SAP Application Interface Framework with Error and Conflict Handler

Applies to:

SAP Application Interface Framework 2.0 with component AIFX and SAP NetWeaver 7.31.

Summary

This document describes the integration scenarios between the SAP Application Interface Framework and the Error and Conflict Handler. Two different scenarios exist. The first scenario allows you to display data of the ECH in Monitoring and Error Handling. The second scenario enables you to handle messages processed with the SAP Application Interface Framework in ECH.

Author: Verena Wörner

Company: SAP AG

Created on: 03 December 2012

Author Bio

Verena Wörner is a developer for SAP Custom Development. She is a member of the development team for the SAP Application Interface Framework.

Table of Contents

Overview

SAP Application Interface Framework provides you with monitoring and error handling functionality. The Error and Conflict Handler (ECH) is an error handling tool implementing the concept of forward error handling provided by SAP NetWeaver.

The integration of the SAP Application Interface Framework and ECH involve two different scenarios:

- Display messages of ECH in Monitoring and Error Handling (/AIF/ERR). This scenario allows you, to also display messages that you would usually see in ECH in *Monitoring and Error Handling* of SAP Application Interface Framework. In order to display this message in *Monitoring and Error Handling* you have to do some customizing in the SAP Application Interface Framework. This guide will show you an example of an interface that will create a customer for the flight model. The example will then show you how you can monitor those messages in the SAP Application Interface Framework
- Display messages processed with the SAP Application Interface Framework in ECH. This scenario applies if you want to make use of the interface implementation capabilities of the SAP Application Interface Framework. However, for doing error handling you would like to use the ECH. The document will provide you with an example of how you can implement such an interface.

Preparation

Before you can use the SAP Application Interface Framework with the Error and Conflict Handler (ECH), you should make sure that the ECH is activated. You can activate the ECH in Customizing (transaction SPRO) under *Cross Application Components→Processes and Tools for Business Applications→Enterprise Services→Error and Conflict Handler→Activate Error and Conflict Handler.*

A service interface should have been created in ESR. Then you can generate the proxy class in your application system.

Display Messages of ECH in Monitoring and Error Handling

If you choose this scenario you can do error handling (for example, edit, save, restart and cancel) for your ECH interface in the SAP Application Interface Framework's *Monitoring and Error Handling*, too. The locking, version handing, and status controlling reuse ECH standard functions, so that they are consistent and synchronized between *Monitoring and Error Handling* in the SAP Application Interface Framework and ECH monitoring. This integration scenario is independent of the SAP Application Interface Framework runtime as *Monitoring and Error Handling* works on the PPO persistence layer utilized by ECH.

First of all you need an implemented proxy interface. If you have not already done so generate the proxy class from the service interface you would like to implement in transaction SPROXY. Implement the proxy method.

Example Implementation of proxy class method to create customers:

The implementation below shows how the implementation of the proxy method could look like. The interface will create a customer for the flight model. In order to create the customer BAPI_FLCUST_CREATEFROMDATA will be called. To be able to display the messages in ECH you need to call method COLLECT of class CL_FEH_REGISTRATION in case the BAPI returns an error.

```
METHOD zaif_x102_ii_ech_aif_in01~test_inbound01.
  DATA: lt_add_data       TYPE zaif_test_zgtp_test_raw_ad_tab,
        ls_customer_data  TYPE bapiscunew,
        lt_return         TYPE bapirettab,
        ls_return         TYPE bapiret2.
  FIELD-SYMBOLS: <ls_add_data> TYPE zaif_test_zgtp_test_raw_add_da.

  lt_add_data = input-zgtp_test_raw_mt-raw_add_data_t.

  LOOP AT lt_add_data ASSIGNING <ls_add_data>.
    ls_customer_data-custname = <ls_add_data>-add_data_key.
```

```
    ls_customer_data-street   = <ls_add_data>-add_data1.
    ls_customer_data-postcode = <ls_add_data>-add_data2.
    ls_customer_data-city     = <ls_add_data>-add_data3.
    ls_customer_data-countr   = <ls_add_data>-add_data4.
    ls_customer_data-custtype = <ls_add_data>-add_data5.
    ls_customer_data-email    = <ls_add_data>-add_data6.

    CALL FUNCTION 'BAPI_FLCUST_CREATEFROMDATA'
      EXPORTING
        customer_data = ls_customer_data
      TABLES
        return        = lt_return.
  ENDLOOP.

*check if an error occured during when creating the customer
  READ TABLE lt_return INTO ls_return WITH KEY type = 'E'.
  IF sy-subrc = 0.
    DATA: lr_feh_registration TYPE REF TO cl_feh_registration,
          lv_error_category   TYPE ech_dte_error_category,
          ls_main_object      TYPE ech_str_object.

    lr_feh_registration = cl_feh_registration=>s_initialize( ).
    lv_error_category = 'PRE'. "as per table  /SAPPO/SERR_CAT.

    ls_main_object-objcat  = '1'.
    ls_main_object-objtype = 'SCUSTOM'.
    ls_main_object-objkey  = ls_customer_data-custname.

    TRY.
        lr_feh_registration->collect(
            i_single_bo      = input
            i_error_category = lv_error_category
            i_main_message   = ls_return
            i_main_object    = ls_main_object ).
      CATCH cx_ai_system_fault .
    ENDTRY.

    cl_proxy_fault=>raise(
      exception_class_name   = 'ZAIF_TEST_CX_ZGT_FAULT'
      bapireturn_tab         = lt_return
       ).
  ENDIF.
ENDMETHOD.
```

Furthermore, you need an ECH action class, which should implement interface IF_ECH_ACTION. Your action class needs a static attribute GO_ECH_ACTION, which should have the type of your action class. The visibility of the attribute should be set to “protected”. Additionally, you have to implement method S_CREATE, with the coding displayed below:

```
METHOD if_ech_action~s_create.
  IF NOT go_ech_action IS BOUND.
    CREATE OBJECT go_ech_action.
  ENDIF.

  r_action_class = go_ech_action.
ENDMETHOD.
```

Furthermore, you have to maintain some ECH specific customizing. Therefore, go to transaction SPRO and perform following steps:

- Create an ECH business process and assign the action class as well as the notification class. To do so go to *Cross Application Components→Processes and Tools for Business Applications→Enterprise Services→Error and Conflict Handler→Define Business Process*. You have to maintain following values:
    - Component
    - Business Process
    - Action Class (**Note:** the action class has to implement interface IF_ECH_ACTION)
    - Notification: CL_FEH_REGISTRATION
    - Business Process Description
- Create a business process for the Postprocessing Office. Go to *Cross Application Components→Processes and Tools for Business Applications→Enterprise Services→Error and Conflict Handler→Postprocessing Office→Business Process→Define Business Processes*. Maintain following fields:
    - Software Component
    - Process
    - Process Description
- Map the ECH Business Process to the PPO Business Process. Go to *Cross Application Components→Processes and Tools for Business Applications→Enterprise Services→Error and Conflict Handler→Define Business Process for Postprocessing Office*. Maintain following fields:
    - Component
    - Business Process (ECH)
    - Software Component (PPO)
    - PPO Process
- Assign the proxy class for your service operation to the ECH process. Go to *Cross Application Components→Processes and Tools for Business Applications→Enterprise Services→Error and Conflict Handler→Assign Caller to a Business Process.* Maintain following fields:
    - API Name
    - API Component
    - Call Type
    - Component Business Process

In order to be able to display messages of this service interface in *Monitoring and Error Handling*, you need to do some SAP Application Interface Framework specific customizing. This can be done in transaction /AIF/CUST. Create a new interface in Customizing activity *Define Interfaces*. Select a namespace and maintain an interface name and an interface version. Enter the name of your proxy class into *Proxy Class Inbound*. The *Raw Data Structure* and *Record Type in Raw Structure* should be filled automatically. Furthermore, enter the record type's type as SAP Data Structure.

Furthermore, you have to define which data messages should belong to your interface in the SAP Application Interface Framework. Therefore, go to Customizing activity *Additional Interface Properties→Assign ECH Component*. Select your interface and enter the *Software Component* and the *ECH Process*.

To enable the *Monitoring and Error Handling* to select the messages processed with ECH, it is necessary to define the corresponding interface engines. You can do this in Customizing activity *Specify Interface Engines*. Select your interface and maintain following engines:

- Application Engine: ECH

- Persistence Engine: ECH
- Selection Engine: ECH
- Logging Engine: ECH

If you want to be able to edit messages in *Monitoring and Error Handling* you have to define those fields as changeable. You can do this in customizing activity *Define Namespace-Specific Features* (see cookbook) and *Define Interface-Specific Features* (see cookbook).

After sending some messages to your new interface you should be able to see them in *Monitoring and Error Handling* (transaction /AIF/ERR).

In order to be able to resolve errors some further coding is required for an ECH interface. You can implement following methods of your action class:

Retry: this method needs to be implemented to trigger the reprocessing of a message. The message will be called if you click *Restart* in *Monitoring and Error Handling* or if you select *Repeat* in *Error and Conflict Handler*

Fail: This method can be used if the error cannot be resolved. After the method is executed the message cannot be reprocessed anymore. You should at least call method CL_FEH_REGISTRATION=>S_FAIL. This method will be called if you select *Cancel* in *Monitoring and Error Handling* or if you select *Discard* in *Error and Conflict Handler*

Finish: Within this method you can for example trigger the sending of an email, to inform a user that the processing steps have to be performed manually. In any case method CL_FEH_REGISTRATION=>S_FINISH should be called at the end. The method will be called if you select *Confirm* in *Error and Conflict Handler*.

Note: Since this document focuses on the SAP Application Interface Framework the implementation of an action class is shown for reasons of completeness. If you need more information on how the action class for the ECH can be implemented refer to the Application Help for the Error and Conflict Handler. In the following only simple examples are given.

Example Implementation of Retry Method

```
METHOD if_ech_action~retry.
  DATA: lr_feh_registration TYPE REF TO cl_feh_registration,
        ls_api_data         TYPE zaif_test_zgtp_test_raw_mt,
        lr_pre_input        TYPE REF TO data,
        lr_post_input       TYPE REF TO data,
        ls_main_object      TYPE ech_str_object,
        ls_customer_data    TYPE bapiscunew,
        lt_return           TYPE bapirettab,
        ls_return           TYPE bapiret2.
  FIELD-SYMBOLS: <ls_raw_data> TYPE any,
                 <ls_rec_type> TYPE any,
                 <ls_add_data> TYPE zaif_test_zgtp_test_raw_add_da,
                 <lt_add_data> TYPE zaif_test_zgtp_test_raw_ad_tab.

*create FEH instance for retry
  lr_feh_registration = cl_feh_registration=>s_retry( i_error_object_id = i_erro
r_object_id ).

  CREATE DATA lr_pre_input TYPE zaif_test_zgtp_test_raw_mt.
  CREATE DATA lr_post_input TYPE zaif_test_zgtp_test_raw_mt.
*retrieve data stored in FEH
  lr_feh_registration->retrieve_data(
    EXPORTING
      i_data                = i_data
    IMPORTING
      e_pre_mapping_data_ref  = lr_pre_input
      e_post_mapping_data_ref = lr_post_input
```

```
          ).

  ASSIGN lr_post_input->* TO <ls_raw_data>.
  IF <ls_raw_data> IS INITIAL.
    ASSIGN lr_pre_input->* TO <ls_raw_data>.
  ENDIF.

  ASSIGN COMPONENT 'ZGTP_TEST_RAW_MT' OF STRUCTURE <ls_raw_data> TO <ls_rec_type
>.
  ASSIGN COMPONENT 'RAW_ADD_DATA_T' OF STRUCTURE <ls_rec_type> TO <lt_add_data>.

*try to re-process the data
  LOOP AT <lt_add_data> ASSIGNING <ls_add_data>.
    ls_customer_data-custname = <ls_add_data>-add_data_key.
    ls_customer_data-street   = <ls_add_data>-add_data1.
    ls_customer_data-postcode = <ls_add_data>-add_data2.
    ls_customer_data-city     = <ls_add_data>-add_data3.
    ls_customer_data-countr   = <ls_add_data>-add_data4.
    ls_customer_data-custtype = <ls_add_data>-add_data5.
    ls_customer_data-email    = <ls_add_data>-add_data6.

    CALL FUNCTION 'BAPI_FLCUST_CREATEFROMDATA'
      EXPORTING
        customer_data = ls_customer_data
      TABLES
        return        = lt_return.
  ENDLOOP.

*check if an error occured during when creating the customer
  READ TABLE lt_return INTO ls_return WITH KEY type = 'E'.
  IF sy-subrc = 0.
    e_return_message = ls_return.
    ls_main_object-objcat  = '1'.
    ls_main_object-objtype = 'SCUSTOM'.
    ls_main_object-objkey  = ls_customer_data-custname.
    TRY.
        lr_feh_registration->collect(
            i_external_guid  = i_error_object_id
            i_single_bo      = <ls_raw_data>"ls_api_data
            i_error_category = 'PRE'
            i_main_message   = ls_return
            i_messages       = lt_return
            i_main_object    = ls_main_object
            i_pre_mapping    = 'X'
               ).

      CATCH cx_ai_system_fault.
    ENDTRY.
  ENDIF.

*update FEH
  lr_feh_registration->resolve_retry( ).

ENDMETHOD.
```

Example Implementation of Fail method

```
METHOD if_ech_action~fail.
  cl_feh_registration=>s_fail(
    EXPORTING
      i_data             = i_data
    IMPORTING
      e_execution_failed = e_execution_failed
      e_return_message   = e_return_message
        ).
ENDMETHOD.
```

Example Implementation of Finish method

```
METHOD if_ech_action~finish.
  cl_feh_registration=>s_finish(
    EXPORTING
      i_data             = i_data
    IMPORTING
      e_execution_failed = e_execution_failed
      e_return_message   = e_return_message
        ).
ENDMETHOD.
```

Display Messages Processed with SAP Application Interface Framework in ECH

A second scenario for the integration of ECH and SAP Application Interface Framework exists. In this scenario the data will be processed with the SAP Application Interface Framework.

Generate a proxy from your service interface and implement the proxy method. Within the proxy method you should call method /AIFX/CL_AIF_ACTION_CLASS=>EXECUTE_AIF_AND_ECH. For example the method's implementation could look as follows:

```
METHOD zaif_x102_ii_ech_aif_in02~test_inbound01.
  DATA: lr_root     TYPE REF TO cx_root,
        lr_previous TYPE REF TO zaif_test_cx_zgt_fault.

  SET UPDATE TASK LOCAL.
  TRY.
      /aifx/cl_aif_action_class=>execute_aif_and_ech( i_input = input ).
    CATCH cx_root INTO lr_root.
      lr_previous ?= lr_root->previous.
      RAISE EXCEPTION TYPE zaif_test_cx_zgt_fault
        EXPORTING
          standard = lr_previous->standard.
  ENDTRY.
ENDMETHOD.
```

Afterwards you have to define your interface in the SAP Application Interface Framework. In customizing for the SAP Application Interface Framework go to *Interface Development→Define Interfaces*. Select a namespace and enter an interface name and version. Furthermore, you should maintain the SAP data structure and a proxy class. After entering the proxy class the raw data structure and record type should be filled automatically. To map the source structure to the destination structure, go to customizing activity *Define Structure Mappings*. Furthermore, you should assign an action in this customizing activity. Optionally, you assign checks, value mappings and so on to the interface. For more information on interface implementation with the SAP Application Interface Framework refer to the cookbook.

Note: If you use the SIW configuration /AIFX/DEFAULT_WO_ESR, you can set the ECH indicator in the *Project Context Data*. If the ECH indicator is set the proxy class implementation will be created accordingly.

Furthermore, you have to maintain some ECH specific customizing. Therefore, go to transaction SPRO and perform following steps:

- Create an ECH business process and assign the action class as well as the notification class. To do so go to *Cross Application Components→Processes and Tools for Business Applications→Enterprise Services→Error and Conflict Handler→Define Business Process*. You have to maintain following values:
  - Component
  - Business Process
  - Action Class: for this scenario action class /AIFX/CL_AIF_ACTION_CLASS is delivered with the SAP Application Interface Framework.
  - Notification: CL_FEH_REGISTRATION
  - Business Process Description
- Create a business process for the Postprocessing Office. Go to *Cross Application Components→Processes and Tools for Business Applications→Enterprise Services→Error and Conflict Handler→Postprocessing Office→Business Process→Define Business Processes*. Maintain following fields:
  - Software Component
  - Process
  - Process Description
- Map the ECH Business Process to the PPO Business Process. Go to *Cross Application Components→Processes and Tools for Business Applications→Enterprise Services→Error and Conflict Handler→Define Business Process for Postprocessing Office*. Maintain following fields:
  - Component
  - Business Process (ECH)
  - Software Component (PPO)
  - PPO Process
- Assign the proxy class for your service operation to the ECH process. Go to *Cross Application Components→Processes and Tools for Business Applications→Enterprise Services→Error and Conflict Handler→Assign Caller to a Business Process.* Maintain following fields:
  - API Name
  - API Component
  - Call Type
  - Component Business Process

To enable the *Monitoring and Error Handling* to select the messages processed with AIF, it is necessary to define the corresponding interface engines. You can do this in Customizing activity *Specify Interface Engines*. Select your interface and maintain following engines:

- Application Engine: ECH
- Persistence Engine: ECH
- Selection Engine: AIF Index tables
- Logging Engine: ECH or AIF Application Log

If you now sent messages to your interface you should be able to see the messages in *Monitoring and Error Handling*. In case an error occurred when the message was processed the status should be *Transferred to External Application*. When you select a data message the data structure and the log messages should be displayed. Furthermore, if you select a component in Data Structure view, the content of the structure will be displayed in Data Content view. You can use the Error and Conflict Handler for editing, restarting and canceling messages.

Note: In this scenario editing, restarting and canceling messages in *Monitoring and Error* Handling is only possible after notes 1768495 and are 1769094 implemented.

Start the Error and Conflict Handler using transaction ECH_MONI_SEL. Enter selection criteria in order to be able to see the messages processed with your interface. For example, enter *Technical Creation Date*, select *Order Assignment: All Orders*. A list that fits the selection criteria is displayed. After selecting a message, the *Postprocessing Desktop* will be displayed. In order to be able to edit a message in ECH you need to be assigned as processor. Select *Process* to be assigned as processor.

To edit a message select *Message Data* in *Order Details* and double click *Change*. The payload editor should start in change mode. Change the content of the fields that require changing and select *As New Version* in the application toolbar. When you return to the post processing office you can restart the message by clicking *Repeat* in the application toolbar.

For more information about error handling with *Error and Conflict* handler, refer to the application help of *Error and Conflict Handler*.

Related Content

Cookbook for the SAP Application Interface Framework

SAP Application Interface Framework and ECH

Application Help for ECH

Copyright