/AIF/CX_ERROR_HANDLING_GENERAL	EXCEPTION		AIF Error Handling Exception Class
/AIF/CX_INF_DET_BASE	EXCEPTION		Generic Exception for AIF Enabler
/AIF/CX_AIF_ENGINE_NOT_FOUND	EXCEPTION		General exception class for AIF engines
/AIF/CX_AIF_ENGINE_BASE	EXCEPTION		Base Exception Class for AIF Engines

```
METHOD transfer_to_aif.

  DATA: ls_finf                   TYPE /aif/t_finf,
        lv_guid         TYPE guid_32,
        lv_guid_old     TYPE guid_32,
        ls_textid       TYPE scx_t100key,
        lv_status       TYPE smw3istate,
        ls_aif_bdoc_stat TYPE zaif_t_bdocstat.
  DATA: lt_aif_bdoc_msgs TYPE smw_errtab,
        lr_aif_bdoc_stat TYPE REF TO zcl_aif_bdoc_status_mapping,
        lt_log_messages  TYPE /aif/bal_t_msg,
        ls_log_message   TYPE bal_s_msg,
        lr_if_det_engine_bdoc     TYPE REF TO /aif/cl_inf_det_engine_xml,
        lv_name                   TYPE /aif/lfieldname_infdet.
  DATA: lv_timestamp          TYPE string,
        ls_bal_context        TYPE /aif/bal_context.

  FIELD-SYMBOLS: <ls_bdoc_msg> TYPE smog_merr.

  lv_guid = is_bdoc_header-bdoc_id.

  lr_if_det_engine_bdoc ?= /aif/cl_aif_engine_factory=>get_inf_det_engine(
  iv_type           = /aif/cl_pers_config=>c_if_type_xml
  iv_cust_ns        = space
  iv_cust_type      = space
).

  CALL METHOD lr_if_det_engine_bdoc->get_typename
    EXPORTING
      iv_input = is_any_structure
    IMPORTING
      ev_name1 = lv_name.

  CALL METHOD lr_if_det_engine_bdoc->/aif/if_inf_det_engine~determine_inf
    EXPORTING
      iv_name1      = lv_name
      iv_name2      = space
      iv_input      = is_any_structure
      iv_msgguid    = lv_guid
    IMPORTING
      es_finf       = ls_finf.

  IF NOT gr_bdoc_enabler IS INITIAL.
    lv_guid_old = gr_bdoc_enabler->get_msgguid( ).
  ENDIF.
  IF gr_bdoc_enabler IS INITIAL OR lv_guid <> lv_guid_old.
```

```
    CREATE OBJECT gr_bdoc_enabler
      EXPORTING
        iv_msgguid   = lv_guid
        iv_ns        = ls_finf-ns
        iv_ifname    = ls_finf-ifname
        iv_ifversion = ls_finf-ifversion.
  ENDIF.

  lv_status = is_bdoc_header-bdoc_state.

*      determine status for AIF
  lr_aif_bdoc_stat = zcl_aif_bdoc_status_mapping=>get_instance( ).
  ls_aif_bdoc_stat = lr_aif_bdoc_stat-
>get_single_status( iv_bdoc_status = lv_status ).

*     move bdoc msg to log messages in order to be able to create alerts
  LOOP AT it_bdoc_msg ASSIGNING <ls_bdoc_msg>.
    ls_log_message-msgty = <ls_bdoc_msg>-type.
    ls_log_message-msgid = <ls_bdoc_msg>-id.
    ls_log_message-msgno = <ls_bdoc_msg>-number.
    ls_log_message-msgv1 = <ls_bdoc_msg>-message_v1.
    ls_log_message-msgv2 = <ls_bdoc_msg>-message_v2.
    ls_log_message-msgv3 = <ls_bdoc_msg>-message_v3.
    ls_log_message-msgv4 = <ls_bdoc_msg>-message_v4.

    ls_bal_context-sydate = <ls_bdoc_msg>-err_date.
    ls_bal_context-sytime = <ls_bdoc_msg>-err_time.

    ls_log_message-context-tabname = '/AIF/BAL_CONTEXT'.
    ls_log_message-context-value = ls_bal_context.
    APPEND ls_log_message TO lt_log_messages.
  ENDLOOP.

  gr_bdoc_enabler->/aif/if_enabler_base~update(
    iv_message_status_flag      = ls_aif_bdoc_stat-aif_status
    is_raw_structure            = is_any_structure
    it_log_messages             = lt_log_messages
    iv_do_commit                = iv_do_commit
        ).

ENDMETHOD.
```

Define Custom-Specific Engines

To be able to use your BDoc-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 BDoc-specific engines in *Customer-Specific AIF Engine*. Furthermore, you can enter a description. Enter the class names of your BDoc-specific engines.

Example:

For the pure monitoring scenario of Business Partner BDocs a BDoc-specific engine class is required for every engine type.

Field	Value
Customer-Specific AIF Engine	BDOC_ENGINES
Application Engine Class	ZCL_AIF_APPL_ENGINE_BDOC
Persistence Engine Class	ZCL_AIF_PERSIST_ENGINE_BDOC
Selection Engine Class	ZCL_AIF_SELECTION_ENGINE_BDOC
Logging Engine Class	ZCL_AIF_LOGGING_ENGINE_BDOC

INTERFACE DEFINITION IN THE SAP APPLICATION INTERFACE FRAMEWORK

In order to be able to monitor the BDoc an interface in the SAP Application Interface Framework is required.

Scenario 1 Implementation:

Define Interface:

The interface can be defined in the Customizing activity *Define Interface* of the SAP Application Interface Framework. Enter the structure of the related data type of your BDoc Model as raw data structure and SAP data structure. You can find the information in the BDoc Modeler (transaction SBDM). For example your interface definition for an interface monitoring Business Partner BDocs could look as follows:

Field	Value
Namespace	BDOC
Interface Name	BUPA_MAIN
Interface Version	1
SAP Data Structure	BUS_EI_MAIN
Raw Data Structure	BUS_EI_MAIN

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.

For the pure monitoring scenario of BDoc for every engine type a BDoc-specific engine exists:

Field	Value
Application Engine	Customer-Specific
Namespace	BDOC
Customer Engine	BDOC_ENGINES
Persistence Engine	Customer-Specific
Namespace	BDOC

Customer Engine	BDOC_ENGINES
Selection Engine	Customer-Specific
Namespace	BDOC
Customer Engine	BDOC_ENGINES
Logging Engine	Customer-Specific
Namespace	BDOC
Customer Engine	BDOC_ENGINES

Scenario 2 Implementation:

Define Interface

The interface can be defined in the Customizing activity *Define Interface* of the SAP Application Interface Framework. Enter the structure of the related data type of your BDoc Model as raw data structure and SAP data structure. You can find the information in the BDoc Modeler (transaction SBDM). For example your interface definition for the BDoc Business Partner monitoring could look as follows:

Field	Value
Namespace	BDOC
Interface Name	BUPA_MAIN
Interface Version	1
SAP Data Structure	BUS_EI_MAIN
Raw Data Structure	BUS_EI_MAIN

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 your BDoc Business Partner interface and maintain the engines that should be used for this interface.

For the AIF enabler scenario of BDoc monitoring only the engine types for the Application Engine and Persistency engine should be used a BDoc-specific engine.

Field	Value
Application Engine	Customer-Specific
Namespace	BDOC
Customer Engine	BDOC_ENGINES
Persistence Engine	Customer-Specific
Namespace	BDOC
Customer Engine	BDOC_ENGINES

Selection Engine	AIF Index Tables
Namespace	
Customer Engine	
Logging Engine	AIF Application Log
Namespace	
Customer Engine	

Implement Call of AIF Enabler

In order to monitor existing BDocs with the AIF Enabler for BDocs, there are two different possibilities for implementing. In the first one the complete Flow Context of the BDoc needs to be encapsulated. Therefor all flow service function modules need to be replaced with new custom flow service functions. After calling the standard flow service function the AIF Enabler for BDocs can be called (see Figure 1).

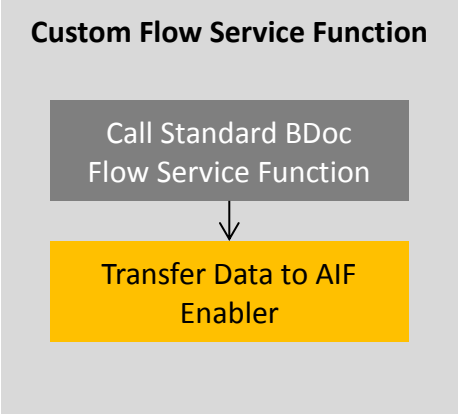

Figure 1: Processing of BDocs with AIF Enabler

Instead of implementing a new flow service function there is also the possibility to call the enabler in an implicit enhancement at the beginning of method CL_SMW_BDOCSTORE->PERSIST_BDOC.

```
ENHANCEMENT 1  ZAIF_PERSIST_BDOC_ENABLING.    "active version


DATA: ls_receiver TYPE smw_recinf.
DATA: lt_receiver_errors TYPE SMW_ERRTAB.
DATA: ls_error LIKE LINE OF lt_receiver_errors.

FIELD-SYMBOLS: <ls_bdoc_body_ext> TYPE any.
DATA: dref_data TYPE REF TO data.

*2.Call AIF Enbaler in order to create AIF index table entries, raise alerts etc


  CREATE DATA dref_data TYPE (bdoc_header-ddic2).
  ASSIGN dref_data->* TO <ls_bdoc_body_ext>.

 CALL METHOD cl_smw_bdocstore=>get_bdoc
   EXPORTING
    bdoc_id           = bdoc_header-bdoc_id
    get_bdoc_header   = ' '
    get_body_ext      = 'X'
   IMPORTING
    bdoc_body_ext     = <ls_bdoc_body_ext>
```

```
  EXCEPTIONS
    invalid_bdoc_id    = 1
    inconsistent_body = 2
    failed             = 3
    OTHERS             = 4.
  IF sy-subrc = 0 AND NOT <ls_bdoc_body_ext> IS INITIAL.
    TRY.
      IF NOT set_errors IS INITIAL.
       CALL METHOD zcl_aif_enabler_bdoc=>transfer_to_aif
         EXPORTING
          is_any_structure = <ls_bdoc_body_ext>
          is_bdoc_header   = bdoc_header
          it_bdoc_msg      = bdoc_errors.

      ELSEIF NOT set_receivers IS INITIAL.
       LOOP AT bdoc_receivers INTO ls_receiver.
         LOOP AT ls_receiver-errors INTO ls_error.
          ls_error-id = '/AIF/MES'.
          ls_error-number = 000.
          ls_error-message_v1 = ls_error-message(50).
          ls_error-message_v2 = ls_error-message+50(50).
          ls_error-message_v3 = ls_error-message+100(50).
          ls_error-message_v4 = ls_error-message+150(50).
          APPEND ls_error TO lt_receiver_errors.
         ENDLOOP.
       ENDLOOP.

       CALL METHOD zcl_aif_enabler_bdoc=>transfer_to_aif
         EXPORTING
          is_any_structure = <ls_bdoc_body_ext>
          is_bdoc_header   = bdoc_header
          it_bdoc_msg      = lt_receiver_errors.
      ENDIF.
     CATCH  /aif/cx_enabler_base /aif/cx_aif_engine_base /aif/cx_aif_engine_not_
found /aif/cx_inf_det_base /aif/cx_error_handling_general.
    ENDTRY.
  ENDIF.

ENDENHANCEMENT.
```

If the enabler for BDocs is called in method CL_SMW_BDOCSTORE->PERSIST_BDOC the BDoc's status in the AIF's index tables will be updated every time the BDoc status is updated. Furthermore, you do not have to create a new flow service function for the BDocs you want to monitor in the SAP Application Interface Framework.

Note: If you set up more than one interface per BDoc Type, you should setup an interface determination in customizing of the SAP Application Interface Framework in *System Configuration→Interface Determination→Interface Determination for XML Interface.* Use the structure of the related data type of your BDoc Model as structure. You can find the information in the BDoc Modeler (transaction SBDM).

TEST MONITORING AND ERROR HANDLING

In order to test the BDoc specific engines, create a Business Partner in the SAP Application System to send a BDoc from the SAP application System to the SAP CRM System.

Start transaction /AIF/ERR and select your interface. Make sure to select all status so that all BDocs that you tried to send will be selected. If you have messages where errors occurred, try to restart and cancel them.

www.sap.com