

Release: 4.0

Authors: Thomas  
Elsässer,  
Verena  
Wörner, Enrico  
Mraß, Yaping  
Gao, Eric  
Hong, Gabriele  
Moll



# SAP Application Interface Framework

Cookbook

## History

Version	Status	Date
4.0	Released	10.10.2012



## Content

<b>1</b>	<b>Organization.....</b>	<b>3</b>
1.1	Project Team / Development Team .....	3
1.2	Document References.....	3
<b>2</b>	<b>Target of this Document .....</b>	<b>3</b>
<b>3</b>	<b>Architecture Overview.....</b>	<b>4</b>
<b>4</b>	<b>Interface Implementation with the SAP Application Interface Framework .....</b>	<b>6</b>
4.1	Prerequisites and Preparation .....	6
4.2	Customizing of Interfaces with the SAP Application Interface Framework .....	8
4.2.1	Namespaces .....	9
4.2.2	Interfaces .....	10
4.2.3	Interface Engines .....	12
4.2.4	Value Mapping .....	14
4.2.5	Fix Values .....	19
4.2.6	Checks .....	21
4.2.7	Actions .....	24
4.2.8	Structure Mapping .....	26
4.2.9	Interface Determination .....	33
4.2.10	Error Handling .....	34
4.2.11	Interface Variants .....	37
4.3	Implementation of Subscreen with Key Fields .....	42
4.3.1	Creating a Single Index Table .....	42
4.3.2	Implementation of Subselection Screen with Key Fields.....	43
4.3.3	Assign Subselection Screen and Single Index Table.....	43
4.3.4	Define Key Fields .....	44
4.4	BADs for the Error Handling Application Toolbars.....	45
<b>5</b>	<b>Enable Users to Monitor Interfaces.....</b>	<b>49</b>

# 1 Organization

## 1.1 Project Team / Development Team

Name	Company	Project Role(s) / Comments
Thomas Elsässer	SAP AG	Solution Architect
Yaping Gao	SAP AG	Solution Architect
Enrico Mraß	SAP AG	Solution Developer
Verena Wörner	SAP AG	Solution Developer
Gabriele Moll	SAP AG	Solution Developer
Eric Hong	SAP AG	Solution Developer
Lianyue Li	SAP AG	Solution Developer

## 1.2 Document References

[OnlineDocu]: Online Documentation including information for business users and system administrators (<http://help.sap.com/> → SAP ERP Add.Ons→ SAP Application Interface Framework)

# 2 Target of this Document

This document describes how to develop interfaces with the SAP Application Interface Framework. It can be used as a step-by-step manual.

The following chapters provide an overview of the structured approach to the implementation of the interfaces using the SAP Application Interface Framework. The features and functionality outlined in the following chapters refer to the functionality of the SAP Application Interface Framework 2.0.

The interface implementation consists of two parts. The first part is dependent on the interface technology you want to use and will not be part of this cookbook. For example, if you want to use an ABAP proxy interface the first step is to a design service interface in the SAP NetWeaver PI. The cookbook focuses on the second step, which is the implementation and Customizing of the interface in the SAP Application Interface Framework.

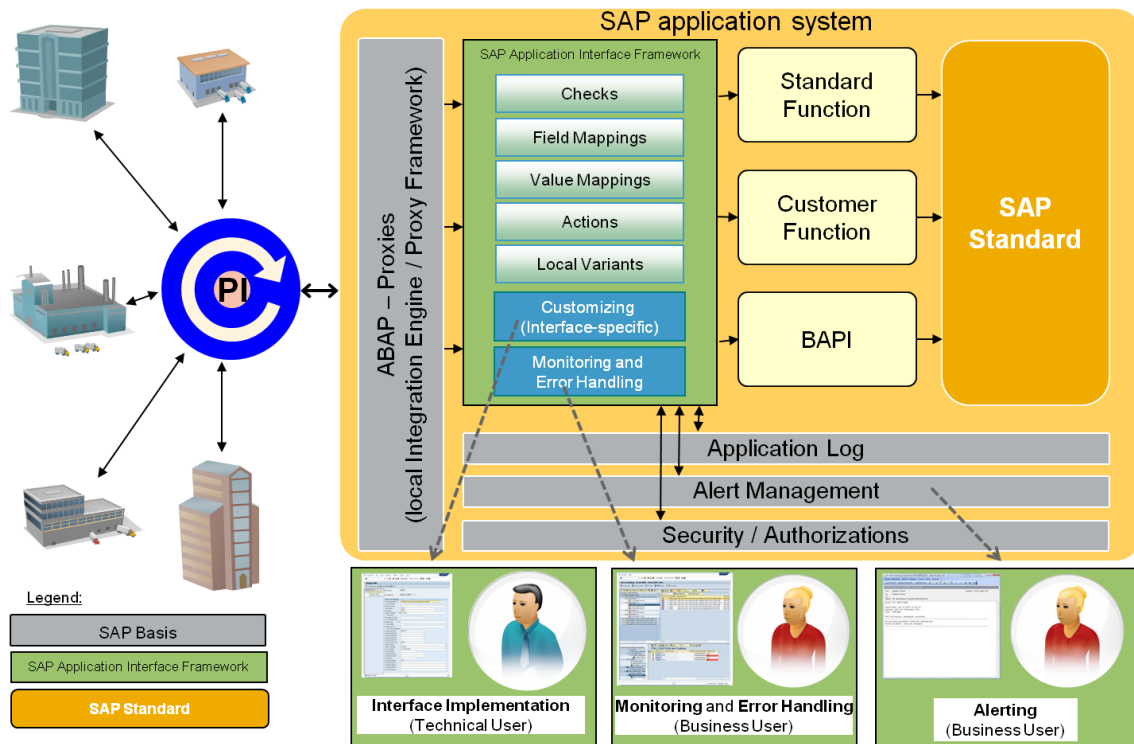
Chapter 3 gives an overview of the system architecture needed for the SAP Application Interface Framework.

Chapter 4 describes the implementation of the interface with the SAP Application Interface Framework. This chapter explains the different Customizing activities that are available.

Chapter 5 gives you a task oriented view on how to create an interface. This chapter contains a short overview of how to enable users to monitor interfaces in the *Interface Monitor*.

### 3 Architecture Overview

The SAP Application Interface Framework is located on an SAP application system (for example an ERP or CRM system). Figure 1 gives an overview of the system architecture containing the SAP Application Interface Framework. The figure displays a system architecture, where data is received from the SAP NetWeaver PI. However, the SAP Application Interface Framework supports also several IDoc scenarios as well as an own runtime and persistence layer.



**Figure 1: SAP Application Interface Framework – Architecture Overview with SAP NetWeaver PI**

Data is sent to the SAP system where the SAP Application Interface Framework is located. The SAP application system receives the data and calls the SAP Application Interface Framework.

In the SAP Application Interface Framework, received data is processed. The data is transformed from the raw data structure (source) to the SAP data structure (destination). During this transformation, checks can be executed. If a check does not succeed, further processing of the message in the SAP Application Interface Framework stops. Value mappings are executed to fill the SAP data structure. After all the data is mapped from the raw data structure to the SAP data structure, actions are called. Actions are used to execute the business logic of the interface in the SAP Application Interface Framework. Within an action, you can call an SAP standard function, your own customer-specific function, or a BAPI. The mapping of data, the executed checks, value mappings, and actions depend on the Customizing of the interface (see chapter 4), which is done by the interface developer.

Errors that occur while processing a message in the SAP Application Interface Framework, for example, if a check does not succeed or if a value mapping fails, are written to the application log. Application log messages are loaded into the *Monitoring and Error Handling* transaction

and support the business user in solving errors. Additionally, you can configure the system to send alerts to users, if certain errors occur during message processing.

The process described above is concerned with how data is received by the SAP Application Interface Framework. It is also possible to send data using the SAP Application Interface Framework. Therefore, an outbound interface is needed. To start the processing, a report, function module, or batch job is necessary to call the SAP Application Interface Framework. In the SAP Application Interface Framework, data is mapped from the SAP data structure (source) to the raw data structure (destination). Customized checks and mappings are executed and errors that occur are written to the application log. If the processing in the SAP Application Interface Framework is successful, the data will be send to the receiver.

## 4 Interface Implementation with the SAP Application Interface Framework

### 4.1 Prerequisites and Preparation

Depending on the interface technology you use different prerequisites apply and different steps have to be executed in preparation to developing an interface with the SAP Application Interface Framework. This chapter will give a general overview of the steps required before an interface can be developed with the SAP Application Interface Framework.

Some of the steps required as preparation are similar for different technologies

- **Find or create BAPI(s) and function module(s):** Usually, the business logic in the backend is executed by methods (for example, of business objects), BAPIs, or function modules. These could be predefined by SAP or implemented specifically for your company. In order to call these objects in an action of the SAP Application Interface Framework later on (see chapter 4.2.7), you need your data to be in the format or structure they expect. To achieve this, you have to build your SAP data structure to accommodate the required data and their expected format.
- **Create SAP (target / source) data structure:** This data structure is required for passing values during the processing of the function modules. The structure can consist of a hierarchy of substructures and tables.

For an outbound interface, the SAP data structure constitutes the source structure. Its components are usually defined by the format of the data stored in the system. When defining the SAP data structure, you collect all the data and corresponding data types you want to send to an external system and include them in the SAP data structure. During the runtime of the SAP Application Interface Framework, the data of the SAP data structure is mapped to the destination structure. Furthermore, the SAP data structure is available in several function modules as the source structure.

For an inbound interface, the SAP data structure constitutes the destination structure. Its components are usually defined by the class methods or function modules you want to execute in the system. When defining the SAP data structure, you collect all the necessary input data and corresponding data types and include them in the SAP data structure. During the runtime of the SAP Application Interface Framework, the data of the source structure is mapped to the SAP data structure. Furthermore, the SAP data structure is available in several function modules as the destination structure.

#### Example of a source structure:

- ZGTP\_AIF Structure
  - ZAIF\_HEADER\_AND\_ITEM1\_T Table Type
    - ZAIF\_HEADER Structure
    - ZAIF\_ITEM\_T Table Type
  - ZAIF\_HEADER\_AND\_ITEM2\_T Table Type
    - ZAIF\_HEADER2 Structure
    - ZAIF\_ITEM\_T2 Table Type
  - ZAIF\_ADD\_DATA\_T Table Type

#### 4.1.1.1 Proxy Interfaces

A service interface has been created in the Enterprise Service Repository and a proxy class and structure has been generated for the service interface in your backend system. After generating the proxy class and structure, you need to implement the proxy method to call the SAP Application Interface Framework. Within this method, static method `PROCESS` of class `/AIF/ENABLER_PROXY` needs to be called. The method passes the data to the SAP Application Interface Framework and starts processing data. To implement the method, double click on the generated proxy class. Select the proxy method and implement it like in the following coding:

```
METHOD <interface>~<method>.
  /aif/cl_enabler_proxy=>process_message (
    is_input          = input
    iv_exception_classname = 'ZCX_FAULT'
  ).
ENDMETHOD.
```

**Note:** You have to exchange the exception class name one that belongs to your service interface. To get the name, access the proxy class you generated and choose *Exceptions*. Furthermore, the name of the method differs.

#### 4.1.1.2 IDoc Interfaces

- Decided on scenario: To monitor IDocs with the SAP Application Interface Framework four scenarios exist. Decide which scenario you want to use, this will depend on your requirements and business needs. Following Scenarios exist:
  - Monitoring of existing IDocs in Monitoring and Error Handling
  - Process IDoc Using AIF; Call IDoc Function Module in Action
  - Process IDoc Using AIF; Call BAPI in Action
  - Process IDoc Using ALE; Wirthe Index Tables with AIF Enabler
- Depending on the scenario you select you might have to create a new IDoc basic type and message type.

#### 4.1.1.3 XML Interfaces

If you want to use the XML persistence and runtime delivered with the SAP Application Interface Framework, you have to create, for example a report or function module where you trigger processing in the runtime. Depending on the data you want to process you have to define a structure for your interface in the SAP Application Interface Framework.

#### 4.1.1.4 Interfaces for the Error and Conflict Handler

For an ECH interface you need to define an interface in the Enterprise Service Repository and generate a proxy class in your backend system. How the implementation of the proxy method will look like depends on the integration scenario you select. Two scenarios exist for the integration for the Error and Conflict Handler and the SAP Application Interface Framework:

- Display messages of ECH in Monitoring and Error Handling
- Display Messages Processed with SAP Application Interface Framework in ECH

## 4.2 Customizing of Interfaces with the SAP Application Interface Framework

Customizing for the SAP Application Interface Framework can be reached via transaction code /AIF/CUST or via the SAP Easy Access menu by choosing *Cross-Application Components* → *SAP Application Interface Framework* → *Interface Development* → *Customizing*. The Customizing tree of the SAP Application Interface Framework is displayed in Figure 2. **Note:** Some of the activities displayed in the customizing tree are only available if component AIFX is installed.



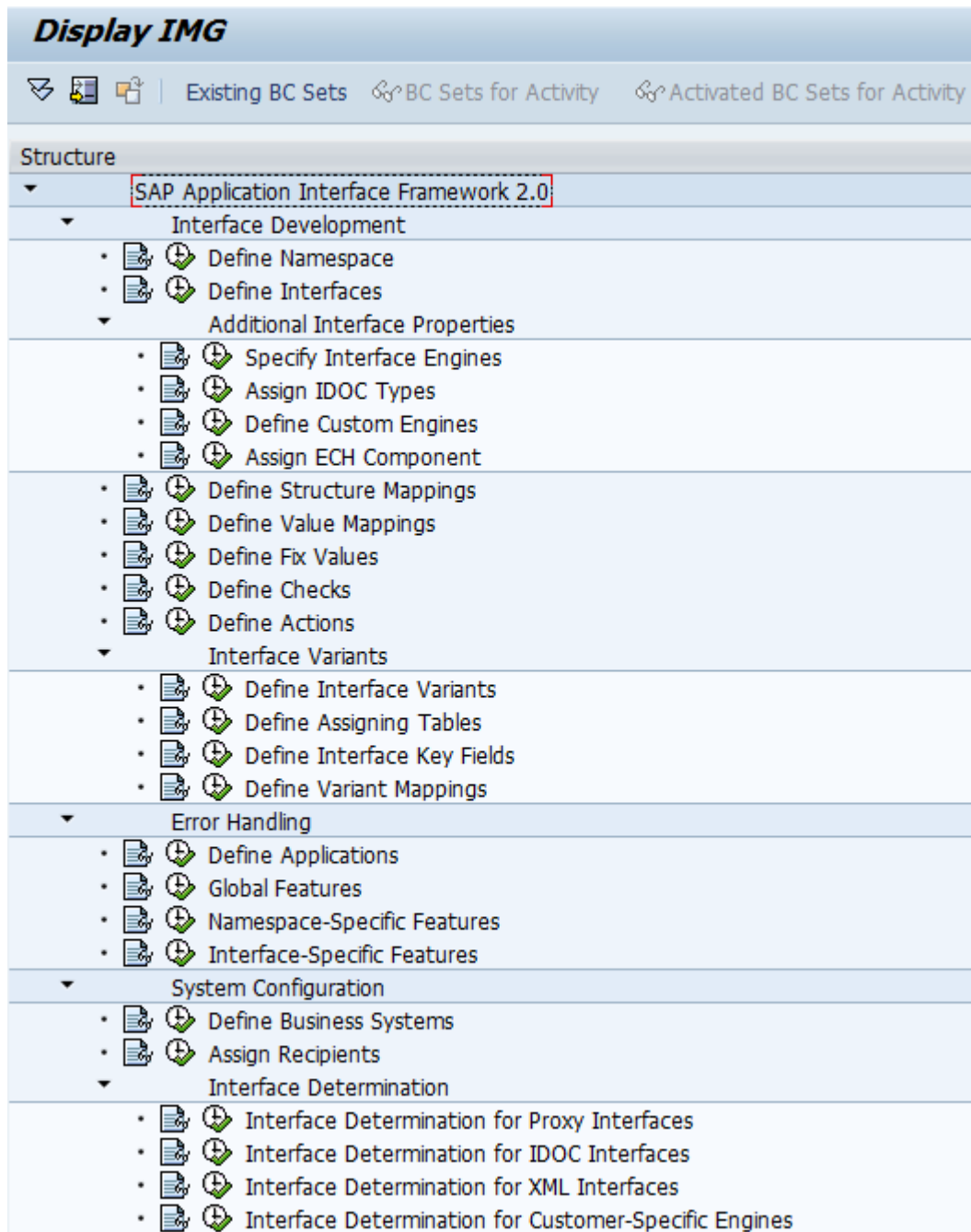


Figure 2: SAP Application Interface Framework – Customizing

#### 4.2.1 Namespaces

In the SAP Application Interface Framework, namespaces are used for the logical structuring of objects and configurations (for example, interfaces, mappings, and actions). Furthermore, the locking mechanism for changing configurations is handled at the namespace level for the Customizing activities *Define Interface* (see chapter 4.2.2), *Define Value Mappings* (see chap-

ter 4.2.4), *Define Fix Values* (see chapter 4.2.5), *Define Checks* (see chapter 4.2.6), and *Define Actions* (see chapter 4.2.7). For *Structure Mappings* (see chapter 4.2.8), locking is done by specifying the namespace, the interface name, and version.

You can also set authorizations for Customizing and error handling based on the namespace (see Master Guide for SAP Application Interface Framework [MasterG] in chapter 1.2 Document References).

To create a new namespace, choose *Define Namespace* in Customizing. Choose *New Entries* and insert a name and a description for the namespace.

## 4.2.2 Interfaces

After defining a namespace, an interface can be created in the SAP Application Interface Framework.

To define an interface, choose *Define Interface* in Customizing and enter the namespace you want to use. Choose *New Entries* to define a new interface. Figure 3 shows the screen of the interface definition. You have to enter an interface name and a version.

**Change View "Define Interfaces": Details**

New Entries

Namespace: X102\_3 Documentation

Interface Name: ZGTP

Interface Version: 1

**Define Interfaces**

Description: Simple AIF interface

SAP Data Structure: ZGTP\_AIF

Raw Data Structure: ZZGTP\_TEST\_RAW\_MT

Record Type in Raw Structure: ZGTP\_TEST\_RAW\_MT

☐ Move Corresponding Structures

Check Function Module:

Init Function Before Mapping:

Init Function Before Processing:

Separate Commit: No Separate Commit

Lifetime of Application Log:

☐ Test Mode

Proxy Class Inbound: ZCL\_ZGTP\_RAW\_SI\_TEST

Proxy Class Outbound:

Field for the Sending System:

Status Handling:

End Date:

☐ Automatic Client Control

**Figure 3: Interface Definition**

The following fields can be filled:

- *Interface Name*: Defines the name of an interface in the SAP Application Interface Framework. An interface is the technical representation of a system interface. It is

the connection point between the application data (SAP data structure) and the data in the external format (raw data structure).

- *Interface Version*: Defines a version of an interface in the SAP Application Interface Framework that is used for processing a data message. A new version of an interface may vary from the original interface, for example, in the internal processing of data, but it should still achieve the same purpose.
- *SAP Data Structure*: Refers to the SAP data structure, for example, ZGTP\_AIF, as described in chapter **Error! Reference source not found.** For an inbound interface, the SAP data structure is the destination structure. Its components are defined by the functions you want to execute in the system. The received data is mapped to the SAP data structure. For outbound interfaces, the SAP data structure constitutes the source structure. The data is mapped to the structure of the outbound interface.
- *Raw Data Structure*: Refers to the structure that has been created during proxy generation. This structure constitutes the source structure for inbound interfaces or the destination structure for outbound interfaces. The raw data structure is required to contain only one substructure (besides the automatically generated controller table) that can be used as a base structure in the SAP Application Interface Framework. This substructure is used as a record type.
- *Record Type in Raw structure*: The record type defines the main component of the raw data structure. The *Record type*, as well as the *Raw Data Structure*, is populated automatically by the SAP Application Interface Framework when the proxy class is inserted.
- *Move Corresponding Structures*: If the source and destination structures are partly or completely identical, you can use this indicator to automatically map the data from source to destination structure 1:1. Mapping needs to be defined only for the unequal parts of the structures.
- *Check Function Module*: You can implement your own function module to check the data in the destination structure. The function module is executed after the mapping and before the execution of the actions. **Note** that the function module should have the same signature as template function module /AIF/FILE\_TEMPL\_CHECK\_MAPPING.
- *Init Function Before Mapping*: You can enter the name of a function module that manipulates the source and /or destination structure before the mapping in the SAP Application Interface Framework. **Note** that the function module should have the same signature as template function module /AIF/FILE\_TEMPL\_INIT\_MAPPING.
- *Init Function Before Processing*: You can enter the name of a function module to manipulate data in the destination structure after the mapping but before the processing of the data. **Note** that the function module should have the same signature as template function module /AIF/FILE\_TEMPL\_INIT\_PROCESS. Depending on what data you want to manipulate, it is recommended that you specify the type of changing parameters data, curr\_line, and/or curr\_data in the newly created function module.
- *Separate Commit*: Specifies if a separate COMMIT WORK is executed after message processing by the SAP Application Interface Framework. **Note** that for outbound interfaces, a separate commit is necessary in order to trigger sending the message.

- *Lifetime of Application Log*: Defines the number of days after which the application log messages reaches the expiry date. You can delete application log messages with transaction SLG2. It is possible to set an indicator to delete all messages that have reached the expiry date. **Note** that you should be careful about deleting application logs, especially in the productive system.
- *Test Mode*: Set this indicator to process this interface in test mode that is if you do not want data to be committed to the database. You can use the test mode, for instance, to check if the Customizing of your interface is correct. In some function modules you plug into the SAP Application Interface Framework, this indicator is available as an import parameter. This way, you can implement separate logic for interfaces in test mode.
- *Proxy Class Inbound*: This field defines the name of a generated proxy class in the Proxy Framework. This field must only be filled for inbound interfaces. You should enter the name of the proxy created in chapter **Error! Reference source not found..**
- *Proxy Class Outbound*: This field is used to define the name of a proxy class of an outbound interface.
- *Field for the Sending System*: If available for this interface, enter the path to the field in the raw data structure containing the sending system name. It is a commonly used case to differentiate the mapping or processing based on the sending system. For this reason, the system field is available in certain places within the SAP Application Interface Framework, for example, in value mappings (see chapter 4.2.4).

If you only want to map the data from the source structure to the destination structure or if you want to reuse already existing actions, value mappings, and so on, you can go directly to chapter 4.2.8. If you want to create a value mapping (see chapter 4.2.4), a fix value (see chapter 4.2.5), checks (see chapter 4.2.6), or action (see chapter 4.2.7), see the corresponding chapters.

### 4.2.3 Interface Engines

Depending on the interface technology you are using you have to define the engines that should be used in *Monitoring and Error Handling* in order to handle data messages. To define the engines needed for your interface go to *Interface Development*→*Additional Interface Properties*→*Specify Interface Engines*. After selecting your interface you can maintain four different engine types:

- *Application Engine*: Defines how messages should be handled in *Monitoring and Error Handling*. The application engine is, for example responsible for restarting and canceling messages
- *Persistence Engine*: Different interface technologies use different persistence layers to store the data received or sent. The persistence engine is responsible for selecting, updating and locking the data content of a message.
- *Selection Engine*: The selection engine defines which messages should be displayed in *Monitoring and Error Handling*. The selection engine will select the messages based on the selection criteria of the selection screen.
- *Logging Engine*: The logging engine is responsible for handling the log messages written during message processing.

Several engines are delivered with the SAP Application Interface Framework.

© 2012 SAP AG Dietmar-Hopp- Allee 16 D-69190 Walldorf	Title: SAP Application Interface Framework Version: 4.0 Date: 10.10.2012	Page 12 of 49
--	--	---------------

- Application Engine
  - *Proxy*: Proxy is the default application engine. You have to select this application engine if your interface processes data of an ABAP proxy
  - *IDoc*: Select this application engine, if your interface handles data of the ALE Runtime
  - *XML*: Is responsible for handling messages, that use the runtime and persistence delivered with the SAP Application Interface Framework
  - *File*: In case this is only a test interface and you trigger processing with transaction /AIF/IFTEST you can maintain this application engine
  - *ECH*: Handles asynchronous data messages received via a proxy from SAP NetWeaver PI but processed by the Error and Conflict Handler application. Each message will have a corresponding postprocessing order (only available with AIFX and NetWeaver 7.31).
  - *Custom Specific*: You can use your own custom specific engine
- Persistence Engine
  - *Proxy*: Proxy is the default persistence engine. You choose this engine if your interface processes data with an ABAP proxy.
  - *IDoc*: Select the IDoc persistence engine to handle the data content (data records) of an IDoc interface.
  - *XML*: Choose this engine for interfaces that store data in the SAP Application Interface Framework's persistence
  - *File*: If you have an interface that is only handling test files processed with /AIF/IFTEST, maintain this persistence engine
  - *ECH*: Handles the data content of messages processed with the Error and Conflict Handler application. Includes load data and save data as new version (only available in AIFX and NetWeaver 7.31)
  - *Custom Specific*: You can use your own custom specific engine
- Selection Engine
  - *AIF Index Tables*: AIF Index tables is the default selection engine. In case the SAP Application Interface Framework was used for processing the data or index table entries were written using the enabler, you can use this selection engine
  - *IDoc Control Records*: This selection engine handles IDocs that have been processed without the SAP Application Interface Framework. Data is selected from the IDoc control records.
  - *ECH*: This selection engine handles messages that have been processed by Error and Conflict Handler application instead of SAP Application Interface Framework. Data is selected from Postprocessing order related tables (only available in AIFX and NetWeaver 7.31).
  - *Custom Specific*: You can use your own custom specific engine
- Logging Engine

- *AIF Application Log*: This is the default logging engine. In case the data is processed with the SAP Application Interface Framework an application log is written. In order to display these logs in *Monitoring and Error Handling* you have to choose this logging engine.
- *IDoc Status Records*: the SAP Application Interface Framework is not involved in processing of an IDoc, the IDoc status records can be displayed as log messages in *Monitoring and Error Handling*. When using the enabler for IDocs, the enabler will not write any application log entries if the IDoc status records logging engine is used.
- *ECH*: Handles messages for a postprocessing order raised during Error and Conflict Handler application processing (only available with AIFX and NetWeaver 7.31)
- *Custom Specific*: You can use your own custom specific engine

Additionally, to the engines delivered with the SAP Application Interface Framework, you can create your own customer specific engines. Those engines have to be maintained in customizing activity *Interface Development* → *Additional Interface Properties* → *Define Custom Engines*.

If you want to use a custom engine with your interface you have to select *Custom Specific*. After you have defined your custom specific engines you can enter the engines in *Specify Interface Engines*. You have to enter the *Namespace* of your engine and the name of your *Customer Engine*. The custom engines that you maintain can be part of different namespaces.

#### 4.2.4 Value Mapping

Value mappings are used to derive one value in the destination structure from up to five values in the source structure and can be optionally enriched with further data from the backend system. There are different ways in which a value mapping can be defined. You can specify a function module in the field mapping (see chapter 4.2.8.4) or you can define a named value mapping. Since named value mappings are treated as objects in the SAP Application Interface Framework, they are easier to find and, therefore, easier to reuse. Value mappings are grouped by namespace. Furthermore, you can take advantage of the processing logic of the different value mapping types. Creating a named value mapping is explained in this chapter.

To create a value mapping, choose *Define Value Mappings* in Customizing for the SAP Application Interface Framework and enter the namespace for which you want to create the value mapping. Choose *New Entries* and enter a name for the value mapping and a description. You can define a conversion exit (for example, ALPHA) and you can specify the direction of the conversion. The conversion exit is executed before the mapping. It is possible to define a default value that is used if no other value can be found. It is also possible to define a conversion exit after mapping. Figure 4 shows the input screen for the definition of a value mapping.

**New Entries: Details of Added Entries**

Dialog Structure

- Define Value Mappings
  - Define Values
  - Define Multi Values

Namespace: X102\_3 Documentation

Value Mapping: TEST\_MAPPING

**Define Value Mappings**

Value Mapping Description	
Data Element for Conversion Before	
Conversion Exit Before Mapping	
Direction of Conversion Exit	
Table Name	
Field Name	
Where Condition for Select Statement	
<input type="checkbox"/> Do Not Buffer SELECT Statement	
Value Mapping Function Module	
<input type="checkbox"/> Use Default Value	
Default Value	
Data Element for Conversion After	
Conversion Exit After Mapping	
Direction of Conversion Exit	
<input type="checkbox"/> Value Mapping Is Not System Dependent	
Data Element for EXT1	
Data Element for EXT2	
Data Element for EXT3	
Data Element for EXT4	
Data Element for EXT5	
Data Element for INT	
Number of External Values	0
Single or Multiple Value Mapping	Both
Customizing or Master Data	Both
Alternative Transaction	
View/Table Name	
View Cluster Name	
Error Msg. Class	
Error Msg. Number	
Message Variable 1 Definition	
Message Variable 2 Definition	
Message Variable 3 Definition	
Message Variable 4 Definition	
Namespace	
Check	
<input type="checkbox"/> Check Before Mapping	
<input type="checkbox"/> Use Validity Period Number	
<input type="checkbox"/> Enter Validity Period Dates Directly	

**Figure 4: Define Value Mappings**

There are different types of value mappings. The different value mapping types have the following processing logic:

1. The SAP Application Interface Framework internal value mapping tables are checked (see chapter 4.2.4.8).
2. Dynamic select statement (see chapter 4.2.4.1; Only executed if step 1 fails)
3. Value mapping function (see chapter 4.2.4.2)



The function module is always executed. Values derived from internal mapping tables or select statements can be overwritten.

4. The default value is used if no value is found.

#### 4.2.4.1 Define Table to Select Value

You can use a database table as source for the values that should be mapped to the destination structure. Therefore, the following fields have to be filled:

- *Table Name:* Enter the name of the database table you want to use as source for the values in the destination structure.
- *Field Name:* Enter the name of the field in the database table containing the target value.
- *Where Condition for Select Statement:* Enter a where condition to define which entries should be selected. You can define up to five placeholders (\$1, \$2, \$3, \$4, and \$5). Those placeholders are replaced at runtime by the values that are contained in the fieldnames during field mapping. The five fields are defined in the field mapping where the value mapping is used (see chapter 4.2.8.4). By using placeholders, value mappings remain independent from the interfaces using them.

**Example (getting the carrier ID for a given connection ID in SAP's flight agency data model):**

**Table Name:** SPFLI

**Field Name:** CARRID

**Where Condition:** CONNID = '\$1'

#### 4.2.4.2 Define Value Mapping Function

It is possible to define a mapping function. Enter the name of the function module you want to use for mapping. **Note** that the function module should have the same signature as the template function module /AIF/FILE\_TEMPL\_VALMAPPING.

#### 4.2.4.3 Alternative Transaction

If you map values using a database table (see chapter 4.2.4.1), you can maintain your own value mapping transaction and enter its transaction code here. **Note** that the transaction must correspond to the defined table in the value mapping.

If you maintain a transaction and an error results from a missing value mapping, an additional *Value Mapping* button is displayed in the *Monitoring and Error Handling* transaction. Selecting this button enables the user to maintain the missing values in the given value mapping transaction.

#### Example

Create a database table and a maintenance view for the table. Furthermore, create a transaction for the maintenance view. Enter the name of the transaction for the maintenance view in the *Alt. transaction* field.

#### 4.2.4.4 Alternative View / Table / View Cluster



Besides maintaining an alternative transaction, you can specify an alternative maintenance view to be used instead of the normal value mapping table. Furthermore, a view cluster can be specified to be used instead of the standard value mapping view. A *Value Mapping* button is displayed if an error occurs due to a missing value mapping.

#### 4.2.4.5 Value Mapping-Specific Error Message

If an error occurs due to a value mapping, you might want to write your own message to the application log instead of the more generic standard message the SAP Application Interface Framework stores in the application log. Therefore, enter an *Error Msg. Class* and an *Error Msg. Number* in the corresponding fields. If the message uses placeholders, you can replace them at runtime by entering the corresponding value in the *Message Variable 1 Definition* to *Message Variable 4 Definition* fields.

#### 4.2.4.6 Define Values

It is possible to create an SAP Application Interface Framework internal value mapping table. Choose *Define Values* in the tree on the left hand side. After choosing *New Entries*, it is possible to enter several external values and corresponding internal values. External values are the values contained in the source structure and internal values are the values contained in the destination structure after the mapping. Furthermore, a business system can be maintained to make it possible to make the mapping system-dependent.

##### Example

External Value	Internal Value	Business System
TEST1	TE01	SYS1
TEST1	TE03	SYS2

If business system SYS1 is used and the value in the source structure is TEST1, the value of the field in the destination structure is TE01. If business system SYS2 is used and the value in the source structure is TEST1, the value in the destination structure is TE03.

#### 4.2.4.7 Define Multiple Values

It is possible to define multiple external values. Up to five fields can be specified as data source in field mapping (see chapter 4.2.8.4). The internal value is determined by the combination of values in the source fields. Wildcards (\*) can be used.

##### Example

External Value	External Value	External Value	Internal Value
AKEY1	TEST	01	INT_1
AKEY2	TEST	01	INT_2
AKEY1	TEST	*	INT_3

In this example, INT\_1 is used if the values in the source structure fit to AKEY1, TEST, and 01. If the values of the source structure contain AKEY1 and TEST and a third external value anything except 01, the internal value INT\_3 is used in the destination structure.

#### 4.2.4.8 Maintain Value Mapping via Transaction /AIF/VMAP



To maintain a value mapping, there is an alternative way using transaction /AIF/VMAP. This kind of value mapping enables you to maintain single and multiple value mappings. Furthermore, it is possible to maintain different values for different sending systems. If you maintain data elements for the fields, you get input help when maintaining values in transaction /AIF/VMAP. Moreover, if a value mapping error occurs, a *Value Mapping* button is displayed in the *Monitoring and Error Handling* transaction. This button enables a business user to navigate to the value mapping and maintain the missing values. It is possible to decide if the user is able to change the value mapping in the productive system. To use it, it is necessary to maintain some settings in Customizing as follows:

- *Value Mapping Is Not System Dependent*: If this indicator is set, it is not possible to define a system dependent value mapping later on in the value mapping transaction.
- *Data Element for EXT1 to Data Element for EXT5*: Optionally maintain the data elements used in the value mapping transaction for the external values. When doing so, input help is available for the fields in the value mapping transaction.
- *Data Element INT*: Optionally maintain the data element of the internal value used in the value mapping transaction. This makes the input help for this field available in the value mapping transaction.
- *Number of External Values*: Define the number of external values that is used for the mapping. This number field is only applicable if you use value mapping type *M* (multiple value mapping).
- *Single or Multiple Value Mapping*: Defines the type of the value mapping. If the type is single, exactly one external value is mapped to one internal value. If multiple is selected, up to five external values are mapped to one internal value. The number of external values needs to be defined in the *External values* field. If *NONE* is selected, values maintained in a value mapping table are ignored.
- *Customizing or Master Data*: If *Customizing* is selected, the entries of the value mapping are saved in a Customizing task and can be transported to the productive system. In this case, it is not possible to make changes to the data directly in the productive system. If *Master* is selected, changing data is possible in the productive system. As default *both* is maintained. However, in order to use the value mapping transaction, *Customizing* or *Master* has to be maintained.











After saving, the value mapping transaction /AIF/VMAP can be called. On the input screen, the *Namespace* and the name of the *Value Mapping* can be entered. The *Database table type* and the *Value mapping type* are loaded automatically. Furthermore, you can define a *Sending System*. Defining a sending system allows you to define system-dependent value mappings.

To maintain value mappings, press F8 or select the *Execute* button. Make sure that you are in change mode. To maintain values, choose *Append* and select the number of lines you want to enter. If a value help is maintained for the data elements you defined before in Customizing, you can use the entry help to maintain values.

**Maintenance of Value Mappings**

Namespace: ZGTP      GTP Test Namespace  
 Value Mapping Name: TEST\_MAPPING  
 Database Type: C      Customizing  
 Value Mapping Type: M      Multiple value mapping  
 Sending System:

	Index	Key1	Key1	ExtValue	IntValue
	1	AKEY2	TEST	EX1	INT_1
	2	AKEY5	TEST	EX2	INT_2

**Figure 5: Maintenance of Value Mappings in Transaction /AIF/VMAP**

**Note:** If you have maintained values in the value mapping tables in the Customizing activity *Define Value Mappings* and if you have selected database type *Customizing*, you can see those entries in transaction /AIF/VMAP and vice versa. Value mappings with database type *Master* can only be maintained in transaction /AIF/VMAP.

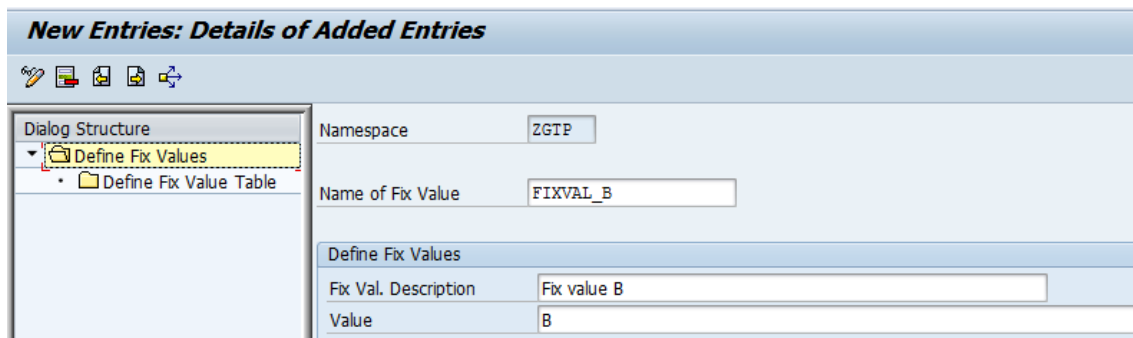
## 4.2.5 Fix Values

You can use fix values if you want a certain field in the destination structure to always have the same value.

There are two possibilities to define a fix value. The first possibility is to enter a fix value directly into the structure mapping (see chapter 4.2.8). The second possibility is to create a named fix value, which can be assigned to a field in the structure mapping. In most cases, it is recommended to use a named fix value. Named fix values can be reused in different structure mappings. If the value of the fix value changes, only the named fix value needs to be maintained centrally.

### 4.2.5.1 Named Fix Value

To create a named fix value, in Customizing for the SAP Application Interface Framework, choose *Define Fix Values* and enter a namespace. Choose *New Entries*, enter a name, a description, and the value of the fix value (see Figure 6). The fix values that are defined here are reusable and can be maintained and changed centrally.



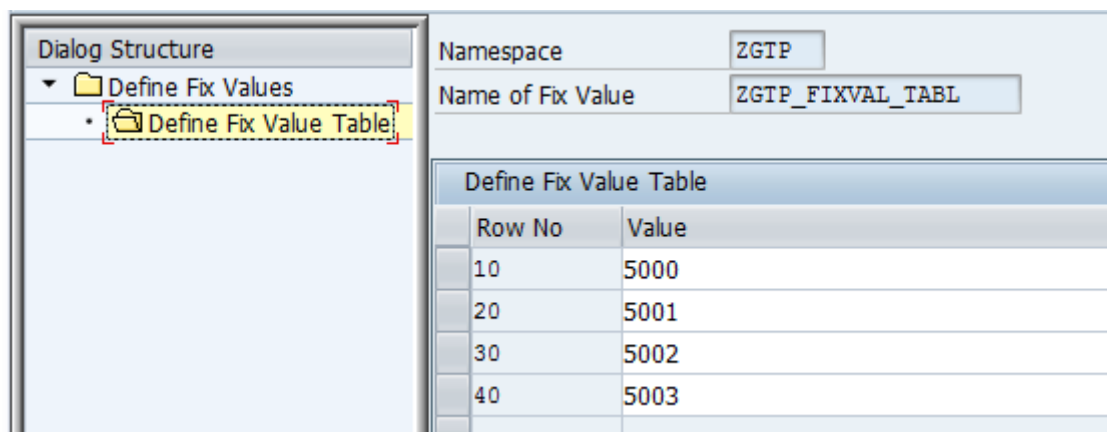
Define Fix Values	
Fix Val. Description	Fix value B
Value	B

**Figure 6: Define Fix Values**

#### 4.2.5.2 Define Fix Value Table

It is possible to define several values for one fix value in a table.

To define a fix value table, choose *Define Fix Value Table*. A new entry needs to be created and a name and a description need to be entered. A list of values can be defined as depicted in Figure 7.









Define Fix Value Table	
Row No	Value
10	5000
20	5001
30	5002
40	5003

**Figure 7: Fix Value Table**

The value can be read in a function module used in the Customizing activity *Define Structure Mapping* → *Assign Destination Structure* (see Figure 8). In the structure mapping, a source line can be expanded to a list. One line is added to the destination structure for every value in the fix value table.

**Change View "Assign Destination Structures": Details**

New Entries      

<b>Dialog Structure</b>	Namespace	ZGTP
▼ Select Source Structures	Interface Name	ZGTP_FIXV
▼ Assign Destination Structures	Interface Version	1
• Assign Checks	Source Structure	RAW_GLOBAL
• Define Fix Values	No of Struct.Mapping	10
▼ Define Field Mappings	<b>Assign Destination Structures</b>	
• Define Conditions	Dest. Structure	HEADER AND ITEM1 T
• Assign Actions	Func. before Mapping	ZGTP_STRUC_MAP_HEADER
	Func. after Mapping	
	<input type="checkbox"/> Move Corr. Fields	
	<input type="checkbox"/> Indirect Mapping	

**Figure 8: Assign Function Module for Fix Value Table**

At the beginning of the mapping of the source structure to the destination structure, the function module is called. In this function module, the list of fix values is read. One line is created for every fix value and added to the destination structure.

The result is that one line of the source structure is mapped to as many lines as there are fix values defined in the fix value list. Below, you can see an example of code where a line is added to the destination structure for every fix value.

```
DATA: lv_text TYPE symsgv.
DATA: ls_fix TYPE /aif/t_tfix.
DATA: lt_fix TYPE /aif/t_tfix_t.

CLEAR append_flag.
SELECT * FROM /aif/t_tfix INTO TABLE lt_fix
  WHERE ns = c_ns
  AND fixvaluename = c_fixvaluename.
IF sy-subrc = 0.
  LOOP AT lt_fix INTO ls_fix.
    dest_line-aif_header-header_key = ls_fix-fieldvalue.
    APPEND dest_line TO dest_table.
  ENDLOOP.
ENDIF.
```

#### 4.2.6 Checks

There are several places where checks can be defined. First of all, you can define a check function module in the interface definition (see chapter 4.2.2 ).

In Customizing for the SAP Application Interface Framework, checks can be created by choosing *Define Checks* and entering the namespace for which you want to create the check. Choose *New Entries* to create a new check. Enter the name and the description of the new check. An error message, which is written to the application log if the check fails, can be defined. Up to four variables can be defined to be used in the message. You can enter either constant values or placeholders that are replaced at runtime.

#### Examples for variables in a message:

- Replace a variable with a field value from the source or destination structure: enter \$1, \$2, \$3, \$4, or \$5.
- Replace a variable with the sending system: enter \$SENDINGSYSTEM
- Replace a variable with a field in destination structure: enter @PATH-TO-FIELD
- Replace a variable with a field in the source structure: enter PATH-TO-FIELD

<b>Example Message</b>	Value &1 has wrong format for field &2
<b>Variable Definition</b>	\$1
	@FIELD_DATE
<b>Example Message Written to Application Log if Check Fails</b>	Value ABC has wrong format for field date

**Note:** Entering a path to a field makes the check dependent on the structure of the message for which this check is called. For easier reuse, use the variable placeholders \$1 to \$5 where possible.

To define the actual check, choose *Define Single Checks*. Multiple single checks can be maintained for one check. Those checks are executed in the order of the number that has to be maintained. **Multiple single checks are linked with an OR condition.** In the case of multiple single checks, a single check is only executed if the single check(s) before was unsuccessful. Figure 9 shows the input screen for defining a single check.

Namespace	X102_3
Check	TEST_CHECK
Number of the Check	1

Define Single Checks	
Check Description	
Field Check	No Check
Allowed Characters	
Operator	
Pattern for Field Check	
Table Name	
Where Condition for Select Statement	
<input type="checkbox"/> Do Not Buffer SELECT Statement	
<input type="checkbox"/> Check Existence in Table	
Field Name for Comparison	
Operator	
Pattern for Field Check	
Check Function Module	
Success Msg. Class	
Success Msg. Number	
Variable Definition	
Variable Definition	
Variable Definition	
Variable Definition	

**Figure 9: Single Check Definition**

There are several ways to define a check:

- *Field Check*: A check can be selected from the drop down box. Only if the field value fits the selected criteria, is the check successful. The following options can be selected:
  - Only Characters a-z
  - Only Characters a-z, A-Z
  - Alphanumeric with special characters and German Umlauts
  - Not Empty
  - Empty
  - Only Characters A-Z
  - Numeric (Comma/Negative)
  - Numeric (Dot/Negative)
  - Numeric or Empty
  - Numeric (Integer)
  - Empty or '0'

- **Allowed Characters:** This field is used to define a set of characters that are allowed as field values. For example, only capital letters and some special characters (for example, !, ?, and &) could be allowed by entering ABCDEFGHIJKLMNOPQRSTUVWXYZ!?!&.
- **Pattern for Field Check:** If you want a field to contain a special pattern, you can use the pattern field check. The field has to be used in conjunction with the *Operator* field. There are several operators you can select from:
  - Contains Any Character
  - Contains Patterns
  - Equal
  - Matches (regular expression)
  - Not Contains Any Character
  - Not Contains Pattern
  - Not Equal

**Example:** You want a field to contain either the value 'AA' or 'BB'. Select operator 'Matches' and enter *AA|BB* in pattern field check.

- **Table Name:** You can define a table to check if a value is allowed. A source for the check can either be a database table or a table of the source or destination structure depending on the context. To use a table check, the *Table Name* and "*Where*" condition fields need to be filled. In *Where Condition for Select Statement*, you can use placeholders \$1, \$2, \$3, \$4, and \$5 that are replaced at runtime by the value of the fields defined in structure mapping (see chapter 4.2.8.2).
- **Fieldname for Comparison:** The fieldname comparison is used in conjunction with the table check. It is used to check if the results of the table selection fit the criteria defined in the fieldname comparison. *Fieldname comparison* contains the database field to be checked. The *Operator* and *Pattern Field Check* fields define which values are allowed. However, **note** that the fieldname comparison is only executed if the *Check existence* indicator is not set.
- **Check Function Module:** A function module can be defined for the check. **Note** that the function module should have the same signature as the template /AIF/FILE\_TEMPL\_CHECK.
- **Success Message:** It is possible to define a message that is written to the application log if the check was executed successfully. Up to four variables, whose values replace placeholders during runtime, can be defined.

## 4.2.7 Actions

Actions define the processing steps of messages in the SAP Application Interfaces Framework. They are responsible for processing business logic at runtime. An action consists of at least one function module call.

To create a new action, choose *Define Actions* and enter the namespace the action should belong to. Choose *New Entries* and enter a name and a description for the action. Furthermore, you can maintain the following fields:

- **Init Function Before Processing:** You can specify an init before processing function, which is executed at the beginning of each action call. The function module maintained



here should have the same signature as the template /AIF/FILE\_TEMPL\_INIT\_ACTION. How often this processing function is executed is finally determined during runtime, based on the content of the message which is mapped and the main component type the action handles. If the component type is a structure, the function(s) is executed only once. If the main component type is a table, the function module(s) is executed for every dataset in the table.

- **Commit Mode and Commit Level:** The commit mode can be “no commit” and then the commit level is not taken into account. Other options for the *Commit Mode* are “commit work” and “commit work and wait”, which relate to their corresponding ABAP statement for committing database changes. The *Commit Level* determines at which point the commit is done as follows:
  - After each function: A commit is done after each separate function call of the action.
  - After each action for all datasets: Leads to a commit after the action was executed for all datasets of a table the action was called with.
  - After each action for each dataset: Leads to a commit after all functions of the action have been executed for the whole dataset.
  - No commit (but maybe after all actions): Does not lead to a commit at the action level. However, at the interface level, there could be a commit (if it is customized accordingly).
- **Main Component Type:** The main component type is the type of the DDIC structure the action is working with. The action only has access to the data that exists in this structure. When assigning the action in the Customizing activity *Define Structure Mapping*, you have to take care that the component name you are calling the action with has the same type as the main component type of the action. Additionally, in Customizing for the SAP Application Interface Framework under *Define Structure Mapping*, you can assign a table as the *Main Recordtype* in the *Assign Action* view. Then, the action is called with each dataset of the table.

Function modules have to be assigned to the action. Therefore, choose *Define Functions*. Enter a unique number for the function. The number specifies the order in which the functions are to be executed (the lower the number the earlier the corresponding function module is called in the sequence). Enter the name of the function module that is to be executed. Furthermore, you can specify if the processing should be stopped completely if an error occurs in the function module by setting the indicator *Stop Processing on Error*. The following functions are not executed anymore. Function modules that ended with an error are reprocessed when a message is restarted. Function modules executed successfully are not rerun during restart. However, in certain cases, it might be desirable to re-execute function modules that were processed successfully in the previous run. In such cases, indicator *Restart Always* can be set. This indicates that a function should always be rerun during restart. **Note** that if an action is executed for a table and two out of three lines were processed successfully, the function is, despite the restart indicator, not executed for the two successful lines during restart (see Example 2).

**Example 1:**

Function Module	Restart Always	Stop Processing on Error
1	-	X
2	-	X
3	X	X

4	-	X
5	X	X

Case 1: *Function Module 1* fails when processed for the first time - function modules 1, 2, 3, 4, and 5 are rerun during reprocessing.

Case 2: *Function Module 2* fails when processed for the first time - function modules 2, 3, 4, and 5 are rerun during reprocessing.

Case 3: *Function Module 3* fails when processed for the first time - function modules 3, 4, and 5 are rerun during reprocessing.

Case 4: *Function Module 4* fails when processed for the first time - function modules 3, 4, and 5 are rerun during reprocessing.

Case 5: *Function Module 5* fails when processed for the first time - function modules 3 and 5 are rerun during reprocessing.

#### Example 2:

Assume you have an action containing two function modules. The first function module does some pre-calculation, for example, it determines the exchange rate between currencies. This data is needed in function module 2. Since you always want to use the up-to-date exchange rates, the indicator *Restart Always* is set. The action is executed for a table that contains three lines. Function module 1 and function module 2 are executed successfully for lines one and two of the table. However, function module 2 fails to process the data of line three. The data message has an error status. Restarting the data message, function module 1 and function module 3 are executed again for the erroneous line three. **Note** that if indicator *Restart Always* was not set for function module 1, only function module 2 would be executed during restart.

Choosing *Assign Checks* means that you can assign checks to a selected function. You can either enter a check that was defined before as described in chapter 4.2.6 or you can define a field check, a pattern field check, or you can define allowed characters. In the five fields *Field-name*, you have to maintain the fields you want to check.

Furthermore, you can specify fields that should be restored in the case of a restart. The values of these fields are available in the function modules when reprocessing the action.

### 4.2.8 Structure Mapping

In structure mapping, the fields of the source structure are mapped to the fields of the destination structure. Furthermore, value mappings, fix values, checks, and actions are assigned.

To create a structure mapping, choose *Define Structure Mappings*. Enter the namespace and the interface you want to create a structure mapping for.

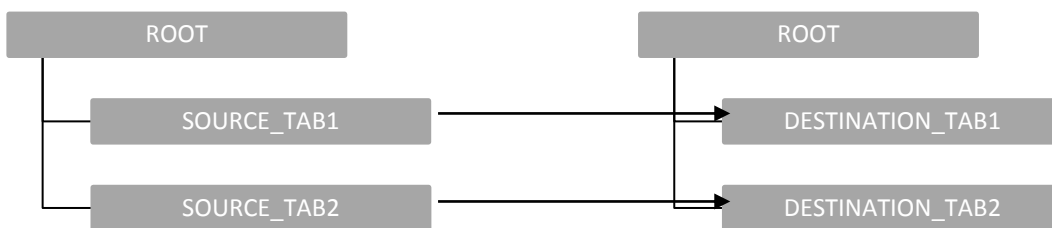
#### 4.2.8.1 Source and Destination Structure

The source structure needs to be mapped to its intended destination structure. Enter the name of the source structure you want to map. The source structure needs to be part of the record type in chapter 4.2.2.

Select the source structure and choose *Assign Destination Structure*. Maintain a number for the structure mapping. Enter the destination structure. The destination structure has to be a component of the SAP data structure defined during interface definition (see chapter 4.2.2). Furthermore, a function before mapping can be defined. The function module should have the same structure as the template /AIF/FILE\_TEMPL\_MAP. The function module defined here is executed before the mapping of the current structure mapping. In the *Function Module After Mapping* field, a function module can be maintained that is executed after the mapping of the current structure mapping. The function module should have the same signature as the template /AIF/FILE\_TEMPL\_MAP. If the selected source and destination structures are completely identical, you can set the *Move Corresponding Fields* indicator. All fields are mapped automatically from the source to the destination structure and no further field mappings need to be maintained (although you can do it if there are additional fields in the destination structure or you want to alter the 1:1 mapping behavior for specific fields).

Setting the *Indirect Mapping* indicator is needed for a hierarchical mapping (see chapter 4.2.8.7) to map from a source substructure to a destination substructure.

**Example:**



**Figure 10: Mapping of Source Structure to Destination Structure**

Figure 10 shows two example structures. The source structure is displayed on the left and the destination structure on the right. To map SOURCE\_TAB1, enter the structure in the table of the *Select Source Structures* screen. To assign the destination structure, select the SOURCE\_TAB1 and choose *Assign Destination Structure*. Maintain a number for the structure mapping. Enter the DESTINATION\_TAB1 in *Destination Structure*.

#### 4.2.8.2 Assign Checks

It is possible to assign checks to the structure mapping. You can either assign checks defined in the Customizing activity *Define Checks* (see chapter 4.2.6) or you can define a check directly in the structure mapping.

If you want to assign a predefined check, you have to enter the namespace and the name of the check you want to use. If the indicator *Check Raw data* is set, the raw data is checked. Otherwise the SAP data structure is checked. If a check is not successful, this is normally treated as an error. However, it is possible to ignore the data if the check was not successful. In *fieldname 1* to *fieldname 5*, the fields can be specified that can be used in a "where" condition, in a check function, or for message variables. The field specified in *fieldname 1* is checked in the case of a field check, pattern field check, or if allowed characters were specified. The other fields specified in *fieldname 2* to *fieldname 5* are ignored in that case.

If you enter the name of a check that does not yet exist, the system offers you the opportunity to create this check. If you decide to create the check, you can access the check definition by double clicking on the check name.

#### 4.2.8.3 Define Fix Values

For a field that is supposed to have the same value every time, you can define a fix value. Enter the name of a field of the destination structure in *Fieldname*. You can either enter a fix value in the *Value* field or you can use a named fix value defined in the Customizing activity *Define Fix Values* (see chapter 4.2.5). Enter the name of the namespace and the name of the named fix value in the corresponding fields. At runtime, the values are written in the fields of the destination structure.

##### Example:

Fieldname	NS	Name of Fix Value	Value
FIELD_1			1
FIELD_2	MY_NS	FIX_VAL	

At runtime, '1' is entered in all the cells of FIELD\_1 of the current destination structure. All the cells of FIELD\_2 are replaced with the value defined in the named fix value MY\_NS/FIX\_VAL.

#### 4.2.8.4 Define Field Mappings

The fields of the source structure need to be mapped to the fields of the destination structure. This is done using field mappings. To define a field mapping, the following fields are available:

- *Field in Destination Structure*: Defines a field in the destination structure to where the data is mapped.
- *Fieldname*: It is possible to insert up to five fields as source fields. These fields contain the name of a field in the source structure. However, those fields can contain the field names of the destination structure as well. In this case, an '@' sign must prepend the fieldname. The defined fields are available as parameters and can be used in for instance value mappings or checks. An offset and a length can be defined. Furthermore, a separator string can be maintained. The contents of the field-name fields are split by this character or symbol in the destination structure. If multiple fieldnames are maintained and no further settings are made, the value of the fields is concatenated into the destination field.
- *Offset and Field Length*: Specify the offset and field length of the data in the destination structure.
- *Value Mapping Function*: A function module for value mapping can be maintained here. This function module should have the same structure as /AIF/FILE\_TMPL\_VALMAPPING.
- *Value Mapping*: A value mapping that was created in the Customizing activity *Define Value Mappings* can be inserted here. The fields defined in fieldnames 1 to 5 are available as parameters for the value mapping. To make use of the parameter you have to insert '\$<Fieldname number>' in an appropriate field in the value mapping. At runtime, '\$<Fieldname number>' is replaced by the current value of the field. They can be used in a "where" condition or in a value mapping function defined in this value mapping. If an error message was defined in the value mapping (see chapter 4.2.4.5), the fields can be used to replace placeholders in the error

message as well. Furthermore, they can be used as external values for a single or multiple value mapping table. If you enter a name of a value mapping that does not exist, the system offers you the opportunity to create the value mapping. Double click on the value mapping to access the Customizing activity *Define Value Mapping* (see chapter 4.2.4).

- *Data Element for Conversion/ Conversion Routine/ Direction of Conversion Exit.*: Enter the conversion routine (for example, *ALPHA*) in the *Conversion Routine* field. Maintain the direction of the conversion in the *Direction of Conversion* field. Instead of entering a conversion routine you can also insert a *Data Element for Conversion* (for example *DATS*).

All further input fields on this screen are needed for the hierarchical mapping and are explained in chapter 4.2.8.7.

#### 4.2.8.5 Conditions

In certain circumstances, a field mapping as defined in chapter 4.2.8.4 might not be applicable. For example, a field in the source structure has a certain value and you do not want to use the value mapping that is usually used for the field. Instead, you want it that the data of an alternative field should be entered in the destination field, or the destination field should be empty, or an alternative mapping should be used. In such cases, conditions can be defined.

You can define one of three condition types as follows:

- *Empty Value*: If the check is true, then the *field in dest. Structure* is blank. Otherwise, the normal field mapping is used.
- *Ignore Value Mapping – Use Alternative Fieldname*: In the 5 fields for *Alt. Fieldname*, you can specify up to 5 alternative field names. If more than one field name is given, then the values of the fields are treated as strings and concatenated in the destination field. You can specify an offset and length for each field. Neither the normally configured value mapping nor an alternative value mapping is executed.
- *Alternative Value Mapping / Fieldname*: This gives you 2 options. In the first option, you specify up to 5 *Alt. Fieldnames*, whose values are used for the normally configured value mapping. In the second option, if you also maintain the fields *Namespace* and *Alt. Value Mapping*, then the *Alt. Fieldnames* is used to call the alternative value mapping.

To use a condition, a check needs to be defined. You can either use a check defined in the Customizing activity *Define Checks* (see chapter 4.2.6) or you can define a check directly in the condition definition. The check evaluates the contents of the field specified in field name 1 (to field name 5 in the case of a where condition). In case a check function module is defined, other fields can be checked as well.

#### Example:

You have a value mapping defined for *DEST\_FIELD\_1* of your destination structure. However, if *SRC\_FIELD\_1* of the destination structure is not empty, the value mapping should not be used. Instead, the value of *SRC\_FIELD\_1* should be entered in *DEST\_FIELD\_1*. Create the field mapping for *DEST\_FIELD\_1* with value mapping as described in chapter 4.2.8.4. Choose *Define Conditions* and enter a number for the condition. Select condition type *Ignore Value Mappings – Use Alternative Fieldname*. In *Field Check*, select *Not Empty*. Enter *SRC\_FIELD\_1* in *Fieldname 1*. Enter *Alt. Fieldname 1* in *SRC\_FIELD\_1*.

#### 4.2.8.6 Assign Actions

The actions created in the Customizing activity *Define Actions* have to be assigned in the structure mapping. Several fields have to have values:

- *Action Number*: Defines the order in which the actions are processed.
- *Namespace and Action*: Defines the action to be used.
- *Record Type*: Determines the data to be handed over to the SAP Application Interface Framework. The record type has to match the structure as defined in the *Main Component Type* field in the Customizing activity *Define Actions*.
- *Stop Processing on Error*: If the indicator is set and an error occurred while processing the action, the processing stops after the action.

#### 4.2.8.7 Hierarchical Mapping

In the SAP Application Interface Framework, it is possible to map fields across different hierarchy levels. This means you can map a table that is on the first hierarchy level in the source structure to a table that is on the second hierarchy level in the destination structure. Using the example structure displayed below, you could map RAW\_HEADER\_T and RAW\_ITEM\_T to the fields of the subcomponents of HEADER\_AND\_ITEM1\_T (marked with (♦)).

Source Structure		Destination Structure	
ZZGTP_TEST_RAW	STRUC	ZGTP_AIF	
• RAW_GLOBAL	STRUC	• HEADER_AND_ITEM1_T(♦)	TABL
• RAW_HEADER_AND_ITEM(♣)	STRUC	○ AIF_HEADER(♦)	STRUC
○ RAW_HEADER2(♣)	STRUC	○ AIF_ITEM_T(♦)	TABL
○ RAW_ITEM2_T(♣)	TABL	• HEADER_AND_ITEM2_T(♣)	TABL
• RAW_HEADER_T(♦)	TABL	○ AIF_HEADER2(♣)	STRUC
• RAW_ITEM_T(♦)	TABL	○ AIF_ITEM2_T(♣)	TABL
• RAW_ADD_DATA_T	TABL	• ADD_DATA_T	TABL
		• GLOBAL_DATA	STRUC

To implement such a mapping, select the table of the raw data structure that should be mapped to the substructure and assign it to the table on the first level in the destination structure. For example, map component RAW\_HEADER\_T of the source structure to component HEADER\_AND\_ITEM1\_T of the destination structure.

Afterwards, you can start with the field mapping (see chapter 4.2.8.4). In the example, you can map the fields of RAW\_HEADER\_T to the fields of HEADER\_AND\_ITEM1\_T.



In order to map the fields of RAW\_ITEM\_T to AIF\_ITEM\_T, you have to define a field mapping as well. AIF\_ITEM\_T is a subcomponent of the destination structure. Enter the subcomponent AIF\_ITEM\_T in *Field in Dest.Struct.* Enter the name of the table in the raw data structure in the *Sub-Table* field (for example, RAW\_ITEM\_T). Enter the field of the subtable in the *Selection Field* field. This field is used to filter the table rows for the hierarchical mapping. Select an *Operator*. *Comparison field* defines a field in the source structure whose value is compared against the value of the field defined in *Selection Field*. The entry in *Comparison field* needs to have the following structure: <name of source structure>!<name of field used for comparison>. It is recommended to use the input help to select a value. Figure 11 shows what the field mapping for a sub table should look like.

To map the data from RAW\_ITEM\_T to AIF\_ITEM\_T, it is necessary to define a field mapping. Therefore, return to *Select Source Structure*, choose *New Entries*, and enter the name of the source structure (for example, RAW\_ITEM\_T). Choose *Assign Destination Structure* and enter the name of the destination structure (for example, AIF\_ITEM\_T). Furthermore, you have to set the *Indirect Mapping* indicator. Afterwards, the field mapping can be performed as described in chapter 4.2.8.4.



Source Structure	RAW_HEADER_T		
No of Struct.Mapping	10		
Field in Dest.Struct	AIF_ITEM_T		
<b>Define Field Mappings</b>			
		Off.	Len.
Fieldname 1	<input type="text"/>	<input type="text"/>	<input type="text"/>
Fieldname 2	<input type="text"/>	<input type="text"/>	<input type="text"/>
Fieldname 3	<input type="text"/>	<input type="text"/>	<input type="text"/>
Fieldname 4	<input type="text"/>	<input type="text"/>	<input type="text"/>
Fieldname 5	<input type="text"/>	<input type="text"/>	<input type="text"/>
Separator String	<input type="text"/>		
Offset	<input type="text"/>		
Field Length	<input type="text"/>		
Value Map. Function	<input type="text"/>		
Namespace	<input type="text"/>		
Value Mapping	<input type="text"/>		
Convers. Rout.	<input type="text"/>		
Direction of conv.	<input type="text"/>		
Sub-Table	RAW_ITEM_T		
Selection Field	HEADER_KEY		
Operator	EQ Equal		
Comparison field	RAW_HEADER_T!HEADER_KEY		
Namespace	<input type="text"/>		
Name of Fix Value	<input type="text"/>		
Value	<input type="text"/>		

**Figure 11: Hierarchical Mapping**

If data is to be mapped from one sub table to another and both tables are on the same hierarchical level, only the *Sub-table* field has to be filled. In the example, such a mapping would be needed to map RAW\_HEADER\_AND\_ITEM to HEADER\_AND\_ITEM2\_T (marked with (♣)). To implement such a mapping, choose *Select Source Structure*, choose *New Entries* and enter the name of the top source structure, for example, RAW\_HEADER\_AND\_ITEM. Select the new source structure, choose *Assign Destination Structure*, and enter the name of the destination



structure, for example, `HEADER_AND_ITEM2_T`. Choose *Define Field Mappings* to map the fields. If the subcomponents of the top source and destination structure are structures, the fields can be mapped directly (for example, the fields of `RAW_HEADER2` can be mapped directly to `AIF_HEADER2`). If the subcomponents of source and destination structure are tables, the table in the destination structure has to be defined as *Field in Dest.Struct* and the table in the source structure has to be defined in *Sub-Table*. For example, in the *Field in Dest.Struct* field, `AIF_ITEM2_T` is inserted and in *Sub-Table*, `RAW_ITEM2_T` is inserted. To define a field mapping for the sub tables, define the table of the source structure as source structure (for example, `RAW_ITEM2_T`) and assign the table of the destination structure as destination structure (for example, `AIF_ITEM2_T`). Furthermore, you have to set the *Indirect Mapping* indicator. The field mapping can be performed as usual (see chapter 4.2.8.4).

#### 4.2.9 Interface Determination

Using interface determination, it is possible to configure which interface in the SAP Application Interface Framework should be used to process a message. The interface that should be used is determined by the proxy class that handles the inbound data along with the fields whose values are used to determine the correct interface. If you create a new version of an interface in the SAP Application Interface Framework, you can use interface determination to set the new version live. Furthermore, interface determination enables you to create different interfaces in the SAP Application Interface Framework for different sending systems or other fields in the header or payload of the message. You can use interface determination to decide which interface should be used for processing based on the content of the message.

To use interface determination, the name of the proxy class needs to be maintained. Furthermore, you have to define up to two fields you want to use to determine the interface. For both fields, a *Field Category* needs to be defined. There are three field categories as follows:

- *Field from XI header data*: Determines that a field in the SAP NetWeaver PI header is used. Based on its value, you can define different interfaces.
- *Field from proxy-generated structure*: Determines that a field in the raw data structure is used and, based on its value, defines one of the available interfaces that should be used for processing in the SAP Application Interface Framework.
- *Sender field from routing data*: Determines that the *Sender* field is used. Based on different values you define in *Assign Interface*, the message is processed by the corresponding interface.

Furthermore, the fieldnames used for the determination have to be assigned.

The interfaces that use the defined proxy class have to be assigned. Choose *Assign Interface* and choose *New Entries*. You can maintain two values and select an operator for each value. The first operator and value are related to the first field maintained in the previous maintenance view *Define Proxy Class*. The second operator and value are related to the second field of the previous view. Furthermore, the namespace and interfaces used if the defined values fit needs to be maintained.

#### Example:

You have an interface MY\_IF in namespace MY\_NS. In the generated proxy structure of your proxy, a SRC\_FIELD\_1 field and a SRC\_FIELD\_2 field exist. You want to use your interface if the value of SRC\_FIELD\_1 equals 'TEST' and the value of SRC\_FIELD\_2 equals 'EXAMPLE'

Enter the name of your proxy class. Select *Field Category: Field from proxy-generated structure* for the first field. Enter SRC\_FIELD\_1 in the first *Fieldname* field. Select *Field Category: Field from proxy-generated structure* for the second field. Enter SRC\_FIELD\_2 in the second *Fieldname* field.

Choose *Assign Interface*. Enter *Equal* in the first *Operator* field. Enter TEST in the first *Value* field. Select *Equal* in the second *Operator* field. Enter EXAMPLE in the second *Value* field. Enter MY\_NS in the *Namespace* field. Enter *Interface Name* MY\_IF and maintain the interface version.

## 4.2.10 Error Handling

### 4.2.10.1 Define Applications

In this Customizing activity, you define the applications that are available in the *Monitoring and Error Handling* transaction of the SAP Application Interface Framework. AIF is the standard application delivered with the SAP Application Interface Framework. However, it is possible to implement custom applications. How this can be done is described in chapter 0, therefore, this chapter only gives a short overview about what can be maintained.

First of all, you can maintain the application itself. The application-specific selection can be maintained as well as the action handler for the *Monitoring and Error Handling* transaction and the entry data façade.

For each application, key fields can be maintained. These key fields structure the messages in the Data Messages view in the *Monitoring and Error Handling*.

Furthermore, the functions have to be maintained for each application. The function name and view that the function belongs to have to be maintained.

### 4.2.10.2 Global Features

*Define dissolved structures:* In some cases, the structure of the messages of an interface includes substructures that are always related to a parent structure and should not be displayed separately. By defining a dissolved structure name, the corresponding structure is not shown on a separate level in the *Monitoring and Error Handling* transaction, but is consolidated with the parent structure.

*Define trace level:* Trace levels define the level of detail for the log messages that is saved in the application log. The standard trace level is 0. However, a higher trace level, which results in more processing information (depending on the setup of the trace levels), can be assigned to a specific interface or data message in the *Monitoring and Error Handling* transaction or directly in the database table /AIF/FINF\_TL.

### 4.2.10.3 Namespace-Specific Features

- *Define namespace specific Features:* Messages processed in the SAP Application Interface Framework are recorded in a single index table where they are identified by a message GUID or file number. If you want to use your own single index table, you have

© 2012 SAP AG Dietmar-Hopp- Allee 16 D-69190 Walldorf	Title: SAP Application Interface Framework Version: 4.0 Date: 10.10.2012	Page 34 of 49
--	--	---------------

to maintain it in this maintenance view. It is recommended that separate single index tables are created for similar groups of interfaces, that is, interfaces sharing the same purpose and key fields. This prevents the single index tables becoming very large and adversely affecting performance. You have to create your own single index table if you want to use key fields (see chapter 4.2.10.4). If no single index table is maintained, the standard table /AIF/STD\_IDX\_TBL is used. Furthermore, you have to specify a module pool that calls the subscreen specified in the *Screen Number* field. This screen should have the key fields of your single index table as input fields. After executing the *Monitoring and Error Handling* transaction, this screen is part of the selection screen as *More specific selection*, where you can select key fields. The creation of an index table and the implementation of a subscreen are explained in chapter 4.3.

- *Define Changeable Fields*: Here you can define fields users are able to change in the *Monitoring and Error Handling* transaction if a message ended in an error status. **Note** that if a user is not allowed to change field values, this might be related to a missing authorization. Those fields are changeable on an interface level. They are editable for all interfaces that belong to this namespace and that contain them.
- *Hide Structures*: In some cases you might not want the user of the *Monitoring and Error Handling* transaction to see which data was transferred in a structure. In this case, the structure can be hidden.
- *Configure Alerts*: Here you can define in which cases an alert should be created. You have to define the interface name and version as well as the recipient that should be alerted. Furthermore, you can define the statuses that create an alert. The recipient assignment can be key field-dependent. To use a key field-dependent recipient assignment, you have to create a recipient assignment table. To do so, copy table /AIF/T\_ALRT\_DEF to a table in your namespace. Enhance your recipient assignment table with your key fields. Enter the name of your table in *Rec. Assgn table*. Furthermore, you have to maintain your key field as relevant for alerts (see chapter 4.2.10.4 *Define Key fields for Multi Search*). **Note** that the key fields you enter in your recipient assignment table have to be key fields in your single index table as well. The component names must match.
- *Define Recipients*: The recipients defined here are needed for the alert configuration. In a subsequent step, users can be assigned to a recipient.

#### 4.2.10.4 Interface-Specific Features

- *Define Key fields for Multi Search*: It is possible to define key fields for messages. Key fields are fields within data messages that are of special importance. In the *Monitoring and Error Handling* transaction, these fields can be used to structure the data message in the hierarchy of the Data Messages view. The key fields are written into the single index table. Therefore, an interface's single index table needs corresponding fields for the key fields entered here. The *Key Field Name* is the name of the field in the single index table that is used for storing the key field value. Furthermore, you can define the name of a selection parameter, which is used in the module pool specified in *Define Namespace specific Features* of the Customizing activity *Namespace-specific Features* (see chapter 4.2.10.3). The parameter can be used on the selection screen of the *Monitoring and Error Handling* transaction to select only those messages where a key field has a certain value. A *Fieldname* has to be defined, which is a field in the source or destination structure. Furthermore, you have to select the selection type. The *Multi Selection Type* specifies where the data is selected from as follows:

**Single Selection:** If the *Hide Node* indicator is set, the key field is not used to structure the message in the data messages view. *Parent Fld. SN.* is used to build the tree structure of the Data Messages view. For that reason, this field has to contain an existing sequence number. You can select an *Icon* and a *Tooltip* which are displayed in the Data Messages view. Using *Fname for alert mgmt* it is possible to specify a field name that is used for the alert recipient determination if the corresponding indicator *Rel. for alert rec.* is set. The alert recipient determination allows you to alert recipients dependent on the key field value. In order to use the alert recipient determination, you have to create an alert recipient table and to configure alerts (see ter 4.2.10.3, *Configure Alerts*). *Category Fieldname* assigns an alert container element name. The value of the key field is transferred to an alert container via the container element.

- **Multiple Selection:** Enter the name of a multi index table. In a multi index table you can store multiple key field values for one data message. An example multi index table is displayed below. In this table, *KEY\_FIELD\_1* is the key field. Your key fields depend on your business case. The other fields are mandatory in each multi index table.

Field	Key Field	Initial Values	Data element	Group
MANDT	X	X	MANDT	
MSGGUID	X	X	GUID_32	
COUNTER	X	X	INT4	
.INCLUDE			/AIF/IFKEYS	AIFKEYS
.INCLUDE			/AIF/ADMIN	ADMIN
PID			SXMSPID	
KEY_FIELD_1			CHAR20	

**Example:** You have an interface to receive sales orders. For this interface you can create a single index table and a multi index table. The single index table contains the sales order number as key field. The multi index table contains a key field for the sales order items.

- **Document ID selection:** The document numbers that are created by the SAP Application Interface Framework are stored in the document index table provided by the SAP Application Interface Framework. The field that contains the document number must be defined as key field.
- **Define Changeable Fields:** You can define fields that are editable in the *Monitoring and Error Handling* transaction at an interface level. Those fields are only editable for messages coming from the corresponding interface.
- **Hide structures:** Structures can be hidden at interface level too. This means that a structure's content is not displayed in the *Monitoring and Error Handling* transaction, if the structure is customized to be hidden here.
- **Assign Recipients without Key Fields:** The corresponding recipients are active for all data messages of the given interface, independent of any key fields that might have been configured. You can use this feature to define fallback recipients that are used in case no key field-specific recipient can be defined or for a simple recipient setup.
- **Assign authorization objects:** You can assign authorizations based on a single message's content. To achieve this, you have to specify key fields that are relevant for au-

thorization and create a custom authorization object (transaction SU21). This object requires a field ACTVT (must be the same as in authorization object /AIF/ERR). One field for each key field that is required for authorization has to be maintained. The authorization object and the authorization fields have to be maintained in Customizing for the SAP Application Interface Framework.

**Example:** Assume that a data message includes a plant identifier. A business user is responsible only for a specific plant, so they should only be allowed to display and/or change messages for the specific plant that is relevant to them.

**Note:** Interface-specific Features can only be maintained if, you have maintained the interface in *Define Namespace-Specific Features* in Customizing activity *Namespace-specific Feature*.

#### 4.2.10.5 Define Recipients

Recipients define in which cases the users that are assigned to them receive an alert. Furthermore, in the *Interface Monitor*, users only see the interfaces of the recipients they are assigned to. You can create new recipients in the Customizing activity *Namespace-Specific Features*. To assign users to recipients, choose *Define Recipients* in Customizing for the SAP Application Interface Framework under *Error Handling*. Enter a *Namespace* and a *Recipient for Alert*.

- **Assign Users:** For each user you want to maintain, you can specify the message type that is included in an alert message. If *Include in overview* is checked, the user is able to see the messages in the *Interface Monitor* of the SAP Application Interface Framework. Furthermore, setting the *Tech. User* indicator allows you to specify users that can see the number of messages with technical errors as well as the number of messages that are *In process* in the *Interface Monitor*.
- **Assign Roles:** Instead of assigning single users to a recipient, you can assign roles to a recipient for alert. The settings apply for all users assigned to this role.
- **Assign External Addresses:** You can assign external addresses that are notified if messages occur with the specified type for a recipient. You can select from different communication methods, for example, fax number, e-mail, or printer.

Depending on the *Message Type* configuration of the recipient, the user receives an alert. The same alert is not raised twice. The same alert means the message category and all other variables are the same. Only after the initial alert is confirmed, is the alert raised again.

**Note:** Using Customizing provides you with a recipient-specific view for assigning users to recipients. However, transaction /AIF/RECIPIENTS provides you with a user-specific view. In this transaction, you get an overview of the recipients a specific user is assigned to. Additionally, you can assign new recipients to the user.

#### 4.2.11 Interface Variants

You can define different variants for an interface. Interface variants are used to enable an alternative processing of specific data messages in the SAP Application Interface Framework. Typically, you would create an interface variant if you want to make some adaptations to an interface but do not wish to make major changes or create a new interface. With interface variants, it is possible to create different behaviors for an interface. In different interface variants, different checks, mappings, or actions can be defined for instance. Which variant is used by a message is defined by key fields based on your settings in Customizing.

**Example:**

© 2012 SAP AG Dietmar-Hopp- Allee 16 D-69190 Walldorf	Title: SAP Application Interface Framework Version: 4.0 Date: 10.10.2012	Page 37 of 49
--	--	---------------

An organization has subsidiaries in multiple countries. Some data is aggregated in a central system. Although processing of the incoming messages is very similar, the inbound structure and, as a result, the mapping differs slightly. Instead of creating two separate interfaces, a single interface is used in combination with an interface variant. Depending on the unique plant number, the interface variant can become active to exchange parts of the original mapping and processing information.

#### 4.2.11.1 Define Interface Variants

Define the interface variant by maintaining a name for the variant and a description.

#### 4.2.11.2 Define Assigning Tables

An assigning table is necessary to determine the correct variant that is used for processing of a message. The table needs to have at least one component to store the value of the key field. Based on the key field value(s) of a message, the interface variant is determined.

Before you can assign a variant assigning table, you might need to create one that you can use, if you or another interface developer have not done so already. The assigning table should have at least the following key fields:

Component Name	Type	Description
NS	/AIF/NS	Namespace
IFNMAE	/AIF/IFNAME	Interface Name
IFVERSION	/AIF/IFVERSION	Interface Version
VARIANT_NS	/AIF/VARIANT_NS	Variant Namespace
VARIANT_NAME	/AIF/VARIANT_NAME	Name of Interface Variant

The table also needs to specify additional fields for the key fields that are used in variant determination. You need to maintain the interface variant in your assigning table using transaction SE16 (or your custom maintenance transaction, if you created one). The fields in the table are later linked to the fields in the actual message data structure using the Customizing activity *Define Interface Key Fields* (see chapter 4.2.11.3).

#### Example:

You want to create a variant MY\_VARIANT that is used for plant 1 for interface IF\_VAR in namespace MY\_NS. The variant assigning table needs one additional component (for example, VARIANT\_KEY) besides the mandatory fields. The following entries have to be maintained in the entry of the assigning table:

Component Name	Value
NS	MY_NS
IFNMAE	IF_VAR
IFVERSION	1
VARIANT_NS	MY_NS
VARIANT_NAME	MY_VARIANT
VARIANT_KEY	PLANT_1



To assign a table, choose *Define Assigning Tables* and enter a namespace. Enter the name of the variant assigning table next to the interface for which you want to create a variant.

#### 4.2.11.3 Define Interface Key Fields

In this activity, you define the key fields that determine the interface variant to be used.

Select the interface for which you want to create a variant. Enter the key field defined in the variant assigning table in the *Interface Variant Key field*. This field is used to select the correct variant from the interface assigning table. In *Fieldname*, enter the name of the field in the message structure that contains the value that determines which variant is to be used. If the value received in the field maintained in *Fieldname* equals the value of the field maintained in the specified key field in the assigning table, the interface variant is used. **Example:** (continued from 4.2.11.2)

In order to use the interface variant MY\_VARIANT of interface IF\_VAR, select *Interface Variants* → *Define Interface Key Fields*. Enter the namespace, interface name, and version of your interface (for example, MY\_NS/IF\_VAR/1). Choose *New Entries* to enter the key field used to determine the variant. In *Interface Variant Key field*, enter the key field of the variant assigning table (for example, VARIANT\_KEY). In *Fieldname*, enter the field of the source data structure that contains the value to determine the interface variant (for example, PLANT\_ID).

Now, if interface IF\_VAR receives a message and the value of field PLANT\_ID is PLANT\_1, interface variant MY\_VARIANT is used.

#### 4.2.11.4 Define Variant Mappings

To define a variant mapping, choose *Define Variant Mappings*, enter the namespace and the interface variant. The actions that can be performed here are similar to the one in the Customizing activity *Define Structure Mapping* (see chapter 4.2.8). The following actions can be performed for a variant.

##### 4.2.11.4.1 Select Interface

You select an interface and navigate to *Add/Select Structure Mapping*.

##### 4.2.11.4.2 Add/Select Structure Mapping

Adding or selecting a structure mapping works in a similar way to the source and destination structure definition in structure mapping (see chapter 4.2.8.1). You can create a new source and destination structure mapping here or define a different mapping (with same key values) in order to overwrite the original definition in the structure mapping,

Whether an entry is added or overridden is based on a comparison of the following keys:

- Namespace
- Interface name
- Interface version
- Source structure number of structure mapping

Only if all these values match an entry in the original structure mapping is the original mapping overridden. Otherwise, the variant structure mapping is executed in addition to the original mapping.

##### 4.2.11.4.3 Assign Checks

© 2012 SAP AG Dietmar-Hopp- Allee 16 D-69190 Walldorf	Title: SAP Application Interface Framework Version: 4.0 Date: 10.10.2012	Page 39 of 49
--	--	---------------

Assigning checks works in a similar way to assigning checks in structure mapping (see chapter 4.2.8.2). You can create a new check assignments here or define a different check assignment (with the same key values) to overwrite the original one.

An entry in the original check assignment is overridden if following keys match, otherwise the check assignment is executed in addition to the original check assignments:

- Namespace
- Interface name
- Interface version
- Source structure
- Number of structure mapping
- Number of the check

To deactivate an original check, assign a variant check with the same key mentioned in the list above and leave all the fields empty. Technically, this replaces the old check with a variant check that always succeeds.

#### 4.2.11.4.4 Define Fix Values

Defining fix values works in a similar way to the definition of fix values in the structure mapping (see chapter 4.2.8.3).

You can create a new fix value assignment here or define a different fix value assignment (with the same keys) to overwrite the original one.

An entry in the original fix value definition is overridden, if the following keys match, otherwise, the fix value definition is valid in addition to the original fix value definitions:

- Namespace
- Interface name
- Interface version
- Source structure
- Number of structure mapping
- Fix value number

#### 4.2.11.4.5 Define Field Mappings

Defining field mappings works in a similar way to the definition of field mappings in structure mapping (see chapter 4.2.8.4).

You can create a new field mappings here or define a different field mapping (with the same keys values) to overwrite the original one.

An entry in the original field mapping definition is overridden, if the following keys match, otherwise, the field mapping definition is valid in addition to the original field mapping definitions:

- Namespace
- Interface name
- Interface version



- Source structure
- Number of structure mapping
- Number of field mapping

**Note:** When you exchange an existing field mapping, the subordinated conditions of the original field mapping no longer apply. If you want to use the same conditions for the exchanged field mapping, you need to redefine the conditions as described in chapter 4.2.11.4.6.

#### 4.2.11.4.6 Define Conditions

Defining conditions works in a similar way to the definition of conditions in the structure mapping (see chapter 4.2.8.5).

You can create a new condition here or define a different condition (with the same keys) to overwrite the original one.

An entry in the original condition definition is overridden, if the following keys match, otherwise, the condition definition is valid in addition to the original condition definitions:

- Namespace
- Interface name
- Interface version
- Source structure
- Number of structure mapping
- Number of field mapping
- Condition number

#### 4.2.11.4.7 Assign Actions

Assigning actions works in a similar way to the action assignment in structure mapping (see chapter 4.2.8.6).

You can create a new action assignments here or define a different action assignment (with the same keys) to overwrite the original one.

An entry in the original action assignment is overridden, if the following keys match, otherwise, the action assignment is valid in addition to the originally assigned actions:

- Namespace
- Interface name
- Interface version
- Source structure
- Action number

#### 4.2.11.4.8 Define Fix Value Variant

You can define a variant of an original fix value definition in a similar way to the description under named fix value (see chapter 4.2.5.1).

#### 4.2.11.4.9 Define Fix Value Table Variant

You can define a variant of an original fix value table definition in a similar way to the description under define fix value table (see chapter 4.2.5.2).

#### 4.2.11.4.10 Define Value Mapping Variant

You can define a variant for an original value mapping definition. This allows you to exchange an existing value mapping for your variant. The original value mapping needs to exist, that is, it needs to be defined as described in chapter 4.2.4. You specify the original value mapping in the first two fields (from the left) and the value mapping it is exchanged with in the last two fields.

#### 4.2.11.4.11 Define Action Variant

You can change the behavior of your actions by defining an action variant. You need to specify the existing, original action using the fields for namespace and action name.

#### 4.2.11.4.12 Define Functions Variant

You can add functions to the action by specifying them using an unused function number or you can exchange existing functions by specifying an existing function number.

#### 4.2.11.4.13 Assign Checks Variant

You can add new or exchange existing action function checks. Checks with a new number are added and checks with existing numbers replace the original check.

### 4.3 Implementation of Subscreen with Key Fields

As described in chapter 4.2.10.3, it is possible to define a custom single index table and a *More specific selection Screen* that makes it possible for the users to search for custom key fields on the selection screen of the *Monitoring and Error Handling* transaction. This chapter explains how to create a *More specific selection screen* for an existing interface.

#### 4.3.1 Creating a Single Index Table

A single index table can be created in transaction SE11. You can use the standard index table /AIF/STD\_IDX\_TBL as a template. An index table has to contain the following fields:

Component	Key	Data element	Group
MANDT	X	MANDT	
MSGGUID	X	GUID_32	
.INCLUDE		/AIF/IFKEYS	AIFKEYS
.INCLUDE		/AIF/ADMIN	ADMIN
PID		SXMSPID	

**Note:** Include structure /AIF/IFKEYS consists of the components namespace, interface name, and version. These fields are the key fields of the standard index table. Include structure /AIF/ADMIN contains components to display the number of log messages. Furthermore, the structure contains components to store, for example, the creation date and user, the application log number, and the status of the message.

Furthermore, you can include your own key fields. The table ZAIF\_EXAMPL\_IDX below shows an example for an index table including the two key fields KEY\_FIELD\_1 and KEY\_FIELD\_2:

Component	Key	Data element	Group
MANDT	X	MANDT	
MSGGUID	X	GUID_32	
.INCLUDE		/AIF/IFKEYS	AIFKEYS
KEY_FIELD_1		CHAR20	
KEY_FIELD_2		CHAR20	
.INCLUDE		/AIF/ADMIN	ADMIN
PID		SXMSPID	

### 4.3.2 Implementation of Subselection Screen with Key Fields

To use the key fields for selecting data, a module pool needs to be implemented that contains the key fields as selection parameters. In the module, it is necessary to define a selection screen. It is important that this screen is defined as a subscreen, since the screen is a subscreen of the selection screen of the *Monitoring and Error Handling* transaction. Furthermore, the module pool must have the event AT SELECTION-SCREEN OUTPUT. The event should at least have the statement `/aif/cl_global_tools=>get_value_from_mem( )`. This statement is needed to get the values from memory if you want to use a parameter transaction based on `/AIF/ERR_BASE` to call *Monitoring and Error Handling*.

The example code below shows a subselection screen that contains two parameters. These parameters are defined as selection parameters in *Define interface specific features*.

```

PROGRAM zaif_exempl_sub_selscr.

SELECTION-SCREEN BEGIN OF SCREEN 0001 AS SUBSCREEN.
  PARAMETERS: p_c2 TYPE string.
  PARAMETERS: p_c1 TYPE string.
SELECTION-SCREEN END OF SCREEN 0001.

AT SELECTION-SCREEN OUTPUT.
  /aif/cl_global_tools=>get_value_from_mem( ).

```

### 4.3.3 Assign Subselection Screen and Single Index Table

After you have created your single index table and implemented your module pool containing the subselection screen, you have to assign them to the interface with which you want to use them. This is done in the Customizing activity *Namespace-specific features* (see ter 4.2.10.3 and Figure 12).

Namespace	ZGTP
Interface Name	ZEXAMPLE
Interface Version	1
Define namespace specific Features	
Message idx table	ZAIF_EXAMPL_IDX
Module	ZAIF_EXAMPL_SUB_SELSCR
Screen number	0001
Weighting Factor	

Figure 12: Example of Defining Subscreen and Single Index Table for Selection

#### 4.3.4 Define Key Fields

If you execute the *Monitoring and Error Handling* transaction and select the interface for which you specified the subscreen, an additional subselection screen appears. In order to use the selection screen, you have to define key fields for the interface in the Customizing activity *Define interface specific features* (see chapter 4.2.10.4). The selection parameters of the subscreen have to be assigned to the key fields. Figure 13 shows how key field KEY\_FIELD\_1 is defined for interface ZEXAMPLE.

Namespace	ZGTP
Interface Name	ZEXAMPLE
Interface Version	1
Field Sequence No.	1
<b>General</b>	
Key Field Name	KEY_FIELD_1
Data element	/AIF/FIELDVALUE
Name Select Para.	P_C1
<input type="checkbox"/> Is field select-opt?	
<input type="checkbox"/> Not displayed as column?	
Weighting Factor	
Fieldname	RAW_GLOBAL-GLOBAL_C1
Raw or SAP structure	Source structure (raw for inbound, SAP for outbound)
Multi.Selection Type	Single selection
<b>Single Selection</b>	
<input type="checkbox"/> Hide Node	
Parent Fld. SN.	
Icon	@BU@
Tooltip	exmample tooltip
Fname for alert mgmt	
<input type="checkbox"/> Rel. for alert rec.	
Category Fieldname	

**Figure 13: Example Define Key Fields**

Enter the name of your key field from your index table in the *Key Field Name* field. In the *Name Select Para* field, you have to enter the name of the parameter in your selection screen that you want to use to select the data from the index table. Furthermore, you have to maintain the *Fieldname* of your raw or SAP data structure that contains the key field value. The value of this field is stored in the single index table.

## 4.4 BADIs for the Error Handling Application Toolbars

### 4.4.1.1 Overview

It is possible to add new buttons to the application toolbars of the different views of the *Monitoring and Error Handling* transaction. In order to provide such functionality, several BADIs are provided:

BAdI Definition	View
/AIF/V1_ACT	Data Messages view (view 1)

/AIF/V3_ACT	Data Content view (view 3)
/AIF/V5_ACT	Log Messages view (view 5)
/AIF/V5_CHANGEABLE_FIELDS	Log Messages view (view 5)

For example, you could add a button to the application toolbar of the Log Messages view. When you select the button, an external transaction is called, for example, MM01 *Create Material Master Record*. The button can be enabled depending on the content of an application log message.

In order to include your own functionality in a view, it is necessary to create your own BAdI implementation. In the following example, a button is added to the application log view. After selecting the button, a message appears on the screen.

#### 4.4.1.2 Example Implementation for /AIF/V5\_ACT

##### 4.4.1.2.1 Method GET\_ACT\_LIST

This method is used to add additional buttons to the application toolbar. The method has the following parameters:

- IT\_BAL\_DATA is a table that contains data from the application log. The table contains the application log messages of the current data message selection.
- IT\_MSG\_INDEX is a table that contains the message ID and the message number of the message in the application log. In this way, it is possible to display or enable buttons depending on the content of the messages in the application log view. One entry exists for each message ID / message number combination of the current selection. IT\_MSG\_INDEX contains component IDXTABLE. IDXTABLE contains the line numbers of the log messages contained in IT\_BAL\_DATA.
- CS\_TOOLBAR contains the application toolbar. You can add your own buttons to the application toolbar in this method.
- CT\_ACT\_LIST is a table that contains toolbar buttons.

**Example Implementation of GET\_ACT\_LIST:** The *Sample* button is only displayed if a message with ID /AIF/MES and message number 001 is contained in the message of the current application log.

```

METHOD /aif/if_v5_act~get_act_list.
  DATA: ls_toolbar TYPE stb_button,
         lv_found TYPE boolean.

  READ TABLE it_msg_index TRANSPORTING NO FIELDS WITH TABLE KEY
                                msgid = '/AIF/MES'
                                msgno = '001'.

  IF sy-subrc = 0.
    lv_found = 'X'.
  ENDIF.
  IF lv_found = 'X'.
    CLEAR ls_toolbar.
    ls_toolbar-butn_type = 0.
    " This is the custom button's function code
    ls_toolbar-function = 'ZSAMPLE'.
    ls_toolbar-icon      = '@OS@'.
    ls_toolbar-text      = 'Sample'.
    ls_toolbar-quickinfo = 'Sample Button'.
    APPEND ls_toolbar TO cs_toolbar-toolbar->mt_toolbar.
  ENDIF.
ENDMETHOD.

```

#### 4.4.1.2.2 Method DO\_ACTION

This method is used to define the actions of your own buttons. The method has the following parameters:

- IT\_BAL\_DATA is a table that contains data from the application log.
- IV\_UCOMM is the function code of the selected button.
- IS\_COMMAND\_DATA contains data from the user command, for example, the function code and the selected line.
- IR\_MSG contains the payload of the data message to which the log message belongs.
- CT\_OUTPUT\_DATA provides an option to output data to the front end.
- CV\_TOOLBAR\_REFRESH indicates if the toolbar should be refreshed. You can use this indicator, for example, if you want to disable a button after the user has selected it.

**Example Implementation of DO\_ACTION:** If the *Sample* button is selected and a message was selected in the Log Messages view that has message ID */AIF/MES* and message number *001*, an information message appears. Otherwise, an error message is displayed.

```
METHOD /aif/if_v1_act~do_action.

DATA: ls_textid TYPE scx_t100key,
      ls_bal_data TYPE /aif/msg_info_s,
      lv_found TYPE boolean.

IF iv_ucomm = 'ZSAMPLE'.
  READ TABLE it_bal_data INTO ls_bal_data
    WITH KEY msghandle = is_command_data-list_msgh.
  IF sy-subrc = 0.
    IF ls_bal_data-msg-msgid = '/AIF/MES'
      AND ls_bal_data-msg-msgno = '001'.
      lv_found = 'X'.
    ENDIF.
  ENDIF.
  IF lv_found = 'X'.
    MESSAGE i000(/aif/error_handling)
      WITH 'This is a sample implementation'.
  ELSE.
    ls_textid-msgid = '/AIF/ERROR_HANDLING'.
    ls_textid-msgno = '000'.
    ls_textid-attr1 = 'Wrong Selection!'.
    RAISE EXCEPTION TYPE /aif/cx_error_handling_general
      EXPORTING textid = ls_textid.
  ENDIF.
ENDIF.
```



## 5 Enable Users to Monitor Interfaces

This chapter describes the process to enable users to monitor the interfaces for which they are responsible. Since the activities you have to carry out have been explained before, they are only mentioned here and not explained in more detail. References to the relevant chapters are provided.

1. Define your interface in the SAP Application Interface Framework (see chapter 4.2.2).
2. Customize your interface as follows:
  - a. If necessary, define the following:
    - Value mappings (see chapter 4.2.4)
    - Fix values (see chapter 4.2.5)
    - Checks (see chapter 4.2.6)
    - Actions (see chapter 4.2.7)
  - b. Create structure mappings (see chapter 4.2.8)
3. If you have multiple interfaces in the SAP Application Interface Framework for the same ABAP proxy, you have to set up interface determination (see chapter 4.2.9).
4. Create a recipient for your interface (see chapter 4.2.10.3). **Note** that you can also re-use existing recipients. In this case, you can go to step 5. You create a new recipient in Customizing for the SAP Application Interface Framework under *Error Handling* → *Namespace-specific Features*. Enter the namespace for which you want to create the recipient and choose *Define Recipients*. Enter a name and a description for your recipient.
5. Assign users to your recipient (see chapter 4.2.10.5). You do this in Customizing for the SAP Application Interface Framework under *Error Handling* → *Define Recipients*. Enter the namespace and the name of your recipient. You can assign users, roles, or external addresses.
  - a. *Assign Users/Roles*: Select the message type that is included in an alert message. If you set the *Include in overview* indicator, the users that are assigned to the recipient can see the messages in the *Interface Monitor* of the SAP Application Interface Framework. The indicator *Tech. User* identifies technical users. They can also see messages that are *In process* or where a technical error occurred in the *Interface Monitor*.
  - b. *Assign External Addresses*: This allows you to define external addresses (for example, e-mail, SMS, and Fax) that are notified if messages with the specified type occur.
6. (Optional) While the process of assigning users to recipients provides you with a recipient-centric view, a user-centric view exists as well. Therefore, transaction /AIF/RECIPIENTS exists. This transaction allows you to display and maintain the recipients a specific user belongs to.

## Copyright

© Copyright 2012 SAP AG. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP AG. The information contained herein may be changed without prior notice.

Some software products marketed by SAP AG and its distributors contain proprietary software components of other software vendors.

Microsoft, Windows, Excel, Outlook, and PowerPoint are registered trademarks of Microsoft Corporation.

IBM, DB2, DB2 Universal Database, System i, System i5, System p, System p5, System x, System z, System z10, System z9, z10, z9, iSeries, pSeries, xSeries, zSeries, eServer, z/VM, z/OS, i5/OS, S/390, OS/390, OS/400, AS/400, S/390 Parallel Enterprise Server, PowerVM, Power Architecture, POWER6+, POWER6, POWER5+, POWER5, POWER, OpenPower, PowerPC, BatchPipes, BladeCenter, System Storage, GPFS, HACMP, RETAIN, DB2 Connect, RACF, Redbooks, OS/2, Parallel Sysplex, MVS/ESA, AIX, Intelligent Miner, WebSphere, Netfinity, Tivoli and Informix are trademarks or registered trademarks of IBM Corporation.

Linux is the registered trademark of Linus Torvalds in the U.S. and other countries.

Adobe, the Adobe logo, Acrobat, PostScript, and Reader are either trademarks or registered trademarks of Adobe Systems Incorporated in the United States and/or other countries.

Oracle is a registered trademark of Oracle Corporation.

UNIX, X/Open, OSF/1, and Motif are registered trademarks of the Open Group.

Citrix, ICA, Program Neighborhood, MetaFrame, WinFrame, VideoFrame, and MultiWin are trademarks or registered trademarks of Citrix Systems, Inc.

HTML, XML, XHTML and W3C are trademarks or registered trademarks of W3C®, World Wide Web Consortium, Massachusetts Institute of Technology.

Java is a registered trademark of Sun Microsystems, Inc.

JavaScript is a registered trademark of Sun Microsystems, Inc., used under license for technology invented and implemented by Netscape.

SAP, R/3, SAP NetWeaver, Duet, PartnerEdge, ByDesign, SAP Business ByDesign, and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP AG in Germany and other countries.

Business Objects and the Business Objects logo, BusinessObjects, Crystal Reports, Crystal Decisions, Web Intelligence, Xcelsius, and other Business Objects products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of Business Objects S.A. in the United States and in other countries. Business Objects is an SAP company.

All other product and service names mentioned are the trademarks of their respective companies. Data contained in this document serves informational purposes only. National product specifications may vary.

These materials are subject to change without notice. These materials are provided by SAP AG and its affiliated companies ("SAP Group") for informational purposes only, without representation or warranty of any kind, and SAP Group shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP Group products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.