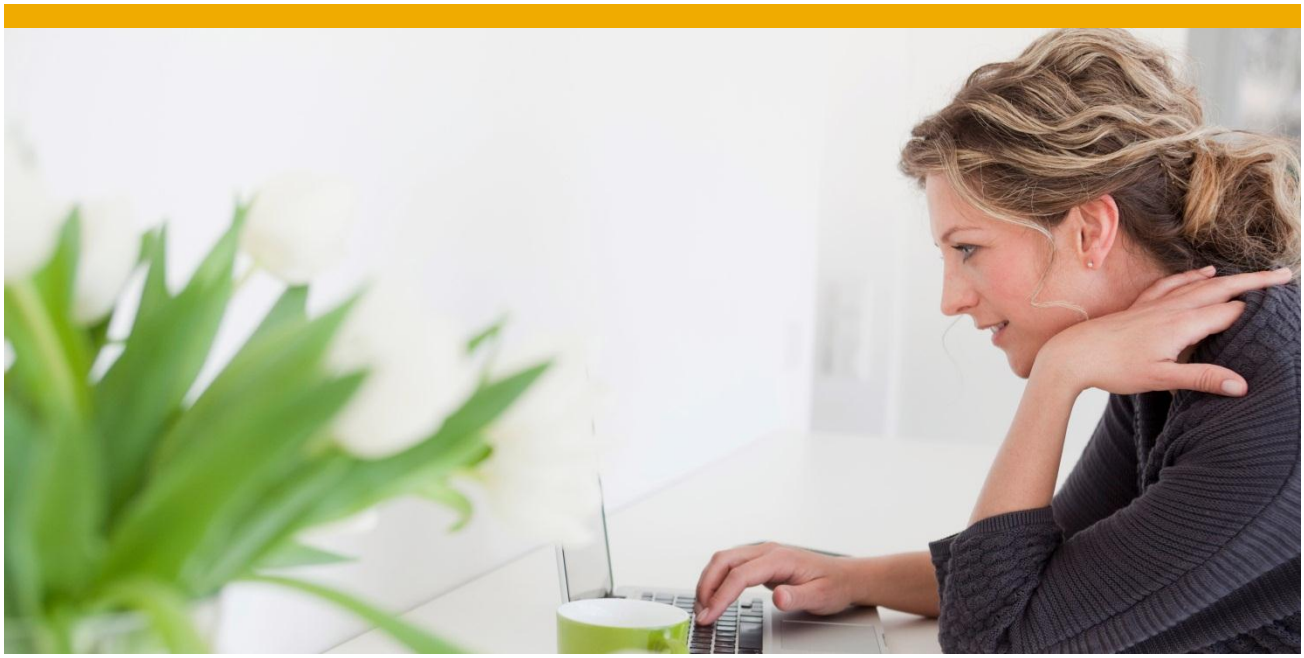

Implementation of Custom Specific Engines for the SAP Application Interface Framework

TABLE OF CONTENTS

EXAMPLE SCENARIO 3
CUSTOM-SPECIFIC APPLICATION ENGINE 7
CUSTOM-SPECIFIC PERSISTENCE ENGINE 10
CUSTOM-SPECIFIC SELECTION ENGINE 14
CUSTOM-SPECIFIC LOGGING ENGINE 20
DEFINE CUSTOM-SPECIFIC ENGINES 22
INTERFACE DEFINITION IN THE SAP APPLICATION INTERFACE FRAMEWORK 22
Create Structure and Define Interface 22
Specify Interface Engines 23
TEST MONITORING AND ERROR HANDLING 24

With the SAP Application Interface Framework several engines are delivered that enable you to monitor interfaces with different interface technologies. However, it is also possible to create your own customer-specific engines. This document will describe the implementation of customer specific engines. Following engine types exist and can be implemented:

- Application Engine
- Persistence Engine
- Selection Engine
- Logging Engine

EXAMPLE SCENARIO

In order to explain the implementation of custom specific engines an exemplary scenario will be used. The scenario will monitor a report that creates flight bookings. The report provides several input parameters where data to create a flight booking can be provided. The input data will be passed to a function module. The function module will call BAPI_FLBOOKING_CREATEFROMDATA. The input data will be stored in a database table. Furthermore, the messages returned from the BAPI will be stored in a separate database table. The status of the booking will be determined based on the status of the messages the BAPI returns. The report to create flight bookings could look as follows:

```
REPORT  zaif_example_create_flbooking.

PARAMETERS: p_carrid TYPE bapisbonew-airlineid,
            p_connid TYPE bapisbonew-connectid,
            p_date   TYPE bapisbonew-flightdate,
            p_custid TYPE bapisbonew-customerid,
            p_class  TYPE bapisbonew-class,
            p_agency TYPE bapisbonew-agencynum.

START-OF-SELECTION.

  DATA: ls_booking     TYPE bapisbonew,
        lt_return      TYPE bapirettab,
        lv_guid32      TYPE guid_32,
        lv_status      TYPE /aif/proc_status,
        lv_counter     TYPE i,
        ls_flbook_head TYPE zaif_flbook_head,
        ls_flbook_msg  TYPE zaif_flbook_msgs,
        lt_flbook_head TYPE STANDARD TABLE OF zaif_flbook_head,
        lt_flbook_msgs TYPE STANDARD TABLE OF zaif_flbook_msgs.
  FIELD-SYMBOLS: <ls_return> TYPE bapiret2.

  ls_booking-airlineid  = p_carrid.
  ls_booking-connectid  = p_connid.
  ls_booking-flightdate = p_date.
  ls_booking-customerid = p_custid.
  ls_booking-class      = p_class.
  ls_booking-agencynum  = p_agency.

  CALL FUNCTION 'GUID_CREATE'
    IMPORTING
      ev_guid_32 = lv_guid32.

  CALL FUNCTION 'Z_AIF_CREATE_FLIGHT_BOOKING'
    EXPORTING
```

```
      is_booking           = ls_booking
      iv_guid32            = lv_guid32
*     IV_CREATE_DATE       =
*     IV_CREATE_TIME       =
          .
```

Function Module Z_AIF_CREATE_FLIGHT_BOOKING

```
  DATA: lt_return      TYPE bapirettab,
        lv_status      TYPE /aif/proc_status,
        lv_counter     TYPE i,
        ls_flbook_head TYPE zaif_flbook_head,
        ls_flbook_msg  TYPE zaif_flbook_msgs,
        lt_flbook_head TYPE STANDARD TABLE OF zaif_flbook_head,
        lt_flbook_msgs TYPE STANDARD TABLE OF zaif_flbook_msgs.
  FIELD-SYMBOLS: <ls_return> TYPE bapiret2.

  CALL FUNCTION 'BAPI_FLBOOKING_CREATEFROMDATA'
    EXPORTING
      booking_data  = is_booking
    IMPORTING
      bookingnumber = ls_flbook_head-bookid
    TABLES
      return        = lt_return.

  READ TABLE lt_return TRANSPORTING NO FIELDS WITH KEY type = 'E'.
  IF sy-subrc = 0.
    lv_status = 'E'.
  ELSE.
    READ TABLE lt_return TRANSPORTING NO FIELDS WITH KEY type = 'A'.
    IF sy-subrc = 0.
      lv_status = 'A'.
    ELSE.
      lv_status = 'S'.
    ENDIF.
  ENDIF.

  ls_flbook_head-guid_32  = iv_guid32.
  ls_flbook_head-status   = lv_status.
  ls_flbook_head-carrid   = is_booking-airlineid.
  ls_flbook_head-connid   = is_booking-connectid .
  ls_flbook_head-fldate   = is_booking-flightdate.
  ls_flbook_head-custid   = is_booking-customerid.
  ls_flbook_head-class    = is_booking-class.
  ls_flbook_head-agencyid = is_booking-agencynum .

  IF iv_create_date IS SUPPLIED AND iv_create_date IS NOT INITIAL.
    ls_flbook_head-create_date = iv_create_date.
  ELSE.
    ls_flbook_head-create_date = sy-datum.
  ENDIF.
  IF iv_create_time IS SUPPLIED AND iv_create_time IS NOT INITIAL.
```

```
    ls_flbook_head-create_time = iv_create_time.
  ELSE.
    ls_flbook_head-create_time = sy-uzeit.
  ENDIF.

  CLEAR lv_counter.
  LOOP AT lt_return ASSIGNING <ls_return>.
    ls_flbook_msg-guid_32 = iv_guid32.
    ADD 1 TO lv_counter.
    ls_flbook_msg-counter         = lv_counter.
    ls_flbook_msg-type            = <ls_return>-type.
    ls_flbook_msg-id              = <ls_return>-id.
    ls_flbook_msg-msg_number      = <ls_return>-number.
    ls_flbook_msg-message         = <ls_return>-message.
    ls_flbook_msg-log_no          = <ls_return>-log_no.
    ls_flbook_msg-log_msg_no      = <ls_return>-log_msg_no.
    ls_flbook_msg-message_v1      = <ls_return>-message_v1.
    ls_flbook_msg-message_v2      = <ls_return>-message_v2.
    ls_flbook_msg-message_v3      = <ls_return>-message_v3.
    ls_flbook_msg-message_v4      = <ls_return>-message_v4.
    ls_flbook_msg-parameter_name  = <ls_return>-parameter.
    ls_flbook_msg-row_in_para     = <ls_return>-row.
    ls_flbook_msg-field_in_para   = <ls_return>-field.
    ls_flbook_msg-log_system      = <ls_return>-system.

    APPEND ls_flbook_msg TO lt_flbook_msgs.
    CLEAR ls_flbook_msg.
  ENDLOOP.

  MODIFY zaif_flbook_head FROM ls_flbook_head.
  MODIFY zaif_flbook_msgs FROM TABLE lt_flbook_msgs.

  COMMIT WORK.

ENDFUNCTION.
```

The function module will fill two database tables. The first database table ZAIF_FLBOOK_HEAD will store the data from the selection screen. Furthermore, the database table will store the status, if the booking was created successfully or if errors occurred.

Database Table ZAIF_FLBOOK_HEAD

Component	Key	Value
MANDT	X	MANDT
GUID_32	X	GUID_32
STATUS		/AIF/PROC_STATUS
CARRID		S_CARRID

CONNID		S_CONN_ID
FLDATE		S_DATE
CUSTID		S_CUSTOMER
CLASS		S_CLASS
AGENCYID		S_AGNCYNUM
BOOKID		S_BOOK_ID
CREATE_DATE		/AIF/CREATE_DATE
CREATE_TIME		/AIF/CREATE_TIME
IS_LOCKED		CHAR1

The second database table will store the messages returned from the BAPI.

Database Table ZAIF_FLBOOK_MSGS

Component	Key	Value
MANDT	X	MANDT
GUID_32	X	GUID_32
COUNTER	X	INT4
TYPE		BAPI_MTYPE
ID		SYMSGID
MSG_NUMBER		SYMSGNO
MESSAGE		BAPI_MSG
LOG_NO		BALLOGNR
LOG_MSG_NO		BALMNR
MESSAGE_V1		SYSMGV
MESSAGE_V2		SYSMGV
MESSAGE_V3		SYSMGV
MESSAGE_V4		SYSMGV
PARAMETER_NAME		BAPI_PARAM
ROW_IN_PARA		BAPI_LINE
FIELD_IN_PARA		BAPI_FLD
LOG_SYSTEM		BAPILOGSYS

Running the report will generate flight bookings, if the input data is correct. Furthermore, entries will be created in the database tables.

In order to monitor the execution of the report with the SAP Application Interface Framework custom-specific engines can be developed.

CUSTOM-SPECIFIC APPLICATION ENGINE

The first engine that needs to be implemented with customer specific code is the application engine. The application engine is the connection point between *Monitoring and Error Handling* and the other engines. Furthermore, the application engine will handle requests that are application specific, for example restarting and canceling.

In order to implement a custom-specific application engine several possibilities exist:

- Implement interface /AIF/IF_APPLICATION_ENGINE
- Inherit from class /AIF/CL_APPL_ENGINE_BASE
- Inherit from one of the concrete engines delivered with SAP Application Interface Framework

Which of the options fits best for your needs depends on your concrete scenario.

Following methods should be redefined:

- RESTART: Triggers the reprocessing of the selected data messages
- CANCEL: Cancels the selected data messages. Depending on your scenario you might have to set the status in the index tables of the SAP Application Interfaces Framework.
- CHECK_MSG_EDITABLE: Checks if the message has a status, that allows editing
- START_EXTERNAL_MONITOR: This will start the monitor that is usually used to monitor messages of the selected interface technology. For example, in cases of proxy messages the report *Display XML Message Versions* (RSXMB_DISPLAY_MSG_VERS_NEW) is called.

Example Implementation:

In the following the implementation of an application engine for the scenario outlined before will be shown.

Create a new class that inherits from /AIF/CL_APPL_ENGINE_BASE.

Implementation of method RESTART

```
METHOD /aif/if_application_engine~restart.
  DATA: lv_answer          TYPE char1,
        lv_guid32          TYPE guid_32,
        ls_aif_flbook_head TYPE zaif_flbook_head,
        ls_booking         TYPE bapisbonew,
        lv_seq_number      TYPE symsgv,
        ls_return          TYPE bapiret2.

  FIELD-SYMBOLS: <ls_treenode> TYPE /aif/treenode_st,
                 <ls_data_row> TYPE /aif/data_row_v_1_st,
                 <lv_guid32>   TYPE guid_32.

  CALL FUNCTION 'POPUP_TO_CONFIRM'
    EXPORTING
      titlebar       = 'Restart Messages '
      text_question  = 'Restat selected message(s)?'
      text_button_1  = 'Yes'(001)
      text_button_2  = 'No'(002)
    IMPORTING
      answer         = lv_answer
    EXCEPTIONS
```

```
      text_not_found = 1
      OTHERS         = 2.
  IF sy-subrc <> 0.
* Implement suitable error handling here
  ENDIF.
  IF lv_answer = 1.
    LOOP AT is_msg_v1-tree_node ASSIGNING <ls_treenode>.
      ASSIGN <ls_treenode>-data_row->* TO <ls_data_row>.
      CHECK <ls_data_row> IS ASSIGNED.
      ASSIGN COMPONENT 'MSGGUID' OF STRUCTURE <ls_data_row> TO <lv_guid32>.
      CHECK <lv_guid32> IS ASSIGNED AND <lv_guid32> IS NOT INITIAL.
      lv_guid32 = <lv_guid32>.
*       get entry with up-to-date data for reprocessing
      SELECT SINGLE * FROM zaif_flbook_head
        INTO ls_aif_flbook_head WHERE guid_32 = lv_guid32.
*    Delete old return messages
      DELETE FROM zaif_flbook_msgs WHERE guid_32 = lv_guid32.
      CALL FUNCTION 'DB_COMMIT'.

      ls_booking-airlineid = ls_aif_flbook_head-carrid.
      ls_booking-connectid = ls_aif_flbook_head-connid.
      ls_booking-flightdate = ls_aif_flbook_head-fldate.
      ls_booking-customerid = ls_aif_flbook_head-custid.
      ls_booking-class = ls_aif_flbook_head-class.
      ls_booking-agencynum = ls_aif_flbook_head-agencyid.

*create flight booking
      CALL FUNCTION 'Z_AIF_CREATE_FLIGHT_BOOKING'
        EXPORTING
          is_booking     = ls_booking
          iv_guid32      = lv_guid32
          iv_create_date = ls_aif_flbook_head-create_date
          iv_create_time = ls_aif_flbook_head-create_time.

      lv_seq_number = <ls_treenode>-seq_number.
      CONDENSE lv_seq_number.
      ls_return-type = 'S'.
      ls_return-id = '/AIF/MES'.
      ls_return-number = '108'.
      ls_return-message_v1 = lv_seq_number.
      APPEND ls_return TO et_return.
      INSERT <ls_treenode> INTO TABLE ct_treenodes.
    ENDLOOP.
  ELSE.
    RAISE EXCEPTION TYPE /aif/cx_error_handling_general.
  ENDIF.

ENDMETHOD.
```

Implementation of method CANCEL

```
METHOD /aif/if_application_engine~cancel.
  DATA: lv_guid32 TYPE guid_32.
  FIELD-SYMBOLS: <ls_treenode> TYPE /aif/treenode_st,
                 <ls_data_row> TYPE /aif/data_row_v_1_st,
                 <lv_guid32>   TYPE guid_32.

  LOOP AT is_msg_v1-tree_node ASSIGNING <ls_treenode>.
```

```
    ASSIGN <ls_treenode>-data_row->* TO <ls_data_row>.
    CHECK <ls_data_row> IS ASSIGNED.
    ASSIGN COMPONENT 'MSGGUID' OF STRUCTURE <ls_data_row> TO <lv_guid32>.
    CHECK <lv_guid32> IS ASSIGNED AND <lv_guid32> IS NOT INITIAL.
    lv_guid32 = <lv_guid32>.
*Set the status of the data record to canceled
    UPDATE zaif_flbook_head SET status = 'C' WHERE guid_32 = lv_guid32.
    CALL FUNCTION 'DB_COMMIT'.
    CLEAR <ls_data_row>-lognumber.
    INSERT <ls_treenode> INTO TABLE ct_treenodes.
  ENDLOOP.

ENDMETHOD.
```

Implementation of method CHECK_MSG_EDITABLE

```
METHOD /aif/if_application_engine~check_msg_editable.
  FIELD-SYMBOLS: <lv_status>   TYPE /aif/proc_status,
                 <lr_data_row> TYPE REF TO data.

  ASSIGN COMPONENT 'DATA_ROW' OF STRUCTURE is_idx_data TO <lr_data_row>.
  ASSIGN <lr_data_row>->('STATUS') TO <lv_status>.


*Only messages that are in an error state should be edited
  IF <lv_status> = 'E' OR <lv_status> = 'A'.
    rv_is_editable = 'X'.
  ELSE.
    rv_is_editable = ''.
  ENDIF.
ENDMETHOD.
```

Implementation of method START_EXTERNAL_MONITOR

Since this example implementation monitors a report no standard monitor exists. Therefore, the method will only raise an exception and the user will be informed that there is no standard monitor.

```
METHOD /aif/if_application_engine~start_external_monitor.
  DATA: ls_textid TYPE scx_t100key.
  FIELD-SYMBOLS: <ls_aifkeys> TYPE /aif/ifkeys.

  ASSIGN ir_row_data->('AIFKEYS') TO <ls_aifkeys>.

*ZAIF_TEST_X102(006): A Standard Monitor does not exist for interface &1/&2/&3
  ls_textid-msgno = '006'.
  ls_textid-msgid = 'ZAIF_TEST_X102'.
  ls_textid-attr1 = <ls_aifkeys>-ns.
  ls_textid-attr2 = <ls_aifkeys>-ifname.
  ls_textid-attr3 = <ls_aifkeys>-ifver.
  RAISE EXCEPTION TYPE /aif/cx_error_handling_general
    EXPORTING
      textid = ls_textid.
ENDMETHOD.
```

CUSTOM-SPECIFIC PERSISTENCE ENGINE

The persistence engine handles the data content of a message. The persistence engine retrieves the data displayed in Data Content view (lower ride side of *Monitoring and Error Handling* screen) from the corresponding persistence layer. Furthermore, it provides functionality to update the data content after editing in Data Content view.

In order to implement a custom-specific selection engine several possibilities exist:

- Implement interface /AIF/IF_PERSISTENCY_ENGINE
- Create a class that inherits from /AIF/CL_PERSIST_ENGINE_BASE
- Create a class that inherits from a concrete persistence engine class

Which of the option fits best for your needs depends on your concrete scenario.

Following methods should be redefined:

- RETRIEVE: The method retrieves the data from the persistence layer and prepares the data for displaying in Data Content view. The method should move the data to the format of the source structure. A reference of the data in the source structure format should be passed to changing parameter CREF_DATA and to component XI_DATA of changing parameter CS_XML_PARSE.
- UPDATE: This method updates the data content on the persistence layer after the data was edited
- LOCK: Locks the selected message(s), in case a user edits the data content. This is required to avoid inconsistencies. Others should not be able to edit, cancel or restart a data message while someone locks it for editing
- UNLOCK: After a user has finished editing data content the data message can be unlocked again.
- DETERMINE_STATUS_ICON: Determines the status icons that should be displayed in Data Message view

Example Implementation

In the following the implementation of a persistence engine for the scenario described before will be shown.

Create a new class that inherits from /AIF/CL_PERSIST_ENGINE_BASE

Implementation of method RETRIEVE

```
METHOD /aif/if_persistency_engine~retrieve.
  DATA: lv_guid32           TYPE guid_32,
        ls_aif_flbook_head  TYPE zaif_flbook_head,
        lr_finf             TYPE REF TO /aif/cl_db_access_finf,
        ls_finf             TYPE /aif/t_finf,
        lr_flbook          TYPE REF TO zaif_sap_example_flightbooking,
        ls_flight_data      TYPE sflight,
        ls_customer_data    TYPE scustom,
        ls_agency           TYPE stravelag,
        ls_booking          TYPE sbook.

  FIELD-SYMBOLS: <ls_flbook>      TYPE zaif_sap_example_flightbooking,
                 <ls_flight_booking> TYPE zaif_flight_booking.

  lv_guid32 = iv_msgguid.
  SELECT SINGLE * FROM zaif_flbook_head INTO ls_aif_flbook_head
    WHERE guid_32 = lv_guid32.
  IF sy-subrc = 0.

    lr_finf = /aif/cl_db_access_finf=>get_instance( ).
```

```
    ls_finf = lr_finf->read_single( iv_ns = iv_ns
                                    iv_ifname = iv_ifname
                                    iv_ifver = iv_ifver ).

    CREATE DATA lr_flbook.
    ASSIGN lr_flbook->* TO <ls_flbook>.
    ASSIGN COMPONENT 'FLIGHT_BOOKING' OF STRUCTURE <ls_flbook> TO <ls_flight_boo
king>.

    <ls_flight_booking>-carrid = ls_aif_flbook_head-carrid.
    <ls_flight_booking>-connid = ls_aif_flbook_head-connid.
    <ls_flight_booking>-fldate = ls_aif_flbook_head-fldate.
    <ls_flight_booking>-customer_id = ls_aif_flbook_head-custid.
    <ls_flight_booking>-class = ls_aif_flbook_head-class.
    <ls_flight_booking>-agencyid = ls_aif_flbook_head-agencyid.

*    Select flight data
    SELECT SINGLE * FROM sflight
      INTO ls_flight_data
      WHERE carrid = ls_aif_flbook_head-carrid
       AND connid = ls_aif_flbook_head-connid
       AND fldate = ls_aif_flbook_head-fldate.
    <ls_flight_booking>-flight_data = ls_flight_data.

*    Select customer data
    SELECT SINGLE * FROM scustom
      INTO ls_customer_data
      WHERE id = ls_aif_flbook_head-custid.
    <ls_flight_booking>-customer_data = ls_customer_data.

*    Select travel agency data
    SELECT SINGLE * FROM stravelag
      INTO ls_agency
      WHERE agencynum = ls_aif_flbook_head-agencyid.
    <ls_flight_booking>-travel_agency = ls_agency.

*in case the flight booking was created successfully,
*the booking data should be selected and displayed
    IF NOT ls_aif_flbook_head-bookid IS INITIAL.
      SELECT SINGLE * FROM sbook
        INTO ls_booking
        WHERE bookid = ls_aif_flbook_head-bookid.

      <ls_flight_booking>-booking_data = ls_booking.
    ENDIF.

    cs_xmlparse-msgguid = iv_msgguid.
    cs_xmlparse-ns = iv_ns.
    cs_xmlparse-ifname = iv_ifname.
    cs_xmlparse-ifver = iv_ifver.

    cref_data = cs_xmlparse-xi_data = lr_flbook.
  ENDIF.
ENDMETHOD.
```

Implementation of method UPDATE

```
METHOD /aif/if_persistency_engine~update.
  DATA: ls_changed      TYPE /aif/cdata_s,
        ls_flbook_head  TYPE zaif_flbook_head,
        ls_textid       TYPE SCX_T100KEY.
  FIELD-SYMBOLS: <ls_xmlparse>       TYPE /aif/xmlparse_data,
                 <ls_flight_booking> TYPE any,
                 <lv_update_field>   TYPE any,
                 <lv_changed_field>  TYPE any,
                 <ls_changed>        TYPE any.

  ASSIGN cr_xmlparse->* TO <ls_xmlparse>.

  SELECT SINGLE * FROM zaif_flbook_head
    INTO ls_flbook_head
    WHERE guid_32 = <ls_xmlparse>-msgguid.

  ASSIGN <ls_xmlparse>-xi_data->('FLIGHT_BOOKING') TO <ls_flight_booking>.

  LOOP AT <ls_xmlparse>-list_of_changed_fields INTO ls_changed.

    ASSIGN COMPONENT ls_changed-field_path
OF STRUCTURE <ls_flight_booking> TO <lv_changed_field>.
    IF sy-subrc = 0.
      IF ls_changed-field_path = 'CUSTOMER_ID'.
        ASSIGN COMPONENT 'CUSTID' OF STRUCTURE ls_flbook_head TO <lv_update_fiel
d>.
        <lv_update_field> = <lv_changed_field>.
        UNASSIGN: <lv_update_field>, <lv_changed_field>.
      ELSE.
        ASSIGN COMPONENT ls_changed-
field_path OF STRUCTURE ls_flbook_head TO <lv_update_field>.
        IF sy-subrc = 0.
          <lv_update_field> = <lv_changed_field>.
          UNASSIGN: <lv_update_field>, <lv_changed_field>.
        ELSE.
          ls_textid-msgid = '/AIF/MES'.
          ls_textid-msgno = '000'.
          ls_textid-attr1 = 'Content of field '.
          ls_textid-attr2 = ls_changed-field_path.
          ls_textid-attr3 = ' cannot be changed'.
          RAISE EXCEPTION TYPE /aif/cx_error_handling_general EXPORTING textid =
 ls_textid.
        ENDIF.
      ENDIF.
    ENDIF.
  ENDLOOP.
  UPDATE zaif_flbook_head FROM ls_flbook_head.

ENDMETHOD.
```

Implementation of method LOCK

For reasons of simplicity only a very simple locking mechanism will be implemented for this scenario: Database table ZAIF_FLBOOK_HEAD contains a flag that will be set if someone starts to edit data in *Monitoring and Error Handling*. After the data is updated on the persistence this flag will be cleared again.

```
METHOD /aif/if_persistency_engine~lock.
  DATA: lv_is_locked TYPE char1,
        lv_guid32    TYPE guid_32.
  FIELD-SYMBOLS: <ls_xmlparse> TYPE /aif/xmlparse_data.

  ASSIGN cr_xmlparse->* TO <ls_xmlparse>.
  lv_guid32 = <ls_xmlparse>-msgguid.
  IF <ls_xmlparse>-is_locked = 'X'.
    ev_can_be_updated = 'X'.
  ELSE.
    SELECT SINGLE is_locked FROM zaif_flbook_head
      INTO lv_is_locked
      WHERE guid_32 = lv_guid32.

    IF lv_is_locked = ''.
      UPDATE zaif_flbook_head SET is_locked = 'X' WHERE guid_32 = lv_guid32.
      <ls_xmlparse>-is_locked = 'X'.
      ev_can_be_updated = 'X'.
    ELSE.
      RAISE EXCEPTION TYPE /aif/cx_error_handling_general.
    ENDIF.
  ENDIF.
ENDMETHOD.
```

Implementation of method UNLOCK

```
METHOD /aif/if_persistency_engine~unlock.
  DATA: lv_guid32 TYPE guid_32.
  FIELD-SYMBOLS: <ls_xmlparse> TYPE /aif/xmlparse_data.

  ASSIGN cr_xmlparse->* TO <ls_xmlparse>.

  lv_guid32 = <ls_xmlparse>-msgguid.

  UPDATE zaif_flbook_head SET is_locked = '' WHERE guid_32 = lv_guid32.
  CLEAR <ls_xmlparse>-is_locked.
ENDMETHOD.
```

Implementation of method DETERMINE_STATUS_ICON

```
METHOD /aif/if_persistency_engine~determine_status_icon.
  FIELD-SYMBOLS: <lv_status> TYPE /aif/proc_status.

  ASSIGN COMPONENT /aif/if_globals=>gc_eh_fix_key_flds-
status OF STRUCTURE is_idx_data TO <lv_status>.
  ev_icon_collapsed = ev_icon_expanded = /aif/cl_aif_global_tools=>determine_aif
_file_stat_icon( <lv_status> ).
ENDMETHOD.
```

CUSTOM-SPECIFIC SELECTION ENGINE

The selection engine is responsible for selecting the data displayed in Data Messages view (upper left side) in *Monitoring and Error Handling*.

In order to implement a custom-specific selection engine several possibilities exist:

- Implement interface /AIF/IF_SELECTION_ENGINE
- Inherit from class /AIF/CL_SELECTION_ENGINE_BASE
- Inherit from one of the concrete engines delivered with SAP Application Interface Framework

Which of the options fits best for your needs depends on your concrete scenario.

Following methods should be redefined:

- COMPOSE_WHERE_CONDITION: This method is used to compose the where condition for selecting the messages that should be displayed in *Monitoring and Error Handling*. The where condition depends on the data inserted into the selection screen. Following fields can be evaluated:
  - Message class and message number
  - Data entered into generic selection subscreen
  - Status selection
  - Data entered into an interface-specific subscreen
- SELECT_IDX_TAB_ENTRIES: Selects the data displayed in Data Messages view depending on the data entered in the selection screen
- SELECT_SINGLE_IDX_TAB_ENTRY: Selects the current index table entry from the database

Example Implementation:

In the following the implementation of a selection engine for the scenario described before will be shown.

Create a new class that inherits from /AIF/CL_SELECTION_ENGINE_BASE.

Implementation of method COMPOSE_WHERE_CONDITION

The where-condition created in this example will only evaluate the message GUID, the create date, create time and the status selection

```
METHOD /aif/if_selection_engine~compose_where_condition.
  DATA: lt_parameters       TYPE /aif/sel_para_val_list_t,
        lr_range_tab        TYPE REF TO data,
        ls_aif_keys         TYPE /aif/ifkeys,
        ls_evaluation       TYPE /aif/evaluation_result,
        lt_evaluation       TYPE /aif/evaluation_result_tt,
        ls_result           TYPE /aif/weight_sel_params,
        lv_where            TYPE string,
        lv_exist            TYPE char1.
  FIELD-SYMBOLS: <ls_parameters> TYPE /aif/sel_para_val_list_s,
                 <lt_range_tab>  TYPE ANY TABLE.

  lt_parameters = is_key_tbl-kflds_for_weighting.
  LOOP AT lt_parameters ASSIGNING <ls_parameters>
    WHERE sel_parameter = 'P_STS'
       OR sel_parameter = 'S_GUID'
       OR sel_parameter = 'S_MTIME'
       OR sel_parameter = 'S_MDATE'.
    CLEAR: lv_where, ls_evaluation, lv_exist.
```

```
    create_range_table( EXPORTING ir_sel_para   = ir_sel_para
                                  is_parameters = <ls_parameters>
                        IMPORTING er_range_tab  = lr_range_tab ).

    MOVE-CORRESPONDING is_key_tbl TO ls_aif_keys.
    MOVE-CORRESPONDING ls_aif_keys TO ls_evaluation.
    MOVE-CORRESPONDING <ls_parameters> TO ls_evaluation.

    ls_evaluation-fieldval_ref = lr_range_tab.
    IF lr_range_tab IS NOT INITIAL.
      ASSIGN lr_range_tab->* TO <lt_range_tab>.
      IF ls_evaluation-is_table = 'X' AND <lt_range_tab> IS NOT INITIAL..
        CONCATENATE 'IN <LS_SEL_PARA>-' <ls_parameters>-
sel_parameter INTO lv_where.
        lv_exist = 'X'.
      ELSEIF <lt_range_tab> IS NOT INITIAL.
        CONCATENATE 'EQ <LS_SEL_PARA>-' <ls_parameters>-
sel_parameter INTO lv_where.
        lv_exist = 'X'.
      ENDIF.
      IF lv_exist = 'X'.
        CONDENSE lv_where.
        IF <ls_parameters>-fieldname = 'MSGGUID'.
          CONCATENATE 'GUID_32' lv_where
            INTO  ls_evaluation-where_cond SEPARATED BY space.
        ELSE.
          CONCATENATE <ls_parameters>-fieldname lv_where
            INTO  ls_evaluation-where_cond SEPARATED BY space.
        ENDIF.
      ENDIF.
      APPEND ls_evaluation TO lt_evaluation.
    ENDIF.
  ENDLOOP.

  CLEAR lv_where.
  LOOP AT lt_evaluation INTO ls_evaluation WHERE where_cond IS NOT INITIAL.
    IF lv_where IS INITIAL.
      lv_where = ls_evaluation-where_cond.
    ELSE.
      CONCATENATE ls_evaluation-
where_cond 'AND' lv_where INTO lv_where SEPARATED BY space.
    ENDIF.
  ENDLOOP.
  ls_result-where_cond = lv_where.
  ls_result-sel_tabname = 'ZAIF_FLBOOK_HEAD'.
  APPEND ls_result TO rt_wresult.
ENDMETHOD.
```

Method COMPOSE_WHERE_CONDITION calls a method CREATE_RANGE_TABLE. The implementation of the method could look as follows:

```
METHOD create_range_table.
  DATA: lt_comp       TYPE cl_abap_structdescr=>component_table,
        ls_comp       TYPE abap_componentdescr,
        lr_data       TYPE REF TO data,
        lr_linetype   TYPE REF TO cl_abap_structdescr,
        lr_tabletype  TYPE REF TO cl_abap_tabledescr,
```

```
        lr_type       TYPE REF TO cl_abap_typedescr,
        lv_value      TYPE char100,
        ls_params     TYPE /aif/sel_para_val_list_s.

  FIELD-SYMBOLS: <lv_field>      TYPE any,
                 <lt_values>     TYPE INDEX TABLE,
                 <lt_fieldvalue> TYPE ANY TABLE,
                 <lv_fieldvalue> TYPE any,
                 <ls_value>      TYPE any,
                 <ls_sel_para>   TYPE any.

  ASSIGN ir_sel_para->* TO <ls_sel_para>.
  ls_params = is_parameters.

  ls_comp-name  = 'SIGN'.
  ls_comp-type ?= cl_abap_typedescr=>describe_by_name( 'CHAR1' ).
  APPEND ls_comp TO lt_comp.

  ls_comp-name  = 'OPTION'.
  ls_comp-type ?= cl_abap_typedescr=>describe_by_name( 'CHAR2' ).
  APPEND ls_comp TO lt_comp.

  ls_comp-name  = 'LOW'.
  CALL METHOD cl_abap_typedescr=>describe_by_name
    EXPORTING
      p_name         = ls_params-ref_name
    RECEIVING
      p_descr_ref    = lr_type
    EXCEPTIONS
      type_not_found = 1
      OTHERS         = 2.
  IF sy-subrc <> 0.
    EXIT.
  ENDIF.
  ls_comp-type ?= lr_type.
  APPEND ls_comp TO lt_comp.

  ls_comp-name  = 'HIGH'.
  CALL METHOD cl_abap_typedescr=>describe_by_name
    EXPORTING
      p_name         = ls_params-ref_name
    RECEIVING
      p_descr_ref    = lr_type
    EXCEPTIONS
      type_not_found = 1
      OTHERS         = 2.
  IF sy-subrc <> 0.
    EXIT.
  ENDIF.
  ls_comp-type ?= lr_type.
  APPEND ls_comp TO lt_comp.

  lr_linetype = cl_abap_structdescr=>create( lt_comp ).
  lr_tabletype = cl_abap_tabledescr=>create(  p_line_type = lr_linetype p_table_
kind = cl_abap_tabledescr=>tablekind_std ).
  CREATE DATA lr_data TYPE HANDLE lr_tabletype.

  ASSIGN lr_data->* TO <lt_values>.
```

```
  IF ls_params-is_table = 'X'.
    ASSIGN COMPONENT ls_params-
sel_parameter OF STRUCTURE <ls_sel_para> TO <lt_fieldvalue>.
    IF sy-subrc = 0.
      <lt_values> = <lt_fieldvalue>.
    ELSE.
      CLEAR er_range_tab.
      EXIT.
    ENDIF.
  ELSE.
    ASSIGN COMPONENT ls_params-
sel_parameter OF STRUCTURE <ls_sel_para> TO <lv_fieldvalue>.
    IF sy-subrc = 0.
      IF  <lv_fieldvalue> IS INITIAL.
        CLEAR er_range_tab.
        EXIT.
      ENDIF.
      INSERT  INITIAL LINE INTO TABLE <lt_values> ASSIGNING <ls_value>.
      ASSIGN ('<ls_value>-SIGN') TO <lv_field>.
      <lv_field> = 'I'.
      ASSIGN ('<ls_value>-OPTION') TO <lv_field>.
      lv_value = <lv_field>.
      FIND '*' IN lv_value.
      IF sy-subrc = 0.
        <lv_field> = 'CP'.
      ELSE.
        <lv_field> = 'EQ'.
      ENDIF.
      ASSIGN ('<ls_value>-LOW') TO <lv_field>.
      <lv_field> = <lv_fieldvalue>.
      APPEND <ls_value> TO <lt_values>.
    ENDIF.
  ENDIF.
  er_range_tab = lr_data.
ENDMETHOD.
```

Implementation of method SELECT_IDX_TAB_ENTRIES

The method will select data from database table ZAIF_FLBOOK_HEAD. The data will be prepared for displaying in Data Message view

```
METHOD /aif/if_selection_engine~select_idx_tab_entries.
  DATA: lt_aif_flbook_head TYPE STANDARD TABLE OF zaif_flbook_head,
        ls_aif_keys        TYPE /aif/ifkeys,
        ls_weight_result   TYPE /aif/weight_sel_params,
        lv_where_condition TYPE string,
        ls_idx_data        TYPE /aif/msg_data_st,
        ls_for_sort        TYPE /aif/key_fld_sor_v_1_st,
        lv_lines           TYPE i,
        lv_index           TYPE i,
        lr_idx_tab         TYPE REF TO data,
        lr_row_data        TYPE REF TO data,
        lr_exception       TYPE REF TO /aif/cx_error_handling_general,
        ls_return          TYPE bapiret2.

  FIELD-SYMBOLS: <ls_aif_flbook_head> TYPE zaif_flbook_head,
                 <lt_idx_tab>         TYPE STANDARD TABLE,
                 <ls_idx_tab>         TYPE any,
```

```
                  <lv_msgguid>          TYPE guid_32,
                  <ls_aif_keys>         TYPE /aif/ifkeys,
                  <ls_msg_data_tbl>     TYPE /aif/msg_data_st,
                  <lt_data>             TYPE INDEX TABLE,
                  <ls_data>             TYPE any,
                  <lv_status>           TYPE /aif/proc_status,
                  <ls_add_info1>        TYPE /aif/additional_info_st2,
                  <ls_add_info2>        TYPE /aif/additional_info_st2,
                  <ls_aif_keys2>        TYPE /aif/ifkeys,
                  <ls_admin>            TYPE /aif/admin,
                  <ls_row_data>         TYPE any,
                  <ls_sel_para>         TYPE any.

  MOVE-CORRESPONDING cs_keys_mapping TO ls_aif_keys.

  ASSIGN ir_sel_para->* TO <ls_sel_para>.
  READ TABLE it_weight_result INTO ls_weight_result INDEX 1.
  lv_where_condition = ls_weight_result-where_cond.

*select the data to be displayed in Data Messages view
  SELECT * FROM zaif_flbook_head INTO TABLE lt_aif_flbook_head
  WHERE (lv_where_condition)
    ORDER BY PRIMARY KEY.

*create a temporary index table for the data selected from zaif_flbook_head
  CREATE DATA lr_idx_tab TYPE STANDARD TABLE OF ('/AIF/STD_IDX_TBL').
  ASSIGN lr_idx_tab->* TO <lt_idx_tab>.
  LOOP AT lt_aif_flbook_head ASSIGNING <ls_aif_flbook_head>.
    APPEND INITIAL LINE TO <lt_idx_tab> ASSIGNING <ls_idx_tab>.
    ASSIGN COMPONENT 'MSGGUID' OF STRUCTURE <ls_idx_tab> TO <lv_msgguid>.
    <lv_msgguid> = <ls_aif_flbook_head>-guid_32.
    ASSIGN COMPONENT 'AIFKEYS' OF STRUCTURE <ls_idx_tab> TO <ls_aif_keys>.
    <ls_aif_keys>-ns      = ls_aif_keys-ns.
    <ls_aif_keys>-ifname  = ls_aif_keys-ifname.
    <ls_aif_keys>-ifver   = ls_aif_keys-ifver.
    ASSIGN COMPONENT 'ADMIN' OF STRUCTURE <ls_idx_tab> TO <ls_admin>.
    <ls_admin>-create_date = <ls_aif_flbook_head>-create_date.
    <ls_admin>-create_time = <ls_aif_flbook_head>-create_time.
    ASSIGN COMPONENT 'STATUS' OF STRUCTURE <ls_idx_tab> TO <lv_status>.
    <lv_status> = <ls_aif_flbook_head>-status.
  ENDLOOP.

  SORT <lt_idx_tab> AS TEXT.
  ls_idx_data-index_tabname = '/AIF/STD_IDX_TBL'.
  ls_idx_data-data = lr_idx_tab.
  ls_idx_data-sel_type = /aif/if_globals=>gc_sel_type-single_sel.
  APPEND ls_idx_data TO cs_keys_mapping-msg_data_tbl.
*prepare data for display in Date Message view
  DESCRIBE TABLE cs_keys_mapping-msg_data_tbl LINES lv_lines.
  DO lv_lines TIMES.
    lv_index = sy-index.
    READ TABLE cs_keys_mapping-
msg_data_tbl ASSIGNING <ls_msg_data_tbl> INDEX lv_index.
    ASSIGN <ls_msg_data_tbl>-data->* TO <lt_data>.
    LOOP AT <lt_data> ASSIGNING <ls_data>.
      ASSIGN COMPONENT /aif/if_globals=>gc_eh_fix_key_flds-
msgguid OF STRUCTURE <ls_data> TO <lv_msgguid>.
      ASSIGN COMPONENT /aif/if_globals=>gc_eh_fix_key_flds-
status OF STRUCTURE <ls_data> TO <lv_status>.
```

```
      ASSIGN COMPONENT 'AIFKEYS' OF STRUCTURE <ls_data> TO <ls_aif_keys>.
      ASSIGN COMPONENT 'ADD_2' OF STRUCTURE <ls_data> TO <ls_add_info1>.
      ASSIGN COMPONENT 'AIFKEYS' OF STRUCTURE ls_for_sort TO <ls_aif_keys2>.
      ASSIGN COMPONENT 'ADD_2' OF STRUCTURE ls_for_sort TO <ls_add_info2>.
      <ls_aif_keys2> = <ls_aif_keys>.
      <ls_add_info2> = <ls_add_info1>.
      ls_for_sort-msgguid = <lv_msgguid>.
      CREATE DATA lr_row_data LIKE <ls_data>.
      ASSIGN lr_row_data->* TO <ls_row_data>.
      <ls_row_data> = <ls_data>.
      GET REFERENCE OF <ls_row_data> INTO ls_for_sort-row_data.
      TRY.
          IF me->/aif/if_selection_engine~check_authorization(
              is_ifkeys         = cs_keys_mapping
              is_data           = ls_for_sort
              iv_ecorr          = iv_emergency_correct_mode ) = abap_false.
* all the single selection data (cross interfaces/namespaces) are stored in the
single sort table
            APPEND ls_for_sort TO ct_for_sort_tbl.
            CLEAR ls_for_sort.
          ENDIF.
        CATCH /aif/cx_error_handling_general INTO lr_exception.
          ls_return-id = lr_exception->if_t100_message~t100key-msgid.
          ls_return-number = lr_exception->if_t100_message~t100key-msgno.
          ls_return-message_v1 = lr_exception->if_t100_message~t100key-attr1.
          ls_return-message_v2 = lr_exception->if_t100_message~t100key-attr2.
          ls_return-message_v3 = lr_exception->if_t100_message~t100key-attr3.
          ls_return-message_v4 = lr_exception->if_t100_message~t100key-attr4.
          APPEND ls_return TO et_return.
          CLEAR ls_return.
      ENDTRY.
    ENDLOOP.
  ENDDO.

ENDMETHOD.
```

Implementation of method SELECT_SINGLE_IDX_TAB_ENTRY

```
METHOD /aif/if_selection_engine~select_single_idx_tab_entry.
  DATA: ls_aif_flbook_head TYPE zaif_flbook_head.
  FIELD-SYMBOLS: <lv_guid32> TYPE guid_32,
                 <ls_idx_data> TYPE any,
                 <lv_status> TYPE /aif/proc_status.

  ASSIGN cs_view1_data-data_row->* TO <ls_idx_data>.
  ASSIGN COMPONENT 'MSGGUID' OF STRUCTURE <ls_idx_data> TO <lv_guid32>.
  SELECT SINGLE * FROM zaif_flbook_head INTO ls_aif_flbook_head
    WHERE guid_32 = <lv_guid32>.

  ASSIGN COMPONENT 'STATUS' OF STRUCTURE <ls_idx_data> TO <lv_status>.
  <lv_status> = ls_aif_flbook_head-status.

ENDMETHOD.
```

CUSTOM-SPECIFIC LOGGING ENGINE

The logging engine is responsible for the selection of data displayed in Log Messages view.

In order to implement a custom-specific logging engine several possibilities exist:

- Implement interface /AIF/IF_LOGGING_ENGINE
- Create a new class that inherits /AIF/CL_LOGGING_ENGINE_BASE
- Create a new class that inherits from an concrete logging engine class

Which of the options fits best for your needs depends on your scenario.

Example Implementation

In the following the implementation of a logging engine for the scenario outlined before will be shown.

You should at least implement following methods:

- RETRIEVE_LOG_MESSAGES: Retrieves the log messages and prepares them for displaying in Log Messages view. **Note:** If your log messages are not stored in the SAP Application Log, you will have to create a temporary application log in order to display the log messages in *Monitoring and Error Handling.*

Create a new class that inherits from /AIF/CL_LOGGING_ENGINE_BASE.

Implementation of RETRIEVE_LOG_MESSAGES

The method will retrieve the data for the selected messages from ZAIF_FLBOOK_MSGS and create a temporary application log.

```
METHOD /aif/if_logging_engine~retrieve_log_messages.
  DATA: lv_guid32            TYPE guid_32,
        lt_msgguids          TYPE TABLE OF guid_32,
        lt_aif_flbook_msgs   TYPE TABLE OF zaif_flbook_msgs,
        ls_msginfo           TYPE /aif/msg_info_s,
        lv_extnum            TYPE balnrext,
        lv_subobject_text    TYPE balsubobjt,
        lv_log_handle        TYPE balloghndl,
        ls_return            TYPE bapiret2.

  FIELD-SYMBOLS: <ls_data1_node>    TYPE /aif/treenode_st,
                 <ls_aif_flook_msg> TYPE zaif_flbook_msgs.

  LOOP AT it_data1_nodes ASSIGNING <ls_data1_node>.
    CLEAR: lt_aif_flbook_msgs,
           ls_msginfo.

    ls_msginfo-msgguid = <ls_data1_node>-msgguid.
    ls_msginfo-ns = is_aif_keys-ns.
    ls_msginfo-ifname = is_aif_keys-ifname.
    ls_msginfo-ifver = is_aif_keys-ifver.

*retrieve the messages from ZAIF_FLBOOK_MSGS
    SELECT * FROM zaif_flbook_msgs INTO CORRESPONDING FIELDS OF TABLE lt_aif_flb
ook_msgs
      WHERE guid_32 = <ls_data1_node>-msgguid.

*   create temporary application log to display message in Log Messages view
    lv_extnum = <ls_data1_node>-msgguid.
    CONCATENATE is_aif_keys-ns '/' is_aif_keys-ifname '/' is_aif_keys-
```

```
ifver INTO lv_subobject_text.
    me->create_bal_log(
      EXPORTING
        iv_object         = iv_object
        iv_object_text    = 'SAP Application Interface Framework'
        iv_extnumber      = lv_extnum
        iv_subobject      = iv_subobject
        iv_subobject_text = lv_subobject_text
        iv_repid          = sy-repid
      IMPORTING
        ev_log_handle     = lv_log_handle
          ).

*add the messages retrieved from ZAIF_FLBOOK_MSGS to the temporary application l
og
    LOOP AT lt_aif_flbook_msgs ASSIGNING <ls_aif_flook_msg>.
      ls_msginfo-msg-msgty = <ls_aif_flook_msg>-type.
      ls_msginfo-msg-msgid = <ls_aif_flook_msg>-id.
      ls_msginfo-msg-msgno = <ls_aif_flook_msg>-msg_number.
      ls_msginfo-msg-msgv1 = <ls_aif_flook_msg>-message_v1.
      ls_msginfo-msg-msgv2 = <ls_aif_flook_msg>-message_v2.
      ls_msginfo-msg-msgv3 = <ls_aif_flook_msg>-message_v3.
      ls_msginfo-msg-msgv4 = <ls_aif_flook_msg>-message_v4.

      CALL FUNCTION 'BAL_LOG_MSG_ADD'
        EXPORTING
          i_log_handle     = lv_log_handle
          i_s_msg          = ls_msginfo-msg
        IMPORTING
          e_s_msg_handle   = ls_msginfo-msghandle
        EXCEPTIONS
          log_not_found    = 1
          msg_inconsistent = 2
          log_is_full      = 3
          OTHERS           = 4.
      IF sy-subrc <> 0.
        ls_return-type = sy-msgty.
        ls_return-id = sy-msgid.
        ls_return-number = sy-msgno.
        ls_return-message_v1 = sy-msgv1.
        ls_return-message_v2 = sy-msgv2.
        ls_return-message_v3 = sy-msgv3.
        ls_return-message_v4 = sy-msgv4.
        APPEND ls_return TO et_return.
        CLEAR ls_return.
        CONTINUE.
      ENDIF.

      INSERT ls_msginfo INTO TABLE et_msginfo.
      CLEAR: ls_msginfo-msg.
    ENDLOOP.
  ENDLOOP.
ENDMETHOD.
```

DEFINE CUSTOM-SPECIFIC ENGINES

To be able to use your custom-specific engines you have to maintain your classes in Customizing activity *Define Custom Engines* of the SAP Application Interface Framework (transaction /AIF/CUST). The custom specific engines are grouped by namespace.

Enter a name for your custom-specific engines in *Customer-Specific AIF Engine*. Furthermore, you can enter a description.

Enter the class names of your custom-specific engines. It is not necessary to enter a class for every engine type. When you assign engines to your interfaces you can mix different customer engines. Furthermore, you can use the engines delivered with the SAP Application Interface Framework.

Example:

For the example implementation a custom-specific engine class is required for every engine type.

Field	Value
Customer-Specific AIF Engine	BOOKING_REPORT
Application Engine Class	ZCL_AIF_APPL_ENGINE_FLBOOK
Persistence Engine Class	ZCL_AIF_PERSIST_ENGINE_FLBOOK
Selection Engine Class	ZCL_AIF_SEL_ENGINE_FLBOOK
Logging Engine Class	ZCL_AIF_LOG_ENGINE_FLBOOK

INTERFACE DEFINITION IN THE SAP APPLICATION INTERFACE FRAMEWORK

In order to be able to monitor the execution of the flight booking report an interface in the SAP Application Interface Framework is required. However, before the interface can be created an SAP data structure and raw data structure is needed. For the example a structure is required that contains the values entered in the selection screen. Furthermore, customer data, flight data and travel agency data should be displayed. If the flight booking is successfully created the booking data should be displayed, too.

Create Structure and Define Interface

Structure ZAIF_FLIGHT_BOOKING

Component	Component Type
CARRID	S_CARRID
CONNID	S_CONN_ID
FLDATE	S_DATE
CUSTOMER_ID	S_CUSTOMER
CLASS	S_CLASS
AGENCYID	S_AGENCY
BOOKING_DATA	SBOOK
FLIGHT_DATA	SFLIGHT

CUSTOMER_DATA	SCUSTOM
TRAVEL_AGENCY	STRAVELAG

Structure ZAIF_SAP_EXAMPLE_FLIGHTBOOKING (interface raw/SAP structure)

Component	Component Type
FLIGHT_BOOKING	ZAIF_FLIGHT_BOOKING

After the structure has been created the interface can be defined in the Customizing activity *Define Interface* of the SAP Application Interface Framework. Enter the structure you created as raw data structure and SAP data structure. For example your interface definition could look as follows:

Field	Value
Namespace	EXAMPL
Interface Name	BOOKING
Interface Version	1
SAP Data Structure	ZAIF_SAP_EXAMPLE_FLIGHTBOOKING
Raw Data Structure	ZAIF_SAP_EXAMPLE_FLIGHTBOOKING

Specify Interface Engines

After the interface was defined, you have to specify which engines should be used for your interface. You can assign the engines in *Interface Development→Additional Interface Properties→Specify Interface Engines* in Customizing for the SAP Application Interface Framework (transaction /AIF/CUST).

Select an interface and maintain the engines that should be used for this interface. In order to enter a custom-specific engine select *Customer-Specific* from the drop-down box. Then you can enter the *Namespace* and the *Customer Engine* that should be used.

You can specify different customer engines for different engine types. Furthermore, you can mix the customer engines with the engines delivered with the SAP Application Interface Framework:

Example 1:

Field	Value
Application Engine	Customer-Specific
Namespace	NS_1
Customer Engine	CUST_ENGINE_1
Persistence Engine	Customer-Specific
Namespace	NS_1
Customer Engine	CUST_ENGINE_1
Selection Engine	AIF Index Tables

Logging Engine	Customer-Specific
Namespace	NS_2
Customer Engine	CUST_ENGINE_2

Example 2:

For the interface example scenario of flight bookings for every engine type a customer-specific engine exists:

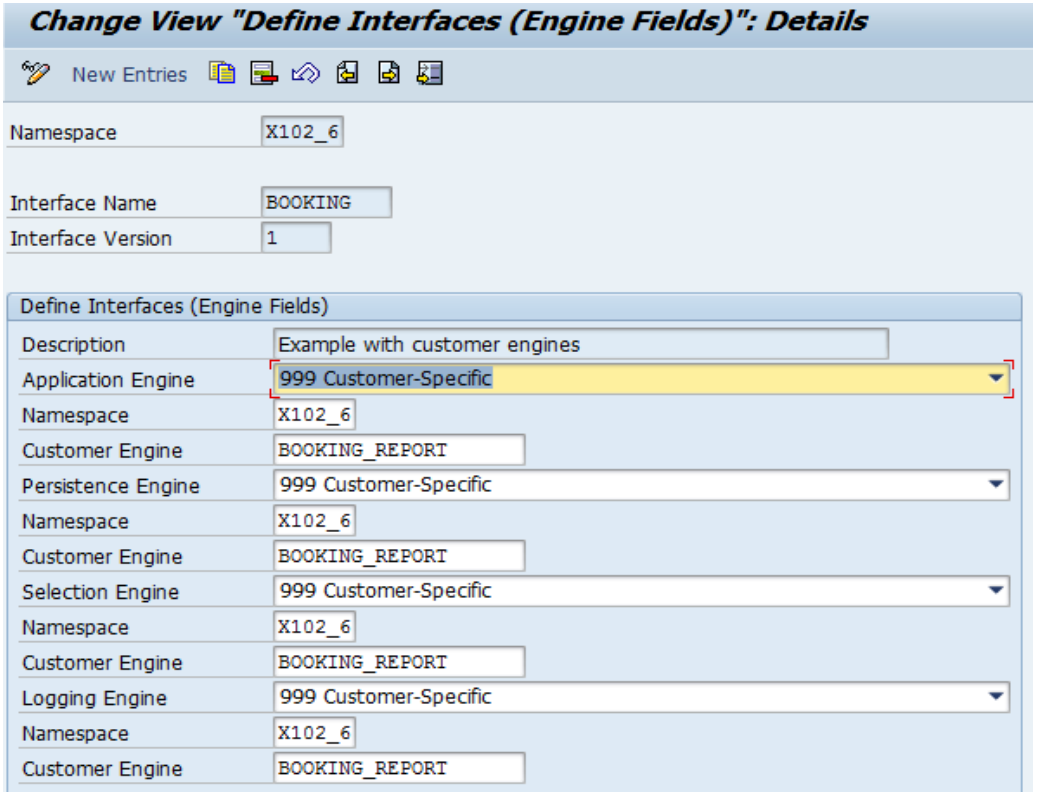

Figure 1: Specify Customer-Specific Interface Engines

TEST MONITORING AND ERROR HANDLING

In order to test the custom-specific engines for the scenario outlined before, execute the report to create flight bookings (p. 3). Enter some correct data in order to create a flight booking. Enter some data where no flight booking can be created, for example, enter a non-existing flight or customer. Execute the report several times in order to have several messages.

Start transaction /AIF/ERR and select your interface. Make sure to select all status so that all flight bookings that you tried to create will be selected.

If you have messages where errors occurred, try to edit, restart and cancel them. In order to be able to edit a message you have to define the fields that you want to change as editable (for more information refer to the cookbook). Alternatively, you can set *Emergency Correction Mode* indicator on the selection screen of *Monitoring and Error Handling.* **Note:** The example code for method UPDATE of the persistence engine (p.10), only allows you to change fields of structure FLIGHT_BOOKING. Figure 2 shows the *Monitoring and Error Handling* transaction with several messages to create flight bookings.

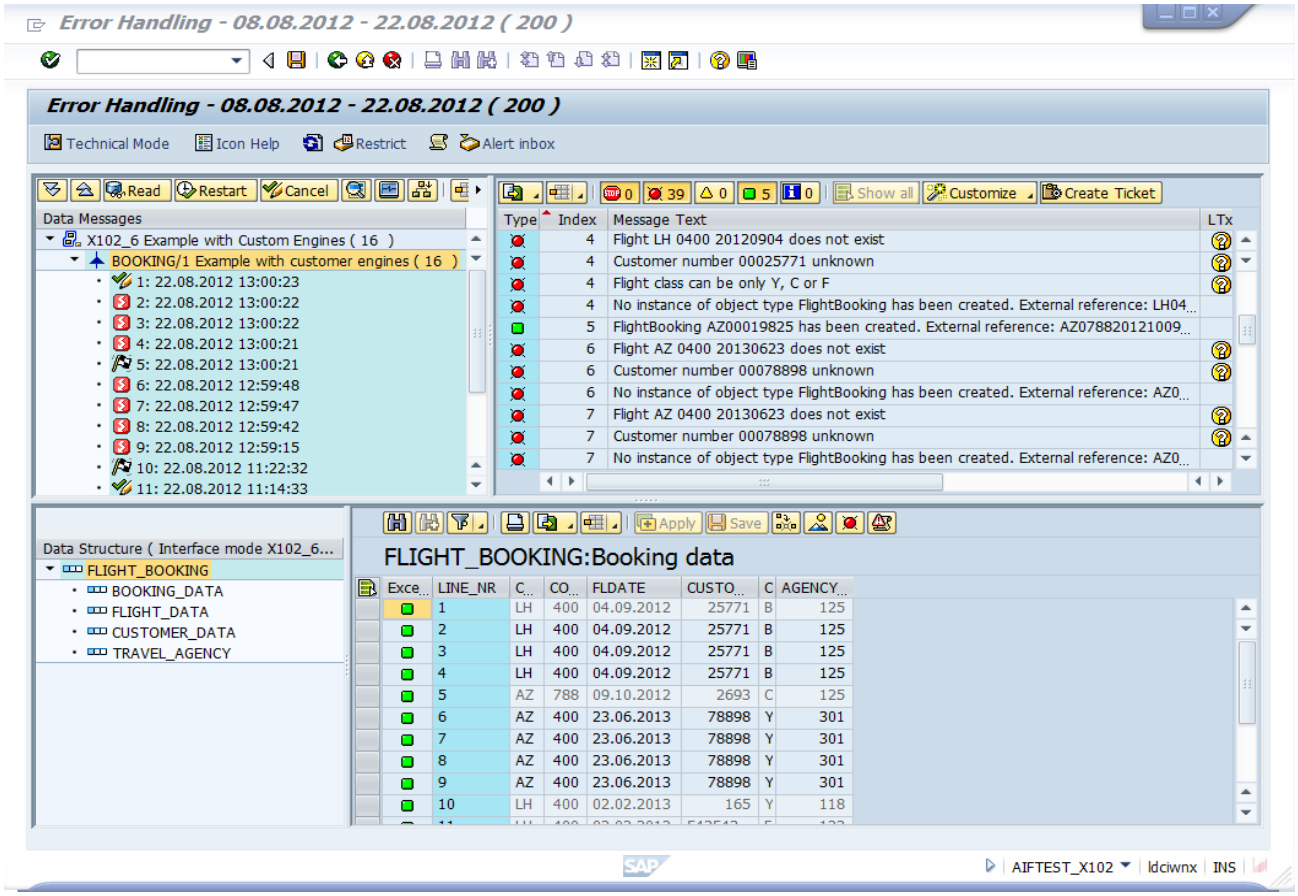

Figure 2: Monitoring Report to Create Flight Bookings with Monitoring and Error Handling

www.sap.com