUID = l_guid ).

CALL METHOD l_case-> get_instances_for_rec_anchor
      EXPORTING
            IM_ANCHOR = 'TEST'
      RECEIVING
            RE_ELEMENT_TABLE = lt_rec_elems.
```

2.3.29 GET_LINKED_OBJECTS

Description: Get objects from the case record

Not yet implemented.

Parameters:

IMPORTING:

IM_BOR_OBJECT_TYPE	TYPE	SWO_OBJTYP (optional)

RETURNING:

RE_OBJECTS	TYPE	TY_NAMEVALUEASSTRING

Exceptions:

FAILED

2.3.30 GET_RECORD_OPEN

Description: Get the internal switch G_RECORD_OPENED: Case record opened

Parameters:

RETURNING:

RE_RECORD_OPEN	TYPE	SRMBOOLEAN

Exceptions:

Example:

```
l_case = cl_scmg_case_api=>OPEN_CASE( IM_CASE_GUID = l_guid ).
l_record_open = l_case->get_record_open( ).
```

2.3.31 GET_VICINITY

Description: Get the supercases and subcases for a case

Parameters:

IMPORTING:

IM_ANCHOR	TYPE	STRING (optional)
	DEFAULT	'CZ'

EXPORTING:

EX_SUPERCASES_GUID_TAB	TYPE	SCMG_TT_CASE_GUID
EX_SUBCASES_GUID_TAB	TYPE	SCMG_TT_CASE_GUID

Exceptions:

Example:

```
l_case = cl_scmg_case_api=>OPEN_CASE( IM_CASE_GUID = l_guid ).
CALL METHOD l_case-> get_vicinity
      EXPORTING
            IM_ANCHOR = 'CZ'
      IMPORTING
            EX_SUPERCASES_GUID_TAB = lt_super_tab.
            EX_SUBCASES_GUID_TAB   = lt_sub_tab.
```

2.3.32 LOG_READ

Description: Get the log

Call sequence: OPEN_CASE -> LOG_READ …->CLOSE_CASE

Parameters:

IMPORTING:

IM_START_DATE	TYPE	D (optional)
IM_END_DATE	TYPE	D (optional)
IM_MAX_ROWS	TYPE	I (optional)

EXPORTING:

EX_ENTRIES	TYPE	SRMPT_PROTO_ENTRY_TAB

Exceptions:

Example:

```
l_case = cl_scmg_case_api=>OPEN_CASE( IM_CASE_GUID = l_guid ).

CALL METHOD l_case->log_read
   IMPORTING
     EX_ENTRIES = l_proto_entries.
```

2.3.33 LOG_WRITE

Description: Write the log

Call sequence: OPEN_CASE -> LOG_WRITE -> SAVE …->CLOSE_CASE

Parameters:

IMPORTING:

IM_ACTIVITY	TYPE	STRING
IM_ARG1	TYPE	STRING
IM_ARG2	TYPE	STRING
IM_ARG_STRING	TYPE	STRING

Exceptions:

FAILED

Example:

```
l_case = cl_scmg_case_api=>OPEN_CASE( IM_CASE_GUID = l_guid ).

CALL METHOD l_case->log_write
     EXPORTING
           im_activity = 'ATTRIBUTE_CHANGE'
           im_arg1 = l_old_value
           im_arg2 = l_value
           im_arg_string = 'CASE_TITLE'.

l_case->save( IM_DEQUEUE = IF_SRM=>TRUE ).
```

2.3.34 NOTES_GET

Description: Get notes

Call sequence: OPEN_CASE -> NOTES_GET …->CLOSE_CASE

Parameters:

IMPORTING:

IM_TEXT_TYPE	TYPE	TDID (optional)
IM_DATE_FROM	TYPE	D (optional)
IM_DATA_TO	TYPE	D (optional)
IM_CREATED_BY	TYPE	SYUNAME (optional)
IM_KEYWORD	TYPE	STRING (optional)
IM_DB_UPDATE	TYPE	XFELD (optional)

RETURNING:

RE_TEXTS	TYPE	TEXT_LH

Exceptions:

FAILED

Example:

```
 l_case = cl_scmg_case_api=>OPEN_CASE( IM_CASE_GUID = l_guid ).

 CALL METHOD lcl_case->notes_get
EXPORTING
             IM_DATE_FROM = '20030301'
             IM_DATE_TO   = '20030806'
       RECEIVING
             RE_TEXTS = l_texte.
```

2.3.35 NOTE_INSERT

Description: Insert a note for a case

Call sequence: OPEN_CASE -> NOTE_INSERT -> SAVE ...->CLOSE_CASE

Parameters:

IMPORTING:

IM_ID	TYPE	TDID
IM_TEXT	TYPE	TLINETAB
IM_LANG	TYPE	TDSPRAS (optional)
	DEFAULT	SY-LANGU

Exceptions:

FAILED

Example:

```
 l_case = cl_scmg_case_api=>OPEN_CASE( IM_CASE_GUID = l_guid ).
 Call l_case->note_insert
      EXPORTING
             IM_ID = '0002'
             IM_TEXT = l_linetab
             IM_LANG = sy-langu.

 l_case->save( IM_DEQUEUE = IF_SRM=>TRUE ).
 l_case->close_case( ).
```

2.3.36 OPEN_CASE

Description: Open a case to be processed

Explanation: For the subsequent modifying operations, call the method with the log flag (IM_ENQUEUE = X); otherwise other case actions will fail.

However, also avoid locking objects unnecessarily, since this generates unnecessary lock objects.

After a OPEN_CASE, a decision is made about whether the case is updated or not. IM_UPDATE_TASK = IF_SRM=>FALSE No update
IM_UPDATE_TASK = IF_SRM=>TRUE Update

To close the transaction, use CLOSE_CASE.

Call sequence: OPEN_CASE -> …-> CLOSE_CASE
Parameters:

IMPORTING:

IM_CASE_GUID	TYPE	SCMG_CASE_GUID
IM_ENQUEUE	TYPE	XFELD (optional)
	DEFAULT	SPACE
IM_MODE	TYPE	STRING (optional)
	DEFAULT	IF_SRM_SP_ENQUEUE=>MODE_EXCLUSIVE
IM_SCOPE	TYPE	STRING (optional)
	DEFAULT	IF_SRM_SP_ENQUEUE=>SCOPE_DIALOG
IM_UPDATE_TASK	TYPE	SRMBOOLEAN (optional)
	DEFAULT	IF_SRM=>FALSE

RETURNING:

RE_CASE	TYPE REF TO	IF_SCMG_CASE_API

Exceptions:

FAILED

ENQUEUE_FAILED

INVALID_GUID

CX_SRM_GSP_BACK

Example:

```
l_case = cl_scmg_case_api=>open_case( IM_CASE_GUID = l_guid ).

l_case->close_case( ).
```

2.3.37 SAVE

Description: Save a case

Explanation: If notes (Notes_Insert) are inserted in the actions (between OPEN_CASE and Save), or if events are triggered when attributes (such as the status) are changed, then a COMMIT_WORK must be performed after Save( ); otherwise the changes or events might be lost.

Call sequence: OPEN_CASE -> … -> SAVE …->CLOSE_CASE

Parameters:

IMPORTING:

IM_DEQUEUE	TYPE	XFELD (optional)
	DEFAULT	SPACE
IM_NEW_VERSION	TYPE	XFELD (optional)
	DEFAULT	SPACE

EXPORTING:

EX_MESSAGES	TYPE	SCMG_T_ATTR_RETURN_VALUE

Exceptions:

FAILED

Example:

```
l_case = cl_scmg_case_api=>OPEN_CASE( IM_CASE_GUID = l_guid ).
...
l_case ->save( IM_NEW_VERSION = 'X ).
l_case->close_case( ).
```

2.3.38 SET_CLIENT_FRAMWORK_ID

Description: Set the client framework ID (if it is not the default framework)

Parameters:

IMPORTING:

IM_CLIENT_FW_ID	TYPE	STRING

EXPORTING:

CHANGING:

RETURNING:

Exceptions:

INSTANCE_EXISTS

FAILED

Example:

```
l_client_fw_id = cl_scmg_case_api=>set_client_framwork_id( ).
```

2.3.39 SET_ROOT_OBJECT

Description: Set the root object (before the first GET, or similar action)

Explanation: You can use this method if you already have a framework, and you want to use this framework for the API methods as well. This method must be called before all other API methods; otherwise an error is triggered.

Parameters:

IMPORTING:

IM_ROOT_OBJECT TYPE REF TO IF_SRM_ROOT

Exceptions:

DIFFERENT_NEW_ROOT_OBJECT

Example:

```
l_root_object = cl_scmg_case_api=>set_root_object( ).
```

2.4 Attributes for IF_SCMG_CASE_API

2.4.1 C_ACTIVITY_FIELD_DISPLAY

Description: Constant *Activity*; field value *Change* (authorization check); initial value *02*

Type: ACTIV_AUTH

2.4.2 C_ACTIVITY_FIELD_MODIFY

Description: Constant *Activity*; field value *Display* (authorization check); initial value *03*

Type: ACTIV_AUTH

2.4.3 G_CASE_GUID

Description: Technical key of the case

Type: SCMG_CASE_GUID

2.4.4 G_CASE_POID

Description: POID of the case

Type: IF_SRM_POID

2.4.5 G_CLIENT_FW_ID

Description: Framework ID – default

Type: STRING

2.4.6 G_DELETED

Description: Case deleted, yes or no (X/space)

Type: XFELD

2.4.7 G_IS_ENQUEUED

Description: Lock set, yes or no (X/space)

Type: XFELD

2.4.8 G_LAST_EXCEPTION

Description: Last exception raised (after FAILED)

Type: CX_ROOT

2.4.9 G_ROOT_OBJECT

Description: Root object

Type: IF_SRM_ROOT

2.4.10 G_SUB_INTERFACE

Description: Table with the interfaces of the subcomponents

Type: SCMG_TT_CASE_SUBCOMPONENTS_API

3 The Subcomponent Programming Model

A subcomponent is a component that is shown in the bottom window of the case screen. (The term *Case Component* is used as a synonym for subcomponent.) On a strip of pushbuttons under the attributes, the user can choose which subcomponent is displayed. SAP delivers four subcomponents: *Linked Objects* (the case record), *Notes*, *Process Route*, and *Log*.

Since Case Management is based on an open architecture, you can implement your own subcomponents, which you can then display within a case.

To do this, develope a class that implements the IF_SCMG_SUBCOMPONENT interface. Then assign the subcomponent to a case type.

3.1 Implementation of IF_SCMG_SUBCOMPONENT

This interface enables the case to communicate with its subcomponents.

The case front end (in the standard system, the CL_SCMG_CASE_VISUALIZATION_WIN class) uses the case type to get the implemented subcomponents from the Customizing settings, and also the corresponding classes that implement the IF_SCMG_SUBCOMPONENT interface. It then calls the following interface methods:

GET_NEW_INSTANCE( ): This is a static method that is called for each case activity. The following are specified: a reference to the front end; the class parameter (see section 3.2); the class name; and a reference to the container in which the subcomponent is visualized. They return a reference to IF_SCMG_SUBCOMPONENT.

CREATE( ): This method is called when the user chooses the *Create Case* activity. Implement this method only if you want to create the subcomponent when you create the case.

DISPLAY( ): This method is called when the user chooses the *Display Case* activity.

MODIFY( ): This method is called when the user chooses the *Change Case* activity. It is used to visualize the subcomponent in change mode.

SET_BUTTON( ): This method is called to determine the icon for the pushbutton of the case component. Implement this method only if you want to overwrite the Customizing settings for the icon dynamically.

SAVE( ): This method is called when the user saves the case. It is used to save the subcomponent.

DELETE( ): This method is called when the user deletes the case. It is used to delete the subcomponent.

PRINT( ): This method is called when the user prints the case. It is used to return the print data of the subcomponent in a hierarchy display.

IS_CHANGED( ): This method is called to determine whether any changes made to the subcomponent need to be saved. In a dialog, the user then decides whether to save the case.

FINISH( ): This method is called when the user exits the case (by switching screens, for example). You might need to release resources. There is no dialog.

ANSWER_ON_REQUEST( ): This method is relevant only for subcomponents that use service providers in the passive in-place visualization. You do not normally need to implement this method.

Note: If you want to access the subcomponent using the case API as well, you must also implement the IF_SCMG_SUBCOMPONENT_API interface. This is documented online in the system.

3.2 Assigning the Subcomponent to a Case Type

1) Perform the IMG activity *Define Functions*. You maintain this activity in the function table SCMGFUNCTION.

   In the *Class for Case Component* field, enter the implementing class for the subcomponent. This class must implement the IF_SCMG_SUBCOMPONENT interface.

   You can enter a parameter in the *Class Parameter* field. This entry is specified by the front end when the objects are generated for the subcomponent (method IF_SCMG_SUBCOMPONENT~GET_NEW_INSTANCE). You use this parameter if you want to use the same class for multiple case components. It gives you information about the subcomponent in which you are currently located.

   In the *Toolbar Type* field, choose the *Subcomponent* entry (T). The toolbar type describes whether the pushbutton is displayed above the attributes and triggers a function, or whether it is displayed below the attributes and calls a subcomponent. Only two values are relevant: F (*Function*) or T (*Subcomponent*).

   Note: From Release 7.0 there is a separate IMG activity *Define Case Components*, in addition to *Define Functions*. Define case components in the *Define Case Components* activity only. The toolbar type is set automatically.

2) In the *Create Function Profile* IMG activity, assign the subcomponent to a function profile.

3) In the *Define Case Types* IMG activity, assign the function profile to a case type. The subcomponent is then available in all cases that are based on this case type.

4 System BAdIs and Customer BAdIs

Case Management provides you with user exits. You can use these user exits to implement functions that are not shipped in the standard system. Business Add-Ins (BAdIs) are used to realize the user exits.

All BAdIs for Case Management are in the SCMG_* namespace. They are documented online in the system.

BAdIs whose names end with _S can only be used internally at SAP. Customers can use those BAdIs whose names end with _C.