SAP Exchange Infrastructure
(BC-XI)

Release XI 2.0 SR1
Icons

<table>
<thead>
<tr>
<th>Icon</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Caution Icon" /></td>
<td>Caution</td>
</tr>
<tr>
<td><img src="image" alt="Example Icon" /></td>
<td>Example</td>
</tr>
<tr>
<td><img src="image" alt="Note Icon" /></td>
<td>Note</td>
</tr>
<tr>
<td><img src="image" alt="Recommendation Icon" /></td>
<td>Recommendation</td>
</tr>
<tr>
<td><img src="image" alt="Syntax Icon" /></td>
<td>Syntax</td>
</tr>
</tbody>
</table>

Typographic Conventions

<table>
<thead>
<tr>
<th>Type Style</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Example text</em></td>
<td>Words or characters that appear on the screen. These include field names, screen titles, pushbuttons as well as menu names, paths and options. Cross-references to other documentation.</td>
</tr>
<tr>
<td><em>Example text</em></td>
<td>Emphasized words or phrases in body text, titles of graphics and tables.</td>
</tr>
<tr>
<td><strong>EXAMPLE TEXT</strong></td>
<td>Names of elements in the system. These include report names, program names, transaction codes, table names, and individual key words of a programming language, when surrounded by body text, for example, SELECT and INCLUDE.</td>
</tr>
<tr>
<td><em>Example text</em></td>
<td>Screen output. This includes file and directory names and their paths, messages, source code, names of variables and parameters as well as names of installation, upgrade and database tools.</td>
</tr>
<tr>
<td><strong>EXAMPLE TEXT</strong></td>
<td>Keys on the keyboard, for example, function keys (such as F2) or the ENTER key.</td>
</tr>
<tr>
<td><em>Example text</em></td>
<td>Exact user entry. These are words or characters that you enter in the system exactly as they appear in the documentation.</td>
</tr>
<tr>
<td>&lt;<em>Example text</em>&gt;</td>
<td>Variable user entry. Pointed brackets indicate that you replace these words and characters with appropriate entries.</td>
</tr>
</tbody>
</table>
# SAP Exchange Infrastructure (BC-XI)

- Introductory Notes ................................................................................................................ 13
- Architecture .......................................................................................................................... 14
- Messages ............................................................................................................................... 15
- Interface-Based Message Processing ............................................................................. 17
- Design and Configuration .................................................................................................. 19
  - Design Time ..................................................................................................................... 19
  - Configuration Time ......................................................................................................... 21
- Integration Engine and Integration Server ....................................................................... 23
- Overview of XI Components ............................................................................................. 24
- Roles and Tool Access ....................................................................................................... 24
- Design and Configuration Time ........................................................................................... 27
  - Integration Builder ............................................................................................................ 27
  - Component Overview ................................................................................................... 27
    - Main Menu and Standard Toolbar ............................................................................ 29
    - Navigation Area ......................................................................................................... 31
    - Object Editors ............................................................................................................. 35
  - Navigation ....................................................................................................................... 39
  - Editing Objects .............................................................................................................. 40
    - Creating a New Object .............................................................................................. 40
    - Copy Object .............................................................................................................. 43
    - Creating Documentation for an Object ...................................................................... 44
    - Displaying Object Properties ..................................................................................... 44
    - Object History ............................................................................................................ 45
    - Change Lists ............................................................................................................. 46
  - Additional Functions ..................................................................................................... 47
    - Drag & Drop .............................................................................................................. 47
    - Search Help .............................................................................................................. 48
- Special Design/Configuration Functions ....................................................................... 50
  - Design Functions .......................................................................................................... 50
    - Importing Software Component Versions ................................................................ 51
    - Displaying/Changing Software Component Versions ........................................... 51
  - Configuration Functions ............................................................................................... 52
    - Value Mapping ....................................................................................................... 53
    - Clearing the SLD Cache ........................................................................................ 55
- Software Logistics ............................................................................................................ 55
  - Organization of Shipment Contents ............................................................................. 56
  - Software Component Versions ..................................................................................... 58
  - Namespaces ................................................................................................................. 59

---

**SAP Exchange Infrastructure (BC-XI) .......................... 623**

**SAP Online Help 28.05.03**
Object References .................................................................................................... 61
Object Reference Types.......................................................................................... 62
Development Cycle in the Integration Repository ..................................................... 65
Version Management.................................................................................................... 66
Compatibility of Changes .......................................................................................... 67
Ensuring the Compatibility of Mappings ................................................................. 68
Release Transfer ....................................................................................................... 69
Versioning in Transports .......................................................................................... 71
Importing and Exporting Integration Objects ............................................................ 73
Design .............................................................................................................................. 76
Designing Business Scenarios .................................................................................... 77
Business Scenario ...................................................................................................... 78
Example of a Business Scenario .............................................................................. 81
Business Scenario Design Environment .................................................................. 82
Quick Start: Designing a Business Scenario ......................................................... 83
Action Editor ............................................................................................................ 87
Assigning Interfaces .............................................................................................. 89
Deassigning Interfaces ......................................................................................... 90
Creating Documentation for an Action ................................................................. 91
Business Scenario Editor ....................................................................................... 91
Inserting Application Components ...................................................................... 94
Displaying/Changing Application Components ................................................. 96
Removing Application Components .................................................................. 96
Inserting Actions .................................................................................................... 97
Displaying/Changing Actions ............................................................................. 97
Removing Actions ............................................................................................... 98
Defining Start and End Actions ........................................................................... 98
Creating a Connection Between Selected Actions ........................................... 99
Display/Change a Connection Between Selected Actions .................................. 100
Deleting a Connection Between Two Actions .................................................. 102
Functions of the Graphical Editor ...................................................................... 102
Creating Documentation for a Business Scenario ........................................ 104
Printing and Exporting the Business Scenario Graphic ................................ 105
Handling Messages ............................................................................................. 105
Guidelines for Modeling Business Scenarios ...................................................... 107
General Modeling Concept .................................................................................. 107
Modeling Business Scenarios .......................................................................... 108
Modeling Application Components .................................................................. 110
Modeling Actions ................................................................................................. 112
Modeling Connections ......................................................................................... 114
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modeling Process Variants of a Business Scenario</td>
<td>117</td>
</tr>
<tr>
<td>Interface Design</td>
<td>117</td>
</tr>
<tr>
<td>Introduction to Interface Development</td>
<td>119</td>
</tr>
<tr>
<td>Communication Parameters</td>
<td>120</td>
</tr>
<tr>
<td>Communication Partners</td>
<td>121</td>
</tr>
<tr>
<td>Developing Message Interfaces</td>
<td>123</td>
</tr>
<tr>
<td>Message Interface</td>
<td>126</td>
</tr>
<tr>
<td>Message Type</td>
<td>128</td>
</tr>
<tr>
<td>Fault Message Type</td>
<td>129</td>
</tr>
<tr>
<td>Data Types</td>
<td>131</td>
</tr>
<tr>
<td>XSD Editor</td>
<td>133</td>
</tr>
<tr>
<td>Elements and Attributes</td>
<td>135</td>
</tr>
<tr>
<td>Facets</td>
<td>136</td>
</tr>
<tr>
<td>Defining Simple Data Types</td>
<td>137</td>
</tr>
<tr>
<td>Defining Complex Data Types</td>
<td>138</td>
</tr>
<tr>
<td>Exporting XSD and WSDL Documents</td>
<td>139</td>
</tr>
<tr>
<td>Connection Using Adapters and Imported Interfaces</td>
<td>140</td>
</tr>
<tr>
<td>Importing Interfaces from an SAP System</td>
<td>141</td>
</tr>
<tr>
<td>Designing a Mapping</td>
<td>143</td>
</tr>
<tr>
<td>Overview of Mapping Types</td>
<td>144</td>
</tr>
<tr>
<td>Interface Mappings</td>
<td>147</td>
</tr>
<tr>
<td>Message Mappings</td>
<td>148</td>
</tr>
<tr>
<td>Assigning a Field</td>
<td>151</td>
</tr>
<tr>
<td>Data-Flow Editor</td>
<td>152</td>
</tr>
<tr>
<td>Standard Functions</td>
<td>154</td>
</tr>
<tr>
<td>copyValue</td>
<td>159</td>
</tr>
<tr>
<td>createlf</td>
<td>160</td>
</tr>
<tr>
<td>removeContexts</td>
<td>162</td>
</tr>
<tr>
<td>exists</td>
<td>164</td>
</tr>
<tr>
<td>SplitByValue</td>
<td>165</td>
</tr>
<tr>
<td>collapseContexts</td>
<td>167</td>
</tr>
<tr>
<td>User-Defined Functions</td>
<td>170</td>
</tr>
<tr>
<td>Duplicating Subtrees</td>
<td>171</td>
</tr>
<tr>
<td>Runtime Procedure</td>
<td>173</td>
</tr>
<tr>
<td>Frequently Occurring Elements</td>
<td>173</td>
</tr>
<tr>
<td>Contexts</td>
<td>175</td>
</tr>
<tr>
<td>Testing a Message Mapping</td>
<td>175</td>
</tr>
<tr>
<td>XSLT and Java Mappings</td>
<td>176</td>
</tr>
<tr>
<td>XSLT Mapping</td>
<td>177</td>
</tr>
<tr>
<td>XSLT Mapping with Java Enhancement</td>
<td>179</td>
</tr>
</tbody>
</table>
Displaying/Changing Pipeline Elements ................................................................. 275
Defining Conditions and Actions for Troubleshooting ........................................ 276
Processing XML Messages ...................................................................................... 278
Quality of Service ..................................................................................................... 279
Commit Handling ...................................................................................................... 282
Receiver Determination ........................................................................................... 284
Displaying System Error Descriptions .................................................................. 285
Processing Status ..................................................................................................... 286
Processing Acknowledgment Messages .................................................................. 288
Monitoring XML Messages ...................................................................................... 290
Logging ..................................................................................................................... 291
Monitor for Processed XML Messages ...................................................................... 293
Selecting Processed XML Messages ......................................................................... 294
Processing Statistics ................................................................................................ 299
Selecting Archived XML Messages ......................................................................... 301
Displaying XML Message Versions ......................................................................... 302
Proxy Runtime .......................................................................................................... 304
ABAP Proxy Runtime ............................................................................................... 305
Sending a Message ................................................................................................... 305
Receiving a Message ................................................................................................. 306
Error Handling .......................................................................................................... 306
Fault Messages ......................................................................................................... 307
Reporting and Handling Errors ............................................................................... 309
Special Services of the Proxy Runtime ..................................................................... 311
The Factory Class ...................................................................................................... 311
Processing Default and Initial Values ...................................................................... 313
Creating a Payload ................................................................................................... 314
Setting the Receiver System .................................................................................... 315
Appending Attachments at the Sender .................................................................... 316
Querying Attachments at the Receiver ..................................................................... 318
Guaranteeing the Sequence of Asynchronous Messages ....................................... 319
Accessing the Message GUID .................................................................................. 319
Receiver Pre-Identification ....................................................................................... 320
Troubleshooting ........................................................................................................ 321
Java Proxy Runtime ................................................................................................. 322
J2EE Applications .................................................................................................... 323
Java Standalone Applications .................................................................................... 326
Configuration Files .................................................................................................. 327
Error Handling .......................................................................................................... 330
Fault Messages ........................................................................................................ 331
SAP Exchange Infrastructure (BC-XI)

Purpose

SAP Exchange Infrastructure (XI) enables you to implement cross-system business processes. The aim of the Exchange Infrastructure is to integrate different versions of SAP and non-SAP systems implemented on different platforms (Java, ABAP, and so on). The Exchange Infrastructure is based on an open architecture, uses open standards (in particular those from the XML and Java environments) and offers those services that are essential in a heterogeneous and complex system landscape: Namely a runtime infrastructure for message exchange; configuration options for managing collaborative processes and message flow; and options for transforming message contents between the sender and receiver systems.

In this documentation, a collaborative process is a process in the business world that requires cross-system implementation. Note that this term does not describe any concrete objects of this implementation.

Integration

The infrastructure is tailored to general standards so as to remain open for the integration of external systems. At the center of the infrastructure is a message-orientated communication that uses HTTP (Hyper Text Transfer Protocol). The application-specific contents are transferred in messages in user-defined XML (eXtensible Markup Language) schema from the sender to the receiver using the Integration Engine. The structure of a message is therefore determined by the interface data structures used.

The central concept is that, during the design phase, all interfaces required are initially developed independently of a platform and made available in the form of a WSDL description (WSDL: Web Service Description Language). Using this description you can, for example, define mappings between interfaces without this having an effect on an existing system landscape. All design phase data is saved in the Integration Repository to be implemented later in a particular system landscape. In this second phase, configuration time, you can select components, interfaces, and mappings saved in the Integration Repository that are appropriate for your system landscape and collaborative processes and assign them to each other in logical routing. The result of this configuration is saved in the Integration Directory and you can call and evaluate it from the Exchange Infrastructure runtime.

The integration knowledge of a collaborative process is therefore saved centrally in the Integration Repository at design time and in the Integration Directory at configuration time. In this way, SAP Exchange Infrastructure follows the principle of Shared Collaboration Knowledge: Information about a collaborative process need no longer be accessed in each of the systems involved, but called centrally instead. This procedure considerably reduces the costs for the development and maintenance of the shared applications.

Features

Exchange Infrastructure enables you to do the following:

- Develop cross-system applications that exchange a multitude of system messages using the runtime infrastructure and synchronous or asynchronous communication. Develop new platform-independent interfaces or connect existing interfaces to the runtime using adapters.
- Produce a bird’s eye view of the collaborative processes by using business scenarios and derive the interfaces and mappings required from them.
- Adjust message values and structures for the receiver using mappings.
- Centrally maintain the message flow between logical systems in the system landscape using logical routing.
• Connect the logical receiver to a technical system using end points; this system can be exchanged easier using this abstraction level (technical routing).
• Control system access using logon data.
• Describe your system landscape as the foundation for the description of your collaborative processes by using the System Landscape Directory.

Introductory Notes

This documentation is aimed at developers, consultants, and administrators and concentrates on the areas of development, configuration, and runtime. The Architecture [Seite 14] section is directed at all users of SAP Exchange Infrastructure and describes its central concepts. The chapters that follow are interesting for various target groups, depending on whether their emphasis is on development, configuration, or runtime. For an overview of the various tasks in SAP Exchange Infrastructure, see Roles and Tool Access [Seite 24].

Print Versions

You can print individual chapters of this documentation by using the HTML print function in Windows. PDF versions of this documentation (for particular XI releases) are available from the Media Library on the SAP Service Marketplace (see: http://service.sap.com/xi).

Feedback

If you find any errors, incomprehensible sections, or gaps in the documentation, please contact the XI documentation team at the following address:

xi.documentation@sap.com

Do not forget to specify the chapter about which you want to give feedback, if applicable. Please note, however, that this address must not be used for support queries regarding SAP Exchange Infrastructure. If you do have a support query, please contact SAP Support Services (see: http://service.sap.com/support), and specify the component BC-XI.

Additional Information

With a few exceptions, the following subjects are not part of this documentation:

• The installation of SAP Exchange Infrastructure components. For more information on this subject, see the SAP Exchange Infrastructure Installation Guide under http://service.sap.com/instguides.
• SAP Exchange Infrastructure settings to enable the system to function (for example, the configuration of server addresses). For more information on this subject, see the SAP Exchange Infrastructure Configuration Guide under the address above.
## Architecture

Beginning with the runtime components of SAP Exchange Infrastructure (XI), below is an overview of the options available in the Exchange Infrastructure:

- The **Integration Server** [Page 266] is a central communication and ‘distribution engine’ that processes **XML messages** [Page 15]. This includes services for determining receivers (logical and technical routing) and for the transformation of message contents between sender and receiver systems (mapping). The Integration Server can also receive messages from, or send messages to (by using HTTP) other runtime components of SAP Exchange Infrastructure by using communication services. Without these components you would not be able to exchange messages using the Exchange Infrastructure.

- **Adapters** [Page 340] enable you to connect interfaces from external systems or from SAP systems with SAP Basis Release 4.6 or earlier, to the Integration Server. Interfaces involved in the integration scenario can be **imported** into a central **Integration Repository** [Page 19].

- SAP Exchange Infrastructure is a new program model for new developments. This model enables you to **define** platform-independent interface descriptions in the central Integration Repository. You can then use the descriptions to generate interface proxies for Java applications and for SAP systems. In this case, XML messages are exchanged with the **Proxy Runtime** [Page 304] using the Integration Server.

The following graphic illustrates again how systems in a heterogeneous system landscape can communicate with each other using the different runtime components of the Exchange Infrastructure (Integration Server, adapters, proxy runtime):

All systems either send messages to, or receive messages from the Integration Server. In this way, messages can be exchanged between all systems in a heterogeneous system landscape, regardless of whether a message was sent to the Integration Server using an adapter or the proxy runtime.
You can of course also use an adapter for existing interfaces in an SAP system based on SAP Web Application Server 6.20 or higher.

**Involved Systems**

Systems that communicate with each other in the Exchange Infrastructure are called *business systems*; regardless of whether they exchange messages using adapters or the proxy runtime. Refer to this abstract layer of business systems when configuring the logical cross-system process to maintain a degree of abstraction from the technical details (server addresses, ports, and so on).

**Proxy Runtime**

To be able to use proxies you require SAP Exchange Infrastructure Add-On 2.0. This add on enables you to connect the following applications to SAP Exchange Infrastructure by means of proxies:

- Java standalone applications that use synchronous calls to call an XI-connected system.
- Java J2EE applications for SAP Web Application Server 6.20 or higher, including XI 2.0 Add-On.
- ABAP applications for SAP Web Application Server 6.20 or higher, including XI 2.0 Add-On. In this case, a client corresponds to a business system.

**Messages**

The Exchange Infrastructure message format is based on XML. Therefore, this documentation will refer to *XML messages* or simply *messages*.

**XML Properties**

XML (eXtensible Markup Language) enables you to describe data in a highly intelligible form. An *XML Schema Definition* specifies which elements can be used, which attributes these elements have, and how they are structured. More than one *Instance* (a document that matches a XML schema definition) can exist for each schema. The following example of an instance illustrates that the elements in a schema are ordered hierarchically:

```
<PurchaseOrder no="1811">
  <ShipToParty>
    <Name>Rodney Washington</Name>
    <Address>200 S Wacker Drive Chicago IL 60606</Address>
  </ShipToParty>
  <Item>Bass Guitar No.14</Item>
  <Status>Confirmed</Status>
</PurchaseOrder>
```

These elements (for example, `<Item>`) are also known as *Tags* in HTML.
You can describe the structure of a schema using an XML Schema. As well as the description of how the structure of an XML document looks (elements, attributes, hierarchy), this language allows you to define simple and complex data types. Note the following difference:

- **XML Schema Language** provides a series of language constructs that you can use to describe an XML schema.
- **XML Schema Definition** describes exactly one XML schema and is described using the XML Schema language.
- More than one schema instance can exist for an XML schema. A schema instance is an XML document; its structure and values are defined using a corresponding XML schema definition. The process whereby the system checks whether an XML document matches a schema definition is called Validation.

You often also refer to an XML instance or schema instance instead of an XML document. The term XML document is normally used to refer to a document on a file system while the storage medium in the other two terms is less prominent.


**Message Structure**

A message in Exchange Infrastructure is structured as follows:

```
Message

SOAP Header

Message Header

- ErrorHeader
- HopList
- RunTimeHeader
- PerformanceHeader

SOAP-Body

Payload (Main)

Payload (Trace)

- 1.Attachment
- 2.Attachment
- N.Attachment

Message attributes
(For example, sender, mode (synchronous/asynchronous))

Additional header for technical purposes

Business data

Arbitrary attachments (optional)
```

The properties of the current message are contained in the message header (for example, the sender that will be used later to determine the receiver); the actual business data is transferred in the Payload (similar to `<PurchaseOrder>` used in the example above). You can also append an unlimited number of attachments to the message before it is sent. Attachments typically comprise non-XML data, for example, pictures, text documents, or binary data.

Exchange Infrastructure messages are based on the W3C note SOAP Messages with Attachments (http://www.w3.org/TR/SOAP-attachments). Technically speaking, the message is sent as a MIME multipart document.

Using an XML schema you can describe any number of XML schema definitions for the payload of a message. The instance of a schema that is described using an XML schema
definition is sent using the Exchange Infrastructure at runtime. The name of XML schema
definition used is contained in the message header.

**Interface-Based Message Processing**

Exchange Infrastructure messages [Seite 15] are based on XML. How can an application
send a message of this type to a receiver? The idea is similar to a Remote Function Call
(RFC): Communication with another system is encapsulated in an interface; the parameters
of this interface are compiled in a message. The following graphic illustrates schematically
how a message is sent to a receiver using a sender interface:

Multiple receivers can exist, the principle remains the same.

The graphic illustrates that for an interface A in a sender system, there is an assigned
interface B in the receiver system:

- For example, interface B gives you access to a service that determines and returns
customer data for a request.
- The task of interface A is simply to generate a message using values transferred to the
interface; this message is then sent to an assigned receiver interface using the
Integration Server [Seite 23].

The application does not need to implement anything to transfer the parameters to the
message. This is the task of the proxy runtime, which forwards or receives the message. To
send a message, simply call interface A and transfer the parameters for the request message.
You must also implement the receiver processing of interface B on the receiver side.
Interface Types

Of course, the question is where do both interfaces originate from. There are two different types of interfaces:

- **Message Interfaces**: You create these initially in an Integration Repository as XML-based, platform-independent descriptions (see Design Time [Seite 19]). You can use this description to generate interface proxies in one or more systems (on different platforms). The interfaces in the previous graphic were generated using message interfaces.

- **Imported Interfaces**: You import these into the Integration Repository using interfaces in SAP systems. You can include imported interfaces in the Exchange Infrastructure by using adapters [Seite 340]. In this case, the adapter takes over the proxy runtime's task of generating the message and forwarding it to the Integration Server. The previous graphic therefore illustrates the communication between two message interfaces.

The difference between message interfaces and imported interfaces also leads to different program models. For message interfaces you use a platform-independent description to provide interfaces in a system. For imported interfaces, however, you access the system externally. Therefore, the first approach is called Outside-In and the second approach is called Inside-Out. In both cases, the description of an interface in the Integration Repository is separated from the implementation, which has the following advantages:

- Interface information is centrally accessible
- You can use the interface description for application development in the systems and simultaneously for the design of Business Scenarios and mappings

Outbound and Inbound

Interface A and interface B complement themselves in that interface B receives the message that was sent by interface A. To emphasize this difference, interfaces that send messages are called **Outbound Interfaces** and interfaces that receive messages and implement a service are called **Inbound Interfaces**. As indicated by the broken arrows, communication can only comprise the transfer of one message, depending on whether the communication is synchronous or asynchronous (also see: Communication Parameters [Seite 120]).

Outbound and inbound interfaces are also used to separate (potential) senders and receivers. The interface parameters do not need to match (a mapping is then required at runtime). This enables various senders and receivers to communicate with each other. In particular, this loose coupling also enables you to assign interfaces of this type to each other, for which one side of the communication must not, or cannot be changed.

Discussion

Using XML technology has, among others, the following advantages:

- XML is the standard exchange format in the Internet. Before this standard was created, there were practically no open exchange formats, which made communication in heterogeneous landscapes very difficult.
- Further standards and tools now exist that make working with XML even easier, examples being XML Schema, XSLT and XPath. XSLT (eXtensible Stylesheet Language for Transformations), for example, enables you to define mappings required to map message structures. XPath expressions enable conditions to be evaluated depending on values in the XML message. Evaluations of this type are required for receiver identification in logical routing.
- As a standardized format, XML also enables you to connect to external systems. If the data from the external system can be converted to XML using an adapter, then the step to convert the data to other XML formats for receivers is a small one.
Design and Configuration

Regardless of whether you connect interfaces that already exist using adapters, or use the proxy runtime to connect message interfaces saved in the Integration Repository to the Integration Server, the information about the interfaces used in the integration scenario is located in the Integration Repository. The centralized storage of data plays an important role in SAP Exchange Infrastructure. All the information regarding a collaborative process is stored in a central location, instead of being spread across the systems involved. This approach is called Shared Collaboration Knowledge.

Note the following:

- At design time [Seite 19], objects are created in the Integration Repository.
- At configuration time [Seite 21], objects are selected from the Integration Repository and are configured for an actual integration scenario in the Integration Directory.

Basically, the Integration Repository contains potentially useful objects that are initially not dependent on a particular system landscape. The Integration Directory, however, manages the selection of objects actually available in the Repository, as well as their configuration for the particular system landscape.

The information from the Integration Directory is then evaluated on the Integration Server at runtime.

Software Components and Namespaces

The contents of the Integration Repository are part of a software shipment and are therefore assigned a software component version [Seite 58]. You can separate these components further into logical units using namespaces [Seite 59] that are then used to identify the objects. Development at customer and partner sites is also organized using these shipment units.

Exchange Infrastructure and the System Landscape Directory

The System Landscape Directory (SLD) is an SAP product that XI uses to store products, software components, technical system data, and logical business systems. From the point of view of XI, both repository data and directory data is stored in the System Landscape Directory (although the contrary is implied by the name System Landscape Directory): The software components and software products [Seite 460] in the SLD are shipment units and reflect installable and not installed components (repository content), while you store the installed systems in the SLD using a system landscape description. Also see: System Landscape Directory in the Exchange Infrastructure [Seite 255].

Design Time

As described in Interface-Based Processing of Messages [Seite 17], proxies encapsulate the sending and receiving of messages. The conversion of interface parameters to and from a message is done by the proxy runtime (a runtime component in the Exchange Infrastructure (see architecture [Seite 14])).
Message interfaces required for communication must be created by the application, but initially not in the system in which they will later be implemented. It is a central concept of the Exchange Infrastructure that all objects relevant for integration are first created platform-independently in a Integration Repository.

You therefore refer to Interface Development in the Integration Repository with Message Interfaces. To use interfaces that already exist in the system (for example, RFC function modules) as part of the Exchange Infrastructure, you can import the respective interfaces into the Integration Repository and use an appropriate adapter [Seite 340].

For message interfaces in the Integration Repository you can generate proxy objects for a system, as illustrated in the following graphic:

The message interface determines which proxy objects are generated:

- (Java, ABAP Object) classes are generated for outbound interfaces. By calling a method of the class you can transfer your data to the proxy runtime, which constructs the message and forwards it to the Integration Server.

- (Java, ABAP Object) interfaces are generated for inbound interfaces that must be implemented by the application for receiver processing. The proxy runtime calls this processing automatically once it has received an appropriate message.

The proxy classes for J2EE applications are additionally encapsulated by beans.

As illustrated in the graphic, you can generate proxy objects for an interface object in different systems. Only one version of the generated proxy can exist in a system, however.
The generated objects are not client specific if the system is an ABAP application.

You not only save interface objects in the Integration Repository, but also additional objects that are important for the collaborative process.

- You can use a Business Scenario to describe the communication of application components on a higher level of abstraction. The application components can be assigned to the products from the System Landscape Directory here.

- You can create mappings for the messages that are exchanged between interfaces. You can define these mappings as graphical message mappings [Seite 148] or you can import them into the Integration Repository as XSLT or Java archives.

None of the objects in the Integration Repository can be used at runtime still. The repository information is used in the design of a business process without the information entered there having an effect on an installed system landscape. This development phase in which you enter data in the repository is also known as design time.

**Configuration Time**

At configuration time you configure the collaborative processes for an existing system landscape. To do so, select the appropriate objects for the scenario from design time [Seite 19] and link them together logically so that the process can be converted at runtime. This is normally the task of consultants and administrators, while developers make settings just for test purposes. To be able to use the following services of the Integration Server, they must be configured first:

- Logical routing, in other words identifying the receiver business system and the receiver interface
- Technical routing, in other words determining the technical system address for the logical receiver business system
- Optionally, executing a mapping

The following graphic provides you with an overview of the execution of these services on the Integration Server, and the information they require from the Integration Directory:

Only the most essential services are shown to keep the graphic simple. Also, the graphic only returns the logical process flow of message processing. Therefore, the directory data is not read directly from the Integration Directory, but from a cache, for example.

**Logical View of Message Processing**
Using the data from the Integration Repository as a foundation, you must configure the following content to be able to use these services:

- The *routing rules* for receiver identification comprise the following:
  - A *receiver determination* that you use to assign a receiver business system to a sender business system and a sender interface. You can also define additional rules that must be fulfilled for the message to be forwarded.
  - An *interface determination* that you use to assign a sender interface to a receiver interface.

- Therefore, the interface determination determines which sender interface belongs to which receiver interface. If a mapping is to be executed for this interface pair, you can select it from the repository and assign it to the interface determination. Before runtime, the actual mapping program is loaded from the repository and saved on the Integration Server.

- You must create *end points* for technical routing through which the technical receiver of a message is defined (in the case of an SAP system this is an application server). The logical receiver (= business system) is linked to the technical receiver (for example, a system or a URL) using an end point.

To be able to determine receivers and define end points, you must have already maintained the corresponding system data in the System Landscape Directory:

- The business systems for the communication partners and for the runtime components of SAP Exchange Infrastructure (for example, the Integration Server)
- The technical systems assigned to the business systems

By separating these two, it is easier to exchange the technical systems and separate the logical process from the technical administration.
Integration Engine and Integration Server

The Integration Engine [Seite 266] is a runtime component of the Exchange Infrastructure that is implemented in the different business systems in different roles. It comprises the following parts:

- **Sender/Receiver Logic**: Used to receive and forward XML messages from a communication partner.
- **Integration Logic**: Used to refer to the central services of the Integration Engine: Logical routing [Seite 214], technical routing (using end points [Seite 231]), mapping [Seite 143], and adapters [Seite 340]

In addition to these Integration Engine components there is also the application logic. This uses the Exchange Infrastructure to exchange messages in a cross-system business process. The selection of application-specific data or the updating of requests in a business system are included in the application logic, for example.

The integration logic only runs on a SAP Web Application Server Release 6.20 or higher, together with Exchange Infrastructure Add-On 2.0. You must also assign one client in the Integration Engine the role of the central Integration Server:

You do not have to configure client 10 in the Integration Engine as the Integration Server. This is just an example.

In business systems, the Integration Engine is configured in such a way that its task is simply to send and receive messages. The integration logic is executed on the Integration Server.

Sender/receiver logic is only relevant for communication between message interfaces and the Integration Engine, since adapters are used to exchange messages in the case of imported interfaces (see: Architecture [Seite 14]).
Overview of XI Components

Roles and Tool Access

The scope of SAP Exchange Infrastructure functions can be divided into task areas: These task areas correspond to the user roles that the users require to perform the tasks in the respective areas. Following an overview of the task areas, this section includes information about how to access the tools for both Java and ABAP.

Task Areas

Administration

Setting up of the various SAP Exchange Infrastructure tools, system monitoring, user management. An administrator must be involved with the installation process. For more information about installation, see the SAP Exchange Infrastructure Installation Guide under http://service.sap.com/instguides.
Technical Configuration

Contains the configuration of the runtime components of SAP Exchange Infrastructure. This includes the settings for the Integration Engine [Seite 266] and the adapters. This configuration depends on the implemented components in the existing system landscape; the components can be configured independently of the design and configuration of the business process by consultants, administrators or both. For more information about technical installation, see the SAP Exchange Infrastructure Configuration Guide under http://service.sap.com/instguides.

Design and Configuration

Includes the design and configuration of various business processes. This involves the maintenance of all information required for the integration of the applications (business scenarios, interfaces, mapping, routing, business systems). This information is also referred to as integration content.

Monitoring

Includes the monitoring of various business processes at runtime. Monitoring means the monitoring of message flow, the analysis and further processing of non-processed messages, and so on.

The transition from technical configuration (an administrative task) to the configuration of the business process, is continuous. Therefore, the IDoc adapter must access metadata that describes the structure of an IDoc, for example. The business process (configuration) determines which IDocs are to be processed. To access the metadata of an IDoc, there must be an RFC connection to the system that needs to send the IDocs (technical configuration). The latter is the task of an administrator, the former that of a consultant.

User Roles

To access the functions of SAP Exchange Infrastructure, you require the authorizations that are connected to the user roles. They roughly correspond to the task areas described above.

Roles for ABAP and Java Tools

<table>
<thead>
<tr>
<th>Role</th>
<th>Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAP_XI_DISPLAY_USER</td>
<td>Combination of the display authorizations of all tools in Exchange Infrastructure.</td>
</tr>
<tr>
<td>SAP_XI_DEVELOPER</td>
<td>Design of business scenarios, interfaces, mappings, proxy generation, and their implementation.</td>
</tr>
<tr>
<td>SAP_XI_CONFIGURATOR</td>
<td>Make settings for logical and technical routing, determine sender and receiver interfaces by using mapping relations, logon data, maintain business systems and configuration data for the IDoc adapter.</td>
</tr>
<tr>
<td>SAP_XI_CONTENT_ORGANIZER</td>
<td>Organize and structure the contents of the Integration Repository, Integration Directory, and System Landscape Directory, tasks not normally carried out by a developer. For example, maintenance of software components.</td>
</tr>
<tr>
<td>SAP_XI_MONITOR</td>
<td>Monitor XML messages (for example, their throughput), error handling, and status tracking. This also includes the monitoring of processed IDocs and RFCs in the corresponding adapters.</td>
</tr>
<tr>
<td>SAP_XI_ADMINISTRATOR</td>
<td>Administration and technical configuration, see above.</td>
</tr>
</tbody>
</table>

On the ABAP side, the transactions assigned to these roles can be accessed from the user-specific menus.
Technically speaking, the roles specified here are composite roles that each reference a single role for accessing Java applications and a single role for accessing the respective ABAP transactions. Users must only be assigned composite roles. You then have automatic access to the ABAP and the Java tools.

Since user-management is ABAP-based, every user must first log on to the ABAP side to change his or her initial password.

**Accessing Tools**

The tools of the Exchange Infrastructure are implemented both as ABAP transactions and Java applications.

- SAP systems that were installed for the SAP Exchange Infrastructure Add-On provide ABAP transactions for the configuration of the Integration Engine and the IDoc adapter, for the generation of ABAP proxies, and for monitoring. When the user logs on to an SAP system of this type and they have been assigned one of the roles specified above, they can use their personal user menu to access the transactions they require. Alternatively, the user can access the same transactions from the area menu (transaction S_EXCH_INFRA).

- The Java part of SAP Exchange Infrastructure is accessed from a start page. You can access this start page from the respective user menu, from the area menu, and from certain ABAP transactions (Start Integration Builder). Alternatively, you can access it directly by calling transaction SXMB_IFR.

If you know the HTTP address of the Integration Builder, you can also call this link directly in the web browser.

**Integration Builder: Start Page**

Besides being able to access the Integration Builder [Seite 27] from the start page, you can also call the following: The Runtime Workbench [Seite 436], the System Landscape Directory (see also System Landscape Directory in SAP Exchange Infrastructure [Seite 255]), and the Adapter Engine [Seite 362] including the RFC adapter and the MarketSet adapter (if installed).

**Functions of the Integration Builder**

<table>
<thead>
<tr>
<th>Phase</th>
<th>Functions</th>
<th>Link to Start Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design [Seite 19]</td>
<td>Designing Business Scenarios [Seite 77], Interfaces [Seite 117] and Mappings [Seite 143], as well as Java Proxy Generation [Seite 198].</td>
<td>Repository (Design)</td>
</tr>
<tr>
<td>Configuration [Seite 21]</td>
<td>Routing Configuration [Seite 214], Defining Mapping Relations [Seite 226], Defining End Points [Seite 231], Defining Logon Data [Seite 247].</td>
<td>Directory (Configuration)</td>
</tr>
</tbody>
</table>
Design and Configuration Time

SAP Exchange Infrastructure centrally manages the integration knowledge in the Integration Repository and Directory. The Integration Repository gathers together all objects relevant for shipment (see: Design [Seite 76]) while the contents of the Integration Directory must be configured at the customer site for the particular system landscape (see: Configuration [Seite 207]).

Integration Builder

Purpose

The Integration Builder is the central development environment for the following:

- Development of all design objects for the Integration Repository at design time [Seite 76]
- Definition of all configuration objects for the Integration Directory at configuration time [Seite 207]

This section contains information about the user interface and all functions that apply to all design and configuration objects. Functions that refer to a particular design or configuration object are discussed in the design [Seite 76] or configuration [Seite 207] sections respectively, in the description for the object concerned.

Component Overview

Purpose

You can use the Integration Builder to create and edit objects in both the design and configuration phases. Two different tools are available to you during these phases. You can call both tools from the start page of the Integration Builder.

- Design tool
  Integration Builder for the design phase, it can be called from the start page of the Integration Builder by choosing Repository (Design).
- Configuration tool
  Integration Builder for the configuration phase, it can be called from the start page of the Integration Builder by choosing Directory (Configuration).

For more information, see Roles and Tools Access [Seite 24].

The structure of the user interface is the same in both tools. This documentation always uses the term Integration Builder and does not differentiate between the design and configuration tools. If a property refers specifically to the design or the configuration tool, then this is mentioned explicitly. Functions that are specific to design or configuration time are described separately under Functions for Design [Seite 50] and Functions for Configuration [Seite 52].
Properties that refer specifically to a certain design or configuration object are described together with the object under Design [Seite 76] or Configuration [Seite 207].

Structure of the User Interface
The user interface of the Integration Builder consists of the following elements (see graphic).

- **Main Menu and Standard Toolbar [Seite 29]**
- **Navigation Area [Seite 31]**
- **Work Area**

The objects [Seite 35] are displayed in the work area. These editors include functions that refer specifically to the objects opened in the editors.

The following calls design and configuration objects, or **Objects** for short. If a property refers specifically to the design or the configuration tool, then this is specifically mentioned.

Access to Functions
You can call the functions in various ways.

- From the **main menu [Seite 29]**
- Using mnemonics (underlined letters in the menu) or shortcut keys

The shortcut letters and shortcut keys are included in the description of the functions under Main Menu and Standard Toolbar [Seite 29].

For example, you can access the function **Create New Object** with a key combination (Alt + O) → N, or directly with Ctrl + N.
• Using the function keys (F1-F12)
• Using pushbuttons (in the toolbar)
  The toolbar contains the most important functions of the main menu.
• In the object editors [Seite 35]
• From the context menu in the navigation area [Seite 31]

Main Menu and Standard Toolbar

Use

The upper part of the main frame contains the main menu and the standard toolbar.

Features

You can use the main menu to access all functions that relate to the overall framework and navigation area. This section describes these functions.

You can access the main menu with the function key F10. You can use the cursor keys (or the shortcut letters) to navigate in the main menu.

Functions in the Main Menu

As well as a description of the functions, we have also included the shortcut keys and letters.

For example, you can access the function Create New Object with a key combination (Alt + O) \( \rightarrow \) N, or directly with Ctrl + N.

Object (Alt + O)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Shortcut Key</th>
<th>Mnemon. Char.</th>
</tr>
</thead>
<tbody>
<tr>
<td>☐ New</td>
<td>Calls the dialog Create New Object [Seite 40]</td>
<td>Ctrl + N</td>
<td>N</td>
</tr>
<tr>
<td>☐ Save All</td>
<td>Saves all entries in the opened editors</td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>Close All</td>
<td>Closes all editors</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>Importing Software Component Versions</td>
<td>In design mode only: Calls function for importing software component versions (see Importing Software Component Versions [Seite 51])</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>Import XI 2.0 Development Objects</td>
<td>Calls the function for importing integration objects (see Exporting and Importing Integration Objects [Seite 73]).</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>Exit</td>
<td>Closes the application</td>
<td>E</td>
<td></td>
</tr>
</tbody>
</table>

Edit (Alt + E)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Shortcut Key</th>
<th>Mnemon. Char.</th>
</tr>
</thead>
</table>
### Cut
Cuts selected text from an input field
- **Shortcut Key**: (Ctrl + X)
- **Mnemon. Char.**: C

### Copy Text
Copies selected text to an input field
- **Shortcut Key**: (Ctrl + C)
- **Mnemon. Char.**: O

### Paste
Pastes cut or copied text to an input field
- **Shortcut Key**: (Ctrl + V)
- **Mnemon. Char.**: I

### View (Alt + V)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Shortcut Key</th>
<th>Mnemon. Char.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Show/Hide</td>
<td>Shows or hides the navigation area</td>
<td></td>
<td>S</td>
</tr>
<tr>
<td>Navigation Area</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Navigation (Alt + N)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Shortcut Key</th>
<th>Mnemon. Char.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Back</td>
<td>Takes you to the object editor for the previously opened object</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>Forward</td>
<td>Takes you to the object editor for the next opened object</td>
<td>F</td>
<td></td>
</tr>
</tbody>
</table>

### Tools (Alt + T)

Further functions can be accessed from this menu item. The choice of functions varies depending on whether you have called the design or configuration tool.

#### Design and Configuration

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Shortcut Key</th>
<th>Mnemon. Char.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Send System Message ...</td>
<td>Sends a system message</td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>Display System Message</td>
<td>Displays system messages</td>
<td>D</td>
<td></td>
</tr>
</tbody>
</table>

#### Design

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Shortcut Key</th>
<th>Mnemon. Char.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java Proxy Generation</td>
<td>Generates proxies in a software component version (see Java Proxy Generation)</td>
<td>J</td>
<td></td>
</tr>
<tr>
<td>Release Transfer</td>
<td>Calls the release transfer function</td>
<td>R</td>
<td></td>
</tr>
</tbody>
</table>

#### Configuration
You can call the most important functions in the main menu directly from the standard pushbutton as well.

**Navigation Area**

**Use**

The navigation area displays the objects in a tree structure.

**Features**

The navigation area consists of two tab pages:

- The design tool consists of the tab pages *Repository* and *Change Lists*
- The configuration tool consists of the tab pages *Directory* and *Change Lists*

The following functions are available on both tab pages:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Shortcut Key</th>
<th>Mnemon. Char.</th>
</tr>
</thead>
<tbody>
<tr>
<td>🍃 Refresh Selected Node</td>
<td>Refreshes the content of the navigation tree underneath the selected node</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| 🕵️‍♂️ Open/Close Search Entry Field | Opens an input field for a search function  
If the input field is open, you can enter a search term and then choose (Start Search) to begin the search.  
The search begins at the selected point in the navigation tree. If no hits are found, the search starts again from the top structure node. |              |               |
Tab Page Repository (for Design) or Directory (for Configuration)

This tab page shows the design objects or configuration objects in a tree structure.

Objects are always indicated by the icon 🏙. Object **types** are indicated by various icons. These icons are specified in the list of object types (see below).

In the configuration tool, you can navigate to object overviews in the navigation area, as well as to individual objects ( icon). In this case, the navigation tree shows the 🏙 icon.

The structure of the navigation tree is different in the design tree and configuration tree.

Structure in the Design Tree

The design tool (Integration Repository) groups the objects of the navigation tree according to

- Software component
- Software component version
- Namespace

An extra hierarchy level is added under the namespace, which enables you to group the objects according to their meaning. This includes the categories **Business Scenario Objects, Interface Objects and Mapping Objects**.

Only those software components are displayed that were previously imported. For more information, see Importing Software Component Versions [Seite 51].

The following design objects are displayed for each namespace:

- **Under Business Scenario Objects**
  - Business scenarios
  - Actions

- **Under Interface Objects**
  - Message interfaces
  - Message types
  - Fault message types
  - Data Types

- **Under Mapping Objects**
  - Interface Mappings
  - Message Mappings
  - Imported archives

The following objects are also displayed:

- Imported objects
  - RFCs
Note that imported objects are displayed only if you specify that interfaces be imported from SAP systems for the appropriate software component version. This is the case if the checkbox **Interfaces from SAP Systems** is selected in the editor [Display/Change Software Component Versions](Seite 51).

You can restrict the navigation tree display on the tab page **Repository** to one software component, one software component version, or to one namespace by using the filter function. To show/hide the input template for the filter, choose **(Open/Close Filter)** beneath the tab index.

### Structure in the Configuration Tool

There is only 'one' release in the Integration Directory. This means that it is not possible to group the objects by software component version. For this reason, object types are already differentiated between in the top node of the navigation tree. The objects are arranged on the next level of the navigation tree.

One exception is receiver determinations, which are grouped according to business scenarios. Receiver determinations are located in the navigation area one level below the business scenarios, and cannot be seen at the top level.

Configuration objects are identified by multiple key fields. The navigation tree displays the value of one of these key fields for each configuration object.

- **Business scenarios**
  
  The name of the business scenario is displayed.

  - **Receiver Determinations**
    
    Receiver determinations are grouped by business scenario. To determine a receiver the name of the sender business system and the sender interface are displayed as follows:
    
    `<Sender-System> | <Sender-Interface-Name>`

- **Interface Determinations**
  
  The name of the outbound interface is displayed for an interface determination.

- **End Points**
  
  The name of the receiver system is displayed for an end point.

- **Logon Data**
  
  The name of the receiver system is displayed for logon data.

### Functions in the Context Menu (for the design tool and configuration tool)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="Image" alt="Expand Further" /></td>
<td>Only when the cursor is not positioned on an object (or on a business scenario if in the configuration tool): Expands the navigation tree</td>
</tr>
<tr>
<td>Icon</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td><img src="image" alt="Sort in Descending/Ascending Order" /></td>
<td>Only when the cursor is not positioned on an object (or on a business scenario if in the configuration tool): Sorts the entries under the structure node in descending/ascending order</td>
</tr>
<tr>
<td><img src="image" alt="Refresh" /></td>
<td>Only when the cursor is not positioned on an object (or on a business scenario if in the configuration tool): Refreshes the entries below the structure node</td>
</tr>
</tbody>
</table>
| ![New](image) | Calls the [Create New Object](#) function  
- In the design tool when the cursor is positioned on one of the nodes Business Scenario Objects, Interface Objects, or Mapping Objects: Software component version and namespace are already entered in the dialog Create New Object  
- In the configuration tool when the cursor is positioned on one of the following nodes: Business Scenarios, Receiver Determination, Interface Determination for Outbound Interface, End Points for Receiver System, Logon Data for Receiver System. Object type is correspondingly entered in the dialog Create New Object |
| ![Open](image) | Only when the cursor is positioned on a software component version: Calls the function [Display Software Component Version](#)  
- In the design tool: [Display Software Component Version](#) [see Object Editors](#)  
- In the design and configuration tool: Opens the editor Display <Object> (see [Object Editors](#)) |
| ![Delete](image) | Only when the cursor is positioned on a design or configuration object: Deletes the object |
| ![Copy Object](image) | When the cursor is positioned on an object: Calls the [copy object](#) function  
When the cursor is positioned on an imported object (IDoc or RFC) this function is no longer available |
| ![Import RFC/IDOC](image) | Only in the design tool when the cursor is positioned on the node Imported Objects: Imports interfaces from an SAP system (see Importing and Importing Integration Objects [Seite 141]) |
| ![Export](image) | Only in the design tool when the cursor is positioned on a software component version or namespace: Exports the objects of the software component version or the namespace in a file (see Exporting and Importing Integration Objects [Seite 73]). |
| ![Export With/Without Interface Determination](image) | Only in the configuration tool when the cursor is positioned on a business scenario: Exports the business scenario into a file (see Exporting and Importing Integration Objects [Seite 73]). |

You can use the cursor keys to navigate in the navigation area. Use the Enter key to expand a node. You can call the context menu with the key combination Shift + F10.

In the configuration tool, a business scenario is both an object and a grouping category (for receiver determinations). This means that if a business scenario is
selected in the navigation tree, not just the Open, Delete, and Copy functions can be used (as with the other configuration objects), but also the functions for sorting the navigation tree.

**Tab Page Change Lists**

This tab page shows the objects modified by the user in a change list [Seite 46].

**Object Editors**

**Use**

You can use the editors Display/Change <Object> to display and edit design or configuration objects.

**Integration**

The object editors are located in the main screen on the right of the navigation area [Seite 31] and below the main menus and toolbars [Seite 29].

**Prerequisites**

You can call up an object editor by

- creating a new object
  
  For more information, see the Creating a New Object [Seite 40] section.
- selecting the object in the navigation area (on the tab page Repository/Directory or the tab page Change List) and
  
  o double clicking, or
  
  o choosing Open ( ) in the context menu.
- dragging an object from the navigation area to the work area of the main screen and releasing it so that the work area is highlighted with a black frame (Drag & Drop [Seite 47]).

You also have navigation options between the individual object editors of different object types. The descriptions of the individual design or configuration objects give you more information about these options.

For example, you can double click an action in the graphical work area of the business scenario editor to navigate to the action editor.

**Features**

**General Structure of the Object Editors**

All object editors have the same basic structure.

For information on special properties and additional functions that are specific to the object type, see the description of the individual objects.
The object editor comprises the following:

- Object Menu and Toolbar
- Header
- Work Area
- Documentation area (can be hidden)

**Object Menu and Toolbar**

As well as the main menu, the Integration Builder also includes an object menu (and toolbar) in the individual object editors. You can use the object menu to access all functions that relate to the object you have opened. This section describes these functions.

You can access the object menu with the key combination Alt + F10. If you keep the Alt key pressed, you can use the cursor keys (or the shortcut letters listed below) to navigate in the object menu.

You can also use shortcut letters and keys to access functions. We have also included the shortcut keys and letters in the description of the functions.

For example, you can access the function *Switch Between Display/Edit* with a key combination (Alt + Object ID) → A, or directly with Ctrl + E.

You can call up the object menu functions with the following entries.

**<Object>** *(Alt + Object abbreviation, Object abbreviation: see below under Key Combinations for Different Object Types)*

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Shortcut Key</th>
<th>Mnemon. Char.</th>
</tr>
</thead>
<tbody>
<tr>
<td>✅️ Switch Between</td>
<td>Switches from display to change mode, and back again</td>
<td>Ctrl + E</td>
<td>D</td>
</tr>
<tr>
<td>Display/Edit mode, and back again</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------------------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Save</strong></td>
<td>Saves the object</td>
<td>(Ctrl + S)</td>
<td>S</td>
</tr>
<tr>
<td><strong>Delete</strong></td>
<td>Deletes the object</td>
<td>(Ctrl + D)</td>
<td>E</td>
</tr>
<tr>
<td><strong>Check</strong></td>
<td>Checks the correctness of the object</td>
<td>F7</td>
<td>C</td>
</tr>
<tr>
<td><strong>Copy Object</strong></td>
<td>Calls the copy object [Seite 43] function</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td><strong>Properties</strong></td>
<td>Shows the properties of the object (see Displaying Object Properties [Seite 44])</td>
<td>P</td>
<td></td>
</tr>
<tr>
<td><strong>History</strong></td>
<td>Displays the history of an object (see Object History [Seite 45])</td>
<td>H</td>
<td></td>
</tr>
<tr>
<td><strong>Transfer from External Change List</strong></td>
<td>Transfers an object from another change list [Seite 46]</td>
<td>O</td>
<td></td>
</tr>
</tbody>
</table>

Only the functions Properties (ℹ️), Delete (Trash Can), and History are available for imported objects (RFCs and IDocs).

### View (Alt + I)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Shortcut Key</th>
<th>Mnemon. Char.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Show/Hide Header</td>
<td>Shows or hides the editor header</td>
<td>D</td>
<td></td>
</tr>
<tr>
<td>Show/Hide Documentation</td>
<td>Shows or hides the object documentation. The documentation window opens on the right of the editor (see Creating Documentation for an Object [Seite 44]).</td>
<td>C</td>
<td></td>
</tr>
</tbody>
</table>

### Window (Alt + W)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Shortcut Key</th>
<th>Mnemon. Char.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Detach Window</td>
<td>Detaches the object editor from the main window. You can then use the cursor to move the object editor (see Navigation [Seite 39]).</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>Close Window</td>
<td>Closes the editor</td>
<td>(Ctrl + F4)</td>
<td>S</td>
</tr>
</tbody>
</table>

Additional functions are available in the object menu for particular object types. These functions only refer to the respective object type in each case. These additional functions are described in the sections for the individual object types.
Header
The header displays the following information about the object.

- Key information on object
  This information includes the object key defined when the object was created. The type of object determines what exactly is included in the key information. For this reason, this part of the header is described in more detail in the descriptions for the different object types.
- Description
  You can enter a language-specific short description of the object here.
- Status
  The status of the object. The object can have the status Being Edited or Active (see Change Lists [Seite 46]).

Work Area
This area is used for displaying and editing various object properties.
The work area is specially adapted to the individual object type. For more details, see the description of the object type.

Navigation
Object editors appear as tab pages, whose tab index (with the name of the object and the symbol of the object type) is located at the bottom edge of the screen. The tab page design of the object editors gives you additional navigation options (see Navigation [Seite 39]).

Key Combinations for Different Object Types (Menu Entry <Object>)
Key combinations for calling the menu entry <Object> for the design and configuration objects:

Design Objects
- Business Scenario Objects
  - Business scenarios (Alt + B)
  - Actions (Alt + A)
- Interface objects
  - Message interface (Alt + M)
  - Message type (Alt + M)
  - Fault message type (Alt + F)
  - Data type (Alt + D)
- Mapping objects
  - Interface mapping (Alt + I)
  - Message mapping (Alt + M)
  - Imported archive (Alt + I)
- Imported objects
  - RFCs (function modules) (Alt + F)
  - IDocs (Alt + I)

Configuration Objects
- Business scenario (Alt + B)
- Receiver determination (Alt + R)
- Interface determination (Alt + I)
- End point (Alt + D)
- Logon data (Alt + L)

**Navigation Features**

The Integration Builder offers the following navigation options:

- **Navigation in the navigation area**
  - In the navigation area [Seite 31], you can navigate between objects in a tree structure. To call up the object editor [Seite 35] Display/Change <Object> by positioning the cursor on an object icon ( ) and
    - double-clicking, or
    - choosing Open ( ) in the context menu.

- **Navigation between object editors [Seite 35]**
  - You can navigate between the object editors [Seite 35] you have opened by
    - choosing the buttons (Forwards) and (Back) (or Navigation → Forwards or Navigation → Back in the main menu)
      - The commands Forwards and Back always refer to the sequence in which you call the object editors.
    - paging between the object editors you have opened, as with tab pages.
      - The tab index of the tab pages is located on the bottom edge of the object editor. It also shows the name and object type symbol of the object. The tab indexes are another way of navigating at the bottom of the screen.

  If you open too many object editors, and not all tab indexes can be displayed on the bottom edge of the screen, the editors you opened last cover the other editors. Other symbols appear on the bottom right edge, which offer the following navigation options:
    - : Click this icon to display a list of the names of all opened objects. Select a name to put the corresponding object editor on top.
    - : Click this icon to put hidden tab pages on top and display the corresponding object. Click to cancel this action.

- **Navigating forwards between different objects**
  - In an object editor, you can navigate to the object editor of another object by double-clicking the name of an object used (see Object References [Seite 61]).

    In the editor Display/Change Message Interface you can double-click the name of a message type to navigate to the editor Display Message Type.

- **Detaching an object editor window from the main window**
  - Choose Detach Object from Main Window ( ) to detach an object editor window from the main Integration Builder window, or to attach it again. You can then use the cursor
to move the object editor (while pressing the left mouse button). It no longer appears as a tab page.

You cannot use the navigation buttons ( and ) to navigate between detached objects.

Editing Objects

Features

You can use the following functions to edit objects:

- Create New Object [Seite 40]
- Display/Change Object
  You display and change objects in object editors [Seite 35]
- Copy Object [Seite 43]
- Create Documentation for Object [Seite 44]
- Display Object Properties [Seite 44]
- Object History [Seite 45]
- Change Lists [Seite 46]

Changes to objects can be organized in change lists.

Creating a New Object

Use

You can create a new object and then edit it in the object editor [Seite 35].

Prerequisites

A software component version with a namespace must exist before you can create a design object. For information on importing software component versions and creating namespaces, see Importing Software Component Versions [Seite 51].

You can call the function Create New Object, by

- choosing Object → New in the main menu (or the pushbutton Create New Object)

  In this case, you must fill all the fields in the New Object dialog (see below).

- positioning the cursor on an object type in the navigation area and choosing New in the context menu

  In this case, entries have already been made in the Create New Object dialog, according to the position of the cursor (see below).
You cannot create objects of type RFC or IDoc again.

Features

You specify the following information in the Create New Object dialog:

- Object type
- Object key

Object Type

You select the object type on the left of the dialog.

If, in the design or configuration phase, you call the Create New Object function from the context menu in the navigation area, the object type is already selected – according to the position of the cursor in the navigation tree.

For more information on the possible object types, see Navigation Area [Seite 31].

Key Attributes

You define the key attributes for an object on the right of the create dialog. These differ for design objects and configuration objects.

Design Objects

The following fields must always be filled for all design objects:

- Object Name
  
  You must enter the name manually. The name of a design object is unique within a namespace and is not language-specific. Blanks and special characters are not permitted in the name of a design object.

  We recommend that you adhere to Upper Camel Case (UCC) notation when you name design objects. This notation does not use blanks to separate parts of names; instead it only uses a switch from lowercase to uppercase letters (for example, CheckAvailability).

- Object Namespace
  
  You can use search help [Seite 48] here. The available namespaces depend on the selected software component version.

- Software component version
  
  You can use search help [Seite 48] here.

  If you call the Create New Object function in the navigation area in the context menu, then entries are already made in the fields Namespace and Software Component Version, according to the position of the cursor in the navigation tree.
You can call the search help by clicking 📁.

**Configuration Objects**

The key attributes of configuration objects depend on the object type.

**Key Attributes for Configuration Objects**

<table>
<thead>
<tr>
<th>Object Type</th>
<th>Key Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business Scenario</td>
<td>• Business scenario name</td>
</tr>
<tr>
<td>Receiver Determination</td>
<td>• Business scenario name</td>
</tr>
<tr>
<td></td>
<td>• Sender business system</td>
</tr>
<tr>
<td></td>
<td>• Sender interface name</td>
</tr>
<tr>
<td></td>
<td>• Sender interface namespace</td>
</tr>
<tr>
<td>Interface Determination</td>
<td>• Receiver business system</td>
</tr>
<tr>
<td></td>
<td>• Sender business system</td>
</tr>
<tr>
<td></td>
<td>• Sender interface name</td>
</tr>
<tr>
<td></td>
<td>• Sender interface namespace</td>
</tr>
<tr>
<td>End Points</td>
<td>• Receiver business system</td>
</tr>
<tr>
<td></td>
<td>• Receiver interface name</td>
</tr>
<tr>
<td></td>
<td>• Receiver interface namespace</td>
</tr>
<tr>
<td>Logon Data</td>
<td>• Receiver business system</td>
</tr>
<tr>
<td></td>
<td>• Receiver interface name</td>
</tr>
<tr>
<td></td>
<td>• Receiver interface namespace</td>
</tr>
<tr>
<td></td>
<td>• Sender business system (optional)</td>
</tr>
<tr>
<td></td>
<td>• Sender interface name (optional)</td>
</tr>
<tr>
<td></td>
<td>• Sender interface namespace (optional)</td>
</tr>
</tbody>
</table>

When creating interface determinations, end points, and logon data, you have the option of using the wildcard character (*) for some of the key attributes. The object is then defined generically for all values of the particular key attribute (see *Generic/Specific Definition of Configuration Objects [Seite 250]*).

You can use the input help [Seite 48] (icon 📁) when specifying the business systems and interfaces.

After you confirm your entries with Create, the object editor [Seite 35] is opened for this object. The specifications that you made when you created the object are displayed in the header. You can edit the object further in the work area of the object editor.

⚠️

Once you have chosen Create in the Create New Object dialog, you can no longer change your entries.
**Copy Object**

**Use**

You can copy objects to another context, for example, to reuse the original object or parts of the original object in the new context. A new object with the same name is created in the new context that is not dependent on the original object.

The context for design and configuration objects varies:

- The context for design objects is specified by the namespace and the software component version. You can copy design objects to another namespace (of the same, or a different software component version).

  ![Warning] Note that the various object reference types [Seite 62] act differently when copying design objects.

- The context for configuration objects (except for Business Scenarios) is specified by a particular combination of the following key attributes:
  - Business Scenario
  - Sender system
  - Sender interface (name and namespace)
  - Receiver system
  - Receiver interface (name and namespace)

Refer to the Creating a New Object [Seite 40] section of this documentation for information about which combination is intended for the various configuration objects.

![Warning] You must differentiate between the name of the original object and that of the copy when copying a Business Scenario in the Integration Directory.

The copy function is not generally used to transfer all objects of a software component version to a new software component version. To do so, use the release transfer [Seite 69] function.

**Features**

Call the Copy Existing Object dialog by either:

- choosing <Object> → Copy Object ( ) from the object menu in the object editor of the object
- positioning the cursor on the object to be copied in the navigation tree and choosing Copy Object ( ) from the context menu

In the Copy Existing Object dialog, specify the key attributes for the object copy. To do so, use the input help [Seite 48] (icon ).

The default name of the object copy is identical to the name in the original version context. You can edit the name in the Copy Existing Object dialog. Confirm your specifications by choosing Copy. The object copy is created in the new namespace and can be edited there.
Creating Documentation for an Object

Use
You can create documentation for an object.

Prerequisites
You have created an object. To call up the editor for object documentation in the object editor [Seite 35], choose Show/Hide Documentation.

Features
The structure of the object documentation editor differs according to whether you are in display or change mode.

- Display mode displays the object documentation. The documentation is structured by using key blocks. Different key blocks are used for different object types.
- Change mode splits the editor into two areas.
  - You can create the documentation in the upper HTML Editor. You can only enter text for one key block at a time. A dropdown list box at the top of the editor window enables you to switch between key blocks.
  - The preview area shows you all the key blocks. Click in the editor to refresh the preview.

Functions in the Documentation Editor

<table>
<thead>
<tr>
<th>Icon</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="create.png" alt="Create" /></td>
<td>Create: Creates new documentation for an object. You switch to change mode when you create an object.</td>
</tr>
<tr>
<td><img src="change.png" alt="Change Documentation" /></td>
<td>Change Documentation: Switches from display to change mode. The mode of the documentation editor is independent of the mode of the corresponding object editor.</td>
</tr>
<tr>
<td><img src="save.png" alt="Save" /></td>
<td>Save: Saves the documentation</td>
</tr>
<tr>
<td><img src="print.png" alt="Print Documentation" /></td>
<td>Print Documentation: Prints the documentation</td>
</tr>
<tr>
<td><img src="close.png" alt="Close Documentation" /></td>
<td>Close Documentation: Closes the documentation editor</td>
</tr>
</tbody>
</table>

Displaying Object Properties

Choose Properties to display the object properties in the object editor [Seite 35]. The dialog Object Properties then displays the following information about the object:
- Type refers to the object type.
• **Description** specifies the short description that you entered in the header of the object editor.

• **Software Component Version** is specified for design objects only.

• **Status** shows whether the object has the status *Being Edited* or *Active*. For more information, see **Change Lists [Seite 46]**.

• **Person Responsible** displays the user name of the person responsible for the object.

  You can change the content of this field. To do this, choose to switch to change mode.

• **Changed On** displays the date and time of the last change.

• **Changed By** displays the name of the person who last changed the object.

---

### Object History

Each object can have more than one version. A new object version is created each time the object is included in a change list [Seite 46] (for example, when you save for the first time). When the object is activated in the change list, the object version is then closed. If the changes made to an object in the change list are rejected, then the corresponding object version is returned.

⚠️ The object version must not be confused with the software component version to which an object can be assigned.

Choose `<Object> → History` from the object menu in the **object editor [Seite 35]** to display the previous versions for each object.

On the screen **Display History**, the system displays all previous versions of the object, in other words, all versions except for the current object version.

The following information is displayed for each version:

• **User** who made created the object version

• **Date and Time** that the object version was created

• **Procedure**

  The following values are possible for this attribute:
  
  o **Created**
    
    Always indicates the first object version. This is created when you activate for the first time or import.

  o **Change**
    
    Indicates versions in which changes were made to the contents.

  o **Deleted**
    
    Indicates objects that have been deleted. Objects with this version are regarded as no longer being available.

  📝 For example, this situation may occur when multiple users open the same object and one user deletes it (then activates it in the change list).
To open a version, on the Display History screen, select the version and proceed by:

- double clicking, or
- choosing Open Version ( ) in the context menu.

It is possible to open multiple versions simultaneously. To do so, select the versions concerned and choose Open Version ( ) from the context menu. An object editor is opened for each version. The object name and the version are specified on the tab page. The version is also displayed in the Status field in the header of the object editor.

**Change Lists**

**Use**

Changes to objects can be organized in change lists.

**Features**

A change list contains objects currently being edited by the user who is logged on.

An object appears in the change list as soon as it is saved for the first time. When a user activates the change list, all objects in the list change status from Being Edited to Active. These objects can then be seen by all other users. You can also activate individual objects in a change list. When you activate an object, it is removed from the change list again.

If the user edits an object that has already been activated, the object is reentered in the change list when he or she saves.

Provided that an object in a change list has not been activated, all changes made to the object since it was entered in the change list can be undone again. This also applies when deleting an object.

A new object version is created each time an object is entered in a change list. Use the object history function to display all previous versions of an object.

**Displaying and Editing Change Lists in the Navigation Area**

Change lists are displayed in the navigation area of the Integration Builder, on the tab page Change Lists.

- The design tool displays change lists for every software component version.
- The configuration tool, however, only shows one change list.

The change list displays all objects of the user who is logged on that are currently being edited. Each object type is indicated by the appropriate object type icon. Double-click the entry to call the editor Change <Object> (see Object Editors).

**Functions in the Context Menu**
This tab page offers you the same functions for sorting the change list as the tab page Repository/Directory in the navigation area [Seite 31]. Furthermore, you can use the following functions to edit change lists and their objects:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>If the whole change list is selected: Activates the change list. All changes made to objects in the list are activated.</td>
</tr>
<tr>
<td>Activate</td>
<td>If a single object in the change list is selected: Activates the selected object.</td>
</tr>
<tr>
<td>Delete Change List</td>
<td>If the whole change list is selected: Deletes the change list. All changes made to objects in the list are canceled.</td>
</tr>
<tr>
<td>Reject Change</td>
<td>If a single object in the change list is selected: Cancels all changes to the object.</td>
</tr>
<tr>
<td>Open</td>
<td>When a single object is selected in the change list (object type icon): Calls the editor Change &lt;Object&gt; for the selected object. If the documentation icon (ู้) appears next to an object in the change list, double click it to go to the documentation editor [Seite 44].</td>
</tr>
<tr>
<td>Copy Object</td>
<td>When a single object is selected in the change list (object type icon): Calls the copy object [Seite 43] function for this object.</td>
</tr>
</tbody>
</table>

Object documentation is indicated by the icon ู้ in the change list.

You can also transfer an object from a different change list to your own change list, and then edit it yourself. To do this, choose <Object> → Transfer from External Change List in the object editor [Seite 35].

**Additional Functions**

**Use**

This section describes further Integration Builder functions offered by the design and configuration tools.

**Features**

You can choose from the following functions:

- Drag & Drop [Seite 47]
- Search Help [Seite 48]

**Drag & Drop**

**Use**

You can use Drag & Drop to move objects between the navigation area and the object editor.
Features

You can use Drag & Drop to do the following:

- Open object editors
  
  You can call up the object editor [Seite 35] Display <Object> for an object by dragging the object from the navigation area to the work area. To do so, drag the object to the edge of the work area. When this edge is selected and turns black, drop the object.

  This is supported within the Integration Repository or the Integration Directory.

- Create object references within object editors
  
  The object editors of some object types contain Drag & Drop targets. These drop targets are indicated by the hand icon (\(\text{\textcopyright}\)). If object O1 is opened in change mode and object O2 is dragged from the navigation area using Drag & Drop to the hand icon (\(\text{\textcopyright}\)), then object O2 is entered in this location in the object editor of object O1. An object reference [Seite 61] from object O1 to object O2 is then created.

  If you have opened a message interface in change mode, then you can drag a message type from the navigation area using Drag & Drop to the Output Message Type line of the message interface. To do so, drag the message type across to the hand icon in the object editor of the message interface. The message type is entered in the Output Message Type line and an object reference is created to the message type.

  Drag & Drop is also supported between the Integration Repository and the Integration Directory.

  You can drag an interface object from the navigation area of the Integration Repository to the “Hand” icon in the Inbound Interface column in the Change Interface Determination editor in the Integration Directory, by using Drag & Drop. This creates an object reference from a configuration object (interface determination) in the Integration Directory to a design object (message interface) in the Integration Repository.

Search Help

Use

Search help can be used in multiple locations in the Integration Builder. This section describes the properties of the search help that apply to both design and configuration objects.

Features

You can use search help in the following instances:

- In the Create a New Object [Seite 40] dialog
- In the object editors [Seite 35]
You can call search help by clicking the icon.

**Design Objects**

- Creating design objects
  
  Search help for namespaces and software component versions is available here. It displays a navigation tree with all software component versions and namespaces.
  
  - Selecting a software component version restricts the selection of the namespace to this software component version.
  
  - Selecting a namespace defines the software component version.

  If you call up the function *Create New Object* in the context menu of the Integration Builder navigation area, then the namespace (and software component version) is already entered (see Creating a New Object [Seite 40]).

- Other search helps in the object editors

  - Search help for interfaces: This search help displays a navigation tree in which you can select a message interface. You see only a section of the navigation tree, according to the context in which you call the search help.

    If you call the search help for message interfaces in the action editor, then only message interfaces (and not the other design objects) are displayed in the search help navigation tree. If the action belongs to an application component of type *Product*, then the selection of message interfaces is restricted to the software component version to which the action belongs. This is because in this case the action and the message interface must be located in the same software component version.

    If you call the search help for message types in the editor for a message interface, then the software component version is already selected.

    - If a namespace is already specified (from the context), only a list of objects (for example, interface objects with name and namespace) is displayed.

**Configuration Objects**

- Creating configuration objects

  When you create configuration objects, you can use a search help in the dialog *Create New Object* [Seite 40] for the following fields:

  - Business System
    
    You can select a business system from a list. All business systems that are defined in the System Landscape Directory are displayed.

  - Interface (Name and Namespace)
    
    You must first select a business system before you can call the search help for interfaces.

    The software component version of the interface is only defined once you have selected a business system.
In the *Select Interfaces* dialog you can choose whether to restrict the selection of interfaces to the selected business system, or whether to display all interfaces. Select a radio button accordingly.

If, in the *Create New Object* dialog, you predefine the *Interface Name* field or the *Interface Namespace* field, then you can restrict the selection in the search help to a particular character string.

For example, by entering `Acc*` for the interface name you restrict the selection to all interfaces whose names begin with the string `Acc`. The selection conditions are displayed in the frame *Search Criteria* in the *Select Interfaces* dialog.

The name, namespace, software component version, direction (inbound/outbound), and mode (synchronous/asynchronous) are displayed in a table for each interface.

- When you create a receiver determination, you can select the business scenario name from a list.

The object editors include search helps for the following objects:

- Search help for business system (for receiver determination)
- Search help for interface (for receiver determination and interface determination) and interface mapping (for interface determination)

### Special Design/Configuration Functions

#### Use

This section describes functions that are specific to either design or configuration.

#### Features

- [Design Functions](#)
- [Configuration Functions](#)

### Design Functions

#### Use

You can call special design time functions from the main menu of the Integration Builder (Design).

#### Features

You can choose from the following functions:

- Importing software component versions
- Java proxy generation
- Release Transfer

In the navigation area, you can call the following function in the context menu when you position the cursor on a software component version:
• Displaying/Changing Software Component Versions [Seite 51]
  In particular, you can use this function to create namespaces.

Importing Software Component Versions

Use
You require software component versions when you create namespaces. Therefore, they are a prerequisite when you create design objects. You import all software component versions that you require to define design objects in the Integration Builder, from the component description [Seite 445].

Procedure
1. In the main menu, choose Object → Import Software Component Version. The screen Import Software Component Versions displays the software component versions from the SLD.
2. Select one or more software component versions that you want to import. Select the corresponding check boxes.
3. Confirm your selections by choosing Import.
4. Proceed as described in Display/Change Software Component Versions [Seite 51].

Displaying/Changing Software Component Versions

Prerequisites
You have the option of displaying and changing an imported software component version. In doing so you can create namespaces.

Procedure
The editor for software component versions comprises a header and a work area.

Header
The header displays the name of the software components, the version, the status, and a description.

Work Area
The work area consists of two tab pages:

Tab Page Definition
In this tab page you can do the following:

- Define whether the import of RFC and IDoc interfaces from SAP systems is permitted or not
  Select the corresponding check box.
  If you choose this option, RFCs and IDocs of this software component version are imported. These objects are then displayed under Imported Objects in the navigation area [Seite 31].
• Specify namespaces for the software component version
  You must specify namespaces so that you can create design objects for this software component version. To create a namespace, choose Insert Line Below Selection ( ). Next, enter the namespace and choose Save ( ).

• Delete namespaces
  To delete a namespace, select the corresponding line of the table and choose Delete Selected Line ( ). It is also possible to delete multiple namespaces.

  !

  Ensure that namespaces do not contain any design objects before deleting them.

Tab Page Key
This tab page displays the key information about this software component version that is stored in the System Landscape Directory.
The following information is displayed about software component versions:
• GUID
• Element type key
• Vendor
• Name
• Version

Configuration Functions

Use
You can call special configuration time functions from the main menu of the Integration Builder (Configuration).

Features
You can choose from the following functions:

• Value Mapping [Seite 53]
• Identify Endpoints Automatically [Seite 245]
• Clear SLD Cache [Seite 55]
• Define End Point as Web Service [Seite 246]
• Integration Builder (Design): Calls the design tool of the Integration Builder.
The Integration Engine system also offers you another important tool for the configuration phase:

• Directory Cache [Seite 254]
Value Mapping

Use
An object can be represented in different ways in different contexts.

For example, a user Julie Moore (object) can be a customer in business system A, but an employee in business system B.

You can use the value mapping function to map different representations of an object to each other. You can store the mapping rules for different objects in a value mapping table.

The value mapping functions are used in the following locations in the SAP Exchange Infrastructure:

- Java mapping
  For more information, see Executing Value Mappings [Seite 184].

- Routing
  When identifying a receiver from message contents, the value mapping function maps a payload field to a receiver business system. For more information, see Defining Receivers [Seite 218] and Specifying the Extractor Parameters [Seite 223].

Structure of the Value Mapping Table
The value mapping table then looks as follows (see graphic).

<table>
<thead>
<tr>
<th>Group 1 (contains various representations of object “Mr Glenn”)</th>
</tr>
</thead>
<tbody>
<tr>
<td>System</td>
</tr>
<tr>
<td>--------</td>
</tr>
<tr>
<td>A</td>
</tr>
<tr>
<td>B</td>
</tr>
<tr>
<td>C</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Group 2 (contains various representations of object “Mr Rodriguez”)</th>
</tr>
</thead>
<tbody>
<tr>
<td>System</td>
</tr>
<tr>
<td>--------</td>
</tr>
<tr>
<td>A</td>
</tr>
<tr>
<td>B</td>
</tr>
<tr>
<td>B</td>
</tr>
<tr>
<td>C</td>
</tr>
</tbody>
</table>

A value mapping table contains different representations of objects.

The representations of the same objects are listed together in groups. A group of entries in the value mapping table contains multiple representations of the same object. One representation of an object takes up a single table line.

A group contains at least one representation.
The example (see the graphic) groups the different representations of the object Mr. Glenn in group 1, and the different representations of the object Mr. Rodriguez in group 2.

The fields of the value mapping table mean the following:

- **System**
  The field *System* indicates the context. Generally, it is a business system.

- **Namespace**
  Specifies the namespace of the object type.

- **Object Type**
  Specifies a ‘role’ in which the object can appear in different contexts.

- **Value**
  Identifies the representation of the object in the group.

**Features**

The Integration Builder (Configuration) contains the central access point for displaying and changing the value mapping table. To do this, choose *Tools → Value Mapping* in the main menu.

The editor in which you display and change the value mapping table is constructed thus.

The upper area of the editor has two groups of input fields. Here you can select the following fields from the value mapping table for two representations of an object in each case:

- **System**
- **Object Type**
- **Namespace**

To call the search help for these fields, choose 📚.

![Warning icon]

Remember that you can only select representations from one group.

The lower area of the editor is where you can complete the entries of the value mapping table. To do so, enter values for the selection made in the upper area (column *Value For*).

**Functions**

<table>
<thead>
<tr>
<th>Icon</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Insert icon]</td>
<td>Inserts a new line</td>
</tr>
<tr>
<td>![Delete icon]</td>
<td>Deletes the selected entry</td>
</tr>
<tr>
<td>![Switch icon]</td>
<td>Switches between display and change mode</td>
</tr>
<tr>
<td>![Save icon]</td>
<td>Saves the entries</td>
</tr>
</tbody>
</table>
Clearing the SLD Cache

Many actions performed during configuration time require you to access System Landscape Directory content from the Integration Builder. To optimize performance, this content is loaded into a cache so that the Integration Builder does not have access the System Landscape Directory directly each time that content is required.

However, this cache is not automatically updated if changes are made to the content of the System Landscape Directory. For this reason, SAP recommends that you clear the System Landscape Directory cache if changes have been made to content in the System Landscape Directory. The cache is then filled each time that the System Landscape Directory is accessed.

Search help is offered when configuring receiver determinations. To optimize performance, the search help function accesses the cache instead of accessing the System Landscape Directory directly. When you create a business system in the System Landscape Directory, the information about the business system is not automatically transferred to the System Landscape Directory cache. For this reason, this business system is not displayed when the search help is called. Therefore, SAP recommends that you clear the System Landscape Directory cache so that the most up-to-date data from the System Landscape Directory is loaded into the cache.

To empty the SLD cache, from the Integration Builder main menu (configuration tool), choose Tools → Empty SLD Cache.

Software Logistics

Purpose

The functions and tools for software logistics as part of SAP Exchange Infrastructure ensure that objects can be versioned, moved (within the Integration Repository or between systems), and shipped. Since the contents of the Integration Directory can be configured at the customer site, this essentially affects objects from the Integration Repository. Nevertheless, you can also use software logistic components for configuration contents from the Integration Directory.

Introductory Notes

A differentiation is made between two types of development objects in the development of applications between different application components:

- Objects from the Integration Repository that are used to design and develop the exchange of messages at design time (Business Scenarios, interfaces, mappings)
- Objects in the systems of the various application components that communicate with each other by using SAP Exchange Infrastructure. This is where the actual application logic is implemented, for example, what happens when a message arrives. The proxy objects that are generated in these systems also belong to these objects.

You can use the software logistics functions of SAP Exchange Infrastructure to manage and ship the various versions of your repository objects. All objects that are developed in the application component systems must be shipped using their infrastructure. In particular, you must ensure that the objects of your application from the Integration Repository are shipped together with the objects of the application system.
Features

Those functions that you can use for objects in the repository or directory are indicated by a green tick (√) in the Design and Configuration column in the table below (since configuration objects in the directory are not shipped, only some of the functions are required):

Software Logistics Functions for Repository and Directory Objects

<table>
<thead>
<tr>
<th>Function</th>
<th>Design (Repository)</th>
<th>Configuration (Directory)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Organization of the shipment content [Seite 56] by using software component versions and namespaces</td>
<td>√</td>
<td></td>
</tr>
<tr>
<td>Version management [Seite 66] on the level of objects and software component versions</td>
<td>√</td>
<td>(✓) (only object versioning)</td>
</tr>
<tr>
<td>Transfer objects to another software component version (Release Transfer [Seite 69])</td>
<td>√</td>
<td></td>
</tr>
<tr>
<td>Exporting and Importing Integration Objects [Seite 73]</td>
<td>√</td>
<td>√</td>
</tr>
</tbody>
</table>

Restrictions

At present there is no tool support for transporting objects. Instead, you can import the objects into another system once you have exported them, by copying or moving a file that you have previously exported.

Organization of Shipment Contents

From the point of view of software logistics, a product is a shipment unit that can exist in different versions. A product comprises one or more software components [Seite 58] that in turn can also have different software component versions. The System Landscape Directory stores the information about which version of a product requires which software component versions (also see: Software Components and Software Products [Seite 460]). Software component versions and product version are located in the Integration Builder:
In this example, the product version SAP APO 1.1 comprises the software component versions BW 1.2B and APO 1.1. The corresponding information about the product version can be found in the System Landscape Directory. To now be able to edit software component versions, they are imported into the Integration Repository and are then displayed in the navigation tree of the Integration Builder. The graphic shows the expanded hierarchy for software component version APO 1.1. The development of objects in the Integration Builder is therefore primarily linked to software component versions.

After you have imported a software component version, create one or more namespaces for it so as to separate the repository objects within the software component versions into more manageable development units. Note that the object references within the Integration Repository are restricted to ensure that objects that belong together are consistently shipped together.

The product version is not visible in the navigation tree, unlike the software component version. This is because the software component versions that make up a product version have already been defined in the System Landscape Directory. However, you can create a reference to product versions when describing a Business Scenario. When creating an application component (as a communication partner) you can assign a product version to it.
Software Component Versions

Definition
A software component version is a shipment unit for repository objects. A software component version can be used in different product versions.

You can access SAP software components (and products) from the software catalog of the System Landscape Directory (see: Software Catalog [Seite 460]).

It is also possible to structure software component versions using other software component versions.

Use
To use software component versions you must first import them into the Integration Builder from the System Landscape Directory (see: Importing Software Component Versions [Seite 51]). A software component version is only relevant for the Integration Repository and not for the Integration Directory because the contents of the latter is not shipped but configured at the customer site.

Structure
When you have imported a software component version, the Key tab page displays attributes from the System Landscape Directory. You can use the following XI-specific attributes on the Definition tab page:

- Interface Import
  Here you can define whether the import of RFCs and IDocs is permitted for the software component version. If it is, then you must specify an SAP system from which these interfaces are to be imported. You can only specify one such SAP system for each software component version.
  See: Connection with Adapters and Imported Interfaces [Seite 140]

- Namespaces
  Using namespaces you can further subdivide the software component version into semantic units. These are required to create objects in the repository (in other words, Business Scenarios, message interfaces, and mappings).
  See: Namespaces [Seite 59], Design Time [Seite 19]

- Object Properties
  Use these two attributes to define the properties for all objects of the software component version:
  a. Objects Are Original Objects:
     The objects of the software component version are the original objects in the repository belonging to the Integration Builder, and should only be modified in this repository. There can only be one original object, in other words, you must not set this attribute for multiple software component versions of other repositories simultaneously.
  b. Objects Are Modifiable:
     This attribute enables you to change objects in the software component version when set. Use this attribute to lock all objects against changes, if required.

  Also see: Versioning in Transports [Seite 71]
The latter two attributes are typically used as follows:

<table>
<thead>
<tr>
<th>Attribute Set</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a) and (b) set</td>
<td>The objects in this repository are the originals and can be modified there. Use this setting for the repository in which the cross-system process is being developed.</td>
</tr>
<tr>
<td>(a) and (b) not set</td>
<td>The objects in this repository are not the originals and cannot therefore be modified here. Use this setting for repositories where the originals from a different repository have been imported. Also see: Exporting and Importing Integration Objects [Seite 73].</td>
</tr>
<tr>
<td>Only (a) set</td>
<td>The original objects are temporarily locked against modifications, for example to ‘freeze’ the development status.</td>
</tr>
<tr>
<td>Only (b) set</td>
<td>Normally you do not change objects in a repository where the objects are the not the originals. However, use this attribute for temporary changes, for example when you do not want to wait for the next import before making an emergency correction. These changes are lost when the next import takes place (due to the conflict resolution function).</td>
</tr>
</tbody>
</table>

**Integration**

The navigation tree in the Integration Builder then provides you with an overview of all software component versions that use integration objects within cross-system applications. To ensure that objects that belong together within a software component version are shipped together, the Integration Builder restricts the references that are possible between objects in the Integration Repository (also see: Object References [Seite 61]).

---

**Namespaces**

**Definition**

Namespaces in the Integration Builder are namespaces in the sense of XML namespaces that are sub-divided further into semantic units within the Integration Builder software component versions. Objects of a namespace can only be shipped as part of the corresponding software component version and not separately.

⚠️ The SLD also works with namespaces but these have nothing to do with the namespaces of the Integration Builder (see: Namespace [Seite 470]).

**Use**

**General**

Each object for a namespace has a unique name within this namespace. An object is identified uniquely by a namespace + object name.

Basically, a namespace in XML can be any constant value. Note the following two conventions:
Conventions for XML Namespaces

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>http://</td>
<td>When you specify an HTTP address as a namespace this does not mean that you can access additional information about the namespace from this address (in some cases an XML schema definition is hidden behind the address). The HTTP address used merely ensures that the name of the namespace is unique. SAP namespaces adhere to the standard <a href="http://sap.com/xi/">http://sap.com/xi/</a>&lt;Name&gt;.</td>
</tr>
<tr>
<td>urn:</td>
<td>Namespaces with this prefix have no value in the sense that they are not connected with a web page in the Internet.</td>
</tr>
</tbody>
</table>

It is assumed that conventions ensure that namespaces are unique.

For further information about namespaces, go to www.w3.org/TR/REC-xml-names/.

**Arbitrary Namespaces**

To create objects (mappings, interfaces, Business Scenarios) in the Integration Builder, you must create at least one namespace for an imported software component version. Each software component version can have multiple namespaces. A namespace is normally assigned to multiple software component versions, but only one software component.

> You cannot rename namespaces. It is possible to copy the objects of one namespace to a different namespace, however relative object references must be changed by hand for each object. (Also see: Object References [Seite 61]).

**Predefined Namespaces**

Besides the namespaces you create yourself, there are also some predefined namespaces. The table below contains some examples:

<table>
<thead>
<tr>
<th>Namespace</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>urn:sap-com:document:sap:idoc:messages</td>
<td>Namespace for imported IDoc interfaces</td>
</tr>
<tr>
<td><a href="http://sap.com/exchange/MessageFormat">http://sap.com/exchange/MessageFormat</a></td>
<td>Internal namespace for fields in a message header</td>
</tr>
</tbody>
</table>

**Integration**

Whether objects from one namespace can reference objects from another namespace depends on the type of object and on the software component versions that the namespace is assigned to. See: Object References [Seite 61].

You usually transfer namespaces for one software component version to the subsequent software component version. However, you can delete empty namespaces.
Object References

The Integration Repository differentiates between the following:

- Independent repository objects. These objects are visible in the navigation tree and are assigned to a namespace.
- Objects assigned beneath the independent repository objects, for example a connection in a Business Scenario.

During the design phase you constantly need to reference existing repository objects, for example from a message mapping to message types. Note the following general restrictions:

- It is only possible to reference independent repository objects.
- Whether you can reference or not depends on the software component that the referencing repository object is located in (see below)
- Whether referenced objects are copied when you copy an object depends on the type of reference [See Seite 62].

The reason behind these last two restrictions is that you must be able to ensure that all the repository objects of one product are delivered together.

Referencing Objects

You create each referencing object in the context of a software component version. It is identified by using the software component version, the namespace, and the technical name.

Business Scenario Objects

Software component versions are assigned to a product version. It is then possible to reference other objects in the same, or in other software component versions, regardless of the object type. This restriction is reflected in the value help in the Integration Builder.

<table>
<thead>
<tr>
<th>Object Type</th>
<th>Possible References</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business Scenario</td>
<td>Since Business Scenarios provides an overview of the communication between application components that are each assigned to a product, they can reference any product versions.</td>
</tr>
<tr>
<td>Action</td>
<td>An action can reference <em>Interfaces</em> of the software component version in which the action was created.</td>
</tr>
</tbody>
</table>

It is also possible to reference non-independent objects within a Business Scenario.

For more information see: Business Scenario [Seite 78].

Interface objects

Message interfaces reference (fault) message types, message types reference data types, and data types can in turn reference other data types. You can only reference interface objects in the same software component version or in a sub software component version. This ensures that objects for a message interface are always in one shipment unit.

Mapping objects

Mappings map messages from different application components to each other. Therefore, referencing cannot be as restrictive as for interfaces. The additional restrictions for interface mappings ensure that mapping programs are shipped together with the interface mapping.
### Interface Mapping
- You can reference interfaces from any software component version.
- You can reference mapping programs (message mappings, XSLT or Java mappings) from the same namespace and the same software component version as the interface mapping.

### Message Mapping
You can reference any message types or the request/response/fault part of an imported interface.

It is also possible to reference non-independent objects. See: [Message Mappings](#) and [References Between Mapping Programs](#).

### Imported archives
You can reference other mapping programs (see: [References Between Mapping Programs](#)) but not independent objects.

---

### Object Reference Types

#### Object Reference Types for Design Objects

There are three different types of object references for design objects. The semantic conditions of the object types involved determines which object references are used. This cannot be altered by the user. Nevertheless, it is important to note the differences between the individual object reference types because they can behave differently when you [copy an object](#).

- **Absolute object reference**
  The software component version of the referenced object is **not dependent** on the software component version of the referencing object.

  ![Image](https://example.com/image.png)

  The software component version of an action that is used in a [Business Scenario](#) within an application component of type *Product*, is based on the product version (the application component) and is not dependent on the software component version of the Business Scenario.

  ![Warning](https://example.com/warning.png)

  After it has been copied to a new software component version, the referencing object references an object in the original software component version. Therefore, the object reference is not damaged by the copying procedure because the referenced object already exists in the original software component version.

- **Object reference to an object in the most recent software component version**
  The software component version of the referenced object is **not dependent** on the software component version of the referencing object. The software component version of the referenced object is however **not** fixed (unlike for an absolute object reference). The object reference always refers to the object in the most recent software component version that contains the object.
An interface mapping references an RFC that is located in the software component SWC. Software component SWC also has the software component versions SWCV1, SWCV2, SWCV3. However, the RFC only exists in the software component versions SWCV1 and SWCV2 because it was only imported there. When navigating to the RFC, the object from software component version SWCV2 is displayed.

- **Relative object reference**

  The software component version of the **referred to** object is **identical** to the software component version of the **referring** object. This also applies when the referred object is located in a software component version on which the software component version of the referring object is structured.

  The software component version of a data type that is used in a message type is the same as the software component version of the message interface, because both objects are delivered together.

  After it has been copied to a new software component version SWCV2, the **referring** object references an object in the same software component version SWCV2. The version context of the referred object is determined by the version context (in other words, the software component version) of the referring object. Since the referred object is not copied as well, the object reference initially cannot be followed. If, in the corresponding object editor in the Integration Builder, you try to navigate to the referred object, the system displays a message to this effect. So that the object reference becomes valid again, the referred object must be copied with the same name to the new software component version SWCV2 as well. The namespace of the copied referred object must be the same as the namespace of the object that was originally referenced.

All object references that occur for design objects are listed and categorized in the table below.

### Object References for Design Objects

<table>
<thead>
<tr>
<th>Referencing Object</th>
<th>Referenced Object</th>
<th>Object Reference Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business Scenario</td>
<td>Action (in an application component of type <strong>Product</strong>)</td>
<td>Absolute object reference</td>
</tr>
<tr>
<td>Business Scenario</td>
<td>Action (in an application component of type <strong>Template</strong>)</td>
<td>Relative object reference</td>
</tr>
<tr>
<td>Business Scenario</td>
<td>Message interface (within a connection if the application component is of type <strong>Product</strong>)</td>
<td>Absolute object reference</td>
</tr>
<tr>
<td>Business Scenario</td>
<td>Message interface (within a connection if the application component is of type <strong>Template</strong>)</td>
<td>Relative object reference</td>
</tr>
<tr>
<td>Business Scenario</td>
<td>Interface mapping (within a connection)</td>
<td>Absolute object reference</td>
</tr>
<tr>
<td>Action</td>
<td>Message Interface</td>
<td>Relative object reference</td>
</tr>
<tr>
<td>Message Interface</td>
<td>Message Type</td>
<td>Relative object reference</td>
</tr>
<tr>
<td>Message Interface</td>
<td>Fault Message Type</td>
<td>Relative object reference</td>
</tr>
<tr>
<td>-------------------</td>
<td>--------------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>Message Type</td>
<td>Data Type</td>
<td>Relative object reference</td>
</tr>
<tr>
<td>Fault Message Type</td>
<td>Data Type</td>
<td>Relative object reference</td>
</tr>
<tr>
<td>Interface Mapping</td>
<td>Message Interface</td>
<td>Object reference to an object in the most recent software component version</td>
</tr>
<tr>
<td>Interface Mapping</td>
<td>RFC, IDoc</td>
<td>Object reference to an object in the most recent software component version</td>
</tr>
<tr>
<td>Interface Mapping</td>
<td>Fault Message Type</td>
<td>Object reference to an object in the most recent software component version</td>
</tr>
<tr>
<td>Interface Mapping</td>
<td>Mapping Program</td>
<td>Relative object reference</td>
</tr>
<tr>
<td>Message Mapping</td>
<td>Message Type</td>
<td>Absolute object reference</td>
</tr>
</tbody>
</table>

If the software component of the message mapping is the same as the software component of the message type, then the absolute object reference behaves as a relative object reference. This means that the reference to the message type is often lost when copying a message mapping.

### Examples

<table>
<thead>
<tr>
<th>Referencing Object</th>
<th>Referenced Object</th>
<th>Object Reference Type</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business Scenario</td>
<td>Action (in an application component of type Product)</td>
<td>Absolute object reference</td>
<td>The software component version of an action within an application component of type Product, is based on the product version and is not dependent on the software component version of the business scenario (see Business Scenario [Seite 78]).</td>
</tr>
<tr>
<td>Business Scenario</td>
<td>Action (in an application component of type Template )</td>
<td>Relative object reference</td>
<td>Actions within an application component of type Template must be located in the same software component version as the Business Scenario (not necessary in the same namespace) (see Business Scenario [Seite 78]).</td>
</tr>
<tr>
<td>Business Scenario</td>
<td>Interface mapping (within a connection)</td>
<td>Absolute object reference</td>
<td>The software component version of the interface mapping is not dependent on the software component version of the Business Scenario (not necessary in the same namespace) (see Business Scenario [Seite 78]).</td>
</tr>
</tbody>
</table>
### Development Cycle in the Integration Repository

#### Purpose

The exemplary process portrayed here shows which steps are required technically speaking in order for development to be able to begin with design in the Integration Repository. It also explains the shipment options and the options for transferring to a new version.

#### Process Flow

1. Before development begins, software component version must be available in the System Landscape Directory and assigned to a product version. This is normally carried out by a development manager.
   - At SAP, software component versions and products are imported into the System Landscape Directory from the PPMS system.
   - Customers can create their own software component versions and products in the System Landscape Directory (see: Maintaining Third-Party Products [Seite 462]).

2. When development begins, the development manager or project manager imports the software component version from the System Landscape Directory into the Integration Repository and defines the namespaces for development. Furthermore, for each imported software component version he or she can define a system from which RFCs or IDocs can be imported (see: Importing Software Component Versions [Seite 51]).

3. With the namespaces now available, development can begin with the design of the business processes in the repository. Changes are saved in user-specific change lists [Seite 48]. To make these changes visible for all users of a repository, the developer must either activate the entire change list or individual objects.

4. To ship the objects in the repository, you must export them in a file. You may want to export the objects so as to import them into a test system, for example.
5. When developing a new version of the product, you normally increase the version number of the software component and the product from step one. By using release transfer [Seite 69], you can transfer objects from other software component versions into the new software component version.

6. Note the information contained in the Compatibility of Changes [Seite 67] section of this documentation when developing the new version.

**Version Management**

**Use**

Version management in the Integration Repository permits versioning as follows:

- The Integration Builder manages multiple versions of a software component in a repository. In this way, different product versions can communicate with each other. Each object is created in the context of a software component version that represents a unit of a product that can be shipped.

- Objects can also have new object versions when changes are made within a software component version. You also have the option of releasing changes to multiple objects simultaneously.

Versioning in the Integration Repository ensures that objects are shipped consistently, and as part of a product.

However, there are no software component versions in the Integration Directory, because the configuration contents is not shipped. Nevertheless, you can also release changes to the configuration for the entire runtime environment here. When you release the objects, the Integration Server updates the directory runtime cache.

**Integration**

You can also export objects of an Integration Repository or an Integration Directory to import them into another repository or directory (for example, during a system relocation). This takes into account versioning of the corresponding software component version.

For more information see: Exporting and Importing Integration Objects [Seite 73]

**Features**

**Change Lists**

The Integration Builder supports object versioning for both the repository and for the directory using the user-specific Change Lists. When an object is saved for the first time, a new object version is created that then is transferred to the change list. When an object in the change list is activated, the object version is closed and is made visible for other users.

For more information see: Change Lists [Seite 46].

**Products and Software Component Versions**

A product can have multiple versions. Each product version is a shipment unit visible for customers. The software component versions used in a product version can be called in the System Landscape Directory.

For more information see: Software Catalog [Seite 460].
In the context of SAP Exchange Infrastructure, the products and software component versions that are of interest are those that are to exchange messages with each other. When development starts they must be imported from the System Landscape Directory into the Integration Builder.

For more information see: Importing Software Component Versions [Seite 51].

Release Transfer

In the transition from one software component version to a new software component version, you can either transfer all, or just some of the objects from the previous version. This release transfer also enables you to transfer objects to older software component versions.

For more information see: Release Transfer [Seite 69].

Note the information contained in the Compatibility of Changes [Seite 67] section of this documentation when developing a new software component version.

Compatibility of Changes

Wherever possible, changes or product enhancements should be compatible with the previous version. In other words, a customer must be able to upgrade to a new version of a product without any difficulty. Incompatible changes are those that render the program code of the previous version in some way invalid. The following section provides an overview of compatible changes to SAP Exchange Infrastructure objects.

Interface objects

For two systems to communicate there must be an outbound and an inbound interface. You must ensure that any change to either interface does not mean that the counterpart must be changed as well in order that the systems can communicate with each other. The following options for compatible changes exist:

- Adding technical and semantically optional attributes or elements to the referenced complex data types. Semantically optional means that a field is not expected by a receiver although it is flagged as optional.
- Communication partners can of course agree on changes. If a new version of a product depends on changes in another product, then the customer must be informed correspondingly (for example, new product A, version 2.0 requires at least product B, version 3.5).

Generally you check for an object that has had incompatible changes made to it whether it is referenced by other objects and whether you should inform the “owner” of the object about the changes (no where-used list function exists at present).

Mappings

You must only make compatible changes to a mapping between two application components for technical reasons. Compatible means that the mapping must function for any combination of subsequent versions of both application components. Therefore, the only changes possible
to a mapping in different software component versions are corrections. If you need to make incompatible changes, you must create a new mapping.

A mapping between the application components CRM 1.0 and APO 1.0 must also function for the following combinations: CRM 2.0 with APO 2.0, CRM 1.0 with APO 2.0 and CRM 2.0 with APO 1.0 (see: Ensuring the Compatibility of Mappings [Seite 68]).

If a mapping comprises multiple mapping programs that are executed one after the other, then they must be compatible for each transitional step.

The mapping programs M1 and M2 are used for the following overall mapping: A (M1) B (M2) C. It is not sufficient when the mapping is only compatible for the mapping from A to C. The mapping from A to B and from B to C must also be compatible.

Ensuring the Compatibility of Mappings

Introduction

You must only make compatible changes to a mapping between two application components. If you need to make incompatible changes, you must create a new mapping. This requirement is relatively easy to fulfill since only optional elements and attributes are normally added to a message in a new release.

Example

A mapping is created based on messages from CRM 2.0 and APO 2.0. However, messages from other releases are also mapped or the message created runs in target systems with differing releases:
The section below discusses how you can ensure that the mapping to CRM 2.0 can process all inbound messages without causing any compatibility problems in the various target systems.

**Inbound Messages at the Integration Server**

**Processing a message from CRM 1.0 using the CRM 2.0 mapping**

Since the only differences between CRM 1.0 and CRM 2.0 are optional elements or attributes, this message could have also come from a CRM 2.0 system that did not include these optional fields. The mapping should therefore be able to map the message. This can be guaranteed if:

- Optional fields in the source message are only mapped to optional fields in the target message
- Any obligatory target fields missing from the source are completed with default values (for example, this can be done for a message mapping by using the standard function `exists` [Seite 164])

**Processing a message from CRM 3.0 using the CRM 2.0 mapping**

The additional elements or attributes are ignored. This can be ensured if the mapping identifies the source fields by using explicit path specifications (for example, "/OrderHeader/ShipTo"), instead of "counting" the number of fields. Do not use any of the following methods:

- Absolute positioning (for example, the third element in the first structure).
- Include the frequency that fields occur (for example, "if there are ten elements in the sub structure then...")

**Receiving Source Messages in the Target Systems**

**Receiving the message in an APO 1.0 system**

The proxies ignore any additional fields so this does not pose a problem.

**Receiving the message in an APO 3.0 system**

APO 3.0 contains additional fields that are not part of APO 2.0. Since these are optional fields, they do not pose a problem.

---

**Release Transfer**

**Use**

Using the release transfer function, you can transfer objects within an Integration Repository to other software component versions:

- From a current software component version to the next version, when development in the current version is finished
- From one software component version to an older version of the same software component, for example to make a correction in an older version available.
- From a software component version to a version of another software component. For example, this may be necessary because you may want to stop objects of software
components that are based on each other from being available in the sub-software component, and only make them available in the superior software component.

**Features**

In the release transfer you can transfer the following:

- All objects of a software component version
- All objects of a particular namespace of a software component version
- Individual objects of a software component version

**Version Conflicts**

As a rule, you transfer all objects of a software component version or a namespace once. Development then continues in the new software component version. You can transfer individual objects to the new version at a later stage if corrections need to be made in the old version that must then be transferred to the new version. If one of these objects has already been changed in the new version, then the release transfer reports a version conflict. You can then display a list of those objects that caused the conflict and exclude them from the transfer process as required.

To conclude, a version conflict only occurs during the release transfer if an object has been changed in both software component versions. If an object has only been changed in one of the software component versions, then the release transfer function takes the object from the most recent version.

**Activities**

Perform the following preparatory steps prior to beginning the release transfer process:

1. The new software component version that you want to transfer the objects to must be entered in the System Landscape Directory (SLD). SAP software component versions are imported from PPMS. The customer can create their own in the System Landscape Directory.

2. Import the new software component version into the Integration Repository (see: Importing Software Component Versions [Seite 51]).

3. To be able to transfer objects to the imported software component version you must first assign namespaces to it. The namespaces must have the same names as those namespaces from which you want to transfer the objects.

To actually transfer the repository objects, proceed as follows:


2. **Enter the following in the Release Transfer window:**
   - Using the search help for **Software Component Version**, specify the release transfer source and target.
   - Under **Object Selection**, define which objects specified under **Source** are to be transferred.

3. Choose **Transfer.**
Versioning in Transports

General

Objects are transported within SAP Exchange Infrastructure for the following reasons:

- To ship objects from the Integration Repository
- To test a development in the repository or a directory configuration separately
  - At present there is no infrastructure for the actual transport of objects. Instead you can export objects from one repository/directory to import them into another repository/directory (see: Exporting and Importing Integration Objects [Seite 73]).

You must move exported files manually to a corresponding import directory. For reasons of simplicity, this process is referred to as a transport in this section.

Transport Logistics

Integration Repository

Repository objects have one original repository, in other words a repository from which an object originates. Within a repository you can differentiate between original objects and copies by using an attribute of the corresponding software component version.

Due to the originality principle, transport landscapes for repositories are star-shaped:

You should only make changes to an object in the Integration Repository from which the object originates. The new versions of this object are then created when it is imported into the target repository so that the object has the same version in both repositories (see graphic):
As shown here, object versions in the various repositories are consistent. This is ensured by the following mechanisms:

- You should lock objects in the target repository against entries (see Object Properties in Software Component Versions [Seite 58]). Nevertheless, you do have the option of temporarily changing objects in the target repository, but this can lead to a conflict when you import the objects. You can choose between two import variants when importing:
  - *Import and Do Not Resolve Conflicts:* If objects being imported were changed in the target directory, then import is canceled and an error message is displayed.
  - *Import and Resolve Conflicts:* Objects changed in the target repository are overwritten during the import.

- When you import older object versions to a target repository, all new object versions that already exist there are not overwritten. The older version being imported is visible in the object history [Seite 45] following the import. The more recent version remains the most recent version.

This means that multiple imports into a repository can lead to the same end result, regardless of the sequence in which they are imported.

**Integration Directory**

At configuration time you have the option of testing the configuration in a test directory. If the tests are successful, you can then transport the configuration objects in the test directory to the directory for the productive landscape.

Therefore, at configuration time, transports are not used to ship objects, but to test the configuration separately from the productive landscape before it is used productively as part of SAP Exchange Infrastructure.

- There are no software component versions in the Integration Directory. The objects in the Integration Directory are instead linked to Business Scenarios and can be exported using Business Scenarios as well.

- No large transport landscape with multiple directories is required. As a rule, one test directory and one productive directory is sufficient. There is also a test repository and a productive repository.

The configuration objects in the Integration Directory reference objects in the Integration Repository. Therefore, you must transport the objects from the test repository to the productive repository before you import the objects into the productive directory.
In the test scenarios at configuration time, it is not always the case that the most recent version from the test directory is transported to the productive directory. Often an older version is the more appropriate version. Therefore, unlike when importing in the Integration Repository, when importing objects to a directory, version management always creates a new version for the objects:

In the example, the initial version V1 of an object is exported from the test directory. Since a new version ID is created when you import objects to the productive directory (here V3’ for V1), you can import version V1 to the productive directory despite the fact that the more recent versions V2 and V3 have already been imported.

**Importing and Exporting Integration Objects**

**Use**

Importing and exporting integration objects enables you to do the following:

- Copy objects from one Integration Repository to another Integration Repository. The software component version must be the same in the source and target repository in this case.
- Copy objects from one Integration Directory to another Integration Directory. In this way, you can copy a test configuration to a productive landscape.

Use the release transfer [Seite 69] function to transfer objects within an Integration Repository from one software component version to another.

**Prerequisites**

During the export, a binary file is created in a directory defined on the repository server (or directory server). To import this file to another repository (or directory), you must manually copy it to an import directory (see below). You require the appropriate authorizations to be able to access directories on the SAP Web Application Server.

**Import and Export Directories**

<table>
<thead>
<tr>
<th>Type</th>
<th>Integration Repository</th>
<th>Integration Directory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Export</td>
<td><code>&lt;systemdir&gt;/xi/repository_server/export</code></td>
<td><code>&lt;systemdir&gt;/xi/directory_server/export</code></td>
</tr>
</tbody>
</table>
### Import

| <systemdir>/xi/repository_server/import | <systemdir>/xi/directory_server/import |

*<systemdir> is the system directory from the server installation.*

### Features

#### Integration Repository

You can either export all objects of a namespace, or all objects of a software component version. The export file also contains information about higher-level software components when exporting objects of a namespace, because the version of the software component of the object to be imported must be checked when you import.

If you have only exported the objects of a namespace, the namespaces of the same software component version are nevertheless visible in the target repository following the import. However, these namespaces are empty because they could not be exported. SAP recommends that you only export complete software component versions so as to avoid confusion.

During the import, version management checks whether the objects to be imported already exist in the target repository, and if so, what their version status is (see: [Versioning in Transports](Seite 71)).

#### Integration Directory

Directory objects are assigned to business scenarios: A receiver determination references interface determinations, end points and logon data. You can export just the receiver determination or the referenced interface determination as well. End points and logon data are specific to the technical system and must therefore be entered manually after the export.

Unlike importing in the Integration Repository, version management always creates a new version for the object to be imported in the target directory. Object versions of the source and target repository are therefore independent of each other. This means that you can also import configuration objects that have a version that is older than the version currently in the directory.

### Activities

#### Importing and Exporting Repository Objects

1. To import or export repository objects, call the design maintenance screen of the Integration Builder.
2. Call the context menu for a namespace or a software component version in the navigation tree and choose Export….
3. Confirm that you want to export in the dialog that is then displayed.

The Integration Builder saves an export file with the suffix `tpt` in the export directory of the repository server (see above). When exporting IDocs or RFC schema from the repository, there is also a file with the suffix `_classic.tpt` in the export directory.

---

*Do not change the file name of the export file. If you do, the Integration Builder will not accept it as the appropriate file when you import. Furthermore, note that this file is a binary file.*

4. To import the export file(s) to another Integration Repository, first copy or move it to the import directory of the target repository.
5. Call the design maintenance screen of the Integration Builder for the target repository. Choose **Object → Import XI 2.0 Development Objects...**

6. Select the export file(s) saved in the import directory by using the dialog that is then displayed.

If the import is successful, the export files are moved to the sub-directory
<serverdir>/xi/repository_server/importedFiles.

**Importing and Exporting Directory Objects**

**Prerequisites**

Test and productive directories both access the same System Landscape Directory. To import configuration objects into the productive directory from the test directory, you must define transport targets in the System Landscape Directory. In the process, business systems of the test directory are assigned to business systems of the productive directory.

See also: [Configuring Groups and Transport Targets](Seite 263).

Objects in the Integration Directory reference objects in the Integration Repository. Therefore, you must import repository objects into the productive repository before you can import them into the productive directory.

**Procedure**

1. To import or export directory objects, call the configuration maintenance screen of the Integration Builder.

2. Call the context menu for the business scenario in the navigation tree and choose **Export With/Without Interface Determinations**.

   The Integration Builder saves an export file with the suffix **tpt** in the export directory of the directory server (see above).

   **Do not change the file name of the export file. If you do, the Integration Builder will not accept it as the appropriate file when you import. Furthermore, note that this file is a binary file.**

3. To import the export file(s) to another Integration Directory, first copy or move it to the import directory of the target directory.

4. Call the configuration maintenance screen of the Integration Builder for the target directory. Choose **Object → Import XI 2.0 Configuration Objects...**

5. Select the export file saved in the import directory by using the dialog that is then displayed.

   If the import is successful, the export file is moved to the sub-directory
   <serverdir>/xi/directory_server/importedFiles. The objects are not activated when they are imported into the target directory. They are visible in the change list of the user who imported them.

6. Check the imported objects in the change list and adjust the configuration data to match your system landscape, if necessary.

7. Activate the change list that contains the imported objects.
Design

Purpose
The development of a cross-business business process begins with its design (see: Design Time [Seite 19]). The Integration Builder [Seite 27] provides you with an environment with which you can describe Business Scenarios, interfaces and mappings independently of a system landscape. These objects are stored in the Integration Repository and are associated with software component versions that belong to a product that is to be shipped (see: Shipment and Versioning [Seite 55]).

Prerequisites
Software component versions are the smallest delivery units that can be shipped using the objects of the Integration Repository. To be able to develop within a software component version, they must be maintained in the software catalog of the System Landscape Directory (SLD). SAP products and the corresponding software component versions are already maintained in the SLD. It is also possible to include external products and software component (versions) in the SLD (see: Software Catalog [Seite 460]).

Process Flow
The following steps are usually carried out in the design phase:

1. To save objects in the Integration Repository, you import the software component versions in which the objects are developed (see: Importing Software Component Versions [Seite 51]). The development manager normally has the authorization required for this.

2. To document the business process on a more abstract level, create a Business Scenario. This can be done regardless of whether interfaces and mappings are already available or not. If objects are available, you can reference them directly. If they aren’t, you can add further information once you have created them. A Business Scenario helps you to understand the entire process (see: Designing Business Scenarios [Seite 77]).

3. To define the contents of a message and the communication type, create new message interfaces in the Integration Repository or import existing interfaces from another system (see: Interface Design [Seite 117]). If interfaces do already exist, you can simultaneously design mappings and develop in the business systems.

4. To map the structure of a message to another message, use a message mapping or import an XSLT or Java archive. You can then specify for which messages a mapping is required in two interfaces, by using an interface mapping (see: Designing a Mapping [Seite 143]).

5. To create executable substitutes for message interfaces, use Proxy Generation [Seite 198]. The proxies you generate are used at runtime (see: Proxy Runtime [Seite 304]).

Result
The objects in the Integration Repository are assigned to software component versions and can therefore be shipped with the product associated with the latter. Before they can be implemented in a system landscape at runtime, you must first configure the contents of the Integration Repository (see: Configuration [Seite 207]).
Designing Business Scenarios

Purpose

A Business Scenario enables you to define the message exchange and process flow for collaborative business processes. These processes do not need to be controlled centrally and only require simple routings [Seite 214] and mappings [Seite 143] to function. Normally, these are collaborative business processes between various business partners that are generally loosely coupled and synchronized.

The business scenario provides you with a central point of access for all integration objects such as interfaces, mappings, software components and so on, which you require for semantic and technical integration.

Note the following about the information compiled during the design phase of a business scenario:

- It is used during the configuration phase [Seite 207] to map the collaborative process to the current productive system landscape.
  
  In this phase you specify the business systems of the business partners on which the application components are installed. In this phase the Business Scenario provides you with the central point of entry for the definition of receiver determinations. These routing relations define the flow of messages in the productive system landscape. For more information, see Configuration [Seite 207].

- It is used in productive operation to ease the execution and administration of collaborative processes.

  Business scenarios make the dependencies between the business partners and the separation of concerns transparent and gather information that can be used for the purposes of monitoring, for example. As a result, the entire collaborative process is easier to maintain, more reliable and more stable.

Features

The Integration Builder constitutes the development environment for the design of business scenarios. You can do the following with the Integration Builder:

- Define the business partners involved in the business scenario and their application components
- Model all interactions for which messages are exchanged between different application components
- Specify each individual message exchange by assigning them interfaces and mappings

The development environment for business scenarios comprises two editors:

- **Business Scenario Editor [Seite 91]**
  
  The business scenario editor provides an interactive graphical work area that enables you to visualize the business scenario process and how the objects used are related to each other. From the context within this editor you can navigate to the design environment of all objects used. The top down and bottom up design of business scenarios is supported.

- **Action Editor [Seite 87]**
  
  The action editor enables you to display and edit actions. Actions are activities within the business scenario process that are not subdivided further.
This section is structured around the following questions:

- What is a business scenario and how do the various components work together? For more information, see Business Scenario [Seite 78].
- How do I develop a business scenario with the Integration Builder? For more information, see Business Scenario Design Environment [Seite 82].
- How do I model a business scenario? For more information, see Guidelines for Modeling Business Scenarios [Seite 107].

**Business Scenario**

**Definition**

A business scenario completely models the exchange of messages for a collaborative process and provides an overview of the process flow.

**Structure**

A business scenario is a design object of the Integration Builder, is uniquely identified by a name and a namespace, and is subject to versioning.

To model the exchange of messages and the process flow, the business scenario comprises the objects described below. The graphic illustrates how these objects are related using the Business Scenario example *Booking a Flight Trip*.

**Business Scenario Booking a Flight Trip and Objects Used**
**Application Component**

An application component is a modeling construct within a business scenario. From a logical point of view, it represents a participant of a business scenario. This contains the business tasks and concerns that this participant represents within the business scenario.

Application components are represented with colored columns.

An application component can stand for a business partner or a component of a business partner. At runtime it would be either one or more business systems.

A product determines the tasks of an application component. Products are versioned. Examples of product versions are **SAP APO 3.0A** and **SAP R/3 4.6C**. For further information about product versions, see *Product Versions and Software Component Versions* below.

The product of an application component can be either of the following:

- A product defined in the **System Landscape Directory**
- A template product when no corresponding product is defined in the System Landscape Directory or it is not specified in more detail at the time of design.

With the latter you have the option of producing a Business Scenario, the implementation of which is unknown or is not defined in the System Landscape Directory (for example, for non-SAP products).

Application components are also described further by specifying a role. This role helps to specify the range of tasks of the application component on a business level. Examples of roles of different business partners are customers and vendors.

**Action**

An action represents a function within an application component that is not subdivided further. Actions subdivide the process flow of the business scenario.

Actions are shown as white rectangles. The vertical progression of actions from top to bottom corresponds to the logical process flow.

An action is an independent design object of the Integration Builder. This means that actions are defined independently of business scenarios. Thus you have the option of reusing actions in different business scenarios.

Note the following for actions: They are used in either of the following types of application components:

- Type *Product* (defined in the System Landscape Directory)
- Type *Template* (not defined in the System Landscape Directory)

For more information, see the *modeling actions* section.

All interfaces associated with an action are assigned to that action in the Integration Builder.

The exchange of messages between components takes place between two actions from different application components. The interfaces used in the message exchange are assigned to the two actions involved.

**Interfaces and Mapping**

Interfaces define the data format of the messages that are used when messages are exchanged between actions from different application components. Furthermore, the properties of the interfaces involved determine whether communication takes place synchronously or asynchronously.
For more information on interfaces, see the Interface-Based Processing of Messages section in this documentation.

If mappings exist between outbound and inbound interfaces, then you can select a mapping for the business scenario.

**Connection**

A connection is a link between two actions within a business scenario. The following different types of connections exist:

**Sequence**

Sequence represents the sequence of two actions within the same application component. A sequence is required for portraying the Business Scenario process, however it contains no additional information.

**Cross-Component Connection**

A cross-component connection connects actions from different application components with each other. These actions exchange messages with each other in a business scenario. A cross-component connection defines the interfaces and mappings used.

Cross-component connections differentiate between synchronous and asynchronous communication.

A connection is not an object that can be reused. It is only defined within a specific business scenario.

**Beginning and End of the Process**

The beginning and end of a process run can be defined for a business scenario. Selected actions can be defined as start actions and end actions for this purpose. There can be more than one start or end action.

**WSCI Interface**

All descriptions of business scenarios within the Integration Builder conform to WSCI (Web Service Choreography Interface) to create a basis for the standardized exchange of business scenarios. A WSCI interface represents all the inbound and outbound channels (including dependencies) of an application component that is involved in a business scenario. Therefore, each application component has a corresponding WSCI interface that needs to be named accordingly.

For further information about WSCI, see:


**Product Versions and Software Component Versions**

Product versions and software component versions are defined in the System Landscape Directory. They play a central role in the definition of business scenarios. Product versions are software units that can be installed on your system and that run independently of other software. (Example: SAP R/3 4.6C). Product versions generally comprise multiple software component versions. Software component versions are independent development units at SAP that cannot function on their own and are not shipped as such. (Examples: SAP BASIS 4.6C, SAP APPL 4.6C, SAP HR 4.6C).

Also see:

- The concepts introduced here are illustrated in Example of a Business Scenario.
- For detailed information about the semantic concepts, see Modeling Business Scenarios.
Example of a Business Scenario

This section explains the concept of the business scenario by using an example.

Business Scenario Booking a Flight Trip

The business scenario Booking a Flight supports a travel agency as it sells a flight trip to a customer. The travel agency needs to work together with the corresponding airlines to carry out the flight bookings required.

The business scenario has two logical participants, namely the travel agency and the airline.

The travel agency is represented by the application component on the left. The travel agency tasks are carried out by the product SAP Web Application Server (Release 6.20). In other words, the product version is SAP Web Application Server 6.20.

The airlines are represented by the application component on the right. Since during the design phase you do not know what product version an airline uses, a template is used in the Business Scenario. The expected functions of the airline are modeled (specified) using this template but are not connected to a particular product version.

Two steps have to be performed by the travel agency in this business scenario, namely searching for suitable flight connections and the booking of the flight trip. The corresponding actions for these steps in the travel agency are Search for Flight Connection and Book Flight Trip. To execute these actions, the travel agency must work together with the corresponding
The airline requires information about the availability of single flights and it wants to make reservations and bookings for particular flights.

The airline has three actions to execute namely Determine Flight Availability, Reserve Single Flight, and Book Single Flight. These actions provide the functions required by the travel agency.

The business scenario graphic displays the process flow of the scenario as well as the points of data exchange between the travel agency and the airline:

1. The business scenario begins with a travel agency employee searching for a suitable flight connection.
2. To determine the availability of a single flight, messages are exchanged with the airline. This is symbolized by a connection between the actions Search for Flight Connection and Determine Flight Availability. The communication is synchronous since the answer is required immediately. In the business scenario, this is represented by the fact that both actions are located horizontally on the same level.
3. The next step is for the travel agency to book the flight trip. The connection (sequence) between the actions Search for Flight Connection and Book Flight Trip implies that the first action is a prerequisite for the second action and that the actions must be executed in this order.
4. The action Book Flight Trip comprises two sub steps. Firstly, messages are exchanged with the airline to reserve seats on the various flights. This is symbolized by the connection between the actions Book Flight Trip and Reserve Single Flight. The communication type is synchronous since the travel agency expects a response to confirm that the reservation was successful.
5. If the reservation was successful, the final booking can be carried out. This requires messages to be exchanged with the airline once again. This is symbolized by the connection between the actions Book Flight Trip and Book Single Flights. In this instance, the communication type is asynchronous. In the business scenario, this is represented by a downwards-pointing arrow.
6. For a booking to be successful, the airline requires there to have been a successful reservation made first. This is represented by the connection (sequence) between the actions Reserve Single Flight and Book Single Flight.
7. The business scenario is complete once the single flights have been booked successfully with the airline.

All actions involved must have the corresponding interfaces available to exchange messages. Furthermore, it is also possible that mappings be executed. Information about interfaces and mappings is not contained in this Business Scenario example.

### Business Scenario Design Environment

**Purpose**

In the Business Scenario design environment of the Integration Builder, you can display and change business scenarios.

**Integration**

This section contains information about the object editors for the Business Scenario objects Business Scenario and Action, including all functions and procedures associated with them.

**Features**

The Business Scenario design environment contains two editors:
In the navigation area, Business Scenarios and actions are put in order of software component versions and namespace, and displayed under Business Scenario Objects. Note that in the navigation tree, actions are grouped under their software component version as independent design objects. This means that not all actions that are used in a Business Scenario are displayed in the same position in the navigation tree as in the Business Scenario.

This section is structured as follows:

- **Quick Start: Designing a Business Scenario** gives a step-by-step guide to creating a simple Business Scenario for the first time.
- **Action Editor** explains how to use the action editor.
- **Business Scenario Editor** explains how to use the Business Scenario editor.
- **Handling Messages** describes what problems can occur and how best to resolve them.

---

**Quick Start: Designing a Business Scenario**

**Use**

This section provides you with a step-by-step guide to defining a simple Business Scenario. In the process, you will learn about the graphical modeling options available, how to insert application components, how to create actions, and how to define synchronous and asynchronous connections.
Prerequisites

- Software component version
  Select a software component version in the navigation area of the Integration Builder.
  You can do the following:
  - Use an existing software component version
  - Import a software component version from the System Landscape Directory [Seite 51]

- A namespace in the software component version
  You can do either of the following:
  - Use an existing namespace
  - Create a namespace [Seite 51]

Procedure

Creating the business scenario and inserting the application component

1. Expand the sub node of the selected namespace in the navigation area of the Integration Builder.
2. Select the Business Scenario node and choose New (New) from the context menu.
3. In the Create New Object dialog, enter the name of the business scenario (FlightTripBooking) and confirm by choosing Create. The new business scenario is opened in change mode.
4. In the field Description, enter Book Flight Trip.
5. To insert an application component for the travel agency, position the cursor on the graphical area and choose Insert Application Component from the context menu. The system displays the Insert Application Component dialog.

6. Select the radio button Template (Not Defined in System Landscape Directory) in the frame Application Component Type. Application components of type Template can be used if you do not want to specify a particular product specification.

7. In the frame Further Attributes, enter Travel Agency as the name of the role (field Role Name).

8. Confirm your entries by choosing ✔. The application component for the travel agency then appears as a colored vertical column in the graphical display area of the Business Scenario editor.

9. To insert the Airline application component, position the cursor on the graphical display area to the right of the travel agency application component and repeat steps 6 and 7. Enter Airline as the name of the role of the application component. You have now defined the application components required in the business scenario.

**Defining Actions**

1. Select the Actions node in the navigation area for the selected namespace and choose New (>Create) from the context menu.

2. In the Create New Object dialog, enter the name of the action (BookFlightTrip) and confirm by choosing Create. The new action is opened in change mode.

3. In the field Description, enter Book Flight Trip.

4. In the Use group, choose the option Templates (Not Defined in System Landscape Directory) and save your entries.

   You can only use the new action in the travel agency and airline application components you created if you select this option.

   The new action is created in the same software component version in which you created the business scenario.

5. To create additional actions, repeat steps 1-4.

   - Action 2: Name ReserveSingleFlight; description Reserve Single Flight
   - Action 3: Name PerformBookingForSingleFlight, description Book Single Flight

   Save your entries each time. Unsaved actions cannot be used in the business scenario.

**Inserting actions and defining connections in the business scenario**

1. Navigate to the business scenario.

2. To insert an action in the Travel Agency application component, in the graphical work area of the Business Scenario editor, position the cursor on the colored column of the application component and choose Insert Action from the context menu.

   In the Select Action dialog, select BookFlightTrip from the list of actions. The selected action is inserted in the application component. To move the action within the application component, use Drag & Drop by holding down the left mouse button. Move the action to the top of the application component column.

3. Next, insert ReserveSingleFlight at the top of the airline application component column, followed by PerformBookingForSingleFlight.
4. Lay out the actions as shown in the graphic above.

5. To create a synchronous connection between *Book Flight Trip* and *Reserve Single Flight*, proceed as follows:

   Select both actions using the rubber band and then choose *Create Connection Between Selected Actions* from the context menu. The system calls the dialog for creating connections.

   ![Rubberband Example]

   For more information about rubberbands and the other graphical functions of the Business Scenario editor, see the Functions of the Graphical Editor [Seite 102] section.

   The relative position of the two actions means that the type of connection is automatically defined as *synchronous*. You do not need to make any further specifications. When you choose ✅, the dialog closes and a line is inserted between the selected actions to indicate synchronous communication.

6. To create an asynchronous connection between *Book Flight Trip* and *Book Single Flight*, proceed as follows:

   Select the two actions as described in step 5. The relative position of the two actions means that the type of connection is automatically defined as *asynchronous*.

7. To create a sequence between *Reserve Single Flight* and *Book Single Flight*, proceed as follows:

   Select the two actions and insert a connection. The relative position of the two actions means that the type of connection is automatically defined as *Sequence*.

8. To define start and end actions, proceed as follows:

   Position the cursor on the *Book Flight Trip* action, and choose *Define Action as Start Action* from the context menu. The system inserts a start point above the action. Next, position the cursor on the *Book Single Flight* action, and choose *Define Action as End Action* from the context menu. The system inserts an end point beneath the action.

### Defining interfaces and creating an example connection

1. In the graphical work area of the Business Scenario editor, double click the *Book Flight Trip* action. The system calls the action editor. If the editor is not already in change mode, switch to change mode by choosing *Switch Between Display and Edit Modes* (🛠).

2. In the *Outbound Interfaces* frame, choose *Insert Empty Line Below Selected Line* (🛠). Enter *FlightBookingReserveOut* as the name of the interface in the name field. Leave the namespace field on the right empty.

3. You must now generate your new interface. To do so, position the cursor on the name field, and choose *New* (🛠) from the context menu. They system calls the dialog for creating new objects. The object type *Message Interface* has already been selected. Select a namespace for the message interface and confirm your specifications by choosing *Create*.

4. The system calls the *Message Interface: Change* dialog. Enter *Reserve Single Flight* in the *Description* field. In the *Attribute* frame, choose *Outbound* for the direction and *Synchronous* for the mode. Save the interface.
Note that the interface is still not completely defined and cannot therefore be activated in the change list [Seite 46].

5. Using the tab pages at the bottom of the screen, navigate back to the action editor and save the changed Book Flight Trip action.

6. Navigate to the business scenario editor and double click the Reserve Single Flight action. Following the procedure described in step 2, create a new Inbound Interface in this action with the name FlightBookingReserveIn.

7. Create the interface in the Message Interface: Change editor as described in step 3, but this time enter Reserve Single Flight in the Description field and specify the direction as Inbound. The mode is already set to synchronous. Save the interface and the Reserve Single Flight action.

8. Navigate to the Business Scenario editor and open the connection editor for the synchronous connection by double clicking the connection line between the Book Flight Trip and Reserve Single Flight actions. The following interfaces are displayed at the bottom of the editor:
   - Outbound interface created in the Book Flight Trip action
   - Inbound interface created in the Reserve Single Flight action

9. Assign the interfaces to the connection by selecting both interfaces. To do so, select the corresponding check boxes. Choose ✓ to confirm.

10. Save the business scenario.

Action Editor

Use

You use the action editor to display and change actions. Furthermore, you can assign one or more outbound or inbound interfaces to an action.

Actions are defined independently of Business Scenarios and can therefore be reused in different Business Scenarios.

Prerequisites

You have created a new action or have opened an existing one.

For more information about creating objects in the Integration Builder, see the Creating a New Object [Seite 40] section.

For more information about opening objects in the Integration Builder, see the Object Editors [Seite 35] section.

Features

The action editor comprises a header area and a work area.

Header

The header contains the following information about an action:

- Name
  
  The name of an action is unique within a namespace and is not language-specific. Blanks and special characters are not permitted in the name of a design object. The name is defined when the action is created and cannot be changed in the action editor.
• Namespace
The namespace is defined when the action is created and cannot be changed in the action editor.

• Description
In this field you can enter a short description of the action in the language you logged on in. This description is then displayed in the action when it is used in a Business Scenario. If you do not enter a description, the name of the action is displayed instead.

• Status
Displays the status of the object. For more information, see the Object Editor [Seite 35] section of this documentation.

Work Area
The work area is divided into three sections:

• Use
• Inbound Interfaces
• Outbound Interfaces

Use
In this area, use the radio buttons to select the following:

• Whether the actions are to be used in products (defined in the System Landscape Directory)
• Whether the action is to be used in templates (when no corresponding product is defined in the System Landscape Directory)

These specifications determine the application components of a Business Scenario that the action can be used in.

For more information, see the Inserting an Action [Seite 97] section of this documentation.

Note that the use cannot be changed once you have saved.

Outbound/Inbound Interfaces
In this area you can assign interfaces to an action or remove any assignments already made.

The assigned interfaces are displayed in a table in the Inbound Interfaces and Outbound Interfaces areas. The name and namespace of each interface are displayed. For more information, see the Assigning Interfaces [Seite 89] and Deassigning Interfaces [Seite 90] sections of this documentation.

Table Functions

<table>
<thead>
<tr>
<th>Icon</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>![ ]</td>
<td>Only in change mode (in the toolbar above the table or in the context menu when a line is selected): Inserts an empty line below the selected line</td>
</tr>
<tr>
<td>![ ]</td>
<td>Only in change mode (in the toolbar above the table or in the context menu when a line is selected): Deletes the selected line</td>
</tr>
</tbody>
</table>
Only in change mode: In the toolbar above the table or in the context menu when a line is selected:

Moves the selected line up or down

In the context menu; cursor positioned on a line with an interface:

Opens the interface editor for the line selected

Only in change mode and only when an interface has been entered that has not yet been defined in the Integration Repository (in the context menu; cursor on the line with an interface):

Creates a new message interface for this entry

Only in change mode (in the context menu; cursor positioned on a line with an interface):

Calls the search help for message interfaces and imported objects (RFCs, IDocs); selection restricted to the software component version of the action.

Additional Display Functions

When you position the cursor on a table line, the following information about the interface is displayed in a quick info box:

- Message type
- Direction (inbound or outbound)
- Mode (synchronous or asynchronous)
- Namespace
- Software component version

Assigning Interfaces

Use

You can assign one or more outbound or inbound interfaces to an action. These can be message interfaces, RFCs, or IDocs.

For more information about the different interface object types, see the Introduction to Interface Development [Page 119] section of this documentation.

Note the following restrictions when assigning interfaces:

- The software component version of the interface must be the same software component version as the action. The name of the interface and the action do not have to be the same.
- When assigning an inbound interface, it must be an inbound message interface, an RFC, or an IDoc.
- When assigning an outbound interface, it must be an outbound message interface, an RFC, or an IDoc.

Prerequisites

You have called the action editor [Page 87] and are in change mode.

Procedure
1. Depending on whether you want to assign an inbound or an outbound interface to the action, choose either Inbound Interfaces or Outbound Interfaces.

2. Choose Insert Empty Line Below Line Selected ( ).

3. You can call the search help in the inserted line. To do so, choose the search help icon ( ) at the end of the table line. In the dialog Select Interface, select the interface you want to use.

   Only those interfaces that match the criteria above are displayed in the dialog.

4. Select the interface you want to use and choose OK ( ) to confirm.

   You also have the option of assigning the action a message interface that has not yet been defined in the Integration Builder. To do so, carry out steps 1 and 2 above. In step 3, enter the name of the message interface manually in the corresponding field. You can also specify an appropriate namespace, if necessary.

   This option is supported by the top-down approach: You have the option of modeling the interface initially as just part of an action, without having to define it as a development object in the Integration Builder first.

   This can be done in the Integration Builder at a later stage. In the action editor, position the cursor on the interface in question and choose New ( ) from the context menu. A message interface is then created with the name and the namespace being taken from the action editor. You can now define the properties of the interfaces.

   This is only possible for message interfaces and not RFCs or IDocs.

   Note that you can change the direction and mode of an interface during its design, but any changes you make will not be taken into account in the action editor or Business Scenario editor. These can therefore lead to inconsistencies.

Result

The interface is assigned to the action and the name of the interface is displayed in the Inbound Interface or Outbound Interface area of the action editor.

Deassigning Interfaces

Use

You can remove the assignment of an interface to a particular action.

Procedure

To deassign an interface, select the table line for the interface concerned and choose Delete Selected Lines ( ).
Ensure that the interface is not part of a connection in a business scenario before removing the assignment. Otherwise, this will lead to inconsistencies in the Business Scenario.

For more information about the corresponding message, see the Handling Messages [Seite 105] section of this documentation.

**Result**

The interface ceases to be assigned to the action and the interface is removed from either the *Inbound Interface* or *Outbound Interface* area of the action editor.

---

**Creating Documentation for an Action**

**Use**

It is possible to write documentation for an action. This documentation contains the following keyblocks:

- **Use**
  Contains information about how the action is used.

- **Notes**
  Contains further specifications.

- **Referenced Interfaces**
  Contains information about the interfaces assigned to the action.

**Prerequisites**

You have called the *action editor* [Seite 87].

**Procedure**

The procedure for writing documentation for actions is described in the Documenting Objects [Seite 44] section.

---

**Business Scenario Editor**

**Use**

You use the business scenario editor to display and change business scenarios. You can visualize the process flow of a business scenario using an interactive graphical work area and assign additional objects that define the exchange of messages.

**Prerequisites**

You have created a new Business Scenario or have opened an existing one.

For more information about creating objects in the Integration Builder, see the Creating a New Object [Seite 40] section.

For more information about opening objects in the Integration Builder, see Object Editors [Seite 35].
**Features**

The business scenario editor comprises a header area and a graphical work area.

**Header**

The header contains the following information about a Business Scenario:

- **Name**
  The name of a business scenario is unique within a namespace and is not language-specific. Blanks and special characters are not permitted in the name of a design object. The name of the business scenario is defined when you create it and cannot be changed in the business scenario editor.

- **Namespace**
  The namespace of the business scenario is defined when you create it and cannot be changed in the business scenario editor.

- **Description**
  In this field you can enter a short description of the business scenario in the language you logged on in.

- **Status**
  Displays the status of the object. For more information, see [Object Editor](#).  

**Graphical Work Area**

Using the interactive graphical interface you can visualize the process flow of the Business Scenario and assign all the objects required. The structure of the Business Scenario editor is shown schematically in the following graphic. It also includes the various context-sensitive areas and the most important functions that can be called from the context menu.

**Business Scenario Editor Graphical Work Area and Context-Sensitive Areas**
The business scenario objects are represented in the business scenario editor as follows:

- Application components are portrayed as colored vertical columns. The colors of the columns are defined according to color conventions.
- The application component column has a header where the role, the name of the product with its release (product version), and the extended name are displayed.
- Actions are portrayed as white rectangles within an application component. The vertical progression of actions from top to bottom corresponds to the logical process flow.
- Sequences are shown as vertical lines with an arrowhead beside the subsequent action. The sequence determines the order in which the actions are executed.
- Synchronous connections are shown as horizontal lines with an arrowhead at each end. The two actions involved are on the same level in the graphic. This shows that the actions that are connected by such a connection are synchronized chronologically.
- Asynchronous connections are shown as bent lines with an arrowhead beside the subsequent action. The two actions involved here are not on the same level in the graphic. The subsequent action must be beneath the source action. This shows that the actions that are connected by such a connection are independent of each other and are not synchronized chronologically.

**Functions for Assigning Objects and Defining a Connection**

To assign Business Scenario objects, call the functions specified below from the context menu. To call a function, position the cursor on a particular area within the Business Scenario. The individual context-sensitive areas are represented in the Business Scenario (see above). To call the editor to display or change an object, double click the object in question in the business scenario.
Functions for Application Components

- Context: Graphical area is empty
  - Insert Application Component [Seite 94]
- Context: Application component (colored column):
  - Display/change application component [Seite 96]
  - Remove application component [Seite 96]

Functions in Actions

- Context: Application component (colored column)
  - Insert action [Seite 97]
- Context: A selected action
  - Display/change action [Seite 97]
  - Remove action [Seite 98]
- Context: Multiple selected actions:
  - Remove actions [Seite 98]

Functions for Connections

- Context: Two selected actions
  - Create a connection between two selected actions [Seite 99]
  - Display/change a connection between two selected actions [Seite 100]
  - Delete a connection between two selected actions [Seite 102]

Functions of the Graphical Editor

- See functions of the graphical editor [Seite 102].

Additional Functions

- Creating Documentation for a Business Scenario [Seite 104]
- Printing and Exporting the Graphic [Seite 105]

Inserting Application Components

Use

Logically speaking, an application component represents a participant of a Business Scenario. An application component is described by the following properties:

- By the role that it has within the Business Scenario (for example, supplier, plant)
- By the product version that the application component implements (for example, SAP R/3 4.6C)

You can assign a Business Scenario both an application component and a role. You must specify the implementing product version for the application component, including its enhanced name if applicable. Furthermore, you define the name of the WSCI interface for the application component.

Prerequisites

You have opened a Business Scenario.
**Procedure**

1. Position the cursor on an empty space in the Business Scenario. Call the function *Insert Application Component* from the context menu.

2. The system displays the *Insert Application Component* dialog.
   
   You have the option of defining the application component type. You can choose either of the following:
   
   - **Type** *Product* (defined in the System Landscape Directory)
   - **Type** *Template* (when no corresponding product is defined in the System Landscape Directory)

   For more information, see [Modeling Application Components][1].

3. Choose an option by selecting one of the radio buttons in the *Application Component Type* frame.
   
   - If you decide on *Product*, start the search help for the *Product Version* field and choose a product version from the list. To do so, choose the search help icon (🔍).
     
     The product version you select is then copied to the *Product Version* field. Furthermore, all software component versions are displayed in the area below that are assigned to this product version.

     ![Search Help Icon](image)

     You can only select a product if at least one software component version has already been imported for it. If this is not the case, the system displays a corresponding error message. See the [Handling Messages][2] section.

   - If you choose *Template*, no entry is displayed in the *Product Version* field. The software component version that is assigned the Business Scenario is displayed in the area below.

     For more information, see the [Modeling Application Components][1] section of this documentation.

4. Specify the role of the application component in the *Further Attributes* frame (optional).

5. Specify an enhanced name (optional).
   
   The enhanced name can be used to do the following:
   
   - Specify the function area of a product in more detail.

     ![Enhanced Name Icon](image)

     For example, you could specify the enhanced name *SAP FIN* for an application component with the product version *SAP R/3 4.6C*. This would indicate that this Business Scenario is relevant for the sub area *Financials*.

   - Specify the product name if the application component type is a template.

6. Specify a name for the WSCI interface (optional).

7. Confirm your specifications by choosing ✓.

**Result**

The system inserts an application component as a colored column in the graphical work area of the Business Scenario editor. The column header displays the role, the product version, and the enhanced name.

[1]: sap的帮助
[2]: sap的帮助
Displaying/Changing Application Components

Use
You have the option of displaying or changing information about an application component.

Prerequisites
You have opened a Business Scenario. This contains at least one application component.

Procedure
1. Position the cursor on the application component.
2. Choose Display/Change Application Component from the context menu. Alternatively, double-click the application component.
   The following is displayed in the Application Component Type frame:
   - Whether it is a Product (defined in the System Landscape Directory)
   - Whether it is a Template (not defined in the System Landscape Directory)
Furthermore, the following information is also displayed:
- Product name and enhanced name
- Imported software component version of the selected product version
- Role of the application component in the Business Scenario
- Name of the WSCI interface
   #
   If you are in change mode, you can change the enhanced name, the role, and the name of the WSCI interface.

Removing Application Components

Use
You have the option of deleting an application component from a Business Scenario.

Prerequisites
The application component does not contain any actions.

Procedure
Position the cursor on the application component and choose Remove Application Component from the context menu.

Result
The application component is deleted from the Business Scenario. The column belonging to the application component is no longer visible in the Business Scenario graphical work area.
Inserting Actions

Use
You can assign an action to a Business Scenario.

Prerequisites
Depending on the type of application component, the following restrictions apply:

- You can only insert actions for products (defined in the System Landscape Directory) if the actions are:
  - located in an application component version of the product version
  - designed for use in products

- You can only insert actions for templates if the action is:
  - located in the same software component version as the Business Scenario itself
  - designed for use in templates

Whether an action can be used in products or templates is defined in the action properties (see: Action Editor [Seite 87]).

Procedure

1. Position the cursor on the application component.
2. Choose Insert Action from the context menu. Select an action from the list displayed in the Select Action dialog.
   
   Only those actions permitted for the selected application component are displayed (see Prerequisites).
3. Select an action.
4. Choose ✓ to confirm.

Result
The action is inserted in the application component where you positioned the cursor. It is shown as a small white box. The description of the action is displayed in the box. If no description for the action exists, the name of the action is displayed instead.

To create a description for an action, navigate to the action editor [Seite 87].

Displaying/Changing Actions

Use
You have the option of displaying or changing an action. To do so, use the action editor [Seite 87].

Procedure
For more information about opening objects, see Object Editors [Seite 35].
You can also open an action used in a Business Scenario as follows: Position the cursor on the action in the graphical area of the Business Scenario editor [Seite 91] and do either of the following:

- Double click
- Choose Open Action from the context menu

### Removing Actions

**Use**

You have the option of removing one or more actions assigned to a Business Scenario.

**Procedure**

1. Select the action or actions in the graphical work area of the business scenario editor and choose Remove Action from the context menu.

   If you select more than one action, choose the function Delete Actions from the context menu. For more information about selecting actions, see Functions of the Graphical Editor [Seite 102].

**Result**

The selected assignment or assignments between the business scenario and an action or actions is removed and is no longer visible in the business scenario graphical work area.

   All connections associated with the action or actions are deleted.

### Defining Start and End Actions

**Use**

You have the option of defining an action as a start of end action or removing an assignment that you have already made.

**Procedure**

To define an action as a start action, select the action in question in the graphical work area of the Business Scenario editor and choose Define Action as Start Action from the context menu.

   From the context menu, you also have the option of undoing the assignment of the selected action as the start action.

To define an action as an end action, select the action in question in the graphical work area of the Business Scenario editor and choose Define Action as End Action from the context menu.
From the context menu, you also have the option of undoing the assignment of the selected action as the end action.

Creating a Connection Between Selected Actions

Use
You have the option of creating a connection between two actions.

There are three different types of connection:

- **Sequence**: Defines a connection between two subsequent actions in the same application component.
- **Synchronous connection**: Defines the synchronous exchange of messages between two actions from different application components.
- **Asynchronous connection**: Defines the asynchronous exchange of messages between two actions from different application components.

Prerequisites
You have opened a Business Scenario. The Business Scenario contains at least two actions.

There are three different situations where you require a connection between two actions. Each situation requires you to implement a particular connection type:

- To connect two subsequent actions within the same application component, you must define a sequence connection.
- To connect two actions from different application components that communicate with each other synchronously, you must define a synchronous connection. Both actions must be on the same level in the Business Scenario graphic.
- To connect two actions from different application components that communicate with each other asynchronously, you must define an asynchronous connection. The target action must be on a lower level than the source action in the Business Scenario graphic.

Procedure

1. Select both actions in the graphical work area of the Business Scenario editor [Seite 91].
2. Choose Create Connection Between Selected Actions from the context menu.
   The system displays the Display/Change Connection editor. Make any additional specifications about the connection here.

   From action (from the connection), to action (to the connection), and the type of connection are determined from the selected actions.

   For more information, see the Display/Change Connection [Seite 100] section in this documentation.
3. Choose ✅ to confirm.
Result
A connection is inserted between the two actions and is shown as a polygon in the graphical work area.

Display/Change a Connection Between Selected Actions

Use
You have the option of displaying or changing a connection between two actions.

To change a connection you can do the following:

• Select an outbound-inbound interface pair from the list of possible interfaces for the action
• Determine a mapping for this interface combination

Prerequisites
There are two ways of calling the Display/Change Connection editor:

• A connection already exists between the two actions.
  In this case:
  o In the graphical work area of the Business Scenario editor [Seite 91], double click a connection
  o Position the cursor on a connection line and choose Display/Change Connection Between Selected Actions from the context menu
  In both cases the actions concerned are selected.
  o First select both actions and choose Display/Change Connection Between Selected Actions. For more information about selecting actions, see Functions of the Graphical Editor [Seite 102].

  If more than one connection is represented by the same connection line, the window Select Connection is called and displays a list of connections. Select a connection from this list and confirm your selection by choosing ✓.

• The connection has still not been created. In this case, follow the steps described in the Create Connection Between Selected Actions [Seite 99] section.

The Display/Change Connection editor is in the same state as the editor for the Business Scenario that the connection belongs to. If the business scenario editor is in display mode, you can also display (but not change) the connection in this editor.

Features
The Display/Change Connection editor is structured as follows:

From and To Frame
This frame displays both actions to be connected. This includes a description of the action, the role of the application component, and the software component version (field Component)
Template is shown in the Component field for an action that is used in an application component of type Template.

In a synchronous connection you can swap from actions (from the connection) and to actions (to the connection) by choosing <>.

Connection Type Frame
This frame defines the connection as a sequence, synchronous, or asynchronous connection. Due to the relative position of both actions in the Business Scenario, when you create a connection the system automatically defines the connection type.

- If both actions belong to the same application component, Sequence is specified.
- If the actions belong to different application components but are on the same level in the Business Scenario graphical work area, Synchronous Communication is specified.
- If the actions belong to different application components but are not on the same level in the Business Scenario graphical work area, Asynchronous Communication is specified.

Interfaces Tab Page
This tab page contains all interfaces that are assigned to the actions belonging to the connection. When you position the cursor on a line for an interface, the following information about the interface is displayed in a quick info box:

- Message type
- Direction (inbound or outbound)
- Mode (synchronous or asynchronous)
- Namespace
- Software component version
- Unspecified Object (if no interface exists for this entry)

This may be the case if the interface was entered in the action editor manually but not in the interface editor.

To call the Display Message Interface editor for non-Unspecified Objects, double click. In change mode you can select exactly one outbound-inbound interface pair for exchanging messages. However, only those interfaces that are appropriate for the type of connection can be selected. Therefore, you cannot select any synchronous message interfaces if the connection is an asynchronous connection.

Mapping Tab Page
This tab page displays which mappings are assigned to the interface pair selected. Select a mapping from the list for this connection.

Double click a mapping to navigate to the mapping editor [Seite 143].

Tab Page Documentation
Here you can display and create documentation for the connection. Connection documentation is part of the business scenario documentation and belongs to the Connections key block.

A connection is not an independent object. It is only defined as part of a particular business scenario.
Therefore, the entire documentation for the Business Scenario [Seite 104] is displayed.

In change mode you can change the part of the Business Scenario documentation that belongs to the Connections key block. The system opens an HTML editor in the upper part of the tab page where you can enter your text. The text for the Connections key block is displayed in a preview window as soon as you have saved by choosing Save ( ).

⚠️ You can switch between display and change mode on the Documentation tab page for a connection, regardless of whether the business scenario editor for the business scenario that the connection belongs to is in display or change mode.

Integration with the Business Scenario Editor

Before saving a business scenario, ensure that all the editors of the individual connections have been closed. The system displays a confirmation prompt that gives the user the option of closing all connection editors automatically, or canceling the save action to close them individually. The procedure is the same if you attempt to close a Business Scenario with connection editors still open.

Deleting a Connection Between Two Actions

Use

You have the option of deleting a connection between two actions.

Procedure

1. Select both actions for the connection concerned in the graphical work area of the Business Scenario editor [Seite 91], or position the cursor on the connection line.

2. Choose Delete Connection Between Selected Actions from the context menu.

💡 If more that one connection is concealed behind a connection line (if a connection line represents more than one connection), the system first displays a list of the connections that you can delete. Select the connections you want to delete from the list.

Result

The selected connections are deleted from the Business Scenario.

Functions of the Graphical Editor

Use

The graphical editor is part of the Business Scenario editor [Seite 91].
**Features**

**Marking and Moving Objects**

This section contains information about the actions you can perform in the graphical editor. They refer to the following objects:

- Application Components
- Actions
- Connections

**All Objects**

<table>
<thead>
<tr>
<th>Activity</th>
<th>Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display/Change</td>
<td>Position the cursor and</td>
</tr>
<tr>
<td></td>
<td>• Double-click</td>
</tr>
<tr>
<td></td>
<td>• Choose Display/Change &lt;Object&gt; from the context menu</td>
</tr>
</tbody>
</table>

**Application Component**

<table>
<thead>
<tr>
<th>Activity</th>
<th>Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Move (horizontally)</td>
<td>Use Drag &amp; Drop</td>
</tr>
<tr>
<td></td>
<td>Drag the application component header horizontally to a new position.</td>
</tr>
<tr>
<td></td>
<td>All actions and connections remain unchanged when you move an application component.</td>
</tr>
</tbody>
</table>

**Action**

<table>
<thead>
<tr>
<th>Activity</th>
<th>Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select</td>
<td>To select an individual action, position the cursor on the action and</td>
</tr>
<tr>
<td></td>
<td>• Click once</td>
</tr>
<tr>
<td></td>
<td>• Call the context menu</td>
</tr>
<tr>
<td></td>
<td>Alternatively, set the focus on an action by using the cursor keys or the</td>
</tr>
<tr>
<td></td>
<td>tab key and then pressing the space bar. This is only possible in change</td>
</tr>
<tr>
<td></td>
<td>mode (for more information about setting the focus, see below).</td>
</tr>
<tr>
<td></td>
<td>To select multiple actions, proceed as follows:</td>
</tr>
<tr>
<td></td>
<td>• Hold down the control button and select the actions by clicking the</td>
</tr>
<tr>
<td></td>
<td>mouse once</td>
</tr>
<tr>
<td></td>
<td>• Click a connection line</td>
</tr>
<tr>
<td></td>
<td>o If the connection line represents exactly one connection,</td>
</tr>
<tr>
<td></td>
<td>both actions belonging to the connection are selected.</td>
</tr>
<tr>
<td></td>
<td>o If the connection line represents multiple connections, all</td>
</tr>
<tr>
<td></td>
<td>actions associated with all the connections are selected</td>
</tr>
<tr>
<td></td>
<td>• Surround the actions with the rubber band. To do so, encircle the</td>
</tr>
<tr>
<td></td>
<td>actions with the cursor by holding down the left mouse button.</td>
</tr>
<tr>
<td></td>
<td>Once you have selected an action, it is surrounded by a red border and</td>
</tr>
<tr>
<td></td>
<td>the background changes to a light red color.</td>
</tr>
</tbody>
</table>
### Move (vertically)

**Use Drag & Drop**

You have the option of moving multiple actions that you have already selected simultaneously.

Note the following restrictions:

- Actions cannot be moved between different application components
- Actions can only be moved vertically
  - if the new relative position of the actions is compatible with the connections defined between the actions. If this is not the case, the system displays a corresponding message.
  - if there is sufficient space in the Business Scenario graphical work area to accommodate the move.

### Connection

<table>
<thead>
<tr>
<th>Activity</th>
<th>Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Move</td>
<td>Connections can only be moved indirectly when you move the actions that belong to the connection.</td>
</tr>
</tbody>
</table>

### Additional Functions

- **Show/hide grid and set focus**
  
  You have the option of including a grid in the graphical work area. If you do so, all the possible positions for actions are indicated by corner points.

  With the grid activated, you can set the focus on a grid point or an action. In this case, the position is indicated by a red border.

  You move the focus by using the cursor keys or the tab key.

  - Once you have set the focus on a position in which an action has already been inserted, press the `Enter` key to call the action editor for this action or press the `Spacebar` to select the action (the background changes to a light red color).

    ![Tip](image)

    In this way, the tool is convenient to use with the keyboard.

  When the grid is hidden, you can only set the focus on a position with one action. In this case, the action is then also selected.

  To show/hide the grid, in the Business Scenario editor menu, choose `View → Grid → Show/Hide`.

- **Zoom**
  
  The size of the graphical work area can set between 100% and 50%. To change its size, from the Business Scenario editor menu, choose `View → Zoom`.

### Creating Documentation for a Business Scenario

**Use**

You have the option of writing documentation for a Business Scenario.

You can use the following keyblocks:
- Purpose
- Prerequisites
- Process Flow
  For a short description of the individual steps that are performed in the Business Scenario.
- Actions
  For information about the actions used in the Business Scenario.
- Connections
  For information about the connections used in the Business Scenario. You can also create the text for this keyblock from the Change Connection [Seite 100] editor.
- Results
- Notes

**Prerequisites**
You have opened a Business Scenario.

**Procedure**
The general procedure for writing documentation for a design object is described in the Documenting Objects [Seite 44] section.

---

### Printing and Exporting the Business Scenario Graphic

**Use**
You can do the following with the Business Scenario graphic you created in the Business Scenario editor [Seite 91]:
- Print
- Export

**Procedure**
To print the Business Scenario graphic, proceed as follows:
- In the Business Scenario editor menu, choose Business Scenario → Print
- In the Business Scenario editor toolbar, choose Print ( ).

To export the Business Scenario graphic, proceed as follows:
- In the Business Scenario editor menu, choose Business Scenario → Export as JPEG (icon ).

---

### Handling Messages
This section contains information about important error messages that can occur, as well as recommendations about how you can react to the error situation.
**Deleting an interface from an action (action editor)**

**Message:** Warning – This interface can still be assigned to a connection. Do you want to proceed?

If you delete an interface that is still used in connections in a Business Scenario, this may lead to inconsistencies because the Business Scenario concerned is not updated automatically and the corresponding connections will continue to reference the deleted interface.

Therefore, SAP recommends that you first ensure that the interface in question is no longer used in any Business Scenarios before deleting it.

**Deleting an action (action editor)**

**Message:** Warning – This action can still be assigned to a Business Scenario. Do you want to proceed?

All the Business Scenarios in which the action to be deleted is used are not automatically updated following deletion. This may lead to inconsistencies. The action continues to be displayed in the Business Scenario graphic, but it no longer exists as an object.

SAP recommends that you ensure that this action is no longer referenced in any Business Scenarios before deleting it.

**Inserting a product version in an application component (Business Scenario graphic)**

**Message:** Imported software component versions are required to use a product. However, none exist for the selected product.

To be able to insert an application component of type Product into a Business Scenario, at least one software component must exist for the product concerned in the System Landscape Directory. Furthermore, at least one of the corresponding software component versions must have been imported into the Integration Repository. If this is not the case, the corresponding product version cannot be inserted in an application component and the system displays the message above.

You have two alternatives:

- Instead of a product, use an application component of type Template.
  
  For more information, see the Modeling Application Components [Seite 110] section of this documentation.

- Import an appropriate software component version for the product concerned
  
  Note that importing software component versions has extensive consequences. Only use this option if you are sure that you actually want to import software component versions.

For more information, see:

- Importing software component versions [Seite 51]

**Deleting the selected actions (Business Scenario graphic)**

**Message:** All selected actions and assigned connections will be deleted. Do you want to proceed?

If you choose to continue, all selected actions and all assigned connections are deleted from the Business Scenario graphic. The connections are deleted completely. Interface and mapping settings for the connections concerned will be deleted and cannot be recovered.

**Double-click on action in the Business Scenario graphic (Business Scenario graphic)**

**Message:** Object not found (......)

Possible reasons for the message occurring:
• The software component that belongs to the action is not installed in your system. Therefore, the action does not exist in your system as an independent object of the Integration Builder. This action is displayed in the Business Scenario graphic but you cannot navigate to the actual object. It is also not possible to change or display the action.

• The software component that belongs to the action is installed on your system but not in the software component version that was originally used to create the action. Again, the action is displayed in the graphic but you cannot display or change it.

This means that you can display the entire Business Scenario, even if some of the products involved are not installed on your system. However, you can only navigate to the objects concerned (actions, interfaces) once you have installed the software component versions required.

Guidelines for Modeling Business Scenarios

This section gives you an overview of what you should note when modeling Business Scenarios and supports you when designing them. Furthermore, it provides you with background knowledge about the modeling concept and design decisions in the Integration Builder (design tool).

Targets of Modeling

A Business Scenario completely models the exchange of messages for a collaborative process and provides an overview of the process flow. The advantages of defining Business Scenarios in the Integration Builder (design tool) are as follows:

• The Business Scenario provides you with an overview of the process and the process flow.

• The Business Scenario groups together all objects that are involved in this process: interfaces and mappings from the Integration Repository and product versions from the System Landscape Directory. You can access all of these objects from the Business Scenario.

• The Business Scenario contains all design time information about the process that are required for its configuration. As a result, configuration is simplified.

Use the above as your starting point when defining and modeling Business Scenarios.

The sections below contain detailed information about what you must note when modeling the individual components of the Business Scenarios:

• General Modeling Concept [Seite 107]
• Modeling Business Scenarios [Seite 108]
• Modeling Application Components [Seite 110]
• Modeling Actions [Seite 112]
• Modeling Connections [Seite 114]
• Modeling Process Variants of a Business Scenario [Seite 117]

General Modeling Concept

This section introduces the general modeling concept on which the design of Business Scenarios in the Integration Builder is based.
Multiple logical participants are involved in a Business Scenario (=collaborative process). These participants are represented by application components.

Local (private) processes run within the application components.

- The parts of the local process that are visible in the entire process must be shown in the Business Scenario (in other words, seen or used by other participants).
- All other parts of the local process are private parts of the application component. They are only represented in the Business Scenario if this is necessary to understand them.
- Actions (= functions running in the application component) and sequences are used to represent the local processes.

The points of interaction and synchronization between the individual application components are also shown in the Business Scenario. This is also known as a global model.

The Business Scenario must contain all communication steps that could potentially occur in the process.

- Communication steps are represented by connections between actions in different application components.
- Synchronous communication is represented by a horizontal double-headed arrow. Both actions are on the same level in the graphic.
- Asynchronous communication is represented by a downward-pointing arrow. The source action is always on a higher level than the subsequent action.

The vertical arrangement of actions and connections (from top to bottom) implies a logical sequence between the steps. This is similar to UML sequence diagrams.

The meta model on which Business Scenarios are based corresponds to the description with the Web Service Choreography Interface (WSCI). This enables you to easily create WSCI documents that describe the process from a Business Scenario.

You must follow the modeling principles described here when modeling your Business Scenario. In the sections that follow you will find detailed information about modeling the individual objects within a Business Scenario.

### Modeling Business Scenarios

#### Name

The name of a Business Scenario is unique (within its namespace). This name is not language-specific. Special characters and blanks are not permitted.

Note the following conventions for names:

- The Business Scenario must have a meaningful name. The name should be written in English so that it is globally understood.

  Separate individual words by switching between uppercase and lowercase characters (see the notes on naming conventions for design objects under [Creating New Objects](#)).

  Example: `BookFlightTrip`
Description
The description gives more details about the Business Scenario. It is language-specific and is displayed in the language you are logged on in.

Follow the usual conventions for short descriptions here.

Example: Book Flight Trip

Assignments to a Software Component Version
You must assign a software component version to a Business Scenario when you create it. This assignment defines the software component versions that are delivered to the customer with the Business Scenario (as a design object of the Integration Builder).

Note the following aspects when selecting a software component version:

- Choose the software component version according to your shipment strategy
- Often there is one leading product (for example, SAP APO) within a Business Scenario that defines it and takes organizational responsibility. In this case, the Business Scenario should be assigned a software component version that is appropriate for this product.
- However, it is also possible to assign the Business Scenario to a software component version that corresponds to another product or to none of the products used, if required.

Level of Detail of Business Scenarios
You have the option of varying the scope and level of detail of Business Scenarios. This is foremost a decision to be made during modeling.

Note the following aspects:

- The Business Scenario must represent a completed business process or clearly defined part of a process.
- The Business Scenario is transferred as a whole to configuration. Different sub processes must therefore only be grouped together for a Business Scenario if all parts are always to be configured together.
- The Business Scenario must not become so large that it ceases to be comprehensible for the user.
- Completed sub processes at the start and end of a Business Scenario can possibly lead on to separate Business Scenarios. This is particularly interesting if a sub process of this type occurs in multiple Business Scenarios (enables it to be reused and reduces complexity).
- Take into account any existing modeling results in the Solution Manager or Business Process Repository.

Variants of a Business Scenario
At present, the Integration Builder does not support any explicit variant-management of Business Scenarios.

Therefore, note the following aspects for variants of a Business Scenario:

- You have the following options for representing the variants of a Business Scenario:
- Represent multiple (or all) variants in a Business Scenario
- Define separate Business Scenarios for different variants
• SAP strongly recommends that you represent all variants in a Business Scenario if possible. You must only define separate Business Scenarios for variants in exceptional cases.
• You might, for example, explicitly model the main variants. See the documentation for variations and secondary variants.

### Modeling Application Components

Logically speaking, an application component represents a participant of a Business Scenario. This contains the business tasks and concerns that this participant represents within the business scenario.

**Modeling at Type Level**

An application component can be installed on multiple business systems at runtime. However, modeling in the Business Scenario takes place at type level.

Note the following when modeling application components:

• If, at runtime, there are multiple participants that
  o have the same business role
  o result in the same range of tasks
  o use the same product
  
  then these are presented by one logical participant and therefore just one application component.

• If different participants fulfill different tasks or they use different products or product versions for the same tasks, then they are represented by different application components.

• If different participants fulfill the same tasks and cannot be specified to their products, then an application component (of type Template) can be used in place of this participant.

  In the Book a Flight Trip Business Scenario example, the travel agency works together with various airlines. Since all airlines perform the same tasks within the Business Scenario and their products cannot be specified in more detail, they are modeled as one single application component with the role Airline.

**Role**

The role describes the task area that the application component is responsible for on a semantic level.

Note the following conventions:

• The role must have a meaningful business name and should not be language-specific

• If multiple business partners are involved in the Business Scenario, the roles must describe them

  Example: Customer and Vendor.

• If a business partner is subdivided into multiple application components, then use composite roles
Example: Vendor – Customer Management and Vendor – Production.

- If the entire Business Scenario only involves one business partner, it does not necessarily have to be included in the roles

Example: Customer Management and Production.

**Products and Templates**

The product specifies the software product that is implemented to perform the tasks of the application component.

Choose type *Product* for the application component if

- at design time you know which product version will fulfill the tasks of the application component and
- if this product version is defined in the System Landscape Directory

Examples of product versions are SAP APO 3.0 and SAP R/3 4.6C.

Choose type *Template* for the application component if

- at design time you do not know which product version will fulfill the tasks of the application component or
- if it is a non-SAP product that is not defined in the System Landscape Directory

*(SAP internal)*

You also use an application component of type *Product* if the product version is defined in the System Landscape Directory but no software component versions have been imported for it yet.

This is particularly relevant during the transitional period when not all application content has been entered in the Integration Builder.

If no software component version has been imported for a product version, you cannot create any actions or ship any actions for it.

You create a Business Scenario that, among others, includes the product version SAP R/3 4.6C. You communicate with this product version using IDocs. No software component versions have been imported for this product version yet since no Integration Repository contents have been shipped. As a result, you cannot define any actions that you can use in this application component.

Therefore, model the application component as a template and include the enhanced name *SAP R/3 4.6C*.

**Enhanced Name**

Use the enhanced name to describe the product in more detail. You can also use it to do the following:

- Specify a smaller function range for a product
For example, you could specify the enhanced name *SAP FIN* for an application component with the product version *SAP R/3 4.6C*. This would indicate that this Business Scenario is relevant for the sub area *Financials*.

- Specify the product name, if it is a template.

**Release Combinations**

The Business Scenario shows each application component with a fixed product (including its release). This means that a Business Scenario is defined for the release combination of the application components specified.

If one or more application components from multiple product versions of the same product line can be implemented, proceed as follows:

- Show the Business Scenario with the product versions (in other words, the releases) that you consider to be the standard combination for this Business Scenario.
- In the documentation, specify whether the Business Scenario also functions for other product versions (or product releases). If necessary, create exceptions in the process flow and in the documentation.
- You also have the option of creating a different release combination in a separate Business Scenario. However, this must only be done in exceptional cases to keep the number of separately defined variants of a Business Scenario in check.

**Modeling Actions**

Actions represent one of the functions performed by an application component within a Business Scenario.

**Name**

The name of an action is unique (within its namespace). The name is not language-specific. Special characters and blanks are not permitted.

Note the following conventions for names:

- The action must have a meaningful business name. The name should be written in English so that it is globally understood.
  
  Separate individual words by switching between upper and lower case characters (see the notes on naming conventions for design objects under *Creating New Objects* [Seite 40]).

  Example: *SearchForFlightConnection*

**Description**

The description gives a brief explanation of the business function of the action. It is language-specific and is displayed in the language you are logged on in. Note that description text is displayed in the Business Scenario graphic when the action is used in a Business Scenario.

Follow the usual conventions for short descriptions here. Keep the description as brief as possible so as not to complicate the Business Scenario graphic unnecessarily.
Example: *Search for Flight Connection.*

**Assignments to a Software Component Version**

You must assign a software component version to an action when you create it. This assignment has the following consequences:

- It determines the software component versions that the action (as a design object of Integration Builder) is shipped to the customer with.
- It determines the application components that the action can be used in.

For more information, see the following section.

**Use in Products or Templates**

You can use an action either in application components of type *Product* or of type *Template*. You **cannot** use an action in both types of application component. You must decide which you want to use when you create the action in the action editor.

Note the following:

1. The action is an implemented action, in other words, the function implemented by the action is part of a known software component version.
   - Select to use the action in an application component of type *Product* (defined in the System Landscape Directory).
   - Select the software component version in which the action is implemented. In other words, the implemented action always belongs to the implementing software component version.
   - You can use this action in any Business Scenarios in all product versions for which the software component version of the action is part of the product version.
   - The action is shipped with its software component version. In other words, the action is generally shipped independently of the business scenario in which it is used.

   **Reason:**
   
   You must be able to reuse an implemented action in the same way that the corresponding function is reused. Therefore, the action is defined in the software component version in which its function is implemented, and not in the software component version of the Business Scenario.

   You define an action in the software component version *SAP APO 3.0A*. Since this software component version is part of the product version *SAP APO 3.0A*, you can use this action in all application components with this product version.

   You define an action in the software component version *SAP APO 4.6C*. This software component version is part of multiple product versions (for example, *SAP R/3 4.6C* and *SAP APO 3.0A*). Therefore, you can use this action in all application components with such product versions.

2. The action is a modeled action. In other words, the action represents a function in products not specified in more detail.
   - Select to use the action in application components of type *Product* (not defined in the System Landscape Directory).
   - Select the software component version of the Business Scenario in which you want to use the action.
You can only use this action in Business Scenarios that have the same software component version as the action. Also, you can only use the action in application components of type Template.

The action is shipped with its software component version. In other words, the action is always shipped together with the Business Scenario in which it is used.

Reason:
A modeled action (for application components of type Template) does not belong to a particular software component version as far as its functions are concerned. Therefore, it is assigned to the software component version of the Business Scenario in which it is used. It is not recommended that you reuse the action on a large scale. For this reason, this action can likewise only be reused in Business Scenarios with the same software component version.

You define an action for application components of type Template in the software component version SAP APO 3.0A. You can use this action in all Business Scenarios that are assigned to the software component version SAP APO 3.0A, too.

Business Meaning of Actions
Note the following aspects when defining and selecting actions:

- The granularity of actions should be based on the granularity of processes or process steps.
- The local, business function that fulfills the task within the application component must be clear for each action (by using an appropriate name and documentation). Actions are the anchor points for the local functions of the application component.
- Besides the connection to the local functions, actions must also provide a logical connection to the customizing settings that are required to use this part of a Business Scenario.

For more information about the level of detail of actions, see the section below.

Using Actions in Application Components
Note the following aspects about scope and level of detail when using actions in Business Scenarios:

- All processes or process steps that require two different application components to communicate with each other must be represented as actions in the Business Scenario. If a local process only requires one (or a few) communication step(s), then you can model the whole process as one action. If a local process requires multiple communication steps, then the process must be modeled with numerous process steps.
- You also have the option of modeling transitional steps within a local process that do not require any communication steps to other application components. However, you must only do so if it is vital for the clarity of the Business Scenario.

Modeling Connections

Sequences
A sequence is a connection between two subsequent actions within an application component. A sequence is represented by a vertical downward-pointing arrow.
You use a sequence if the execution of the first action is the prerequisite for the second action being executed.

You can also use a sequence to define the sequence of inbound processing during asynchronous communication.

### Designing Synchronous and Asynchronous Communication Steps

When designing your business scenario, you must pay particular attention to the exchange of information between the application components involved.

Note the following:

- Identify all instances in your business scenario where information must be exchanged between the components by using messages. This also includes communication steps.
- Define whether messages are to be exchanged synchronously or asynchronously.
  - Synchronous communication means that the sender waits for a response from the receiver before proceeding with processing.
  - Asynchronous communication means that the sender does not expect an immediate response from the receiver and therefore proceeds with processing immediately.

In extensive business scenarios, you should, where possible, always work with asynchronous communication. As a result, your business scenarios will be more robust and reliable.

- Depending on whether you select synchronous or asynchronous, identify the interfaces you require for communication. If they do not already exist, you must define them in the Integration Builder (design tool). Ensure that the properties of the interfaces (outbound/inbound, synchronous/asynchronous) correspond to your design.

### Representing Communication Steps in the Business Scenario

You must reproduce the decisions you make at design time in the business scenario graphic of the Integration Builder.

Note the following aspects:

- Each message exchange must be represented in the business scenario by a connection between the corresponding actions.

Note that this also referred to as a type level. For example, if a communication step is used more than once, but involves the same action and interfaces each time, then this is represented by just one connection.

For example, in the Book a Flight Trip [Seite 81] business scenario example, the action Search for Flight Connection is executed repeatedly until an appropriate flight connection has been found. This action queries flight availability at the various airlines. All these communication steps are however of the same type and can therefore be represented by just one connection.

- Define synchronous or asynchronous connections as per your design.
  - A synchronous connection is represented by a horizontal double-headed arrow. Both actions must be on the same level in the graphic.
  - An asynchronous connection is represented by a downward-pointing arrow. The target action must be on a lower level than the source action in the business scenario graphic.
The business scenario design environment uses the relative position of the two actions in the graphic to automatically define the communication type. Therefore, ensure that the actions are in the correct order in the graphic, as per the guidelines above.

See the business scenario example Book a Flight Trip [Seite 81]. The connection between the actions Search for Flight Trip and Determine Flight Availability is synchronous. The connection between Book Flight Trip and Book Single Flight is asynchronous.

- Complete the specifications for the connection by selecting the outbound and inbound interfaces to be used for exchanging messages. If required, select a mapping to be executed for this connection.
- If there are different alternatives as to which interfaces and mappings you can use for exchanging messages, then model each of the alternatives as a separate connection.
- If more than one connection exists between two actions, then these must be alternatives. At configuration time, you can only select one connection for each sender/receiver relation.
- Note the following rules regarding the sequence of communication steps that are to be processed by an action:
  - Inbound communication steps always come before outbound communication steps chronologically.
  - If more than one inbound communication step exists, then no order is defined between them.
  - If more than one outbound communication step exists, then no order is defined between them.
  - “No order defined” means that the order is unknown and is therefore not relevant.

If an application case occurs that conflicts with the rules above, then you must model your connection differently. A solution might be to divide the action concerned into two (or more) actions.

Start and End Actions

The purpose of start and end actions is on the one hand, to improve clarity for the user, and on the other, to specify possible points for a mapping to take place.

- A start action can be any action from which a business scenario can begin. It is possible to have more than one start point.
- An end action can be any action, which brings a business scenario to a logical business conclusion. It is possible to have more than one end action.
- If an action is classified as a start action, you must position it at the start of your application component. Likewise, an end action must be positioned at the end of your application component.
Modeling Process Variants of a Business Scenario

The Business Scenario design environment in the Integration Builder does not permit the qualified representation of branching (AND, OR, XOR) or any other complex process modeling constructs.

- When representing different process variants you must find an appropriate compromise between the clarity of the Business Scenario on the one hand, and the number of separately defined Business Scenarios on the other.

- Note the following conventions:
  - Any optional actions (and communication steps) must be included in the Business Scenario and documented explicitly as optional. It is then clear in the Business Scenario that these optional steps do not necessarily have to be executed.
  - If there are various branches in the process logic (OR, XOR) and the branches are located in different application components, then both branches must be included in the Business Scenario and documented accordingly.
  - If there are various branches in the process logic (OR, XOR) and the branches are located in the same application components, then you must decide if it is vital that the branches are modeled at all.

- The Business Scenario design environment in the Integration Builder does not permit any explicit variant management. Therefore, it is recommend that you include the different variants in a Business Scenario wherever possible. However, if the Business Scenario becomes overly complicated as a result, or it ceases to be comprehensible, SAP recommends that you model the variant as a separate Business Scenario.

Interface Design

Purpose

Generally speaking, interfaces are where functions in a system can be executed. In the context of SAP Exchange Infrastructure, the only interfaces that are of interest are those that enable data to be exchanged between multiple systems. Starting with a cross-system business process, you can then derive the corresponding interfaces required. SAP Exchange Infrastructure supports this process by using a business scenario to describe the collaborative process. The business scenario summarizes the interfaces required for this collaborative process. There are two different types of interfaces:

- **Outbound Interfaces**: Called in a sender system to send a message and in the case of synchronous messaging, to receive a response.

- **Inbound Interfaces**: Process inbound messages in a receiver system and implement a service that returns a response in the case of synchronous messaging.

In the remainder of this documentation, a message from a sender to a receiver is referred to as a request or request message, and the response to this request from a receiver is referred to as a response or response message.

You can use SAP interfaces that already exist in systems, non-SAP interfaces that are connected to SAP Exchange Infrastructure using adapters [Seite 340], or define new interfaces called message interfaces in the Integration Repository. Both worlds can also be interconnected in a collaborative process.
Integration

Interfaces are an essential component of SAP Exchange Infrastructure:

- You define the interfaces to be used in a business scenario
- You define the XML transformations for messages that are to be exchanged between two interfaces in a mapping
- You assign an interface in a sender system to one or more interfaces in a receiver system in logical routing
- You generate proxies to implement your scenario based on message interfaces

You save interface descriptions (message interfaces, BAPIs, RFCs and IDocs) in the Integration Repository so that they can be referenced throughout SAP Exchange Infrastructure. However, adapters for external systems do not normally use interfaces, instead they access files or database tables in order to function, for example. For this reason, it is not necessary to import these interfaces into the Integration Repository. Instead, you define the sender and receiver data for routing and mapping in the Integration Engine when you configure the corresponding adapters. Also see: Communication Partners [Seite 121].

Features

There are two approaches when developing interfaces:

- You can develop new, platform-independent message interfaces using the Integration Builder. Message interfaces are based on the WSDL standard Web Services Description Language), an XML schema for describing network services. Using this description, you can generate platform-specific proxies in Java or ABAP that you can then use to implement the actual data exchange.
- You can connect interfaces from SAP and non-SAP systems to SAP Exchange Infrastructure by using adapters. The Integration Builder can import interface descriptions in XML format for BAPI, RFC and IDoc interfaces from SAP systems Release 4.0 or higher.

By assigning the appropriate inbound and outbound interfaces, these two approaches can be integrated so that any combination of systems for exchanging messages using SAP Exchange Infrastructure is possible.

The Integration Builder supports the following for interface development:

- Platform-independent definition of message interfaces. Data types for message interfaces are defined in XML Schema Definition Language (XSDL). An XSD editor is available for this purpose.
- Import of BAPIs, RFCs and IDocs from SAP systems Release 4.0 or higher.
- Generation of Java proxies directly from the Integration Builder.

For more information about the general functions of the Integration Builder that are also useful for interfaces, see Integration Builder [Seite 27].

To generate an ABAP proxy, call transaction SPROXY in the SAP system in which you want to create the proxy. This is possible for SAP systems that have SAP Exchange Infrastructure Release 2.0 or higher installed.
Introduction to Interface Development

Interfaces are at the heart of every cross-system process. They are referenced almost constantly during design and configuration. For this reason, it is vital that an interface description can be accessed from a central point. Essentially, there are two types of **Interface Descriptions** that you can save in the Integration Repository, namely platform-independent message interfaces and imported interfaces, in which an XML representation of a BAPI, RFC or IDoc is generated in the Integration Repository (also see: Interface-Based Message Processing [Seite 17]).

Only information about the structure of message interfaces and imported interfaces is stored in the Integration Repository (interface type, parameters used, and data types) and not how the interfaces are actually implemented. By keeping this information separate, you can access the interface descriptions when defining scenarios, mappings or in routing, without having to communicate with the systems in which the interfaces are implemented.

In the context of the Integration Builder and in this documentation, we only refer to interfaces (for example, message interfaces or imported interfaces) and not to interface descriptions.

Prerequisites

The interface types you use for the integration of your collaborative process depend on the process itself and your system landscape. Usually, you will have to use different interface technologies for the integration process. The following technical restrictions exist:

### Prerequisites for Proxy Generation from Message Interfaces

<table>
<thead>
<tr>
<th>Target Language</th>
<th>Version of Platform Required to Implement Proxies</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABAP</td>
<td>SAP systems that have SAP Exchange Infrastructure Release 2.0 or higher installed.</td>
</tr>
<tr>
<td>Java</td>
<td>JDK 1.3.1 (either standalone or J2EE) or higher</td>
</tr>
</tbody>
</table>

To connect RFCs, BAPIs and IDocs originating from systems Release 3.1I or higher to SAP Exchange Infrastructure, you require either the RFC adapter [Seite 349] or the IDoc adapter [Seite 340]. RFCs, BAPIs and IDocs from systems Release 4.0 or higher can be imported into the Integration Repository. To integrate non-SAP systems you require the Adapter Engine [Seite 362], which can forward messages as files to the Integration Engine, for example.

It is of course still possible to develop RFCs, BAPIs and IDocs after SAP Web AS 6.20 and to use relevant adapters to integrate them. However, SAP recommends that you use message interfaces for new developments.
Communication Parameters

Both message interfaces and imported interfaces can be designed to send a request or to respond to a request. You may or may not receive a direct response message to this request. For message interfaces, you define this using the characteristics synchronous or asynchronous and outbound and inbound.

Outbound and Inbound Interfaces

These characteristics determine the direction of an interface:

- An outbound interface sends a request message that is only used to make a communication partner aware of data sent, without waiting for a response message. In the case of the latter, we also refer to publishing interfaces.
- An inbound interface receives a request message that you reply to with a direct response message or whose data you can process in the system without a response.

You require an inbound and an outbound interface pair for two components to be able to communicate with each other. This means that an outbound message interface can be assigned to an inbound message interface as well as to an RFC interface that replaces the inbound counterpart semantically (and the other way around). Also see: Communication Partners [Seite 121].

Synchronous and Asynchronous Communication

A distinction is made between synchronous and asynchronous communication for both imported interfaces and message interfaces. You define the mode of communication when you define an interface:

- In synchronous communication, a response is expected from the receiver system on arrival of a query sent by the sender system. Once this message has been sent, no further messages can be sent until the answer to the query has arrived back at the sender system.
- However, in asynchronous communication the response is not immediate. A sending process can send multiple messages to a receiver in a bundle and then continue executing the process.

The pair of interfaces must have the same mode.

Message Exchange

The following graphic shows how the mode and direction of interfaces influence message exchange between interfaces:
Synchronous Communication

Interface A:
- outbound
- synchronous

Message 1: Request

Interface B:
- inbound
- synchronous

Message 2: Response

Asynchronous Communication

Interface C:
- outbound
- asynchronous

Message: Request

Interface D:
- inbound
- asynchronous

Communication Partners

In the example below, multiple receiver systems need to communicate with one sender system:
In this example, the receiver SAP system does not support proxy generation. Instead, an IDoc and an RFC are used on the inbound side. To be able to access the names of these interfaces later during logical routing and mapping, you must import an XML description into the Integration Repository.

However, you can generate an inbound proxy for the ABAP sender and the Java receiver. To do so, create two message interfaces in the Integration Repository.

Therefore, an outbound message interface can be connected to different receiver interfaces and the other way around. To model or document which interfaces belong together semantically, use a business scenario [Seite 77].

**Interfaces and Message Types**

As illustrated in the example above, it is possible to interconnect different interface types. To do so, both interfaces must have the same mode (synchronous or asynchronous), and the parts that determine the payload [Seite 15] of the message must be assigned. When a synchronous proxy communicates with a synchronous RFC module, the following parts must be assigned to each other in the Integration Repository:
Therefore, the following is the case for a message interface:

- The *output message type* determines how the message to be sent is structured.
- The *input message type* determines how the received message is structured.

Note the following for different interface types:

- No data types are created in the Integration Repository when RFCs or IDocs are imported into the Integration Repository. If an imported interface is the counterpart of a message interface, you must create the corresponding data types and message types for the message interface yourself.
- The respective adapter converts the interface call to XML format that is supported by the interface. When an RFC is called, the adapter generates, for example, RFC XML that is supported by the Integration Engine. If the receiver is not the same interface type, for example a proxy (that expects or generates proxy XML), you must map the corresponding fields to each interface by using mappings.

The interfaces are not assigned when they are created, but during logical routing. Furthermore, you also create interface pairs when you define interface mappings.

### Assigning Without Importing

The advantage of importing RFCs and IDocs is that a description is saved in the Integration Repository for all interfaces in a cross-system process. This enables you to directly reference the technical name during mapping, routing, or from a business scenario, without having to search for this information in the corresponding systems. However, the Integration Builder only permits you to enter these technical names manually, thus enabling systems from which no interfaces can be imported to be connected.

- For technical reasons, it is only possible to import RFCs and IDocs for SAP systems Release 4.0 or higher. However, the RFC and IDoc adapters can be implemented with SAP systems Release 3.1l and higher. In this case you must enter the interface names manually. More information about IDoc and RFC interfaces Release 3.1l and higher is available in the Interface Repository (http://ifr.sap.com).
- The [Adapter Engine](http://ifr.sap.com) supports the connection of external systems that are not necessarily connected by means of an interface (for example, the file adapter). When configuring the inbound adapter, specify instead the ID of the logical sender and receiver by using the respective business system, a namespace, and any interface name.

Although you cannot import this interface information into the central Integration Repository, the Integration Engine still recognizes the corresponding communication partner, provided you entered the [namespace](http://ifr.sap.com) and the interface name correctly in mapping and routing.

### Developing Message Interfaces

#### Purpose

By using platform-independent message interfaces you can define the type of communication and the data to be exchanged before you actually implement the cross-system process. Also see: [Introduction to Interface Development](http://ifr.sap.com).
Overview

The following illustration displays the class model for interface objects in the Integration Builder:

As with all repository objects, interfaces are organized by using namespaces that are assigned to a software component version. Imported interfaces (not shown in the class model) are assigned to a namespace belonging to an interface type, however you can create new namespaces for message interfaces (see also: Organization of Shipment Contents [Seite 56]).

Message interfaces comprise message types and data types. This two-layer structure uses WSDL (Web Service Description Language) and is oriented towards maximum reusability. Fault message types are a special kind of message type that you use to handle application-specific errors. Message interfaces, message types, fault message types and data types are all commonly referred to as interface objects.

The introduction of an intermediate message type layer seems at first glance unnecessary, however it is required in XML so that a message can be handled as a separate instance. Data types in XML schema do not yet define an instance of this type because a data type does not yet define an element.

Prerequisites

Before you can develop message interfaces in the Integration Builder, you must import a software component version and create one or more namespaces for it (see: Displaying/Changing Software Component Versions [Seite 51]).

For more information about system prerequisites, see the Prerequisites section in Introduction to Interface Development [Seite 119].
**Process Flow**

The process flow outlined here only covers the technical side of message interface development and not general guidelines for designing interfaces and data types.

1. To define the contents of the messages to be exchanged you require data types. Establish whether data types already exist for the interfaces you require. If necessary, create new data types. Also note that when you import an interface, no data types are created for the counterpart (outbound or inbound) interface (also see: Communication Partners).

2. To reference a message yourself (in mapping, for example), you must use a message type that links to a data type. Check whether a corresponding message type already exists and, if necessary, create a new message type. In the simplest case, you require a message type for the request message and for synchronous communication you also require a message type for the response message.

3. Optionally, you can use fault messages to handle application-specific errors that occur on the inbound side. Check whether a corresponding fault message type already exists and, if necessary, create a new fault message type.

4. To send a message between two systems you always require an interface pair, namely an outbound interface and an inbound interface:
   - To send messages using a communication partner on the inbound side, you must create an outbound message interface.
   - To implement a service on the inbound side that can receive messages, you must create an inbound message interface.

   The communication partner of a message interface does not necessarily have to be a message interface. You can also connect message interfaces with senders or receiver that are connected to SAP Exchange Infrastructure by means of an adapter.

5. Compile documentation for the objects you created.

**Result**

The Integration Builder saves the message interface with the referenced message types and data types in the Integration Repository. You can now do the following:

- Use proxy generation to generate a Java or ABAP proxy from a message interface.
- Begin designing mappings while simultaneously developing runtime components.
- Reference the message interface while designing business scenarios.
- Optionally export an XSD or WSDL document as a local file from the repository.

You can then configure the contents in the Integration Directory.
Message Interface

Use
You use a message interface to describe an interface independent of a platform or programming language. You can then use this interface to exchange messages between application components using SAP Exchange Infrastructure. Depending on the communication parameters defined for a message interface, it can either have the task of sending a request (outbound) or receiving a request (inbound). Assigning an outbound interface to an inbound interface enables multiple communication partners to exchange messages with each other. Therefore, the following connections are possible for message interfaces:

- A message interface with other message interface
- A message interface with an imported interface
- A message interface with an interface configured by an adapter

By using proxy generation, message interfaces enable ABAP and Java applications to be interconnected.

Prerequisites
Message interfaces reference message types and fault message types that ideally you have already created. This is not compulsory in order to save a message interface but this would mean that its definition would be incomplete.

For more information see the Prerequisites section in Introduction to Interface Development.

Features
Communication Parameters
When you create a message interface you define the communication parameters by using the attributes Mode and Direction. There are four different types of message interfaces you can create:

<table>
<thead>
<tr>
<th></th>
<th>Synchronous</th>
<th>Asynchronous</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inbound</td>
<td>Synchronous Inbound</td>
<td>Asynchronous Inbound</td>
</tr>
<tr>
<td></td>
<td>Message Interface</td>
<td>Message Interface</td>
</tr>
<tr>
<td>Outbound</td>
<td>Synchronous Outbound</td>
<td>Synchronous Outbound</td>
</tr>
<tr>
<td></td>
<td>Message Interface</td>
<td>Message Interface</td>
</tr>
</tbody>
</table>

Depending on the attributes you define for the message interface, you either need to reference one or two message types in Message Types. If you want to handle application-specific errors or persist them in monitoring, also assign the corresponding fault message types to the message interface. The Integration Builder restricts the input options according to the attributes selected, as follows:

<table>
<thead>
<tr>
<th>Reference Message Types</th>
<th>Fault Message(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synchronous Outbound Message Interface</td>
<td>Output Message Type, Input Message Type yes</td>
</tr>
<tr>
<td>Synchronous Inbound Message Interface</td>
<td>Input Message Type, Output Message Type yes</td>
</tr>
<tr>
<td>Synchronous Outbound Message Interface</td>
<td>Output Message Type no</td>
</tr>
</tbody>
</table>
Asynchronous Inbound  
Message Interface  
| Input Message Type | yes |

Therefore, the input message type defines the expected message, while the output message type defines the message sent.

**Enhanced Functions**

Once you have completed a message interface you can do the following:

- Generate a proxy for use at runtime by using ABAP proxy generation [Seite 190] or Java proxy generation [Seite 198].
- Export the interface description as an XML document.
  
  The XML format complies with WSDL (Web Service Description Language). This format is subject to the standardization process of the W3C consortium.

**Activities**

1. Create a message interface on the design maintenance screen of the Integration Builder (see: Creating a New Object [Seite 40]).
2. Enter a description for your message interface.
3. Define the communication parameters [Seite 120] in the Attributes frame.
4. In Message Types, reference the required input or output message types either by using the search help [Seite 48] or by entering the message types manually.
5. If the interface is not an asynchronous outbound message interface, you have the option of referencing fault message types in the lower table in the Message Types frame. To do so, use the following functions:

**Edit References to Fault Messages**

<table>
<thead>
<tr>
<th>Function</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Add an additional line to the table. Enter the fault message type you want either manually or by using the search help (see below)" /></td>
<td>Add an additional line to the table. Enter the fault message type you want either manually or by using the search help (see below)</td>
</tr>
<tr>
<td><img src="image" alt="Call the search help [Seite 48] to select a fault message type" /></td>
<td>Call the search help [Seite 48] to select a fault message type</td>
</tr>
<tr>
<td><img src="image" alt="Change the sequence of the fault message types" /></td>
<td>Change the sequence of the fault message types</td>
</tr>
<tr>
<td><img src="image" alt="Delete a reference to a fault message type" /></td>
<td>Delete a reference to a fault message type</td>
</tr>
</tbody>
</table>

6. Save your changes.
**Message Type**

**Use**

A message type comprises a data type that describes the structure of a message. At the following points in SAP Exchange Infrastructure you can refer to the message to be exchanged at runtime by using the message type:

- More than one message interface can use the same message type. For example, an asynchronous outbound message interface and an asynchronous inbound message interface can reference the same message type because the request message does not need to be mapped.
- When defining a message mapping you can directly reference message types to map messages from an outbound interface to messages from a receiver interface.

For technical reasons a data type is not sufficient to describe the instance of a message. In XML schema, data types are defined as abstract types that are not yet fixed to an element. You can only describe an instance of a message when you have specified a data type as an element type. Therefore, a message type defines the root element of a message.

A message type does **not** define the direction of the message exchange, in other words, whether it is a request or a response. In exceptional cases you can use the same message type for both the request and the response message.

**Prerequisites**

You have already created a data type that you want to reference from your message type. For more information see the Prerequisites section in Introduction to Interface Development [Seite 119].

**Features**

A message type:

- comprises exactly one data type that determines the contents of the message to be transferred.
- is identified by a name and a namespace. The name (together with the namespace) of a message type is located in the uppermost tag of the exchanged message. This is an element tag.

You have defined a message type `GreetingRes` that references a simple data type of type `xsd:string`. The namespace of the message type is `http://sap.com/xi/myExample`. The corresponding payload might look as follows:

```xml
<?xml version="1.0" encoding="utf-8" ?>
    Hello Mrs. Miller
</ns1:GreetingRes>
```

You can specify how a message type is used in more detail in the description.

**Activities**
1. Create a message type on the design maintenance screen of the Integration Builder (see: Creating a New Object [Seite 40]).

2. Enter a description for your message type.

3. Reference a data type, either by entering the corresponding name and namespace of the data type in the frame Data Type, or by using the search help [Seite 48].

4. Save your changes.

**Example**

In the example below, two message interfaces each use the same message type for the request message and the response message:

```
<table>
<thead>
<tr>
<th>Message Interface (Outbound)</th>
<th>Message Type</th>
<th>Message Interface (Inbound)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output Message Type</td>
<td>myReq</td>
<td>Input Message Type</td>
</tr>
<tr>
<td>Input Message Type</td>
<td></td>
<td>Output Message Type</td>
</tr>
</tbody>
</table>
```

**Fault Message Type**

**Use**

Fault message types are designed for application-specific errors that occur on the inbound side and that are reported back to the sender or persisted in monitoring.

- When an application-specific error occurs on the inbound side, instead of returning a response message to the sender, the application returns a fault message to handle the error.
- The fault message of the application for an asynchronous inbound interface is persisted for monitoring [Seite 290].

Application-specific here means that the application on the inbound side triggers the error itself because, for example, the request message did not contain sufficient information.

**Integration**

A fault message type is a special message type that can be used in message interfaces [Seite 126]. As with a message type, a fault message type is made up of data types (see below). Fault message types are not permitted for asynchronous outbound message interfaces.

**Prerequisites**

For more information see the Prerequisites section in Introduction to Interface Development [Seite 119].
**Features**

**Structure of a Fault Message Type:**

<table>
<thead>
<tr>
<th>Data Part</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Standard Data</strong> (obligatory)</td>
<td>Use this data part to return the fault message standard information for an error at runtime. All fault message types reference for this part the data type ExchangeFaultData and, indirectly, the data type ExchangeLogData. These data types are automatically created in a namespace when you create the first fault message type there.</td>
</tr>
<tr>
<td><strong>Additional Data</strong> (optional)</td>
<td>Use this data part to attach any additional application-specific information to the fault message. To do so, reference any data type in the same software component version or in a sub-software component version.</td>
</tr>
</tbody>
</table>

When a proxy is generated from a message interface, an exception class is generated for a fault message type that you can use to handle application errors at runtime (also see the example below).

Also see: [Fault Messages](ABAP) or [Fault Messages](Java).

**Activities**

1. Create a fault message type on the design maintenance screen of the Integration Builder (see: Creating a New Object).
2. Enter a description for your fault message type.
3. The fault message only comprises the standard data part in the default setting. If you require the additional data part, select the checkbox Additional Data and specify a data type for the structure of this data. The data type can be in the same software component version or in a sub-software component version.
4. Save your changes.

**Example**

In the example below, an ABAP receiver reports back to a Java sender that an application error has occurred during inbound processing:

![Integration diagram](image)

```
try {...}
catch FmException;
```

SAP Exchange Infrastructure (BC-XI)  623  130
To handle the error, you first create a fault message type Fm in the Integration Repository that is referenced by both message interfaces. When you generate the proxies for the message interfaces, the exception classes FmException (Java) and CX_FM (ABAP) are generated. The graphic shows how the error is transferred after the application on the ABAP side has transferred the corresponding data to the exception class CX_FM and triggered it with RAISE. The proxy runtime generates a fault message from the exception class data that is sent to the sender. The exception for the exception class FmException is then triggered at the sender so that the error can be handled there.

Data Types

Use

A data type is a basic unit for defining the structure of the data for a message type and, as a result, for the message to be exchanged.

Integration

A data type is an XML Schema Definition (XSD). This definition determines how the valid payload [Seite 15] of a message will look. Data types in XML schema are only an abstract description of a structure that you can assign to elements in an XML schema definition. A message type [Seite 128] or a fault message type [Seite 129] that links to a data type then determines the name of the root element of a message.

Prerequisites

For more information see the Prerequisites section in Introduction to Interface Development [Seite 119].

Features

Data Type Categories

<table>
<thead>
<tr>
<th>Category</th>
<th>Meaning</th>
<th>Examples</th>
</tr>
</thead>
</table>
| Built-In Type  | Built-in data types are part of the XML schema language and have the prefix xsd. (They can also be subdivided into primitive and derived data types, but this is not important in the definition of data types). XML schema defines a value range for each built-in data type. | xsd:string  
                 |                                                                         | xsd:decimal  
                 |                                                                         | xsd:integer |
| Simple Type    | Simple data types refine built-in data types, for example, by limiting the length of a string or by specifying a specific value range. Like built-in data types, they do not contain any further elements. | <complexType name="germanPhoneNumber" base="string">  
                 |                                                                         | <pattern value="\d{4}\d{6}="/>  
                 |                                                                         | </complexType> |
| Complex        | Data types that can contain                                             | <complexType name="person">                 |
Type further elements and attributes [Seite 135]. Elements and attributes can in turn reference built-in, simple, complex, or global data types. This enables large complex data types to be structured using smaller complex data types.  

<element name="firstname" type="string" />  
<element name="lastname" type="string" />  
</complexType>

You can refine simple data types and fields of complex types by using facets [Seite 136].

Referencing of Data Types

You can structure data types in a nested way by referencing data types from a complex data type. However, you must ensure that the referencing and the referenced data type are shipped together otherwise the data type definition is not complete. For this reason, the following restrictions exist for the use of data types (also see: Versioning and Compatibility [Seite 66]).

Referencing of Data Types

<table>
<thead>
<tr>
<th>Category</th>
<th>Can Be Referenced In</th>
</tr>
</thead>
<tbody>
<tr>
<td>Built-In Type</td>
<td>All software component versions</td>
</tr>
<tr>
<td>Simple Type</td>
<td>The same software component version or a sub-software component version of the referencing data type.</td>
</tr>
<tr>
<td>Complex Type</td>
<td>The same software component versions or sub-software component versions of the referencing data type.</td>
</tr>
</tbody>
</table>

You cannot structure a data type recursively, in other words, references of type \textit{Data Type}_1 \rightarrow \textit{Data Type}_1 and transitive references of type \textit{Data Type}_1 \rightarrow \textit{Data Type}_2 \rightarrow \textit{Data Type}_1 are not permitted ("\rightarrow" means "referenced" in this example).

Activities

1. Create a data type on the design maintenance screen of the Integration Builder (see: Creating a New Object [Seite 40]).
2. There are two further tab pages on the tab page for data types:
   - Choose the Type Definition tab page to use the XSD editor [Seite 133] to describe your data type in XML schema. The editor simplifies the structure of the data type by limiting it to the essential language elements only and converting it to a valid XML schema definition.
   - Choose the tab page XSD Definition to display the XML schema definition as text.
3. Use the XSD editor to define simple data type [Seite 137] or a complex data type [Seite 138].
4. Save your changes.
XSD Editor

Use

The XSD editor enables you to define data types [Seite 131] in XML schema, without you requiring extensive knowledge of XML. In the XSD editor you create a data type using a table that returns the structure of the data type. You can define both simple and complex data types that the editor automatically converts to XML Schema Definition (XSD). The XML Schema Definition determines which tags can be transferred with which values in the payload [Seite 15].

When defining data types, the selection of values that you can enter in the table is context-dependent. Therefore, you cannot enter invalid entries.

Prerequisites

You have already created a data type. To access the XSD, select the Definition tab page (see: Data Types [Seite 131]).

Features

The XSD editor comprises a toolbar, a tabular input area, and a context menu. The uppermost line in the table (the root element) determines the type of the data type:

- There are no other table lines for simple data types.
- The sub-nodes of the root element determine the structure of complex data types. Elements and attributes [Seite 135] can also be sub-nodes.

The XSD editor automatically knows the difference between simple and complex types by determining whether the root element has any sub-nodes. For this reason, the Category column is locked. The tabular input area contains the following columns for which differing values (or no values at all) are permitted, depending on the line type:

Columns in the XSD Editor

<table>
<thead>
<tr>
<th>Column</th>
<th>Meaning</th>
<th>Mouse Clicks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structure</td>
<td>Nodes inserted below the root element are represented in a hierarchy using indents in this column. The field names within a data type must not be the same. It is possible to change all field names at a later stage, except that for the root element.</td>
<td>1 / 3</td>
</tr>
<tr>
<td>Category</td>
<td>This column declares whether the root element has sub-nodes or not (see above). You cannot change values in this column.</td>
<td>-</td>
</tr>
<tr>
<td>Type</td>
<td>Select a built-in data type or reference to an existing data type for an element or attribute. You must specify a type for attributes.</td>
<td>1/2</td>
</tr>
<tr>
<td>Occurrence</td>
<td>Determines how often an element occurs in the message. The values 0 to 2147483647 are permitted. You can determine whether the attribute is optional or mandatory for attributes.</td>
<td>2</td>
</tr>
</tbody>
</table>
**Details**
You can only make an entry here if you specified a built-in type for the element or the attribute. Depending on whether the type is numerical or alphanumerical, you can select facets [Seite 136] from the selection dialog for the column to specify the value range of the type more precisely (for example, restricting the string length).

**Description**
Informal description of a node.

You can call a selection dialog for cells in the Type, Occurrence, and Details columns by double clicking. You can also call a selection list in the Type column with one click. To change a field name below the root element, select it by triple-clicking. The name of the root element itself cannot be changed, however.

**Functions of the XSD Editor**

<table>
<thead>
<tr>
<th>Function</th>
<th>Access from Toolbar (TB) and/or Context Menu (CM)</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Icon" /> TB/CM</td>
<td>Call the search help [Seite 48] to reference other data types in the Type column.</td>
<td></td>
</tr>
<tr>
<td><img src="image2" alt="Icon" /> TB/CM</td>
<td>Opens a data type on a new tab page that is referenced in the Type column.</td>
<td></td>
</tr>
<tr>
<td><img src="image3" alt="Icon" /> TB/CM</td>
<td>Inserts a new element on the same hierarchy level as the line currently selected.</td>
<td></td>
</tr>
<tr>
<td><img src="image4" alt="Icon" /> TB/CM</td>
<td>Inserts multiple elements on the same hierarchy level as the line currently selected.</td>
<td></td>
</tr>
<tr>
<td><img src="image5" alt="Icon" /> TB/CM</td>
<td>Deletes a line. Sub-nodes are also deleted if any exist.</td>
<td></td>
</tr>
<tr>
<td><img src="image6" alt="Icon" /> TB/CM</td>
<td>Expands the hierarchy structure of the line currently selected. Unlike the functions below, nodes that reference another data type are not expanded.</td>
<td></td>
</tr>
<tr>
<td><img src="image7" alt="Icon" /> TB/CM</td>
<td>Expands all levels of the hierarchy structure for a selected line.</td>
<td></td>
</tr>
<tr>
<td><img src="image8" alt="Icon" /> TB/CM</td>
<td>Collapses all levels of the hierarchy structure for a selected line.</td>
<td></td>
</tr>
<tr>
<td><img src="image9" alt="Icon" /> TB/CM</td>
<td>Maximizes the column width.</td>
<td></td>
</tr>
<tr>
<td><img src="image10" alt="Icon" /> CM</td>
<td>Calls the value help for the Type, Occurrence, and Details columns.</td>
<td></td>
</tr>
<tr>
<td><img src="image11" alt="Icon" /> CM</td>
<td>Inserts a subelement beneath the element for which you called the context menu.</td>
<td></td>
</tr>
<tr>
<td><img src="image12" alt="Icon" /> CM</td>
<td>Insert an attribute for the element for which you called the context menu.</td>
<td></td>
</tr>
<tr>
<td><img src="image13" alt="Icon" /> CM</td>
<td>Copies the selected node, including all its subnodes to the clipboard.</td>
<td></td>
</tr>
<tr>
<td><img src="image14" alt="Icon" /> CM</td>
<td>Cuts the selected node, including all its subnodes to the clipboard.</td>
<td></td>
</tr>
<tr>
<td><img src="image15" alt="Icon" /> CM</td>
<td>Inserts the node from the clipboard onto the same level as the node for which you called the context menu.</td>
<td></td>
</tr>
</tbody>
</table>
The contents of the clipboard can only be reinserted within a data type.

**Activities**

Use the XSD editor to do the following:

- [Create a simple data type](Seite_137)
- [Create a complex data type](Seite_138)

You can display the corresponding XML schema on the **XSD Display** tab page and export it as a file (also see: [Exporting XSD and WSDL Documents](Seite_139)).

---

**Elements and Attributes**

You create complex data types using elements and attributes in the [XSD editor](Seite_133):

<table>
<thead>
<tr>
<th>Node Type</th>
<th>Use</th>
<th>Example Instance</th>
</tr>
</thead>
</table>
| Element   | Create structured data types. Elements that have a type cannot contain subelements. | `<myElem>
  <f1> Value of f1 </f1>
  <f2> Value of f2 </f2>
</myElem>` |
| Attribute | Add attributes to elements. Attributes cannot usually have subnodes. | `<myElem
  myAttr="AttributeValue">
  Element Value
</myElem>` |

You can flag an attribute as *optional* or *required* in the *Occurrence* column. These values mean the same for elements with an occurrence of $0..1$ or $1$. The only difference between elements and attributes is that attributes cannot have subnodes and that the same attribute cannot be used more than once in an element.

**Frequently Occurring Elements**

XML schema does not recognize tables but permits instead the definition of elements that can occur frequently in a schema (maxOccurs="unbounded"). Depending on the target language, [proxy generation](Seite_186) generates either a table type with a structure for the line type ([ABAP](Seite_192) example), or class for accessing using a list ([Java](Seite_202) example).
Facets

Use
Facets enable you to restrict the value range of a data type in XML schema to a particular subset. This enables you to determine exactly which value is expected in a special case for a simple data type, an element, or an attribute. For example:

- A telephone number (use of a pattern on a string)
- A selection of permitted values, for example to classify the different classes for a flight (using an enumeration)

Prerequisites
Before you can use a facet, you must have selected a built-in data type in the Type column for your simple data type, element, or attribute. Otherwise the input field in the Details column remains locked.

Features
Using the XSD editor, you can further restrict the value range of a built-in data type for simple data types, elements or attributes.

Input Options in the Details Column

<table>
<thead>
<tr>
<th>Facet</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>enumeration</code></td>
<td>Restricts the value range to a set of individual values. The values must be separated by blanks in the XSD editor.</td>
</tr>
<tr>
<td><code>fractionDigits</code></td>
<td>Specifies the number of places permitted after the comma. Only <code>xsd:nonNegativeInteger</code> type values are valid.</td>
</tr>
<tr>
<td><code>length, maxLength, minLength</code></td>
<td>Exact (length), maximum (maxLength), or minimum (minLength) length of a data type. The XSD editor only permits these facets for data types that are based on <code>xsd:string</code>. The value of this facet must be of type <code>xsd:nonNegativeInteger</code>.</td>
</tr>
<tr>
<td><code>maxExclusive, maxInclusive</code></td>
<td>Upper limit for the value set exclusive (maxExclusive) or inclusive (maxInclusive) of the specified value. The value of this facet must not exceed the value range of the data type that is restricted by the facet.</td>
</tr>
<tr>
<td><code>minExclusive, minInclusive</code></td>
<td>Lower limit for the value set exclusive (minExclusive) or inclusive (minInclusive) of the specified value. The value of this facet must not exceed the value range of the data type that is restricted by the facet.</td>
</tr>
<tr>
<td><code>pattern</code></td>
<td>Specifies a pattern for string-based data types. The pattern must have the form of a regular expression that describes a set of appropriate character sequences.</td>
</tr>
<tr>
<td><code>totalDigits</code></td>
<td>Specifies the total number of digits in a number. Only <code>xsd:positiveInteger</code> type values are valid.</td>
</tr>
</tbody>
</table>
| `whiteSpace`                  | Specifies how to apply white space characters (line feed, tabs, blanks, and carriage returns). Three values are permitted:  
  - **preserve**: Retains all white space characters.  
  - **replace**: Replaces every line feed, tab, and carriage return with a blank.  
  - **collapse**: As for replace but subsequent blanks are replaced by a single blank and leading and final blanks are deleted. |
Regular Expressions in XML Schema

Regular expressions originate from the Unix world and are expressions that you can use to describe any type of string. Regular expressions for XML schema are defined under http://www.w3.org/TR/xmlschema-2/#regexs. Instead of repeating this definition, here are a few common examples of how regular expressions are used:

<table>
<thead>
<tr>
<th>Regular Expression</th>
<th>Appropriate Strings (Separated by Commas and Blanks)</th>
</tr>
</thead>
<tbody>
<tr>
<td>[abcdefg]</td>
<td>a, b, c, d, e, f, g</td>
</tr>
<tr>
<td>[A-Z][A-Z][A-Z]</td>
<td>ABC, DEF, SAP, SPD, CDU, FDP, PDS</td>
</tr>
<tr>
<td>[0-9][0-9][0-9]</td>
<td>007, 459, 471, 999, 879</td>
</tr>
<tr>
<td>[a-z][0-9]*</td>
<td>a, b, c, d, e, i12784937, e3321, t4000, b345</td>
</tr>
<tr>
<td>[a-z][0-9]+</td>
<td>a304, i20, e3012, t5, b90000</td>
</tr>
<tr>
<td>([a-z][0-9]+</td>
<td>a, ab, abc, abcd, 0, 1, 01, 012, 0123</td>
</tr>
</tbody>
</table>

When using a star (*), the character may not occur at all in the string, or it can occur an unlimited number of times. When using a plus (+), the character must occur at least once in the string. The square brackets specify the character selection.

Additionally, there are also special characters that can be introduced with a backslash, for example:

- \d: A number
- \c: A character in the sense of a character permitted for names in XML schema

Defining Simple Data Types

Procedure

1. Create a data type in your namespace on the design maintenance screen of the Integration Builder (see: [Creating a New Object][40]).

2. Choose the Definition tab page. The XSD editor automatically selects the category Simple Type when you create a data type.

3. In the Type column, define a built-in data type by double clicking the cell and copying a value from the list box.
You can also reference other simple data types in the toolbar or in the list box by using the search help. You cannot reference complex data types.

You cannot make any entries for simple data types in the Occurrence column.

4. To define the facets [Seite 136] of the simple data type in the Details column, call the input help by double clicking.

5. Enter a description for your simple data type.

6. Save your changes.

Result

You can now use your simple data type in other data types or (fault) message types.

---

Defining Complex Data Types

Procedure

1. Create a data type in your namespace on the design maintenance screen of the Integration Builder (see: Creating a New Object [Seite 40]).

2. Choose the Definition tab page. The XSD editor automatically selects the category Simple Type when you create a data type.

3. To insert a subnode, call the context menu for the root element and choose either Insert Subelement or Insert Attribute. This is only possible if the root element has not been assigned a type.

   If a type has already been specified in the Type column, you can delete it again by selecting the type with the cursor, deleting the selected text by choosing the Del button, and then confirming the change by choosing Return.

4. The type then automatically changes to Complex Type. Using the context menu, insert further elements and attributes to build up the structure of your new complex type. Note the following:

   - Unlike elements, attributes cannot have subnodes. Therefore, you can build up structures using elements, but use attributes to specify additional information for an element in a message. Also see: Elements and Attributes [Seite 135].
   - Subelements can only be inserted for elements that do not have a type.
   - If a single substructure becomes too complicated, you are recommended to use multiple small data types instead, especially when these can then be used elsewhere. To reference other data types, choose Search Help [Seite 48] in the list box for the Type column.

5. As long as the element concerned is not the root element, you can change the field name by triple-clicking it.
Exporting XSD and WSDL Documents

Use
The Integration Builder editors access repository objects directly. For example, you can load message types in the mapping editor to define a message mapping. You can also use tools from a third-party to design a mapping and then import the XSLT or Java mapping into the repository. The Integration Builder supports the export of interface objects for such scenarios.

<table>
<thead>
<tr>
<th>Interface Object</th>
<th>Export Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message Interface</td>
<td>WSDL (Web Service Description Language)</td>
</tr>
<tr>
<td>Data type, (fault) message type</td>
<td>XSD (XML Schema Definition)</td>
</tr>
</tbody>
</table>

If the object to be exported references another interface object, then this object is also included in the exported file. This means that when you export a message interface the referenced message types and data types are part of the WSDL file (in the XSD namespace).

You can also call the XSD format as text for data types by selecting the XSD Representation tab page.

At present it is not possible to import XSD or WSDL documents.

Procedure
You are on the design maintenance screen in the Integration Builder.

1. Open an interface object.
2. The export function is located in the Tools menu of the interface object:
   - For message interfaces, choose Tools → Export WSDL.
   - For (fault) message types and data types, choose Tools → Export XSD.

   The system displays a dialog window to navigate in the file system of your computer.
3. Navigate to the directory where you want to save the export.
4. Enter a name for the file.
   - You can choose the file ending yourself. If you choose .xml, for example, you can open the existing file in a browser.
5. Save the file.

Result
The system saves the document in the specified directory.
Connection Using Adapters and Imported Interfaces

Purpose

If interfaces are already available for exchanging data in a system, you can connect these to SAP Exchange Infrastructure using adapters. In the case of RFCs and IDocs you can import an XML description of this interface to the Integration Repository. This has the advantage that you can access the signature and structures of the imported interface used centrally. Furthermore, you can access the technical names of the imported interfaces using the value help (for example, in mapping or logical routing).

Generally, you do not need to import interfaces to be able to use adapters because you can also enter the interface names manually.

Prerequisites

RFCs and IDocs can be imported from SAP systems Release 4.0 and higher. However, you must have the authorization required to import from a system.

It is the adapter type and not the import function that determines whether a system can be connected to SAP Exchange Infrastructure using an adapter. Note the prerequisites for the respective adapter in this case.

The interface to be used should already exist in the system.

Process Flow

The procedure described below only applies for interfaces that cannot be imported into the Integration Repository. Therefore, if applicable, ignore the first step.

1. Import the RFC or IDoc that you want to use to exchange messages using SAP Exchange Infrastructure to the design maintenance screen of the Integration Builder (also see: Importing Interfaces from an SAP System [Seite 141]).

2. The imported interface corresponds to either an inbound or an outbound interface. Depending on the communication partner, create the corresponding counterpart in the Integration Repository. If the counterpart is a message interface for communication using proxies, you must create appropriate data types and message types for the message interface for the data that is to be transferred using the adapter (see: Developing Message Interfaces [Seite 123]).

An ABAP outbound proxy is to exchange data with an RFC interface that implements an inbound service. Besides the imported RFC interface, you must also create a message interface with appropriate message types and data types in the Integration Repository. Adapter to adapter communication is also possible. Also see: Communication Partners [Seite 121].

3. Different adapters generate, or expect different XML formats. Likewise, the proxy runtime generates and expects a different XML format to the various adapters. For this reason, you must map the XML structures to each other using a message mapping [Seite 148]. The parameters of imported interfaces can be loaded directly as source or target structures into the mapping editor (also see: Designing a Mapping [Seite 143]).
You can also use the mapping editor to import XML files from the local file system and therefore to define a message mapping for interfaces that cannot be imported.

Result

The subsequent steps for design and configuration in the Integration Builder correspond to the procedure for developing message interfaces (however, you need to define different end points depending on the adapter).

Importing Interfaces from an SAP System

Use

By importing interfaces you make the signature and the technical name of the interface known to the Integration Builder, thus easing further design and configuration:

- In mapping you can import the parameters and structures for the imported interface directly to the mapping editor for the source or target structure.
- During the design and configuration phase you must constantly reference the interfaces you have used. If the interface has been imported, you can avoid typing errors by using the value help.

You do not have to import an interface to use it in SAP Exchange Infrastructure, however, it eases the entire design and configuration process at a later stage. You can import RFC function modules, BAPIs, and IDocs from SAP systems Release 4.0 or higher.

Prerequisites

The import of interfaces must be permitted for the software component versions for which you want to import interfaces from an SAP system. To check this, call the context menu for the software component versions concerned and choose Open. The checkbox Interfaces from SAP Systems Permitted must be selected on the Definition tab page, and the connection data for the SAP system concerned must be maintained (also see: Displaying/Changing Software Component Versions [Seite 51]). You must also log on with a user that has the authorizations required to import interfaces. For a SAP Web Application Server Release 6.20, the following authorizations are required:

Authorization Required to Import from Systems Based On SAP Web Application Server 6.20

<table>
<thead>
<tr>
<th>Authorization Object: S_RFC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authorization Field</td>
</tr>
<tr>
<td>ACTVT</td>
</tr>
<tr>
<td>RFC_NAME</td>
</tr>
<tr>
<td>RFC_TYPE</td>
</tr>
</tbody>
</table>
Authorization Object: S_DEVELOP

<table>
<thead>
<tr>
<th>Authorization Field</th>
<th>Authorization Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTVT</td>
<td>Display</td>
</tr>
<tr>
<td>DEVCLASS</td>
<td>*</td>
</tr>
<tr>
<td>OBJNAME</td>
<td>*</td>
</tr>
<tr>
<td>OBJTYPE</td>
<td>*</td>
</tr>
<tr>
<td>P_GROUP</td>
<td>*</td>
</tr>
</tbody>
</table>

**Importing Interfaces from an SAP System**

You are on the design maintenance screen in the Integration Builder.

1. Expand the subnodes for the software component version for which you want to import interfaces.
2. Call the context menu for the subnode *Imported Objects* of this software component version and choose *SAP Import*.
   
The system displays the *SAP Import* dialog on the tab page *Logon*.
3. In the *User Account* frame, enter the logon data for a user that has sufficient authorization for the import (see *Prerequisites*), and choose the *Continue* button ( ).
4. The dialog switches to the *Selection* tab page so you can select the interfaces to be imported. The default setting displays the RFC modules and IDocs under the corresponding node as an alphabetically ascending list. You can use the following functions to find and select an interface:

**Selection Tab Page Functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Refreshes the object list for all subnodes of a selected node</td>
</tr>
<tr>
<td></td>
<td>Searches for interfaces in the entire object list. This search corresponds to the search in the [Integration Builder navigation tree][Seite 31].</td>
</tr>
<tr>
<td></td>
<td>Opens a list in the lower area of the tab page in which all previously selected interfaces are displayed. This enables you to check whether there are interfaces in your selection that you do not want to import and so can delete them.</td>
</tr>
<tr>
<td></td>
<td>Closes the list of selected interfaces.</td>
</tr>
<tr>
<td></td>
<td>Displays the imported interfaces ordered according to the application component hierarchy.</td>
</tr>
<tr>
<td></td>
<td>Displays the imported interfaces in a list in alphabetical order.</td>
</tr>
</tbody>
</table>

5. In the list or in the application component hierarchy select the RFCs or IDocs that you want to import and choose *Continue* ( ).
6. The system navigates to the *Import* tab page that comprises a status window and the list *Interfaces Selected for Import*. To start the import, choose *Start* ( ). To select further interfaces to be selected, choose *Back* ( ).

**Result**

The selected interfaces are imported from the SAP system and are then available in your software component version under *Imported Objects*.
Designing a Mapping

Purpose

In SAP Exchange Infrastructure, interface descriptions of function calls (interfaces) are saved in XML format. The parameters of these interfaces determine the contents of the messages that are exchanged between systems (see also: Interface-Based Message Processing [Seite 17]).

A mapping refers to the mapping of interfaces or messages to other interfaces or messages. You may need to use mappings of this type in a cross-system business process for a variety of reasons:

- A business partner expects other technical field names or another structure for transferring data.
- For technical reasons. For example, an outbound RFC adapter [Extern] connected to SAP Exchange Infrastructure expects to receive messages from the Integration Engine in the format that it can understand, namely RFC XML. It then creates an RFC call in a system assigned to it. If the Integration Engine receives a message from an outbound proxy [Extern], the corresponding message must be mapped to RFC XML using a mapping so that the outbound adapter can process it.

In the case of proxies, outbound means outbound from the point of view of the application. However, in the case of adapters, it means from the point of view of the Integration Engine.

You can define one or more mappings for an interface pair at design time [Seite 19], and save them in the Integration Repository. At configuration time [Seite 21], you can then select the mapping that corresponds to a specified business scenario. The Integration Engine transforms the message at runtime, in accordance with the saved mappings. Ideally, no mapping is required. In other words, the sender and receiver interface parameters are identical.

Introductory Notes

Mapping is an essential part of SAP Exchange Infrastructure. At design time mappings can be saved in the Integration Repository and then selected for a business process at configuration time (see: Defining Mapping Relations [Seite 226]). The Integration Engine executes the mappings saved in the Integration Directory at runtime, following receiver identification. If no mapping is required for a connection then the Integration Engine skips the mapping step.

Features

SAP Exchange Infrastructure differentiates between the following at runtime:

- Mapping Programs: Used to implement the mapping.
- Interface Mappings: Used to register mapping programs for an interface pair. You can also enter multiple mapping programs that are executed in succession.

Mapping Programs

The Integration Builder supports the following:
• The development of message mappings using a graphical editor integrated in the Integration Builder. Java classes are generated for use at runtime from a graphical description. You also have the option of enhancing the standard functions with user-defined functions.

• The import of XSLT or Java mapping as an archive. If, for example, XSLT mappings already exist or are to be developed using third-party tools, you can use the import function to import such mapping programs into SAP Exchange Infrastructure.

**Imported Archives**

<table>
<thead>
<tr>
<th>Type</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>XSLT (eXtensible Stylesheet Language Transformations)</td>
<td>You use XSLT to convert an XML document to another document. This is achieved by using XSLT commands to search for tags in the source document and replacing them with other tags or values in the target document. It is also possible to switch to a Java program from XSLT.</td>
</tr>
<tr>
<td>Java Program</td>
<td>The message is transferred to a Java program as an inbound stream and can be imported using any parser (for example, SAX or DOM). The modified message is then transferred to an outbound stream. A Java program can also be a help program that in turn is used in another Java program or self-defined functions of a message mapping.</td>
</tr>
</tbody>
</table>

**Overview of Mapping Types**

You can define mappings for an interface pair by using message interfaces and message types in the Integration Repository.

You can also define the corresponding mappings for outbound interfaces, inbound interfaces, IDocs, RFCs or other interfaces connected using an adapter.
As illustrated in the graphic, a request contains the data that is defined using the output message type of the outbound message interface. At runtime, a request passes from the outbound to the inbound interface. In the same way, the response contains the data that is defined using the output message type of the inbound message interface. It passes in the opposite direction to the request. If errors are thrown on the inbound side, these can be reported to the sender by using fault messages and then handled there. A fault message, like a response, passes from the inbound interface to the outbound interface. You can define mappings for request messages, response messages, and fault messages.

The Integration Builder supports the following mapping programs for this purpose:

- **Message mappings** [Seite 148] that you define using a graphical editor.
- **XSLT and Java mappings** [Seite 176] that you can import into the Integration Repository as an archive.

Essentially it is possible to create or import multiple mapping programs in the Integration Repository for the same output or input message type pair (or the corresponding part of an imported or external interface).

**Interface Mappings**

When defining mapping programs for request, response, or fault messages, the definition is first separated from the interfaces that reference the corresponding message types. Furthermore, you can reuse a message type for multiple interfaces. This means that the simple definition of a mapping program is not sufficient to establish a connection (that is based on the assignment of outbound and inbound interfaces).

This role is undertaken by the **interface mapping** [Seite 147]:

- An interface mapping specifies the corresponding mapping programs for request, response, or fault messages for a selected interface pair. You use an interface mapping to register mappings for an interface pair.
- You can also specify multiple mapping programs to be executed one after the other in the case of requests and responses for an interface mapping.
You can also define multiple interface mappings for the same interface pair, to provide multiple variants in the repository. At configuration time, the customer can select the appropriate mapping using a mapping relation and save it in the Integration Directory.

**Structure and Value Mapping**

The following two types of mappings are differentiated between, regardless of the technology that you use for mappings:

- *Structure mappings* where the structure of an XML document is mapped to the structure of another XML document.
- *Value mappings* where the values of an XML document are mapped to the values of another XML document.

**Example**

In the example below, data about a flight booking is exchanged using XML documents. The XML schema of the source document provides more hierarchy nodes than the XML schema of the target document. Furthermore, the target document schema does not use all the information that is provided by the source document schema. Both structures can be mapped to each other by using a structure mapping.

The passenger class is an example of a value mapping. This information is coded using a string in the source document and a decimal value in the target document. Another example of a value mapping would be the date format.

Using the Integration Builder graphical mapping editor, you can define structure mappings easily using Drag & Drop and then assign fields from source and target structures to each other by using a function (required for a value mapping).
Interface Mappings

Use

Interface mappings register your mapping program for an interface pair in the Integration Repository.

Prerequisites

The interfaces that you want to connect using a mapping must already be saved in the Integration Repository.

If the interface cannot be imported or cannot be created in the Integration Repository (in the case of an external adapter, for example), you must enter the interface names manually. However, it is not possible to check the technical name.

Furthermore, all the mappings that you want to use must already be saved in the Integration Repository.

Activities

1. Create an interface mapping on the design maintenance screen of the Integration Builder (also see: Creating a New Object [Seite 40]).

   You can also create multiple interface mappings for the same interface pair.

2. Enter the outbound and inbound interfaces that require a mapping of the request message, the response message, the fault message, or all three.

3. To import the properties of the interfaces, choose Read Interfaces. The Mapping Program table displays the lines for the request message, response message, and if available, for the fault message, for each mode of the interfaces (either synchronous or asynchronous).

4. In the Mapping Program column, select a mapping category (message mapping, XSL or Java) using the search help (Help). In the subsequent window, select the mapping program for the respective message type (request, response, or fault).

   You can select mapping programs for an interface mapping that have the same namespace and the same software component version as the interface mapping. This ensures that the mapping program can be shipped together with the interface mapping (see: Software Logistics [Seite 55]).

5. It is possible to execute multiple mapping programs in succession for requests and responses.
   - To insert an additional line for a mapping program, choose .
   - To delete the registration for a mapping program, choose .

   At runtime, the mapping programs are executed from top to bottom.

6. Save the interface mapping.
Message Mappings

Use

A message mapping refers to a mapping of messages and is supported by a graphical
mapping editor. The editor enables you to design a structure mapping between any two XML
structures and the connection to a value mapping.

The mapping editor generates Java source code from the graphical mapping description that
is then compiled and packed in a Jar file that the Integration Engine executes at runtime.

Features

The mapping editor can import any schema ((fault) message types, imported RFCs, or IDocs)
from the Integration Repository, as well as XML schema definitions (XSD) and any XML file.
In the case of XSD files (the repository schemas also use this format), the mapping editor
generates an XML representation for both the source structure and the target structure.

⚠️

The following element or attribute construct is not supported for uploaded XSD
files: import, include, redefine, notation, nillable, list, union,
anyAttribute, mixed.

The following constructs are not supported in the restriction of simple types:
precision, maxExclusive, minExclusive, maxInclusive,
minInclusive, pattern, totalDigits, fractionDigits, enumeration,
minLength, maxLength, length, whiteSpace und simpleType.

Main Functions Area of the Mapping Editor

Once you have created a message mapping the system navigates to the mapping editor
Design tab page. In the following screenshot, two message types have been loaded and a
simple mapping has been defined (the tab page was detached from the Integration Builder
beforehand):
You have the following options when defining a message mapping:

- The functions in the **Main Function Toolbar** relate to the entire mapping. For example, you can save your mapping or automatically assign fields with the same name to each other.

- When you create a new message mapping the **Structure Overview** in the mapping editor is initially empty. Here you must specify a repository schema for both the source structure and the target structure or import an XML or XSD file. The mapping editor imports the data and displays the XML representation for the mapping in the **Structure Overview**. In this screen area, the source structure is on the left and the target structure is on the right. Source structure fields then have to be mapped to target structure fields. You have various options for displaying the source and target structure (**Tree View**, **Tree and Table View**, display as a text document, or display in the browser) which you can select from.

- The structure overview is strongly linked to the **Data-Flow Editor**, which you use to describe the mapping of one or more source fields to one target field. This type of mapping is also referred to as a **Target Field Mapping**. If a field in the target structure has already been mapped, you can navigate to the respective mapping by double clicking the corresponding target field in the structure overview.

Therefore, a message mapping comprises various target field mappings. Once you have defined a mapping for each mandatory target field your message mapping is complete. To test your message mapping, navigate to the **Test** tab page (see: **Testing Message Mappings [Seite 175]**).

### Functions of the Object Toolbar

<table>
<thead>
<tr>
<th>Function</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="icon" alt="Compile" /></td>
<td>Provided that the message mapping is complete, you can use this function to compile the Java source files generated for the mapping.</td>
</tr>
</tbody>
</table>
If you called a repository schema in the mapping editor that has since been changed in the Integration Repository, the changes will not be visible in the mapping editor. To apply changes of this type to the source and target structures, choose Update.

Deletes all target field mappings.

Displays all target field mappings including the functions used as text (the data-flow editor is hidden from view). Lines in red indicate missing target field mappings. The Quickinfo for a green target field displays the corresponding mapping.

Before you execute this function, first select a node in both the source and target structure that belong to each other. You can then automatically assign subnodes that have identical names (names are case-sensitive here). This function is particularly useful for simple mappings, for example from RFC XML to proxy XML.

This function is only available when an IDoc is loaded as a target structure in the structure overview. It enables you to define whether the IDoc header should be obligatory or optional. This may be useful if you want to assign the IDoc header after the mapping value. Depending on your selection, the IDoc header is either shown in red in the structure overview, or not.

The name of the loaded schema is displayed in the header area of the structure overview. You can call a context menu with the following functions for this name:

- Open:
  Displays the message type or the imported schema for an RFC or IDoc on a new tab page.

- Copy:
  Copies the name to the clipboard for input fields of other objects.

- Expand:
  In the default setting, the header area of the structure overview only shows the name of the loaded schema due to space restrictions. You can also display the namespace and the software component version of the object by choosing the Expand function.

To define target mappings for different positions of a target element that occurs frequently, the context menu for the target structure has the function Duplicate Subtree (see: Duplicating Subtrees [Seite 171]). You have the option of deleting any duplicates that you create.

To define a mapping between two XML structures, proceed as follows:

1. Create a message mapping on the design maintenance screen of the Integration Builder (also see: Creating a New Object [Seite 40]).

2. Load a source and target structure in the mapping editor by either using the search help, or by using Drag & Drop to drag a name from the navigation tree and drop it on the corresponding hand.

3. Using the structure overview and the data-flow editor, assign one or more source fields to a target field. This type of mapping is also called a Target Field Mapping. It is also possible to assign one source field to multiple target fields (see: Assigning a Field [Seite 151]).

4. If necessary, enhance the target field mapping in the data-flow editor. To do so, connect the source fields to the target fields by using the standard functions [Seite 154].
of the mapping editor. You can also create user-defined functions [Seite 170] in Java in the mapping editor and then also use them in the data-flow editor (see also: Data-Flow Editor [Seite 152]).

5. Carry out the last two steps for at least all mandatory target fields.

**Result**

The mapping is complete and ready for testing [Seite 175] when all the fields in the target structure have either green or white icons. A yellow icon indicates that a target field mapping is incomplete, while a red icon indicates that a target field is unassigned.

---

### Assigning a Field

**Use**

Using the structure overview and the data-flow editor you can map source fields of the source structure to a target field.

**Features**

You can do the following:

- Use Drag & Drop to select a source field in the structure overview and map it to a target field, or the other way around. The mapping editor displays this simple assignment in the data-flow editor.

- Drag a target field and one or more source fields from the structure overview to the data-flow editor. The data-flow editor automatically assigns the first source field/target field pair. If a target field has already been mapped, then the mapping is displayed in the data-flow editor.

- Select a source and target field from the structure view by double clicking. Note that you always transfer source fields to the data-flow editor by double clicking. However, this is only possible for target fields if they have not already been assigned a source field. When you double click a target field that has already been assigned, the system navigates to the corresponding mapping.

In the data-flow editor you can then connect source fields with standard functions or with your own functions (see: Data-Flow Editor Functions [Seite 152]). In the simplest instance, assign a source field to a target field. The value of the source field is then transferred to the target field at runtime.

A source field can also be mapped to multiple target fields. Simply define multiple target field mappings in which the source field is assigned to the respective target field.

**Status Display**

**Icon Types**

<table>
<thead>
<tr>
<th>Icon</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>📊</td>
<td>Attribute</td>
</tr>
</tbody>
</table>
Icon Colors

<table>
<thead>
<tr>
<th>Color</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>White</td>
<td>Attribute or element not assigned</td>
</tr>
<tr>
<td>Red</td>
<td>Attribute or element must be assigned to complete the mapping</td>
</tr>
<tr>
<td>Yellow</td>
<td>Attribute or element has already been assigned but the corresponding mapping in the data-flow editor is not complete</td>
</tr>
<tr>
<td>Green</td>
<td>Mapping to target field complete</td>
</tr>
</tbody>
</table>

Furthermore, collapsed target field names in the mapping editor are displayed in red if target field mappings for subnodes of the marked field name are still missing.

The colors used for the data-flow objects in the data-flow editor have the same meanings as those used above.

---

**Data-Flow Editor**

**Use**

You use the data-flow editor (at the bottom of the mapping editor) to describe the flow of data from one or more source fields to one target field. Since this mapping is only a part of the actual message mapping and refers to a particular target field, this type of mapping is known as a **Target Field Mapping**.

**Features**

The data-flow editor comprises the following:

- An editor window in which you can insert functions and outbound and target fields as rectangles, and move them as required. These rectangles are known as **Data-Flow Objects** because you use them to describe the data-flow between source fields and target fields.

- A function chooser in the lower screen area of the data-flow editor that contains the functions of various function categories. Using this toolbar, you can drag functions as new data-flow objects and drop them in the editor window.

On the left-hand side of the editor window there are general functions that are useful when editing a target field mapping.

**General Functions of the Data-Flow Editor**

<table>
<thead>
<tr>
<th>Function</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delete Floating Objects</td>
<td>Deletes all data-flow objects that are not connected to a source and target field. Floating data-flow objects are displayed in white.</td>
</tr>
</tbody>
</table>
### Lay Out Mapping

Produces an ordered representation of the target field mapping

### Delete Current Mapping

Deletes the target field mapping currently displayed

## Activities

6. Once you have selected the source fields and the target field you want (see: Assigning a Field [Seite 151]), select a function category from the function chooser.

7. If you have already created a user-defined function [Seite 170], it is located in the User-Defined function category.
   - All other categories contain the most important standard functions [Seite 154] for message mappings.
   - Select a function from the function chooser either by double clicking, or by using Drag & Drop to drag it to the editor area.

8. Connect the selected function to the other data-flow objects for the source fields and the target field by connecting the small rectangles for the input and output values using Drag & Drop. Arrows represent connected elements.

9. To undo a connection, remove the arrow from the corresponding rectangle and drop it anywhere outside the data-flow object.

10. To delete any data-flow objects that you no longer require, choose Delete from the context menu. To delete all floating data-flow objects (displayed in white), choose Delete Floating Objects.

11. Logically speaking, once all data-flow objects are green the mapping for the target field is complete.

   ![Warning Icon]

   All functions (including user-defined functions) expect string arguments and return a string value. The strings true or false are expected for conditions. If a condition transfers a value other than true or false, it is interpreted as false.

### Example

In the example above, a mapping to the target field name was selected that is connected to the source fields firstName and lastName using the standard function concat. Input values of any data-flow objects are displayed in small rectangles on the left of the object, while output values are displayed on the right of the object. You use these small rectangles to connect data-flow objects to each other so as to define a data flow between multiple source fields and exactly one target field. Technically speaking, a target field mapping is complete once a source field has been assigned to a target field and all arguments for each connecting function have values. Since this is the case in the example illustrated here, the data-flow editor displays the connected objects in green.
Standard Functions

General

Values of elements or attributes of the XML message (the payload) are technically speaking a string. Therefore, all standard functions (including user-defined functions) expect string arguments and return a string value. Nevertheless, the transferred value can of course have a different data type semantically. This is the data type you specified when defining the schema for the payload for the field. The following is the standard procedure for standard functions:

- Depending on the standard function, the transferred values are converted to values applicable for the function using a data type conversion (a cast). If the value cannot be interpreted, the mapping runtime triggers a Java exception.
- If-clauses evaluate conditions that get Boolean values. Standard functions that return Boolean values return either a true or false string. Standard functions that expect Boolean values interpret the values "1" and "true" (not case-sensitive) as true and all other values as false.

Also see: Runtime Procedure [Seite 173].

Overview of Functions

Data-flow objects for standard functions are structured as follows:

- Most data-flow objects have two or three inbound channels on the left-hand side, and one outbound channel on the right-hand side.
- If-functions are shaped as rhombuses or triangles.
- There are also conversion functions with one inbound and outbound channel and functions that get a value without making any entries.
- Many of the standard functions require you to make additional specifications that you can enter in a dialog for the Function Properties. Call this dialog by double clicking the corresponding data-flow object.

The following graphic provides an overview of all the different forms of data-flow objects:
To illustrate how the individual functions are used, the input boxes in the graphic above have the variables $X$, $Y$, $Z$ or $I$, and the result variable is either $R$ or $O$. All standard functions in subsequent tables are described using these input and output variables. In the data-flow editor, other data-flow objects that are connected with the functions using the respective connecting boxes either get the input values or receive the result of the operation. You can combine the functions in any way you require (provided that the values transferred to a function can be interpreted).

**Function Category: Arithmetic**

You can only enter numerical values for this category (this includes values with digits after the decimal point). If the value cannot be interpreted as a digit, then the system triggers a Java exception. Otherwise, all calculations are executed with the precision of the Java data type `float`. The format of the result value depends on the result:

- If the result is a value with digits after the decimal point, these remain unchanged.
- Exception: If a zero follows the decimal point, then this is cut off. This means that the result of the calculation $4.2 - 0.2$ is $4$ and not $4.0$.

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>$R = X + Y$</td>
</tr>
<tr>
<td>sqrt</td>
<td>$R$ is the square root of $X$</td>
</tr>
<tr>
<td>sqr</td>
<td>$R$ is the square of $X$</td>
</tr>
</tbody>
</table>
| sign          | $R = 1$, when $X$ is a positive number  
$R = 0$, when $X$ equals $0$  
$R = 1$, when $X$ is a negative number |
<p>| neg           | $R = -X$ |
| $1/x$         | $R$ is the reciprocal of $X$ |
| power         | $R = X^T$ |
| less          | true, when $X &lt; Y$, otherwise false |
| greater       | true, when $X &gt; Y$, otherwise false |
| multiply      | $R = X * Y$ |
| divide        | $R = X / Y$ |</p>
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>max</td>
<td>( R = \text{Maximum of values } X \text{ and } Y )</td>
</tr>
<tr>
<td>min</td>
<td>( R = \text{Minimum of values } X \text{ and } Y )</td>
</tr>
<tr>
<td>subtract</td>
<td>( R = X - Y )</td>
</tr>
<tr>
<td>FormatNum</td>
<td>Converts ( I ) according to a pattern that you define using the function properties. The possible patterns are the same as in the Java class java.text.DecimalFormat.</td>
</tr>
</tbody>
</table>

**Function Category: Boolean**

All functions in this category expect Boolean input values (see above).

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>And</td>
<td>( R \text{ is true if } X \text{ and } Y \text{ have the value true. Otherwise, } R \text{ is false.} )</td>
</tr>
<tr>
<td>Or</td>
<td>( R \text{ is true if } X \text{ or } Y \text{ have the value true. Otherwise, } R \text{ is false.} )</td>
</tr>
<tr>
<td>Not</td>
<td>• ( O \text{ is false if } I \text{ has the value true} )&lt;br&gt;• ( O \text{ is true if } I \text{ has the value false} )</td>
</tr>
<tr>
<td>Equals</td>
<td>Compares both boolean values ( X ) and ( Y ) and returns true if both are equal, otherwise false. Non-Boolean values are interpreted as false (see above). Use the functions equalsS or compare from the Text category to compare strings.</td>
</tr>
<tr>
<td>notEquals</td>
<td>( R = \text{Not}(\text{Equals}(X, Y)) )</td>
</tr>
</tbody>
</table>
| if            | • If condition \( X \) is fulfilled (gets true): \( R = Y \)  
• If condition \( X \) is not fulfilled (gets false): \( R = Z \) |
| ifWithoutElse | If condition \( X \) is fulfilled (gets true): \( R = Y \). Otherwise, the target field is not created. |

**Function Category: Constants**

Since these functions do not have any input values, they are generating functions [Seite 173].

<table>
<thead>
<tr>
<th>Function</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant</td>
<td>( O ) gets any string constant that you can enter in the dialog for the function properties.</td>
</tr>
<tr>
<td>CopyValue</td>
<td>For a frequently occurring element, copies the value at a defined position in the source structure to the assigned target field.</td>
</tr>
<tr>
<td>sender</td>
<td>( O ) gets the name of the sender system. Test_Sender_System is outputted when you test the message mapping in the Integration Builder.</td>
</tr>
<tr>
<td>receiver</td>
<td>( O ) gets the name of the receiver system. Test_Receiver_System is outputted when you test the message mapping in the Integration Builder.</td>
</tr>
</tbody>
</table>

**Function Category: Conversions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>FixValues</td>
<td>Executes a value mapping using a fixed value table that you complete using the function properties. The table is saved together with the most current message mapping and can only be used once.</td>
</tr>
</tbody>
</table>
Value mapping

Executes a value mapping for the value I entered. In the function attributes, assign the both input value I and source value O to a representation in the value-mapping table. In this way, you can assign source and target structure fields to a representation. You complete the value-mapping table on the configuration maintenance screen in the Integration Builder. Since these entries are not available at design time, the mapping result is just an estimate when you test the message mapping. Also see: Value Mapping [Seite 53].

Function Category: Date

The following functions enable you to define the date format for the source or target format using the dialog for the function properties.

The format adheres to the convention as defined in the standard Java class java.util.SimpleDateFormat. The function properties dialog also provides you with a wizard that you can use to enter the most frequently used date formats.

<table>
<thead>
<tr>
<th>Function</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>currentDate</td>
<td>Returns the current data using O.</td>
</tr>
<tr>
<td>DateTrans</td>
<td>Converts date format I to another date format O.</td>
</tr>
<tr>
<td>DateBefore</td>
<td>R = true, when date X comes before date Y, otherwise false</td>
</tr>
<tr>
<td>DateAfter</td>
<td>R = true, when date X comes after date Y, otherwise false</td>
</tr>
<tr>
<td>CompareDates</td>
<td>R = 1, when date X is after date Y R = 0, when date X is the same as date Y R = -1, when date X is before date Y</td>
</tr>
</tbody>
</table>

Function Category: Node Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>createIf [Seite 160]</td>
<td>If there are structuring elements in your target structure that do not exist in the source structure, insert them using this function. Using condition I you can control whether the element is inserted or not. You connect the element in the target structure using O.</td>
</tr>
<tr>
<td>removeContexts [Seite 162]</td>
<td>Removes the top context [Seite 175] of a source field. In this way, you can delete a hierarchy level and generate a list.</td>
</tr>
<tr>
<td>replaceValue</td>
<td>Replaces the value I with a value that you can define in the dialog for the function properties.</td>
</tr>
<tr>
<td>exists [Seite 164]</td>
<td>O = true, if the source field assigned to inbound channel I exists in the XML instance. Otherwise, false.</td>
</tr>
<tr>
<td>SplitByValue [Seite 165]</td>
<td>Inserts a context change [Seite 175] for an element.</td>
</tr>
<tr>
<td>collapseContexts [Seite 167]</td>
<td>Replaces all values within all contexts by an empty string. This is a useful function when used in combination with SplitByValue.</td>
</tr>
</tbody>
</table>

Function Category: Statistic

The functions of this function category are designed for source fields that occur more than once in the source structure (maxOccurs = i > 1)

<table>
<thead>
<tr>
<th>Function</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function</td>
<td>Use</td>
</tr>
<tr>
<td>----------</td>
<td>-----</td>
</tr>
<tr>
<td><strong>sum</strong></td>
<td>( R = \text{Sum of values } X_1 \text{ to } X_i )</td>
</tr>
<tr>
<td>average</td>
<td>( R = \text{Average of values } X_1 \text{ to } X_i )</td>
</tr>
<tr>
<td>count</td>
<td>( R = i )</td>
</tr>
</tbody>
</table>

**Function Category: Text**

In position specifications, the 0 position corresponds to the first character in the string.

<table>
<thead>
<tr>
<th>Function</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>substring</strong></td>
<td>Returns a sub string ( O ) for a string ( I ). Use the dialog for the function properties to specify the position of the substring. Example: substring(“Hello”, 0,1) = “H”, means that the substring from Startindex 0 to Endindex 1 <strong>(not including position 1)</strong> is shown.</td>
</tr>
<tr>
<td><strong>concat</strong></td>
<td>( R = \text{linking of strings } X \text{ and } Y ) (without blanks). Example: ( X = &quot;Mrs.&quot;; Y = &quot;Miller&quot;; R = &quot;Mrs.Miller&quot;. Use the dialog for the function properties to insert a separator in the string.</td>
</tr>
<tr>
<td><strong>equalsS</strong></td>
<td>( R = \text{true, if string } X \text{ equals string } Y ), otherwise ( R = \text{false} ).</td>
</tr>
<tr>
<td><strong>indexOf</strong> (2 input parameters)</td>
<td>( R = \text{first position at which string } Y \text{ is found in } X \text{ and } -1 \text{ if } Y \text{ does not occur at all.}</td>
</tr>
<tr>
<td><strong>indexOf</strong> (3 input parameters)</td>
<td>( R = \text{first position from position } Z \text{ at which string } Y \text{ is found in } X \text{ and } -1 \text{ if } Y \text{ does not occur at all.}</td>
</tr>
<tr>
<td><strong>lastIndexOf</strong> (2 input parameters)</td>
<td>( R = \text{last position at which string } Y \text{ is found in } X \text{ and } -1 \text{ if } Y \text{ does not occur at all.}</td>
</tr>
<tr>
<td><strong>lastIndexOf</strong> (3 input parameters)</td>
<td>( R = \text{Last position from position } Z \text{ at which string } Y \text{ is found in } X \text{ and } -1 \text{ if } Y \text{ does not occur at all.}</td>
</tr>
<tr>
<td><strong>compare</strong></td>
<td>Compares string ( X ) with string ( Y ): ( R = 0 ), when the strings are equal ( R = \text{positive number } i ), when ( X ) is lexicographically larger than ( Y ) ( R = \text{negative number } i ), when ( X ) is lexicographically smaller than ( Y ) ( i ) specifies the difference between the two strings lexicographically. The function acts in the same ways as the method <strong>compareTo()</strong> of the JDK class <strong>java.lang.String</strong>.</td>
</tr>
</tbody>
</table>
**replaceString**

- **X**: String in which something is to be replaced
- **Y**: Sequence to be replaced in X
- **Z**: Sequence to replace Y
- **R**: Sequence in which each occurrence of **Y** in **X** is replaced by **Z**.

<table>
<thead>
<tr>
<th>X</th>
<th>&quot;sparring with a purple porpoise&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>&quot;p&quot;</td>
</tr>
<tr>
<td>Z</td>
<td>&quot;t&quot;</td>
</tr>
<tr>
<td>R</td>
<td>&quot;starring with a turtle tortoise&quot;</td>
</tr>
</tbody>
</table>

- **length**: 
  - O = Length of string

- **endsWith**
  - R = true when **Y** is the last string in **X**; otherwise false.

- **startsWith** 
  - (2 input parameters)
  - R = true when **Y** is the first string in **X**; otherwise false.

- **startsWith** 
  - (3 input parameters)
  - R = true when **Y** matches **X** from position **Z**; otherwise false.

- **toUpperCase**
  - Converts all lower case characters to upper case from **I**

- **trim**
  - Removes all white space characters at the start and end of a string.
  - Acts the same as the *method trim()* of the JDK class *java.lang.String*.

- **toLowerCase**
  - Converts all upper case letters from **I** to lower case letters

---

**copyValue**

**Use**

You can use CopyValue() to copy the value of a position in the source structure and assign it to a target field for a frequently occurring source structure element. The value is copied each time the target field occurs in the target structure. However, it is copied a maximum of maxOccurs times (also see: Frequently Occurring Elements [Seite 173]).

**Example**

In the example below, the address data is represented in the source structure (street, city, zip code) by the field `addrDat`, which occurs three times. However, in the target structure, this information is separated into three individual fields (street, city, zipCode):

**Source Structure (left) and Target Structure (right)**
Using the CopyValue() function you can copy the values of field addrDat to the individual target fields. To do so, specify the position that you want to copy. Note that the 0 position refers to the first occurrence of the source field:

Message Mapping Using CopyValue()

<table>
<thead>
<tr>
<th>Target Field Mapping</th>
<th>Position for CopyValue()</th>
</tr>
</thead>
<tbody>
<tr>
<td>/CustomersMsg=/PartnersMsg</td>
<td></td>
</tr>
<tr>
<td>/CustomersMsg=/PartnersMsg</td>
<td></td>
</tr>
<tr>
<td>/CustomersMsg/customer/name=</td>
<td>/PartnersMsg/partner/name</td>
</tr>
<tr>
<td>/CustomersMsg/customer/street=</td>
<td>CopyValue(/PartnersMsg/partner/addrDat)</td>
</tr>
<tr>
<td>/CustomersMsg/customer/city=</td>
<td>CopyValue(/PartnersMsg/partner/addrDat)</td>
</tr>
<tr>
<td>/CustomersMsg/customer/zipCode=</td>
<td>CopyValue(/PartnersMsg/partner/addrDat)</td>
</tr>
</tbody>
</table>

createIf

Use

You use createIf() to create a tag in the target structure independently of a condition.

Example

In the example below, a mapping is required between two different formats for a list of messages. The attribute type is used in the source structure to classify the message as either internal (type="internal") or external (type="external"). However, in the target structure the tag internalRequest or externalRequest is used:

Source Structure (left) and Target Structure (right)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>minOccurs</th>
<th>maxOccurs</th>
<th>Field Name</th>
<th>minOccurs</th>
<th>maxOccurs</th>
</tr>
</thead>
<tbody>
<tr>
<td>RequestListMsg</td>
<td>1</td>
<td>1</td>
<td>RequestListEIMsg</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>request</td>
<td>0</td>
<td>unbounded</td>
<td>internalRequest</td>
<td>0</td>
<td>unbounded</td>
</tr>
<tr>
<td>type</td>
<td>1</td>
<td>1</td>
<td>initiator</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
You use `createIf()` to create the respective target field independently of attribute type:

**Message Mapping Using createIf()**

<table>
<thead>
<tr>
<th>Target Field Mapping</th>
<th>Value of Constant()</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>/RequestListEIMsg=/RequestListMsg</code></td>
<td></td>
</tr>
</tbody>
</table>
| `createIf(  
  equalsS(  
    removeContexts(`/RequestListMsg/request/type),  
    Constant()))` | `internal` |
| `/RequestListEIMsg/internalRequest/initiator=` | `/RequestListMsg/request/creator` |
| `/RequestListEIMsg/internalRequest/contact=` | `/RequestListMsg/request/contact` |
| `/RequestListEIMsg/internalRequest/issue=` | `/RequestListMsg/request/description` |
| `createIf(  
  equalsS(  
    removeContexts(`/RequestListMsg/request/type),  
    Constant()))` | `external` |
| `/RequestListEIMsg/externalRequest/initiator=` | `/RequestListMsg/request/creator` |
| `/RequestListEIMsg/externalRequest/contact=` | `/RequestListMsg/request/contact` |
| `/RequestListEIMsg/externalRequest/issue=` | `/RequestListMsg/request/description` |

The function `removeContexts()` is required because the attribute type is mapped in place of the source field `request`. Therefore, there is no target field for `request` and no context in the target structure. However, the type queue has nevertheless saved the request context so that processing would terminate early if you did not delete this context.

**Example**

<table>
<thead>
<tr>
<th>Source Instance</th>
<th>Result</th>
</tr>
</thead>
</table>
**removeContexts**

**Use**

You use `removeContext()` to delete the top context [Seite 175] for an element. The top hierarchy level is then deleted.

**Example**

In the following example, a list of debtors that lists one or more payments made in the field `payment` is to be transformed into a list that only contains payments made. The payments and the debtors also have the attribute `id` in each case to help identify them.

**Source Structure (Left) and Target Structure (Right)**

<table>
<thead>
<tr>
<th>Field Name</th>
<th>minOccurs</th>
<th>maxOccurs</th>
<th>Field Name</th>
<th>minOccurs</th>
<th>maxOccurs</th>
</tr>
</thead>
<tbody>
<tr>
<td>DebitorListMsg</td>
<td>1</td>
<td>1</td>
<td>AllDebtsMsg</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Debtor</td>
<td>0</td>
<td>unbounded</td>
<td>debt</td>
<td>0</td>
<td>unbounded</td>
</tr>
</tbody>
</table>
To delete the top hierarchy level Debitor, connect the element payment to the function removeContexts(). Specify as the target field the field in which you want the contents of payment to be copied to, in this case debt.

Message Mapping Using removeContexts()

<table>
<thead>
<tr>
<th>Target Field Mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>/AllDebtsMsg=/DebitorListMsg</td>
</tr>
<tr>
<td>/AllDebtsMsg/debt=removeContexts(/DebitorListMsg/Debitor/payment)</td>
</tr>
<tr>
<td>/AllDebtsMsg/debt/id=/DebitorListMsg/Debitor/payment/id</td>
</tr>
</tbody>
</table>

Example

<table>
<thead>
<tr>
<th>Source Instance</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;?xml version=&quot;1.0&quot; encoding=&quot;UTF-8&quot;?&gt;</td>
<td>&lt;?xml version=&quot;1.0&quot; encoding=&quot;UTF-8&quot;?&gt;</td>
</tr>
<tr>
<td>&lt;DebitorListMsg&gt;</td>
<td>&lt;ns0:AllDebtsMsg xmlns:ns0=&quot;workshopDemo&quot;&gt;</td>
</tr>
<tr>
<td>&lt;Debitor id=&quot;1&quot;&gt;</td>
<td>&lt;debt id=&quot;1&quot;&gt;</td>
</tr>
<tr>
<td>50$</td>
<td>50$</td>
</tr>
<tr>
<td>&lt;/payment&gt;</td>
<td>&lt;/debt&gt;</td>
</tr>
<tr>
<td>&lt;Debitor id=&quot;2&quot;&gt;</td>
<td>&lt;debt id=&quot;2&quot;&gt;</td>
</tr>
<tr>
<td>150$</td>
<td>150$</td>
</tr>
<tr>
<td>&lt;/payment&gt;</td>
<td>&lt;/debt&gt;</td>
</tr>
<tr>
<td>&lt;Debitor id=&quot;3&quot;&gt;</td>
<td>&lt;debt id=&quot;3&quot;&gt;</td>
</tr>
<tr>
<td>90$</td>
<td>90$</td>
</tr>
<tr>
<td>&lt;/payment&gt;</td>
<td>&lt;/debt&gt;</td>
</tr>
<tr>
<td>&lt;Debitor id=&quot;4&quot;&gt;</td>
<td>&lt;debt id=&quot;4&quot;&gt;</td>
</tr>
<tr>
<td>1150$</td>
<td>1150$</td>
</tr>
<tr>
<td>&lt;/payment&gt;</td>
<td>&lt;/debt&gt;</td>
</tr>
<tr>
<td>&lt;Debitor id=&quot;5&quot;&gt;</td>
<td>&lt;debt id=&quot;5&quot;&gt;</td>
</tr>
<tr>
<td>2230$</td>
<td>2230$</td>
</tr>
<tr>
<td>&lt;/payment&gt;</td>
<td>&lt;/debt&gt;</td>
</tr>
<tr>
<td>&lt;Debitor id=&quot;6&quot;&gt;</td>
<td>&lt;debt id=&quot;6&quot;&gt;</td>
</tr>
<tr>
<td>3390$</td>
<td>3390$</td>
</tr>
<tr>
<td>&lt;/payment&gt;</td>
<td>&lt;/debt&gt;</td>
</tr>
<tr>
<td>&lt;Debitor id=&quot;7&quot;&gt;</td>
<td>&lt;debt id=&quot;7&quot;&gt;</td>
</tr>
<tr>
<td>&lt;/DebitorListMsg&gt;</td>
<td>&lt;/ns0:AllDebtsMsg&gt;</td>
</tr>
</tbody>
</table>
exists

Use

Use this function to determine whether a particular source field exists in the XML instance to be processed. If it does, exists() returns the value true, otherwise it returns the value false.

Example

In the example below, a function description is to be mapped to a structure that merely comprises the technical name and the description of the function. The source structure field description is optional, however, the target structure field descriptionOfUse is mandatory (minOccurs = 1).

Source Structure (Left) and Target Structure (Right)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>minOccurs</th>
<th>maxOccurs</th>
<th>Field Name</th>
<th>minOccurs</th>
<th>maxOccurs</th>
</tr>
</thead>
<tbody>
<tr>
<td>FunctionDeclMsg</td>
<td>1</td>
<td>1</td>
<td>FunctionHeadingMsg</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>name</td>
<td>1</td>
<td>1</td>
<td>technicalName</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>description</td>
<td>0</td>
<td>1</td>
<td>descriptionOfUse</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>inputArg</td>
<td>0</td>
<td>64</td>
<td>name (Attr)</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>type (Attr)</td>
<td>1</td>
<td>1</td>
<td>type (Attr)</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>returnValue</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Using the exists() function and other Boolean functions you can output a specified string by default if the description-tag does not exist here. This ensures that the target field descriptionOfUse is always completed and is never empty.

Message Mapping Using exists()

<table>
<thead>
<tr>
<th>Target Field Mapping</th>
<th>Value of Constant()</th>
</tr>
</thead>
<tbody>
<tr>
<td>/FunctionHeadingMsg=/FunctionDeclMsg</td>
<td></td>
</tr>
<tr>
<td>/FunctionHeadingMsg/technicalName=/FunctionDeclMsg/name</td>
<td></td>
</tr>
<tr>
<td>/FunctionHeadingMsg/descriptionOfUse= if( Constant() , Not(exists(/FunctionDeclMsg/description)) , /FunctionDeclMsg/description )</td>
<td>(No description available)</td>
</tr>
</tbody>
</table>

The text description of the if standard function must be read as follows: if ( <result if condition true>, <condition>, <result if condition false>).
Example

<table>
<thead>
<tr>
<th>Source Instance</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;?xml version=&quot;1.0&quot; encoding=&quot;UTF-8&quot;?&gt;</code></td>
<td><code>&lt;?xml version=&quot;1.0&quot; encoding=&quot;UTF-8&quot;?&gt;</code></td>
</tr>
<tr>
<td><code>&lt;FunctionDeclMsg&gt;</code></td>
<td><code>&lt;ns0:FunctionHeadingMsg xmlns:ns0=&quot;workshopDemo&quot;&gt;</code></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The context for personalId of the source structure is managers. There is only one context of this type. All personalId-elements are located in this context. To receive an additional context after each personalId tag, connect both personalId elements of the source and target structures, and in the function properties set the context change for each subelement of managers (Each Value). The manager context must also exist in the target structure. Therefore, assign the target field room to the source field managers.

### Message Mapping Using SplitByValue()

<table>
<thead>
<tr>
<th>Target Field Mapping</th>
<th>Value of SplitByValue()</th>
</tr>
</thead>
<tbody>
<tr>
<td>/RoomsMsg=/ManagerMsg</td>
<td></td>
</tr>
<tr>
<td>/RoomsMsg/room=/ManagerMsg/managers</td>
<td></td>
</tr>
<tr>
<td>/RoomsMsg/room/personalId= SplitByValue(/ManagerMsg/managers/personalId)</td>
<td>Each Value</td>
</tr>
</tbody>
</table>

Since the frequency of room is set to exactly four, four contexts are generated. In other words, four times the element room with each context having a subelement personalId (the other managers do not get a room):

### Example

<table>
<thead>
<tr>
<th>Source Instance</th>
<th>Result</th>
</tr>
</thead>
</table>
collapseContexts

Use

Using collapseContexts() replaces all values of all contexts [Seite 175] with an empty string:
This can be useful if each time an upper-level node is to be created if a lower-level node exists, for example. The function is useful in combination with the function \textcolor{red}{\texttt{SplitByValue()}} [Seite 165].

**Example**

In the following example, a separate upper-level element \texttt{<destSub>} is to be created in the target structure for each occurrence of element \texttt{<srcItem>}:

**Source Structure (left) and Target Structure (right)**

<table>
<thead>
<tr>
<th>Field Name</th>
<th>minOccurs</th>
<th>maxOccurs</th>
<th>Field Name</th>
<th>minOccurs</th>
<th>maxOccurs</th>
</tr>
</thead>
<tbody>
<tr>
<td>srcRoot</td>
<td>1</td>
<td>1</td>
<td>destRoot</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>srcSub</td>
<td>0</td>
<td>unbounded</td>
<td>destSub</td>
<td>0</td>
<td>unbounded</td>
</tr>
<tr>
<td>srcItem</td>
<td>0</td>
<td>unbounded</td>
<td>destItem</td>
<td>0</td>
<td>unbounded</td>
</tr>
</tbody>
</table>

This is achieved by using a combination of the functions \textcolor{red}{\texttt{SplitByValue()}} and \textcolor{red}{\texttt{collapseContexts()}}. To create a queue for \texttt{<destItem>} in which a context change is performed after each value, use \textcolor{red}{\texttt{removeContexts()}} and \textcolor{red}{\texttt{SplitByValue()}}. To create a queue with exactly as many context changes, but with empty values (the aim is to create just as many upper-level tags without values), apply \textcolor{red}{\texttt{collapseContexts()}} to the same nesting of functions:

**Message Mapping with collapseContexts() and SplitByValue()**

<table>
<thead>
<tr>
<th>Target Field Mapping</th>
<th>Value of SplitByValue()</th>
</tr>
</thead>
<tbody>
<tr>
<td>/destRoot=./srcRoot</td>
<td></td>
</tr>
<tr>
<td>/destRoot/destSub=</td>
<td></td>
</tr>
<tr>
<td>collapseContexts(</td>
<td></td>
</tr>
<tr>
<td>SplitByValue(</td>
<td></td>
</tr>
<tr>
<td>removeContexts(/srcRoot/srcSub/SrcItem)))</td>
<td>Each Value</td>
</tr>
<tr>
<td>/destRoot/destSub/=</td>
<td></td>
</tr>
<tr>
<td>SplitByValue(</td>
<td></td>
</tr>
<tr>
<td>removeContexts(/srcRoot/srcSub/SrcItem)))</td>
<td>Each Value</td>
</tr>
</tbody>
</table>

**Example**

<table>
<thead>
<tr>
<th>Source Instance</th>
<th>Result</th>
</tr>
</thead>
</table>
<?xml version="1.0" encoding="UTF-8"?>
<srcRoot>
  <srcSub>
    <srcItem>
      PID001
    </srcItem>
    <srcItem>
      PID002
    </srcItem>
  </srcSub>
  <srcSub>
    <srcItem>
      PID003
    </srcItem>
    <srcItem>
      PID004
    </srcItem>
    <srcItem>
      PID005
    </srcItem>
  </srcSub>
  <srcSub>
    <srcItem>
      PID006
    </srcItem>
    <srcItem>
      PID007
    </srcItem>
  </srcSub>
</srcRoot>

<?xml version="1.0" encoding="UTF-8"?>
<ns0:destRoot xmlns:ns0="workshopDemo">
  <destSub>
    <destItem>
      PID001
    </destItem>
  </destSub>
  <destSub>
    <destItem>
      PID002
    </destItem>
  </destSub>
  <destSub>
    <destItem>
      PID003
    </destItem>
  </destSub>
  <destSub>
    <destItem>
      PID004
    </destItem>
  </destSub>
  <destSub>
    <destItem>
      PID005
    </destItem>
  </destSub>
  <destSub>
    <destItem>
      PID006
    </destItem>
  </destSub>
  <destSub>
    <destItem>
      PID007
    </destItem>
  </destSub>
</ns0:destRoot>
User-Defined Functions

Use
If the standard functions for a target field mapping do not fulfil your requirements, or if the graphical representation becomes unclear due to the complexity of the mapping, you have the option of creating your own user-defined functions. Within this function you can compile Java source text and use Java programs from imported archives with the same namespace and software component version as the message mapping. You can then insert the function in the data-flow editor by using the function category User-Defined, just like a standard function [Seite 154].

A Java method is generated within the message mapping for user-defined functions.

Prerequisites
You have created a message mapping and have loaded an XML structure for both the source and the target structure.

Procedure
1. To create a new user-defined function, choose Create New Function in the data-flow editor at the bottom of the screen, on the left-hand side.
2. Specify the attributes of the new function in the subsequent window:
   - **Label**: Text that appears in the function chooser and on the data-flow object to identify the new function.
   - **Name**: Technical name of the function.
   - **Argument Count**: The number of input values that should be transferred to the function.
   - **Description**: Description of the function’s use.
3. In the subsequent window you can create Java source code:
   a. You can import Java packages to your methods from the input field Imports, by specifying them separated by a comma or semi-colon:
      - You do not need to import the packages java.lang.*, java.util.*, java.io.* and java.lang.reflect.* since all message mappings require these packages and therefore import them. You should be able to access standard JDK and J2EE packages of the SAP Web Application Server by simply specifying the package under Import. In other words, you do not have to import it as an archive into the Repository.
      - Besides the standard packages, you can also specify Java packages that you have imported as archives and that are located in the same namespace and in the same software component version as the message mapping.
   b. Create your Java source text in the editor window or copy source text from another editor.
All functions have the return value of type `java.lang.String` (also see the General section in standard functions [Seite 154]).

4. Choose OK to confirm.

5. User-defined functions are limited to the message mapping in which you created the function. To save the new function, simply save the message mapping.

6. To test the new function, test the message mapping [Seite 175].

**Result**

The new function is now visible in the *User-Defined* function category. When you select this category, a corresponding button is displayed in the function chooser. To edit, delete, or add the function to the data-flow editor, choose the arrow next to the button and select from the list box is displayed.

---

**Duplicating Subtrees**

**Use**

Even if elements are shown to occur more than once in the XML instance according to XML Schema definition, they are only displayed once in the structure overview. To assign source fields to multiple positions of an element in the target structure, you can either copy individual elements or the entire subtree using the context menu in the target structure.

**Example**

The records of a CD catalog are to be mapped to a simple HTML table. There are six subelements of `<cd>` in the source structure that are to be shown in one table line. The target field `<td>` is only shown once in the structure view, however.

**Source Structure (left) and Target Structure (right)**

<table>
<thead>
<tr>
<th>Field Name</th>
<th>minOccurs</th>
<th>maxOccurs</th>
<th>Field Name</th>
<th>minOccurs</th>
<th>maxOccurs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Catalog</td>
<td>1</td>
<td>1</td>
<td>CatalogTable</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>cd</td>
<td>0</td>
<td>unbounded</td>
<td>table</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>title</td>
<td>1</td>
<td>1</td>
<td>tr</td>
<td>0</td>
<td>unbounded</td>
</tr>
<tr>
<td>artist</td>
<td>1</td>
<td>1</td>
<td>td</td>
<td>0</td>
<td>unbounded</td>
</tr>
<tr>
<td>country</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>company</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>price</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>year</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Before you can define the target field mapping to the `<td>` elements, you must first duplicate the `<td>` element five times using the context menu. Each target field mapping then comprises simply a 1:1 assignment of the source and target fields:

**Message Mapping with Duplicated Elements**
**Target Field Mapping**

/CatalogTable=/Catalog

No mapping for/CatalogTable/table

/CatalogTable/table/tr=/Catalog/cd

/CatalogTable/table/tr/td[1]=/Catalog/cd/title

/CatalogTable/table/tr/td[2]=/Catalog/cd/artist

/CatalogTable/table/tr/td[3]=/Catalog/cd/country

/CatalogTable/table/tr/td[4]=/Catalog/cd/company

/CatalogTable/table/tr/td[5]=/Catalog/cd/price

/CatalogTable/table/tr/td[6]=/Catalog/cd/year

---

**Example**

<table>
<thead>
<tr>
<th>Source Instance</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;xml version=&quot;1.0&quot; encoding=&quot;UTF-8&quot;?&gt;</code></td>
<td><code>&lt;xml version=&quot;1.0&quot; encoding=&quot;UTF-8&quot;?&gt;</code></td>
</tr>
<tr>
<td><code>&lt;Catalog&gt;</code></td>
<td><code>&lt;ns0:CatalogTable xmlns:ns0=&quot;workshopDemo&quot;&gt;</code></td>
</tr>
<tr>
<td><code>&lt;cd&gt;</code></td>
<td><code>&lt;table&gt;</code></td>
</tr>
<tr>
<td><code>&lt;title&gt;</code> Empire Burlesque &lt;<code>/title&gt;</code></td>
<td><code>&lt;td&gt;</code> Empire Burlesque &lt;<code>/td&gt;</code></td>
</tr>
<tr>
<td><code>&lt;artist&gt;</code> Bob Dylan &lt;<code>/artist&gt;</code></td>
<td><code>&lt;td&gt;</code> Bob Dylan &lt;<code>/td&gt;</code></td>
</tr>
<tr>
<td><code>&lt;country&gt;</code> USA &lt;<code>/country&gt;</code></td>
<td><code>&lt;td&gt;</code> USA &lt;<code>/td&gt;</code></td>
</tr>
<tr>
<td><code>&lt;company&gt;</code> Columbia &lt;<code>/company&gt;</code></td>
<td><code>&lt;td&gt;</code> Columbia &lt;<code>/td&gt;</code></td>
</tr>
<tr>
<td><code>&lt;price&gt;</code> 10.90 &lt;<code>/price&gt;</code></td>
<td><code>&lt;td&gt;</code> 10.90 &lt;<code>/td&gt;</code></td>
</tr>
<tr>
<td><code>&lt;year&gt;</code> 1985 &lt;<code>/year&gt;</code></td>
<td><code>&lt;td&gt;</code> 1985 &lt;<code>/td&gt;</code></td>
</tr>
<tr>
<td><code>&lt;/cd&gt;</code></td>
<td><code>&lt;/tr&gt;</code></td>
</tr>
<tr>
<td><code>&lt;cd&gt;</code></td>
<td><code>&lt;tr&gt;</code></td>
</tr>
<tr>
<td><code>&lt;title&gt;</code> Hide your heart &lt;<code>/title&gt;</code></td>
<td><code>&lt;td&gt;</code> Hide your heart &lt;<code>/td&gt;</code></td>
</tr>
<tr>
<td><code>&lt;artist&gt;</code> Bonnie Tyler &lt;<code>/artist&gt;</code></td>
<td><code>&lt;td&gt;</code> Bonnie Tyler &lt;<code>/td&gt;</code></td>
</tr>
<tr>
<td><code>&lt;country&gt;</code> UK &lt;<code>/country&gt;</code></td>
<td><code>&lt;td&gt;</code> UK &lt;<code>/td&gt;</code></td>
</tr>
<tr>
<td><code>&lt;company&gt;</code> CBS Records &lt;<code>/company&gt;</code></td>
<td><code>&lt;td&gt;</code> CBS Records &lt;<code>/td&gt;</code></td>
</tr>
<tr>
<td><code>&lt;price&gt;</code> 9.90 &lt;<code>/price&gt;</code></td>
<td><code>&lt;td&gt;</code> 9.90 &lt;<code>/td&gt;</code></td>
</tr>
<tr>
<td><code>&lt;year&gt;</code> 1988 &lt;<code>/year&gt;</code></td>
<td><code>&lt;td&gt;</code> 1988 &lt;<code>/td&gt;</code></td>
</tr>
<tr>
<td><code>&lt;/cd&gt;</code></td>
<td><code>&lt;/tr&gt;</code></td>
</tr>
<tr>
<td><code>&lt;/Catalog&gt;</code></td>
<td><code>&lt;/table&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;ns0:CatalogTable&gt;</code></td>
</tr>
</tbody>
</table>
**Runtime Procedure**

The mapping editor generates Java source code from the graphical description of the message mapping. This section gives you a brief overview of how the generated Java source code processes an XML source structure at runtime.

**Basics**

In the mapping editor you assign the fields of two XML structures to each other. Note that at runtime no target structure exists prior to the mapping. The target structure is only generated when the mapping is executed.

The message mapping must therefore generate the target structure at runtime. Two things determine how the target structure looks after it has been generated:

- The number and type of tags which exist in the source structure instance
- The maximum occurrence of target fields permitted in the target structure

In both cases, this in turn depends on the value for Occurrence that you specified in the type definition in the XSD editor. The values for Occurrence are mapped to the XSD attributes minOccurs and maxOccurs. You can display these attributes in the tree and table view [Seite 148]. minOccurs specifies the minimum number of times a tag must occur, while maxOccurs specifies the maximum number of times a tag may occur. The mapping editor maps the values of required to minOccurs=maxOccurs=1 and those of optional to minOccurs=0 and maxOccurs=1 for attributes.

If the occurrence of a tag in the source or target structure is not explicitly specified, the mapping editor makes this visible to the user. These default values correspond to the XSD specification (minOccurs="1", maxOccurs="unbounded").

The message mapping processes the source structure and generates the target structure taking into account to the restrictions for the target fields:

- This mapping is clear for elements that occur only once.
- Frequently occurring elements [Seite 173] are mapped appropriately at runtime.
- By using contexts [Seite 175] you can create special structure mappings.

**Frequently Occurring Elements**

Even if a source field in accordance with an XML schema definition is not limited in how often it can occur, it only actually occurs in the XML document for which the message mapping was executed. The target field attributes minOccurs and maxOccurs then determine how often it occurs in the target structure. For the purpose of the statements below, sourceElem is a source structure tag and targetElem is a target structure tag that is assigned to this field in the mapping editor (without using functions):
• The tag `targetElem` with `minOccurs=“<n>”` occurs a minimum of `<n>` times in the target structure. If `sourceElem` does not occur often enough in the source structure, the message mapping generates `targetElem`-tags without values.

• The tag `targetElem` with `maxOccurs=“<n>”` occurs a maximum of `<n>` times in the target structure. If `sourceElem` occurs frequently in the source structure, values are lost. If it occurs less frequently, then an equal number of target and source fields are generated.

### Generating Functions

The standard functions of the `Constants` functions category do not have an input value. Therefore, they generate a value for a target field instead of taking it from a source field. However, what happens if the target field can occur more than once? Should the minimum number of possible fields then be generated, or the maximum?

In the example instance below, `maxOccurs` is set to 40 for the `<count>` target field and `minOccurs` is set to 1:

```xml
<Material>
  <nr>345200</nr>
  <count>4</count>
</Material>

<Material>
  <nr>98673</nr>
  <count>6</count>
</Material>

[...]
```

The source structure gets multiple elements of type `<Material>`, which is a sub node of the element `<count>`. Let us assume you want to reverse each occurrence of the counter `<count>` in this target structure by using a mapping. To do so, you would intuitively assign the constant 0 to the `<count>` field. However, if the value is only generated once, because `minOccurs` is set to 1, what should then happen with the other fields? The best solution is to leave them empty. If you connect the `Constant` function to the `<count>` target field, the message mapping assigns the specified constant to all 40 fields.

To conclude, note the following:

• All functions (regardless of whether they are standard or user-defined functions) that do not have input values are referred to as *Generating Functions*. This also includes the function `CopyValue()`.

• A target field with `maxOccurs=“<n>”` that is assigned a generating function is generated `<n>` times in the target structure. If `maxOccurs=unbounded` for the target field, then exactly 5 target fields are generated.
**Contexts**

When the instance for the source structure is parsed, a message mapping works internally by using queues.

### XML Instance

```
<item> 3 </item>
<item> 2 </item>
</item>
```

### Item Queue

3
2

---

### XML Instance

```
<item> 10 </item>
<item> 11 </item>
<item> 43 </item>
<item> 2 </item>
</item>
```

### Item Queue

10
11
43
2

---

### XML Instance

```
<item> 7 </item>
```

### Item Queue

7

---

If no further elements are imported at a particular hierarchy level, a **Context Change** is inserted in the queue. Therefore, in the example above elements 2 and 3 belong to a context. By inserting or removing context changes you can make use of these internal representations for special structure mappings:

- To delete all context changes of a queue, simply delete the relevant hierarchy level. Example: If you delete the context for the queue for element `<item>`, then all elements of the item queue are only assigned to an upper-level context, in other words, they have a joint upper-level element `<A>`. Also see: removeContexts [Seite 162].
- You can insert additional context changes depending on the values in a queue, for example after each value. In doing so, you insert a new hierarchy level. Also see: SplitByValue [Seite 165].

---

**Testing a Message Mapping**

### Purpose

You can check whether a complete message mapping functions at runtime once you have described it, by using the **Test** tab page. For this purpose, the mapping editor generates, compiles, and then executes Java source code for the message mapping.

The Integration Builder cannot yet access the value-mapping table at this point. When using the ValueMapping function (Conversions category), the value mapping is merely estimated as text output in the test results.
**Prerequisites**

The defined message mapping must be complete. This means that each mandatory target field must be assigned to one or more source fields. Furthermore, there must not be any unassigned arguments when using functions.

If you used two XSD files for the message mapping description, you must create an XML instance for the source structure. Using this instance, the mapping editor executes a transformation to the target structure in accordance with the mapping described.

**Process Flow**

1. The system navigates to the **Import** tab page.
   - On the **Test** tab page the left-hand side is used to view the **Source XML Instance** and the right-hand side to view the result of the transformation (**Target XML Instance**). You have the same display options as for the **Design** tab page.

2. To load an XML instance for the source structure, choose **Load XML Instance**.

3. To edit the XML instance before the test, go to the source document view. Only a copy of your XML instance is modified.

4. To test your mapping using the XML instance, choose **Execute Mapping**.

**Result**

If you are testing the message mapping for the first time or have modified it since the last test, the mapping editor compiles the generated Java source code. The result of the transformation is displayed on the right-hand side. Any errors that occurred during compilation are displayed in the lower text frame for **Status Messages**. An error may occur because a user-defined function was not syntactically correct, for example.

---

**XSLT and Java Mappings**

**Use**

You can also implement XSLT and Java mappings yourself and save them as archives in the Integration Repository. This enables you to reuse mapping programs that already exist.

**Prerequisites**

You have already implemented an XSLT or Java mapping and packed it in a Jar or Zip file. SAP recommends that you pack XSLT and Java mappings separately in archives since this improves system performance when searching for programs.

⚠️ In the case of Java archives, it is very important that the archive contains the compiled classes (*class). Java source files are ignored by the Integration Builder and the mapping runtime.

**Features**

Details about the options that you have in XSLT and Java are contained in the XSLT Mapping [Seite 177] and Java Mapping [Seite 183] sections. It is also possible to call other mapping
programs from a mapping program (see: References Between Mapping Programs [Seite 186]).

**Displaying and Changing Imported Mapping Programs in the Integration Builder**

<table>
<thead>
<tr>
<th>Function</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>📇 Display Program</td>
<td>Displays the selected XSLT program</td>
</tr>
<tr>
<td>(corresponds to a double click on the XSLT program)</td>
<td></td>
</tr>
<tr>
<td>🔧 Change Program</td>
<td>Changes the selected XSLT program. In an Integration Builder editor you can either change lines or import another XSLT program. You must edit Java mappings externally and then import them again.</td>
</tr>
<tr>
<td>🌐 Open in Client Browser</td>
<td>Displays the selected program in the client browser. To display the selected mapping file, start the Integration Builder from the default browser on the PC for class- or xsl-files.</td>
</tr>
</tbody>
</table>

**Activities**

**Importing XSLT and Java Archives**

1. Create a mapping object of type *Imported Archive* on the design maintenance screen of the Integration Builder (also see: Creating a New Object [Seite 40]).
2. Import the archive by choosing *Import Archive*. Once you have activated the archive other users can export it again.
3. Save the archive.

**Result**

You have saved the mapping programs in the Integration Repository. To be able to select these mapping programs for a scenario at configuration time, you must assign them to an interface pair using an interface mapping.

---

**XSLT Mapping**

**Use**

Interface descriptions are in the form of XML documents. XSL Transformation (XSLT) is a member of the XML family of languages. It describes how an XML structure is transformed into another XML structure.

**Features**

You can define mappings using XSLT together with XPath. XPath is also a specification of the XML family. Using XPath you can address any node in an XML document. XSLT implements XPath expressions to select substructures of an XML document. Using templates in XSLT you can define the mapping rules for the selected substructures.

You can use the XSLT tags `<xsl:include>` and `<xsl:import>` to include predefined templates for substructures in a complete mapping definition. In this way, you can reuse mappings for data types.
Moreover, using an XSLT definition you can call external Java methods to convert XML structures. This procedure gives you more flexibility when defining mappings. See further documentation for more information about XSLT mapping with Java enhancement [Seite 179].

See the Web page of the W3 Consortium for more information about XSL/XSLT.

**Example**

The function for booking a flight contains a parameter for the flight class. The outbound interface represents flight classes as a symbolic values; in the inbound interface, flight classes have numerical values. The XSLT section converts the symbolic representation of the source system into the numerical representation of the target system.

The XPath expression match="/hh:class" selects the node <hh:class> in the outbound interface. The XSLT statement <xsl:choose> then defines the mapping rule for how the Integration Engine converts the symbolic values in the source system into the numerical values in the target system.
XSLT Mapping with Java Enhancement

Use

The XSLT specification proposes an enhancement mechanism that enables the system to call external routines from XSLT for transforming XML structures. Java is a programming language that is supported by some Java XSLT processors as an enhancement language.

Using the XSLT mapping with Java enhancement, you can implement mappings quicker and more flexibly.
The current working draft of the specification for XSL transformations (XSLT) version 2.0 (version from December 20, 2001), does not define the features of the enhancement mechanism of the XSL transformation. There are also no accompanying standards that standardize the enhancement mechanism with Java. For this reason, XSLT processors available on the open market vary considerably in this respect.

Below is a description of one of the ways that the SAP J2EE Engine supports the XML Toolkit 2.0.

**Activities**

To implement an XSLT mapping with Java enhancement, proceed as follows:

1. Implement a Java class that contains the static methods of transforming XML documents or structures.
2. Include the method calls in the XSLT mapping program.
3. Import the XSLT mapping program and the Java class as an archive to the Integration Repository. If you have two separate archives they must be imported into the same namespace.

The following example illustrates the procedure using a simple message mapping.

**Example**

The following graphic shows a source message that is to be transferred to the target message, using an XSLT mapping program with Java enhancement.

```
Source Message                   Target Message

<person>
  ...
  <first-name>Robert</first-name>
  <last-name>White</last-name>
  ...
</person>

<person>
  ...
  <name>Robert White</name>
  ...
</person>
```

The class method `merge()` of the Java class `com.company.group.MappingClass`, groups the values of the elements `<first name>` and `<last name>` of the source message together in a string that represents the value of the element `<name>` of the target message.

```
package com.company.group;

public class MappingClass {
  ...
  public static String merge(String first, String last) {
    return first + ' ' + last;
  }
  ...
}
```
In the XSLT mapping program in which you want to call the Java method above, declare the Java class with namespace definition as an attribute of the element `<xsl:stylesheet>`. The namespace name can be any string you want; in the example it is called `javamap`. The namespace ID comprises the string `java:` as a prefix and the full name of the class. The complete name of a Java class comprises the name of the package and the class name.

Use `<xsl:param>` to define the parameters that you want to transfer in the method calls. Give these parameters the appropriate values. In this example, two parameters of the signature of the method `merge()` are defined correspondingly. The parameter `first` adopts the value of the element `<first name>`, while the parameter `last` adopts the value of the element `<last name>`.

In this example, the method call is called with the element `<xsl:value-of>`. Specify the method with the corresponding parameters using the attribute `select`. The name for the namespace `javamap` replaces the complete class name as a prefix.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:javamap="java:com.company.group.MappingClass">
    <xsl:param name="first" select="/first-name"/>
    <xsl:param name="last" select="/last-name"/>
    <xsl:template match="/">
        ...
        <name>
            <xsl:if test="function-available('javamap:merge')">
                <xsl:value-of select="javamap:merge($first, $last)"/>
            </xsl:if>
        </name>
        ...
    </xsl:template>
</xsl:stylesheet>
```

Using this mapping program, the source message is converted to the target message.

---

**Error Handling in XSLT**

**Use**

In substantial XSLT programs it makes sense that the system displays an explanatory message when an error occurs. For this purpose there is the XSLT command `<xsl:message>`. For example, the source document might contain a value that is not expected by the mapping program. In this case the system can display a message, and, if necessary, terminate processing of the mapping.

**Integration**

Messages that are produced using the XSLT command `<xsl:message>` are visible in the trace function of the technical monitor (also see: Monitoring XML Messages [Seite 290]).
Features

The XLST command `<xsl:message>` has the following syntax:

```
<xsl:message terminate="[yes|no]">
  <!-- Output Message -->
</xsl:message>
```

The attribute `Terminate` determines whether processing of the XSLT program should be terminated (terminate="yes") or not (terminate="no"). In both instances you can view the error message text by using the trace function for a message in monitoring (provided that you activated the trace function for the Integration Engine).

Example

The following example of a section of an XSLT program is taken from a mapping program with the name `Person2Passenger.xsl`. The XSLT commands in this section evaluate the current value of a tag from the source document and map the values as follows:

<table>
<thead>
<tr>
<th>Source Document</th>
<th>Target Document</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Mr.</td>
</tr>
<tr>
<td>1</td>
<td>Mrs.</td>
</tr>
<tr>
<td>(Other)</td>
<td>Mr. or Mrs.</td>
</tr>
</tbody>
</table>

The last instance is an exception for non-interpretable values. To retain this exception in the trace, a message is produced using `<xsl:message>`:

```
<xsl:choose>
  <xsl:when test="test()=0">Mr.</xsl:when>
  <xsl:when test="test()=1">Mrs.</xsl:when>
  <xsl:otherwise>Mr. or Mrs.
    <xsl:message terminate="no">
      Gender has neither value 0 nor 1!
      "Mr. or Mrs." was chosen for title.
    </xsl:message>
  </xsl:otherwise>
</xsl:choose>
```

Since the target field can still be completed appropriately, processing is not terminated (terminate="no"). In the technical monitor, the section in the trace for the third instance looks as follows:

```
<CALL kind="ENTER" func="CL_XMS_PLSRV_MAPPING~ENTER_PLSRV" />
<trc1>XSLT transformation: xslt/wsxx/Person2Passenger completed. Gender has neither value 0 nor 1! "Mr. or Mrs." was chosen for title. </trc1>
<CALL kind="EXIT" func="CL_XMS_PLSRV_MAPPING~ENTER_PLSRV" />
```
Java Mapping

Use

You can implement mapping programs in Java. To process XML documents, use Java API for XML Processing (JAXP), for example. The JAXP supports the Document Object Model (DOM) and the Simple API for XML (SAX). This gives you great flexibility for mapping definitions with Java.

Activities

To use Java mapping, you must define a Java class that implements the Java interface `com.sap.aii.mapping.api.StreamTransformation`. This interface has two methods:

1. `public void execute(java.io.InputStream in, java.io.OutputStream out)`

   At runtime, the Integration Engine calls this method to execute a mapping. This method contains an input stream for the outbound document and an output stream for the target document as parameters. These streams are often XML documents. You can import the substructure that you want to convert from the input stream and output the converted target document in the output stream.

2. `public void setParameter(java.util.Map param)`

   The Integration Engine transfers parameters to the mapping program with this method. It evaluates these parameters at runtime in the method `execute()`. This enables you to control the process run of the mapping.

   The transferred object that implements the Java interface `java.util.Map`, contains seven key/value pairs as parameters. The value objects are of type `java.lang.String`. The key objects are defined in the class `com.sap.aii.mapping.api.StreamTransformationConstants`:

   - `SENDER_NAME`: Name of sender.
   - `SENDER_NAMESPACE`: Namespace of sender.
   - `SENDER_SYSTEM`: Sender system identifier.
   - `RECEIVER_NAME`: Name of receiver.
   - `RECEIVER_NAMESPACE`: Namespace of receiver.
   - `RECEIVER_SYSTEM`: Receiver system identifier.
   - `DIRECTION`: Direction of message mapping. This parameter is only relevant for message mappings; only the strings “Request”, “Response” and “Fault” are permitted as values.

Example

The following example shows how parameters are set and evaluated in a Java mapping program:
import com.sap.aii.mapping.api.StreamTransformation;
import com.sap.aii.mapping.api.StreamTransformationConstants;

class public JavaMapping implements StreamTransformation {
    private Map myParam;

    public void setParameter(Map param) {
        myParam = param;
    }

    public void execute(InputStream in, OutputStream out) {
        // ...
        String senderName = (String)myParam.get(
            StreamTransformationConstants.SENDER_NAME);
        // ...
    }
}

Executing Value Mappings

Use

At configuration time you can complete a value-mapping table [Seite 53] in the Integration Directory. Values that are semantically the same are summarized in one group in this table.

The mapping runtime provides an API to determine a target value during a Java mapping for a source value.

Features

The classes of package com.sap.aii.mapping.key.* provide the functions required to execute a mapping:

- Using the abstract class KeyMappingService you can do the following:
  - Return an instance of this service class
  - Return an instance for a representation
  - Query a target value for a display by using query methods

- Interface IRepresentation enables you to access the value representation (system, namespace, object type and value).

Methods of Class KeyMappingService

<table>
<thead>
<tr>
<th>Method</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Returns an instance of the class KeyMappingService. Use this service class to execute the value mapping.

Generates a new representation for a value.

Generates an instance for the representation of a value without the value itself.

Queries the target value for a source representation. The result is returned using both the parameter target and the return value.

Queries the target value for a source representation without using the class for the representation. The result is only returned using the return value.

Unlike query, no exception is thrown here if no target value is found.

Methods of the Interface IRepresentation

<table>
<thead>
<tr>
<th>Method</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>public String getSystem();</td>
<td>Returns the system for a representation</td>
</tr>
<tr>
<td>public String getNamespace();</td>
<td>Returns the namespace for a representation</td>
</tr>
<tr>
<td>public String getObjectType();</td>
<td>Returns the object type for a representation</td>
</tr>
<tr>
<td>public String getKey();</td>
<td>Returns the value for a representation</td>
</tr>
</tbody>
</table>

Activities

1. Import the classes of the package com.sap.aini.mapping.key.* to your Java mapping.
2. Generate an instance of the class KeyMappingService:
   
   ```java
   KeyMappingService s = KeyMappingService.getInstance();
   ```
3. Using the methods of the generated instance, query a target value for a source value:
   
   - Generate an instance for the representation of the source and the target value (class representation) and leave out the value in the representation of the target value. Then transfer the representations to the method `query`. If no target value is found, the mapping runtime throws an exception.
   - If you do not want to throw an exception here, use the method `queryS`. In this case, you do not need to generate instances for the representation.
Result

You can now use the target value in the target structure.

References Between Mapping Programs

Mapping programs that come under the same namespace can reference and use each other:

- The Java classes of an archive can be used in the user-defined functions [Seite 170] of a message mapping provided that the archive is in the same namespace as the message mapping.
- Java classes in different archives can use each other provided that the archives are in the same namespace.
- XSLT programs in different archives can include or import each other provided that the archives are in the same namespace. It is also possible to call Java methods from an XSLT mapping (see: XSLT Mapping with Java Enhancement [Seite 179]).

SAP recommends that you separate Java classes and XSL when structuring archives since this can improve performance.

Proxy Generation

Purpose

Using proxy generation, you can generate executable components for interfaces that you have created on the interface maintenance screen in the Integration Repository [Seite 19]. You generate a proxy from a platform-independent description of an interface in the Integration Repository. This proxy enables systems to communicate with each other using the SAP Exchange Infrastructure. There are two types of interface objects in the Integration Repository:

- Interfaces from existing components that you can import into the Integration Repository (for example, RFC function modules). An adapter [Seite 340] enables the Integration Engine to convert calls from these interfaces into XML format; this format can then be understood by the Integration Server [Seite 23].
- Interfaces that you create in the Integration Repository are platform-independent. To use an interface, you must first generate a proxy for a component.

The first approach is referred to as inside-out development, because interfaces that already exist in the system are called externally. The second approach is referred to as outside-in development, because interfaces can first be drafted independently of a component and then be implemented in a system landscape later. The outside-in approach is detailed below.
Integration

Proxy generation fetches information about an interface into the Integration Repository using HTTP, and converts it into a proxy object. The format in which the interfaces are read corresponds to the WSDL (Web Service Description Language) standard. In WSDL, data that is transferred using messages is described using XSD (XML Schema Definition Language).

Features

Outbound and Inbound

There are two kinds of interfaces: Outbound and Inbound Interfaces (also see: Properties of Message Interfaces [Seite 123]). You must generate two types of proxies correspondingly:

- For an outbound interface you generate an outbound proxy. This is a class with a method in the respective target language (see below). The application calls this method to send a message to a receiver.

- For an inbound interface you generate an inbound proxy. This is not a class as in the outbound case, but a (Java or ABAP object) interface that the application must implement to be able to process an inbound message.

The generated proxies encapsulate the conversion of a method call in an XML message (outbound case), or the parsing of such a message and the call of the implementing class (inbound case).

Target Languages

You can generate proxies for the following languages:

- ABAP Proxies  
  (only possible for SAP systems that have installed the SAP Exchange Infrastructure Add-On)

- Java proxies for J2EE applications on the SAP Web AS and for standalone applications

Example

The following graphic is a simplified illustration of how to generate and use proxies. The starting point is an outbound interface and a corresponding inbound interface in the Integration Repository. To implement this interface at runtime, the application generates corresponding proxies for each platform. (Also see: Interface-Based Message Processing [Seite 17]).

You can, of course, also generate inbound proxies for Java and outbound proxies for ABAP.
The generation of proxies represents the transition from the design phase to runtime. The graphic outlines how a message is sent from a Java application to the Integration Engine, using an outbound proxy. There it is assigned to a logical receiver business system as per the settings in logical routing [Seite 214]. It can then be assigned to a receiver interface and a mapping [Seite 143] using a mapping relation. Furthermore, an end point [Seite 231] determines the technical receiver of the message (technical routing). Finally, the Integration Engine forwards the message to a runtime component at the receiver, which forwards it to an ABAP application using an inbound proxy that complements the outbound proxy (also see: Runtime [Seite 264]).

The graphic only shows one direction of communication, which is the case with synchronous communication, for example. The interfaces in the Integration Repository have corresponding attributes that indicate whether they are synchronous or asynchronous interfaces.

---

**Converting WSDL to the Target Language**

As illustrated in the graphic below, a WSDL document converted by proxy generation comprises two parts, namely a description of data types and an interface description, the parameters of which are used by these data types:
The proxy generation functions fetch the WSDL description from the Integration Repository [Seite 19] and convert it to proxies:

- The WSDL elements of the interface description are converted on a 1:1 basis into objects of the respective target language (classes, interfaces, methods).
- The conversion of the data types to XSD (XML Schema Definition Language) can be subject to restrictions, depending on the target language (see below).

The WSDL description contains, among other things, the structure of a message. The message that is then sent using the Exchange Infrastructure therefore corresponds to an instance of this description (for more information see: Messages [Seite 15]).

Also see:

| ABAP Proxy Generation [Seite 190] | Java Proxy Generation [Seite 198] |

**Data Type Generation**

Data types from the Integration Repository are described using XSD. You can also define your data types with XSD tags in the XSD editor on the interface maintenance screen. Although you can only select tags in the XSD editor that you can convert to the target language, there can be additional restrictions as well as those in the XSD editor. These restrictions are described in the attachment in the target language section in each case (for more information see: Attachment: Mapping XSD to ABAP [Seite 197] or Attachment: Mapping XSD to Java [Seite 205]). The following section is a brief introduction to XSD.

**XSD: Brief Introduction**

The following XSD data types exist:

- XSD provides numerous in-built data types (for example `xsd:decimal`) and defines a clearly defined value range. You can use them globally. Many of these data types cannot be mapped exactly to ABAP data types because the value ranges do not match. In cases such as these, you must check yourself whether a received value is within the value range of the data type in the respective target language.

- You can reconstruct new data types from in-built data types using the tag `<simpleType>`. For example, you can define a pattern for `xsd:string` for telephone numbers. The proxy runtime does not check this type of enhancement because they are checked when the system validates an XML document.
• You can then define structured data types using the tag `<complexType>`. Elements that are located between this and the corresponding closing tag are local and you can only access them using the upper-level type. Instead of this nesting you can reference other structured or simple data types for complexType tags.

Use the tag `<element>` to then define the elements that can occur in the XML schema and use `<attribute>` to define the attributes of these elements. You can then use the XSD data types to assign types. Definitions with `<element>`, `<attribute>`, `<simpleType>`, and `<complexType>` are visible globally if the subnodes of the top-level node are `<schema>`. You can only use nested types, elements, or attributes locally.


---

**ABAP Proxy Generation**

<table>
<thead>
<tr>
<th>ABAP</th>
<th>Java</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Seite 198]</td>
<td></td>
</tr>
</tbody>
</table>

**Use**

Using ABAP the proxy generation function (transaction SPROXY) you can generate ABAP proxy objects in an SAP system from an interface description in the Integration Repository.

**Prerequisites**

ABAP proxy objects [Seite 192] can only be generated for an SAP system for which SAP Exchange Infrastructure Add-On 2.0 has been installed.

The proxy generation functions fetch the WSDL description of the interface from the Integration Repository, using HTTP. The address of the corresponding server is taken from the exchange profile (parameter 1, 2, and 3. See below). Queries to the repository undergo an authentication procedure. User and password for these queries are also taken from the exchange profile (parameter 5 and 6). Information from the exchange profile (parameters 1, 2, and 4) is also required to navigate from transaction SPROXY to the Integration Builder start page.

**Example Configuration for Accessing Interfaces in the Integration Repository (Exchange Profile)**

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Section</th>
<th>Parameter</th>
<th>Value (Example)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Connections</td>
<td>com.sap.aii.connect.repository.name</td>
<td>pwdf0436</td>
</tr>
<tr>
<td>2</td>
<td>Connections</td>
<td>com.sap.aii.connect.repository.httpport</td>
<td>1080</td>
</tr>
<tr>
<td>3</td>
<td>Connections</td>
<td>com.sap.aii.connect.repository.contextRoot</td>
<td>rep</td>
</tr>
<tr>
<td>4</td>
<td>Connections</td>
<td>com.sap.aii.connect.integrationrep/start/index.jsp</td>
<td>rep/start/index.jsp</td>
</tr>
</tbody>
</table>
These parameters are not to be set by developers but by the administrator responsible for the technical configuration of the XI system landscape.

You must also have created interfaces in the interface maintenance of the Integration Repository before you can generate for these proxies.

**Features**

The system displays a navigation tree when you call transaction SPROXY. In the tree you can view the interface objects from the Integration Repository. If no connection to the Integration Repository exists, the tree presents an overview of those interface objects for which a proxy object already exists in the system. You can display or generate objects from the context menu in the navigation tree. The system displays the properties of the proxy objects on the right-hand side of the screen:

**Tab Pages for Transaction SPROXY**

<table>
<thead>
<tr>
<th>Tab Page</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Properties</td>
<td>Generation attributes such as package, last changed by, and so on. For inbound proxies, specify the name of the implementing class here.</td>
</tr>
<tr>
<td>Problems</td>
<td>The system only displays this tab page directly after generation. You can correct the names of labels that had to be abbreviated, or due to a naming conflict, had to be changed during generation.</td>
</tr>
<tr>
<td>Generation</td>
<td>All objects generated for the object as a list.</td>
</tr>
<tr>
<td>Structure</td>
<td>Similar to generation, except that here objects are ordered in a tree structure according to their use. Example: Class CO_X -&gt;Method MYMETHOD -&gt;Importing OUTPUT</td>
</tr>
<tr>
<td>Documentation</td>
<td>The system displays the documentation from the Integration Repository for the outbound object.</td>
</tr>
<tr>
<td>Warning Messages</td>
<td>Even if proxy generation was successful, there may be cases where generation was only possible on the basis of implicit assumptions (for example, restrictions to the value range are checked by the programmer). If cases such as these occur during generation, the system lists them in an application log.</td>
</tr>
</tbody>
</table>

**Navigation Tree Views**

**Default**

The navigation tree only displays those software component versions for SAP components that are available locally in the system. The selection of software component versions in the tree is therefore a subset of the software component versions in the Integration Repository.
While all imported CRM, ABA or APO software component versions are displayed in the Integration Builder, the SAP system has the corresponding components of a particular version, for example ABA 6.20 and CRM 3.0. For this reason, only these software component versions are displayed in the navigation tree.

Displaying Namespaces Known in the System
Transaction SPROXY structures the default navigation tree using information from the Integration Repository. If namespaces are deleted in the Integration Repository then they are no longer visible in the navigation tree in transaction SPROXY, however it is possible that proxy objects already exist in the system for these namespaces. To display these proxy objects, choose Goto → View → Only Local Objects and display all namespaces for which ABAP proxy generation has ever generated proxy objects. Even proxy objects that have already been deleted in the Integration Repository are visible. To return to the default setting, choose Goto → View → All Objects.

Activities
1. Call transaction SPROXY.
2. If no connection can be established with the Integration Repository when you start the transaction, choose the menu path Goto → Test Connection to establish the exact cause of the error and to correct it.
3. Generate proxy objects for an interface or for other interface objects (see: Generating Proxies [Seite 195]).
4. Check whether you need to make any changes following the automatic naming of proxy objects [Seite 195].
5. If the description was changed in the Integration Repository, you must regenerate the proxy objects [Seite 196].

ABAP Proxy Objects
Definition
All ABAP objects that are created when you generate an interface object from the Integration Repository in an SAP system are called ABAP Proxy Objects.

Structure
You can divide the different interface objects (the objects from the Integration Repository from which you generate proxies) into two classes: The message interface itself and the corresponding parameters and data types (also see: Converting WSDL to the Target Language [Seite 188]). Each generated proxy object can be assigned to its corresponding interface object in the Integration Repository even after generation. The ABAP proxy generation function notes this assignment using metadata, which is transported with the proxy object.
Interfaces

There are two types of interfaces to differentiate between here, namely synchronous and asynchronous interfaces. In the case of asynchronous interfaces, the sender system can continue with the business process locally after it has sent the message, since the receiver system does not return a response to the sender system. In the case of synchronous interfaces, the receiver system returns a result, therefore the sender system is blocked until the response has arrived.

You call an outbound interface to send a message to an inbound interface. An outbound interface is mapped to an ABAP object class (prefix CO_).

You call an inbound interface to start a service that, in the synchronous case, returns a result. The proxy generation functions generate an ABAP object interface (prefix II_) for an inbound interface; you must implement this interface using an ABAP object class to make this service available.

This class must be recognized by the ABAP proxy runtime so that it can call the appropriate method when a message arrives. The proxy generation function enters an implementing class as the default setting on the tab page Properties; this class is generated after the proxy has been activated. If you want to use a class that already exists, enter it in this field.

The system generates methods with the following names, depending on the type of communication:

- In the case of synchronous interfaces, the method is called EXECUTE_SYNCHRONOUS
- In the case of asynchronous interfaces, the method is called EXECUTEASYNCHRONOUS

These names count regardless of whether the interfaces are inbound or outbound interfaces.

Parameters and Data Types

The following can be established for both inbound and outbound interfaces:

- The parameter for the Output Message Type is called OUTPUT. The parameter for the Input Message Type is called INPUT. OUTPUT refers to an outbound message, INPUT to an inbound message. Both parameters are structures of the respective message type with a field for the data component of the message.
- The data component of the message is communicated with by using the message type name. For example, if the message type is Report and it references the data type InvoiceData, then a structure Report of type InvoiceData is generated.

Proxy generation saves a reusable data type in the ABAP Dictionary for global XSD data types, for example data elements or structures. It generates fields of a structure for local elements or attributes. The reuse of data types in the Integration Repository is reflected in the data types created in the system.

Tables

There is no explicit table type in XSD. However, you can define elements so that there is no limit as to how often they may occur (also see: Elements and Attributes [Seite 135]). For example, assume you require a data type staff, which you can use to select a group of colleagues as members of staff and assign them a manager. The following table reflects the definition of this data type in the XSD editor:

Definition of an Unbounded XSD Element Within a Structure

<table>
<thead>
<tr>
<th>Structure</th>
<th>Category</th>
<th>Type</th>
<th>Occurrence</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>staff</td>
<td>Complex Type</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
According to this definition, the element `employee` can occur more than once in the corresponding XML schema. For `employee`, the proxy generation functions generate an ABAP Dictionary table type and a structure for the line type with the fields `name` and `birthday`.

**Integration**

You cannot change ABAP proxy objects manually in the SAP system. You can only change an ABAP proxy object by calling transaction `SPROXY`. ABAP proxy objects only map data structures defined in the Integration Repository. Therefore, they are not supported and must not be used on the interface.

**Example**

The following graphic illustrates how an interface from the Integration Repository is mapped to an ABAP proxy:

```
Outbound Message Interface (Integration Repository)  ABAP Proxy
```

Message Interface MI

- **Input Message Type IM**
  - Simple Data Type `ST`: `xsd:string`

- **Output Message Type OM**
  - Complex Data Type `CT`
    - `par1` `xsd:string`
    - `par2` `xsd:date`

Class `CO_MI`

- **Method** `EXECUTE_SYNCHRONOUS`
- **Importing OUTPUT Type OM**
  - OM Type `CT`
    - Structure `CT`
      - `par1` `type string`
      - `par2` `type date`
- **Exporting INPUT Type IM**
  - IM Type `ST`
Proxy Generation

Procedure

1. Call transaction SPROXY in the system in which you want to generate an inbound or outbound proxy.

The system displays a navigation tree on the left-hand side of the screen; it has the same structure as the interface maintenance tree in the Integration Builder. The interface objects are ordered according to software component version, namespace and object type (also see: ABAP Proxy Generation [Seite 190]).

If no connection to the Integration Repository exists, the system only displays interface objects in the tree structure for which a proxy exists in the system.

2. Choose an interface object for which no proxy already exists. To create a proxy for this interface object, double click the context menu Create Proxy.

3. In the dialog box, specify the name of the package where you want to create the proxy objects. You can also specify a prefix for the names of all objects to be created to avoid naming conflicts with names that already exist in the system.

Result

To generate proxies, the WSDL description of the message interface is read using HTTP. If no naming conflicts [Seite 195] occur during the conversion, you can call the hierarchy of the generated objects from the tab page Structure. The objects are only then saved in the system once you choose Activate. Up to the point of activation, you can save metadata that contains all information for generation and that is managed by the transaction, and continue with generation at a later date.

Naming Proxy Objects

Use

The length of names of interfaces, messages types, and data types is unlimited in the Integration Repository. Also, unlike ABAP, the system differentiates between upper case and lower case text; this is normally used to distinguish between words within a name (for example: FlightBooking).

Features

Standard Procedure

The different naming conventions in XSD and ABAP have the following consequences for proxy generation:

- When the text switches from lower to uppercase, the generation function inserts an underscore to make the change more clear (for example: FLIGHT_BOOKING).
- The system shortens names that are longer than 30 characters.
• If a naming conflict occurs, the generation function appends a number to the new name to enable you to distinguish between the names.

In the last two points, the system displays a dialog box indicating that you need to change the original name.

Adjusting Names Changed by Generation

After the system has indicated the changes that you need to make to the original name, you can adjust the names to make them clearer using the tab page Problems.

Adjusting Names Unchanged by Generation

You can also change the names of proxy objects that did not need to be changed during generation. It is a good idea, for example, if you have had to make changes to the tab page Problems, to adjust the remaining names of parameters of the same interface in the same way. To do this, switch to the tab page Structure after you have adjusted all the names on the tab page Problems. Provided that the proxy objects have not been used in other proxy objects, you can also change these names by choosing Change Name/Short Text in the context menu.

Accepting Changed Names

When you are happy with your changes, to create the objects in the system, choose Activate. In addition to the generated objects, proxy generation also saves meta data with information about assigning it to an interface in the Integration Repository. This meta data is transported together with the proxies.

Regenerating Proxy Objects

Use

You also use the Integration Builder after you have generated a proxy to administer the corresponding platform-independent interface. Therefore, proxy generation locks the objects created in the system against manual changes. To change a proxy object that has already been generated, change the object on the interface maintenance screen of the Integration Builder, and regenerate the object.

Procedure

1. Change the interface (for example, a data type) in the Integration Builder and save your changes.
2. Call transaction SPROXY. In the navigation tree, either choose the changed interface object itself, or an object that contains the changed object.
3. Call the context menu for the interface object and choose Change Proxy.

Result

Using the interface description in the Integration Repository, the proxy generation function determines whether the specified object, or the objects used by the object, have been changed. If this is the case, the system regenerates the changed objects.
## Attachment: Mapping XSD to ABAP

### Simple XSD Data Types and Their Exact Equivalent in ABAP

<table>
<thead>
<tr>
<th>XSD</th>
<th>ABAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>String (Length &gt; 65535)</td>
<td>STRING</td>
</tr>
<tr>
<td>String (Length &lt;= 65535)</td>
<td>CHAR</td>
</tr>
<tr>
<td>String with pattern=&quot;\d+&quot; (Length &lt;= 65535)</td>
<td>NUMC</td>
</tr>
<tr>
<td>Boolean</td>
<td>Mapping to a character field of length 5 using the constants sai_true and sai_false (Type Pool: sai).</td>
</tr>
<tr>
<td>Int</td>
<td>INT4</td>
</tr>
<tr>
<td>Short</td>
<td>INT2</td>
</tr>
<tr>
<td>UnsignedByte</td>
<td>INT1</td>
</tr>
<tr>
<td>Date</td>
<td>DATS</td>
</tr>
<tr>
<td>Time</td>
<td>TIMS</td>
</tr>
<tr>
<td>Decimal with fractionDigits&lt;=14, totalDigits&lt;=31</td>
<td>DEC</td>
</tr>
<tr>
<td>Binary (Length &lt;= 65535)</td>
<td>RAW</td>
</tr>
<tr>
<td>Binary (Length &gt; 65535)</td>
<td>RAWSTRING</td>
</tr>
</tbody>
</table>

### Tags Supported for Complex Data Types (in Alphabetical Order)

<table>
<thead>
<tr>
<th>XSD</th>
<th>Equivalent in ABAP Dictionary</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;attribute&gt; (local)</td>
<td>Structure field</td>
</tr>
<tr>
<td>&lt;complexType&gt; (global)</td>
<td>Structure</td>
</tr>
<tr>
<td>&lt;complexType&gt; (local)</td>
<td>Structure</td>
</tr>
<tr>
<td>&lt;element maxOccurs=&quot;unbounded&quot;&gt;</td>
<td>Structure field and a table type</td>
</tr>
<tr>
<td>&lt;element&gt; (local)</td>
<td>Structure field</td>
</tr>
</tbody>
</table>

⚠️ Only those tags permitted by the XSD editor [Seite 133] are executed.

### Restrictions for XSD Elements

<table>
<thead>
<tr>
<th>XSD Element</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;extension&gt;</td>
<td>These elements cannot be converted.</td>
</tr>
<tr>
<td>and abstract types</td>
<td></td>
</tr>
</tbody>
</table>
Substitution group and
<element abstract="true">
<redefine>
Restriction of value area
(<pattern>, <enumeration>,
<maxInclusive>, <maxExclusive>, etc.)
Key references, uniqueness (<key>, <keyref>,
<unique>, and so on.)
These tags are ignored by the proxy runtime. The schema validation checks these properties.

### Java Proxy Generation

#### Use
Using the Java proxy generation function, you can generate Java classes or Java proxy objects [Seite 200] from the interface description in the Integration Repository. Using these objects, you can then implement sender and receiver applications in Java; the proxy objects establish the connection to the Exchange Infrastructure by using the Java proxy runtime [Seite 322].

#### Features
There are two application instances for which you can generate proxies:

- J2EE applications on the SAP Web Application Server. Proxy generation generates J2EE beans and proxy classes for this purpose.
- For standalone applications that want to call a system synchronously. No J2EE beans are required in this case.

A subset of Java classes are generated for standalone applications. Java proxy generation in the Integration Builder supports the following for both application instances:

- Generation of a Jar file for one or more message interfaces of the same software component version.
- Opening of existing Jar files to regenerate proxies. Proxy generation knows the original message interfaces for which proxy objects are contained in the Jar file.

#### Selecting Message Interfaces
You can call Java proxy generation from the design maintenance screen of the Integration Builder in the following ways:

- From the main menu, choose Tools → Java Proxy Generation.
- From the context menu, choose Java Proxy Generation... for message interfaces in the navigation tree.
The latter method has the advantage that the selected message interface and the corresponding software component version can be copied directly.

Once the Java proxy generation window is open, you can transfer multiple message interfaces of a software component version to a list for subsequent generation. Use either of the following methods:

- Context menu *Java Proxy Generation...* (see above).
- Drag&Drop from the navigation tree to the window.

### Activities

#### Generating New Java Proxies

1. Navigate to the design maintenance screen of the Integration Builder.
2. Select the message interfaces for which the Java proxies are to be generated (see above).
3. The default setting generates J2EE beans. If you do not require any beans, deselect this option.
4. Choose *Recreate Jar File* and specify a file name for a new Jar file. If the specified file already exists it will be overwritten.
5. To start generation, choose *Generate Proxy*. A Jar file is not generated if errors occur during generation.

#### Regenerating Java Proxies

If you want to change any data types, message types, or message interfaces, you must do so using the Integration Builder in the Integration Repository. After any such changes, new Java proxies must be generated. You can only regenerate proxy classes for the entire message interface. The procedure is the same as when you generate Java proxies for the first time, with the exception that you must first identify the message interface from which the Jar file was generated. To do so, choose *Open Jar File*. In this way, Java proxy generation automatically includes it in the list again.

### Result

The resulting Jar file contains the corresponding Java source text files of the Java proxy objects [Seite 200].

### Tips

- Technically speaking, a Jar archive corresponds to a Zip archive. You can view the contents of the generated Jar archive in Windows, using WinZip, for example. Using the `jar` command, you can display the contents by using `jar -tf <Filename>` and read the contents by using `jar -xf <Filename>`.

- To gain an overview of the generated classes, generate it with `javadoc` HTML documentation.
Java Proxy Objects

Definition

All Java objects that are created from a message interface [Seite 126] from the Integration Repository in an SAP system, are called Java Proxy Objects. Multiple proxy objects are created in the system for each message interface (Java classes or Java interfaces).

Message interfaces contain message types, and these in turn contain data types. All objects referenced to message interfaces (including fault message types), are also called Interface Objects. They are the counterparts of the generated proxy objects.

Structure

You can divide the different interface objects into two classes: The message interface itself and the corresponding parameters and data types (also see: Converting WSDL to the Target Language [Seite 188]).

Each generated proxy object can be assigned to its corresponding interface object in the Integration Repository even after generation.

Interfaces

Interfaces have a direction (outbound or inbound) and a mode (synchronous or asynchronous). For more information, see Communication Parameters [Seite 120].

The Java runtime is at present still unable to provide queues for the asynchronous processing of messages. If an error occurs when sending an asynchronous message, an exception is thrown.

Depending on the interface direction, the following proxy objects are generated:

- You call an inbound interface to start a service that, in the synchronous case, returns a result. Proxy generation generates a Java interface for an inbound interface. To make this service available, implement this interface using a Java class. Note the following for the implementing class: If the name of the generated Java interface is JavaInterface, then the implementing class must be JavaInterfaceImpl and it must be located in the same package as the generated Java interface.

- You call an outbound interface to send a message to an inbound interface. An outbound interface is mapped to a Java class.

- You must generate additional bean classes for J2EE applications. In this case, send and receive messages by using the bean classes and not directly by using the proxy classes. These classes are not required for Java standalone applications.

In general, the signature of the generated method for a message interface $MI$ from the Integration Repository, looks as follows (in the synchronous case, without return parameters):

- Outbound: $\text{public <class for input message>} \ MI(<\text{class output message}>)$
- Inbound: $\text{public <class for output message>} \ MI(<\text{class input message}>)$

The class for the input or output message corresponds to the data type referenced by the message type used. No public global classes are generated for message types (fault message types are the only exception here).
Parameters and Data Types

The following objects are generated, regardless of whether they are inbound or outbound:

- A class for each complex data type generated that contains set/get methods for accessing the respective fields.
- No classes are required for simple data types. For example, if a simple data type of type `xsd:string` is used, Java proxy generation does not generate a separate global class for the data type. Instead, it uses the corresponding Java data type `java.lang.String` directly.

In XSD, you can also define elements that can occur infinitely in the message. This exceptional case is described in Tables [Seite 202].

Example

The following graphic illustrates how an interface from the Integration Repository is mapped to a Java proxy:

Outbound Message Interface (Integration Repository)

![Graphical representation of the message interface and Java proxy]

The original names of the interface objects are highlighted in bold. You can see that proxy generation had enhanced the names (see also: Naming Java Proxy Objects [Seite 205]). The method `bookOut` expects the formal parameter `om` of type `Ct_Type`. Therefore, you transfer the data type directly. The message type is included internally by the proxy runtime by using the private class `Om_Message`. 
Tables

Unbounded Occurring Elements in XSD

Neither XSD nor Java has an explicit table type. However, in XSD you can define elements so that there is no limit to how often they may occur (also see: Elements and Attributes [Seite 135]). For example, assume you require a data type staff, which you can use to select a group of colleagues as members of staff and assign them a manager. The following table reflects the definition of this data type in the XSD editor:

Definition of an Unbounded XSD Element Within a Structure

<table>
<thead>
<tr>
<th>Structure</th>
<th>Category</th>
<th>Type</th>
<th>Occurrence</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>staff</td>
<td>Complex Type</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[-] manager</td>
<td>Element</td>
<td></td>
<td>1</td>
<td>staff manager</td>
</tr>
<tr>
<td>name</td>
<td>Element</td>
<td>xsd:string</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>birthday</td>
<td>Element</td>
<td>xsd:date</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>[-] employee</td>
<td>Element</td>
<td>1..unbounded</td>
<td>staff members</td>
<td></td>
</tr>
<tr>
<td>name</td>
<td>Element</td>
<td>xsd:string</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>birthday</td>
<td>Element</td>
<td>xsd:date</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

According to this definition, the element employee can occur more than once in the corresponding XML schema.

Conversion to Java Classes

Java proxy generation generates the class Staff_Type for the data type staff. This class can be reused just like the data type in the Integration Repository. The class provides two alternatives for accessing the values of the element employee that can occur infinitely in the message:

- The method `getEmployee()` returns an array type `Staff_Type.Employee_Type[]`
- The method `get_as_listEmployee()` returns an object of type `Staff_Type.Employee_List` for accessing the values with list methods.

The class Staff_Type contains the inner class Staff_Type.Employee_List for accessing with list methods as well as the inner classes Staff_Type.Employee_Type and Staff_Type.Manager_Type.

`StaffType.EmployeeList` implements the Java interface `java.util.List`, so that its general methods can be used (for example, the method `size()` determines the number of elements in the list).

Class Hierarchy with Public Methods in UML Notation
Since the elements manager and employee are defined locally in the data type staff, the Java proxy generation functions generate inner classes. If you reference reusable data types for these elements, the Java proxy generation functions also generate global classes. (However, the inner classes can also be reused).

### Enumerations

An enumeration is a [facet](#) that you can use to restrict the value range of a simple data type, attribute, or element. Java proxy generation generates a Java class for accessing the values. This class provides a type and value secure mapping of the enumeration by generating constants for each value and only enabling access using methods:

- There are two constants for each value that, where possible, have the same name as the value itself (for example, in the case of string values). Otherwise, the constants have the names value1, value2, ..., valueN (with or without an underscore):
  - A `value constant` of the type of the simple data type, attribute, or element. The constant therefore has the value of a possible value in the enumeration. In
addition, the constant name has an underscore as a prefix (for example, 
_value1).

- A object constant of the type of the inner class. An object of the inner class is 
  instantiated for the constant and the value constant is set for an attribute of the 
  object.

- The internal class provides two methods for accessing the object constant or the value 
  of the value constant:

  - Both fromValue(<enumeration value>) and fromString(<enumeration 
    value (as a string)> ) return the object constant of an enumeration. 
  fromValue() expects the value constant and fromString() expects the 
  value in its string representation. If the values are incorrect, the respective 
  method throws the runtime exception java.lang.IllegalArgumentException.

  - getValue() and toString() get the value for an object constant.

The enumeration classes are based on the JAX RPC specification that you can 
download from [http://java.sun.com/xml/downloads/jaxrpc.html](http://java.sun.com/xml/downloads/jaxrpc.html) (see chapter 4.2.4 
at this location).

**Example**

The element fruit is restricted to the value range "apple pear banana plum orange". 
The element fruit is part of the data type Grocery. Java proxy generation generates a 
global class Grocery_Type with the inner class Grocery_Type.Fruit_Type. Within this 
inner class it also generates the following:

- A string constant for each value of the enumeration, for example 
  public static final java.lang.String _apple = "apple";

- A object constant for each value of the enumeration, for example 
  public static final Fruit_Type apple = new Fruit_Type(_apple);

If you instantiate the object g for the class Grocery_Type, the methods for the enumeration 
fetch the following values:

<table>
<thead>
<tr>
<th>Method Call</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>g.Fruit_Type.fromValue(_orange)</td>
<td>Object constant orange of type Fruit_Type</td>
</tr>
<tr>
<td>g.Fruit_Type.fromString(&quot;orange&quot;)</td>
<td>Object constant orange of type Fruit_Type</td>
</tr>
<tr>
<td>orange.getValue()</td>
<td>&quot;orange&quot;</td>
</tr>
<tr>
<td>orange.toString()</td>
<td>&quot;orange&quot;</td>
</tr>
</tbody>
</table>
**Naming Java Proxy Objects**

**Use**

Compared to ABAP, Java has fewer restrictions regarding names for labels (for example, for classes or data types). This documentation deals with cases where proxy generation cannot copy names without making adjustments.

**Features**

The length of labels in Java is practically unlimited and they are case-sensitive. The technical names of the generated Java proxy classes are based on the JAX RPC specification:

- Namespaces are mapped to packages.
- Classes for interfaces have the suffix `_PortType`.
- Classes for data types have the suffix `_Type`.

The suffixes correspond to the element names from the imported WSDL document and help to avoid naming collisions. For this reason, naming conflicts are very unlikely. If a naming collision occurs, proxy generation terminates and triggers an error message.

The following conversions are made during generation:

<table>
<thead>
<tr>
<th>Conversion</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>In class names the first letter is changed to upper case.</td>
<td>Mapping of names based on Java conventions and the JAX Rpc specification (<a href="http://java.sun.com/xml/downloads/jaxrpc.html">http://java.sun.com/xml/downloads/jaxrpc.html</a>). These mappings should avoid naming conflicts.</td>
</tr>
<tr>
<td>Java key words are preceded by an underscore (for example, <code>'_private'</code>).</td>
<td></td>
</tr>
<tr>
<td>The system attaches the WSDL tags <code>_PortType</code>, <code>_Type</code> and <code>_Message</code> to the corresponding label name.</td>
<td></td>
</tr>
<tr>
<td>The Java package name is derived from the namespace that the object is assigned to.</td>
<td></td>
</tr>
</tbody>
</table>

Some operating systems have fixed conventions for file names. In Windows, for example, upper case characters within a file name are replaced by lower case characters. Therefore, the file name `FlightBooking` would become `Flightbooking`.

**Attachment: Mapping XSD to Java**

<table>
<thead>
<tr>
<th>ABAP</th>
<th>Java</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Seite 197]</td>
<td></td>
</tr>
</tbody>
</table>

Built-In XSD Data Types and Their Exact Equivalent in ABAP

<table>
<thead>
<tr>
<th>XSD</th>
<th>Java</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>java.lang.String</td>
</tr>
</tbody>
</table>
No global types are generated for simple data types (tag `<simpleType>`), because they only reference built-in data types.

### Tags Supported for Complex Data Types (in Alphabetical Order)

<table>
<thead>
<tr>
<th>XSD</th>
<th>Java</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;attribute&gt; (local)</code></td>
<td>Attributes of a class (including access methods)</td>
</tr>
<tr>
<td><code>&lt;complexType&gt; (global)</code></td>
<td>Class</td>
</tr>
<tr>
<td><code>&lt;complexType&gt; (local)</code></td>
<td>Inner class</td>
</tr>
<tr>
<td><code>&lt;element maxOccurs=&quot;unbounded&quot;&gt;</code></td>
<td>• Attributes of an (inner) class (including access methods)</td>
</tr>
<tr>
<td></td>
<td>• Inner table class</td>
</tr>
<tr>
<td><code>&lt;element&gt; (local)</code></td>
<td>Attribute of an (inner) class (including access methods)</td>
</tr>
</tbody>
</table>

Only those tags permitted by the XSD editor [Seite 133] are executed.

### Restrictions for XSD Elements

<table>
<thead>
<tr>
<th>XSD Element</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;extension&gt;</code> and abstract types</td>
<td>These elements cannot be converted.</td>
</tr>
<tr>
<td>substitutionGroup and <code>&lt;element abstract=&quot;true&quot;&gt;</code></td>
<td></td>
</tr>
<tr>
<td><code>&lt;redefine&gt;</code></td>
<td></td>
</tr>
<tr>
<td>Restriction of value area (&lt;pattern&gt;, &lt;enumeration&gt;, &lt;maxInclusive&gt;, &lt;maxExclusive&gt;, etc.)</td>
<td>These tags are ignored by the proxy runtime. The schema validation checks these properties.</td>
</tr>
<tr>
<td>Key references, uniqueness (&lt;key&gt;, &lt;keyref&gt;, &lt;unique&gt;, and so on.)</td>
<td></td>
</tr>
</tbody>
</table>
Configuration

Purpose
You use the Integration Builder [Seite 27] to configure business processes that are based on the cross-component exchange of messages. You can reproduce the “choreography of the messages” that you have defined in a business scenario [Seite 78] in the productive system landscape and define the message flow between the participating business systems. This enables you to gather together all the information in the Integration Directory that is required to process messages at runtime [Seite 264].

Prerequisites
To be able to access the information required for configuration, note the following:

- Ensure that your system landscape is described in the System Landscape Directory.
- Ensure that the relevant business scenarios, interfaces, and mappings are defined in the Integration Repository.

Process Flow
The following steps are usually carried out in the configuration phase:

1. Ideally, you have already defined a business scenario at design time and have therefore outlined all the cross-component communication steps of your business process where messages are exchanged (see Designing Business Scenarios [Seite 77]).

2. You then create a business scenario in the Integration Directory [Seite 208] to use as a basis for all other configuration data.

3. In the configuration of the receiver determination [Seite 214], the possible receivers that the messages from particular outbound interfaces are to be forwarded to are defined for the senders involved. You also have the option of specifying further conditions in the receiver determination regarding the forwarding of the message to the receiver.

4. Within an Interface Determination, the interfaces on the sender and receiver sides are assigned to each other. Here you can also define which interface mappings from the Integration Repository are to be used (see Definition of Interface Determinations [Seite 226]).

5. For technical receiver determination to take place, you must specify technical access data about the receiver systems, for example, the physical address. You define the information required here when you define end points [Seite 231]. Different end point types are used depending on whether adapters are implemented for message processing, and if so, which particular adapters.

6. You can also specify logon data [Seite 247] for receiver systems to keep control of system access at the message receiver. Whether logon data is applied, and if so, how, depends on the end point type.

💡
You can define the configuration objects for a particular combination of key attributes (business scenario, sender and receiver system, sender and receiver interface). You also have the option of using a wildcard character (*) when creating interface determinations, end points, and logon data. This enables you to specify a very general definition range for these configuration objects (see Generic/Specific Definition of Configuration Objects [Seite 250]).
Business Scenarios in the Integration Directory

Purpose

Business scenarios in the Integration Directory form the basis for the configuration of logical and technical routing. Starting with a business scenario in the Integration Repository and the system landscape on which it is based, enter all information in the Integration Directory that is required for configuration (see Reference to Business Scenarios in the Integration Repository).

In the Integration Directory, receiver determinations are grouped within a business scenario. All other configuration objects are not created for a specific business scenario.

Features

A business scenario has the following functions in the Integration Directory:

- A business scenario forms the basis for further configuration:
  - The configuration overview provides you with an overview of all configuration objects relevant for the business scenario (see Displaying/Changing a Business Scenario in the Integration Directory).
  - The configuration assistant helps you to gather together all information required for the configuration of the connection in a business scenario.

- You can import and export business scenarios (see Exporting and Importing Integration Objects). For example, this enables you to transfer all receiver determinations (together with interface determinations, if required) belonging to a business scenario as a whole, from a test environment to a productive environment.

  For this purpose, groups of business systems can be defined in the System Landscape Directory in which the business systems that have been pre-defined for different areas of use (for example, testing and production operation) are grouped together (see System Landscape Directory in SAP Exchange Infrastructure).

  To ensure configuration content can be imported and exported without any problems, in the System Landscape Directory, you must define (prior to import) which business systems correspond to each other in the various business system groups (see Configuring Groups and Transport Targets).

Restrictions

To be able to begin with configuration in the Integration Directory you must first manually create a business scenario there (see Creating a New Object).
Displaying/Changing Business Scenarios in the Integration Directory

Use
Business scenarios [Extern] form the basis for configuration.

Prerequisites
You have created [Seite 40] a business scenario in the Integration Directory or have opened [Seite 35] an existing business scenario in the Integration Directory.

Specify the name of the business scenario when you create it. Note that the name must be unique and not language-specific. Blanks and special characters are not permitted in the name of a design object. We recommend that you keep to the naming conventions that are also recommended for design objects (see Creating a New Object [Seite 40]).

Features
The Display/Change Business Scenario editor has the following functions:

Configuration Assistant
The configuration assistant [Seite 210] (Configuration Assistant) enables you to configure a connection in a business scenario.

Create a Receiver Determination
You can add an additional receiver determination [Seite 214] to the business scenario (Receiver Determination).

Configuration Overview and Creating Additional Configuration Objects
The lower part of the editor contains an overview of additional configuration objects. The objects are grouped in a table according to sender/receiver pairs. You can use the pushbuttons in the table header to create new configuration objects.

The table contains the following columns:

- **Sender -> Receiver Column** (sub-divided into Outbound Interface):
  This column displays all sender/receiver pairs for which configuration objects are defined. Expand the overview further (icon) to display all outbound interfaces for which receiver interfaces are defined. Double click an outbound interface to open the receiver determination [Seite 216].

- **Interface Determination Column** (sub-divided into the columns Inbound Interface and Interface Mapping):
  If an interface determination is defined for the sender/receiver pair and the outbound interface, then the name of the inbound interface and the name of the interface mapping (if assigned) is displayed. Double click a cell in one of the two columns to open the interface determination [Seite 228].

- **End Points Column**:
  If an end point is defined for the receiver of the sender/receiver pair, then the end point type is displayed in this column. Double click the cell to open the end point [Seite 233].

- **Logon Data Column**: 

...
If logon data is defined for the receiver of the sender/receiver pair, then the logon procedure is displayed in this column. Double click the cell to open the logon data [Seite 248].

If you position the cursor on a cell in one of the columns for an interface determination, an end point, or logon data, the values of the key attributes for the relevant configuration object are displayed in a quick info box (see Creating a New Object [Seite 40]).

If you have modified the Business Scenario, save it first. Only then refresh the configuration object by choosing (Refresh).

To create further configuration objects, choose the pushbuttons Interface Determination, End Point and Logon Data (). Using the dropdown list box, select whether you want to define the object for the selection of key attributes predefined by the sender/receiver pair or for a different selection of key attributes.

• If you choose Other Selection, the system calls the Creating a New Object [Seite 40] dialog for the relevant configuration object. There you define the key attributes of the configuration object. Only then does the system navigate to the relevant object editor.

• If you select the predefined selection of key attributes, the system navigates directly to the relevant object editors [Seite 35].

If you want to define an interface determination, select whether you want to define it for all business systems, for the sender/receiver pair (specified by the cursor position), or for another selection.

If the configuration object concerned is already defined for a specific combination of key attributes (business systems or interfaces, or both), then you cannot create it generically for all business systems or interfaces (for more information see Generically/Specifically Defining Configuration Objects [Seite 250]).

Configuration Assistant

Use
The configuration assistant supports you when configuring a connection in a business scenario.

Integration
All the information that describes a communication step between two different application components is stored in a cross-component connection in a business scenario [Seite 78] from the Integration Repository. This includes the outbound and inbound interface, and the interface mapping used.

This design time information must be mapped to a specific system landscape during configuration.

Therefore, the business systems used and the interfaces on both sender and receiver sides must be specified to complete the configuration of a connection. Furthermore, the end points
and (optional) logon data must also be known to the receiver system for technical routing to 
be performed.

The configuration assistant helps you to complete all this information in a step-by-step 
procedure.

The configuration assistant enables you to reuse configuration objects that already exist, or 
create new ones, as required.

**Prerequisites**

You have decided on the following:

- Which business systems are to be used
- Which interfaces on the sender and receiver side, and which interface mapping are to 
  be used for the communication step

To call the configuration assistant, choose *Configurations Assistant* in the 
*Displaying/Changing a Business Scenario in the Integration Directory* editor.

**Activities**

Using the configuration assistant you can specify the following information in a series of 
steps:

1. Define Sender
   
   First, select a sender system. Next, select an outbound interface. To do so, call input 
   help.

2. Define Receiver
   
   Proceed as described in step 1.

3. Define Interface Mapping
   
   Using the radio button, specify whether an interface mapping is to be used for the 
   connection or not. If an interface mapping is to be used, select it here. To do so, call 
   input help.

   The configuration assistant proceeds differently depending on whether an interface 
   determination exists in the Integration Directory for the information that has already 
   been specified (sender, receiver, and interface mapping):

   - No interface determination exists
     
     A new generic interface determination is created based on the specifications 
     already made. This means that the interface determination is defined for all 
     business systems.

   - A (generic) interface determination exists
     
     A new specific interface determination is created based on the specifications 
     already made. This means that the interface determination only applies for the 
     sender/receiver pair that you specified with your previous selections.

   - A (specific) interface determination already exists that matches your selections 
     exactly
     
     This interface determination is used.

   ! Before creating generic and specific interface determinations, see the 
   information under *Generic/Specific Definition of Interface Determinations*.

   Before creating generic and specific interface determinations, see the 
   information under *Generic/Specific Definition of Interface Determinations*.

   Before creating generic and specific interface determinations, see the 
   information under *Generic/Specific Definition of Interface Determinations*.
4. Define End Point
   Define an end point for technical routing.

5. Define Logon Data
   Define logon data for technical routing.

   Use the radio buttons to specify whether logon data is to be used or not. The use of logon data is closely linked to the end point type used. Since no end point attributes have been defined when you first create an end point with the configuration assistant, you can decide whether you want to use logon data or not. However, note that in certain circumstances it may be possible to define the end point type from the information already specified for the receiver.

   For example, if an IDoc is select for the receiver interface, then the end point type End Point for IDoc Adapter is selected. No logon data is required for this kind of end point type (see End Point for IDoc Adapter). Therefore, on the Define Logon Data screen you do not have the option of using logon data.

6. Summary
   This screen lists all configuration objects that are to be created or reused based on the specifications you made with the configuration assistant. Choose Finish to confirm your specifications.

   Note that you can only create “empty” configuration objects when you initially create end points and logon data. To define end points and logon data you must complete additional data in the relevant object editors once the configuration assistant is finished (see Displaying/Changing End Points and Displaying/Changing Logon Data).

Reference to Business Scenarios in the Integration Repository

The application components implemented by the involved business partners are represented within a business scenario at design time. Furthermore, the connections in a business scenario contain all the information that describes the exchange of messages between application components.

Such a business scenario from the Integration Repository is an ideal starting point for configuring logical and technical routing.

Apart from those developed in the Integration Repository, you can also use other representations of business scenarios as a basis for configuration. An example would be component views of business scenarios shipped by SAP that are available on the SAP Service Marketplace as PowerPoint graphics.

Before you actually begin configuration in the Integration Directory, decide which business systems you want to assign to the application components of the business scenario.

The associations represented in a business scenario (Integration Repository) provide you with the following information that is important for the configuration:
• The application component overview shows the assignments between sender and receiver systems.

• Connections contain the assignments between sender and receiver interfaces as well as an overview of the interface mapping used.

These assignments are described in the Integration Directory by the receiver determination [Seite 214] and interface determination [Seite 226] configuration objects. Both configuration objects define the routing of messages.

At present, it is not possible to automatically transfer information stored in the Integration Repository to the Integration Directory.

The association described above is represented by the following graphic:
Definition of Receiver Determinations

Purpose
You use the configuration object receiver determination [Seite 214] to define the message flow in the system landscape. You also have the option here of specifying conditions to control the forwarding of the message between a sender system and one or more receiver systems.

All receiver determinations that belong together as part of a particular business context are grouped together in a Business Scenario [Extern] at configuration time.

Features
You can do the following with the Integration Builder at configuration time:

- Display/Change Business Scenarios in the Integration Directory [Seite 209]
- Display/Change Receiver Determinations [Seite 216]

You can also define optional conditions [Seite 219], within a receiver condition for forwarding messages to different receivers.

Receiver Determination

Definition
Receiver Determination
In a receiver determination, the possible receivers of messages from an outbound interface are defined for a sender. You also have the option of specifying the conditions that should apply when forwarding a message to the receiver(s) for a receiver determination.

There are two ways of determining the receiver:

- Receiver system(s) are specified explicitly Here you have the option of specifying a condition for forwarding the message to the receiver(s). This condition is analyzed at runtime.
- Receiver system is determined at runtime from the contents of the message (payload).

For more information, see Defining Receivers [Seite 218].

Condition
The distribution of the message to one or more possible receivers is controlled through a condition. A condition usually refers to the application data from the payload.

In the Business Scenario example Book a Flight Trip [Seite 81], the following is a possible condition: “Send message to receiver business system AIR_LH of airline Lufthansa to check flight availability if field Airline_Id in the payload of the message has the value LH (Lufthansa).”

Conditions are specified using the language XPath.

For more information, see Defining a Condition [Seite 219].
Example of a Receiver Determination

Example 1

In the Business Scenario Book a Flight Trip (BookFlightTrip), when finding flight information, a travel agency must determine the availability of flights by contacting the airlines involved, since this data cannot be kept up to date in the travel agency’s system. For this step, the Business Scenario features a synchronous message exchange between the application components at the travel agency and the airline(s) (see Example of a Business Scenario [Seite 81]).

The following systems are being used:

- Business system TRA at the travel agency
- Business system AIR_LH at Lufthansa
- Business system AIR_AA at American Airlines

To define the flow of messages that request flight availability from the airlines, the receiver business systems AIR_LH and AIR_AA are assigned to the sender system (TRA) with the outbound interface CheckAvailabilityOut in a receiver determination.

A condition for the forwarding of the message from the sender to the receiver then might look as follows: “Send the message of the outbound interface CheckAvailabilityOut to receiver system AIR_LH if field Airline_Id in the payload of the message has the value LH (Lufthansa). If this field has the value AA (American Airlines), then forward the message to AIR_AA.”

This example is shown in the graphic.

Example of a Receiver Determination

```
<table>
<thead>
<tr>
<th>Receiver</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIR_LH</td>
<td>//Airline_Id = LH</td>
</tr>
<tr>
<td>AIR_AA</td>
<td>//Airline_Id = AA</td>
</tr>
</tbody>
</table>
```
Displaying/Changing a Receiver Determination

Use
You can either display or change a receiver determination, as well as specify optional conditions for forwarding of the message to one or more receivers.

Prerequisites
You have created a new receiver determination or have opened an existing one.

Note the following when creating a receiver determination:

- To be able to create a receiver determination, a business scenario must be defined in the Integration Directory (see Displaying/Changing a Business Scenario in the Integration Directory).
- Specify the name of the Business Scenario as well as the sender system and sender interface (name and namespace) for a receiver determination. You can use search help here.

Features

Header
The editor header displays the following fields:

- Business Scenario
- Sender system
- Sender interface name
- Sender interface namespace

You cannot enter data in these fields, since the values were defined when the receiver determination was created.

The following information is also displayed:

- Information about the status of the receiver determination.
- Description
  Here you can enter a short description of the receiver determination.

Work Area

Receiver Determination
The receiver determination is displayed in this area in the form of a table. The possible receivers of the message are specified in the column Receivers and a condition is displayed in the column Condition.

💡 If the assignment of sender business system to receiver systems is tied to different conditions, then multiple rows are required for representing the receiver determination.

The following functions are available to you for editing and managing these tables:

Functions for Editing Tables

| Icon | Function |
Inserts a row below the selected row.

Deletes the selected row.

Moves the selected row up one place.

Moves the selected row down one place.

• In the column Receivers
  Calls a window which enables you to:
  o define whether you want to specify the receiver explicitly or whether it should be determined from the message contents
  o Select the receiver systems
  (See Defining Receivers [Seite 218]).

• In the column Condition
  Calls an editor to enable you to specify a condition, depending on whether the receiver is to be specified explicitly or whether it is determined from the contents of the message:
  o Conditions Editor [Seite 220] – when explicitly entering the receivers
  o Editor for Specifying the Extractor Parameters [Seite 223] – when determining a receiver from the contents of the message
  (See Defining a Condition [Seite 219]).

⚠️

To display the search help icon (.salary-time) in either column, position the cursor on the relevant input field while in change mode.

### Column Receivers

Here you can define the receivers of the message (see Defining Receivers [Seite 218]).

### Column Condition

Here you can define optional conditions for forwarding messages to the specified receivers (see Defining a Condition [Seite 219]).

### Configuration Overview

The lower part of the editor contains a table with an overview of all additional configuration objects that are defined for the receiver involved in the receiver determination. This table contains the following columns:

• **Sender Column** (sub-divided into **Receiver**):
  Displays the sender and outbound interface of the receiver determination. If you expand the overview further (salary-time), all the receivers are displayed that are assigned to the sender within the receiver determination.

The configuration objects that are defined for the receiver are displayed in the remaining columns.

• **Interface Determination Column** (sub-divided into the columns **Inbound Interface** and **Interface Mapping**):
  If an interface determination has been defined, then the inbound interface for this (only the name) and the interface mapping (provided it has been assigned) is displayed.
Double click a cell in one of the two columns to open the interface determination [Seite 228].

- **Column End Points**
  If an end point is defined, then the corresponding end point type is displayed in the table line. Double click the cell to open the end point [Seite 233].

- **Logon Data Column**
  If logon data is defined, then the logon procedure is displayed in the table cell. Double click the cell to open the logon data [Seite 248].

  If you position the cursor on a cell within the columns for the interface determination, the end point, or the logon data, then the values of the key attributes for the relevant configuration object are displayed in a quick info box (see Creating a New Object [Seite 40]).

  If you have modified the receiver determination, save it first. Only then refresh the configuration object by choosing (Refresh).

To create further configuration objects, choose the pushbuttons **Interface Determination, End Point** and **Logon Data**. Using the dropdown list box, select whether you want to define the object for the selection of key attributes predefined by the selected line or for a different selection.

- If you choose **Other Selection**, the system calls the Creating a New Object [Seite 40] dialog for the relevant configuration object. There you define the key attributes of the configuration object. Only then does the system navigate to the relevant object editor.

- If you select the predefined selection of key attributes, the system navigates directly to the relevant object editors [Seite 35].

  If you want to define an interface determination, select whether you want to define it for all business systems, for the sender/receiver pair (specified by the cursor position), or for another selection.

  If the configuration object concerned is already defined for a specific combination of business systems or interfaces, or both, then you cannot create it generically for all business systems or interfaces (for more information see Generically/Specifically Defining Configuration Objects [Seite 250]).

---

**Define Receiver**

**Use**

You can define one or more receivers for the message here.

**Prerequisites**

You have opened a receiver determination.
Features

In the change receiver determination [Seite 216] editor, the Receiver column displays one or more receiver systems.

In change mode, here you can:

- Enter receiver systems manually

  When entering this information manually, make sure that you separate the entries using commas, and that no special characters can be used for system names (apart from underscores "_").

- Define receiver systems using Search Help [Seite 48]

  You can call the search help by choosing . In the Edit Receiver dialog, you can define how the receiver of the message is to be determined.

  o Specifying the receiver explicitly from a list

    Here, the possible receivers are displayed in a list. You can use the check box to select one or more receiver systems.

    In the column Condition of the editor Change Receiver Determination, you have the option of specifying a condition for forwarding a message to these receivers. To do so, use the condition editor [Seite 220].

  o Identifying a receiver from the message contents

    Here the receiver of the message is determined at runtime from a particular payload element. Instead of system names, the payload usually contains business application data. However, a receiver system must be known for receiver determination. The corresponding element in the payload is therefore first mapped onto a business system using Value Mapping [Seite 53]. Only then is the receiver determination at runtime guaranteed.

    You can enter the information for the payload element selection and for the value mapping in the editor for specifying the extractor parameters [Seite 223].

    In many cases, the receiver is included in the message in the form of a supply partner or business partner ID. In cases such as this, this information is determined from the payload during receiver determination, and is mapped onto a business system by way of Value Mapping [Seite 53].

Defining a Condition

Use

Depending how the receiver of the message is to be determined (see Defining a Receiver [Seite 218]) there are two ways of formulating a condition.

- Specify the receiver system explicitly

  You have the option of specifying a condition for forwarding messages to the specified receiver systems (see Condition Editor [Seite 220]).

- Determine the receiver from the message contents
You can select an element from the payload and specify additional information for value mapping which you can then use to map the element to a receiver system (see Specifying the Extractor Parameters [Seite 223]).

To call the definition of the corresponding condition in the respective editor for both cases, choose in the column Condition.

**Condition Editor**

**Use**

If you specified the receiver systems explicitly, you can specify a condition for forwarding the message to these receivers.

You can specify a condition that refers to elements in the payload of the message. If the specified condition is fulfilled for these elements (the corresponding element has a certain value, for example), then the message is forwarded to the specified receiver systems.

Conditions are specified using the language XPath. The language XPath addresses parts of an XML document. A description of XPath would go beyond the scope of this documentation. See the W3C recommendation from November 16, 1999 (English/German/Other translations).

Three options are available for formulating a condition:

- **Type: Field value**
  You can select an element from the payload with an XPath expression and then compare the value of this element with a pre-defined value at runtime.
  
  You can use AND and OR operators to link multiple occurrences of these “equations” to a more complex condition.

  In a flight booking Business Scenario, a message contains the element Destination_From for the departure airport of a flight. An example of a possible condition would be: "If the element Destination_From has the value FRA (departure airport: Frankfurt am Main) or JFK (departing airport: Kennedy Airport New York), then forward the message to the business systems specified in the Receiver column." The text in the column Condition would then read: 
  
  
  (//Destination_From = FRA) OR (//Destination_From = JFK).

- **Type: Logical expression**
  If you are unable to use the field value method and want to make full use of XPath, you can enter a complex logical expression directly. If this expression is fulfilled, then the message is forwarded to the specified receivers.

- **Type: Context**
  The adapters contain information that may be relevant for receiver determination. In such cases, where IDoc outbound interfaces or RFC outbound interfaces are to be processed, in the condition you can specify context objects which you want to use to access this information.
  
  - IDoc
    This gives you the option of accessing selected fields of the IDoc control record (see Features).
Here you can specify the name of the RFC destination in the sender system (see Features).

Features
The condition editor comprises three areas.

Table for Defining the Condition
You define the condition in the upper area. You can enter a separate expression within a table row. You can then use AND and OR operators to combine several of these expressions to form any complex condition.

Functions for Editing Tables

<table>
<thead>
<tr>
<th>Icon</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insert Group: Inserts a new row for a new expression below the selected row. The new expression is linked to the expression in the selected row by the OR operator.</td>
<td></td>
</tr>
<tr>
<td>Insert Expression: Inserts a new row for a new expression below the selected row. The new expression is linked to the expression in the selected row by the AND operator.</td>
<td></td>
</tr>
</tbody>
</table>

Expressions that are linked through an AND operator are combined using a parenthesis symbol. The complete condition is displayed below the table in a preview window.

An expression (a line) in the table comprises an entry in the XPath column that is compared with a value in the Value column by using a relational operator. You select the relational operator from the dropdown list box in the Op column.

You can choose from the following relational operators:
- Equals (=)
- Not Equal To (≠)
- Contains Pattern (CP)Operator (∝)

Using this operator you can query a string with a pattern. You can use the following characters for this:
- + for any character
- * for any string

Field value
Here you can enter the XPath expression that you want to compare with the value from the database (for example, employmenttaxdeclaration).
Use the input help in the XPath column to display the structure of the payload and to select a node from this structure. The XPath expression that identifies this node is then written to the field in the XPath column and can then be edited there. Call the input help by choosing in the table cell. Note that the input help result alone does not represent a complete logical expression. Therefore, if you choose the expression type Logical Expression, then this must be completed manually.

Input help is only available if the interface is defined in the Integration Repository.

<table>
<thead>
<tr>
<th>Logical expression</th>
<th>To use to select the value of a payload element (for example: //Airline_Id).</th>
<th>between =, ≠, and ≈</th>
<th>value for the value of an element in the payload (that is selected by the expression in column XPath)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Context IDoc</td>
<td>Here you can enter a complex logical XPath expression. The entire expression is then compared with the value TRUE (in the column Value). This means: If the expression specified here is true, then the message is forwarded to the specified receivers.</td>
<td>= TRUE (The expression specified in column XPath is checked for TRUE.)</td>
<td></td>
</tr>
<tr>
<td>Context (RFC)</td>
<td>To identify the IDoc partner, you can access one or more of the following fields of the IDoc control record: RCVPRN, RCVPRT, RCVPFC, SNDPRN, SNDPRT, SNDPFC. Use the following syntax with the IDoc namespace (example for field RCVPRN): urn.sap-com:document:sap:idoc:messages.RCVPRN</td>
<td>Choose between =, ≠, and ≈</td>
<td>Comparison value for the value selected in column XPath</td>
</tr>
</tbody>
</table>

Use the input help in the XPath column to display the structure of the payload and to select a node from this structure. The XPath expression that identifies this node is then written to the field in the XPath column and can then be edited there. Call the input help by choosing in the table cell. Note that the input help result alone does not represent a complete logical expression. Therefore, if you choose the expression type Logical Expression, then this must be completed manually.

Input help is only available if the interface is defined in the Integration Repository.

**Preview Area**
This area displays the XPath expression that you defined in the upper area of the editor.

**Table for Defining Prefix/Namespace Pairs**
If the elements used in the XPath expression are specified by a namespace, you can assign them to a prefix. This prefix can be used to make the XPath expression more clear. On the tab page Namespaces, you can define prefix namespace pairs to specify the prefixes that are used in the XPath expression.

<table>
<thead>
<tr>
<th>Icon</th>
<th>Function</th>
</tr>
</thead>
</table>
Insert a Row Below the Selection: Inserts a new row for a new prefix/namespace pair below the selected row.

Delete Selected Lines: Deletes the selected row.

If you confirm the details you have entered in the condition editor by choosing ✔, then the selected condition is saved and Display/Change Receiver Determination is displayed in the editor in the Condition column.

**Specifying the Extractor Parameters**

**Use**

If the receiver of a message is not explicitly specified, rather is to be determined from the message contents at runtime, then you must enter the following information when configuring the receiver determination.

For the purpose of clarity, let us use an example.

Let us assume that in a flight booking Business Scenario, the receiver of a message from business partner Airline is to be determined. The airline is identified in the payload of the message by the element `Airline_Id`. The following condition should apply: If the business partner is the airline Lufthansa, that is, the `Airline_Id` element in the message has the value LH, then the message is to be forwarded to the business system `AIR_LH` of airline Lufthansa.

When configuring the receiver determination you must ensure the following:

1. The value of the payload element that identifies the business partner (Lufthansa) can be selected.
2. A receiver business system (the business system `AIR_LH`) can be derived from this concrete business partner (identified by the value LH in the payload).

For the second step, Value Mapping [Seite 53] is used in the way described below.

We start with two representations of the message receiver:

- Representing the receiver in the role of a “business partner”.
  This representation can be determined from the payload.
- Representing the receiver in the role of a “business system”.
  This representation is relevant for actually determining the receiver at runtime and must be determined from the first representation by means of value mapping.

Both representations must be defined in the value mapping table. The representation of the receiver as business partner (source representation) can then be mapped onto the representation of the receiver as business system (target representation).

In the dialog box Specifying the Extractor Parameters, enter the required information so that:

- The payload element that identifies the business partner can be selected.
- The link can be produced between the payload element and the representation of the business partner stored in the value mapping table.

The mapping of the business partner representation onto the business system is then ensured by the entries in the value mapping table.
Prerequisites

In the Change Receiver Determination editor, you have selected Identifying a receiver from the message contents under Defining Receivers [Seite 218].

Procedure

1. In the editor Change Receiver Determination [Seite 216], choose from the column Condition. The dialog box Specifying the Extractor Parameters is called.

2. In the dialog box Specifying the Extractor Parameters, enter the following information:
   - Field Extractor
     Here you must specify the element from the payload from which the receiver is to be determined. For this you must use an XPath expression (see Condition Editor [Seite 220]).

Under Field Type, enter the following information for the value mapping:
   - Field Namespace
     Here you must specify the namespace for the object type of the selected payload element. This value must be entered in the value mapping table for the representation of the receiver as business partner, in the column Namespace.
   - Field Name
     Here you must specify the object type for the selected payload element. For the value mapping, the object type describes the “role” that the receiver has in the message, that is, the role of business partner. For receiver determination, it is the constant value Business_Partner. This value must be entered in the value mapping table, in the column Object Type.

3. Confirm your specifications by choosing Confirm (✔).

Result

In the editor Display/Change Receiver Determination, the XPath expression for the selected payload field in the table cell under column Condition is displayed in the following form:

Receiver <namespace>.<Name> From <extractor>

Example

Here, the following is specified for the example given (flight booking Business Scenario, see above):

- Which value you must enter when specifying the extractor parameters
- What the corresponding entries in the value mapping table must look like (see graphic)

Connectivity Between Extractor Parameters and Value Mapping Table
In the value mapping table, two representations must be entered in the same group. Both representations describe the object “Message Receiver”.

- **Representation of business partner**
  
  The business partner is selected in the message using the element `Airline_Id`. The column `Value` must therefore contain the value that identifies the business partner in the message (in the example, *Lufthansa*: LH (for the field `Airline_Id`).

- **Representation of business system**
  
  Note the following specifications that you must make to display the receiver as a business system:
  
  - The constant value *IS* (Integration Server) must be entered in the `System` column
  - The constant value `http://sap.com/xi` must be entered in the `Namespace` column
  - The constant value `Business_System` must be entered in the `Object Type` column
  - The `Value` column must contain the name of the business system (AIR_LH) onto which the business partner Lufthansa is mapped.

When specifying the extractor parameters, enter the following information:

- **Extractor**: //Airline_Id
- **Namespace**: Example: `http://travelagency`
- **Name**: constant value: Business_Partner
Definition of Interface Determinations

Purpose
In an interface determination you define the following for the outbound interface of a sender:

- Receiver interface
- Mapping name and version

You must use interface mappings from the Integration Repository.

A message can be divided up and sent to multiple interfaces. In this case, you need to specify further conditions for splitting the message (see Interface Determination with Message Splitting [Seite 230]).

You can either define a cross-business system or a business system-specific interface determination for an outbound interface (see Generic/Specific Definition of Configuration Objects [Seite 250]).

For more information, see the interface determination [Seite 226] section in this documentation.

Features
You can do the following with the Integration Builder at configuration time:

- Display/Change Interface Determinations [Seite 228]
- Display an overview of interface determinations for an outbound interface [Seite 227]

Interface Determination

Definition
With an interface determination you assign the following to an outbound interface:

- An inbound interface
- The name and the version of an interface mapping [Seite 147]

Structure
The graphic below shows the structure of an interface determination for the outbound interface IF_1:

Example of an Interface Determination
In the example interface determination, the outbound interface IF_1 on the sender system BS_1 is assigned the inbound interface IF_2 on the receiver system BS_B.

Displaying Interface Determinations the Integration Builder Navigation Area

Interface determinations are grouped according to outbound interfaces in the Integration Builder navigation area [Seite 31].

Note the following:

- If exactly one interface determination is defined for an outbound interface, then the icon is displayed next to the outbound interface in the navigation tree. Choose this icon to navigate directly to the editor for displaying/changing interface determinations [Seite 228].

- If more than one interface determination is defined for an outbound interface, then the icon is displayed next to the outbound interface in the navigation tree. In this case, you first navigate to an overview table for all interface determinations (Overview of Interface Determinations [Seite 227]). Then you can navigate to the display/change editors for the individual interface determinations.

Display Overview of All Interface Determinations

Use

If more than one interface determination was defined for an outbound interface, they are shown in an overview.

In this case, the icon is displayed next to the outbound interface under Interface Determination for Outbound Interface in the Integration Builder navigation area. If only one interface determination is defined for an outbound interface, then the icon is displayed instead.

Prerequisites

To display the overview of interface determinations for an outbound interface, in the navigation area of the Integration Builder, position the cursor on the outbound interface and double click, or choose Open (.) from the context menu.

Features

The following information is displayed in a table for each interface determination:

- Sender system
- Receiver system
• Inbound interface (name and namespace)

**Functions for Editing/Displaying the Table**

<table>
<thead>
<tr>
<th>Icon</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Open Icon" /></td>
<td><strong>Open</strong>: Calls the Display/Change Interface Determinations [Seite 228] editor for the selected table line.</td>
</tr>
<tr>
<td><img src="image" alt="Create New Object Icon" /></td>
<td><strong>Create New Object</strong>: Calls the Create New Object [Seite 40] function. The objects type (Interface Determination) and the sender interface (name and namespace) have already been selected. The new interface determination is inserted in the table beneath the line selected.</td>
</tr>
<tr>
<td><img src="image" alt="Delete Object Icon" /></td>
<td><strong>Delete Object</strong>: Deletes the selected interface determination. It is then removed from the overview.</td>
</tr>
<tr>
<td><img src="image" alt="Update Icon" /></td>
<td><strong>Update</strong>: Updates the overview of interface determinations.</td>
</tr>
</tbody>
</table>

---

**Displaying/Changing Interface Determinations**

**Use**

You use this editor to display and change an interface determination for an outbound interface. Within an interface determination you assign an inbound interface of a receiver and optionally a mapping to the outbound interface.

The complicated scenario whereby a message is divided up and sent to multiple receiver interfaces is described in Interface Determination with Message Splitting [Seite 230].

**Prerequisites**

You have created [Seite 40] a new interface determination or have opened [Seite 35] an existing one.

When creating an interface determination you can specify the wildcard character (*) for either the sender system or the receiver system (see Generic/Specific Definition of Configuration Objects [Seite 250]).

You can also open an interface determination by navigating to the Display Overview of Interface Determinations [Seite 227] editor, positioning the cursor on an interface determination and choosing Open ( ).

**Features**

The editor is structured as follows:

**Header**

Contains the following fields:

• Sender system
• Sender (outbound) interface (name and namespace)
• Receiver system

You cannot change any entries in these fields since the values were defined when you created the interface determination.

The status of the interface determination is also displayed and you have the option of entering a short description.

**Work Area**

The interface determination is displayed in the form of a table.

Each table line represents an assignment of an outbound interface to an inbound interface. If the scenario occurs whereby a message is divided up and sent to multiple inbound interfaces, then multiple table lines are shown (see: Interface Determination with Message Splitting [Seite 230]).

The table has the following columns:

**Inbound Interface (Columns Name and Namespace)**

Displays the name and the namespace of the inbound interface of the receiver. You can enter an interface in change mode. You have the following options:

• Manual entry

If you enter an interface manually, the system does not check whether the interface is defined in the Integration Repository.

• Search help

To call the search help [Seite 48], choose the Search Help icon (🔍). The system displays a list of interfaces (including their name and namespace) that you can select from.

• Drag & Drop

Drag an interface from the navigation area to the hand icon (✍) using Drag & Drop [Seite 47].

If the receiver system is an SAP system, you can also use the RFC namespace.

**Condition**

This column is only displayed if more than one inbound interface is assigned (for more than one table line. See Interface Determination with Message Split [Seite 230]). The condition is displayed that controls the distribution of the message to the inbound interfaces. If you want to define a condition for forwarding messages to an inbound interface, select the relevant field in change mode. The search help icon (🔍) is displayed. Call the Condition Editor [Seite 220] by clicking the search help icon.

**Interface Mapping (Name, Namespace, and Software Component Version columns)**

Here you can select an interface mapping [Seite 147] from the Integration Repository. To do so, call the search help for interface mappings (search help icon (🔍) in the Software Component Version column).
Within an interface determination it is only possible to reference interface mappings that are saved in the Integration Repository. Design objects in the Integration Repository are release-dependent. To identify an interface mapping uniquely, you must specify the release from which the selected interface mapping is to be taken. Therefore, you must use search help [Seite 48] to specify an interface mapping and not enter one manually.

**Functions for Managing and Editing the Table**

(only visible in change mode)

<table>
<thead>
<tr>
<th>Icon</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Insert a Row Below the Selection" /></td>
<td><strong>Insert a Row Below the Selection</strong>&lt;br&gt;Inserts a new table line in which you can select an additional inbound interface and interface mapping for the interface determination.</td>
</tr>
<tr>
<td><img src="image" alt="Delete Selected Lines" /></td>
<td><strong>Delete Selected Lines</strong>&lt;br&gt;Moves the selected line upwards (or downwards). Only possible if at least two lines are in the table.</td>
</tr>
<tr>
<td><img src="image" alt="Inbound Interface" /></td>
<td>• In the <strong>Inbound Interface</strong> column&lt;br&gt;Calls search help to select the inbound interface.</td>
</tr>
<tr>
<td><img src="image" alt="Condition" /></td>
<td>• In the <strong>Condition</strong> column (for more than two lines)&lt;br&gt;Calls the conditions editor [Seite 220].</td>
</tr>
<tr>
<td><img src="image" alt="Interface Mapping" /></td>
<td>• In the <strong>Interface Mapping</strong> column&lt;br&gt;Calls search help for the interface mapping.</td>
</tr>
</tbody>
</table>

**Interface Determination with Message Split**

In certain instances it may be necessary to assign multiple inbound interfaces on the receiver side when defining an interface determination.

Example: A very large IDoc interface is being used on the outbound side at the sender. However, on the inbound side at the receiver, only small IDoc interfaces are available for processing the message. In this case, the message must be divided up and sent to multiple inbound interfaces.

You have the option here of formulating conditions for the distribution of the message to the various receiver interfaces. An interface splitting of this type is indicated in the Display/Change Interface Determination editor by multiple table lines.

An additional **Condition** column is shown in the table that displays the condition for forwarding the message to the corresponding interface. In change mode you...
can choose the search help icon ( ) to call the conditions editor [Seite 220].

To display the search help icon, click the input field for the condition.

The graphic below illustrates an interface determination that has been assigned multiple receiver interfaces.

**Example of an Interface Determination with Message Split**

![Diagram of Interface Determination with Message Split]

<table>
<thead>
<tr>
<th>Editor: Display Interface Determination</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inbound Interface</td>
</tr>
<tr>
<td>-------------------</td>
</tr>
<tr>
<td>1 IF_a1</td>
</tr>
<tr>
<td>2 IF_a2</td>
</tr>
</tbody>
</table>

**Defining End Points**

An end point represents a technical point of access to a system. End points are defined at configuration time to specify the technical call information for the business systems implemented in the system landscape. This information is located in the Integration Directory and is used at runtime to map logical receivers (business systems) to physical addresses during technical routing.

End points are defined for the business systems that receive messages.

Different end point types [Seite 234] must be specified depending on whether adapters [Seite 340] are implemented on the receiver side for message processing, and if so, which particular adapters.

For a complete description of the technical access to the system, you can also specify logon data [Seite 247] when using particular end point types. Logon data is created and edited independently of end points to ensure that collaborative processes are as flexible and reusable as possible during configuration.
Integration

Logical business systems and technical systems in the system landscape are described and managed in the System Landscape Directory [Seite 442]. Logical systems are used when configuring a collaborative process in order to map the message flow within the system landscape. Technical systems in turn specify access points (protocol, host, port), with which a system can be accessed.

By separating logical and technical systems you can make changes to the technical settings without altering the configuration of the entire process.

Access information for technical systems (end points) is saved in the Integration Directory. The required information can be imported from the System Landscape Directory at configuration time, provided it is located there. This information is used to automatically identify end points for a receiver system and is used to provide default values when creating end points.

Features

End points are defined as configuration objects at configuration time. The Integration Builder has a variety of tools for creating, editing, and displaying end points. Use these tools to perform the following tasks:

- Display an overview of end points [Seite 232]
- Display/change end points [Seite 233] for an individual receiver

End points can also be identified automatically from the System Landscape Directory. For more information, see Identifying End Points Automatically [Seite 245].

You can define an end point as a Web Service to call interfaces from receiver systems from any platform (see Define End Point as Web Service [Seite 246]).

You can specify one or more end points for a receiver business system.

Also see:

- Technical System Landscape [Seite 256]
- Configuring a Business System [Seite 261]

Displaying an Overview of End Points

Use

If you have defined multiple end points for a receiver system you can display them all in an overview.

In this case, the icon 🖼 is displayed next to the receiver under Receiver System End Points beside the name of the receiver business system in the Integration Builder navigation area. If only one end point is defined for the receiver system, the 🖼 icon is displayed.

Prerequisites

To display the end point overview, in the navigation area of the Integration Builder position the cursor on the receiver and double click, or choose Open (odon) from the context menu.
**Features**

The end point overview header displays the receiver system. The end points are displayed underneath in a table with the following information:

- Inbound interface name
- Inbound interface namespace
- End Point Type

**Functions in the Toolbar Above the Table**

<table>
<thead>
<tr>
<th>Icon</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>📝</td>
<td>Open: Calls the Display/Change End Point [Seite 233] editor for the selected table line</td>
</tr>
<tr>
<td>🗑</td>
<td>Create New Object: Calls the Create New Object [Seite 40] function</td>
</tr>
<tr>
<td>🗑</td>
<td>The fields for the object type (end point) and the receiver business system are already defined in the Create New Object dialog. Once you save, the new end point is included in the overview.</td>
</tr>
<tr>
<td>🗑</td>
<td>Delete Object: Deletes the selected end point. The deleted end point is removed from the overview.</td>
</tr>
<tr>
<td>🔄</td>
<td>Update Data: Updates the end point overview.</td>
</tr>
</tbody>
</table>

**Displaying/Changing End Points**

**Use**

You have the option of displaying or changing an end point.

**Prerequisites**

You have created [Seite 40] a new end point or have opened [Seite 35] an existing one.

Note the following when creating end points:

- You can specify the wildcard character (*) for the receiver interface (name and namespace) (see Default Procedure [Extern]).
- If the business system is an SAP system, you can also use the RFC namespace.
- In the Create New Object dialog, select the receiver business system first. Next, select the receiver interface. In both cases you can also call search help [Seite 48]. Search help only displays those interfaces that are available in the receiver business system.

You can also open an end point by selecting one in the overview of end points [Seite 232] and choosing Open (📝) or double clicking.

**Features**

**Header**

The editor header displays the following fields:

- Receiver system
- Receiver interface (name and namespace)

You cannot change any entries in these fields since the values were defined when you created the end point.

Fields for the sender business system and sender interface (name and namespace) are also shown. You cannot make any entries in these fields and they are pre-defined with the wildcard character (*). For reasons of consistency, this information is displayed in the header just like in the other editors. However, you do not make any specifications about the sender when you create an end point. The fields referring to the sender cannot be changed since the end point for the specified receiver applies, regardless of the sender information.

For more information, see Default Procedure [Extren].

The editor header also displays the status of the objects and you have the option of entering a brief description of the end point.

**Work Area**

Here you specify the attributes for the end point. The type of end point determines which attributes you must specify here.

See End Point Types [Seite 234] for more information.

**Activities**

To edit an end point, select the end point type from the dropdown list box in the work area of the editor. Then enter information about the end point, as described in End Point Types [Seite 234].

⚠️ If the RFC namespace (urn:sap-com:document:sap:rfc:functions) or the IDoc namespace (urn:sap-com:document:sap:idoc:messages) is already specified in the editor header, then the respective end point type (in other words end point for an RFC adapter [Seite 236] or end point for an IDoc adapter [Seite 235]) is pre-defined in the work area and cannot be changed.

**Example**

See End Point Types [Seite 234] for examples.

**End Point Types**

**Definition**

The following end point types exist:

- End Point for the IDoc Adapter [Seite 235]
- End Point for the RFC Adapter [Seite 236]
- End Point for the Integration Engine [Seite 238]
- End Point for the HTTP Adapter [Seite 240]
- End Point for Other XI Connections (XI Connectivity) [Seite 243]
- End Point for the Integration Engine for Acknowledgments [Seite 244]

The end point type determines the attributes that you must specify when editing an end point.
End Point for the IDoc Adapter

Definition
An end point of this type is an access point for the IDoc adapter [Seite 340]. It is required by the outbound IDoc adapter for each logical receiver to which an IDoc is sent so that a physical address can be determined from the logical receiver.

The following parameters must be specified for the IDoc adapter for this end point type.

<table>
<thead>
<tr>
<th>RFC Destination</th>
<th>RFC Destination on Integration Server for Receiver System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Segmentversion</td>
<td>Segment definition for an SAP release</td>
</tr>
<tr>
<td></td>
<td>If this field contains a release, the segment definition that was released for the specified release is used for each segment type from the current IDoc. Since the release is fixed, the same segment definition is still used after an upgrade.</td>
</tr>
<tr>
<td>Interfaceversion</td>
<td>Technical IDoc interface to be used</td>
</tr>
<tr>
<td></td>
<td>Possible values:</td>
</tr>
<tr>
<td></td>
<td>• SAP Release 3.0/3.1</td>
</tr>
<tr>
<td></td>
<td>Function module INBOUND_IDOC_PROCESS is used</td>
</tr>
<tr>
<td>Port</td>
<td>Receiver port of an IDoc transfer (adheres to the naming conventions of IDoc ports: SAP&lt;System-Id&gt;, for example SAPBCE)</td>
</tr>
<tr>
<td>SAP Release</td>
<td>Version of the receiver system</td>
</tr>
</tbody>
</table>

Integration
When you use an end point for an IDoc adapter, the logon data [Seite 247] is already maintained by the RFC destination.

You can create end points for the IDoc adapter for both SAP and non-SAP systems acting as receiver systems.

Example
See: Example of an End Point (IDoc Adapter) [Seite 235].

Example of an End Point (IDoc Adapter)
The end point is used in a business scenario in which a flight is booked using IDoc communication.

SAP system TAG (travel agency) is to book a flight in SAP system AIR (airline) by sending the IDoc FLIGHTBK.FLIGHTBK01.
End Point

Header Data

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Receiver business system</td>
<td>AIR</td>
</tr>
<tr>
<td>Receiver interface name</td>
<td>FLIGHTBK.FLIGHTBK01</td>
</tr>
<tr>
<td>Receiver interface namespace</td>
<td>urn:sap-com:document:sap:idoc:messages</td>
</tr>
<tr>
<td>Description</td>
<td>Book a Flight</td>
</tr>
</tbody>
</table>

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>RFC Destination</td>
<td>DESTAIR</td>
</tr>
<tr>
<td>Segversion</td>
<td></td>
</tr>
<tr>
<td>Interfaceversion</td>
<td>SAP Release 4.0 or higher</td>
</tr>
<tr>
<td>Port</td>
<td>SAPAIR</td>
</tr>
<tr>
<td>SAP Release</td>
<td>620</td>
</tr>
</tbody>
</table>

Logon Data

Logon data is defined within the RFC destination DESTAIR on the Integration Server.

End Point for the RFC Adapter

Definition

This end point is required by the outbound RFC adapter to call a function module in a receiver system. For more information about the RFC adapter, see RFC Adapter [Seite 349].

The following parameters must be specified in the end point for the RFC adapter. Select a radio button to activate or deactivate load balancing. You must specify different information depending on whether you activate or deactivate load balancing. This also determines which fields require entries to be made or can be changed.

Load Balancing Deactivated

<table>
<thead>
<tr>
<th>Application Server</th>
<th>Host name of the application server</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Number</td>
<td>SAP system number on this application server</td>
</tr>
</tbody>
</table>

Load Balancing Activated

<table>
<thead>
<tr>
<th>Message Server</th>
<th>Host name of the message server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message Server Service</td>
<td>OS service of the message server or the message server port</td>
</tr>
</tbody>
</table>

⚠️ Note that the RFC environment on the computer on which the RFC adapter is installed must be correctly set up.

<table>
<thead>
<tr>
<th>System ID</th>
<th>ID of the SAP system</th>
</tr>
</thead>
</table>
Logon Group
Logon group for the dynamic logon load for RFC calls

Integration

If you are using an end point of this type you must use logon data [Seite 247] for the receiver system. If this logon data is only to be used for this end point type, it must be specified in the namespace `urn:sap-com:document:sap:rfc:functions`. Generally, you can use the wildcard character (*) to define the receiver interface when defining logon data so that this logon data does not just apply for this end point type (see Generic/Specific Definition of Configuration Objects [Seite 250] and Example [Seite 237]).

You can only create end points for the RFC adapter for SAP systems acting as receiver systems.

Example

See: Example of an End Point (RFC Adapter) [Seite 237].

Example of an End Point (RFC Adapter)

The end point is used in a Business Scenario in which the availability of flights is checked by using a function module call.

SAP system TAG (travel agency) checks the availability in SAP system AIR (airline) by calling the module `CHECK_FLIGHT_AVAILABILITY_IN`.

End Points

Header Data

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Receiver business system</td>
<td>AIR</td>
</tr>
<tr>
<td>Receiver interface name</td>
<td>CHECK_FLIGHT_AVAILABILITY_IN</td>
</tr>
<tr>
<td>Receiver interface namespace</td>
<td><code>urn:sap-com:document:sap:rfc:functions</code></td>
</tr>
<tr>
<td>Description</td>
<td>Check Flight Availability</td>
</tr>
</tbody>
</table>

The calls are made to system AIR by using load balancing. The radio button `Load Balancing Activated` is selected correspondingly.

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message Server</td>
<td>airmain</td>
</tr>
<tr>
<td>Message Server Service</td>
<td>3642</td>
</tr>
<tr>
<td>System ID</td>
<td>AIR</td>
</tr>
<tr>
<td>Logon Group</td>
<td>PUBLIC</td>
</tr>
</tbody>
</table>

Logon Data

All RFC calls from system TAG arrive at the receiver with the same SNC logon data.

Header Data
### End Point for the Integration Engine

**Definition**

An end point of this type leads to an Integration Engine [Seite 23] in a receiver system. This is identified by a URL with defined query parameters.

**Use**

Choose the addressing mode you want by selecting the appropriate radio button. You can choose between two addressing modes:

- **URL Addressing**
  
  In this case you must specify a URL in the input field. The URL is structured as follows:
  
  ```
  http(s)://<host>:<port><path>?type=receiver
  ```

  Note the following:
  
  - `<host>` is the host name of the receiver system
  - `<port>` is the HTTP port
  - `<path>` is the ICF service for the Integration Engine in the receiver system (for example, `/sap/xi/engine`).

---

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sender business system</td>
<td>TAG</td>
</tr>
<tr>
<td>Sender interface name</td>
<td>*</td>
</tr>
<tr>
<td>Sender interface namespace</td>
<td>*</td>
</tr>
<tr>
<td>Receiver business system</td>
<td>AIR</td>
</tr>
<tr>
<td>Receiver interface name</td>
<td>*</td>
</tr>
<tr>
<td>Receiver interface namespace</td>
<td>*</td>
</tr>
<tr>
<td>Description</td>
<td>*</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authentication Procedure</td>
<td>Secure Network Connection for RFC</td>
</tr>
<tr>
<td>User Name</td>
<td>FLIGHTCHK</td>
</tr>
<tr>
<td>SAP Client</td>
<td>100</td>
</tr>
<tr>
<td>Logon Language</td>
<td>English</td>
</tr>
<tr>
<td>Security Level</td>
<td>Level 4 – “Maximum Protection of IS”</td>
</tr>
<tr>
<td>Partner Name</td>
<td>TAG</td>
</tr>
</tbody>
</table>

**Also see:**

- Defining Logon Data [Seite 247]
- For further information about using the wildcard character (*), see Generic/Specific Definition of Configuration Objects [Seite 250].
When you use an end point for an Integration Engine (and URL address mode) there are two ways that you can log on to the receiver system. Select the way you want to log on by using the radio buttons.

- **Anonymous Logon**
  In this case, the receiver does not authenticate the incoming message. In order for the message to be received, logon data must be saved in the receiver system for the ICF service (for example, `/sap/xi/engine`).

- **Apply Logon Data**
  In this case, logon data is checked when a message arrives. You specify this data in Logon Data [Seite 247].

- **HTTP Destination**
  In this case, the RFC destination is a type H destination (HTTP connection to an R/3 system) that is defined on the Integration Server. Logon data is already defined in the destination.

You can only create end points for an Integration Engine for SAP systems acting as receiver systems.

**Example**

See: Example of an End Point (Integration Engine) [Seite 239].

---

**Example of an End Point (Integration Engine)**

The end point is used in a business scenario in which a flight is booked using a proxy call in the receiver system.

A flight is to be booked in SAP system AIR2 (airline 2) by calling the proxy interface `BookFlightIn` in the namespace `http://ai2.com/FlightBooking` from SAP system TAG (travel agency).

The calls are to go to the Integration Server ai2main for which the ICF service `/sap/xi/engine` is defined in system A12. The logon data that will be defined later is authenticated when logging on.

**End Points**

**Header Data**

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Receiver business system</td>
<td>AIR2</td>
</tr>
<tr>
<td>Receiver interface name</td>
<td><code>BookFlightIn</code></td>
</tr>
<tr>
<td>Receiver interface namespace</td>
<td><code>http://ai2.com/FlightBooking</code></td>
</tr>
<tr>
<td>Description</td>
<td>Book a Flight</td>
</tr>
</tbody>
</table>

**End Point Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address Mode</td>
<td>URL Addressing</td>
</tr>
<tr>
<td>URL</td>
<td><code>http://ai2main.sap/xi/engine?type=receiver</code></td>
</tr>
</tbody>
</table>
The Apply Logon Data radio button is selected under Logon.

### Logon Data

**Header Data**

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sender business system</td>
<td>TAG</td>
</tr>
<tr>
<td>Sender interface name</td>
<td>*</td>
</tr>
<tr>
<td>Sender interface namespace</td>
<td>*</td>
</tr>
<tr>
<td>Receiver business system</td>
<td>AI2</td>
</tr>
<tr>
<td>Receiver interface name</td>
<td>*</td>
</tr>
<tr>
<td>Receiver interface namespace</td>
<td>*</td>
</tr>
</tbody>
</table>

**Description** | *

### Logon Data

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authentication Procedure</td>
<td>Password-based</td>
</tr>
<tr>
<td>User Name</td>
<td>FLIGHTBK</td>
</tr>
<tr>
<td>Password</td>
<td>*****</td>
</tr>
<tr>
<td>Logon Language</td>
<td>English</td>
</tr>
</tbody>
</table>

---

### End Point for the HTTP Adapter

**Definition**

An end point of this kind is an access point with the technical data for the [HTTP adapter][Seite 357] to send XML documents to an HTTP server.

The editor work area for this end point type has a more complex structure than the other editors and comprises the three tab pages Addressing, Header, and Payload. On the Addressing tab page you define the actual point of access, while the other two tab pages are used to enhance the message (header and payload) with additional data.

**Use**

**Addressing Tab Page**

You define the point of access on this tab page. Choose the addressing mode you want by selecting the appropriate radio button. You can choose between two addressing modes:

- **HTTP Destination**
  
  You must select this addressing mode if you want to use SSL (Secure Sockets Layer Protocol).
  
  Specify an HTTP destination. If the receiver system is an SAP system, the destination is defined as a type H RFC destination (HTTP connection to an R/3 system) on the...
Integration Server. In all other cases, it is defined as a type G RFC destination (HTTP connection to an external server).

If an HTTP destination is used to address the receiver system, you do not need to maintain any logon data because this is already maintained in the HTTP destination.

- **URL Addressing**

  In this case you must specify the URL of the HTTP server in the input field.

  If you address the receiver system using a URL address, then you can specify the following optional parameters for the HTTP adapter:

  | **Proxy: Host** | Optional. The host name of a proxy server, for example for a firewall. |
  | **Proxy: Port** | Optional. The port number of a proxy server, for example for a firewall. |

  There are two ways to log on if you address the receiver system by using a URL address. Select the way you want to log on by using the radio buttons.

  - **Anonymous Logon**
    
    In this case, the receiver does not authenticate the incoming message.

  - **Apply Logon Data**
    
    In this case, logon data is checked when a message arrives. You specify this data in Logon Data [Seite 247].

**Header Tab Page**

On this tab page you can enhance the message header. Specify the content type of the HTTP post in the input field **Content Type**.

Note the following if the **Content Type** field does not contain a value:

- If special characters are to be masked (see the Payload tab page), then the content type is set to: application/x-www-form-urlencoded.

- If special characters are not to be masked, then the content type is set to: text/xml.

If the HTTP server requires additional fields in the header, you can specify these as Name/Value pairs in your end point. You must not use Content Type, Content Length, Authentication or Host as the name, because these attributes are either set automatically by the system, or explicitly by the sender application.

**Payload Tab Page**

You can make the following specifications here:

| **XML Encoding** | If the specified code page for the HTTP body does not correspond to the default setting UTF 8, the system converts the HTTP body to the corresponding code page; in the XML document, the system replaces the tag encoding=UTF-8 with the corresponding code page tag. |

Only select code pages that are installed on the Integration Server.
Use the **Mask Special Characters** check box to define whether special characters in the payload are URL-coded or not.

Examples for escape codes:

- **Escape Code:** `%2F` for the special character `/`
- **Escape Code:** `%26` for the special character `&`

Special characters are not masked within the additional (technical) data in the prolog or epilog because these might contain special characters that must not be encoded.

The HTTP body comprises the prolog, message payload (message contents = message without message header), and epilog. Prolog and epilog are optional and enhance the entire payload that is transferred for particular servers (or CGI servers).

<table>
<thead>
<tr>
<th><strong>Prolog</strong></th>
<th>Optional prolog for the payload. The prolog enables you to enhance the payload with additional (technical) data, for example to include an additional “header” or to comment on the data it contains.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Epilog</strong></td>
<td>Optional epilog for the payload. The epilog enables you to enhance the payload with additional (technical) data.</td>
</tr>
</tbody>
</table>

You can create end points for the HTTP adapter for both SAP and non-SAP systems acting as receiver systems.

**Example**

See: [Example of an End Point (HTTP Adapter)](Seite 242).

---

**Example of an End Point (HTTP Adapter)**

The end point is used in a Business Scenario in which catalog data is sent to a catalog server (mySAP SRM).

Catalog data is sent from the SAP system SUPPLIER to a catalog server CAT, using the interface `CatalogIn`.

**End Points**

The request to the catalog server is structured according to its requirements. The message body must be appended by the HTTP adapter after the ID `&loaderXML=`. The ID `&end=yes` must also mark the end of the message body. Special characters must be transferred encoded in the message body. The encoding must adhere to ISO-8859-1.

No authentication is required on the catalog system side.

**Header Data**

<table>
<thead>
<tr>
<th><strong>Field</strong></th>
<th><strong>Value</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Receiver business system</td>
<td>CAT</td>
</tr>
<tr>
<td>Receiver interface name</td>
<td>CatalogIn</td>
</tr>
<tr>
<td>Receiver interface namespace</td>
<td>Catalog</td>
</tr>
<tr>
<td>Description</td>
<td>Catalogin</td>
</tr>
</tbody>
</table>

---

SAP Exchange Infrastructure (BC-XI) 623 242
Attributes – Addressing Tab Page

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address mode</td>
<td>URL Addressing</td>
</tr>
<tr>
<td>URL</td>
<td><a href="http://requisite:50555/CatalogIn?allowDuplicateItems=FALSE">http://requisite:50555/CatalogIn?allowDuplicateItems=FALSE</a></td>
</tr>
</tbody>
</table>

Logon: Anonymous Logon selected.

Attributes – Payload Tab Page

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML Encoding</td>
<td>ISO-8859-1</td>
</tr>
<tr>
<td>Mask Special Characters</td>
<td>(encoded)</td>
</tr>
<tr>
<td>Prolog</td>
<td>&amp;loaderXML=</td>
</tr>
<tr>
<td>Epilog</td>
<td>&amp;end=yes</td>
</tr>
</tbody>
</table>

End Point for Other XI Connections (XI Connectivity)

Definition
An end point of this type is an access point for other connections to SAP Exchange Infrastructure. This might be an adapter from the Adapter Engine [Seite 362] shipped by SAP, or an adapter developed by a customer or partner.

Use
Choose the addressing mode you want by selecting the appropriate radio button. You can choose between two addressing modes:

- HTTP Address
  Specify an HTTP destination. This destination is defined on the Integration Server as a type G RFC destination.

- URL Addressing
  In this case you must specify a URL in the input field.
  You can display all adapters for this end point type that are connected with the receiver system in the system landscape. Choose under the input field for the URL to select an adapter. The dialog Adapter Selection displays the type of adapter and the respective URL for accessing the adapters for each adapter. Select a URL and choose .

  The adapter selection is only offered for this end point type.

When you use this end point and URL addressing there are two ways that you can log on to the receiver system. Select the way you want to log on by using the radio buttons.

- Anonymous Logon
  In this case, the receiver does not authenticate the incoming message.
Apply Logon Data

In this case, logon data is checked when a message arrives. You specify this data in Logon Data [Seite 247].

These end points cannot be created for SAP systems acting as receiver systems.

Example

See: Example of an End Point (Other XI Connections) [Seite 244].

Example of an End Point (Other XI Connections)

The end point is to be used in a purchasing Business Scenario (mySAP Marketplace).

Car spare parts are ordered from SAP system CAR (car manufacturer), by using the mySAP Marketplace AUTOMOTIVE.

End Points

All calls to the marketplace use the URL of the messaging adapter and the same logon data. Both are already entered in the HTTP destination DESTMP on the Integration Server.

Header Data

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Receiver business system</td>
<td>AUTOMOTIVE</td>
</tr>
<tr>
<td>Receiver interface name</td>
<td>*</td>
</tr>
<tr>
<td>Receiver interface namespace</td>
<td>*</td>
</tr>
<tr>
<td>Description</td>
<td>Order Management</td>
</tr>
</tbody>
</table>

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address mode</td>
<td>HTTP Destination</td>
</tr>
<tr>
<td>HTTP Destination</td>
<td>DESTMP</td>
</tr>
</tbody>
</table>

Logon Data

Logon data is defined within the HTTP destination DESTMP on the Integration Server.

End Point for the Integration Engine for Acknowledgments

Definition

An end point of this type is defined for a sender system that expects an acknowledgment from the receiver of its message.

To define this type of end point, proceed as described in End Point for an Integration Engine [Seite 238], but instead of entering type=receiver in the URL, specify type=sender_back.

Also specify the following values:
For the receiver interface name: **ACKNOWLEDGMENT**

For the receiver interface namespace: **http://sap.com/xi/internal**

⚠️ You can only create acknowledgment end points for an Integration Engine for SAP systems acting as receiver systems.

**Also see:**

For more information, see the Processing of Acknowledgment Messages [Seite 288] section in this documentation.

### Identify End Points Automatically

**Use**

End points can be identified for the Integration Directory from the information for technical systems in the System Landscape Directory [Seite 442]. The Integration Builder has a tool to help you with this process at configuration time.

**Prerequisites**

The information about the corresponding systems must be defined in the System Landscape Directory.

**Procedure**

1. To call the tool, in the Integration Builder main menu (Configuration) choose Tool → Identify End Points Automatically.

2. Choose one or more business systems from the list by selecting the corresponding check boxes. To select all the business systems listed choose 🌟.

3. To confirm, choose Create Proposal 🔄.

   The end points that already exist in the Integration Directory are not displayed in the list. This ensures that nothing is overwritten.

4. The inbound interface name and namespace for the selected business systems are displayed in the frame on the right under Proposal.

5. To create the end point, select one or more lines and choose Create End Points 🔄.

   You can also generate end points for acknowledgment messages. The name **ACKNOWLEDGMENT** and the namespace **http://sap.com/internal** are reserved for acknowledgment message interfaces.

**Result**

The corresponding end points are created in the Integration Directory. You can display the imported end points and edit them further if required. To do so, navigate to the corresponding receiver business system in the navigation area in the Integration Builder, and call the Display/Change End Point [Seite 233] editor.
**Define End Point as Web Service**

You can define end points as web services to call interfaces from receiver systems from any kind of platform. A web service description in WSDL format is created for a receiver (system and interface) that can then be used by the external platform.

So that the SOAP message format used by the Web Services can be converted to the message format used by SAP Exchange Infrastructure, you require the SOAP adapter (see Configuring the SOAP Adapter [Seite 431]).

To define an end point as a Web Service, either:

- Choose **End Point → Define as Web Service** in the Edit End Point editor.

You can only define an end point as a Web Service if it is not an end point for an RFC adapter or an IDoc adapter.

End points that have the wildcard character (*) in the header data also cannot be defined as Web Services.

The Define Web Service dialog contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integration Server Name</td>
<td>Name of the Integration Server that connects SAP Exchange Infrastructure to the external platform. This field is for information purposes only and cannot be changed.</td>
</tr>
<tr>
<td>Integration Server SOAP Inbound Channel (URL)</td>
<td>URL to access the SOAP adapter</td>
</tr>
<tr>
<td>Receiver Name</td>
<td>Name of the receiver system</td>
</tr>
<tr>
<td>Interface Name</td>
<td>Name of the inbound interface at the receiver</td>
</tr>
<tr>
<td>Interface Namespace</td>
<td>Namespace of the inbound interface at the receiver</td>
</tr>
<tr>
<td>SOAP Sender Name</td>
<td>Name of the sender system (external platform)</td>
</tr>
<tr>
<td>SOAP Interface Name</td>
<td>Name of the outbound interface at the sender (external platform)</td>
</tr>
<tr>
<td>SOAP Interface Namespace</td>
<td>Namespace of the outbound interface at the sender (external platform)</td>
</tr>
</tbody>
</table>

If you call this dialog from the Change End Point editor, the fields for the receiver system and the inbound interface (name and namespace) are already completed.

When you have completed all the fields, choose **Generate** (_possible) to generate a WSDL description for the Web Service.
Choose *Save in File* to save the WSDL description locally for use in the future.

The WSDL description can be publicized in the UDDI Business Registry, for example.

**Also see:**
*Converting WSDL to the Target Language* [Seite 188].

## Defining Logon Data

### Purpose

You have the option of defining logon data to control system access at the receiver systems. Some *end point types* [Seite 234] require logon data to be specified. The end point type of the receiver business system determines whether logon data must be specified or not.

The *end point for the RFC adapter* [Seite 236] requires logon data, for example.

You can choose whether you want to use logon data or anonymous log on for the following end point types:

- *End point for an Integration Engine* [Seite 238] and *end point for an Integration Engine for Acknowledgments* [Seite 244] (for URL addressing)
- *End point for the HTTP adapter* [Seite 240] (for URL addressing)
- *End point for other XI connections (XI Connectivity)* [Seite 243] (for URL addressing)

When addressing using an HTTP destination, the logon data is already defined in the destination for these end point types.

Logon data for an *end point for an IDoc adapter* [Seite 235] is already defined by using an RFC destination and does not need to be specified explicitly.

### Features

The logon data for a receiver business system is contained in one configuration object and defined at configuration time. The Integration Builder provides you with various tools for performing the following tasks.

- *Display an Overview of Logon Data* [Seite 247]
- *Display/Change Logon Data* [Seite 248]

## Display Overview of Logon Data for a Receiver

### Use

If you have specified multiple logon objects for a receiver system you can display them all in an overview.
In this case, the icon 🌟 is displayed next to the receiver under Logon Data for Receiver System in the Integration Builder navigation area. If a logon data object is specified, then the icon 🌟 is displayed.

Prerequisites
To display the overview of logon data for a receiver, in the navigation area of the Integration Builder position the cursor on the receiver and double click, or choose Open ( ) from the context menu.

Features
The overview header displays the receiver system that the logon data is defined for. The logon data is displayed in a table with the following information:

- Sender
  - Business System
  - Interface Name
  - Interface Namespace

- Receiver
  - Interface Name
  - Interface Namespace

The editor header displays the receiver system since it applies for all logon data objects listed.

- Logon Procedure

Functions for Editing/Displaying the Table

<table>
<thead>
<tr>
<th>Icon</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>🌟</td>
<td><strong>Open</strong>: Calls the Display/Change Logon Data [Seite 248] editor for the selected table line.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Document</th>
<th>Page</th>
<th>Icon</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>📄</td>
<td><strong>Create New Object</strong>: Calls the Create New Object [Seite 40] function</td>
</tr>
<tr>
<td></td>
<td></td>
<td>🗑️</td>
<td><strong>Delete Object</strong>: Deletes the selected logon data object. The deleted object is removed from the overview.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>🔄</td>
<td><strong>Update Data</strong>: Updates the overview</td>
</tr>
</tbody>
</table>

Displaying/Changing Logon Data

Use
You have the option of displaying or changing logon data.
**Prerequisites**

You have either created new logon data or have opened existing logon data.

Note the following when creating logon data:

- You can also use the wildcard character (*) to complete input fields. For further information, see [Generic/Specific Definition of Configuration Objects](#).
- If the business system is an SAP system, you can also use the RFC namespace.
- In the `Create New Object` dialog, select the receiver system first. Next, select the receiver interface. In both cases you can also call search help. Search help only displays those interfaces that are available in the receiver business system.
- You also have the option of defining the logon data for a particular sender. In the `Create New Object` dialog, show or hide the input fields for the sender by choosing `Sender`.

You can also open logon data in the overview by positioning the cursor on a line and choosing `Open` or by double clicking.

**Features**

**Header**

The editor header displays the following fields:

- **Sender**
  - Business System
  - Interface (Name and Namespace)

  These fields can also use the wildcard character (*) as a value. For further information, see [Generic/Specific Definition of Configuration Objects](#).

- **Receiver**
  - Business System
  - Interface (Name and Namespace)

You cannot make any changes to these fields because the values for them were defined when you created the logon data.

The editor header also displays the status of the objects and you have the option of entering a brief description of the logon data.

**Work Area**

Here you can specify the following attributes for the logon data.

**Attributes for Logon Data**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
</table>

SAP Exchange Infrastructure (BC-XI) 623 249
Authentication Procedure | Choose either of the following values:
| • Password-based
| • Password-based (SAP)
| You must also specify the SAP client in the logon data for the latter.

| SAP Client | Only applicable if you choose Password-Based (SAP) as the authentication procedure.
| SAP system client

| User Name | Name of the user who is logged on the receiver system

| Password | User password

| Logon Language | Language user logged on with

If you specify whether the receiver system is an SAP system or not in the System Landscape Directory, the appropriate authentication procedure is automatically selected. The corresponding attribute is then not displayed here.

**Example**
See End Point Types [Seite 234] for examples.

**Generic/Specific Definition of Configuration Objects**

**Use**

When creating [Seite 40] interface determinations, end points, and logon data, you have the option of specifying a wildcard character (*) instead of a specific value for some key attributes. The object is then defined for all values of the key attribute. This means that two kinds of object exist: generically defined objects and specifically defined objects.

At runtime, the object with the most specific key attributes takes precedence (“specific object has priority”).

If when you create an end point (EP1) the field for the name of the receiver interface contains the wildcard character, then this end point is defined for all interfaces of the receiver. If another end point EP2 exists where the receiver interface is specified but otherwise all key attributes are as pronounced as those in EP1, then from the runtime perspective, end point EP2 takes precedence.

Part masking of fields (for example, field value=BUS*) is not supported.

Whether the wildcard character can be used for a particular key attribute depends on the object type (see Generic/Specific Definition of Interface Determinations [Seite 251], Generic/Specific Definition of End Points [Seite 253] and Generic/Specific Definition of Logon Data [Seite 253]).
Generic definition of configuration objects enables you to minimize the work effort required when creating and changing configuration data. In particular, the likelihood of errors occurring when changing configuration data is greatly reduced because changes to configuration objects automatically apply for all values of the masked key attributes.

In most cases it is advisable to define an end point for all interfaces of a receiver because the technical access information for the receiver systems is not usually dependent on the receiver interface. If you specifically defined end points for each individual interface of a receiver, you would have to enter the same data numerous times since you would have to create a separate end point object for each interface of the receiver. Any changes would then have to be made manually in each end point object. However, changes to a generically defined end point are less likely to result in errors because they are only created once and are then applied automatically for each individual interface of the receiver.

Note that creating configuration objects generically can result in undesired side-effects. Therefore, refer to the restrictions and recommendations outlined in this documentation (see Generic/Specific Definition of Interface Determinations [Seite 251], Generic/Specific Definition of End Points [Seite 253] and Generic/Specific Definition of Logon Data [Seite 253]).

It is not possible to generically define receiver determinations.

**Generic/Specific Definition of Interface Determinations**

**Use**

The wildcard character (*) can be used for the following key attributes when creating an interface determination:

- Sender system
- Receiver system

However, you must specify both the outbound and inbound interfaces.

Furthermore, simultaneous definition of the following two interface determinations is not permitted:

<table>
<thead>
<tr>
<th>Interface Determination</th>
<th>Sender system</th>
<th>Outbound Interface</th>
<th>Receiver system</th>
<th>Inbound Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID1</td>
<td>A</td>
<td>OIF</td>
<td>*</td>
<td>IIF</td>
</tr>
<tr>
<td>ID2</td>
<td>*</td>
<td>OIF</td>
<td>B</td>
<td>IIF</td>
</tr>
</tbody>
</table>

In this case, the following situation can arise when a message is processed:

Sender System=A and Receiver System=B

This case applies for the definition of both interface determinations. Therefore, the unique assignment of an interface mapping is no longer guaranteed.
Note that when you define an interface determination generically, you must be aware that if you then define an interface determination specifically (which has key attributes that partly match the key attributes of the generically defined interface determination), business scenarios that use the generic interface determination may be affected.

The reason for this is that interface determinations are not defined for specific business scenarios. It is not possible to assign configuration data defined in an interface determination exclusively to one particular business scenario. The most specifically defined object always takes precedence at runtime, regardless of the business scenario involved.

Therefore, you must always be aware when using the wildcard character that the definition you make may be made redundant by a more specific definition of an interface determination at a later date.

If you use the wildcard character for the receiver system when creating an interface determination, define this interface determination (with a particular interface mapping /M1) as valid for all receiver systems. This generic interface determination is used in a business scenario BS_1.

Now a more specific interface determination (for example, for a particular sender/receiver pair) is created for the same interfaces (for example, for a second business scenario BS_2). This interface determination might reference another interface mapping (IM2), for example. In this case, the definition of the interface mapping IM1 (runtime perspective) made in the generic definition is now redundant for the specified sender/receiver pair. In this way, a more specific definition of an interface determination can influence the business scenario BS_1 at runtime.

Example

The wildcard character is used as follows when defining three interface determinations:

**Definition of Interface Determinations**

<table>
<thead>
<tr>
<th>Interface Determination</th>
<th>Sender system</th>
<th>Outbound Interface</th>
<th>Receiver system</th>
<th>Inbound Interface</th>
<th>Mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID1</td>
<td>*</td>
<td>OIF</td>
<td>*</td>
<td>IIF</td>
<td>M1</td>
</tr>
<tr>
<td>ID2</td>
<td>A</td>
<td>OIF</td>
<td>*</td>
<td>IIF</td>
<td>M2</td>
</tr>
<tr>
<td>ID3</td>
<td>B</td>
<td>OIF</td>
<td>C</td>
<td>IIF</td>
<td>M3</td>
</tr>
</tbody>
</table>

**Triggering at Runtime**

- **Message 1:**
  
  Sender System=A and Receiver System=B
  
  In this case, mapping M2 is used.

- **Message 2:**
  
  Sender System=B and Receiver System=C
  
  In this case, mapping M3 is used.

- **Message 3:**
  
  Sender System=B and Receiver System=D
  
  In this case, mapping M1 is used.
**Generic/Specific Definition of End Points**

**Use**
The wildcard character (*) can be used for the following key attributes when creating end points:

- Interface Name
- Interface Namespace

Additionally, note the following:

- You must always specify the business system
  Since an end point contains the technical access information for a receiver system, the configuration object End Point must refer to a particular system.

- The combination (interface namespace with *) and (interface name without *) is not permitted. If the interface namespace uses a *, then the interface name must also use one.
  If the interface namespace uses the wildcard character, it is recommended that you do not specify the interface name since the interface is not unique if you do not specify the interface namespace. However, if a wildcard character is used for the interface name but a specific interface namespace is specified, this means that the end point is defined for all interfaces in this namespace.

- The wildcard character applies for all message interfaces and not RFCs or IDocs.

**Example**

**Defining an End Point for an RFC Adapter**

<table>
<thead>
<tr>
<th>Receiver system</th>
<th>Receiver interface name</th>
<th>Receiver interface namespace</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIR</td>
<td>*</td>
<td>urn.sap-com:document:sap:rfc:functions</td>
</tr>
</tbody>
</table>

This end point type is defined when the RFC namespace is specified.

---

**Generic/Specific Definition of Logon Data**

**Use**
The same applies for the use of the wildcard character (*) when creating logon data as for creating end points [Seite 253].

Additionally, note the following:

- When defining logon data it is possible to specify the fields for business system and interface in both the receiver and sender systems.
  The background behind this is that in principle, business processes are possible in which the logon data for a receiver depends on who sent the message.

**Example**

In this example for the definition of logon data for a receiver system (AIR), the key fields are completed as follows:

**Example for the Specification of Key Fields for Logon Data**
### Directory Cache

To ensure that the relevant Integration Engines have quick access to the contents of the Integration Builder at runtime, the information required is loaded into corresponding caches. Caches also contain the runtime-relevant data from the System Landscape Directory.

The directory cache is always updated:

- When a change list [Seite 46] in the Integration Directory is updated
- When a change list [Seite 46] in the Integration Repository is updated, that contains objects that are used in the Integration Directory

This only concerns mapping objects.

#### Analyzing the Directory Cache

There is a transaction that you can call to analyze the directory cache. This transaction is useful for troubleshooting and support within SAP Exchange Infrastructure.

To call the transaction, log on to the system with the Integration Engine client and choose Exchange Infrastructure: Configuration → SXI_CACHE XI Directory Cache (transaction SXI_CACHE).

The configuration objects are displayed in a tree structure as in the Integration Builder (also see: Navigation Area [Seite 31]).

Business Scenarios from the Integration Directory are not displayed.

To display a configuration object on the right-hand side of the screen, double click an object in the tree structure.
A traffic light is used to indicate whether the cache is up-to-date or not. If this is not the case (red light), this means that there are objects in the activated change list of the Integration Directory that are not yet in the directory cache.

You can choose from the following functions:

- **Update the directory cache**
  To update the directory cache choose *XI Directory Cache → Start Cache Comparison*. The missing activated change lists are included in the directory cache.

- **Refresh the complete directory cache**
  To refresh the directory cache completely, choose *XI Directory Cache → Start Complete Cache Refresh*.

- **Display errors that occurred when updating the directory cache**
  If an error occurred while updating the directory cache, a red traffic light is shown. To display information about the error, *Goto → Display Refresh Error*.

- **Display the event table**
  To display a list of processed change lists, choose *Goto → Event Table*.

- **Analyze data about IDoc and RFC processing from the System Landscape Directory**
  Data that is required to identify receivers of IDocs and RFCs logically is stored and managed in the System Landscape Directory. This information is stored in a table on the Integration Server so that it does not have to be imported into the directory cache each time an IDoc or RFC is processed. To display this data, choose *Goto → Business Systems*.

### Notes on displaying configuration objects in the directory cache

The attributes of the configuration objects are described in the documentation for the respective object (see the description of the corresponding object in *Configuration [Seite 207]*). Note the differences in the transaction for analyzing the directory cache:

- If you chose *Receiver Identification from Message Contents* when defining a routing relation, the value *BP* (Business Partner) is displayed when you display the routing relation in the directory cache in the *RecTyp* table. *BS* (Business System) means that the receiver is explicitly stated (Also see: *Defining Receivers [Seite 218]*).

- Additional information for technical routing is displayed for *end points [Seite 233]*.

  The details are displayed here in the same way as in the message header.

### SAP System Landscape Directory in the SAP Exchange Infrastructure

#### Purpose

The *SAP System Landscape Directory [Seite 442]* (SLD) is the central information provider in a system landscape.

The SLD contains two types of information:
• **Component information:** This is information about all available SAP products and components, including their versions. If there are any third-party products in the system landscape, they are also registered here.

• **Landscape description:** This contains all installed systems in a system landscape.

At **design time** [Seite 76] of the integration objects, the component information is extracted from the SLD to define business scenarios.

When a collaborative business process is **configured** [Seite 207], the landscape description is needed to determine the system information of the business partners involved.

The **runtime environment** [Seite 264] of the SAP Exchange Infrastructure (the Integration Engine), receives the address of the Integration Server of a business system from the SLD.

---

### Technical System Landscape

#### Purpose

Technical systems are installed application systems (a CRM server, for example). You can manage installed application systems by using the system landscape description. To do this, you have to register all the installed systems by using the technical system browser.

#### Features

You can use the technical system browser to perform the following tasks:

- **Display technical systems** [Seite 256] [Seite 256]
- **Configure an SAP system** [Seite 257] [Seite 257]
- **Configure a standalone Java system** [Seite 259] [Seite 259]
- **Configure a third-party system** [Seite 260] [Seite 260]

---

### Technical System Browser

#### Purpose

You can use the technical system browser to display all registered application systems. From there you can start additional activities with technical systems.

#### Displaying Technical Systems

To call the technical system browser, go to the initial screen of the SLD and choose **System Landscape → Technical Landscape**.

The browser displays the screen **Technical System Browser**.

You can use the drop-down list **Technical System Type** to choose between the following types of technical system:

- **SAP Systems:** Represents SAP application systems that run on the SAP Application Server.
- **SAP Standalone Java Systems:** Represents standalone application systems that are implemented in Java.
- **Third-Party Systems:** Represents application systems of third-party vendors.
The technical system browser displays all the application systems of the relevant type.

In the field Display Filter, which is next to the selection list, you can enter a string that you want the displayed rows to contain. In this way you can restrict the display list.

To view or process detailed information about a particular system, select the system directly in the list.

The browser displays the system as a detailed view.

To save changes to a system, choose Update.

To remove a system from the SLD, choose Remove.

To remove multiple systems from the SLD, select the systems in the overview list, and then choose Remove Selected.

Configuring an SAP System

Use
This section describes how to configure an SAP system and its clients in the system landscape description.

Prerequisites
You require the user role LcrInstanceWriterLD to execute this task.

Registering an SAP System
You have navigated to the Technical System Browser.

1. Choose New Technical System...
   The system displays the page Technical System Wizard.
2. Choose SAP Technical System, and then choose Next.
   The browser displays all the registered product versions in the list Available Products.
3. Select the products that are installed in this SAP system, and then choose Add.
   The browser displays on the lower part of the page a list of all the software components that are contained in the selected products.

   Repeat this step as many times as necessary until you have added all products.
   To remove a product, select it in the list, and then choose Remove.
   You can add more products to this SAP system at a later stage.
4. Select the software components that are actually installed in the system.
5. Choose Next to continue.
6. Enter the 3-character system name, the installation number of the system, and the name of the corresponding database server, and then choose Next.
7. Enter the host name and the port of the corresponding message server, and the host name of the central application server including its instance number. Choose Next to continue.

You can run multiple software instances of the application server on one host. Specify the instance that acts as a central server.

8. Enter all the application servers of this system, and then choose Next.

In the first field, enter the host where an application server runs. In the second field, enter the software instance that runs on this host.

9. Enter at least one client, and then choose Finish.

The SLD server creates the new SAP system.

**Maintaining Application Servers of an SAP System**

An SAP system often consists of a server group, which means that the load can be distributed. You can create a server group directly when you create the SAP system. If necessary, you can add more application servers to the group. To register these additional servers in the SLD, proceed as follows:

1. On the Technical System Browser page, select the system to which you want to add application servers.

The system displays the detail page for the selected system.


The system displays the page for creating a new application server.

3. Enter the host name and the software instance of the new application server.

   Optionally, you can enter other data.

4. To create the new application server, choose Save.

   The SLD server adds the new application server to the selected SAP system.

To process or remove an application server, select the server on the detail page of the SAP system. The system displays the page for the selected application server. Proceed according to the instructions displayed.

**Maintaining Clients of an SAP System**

You can set up new clients in an SAP system. To register a new client in the SLD, proceed as follows:

1. On the Technical System Browser page, select the system to which you want to add a new client.

The system displays the detail page for the selected system.

2. Under Clients, choose Add.

   The system displays the page for creating a new client.

3. Specify the client number.

4. To create the new client, choose Save.

   The SLD server adds the client to the selected SAP system.
To process or remove a client, select the client on the detail page of the SAP system. The system displays the page for the selected client. Proceed according to the instructions displayed.

**Maintaining Installed Products of an SAP System**

To add another product to the system, proceed as follows:

1. Go to the Technical System Browser, and then select the system to which you want to add a new product.
   
   The system displays the detail page for the selected system.
2. Under Installed Products, choose Add.
   
   The system displays the page for registering a new installed product.
3. Select the product and the relevant software component that you want to add, and choose Add Product.
   
   The SLD server adds the new product to the system.

To process or remove an installed product, select the product on the detail page of the SAP system. The system displays the detail page for the installed product. Proceed according to the instructions displayed.

---

**Configuring a Java System**

**Use**

Whereas SAP systems handle business processes by using a highly developed client system, most applications run as a standalone system. These systems are called standalone systems.

This section describes how to define a standalone Java system.

**Prerequisites**

You require the user role LcrInstanceWriterLD to perform this task.

**Procedure**

In the Technical System Browser, proceed as follows:

1. Choose New Technical System….
   
   The browser displays the screen Technical System Wizard.
2. Choose SAP Standalone Java System, and then choose Next.
   
   The browser displays the screen for creating a standalone Java system.
3. Enter the technical system identification and the system host name, and then choose Finish.
   
   The SLD server creates the standalone Java system.

   The technical system identification is a unique system ID.
   
   The system host name specifies the host where the application runs.
Configuring a Third-Party System

Use
Unlike SAP products, third-party products are not necessarily included in the component description. Therefore, you must first register the third-party product before you can register an installed application system of this product in the system landscape description.

This section describes how to do this task.

Prerequisites
You require the user role LcrInstanceWriterLD to execute this task.

Procedure
You have navigated to the Technical System Browser.

   The browser displays the screen Technical System Wizard.
2. Choose Third-Party System, and then choose Next.
   The browser displays all the registered product versions in the list Available Products.
3. Select the products that are installed in this system, and then choose Add.
   The browser displays on the lower part of the screen a list of all the software components that are contained in the selected products.

   Repeat this step as many times as necessary until you have added all products.
   To remove a product, select it in the list, and then choose Remove.
   You can add more products to this system at a later stage.
4. Select the software components that are actually installed in the system.
5. Choose Next to continue.
6. Enter the system name and the host name of the system, and then choose Finish.
   The SLD server creates the new third-party system.

SAP Exchange Infrastructure

Purpose
Systems use the SAP Exchange Infrastructure to communicate with each other. Business systems are logical systems that send or receive messages. They can be SAP systems or third-party systems.

An SAP system usually has one or more clients that function independently of each other as logical units at runtime. Each of these clients represents a business system in the SAP Exchange Infrastructure. You can define an RFC destination for sender business systems.

A third-party system is also a logical unit that functions as a sender or receiver. Therefore, third-party systems are also business systems in this sense.
The SAP Exchange Infrastructure also enables you to make function calls between systems that are not registered as technical systems in the system landscape description. You only need to register these systems as third-party business systems. These third-party business systems are referred to as business partners.

The following provides you with information about configuring:

- Business systems [Seite 261]
- Business partners [Seite 262]

## Integration Server and Business System Group

In the SAP Exchange Infrastructure, a business system is one of the following:

- Application system
- Integration server

Application systems are business applications that are involved in the business processes.

An integration server, on the other hand, enables communication between application systems at runtime. It cannot execute business logic. For detailed information about the functions of the integration server, see the Runtime [Seite 264] section.

You can assign an integration server to a business system group. A business system group is a logical grouping of business systems that use an integration server to communicate with each other.

## Configuring a Business System

### Prerequisites

To define business systems for the Exchange Infrastructure, you require the user role LcrInstanceWriterAll.

### Registering an SAP System as a Business System

You have navigated to the initial screen of the SLD.

To define a client of an SAP system as a business system, proceed as follows:

1. On the initial screen, choose System Landscape → XI Landscape.
   
   The browser displays the XI Landscape screen.

2. Choose New Business System... .
   
   The browser displays the screen Business System Wizard.

3. Choose SAP Business System and then choose Next to continue.
   
   The browser displays a drop-down list of all the SAP systems that are available, and a list of their clients.

4. Select the SAP system and the client, and then choose Next to continue.

5. Select all the installed products that are involved in the planned business process. Choose Next.

6. Specify the name of the business system, and then choose Next.
7. Select the role of the business system.
   a. If the business system is an application system, you can specify the corresponding integration server.
   b. If the business system is an integration server, specify its pipeline URL.

   For detailed information about the pipeline setting, see the section Integration Engine [Seite 266].

8. To save the business system, choose Finish.
   The SLD server registers the specified client as a business system.

**Registering a Standalone Java System as a Business System**

To register a standalone Java system as a business system, you proceed in a similar way as for registering an SAP system. However, note the following differences:

- You do not need to enter a client for the selected system.
- You do not need to specify for the system which products are installed in the system. Only one product runs in a standalone system.
- A standalone Java system cannot act as an integration server. You can specify only one integration server for the selected system.

**Registering a Third-Party System as a Business System**

To register a third-party system as a business system, you proceed in a similar way as for registering an SAP system. However, note the following differences:

- You do not need to enter a client for the selected system.
- A third-party system cannot act as an integration server. You can specify only one integration server for the selected system.

**Configuring a Business Partner**

**Prerequisites**

To enter business partners in the system landscape description, you require the user role LcrInstanceWriterLD.

**Procedure**

To register a business partner, proceed as follows:

1. On the initial screen of the SLD, choose System Landscape → XI Landscape.
   The browser displays the XI Landscape screen.
2. Choose New Business System.
   The browser displays the screen Business System Wizard.
3. Choose Business Partner System and then choose Next to continue.
4. Enter the name of the business partner and its URL. Choose Next to continue.
5. In the list Related Integration Server, select the corresponding server.
6. To save the business partner, choose Finish.
The SLD server creates the business partner.

#### Configuring Groups and Transport Targets

**Use**

In the System Landscape Directory you can define groups of business systems to identify different components in your system landscape and to control these components individually.

The systems employed in the SAP Exchange Infrastructure test can be grouped together in a group called *Test*, while those systems used in production operation can be grouped together in the group *Productive*.

You can define business system transport targets to map the names of business systems in different groups in your system landscape, which correspond to each other. This setting is a prerequisite for exporting and importing configuration contents from the Integration Directory (see Exporting and Importing Integration Objects [Seite 73]).

In your test configuration of SAP Exchange Infrastructure you defined a business scenario with all the corresponding configuration objects. To transfer this configuration to the productive environment by using importing and exporting and to then employ it in productive operation, you must identify the corresponding business systems in the test and productive environments. For this purpose, assign the test system to group *Test* and the productive system to the group *Productive*. The mapping of the names to the business systems that correspond to each other is defined by using transport targets.

---

**System Landscape**

<table>
<thead>
<tr>
<th>Group Test</th>
<th>Group Productive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integration Server (Test)</td>
<td>Integration Server (Productive)</td>
</tr>
<tr>
<td>Business Syst. 1 (Test)</td>
<td>Business Syst. A (Productive)</td>
</tr>
<tr>
<td>Business Syst. 2 (Test)</td>
<td>Business Syst. B (Productive)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**Mapping Rule Using Transport Targets:**

*Business Syst. 1 → Business Syst. A*  
*Business Syst. 2 → Business Syst. B*  
*...*
Prerequisites
All business systems are entered in the System Landscape Directory.

Procedure

Defining a Group
Create groups for the various areas of use and assign each group an Integration Server.

1. Choose XI Landscape on the initial screen of the System Landscape Directory.
2. From the dropdown list box for the input field Groups, choose Edit Groups....
3. In the Define Business System Group dialog, choose Create.
4. Enter a name for the group and assign it an Integration Server.
   Business systems are assigned to a group indirectly by using the Integration Server. To assign a business system and an Integration Server, proceed as described in Configuring a Business System [Seite 261].
5. Create the group by choosing Create.

Defining a Transport Target for a Business System
Define which business systems in two different groups (G1 and G2) correspond to each other.

1. Choose XI Landscape on the initial screen of the System Landscape Directory.
2. Open the business system by clicking its name.
3. Choose Transport Target.
4. From the dropdown list box for the input field Group, choose the group (G2). This group is assigned the system corresponding to the business system that is open.
5. From the dropdown list box for the Target input field, choose the business system that corresponds to the business system that is open in the other group.

Runtime
Exchange Infrastructure runtime serves the exchange of messages as per the configured business processes. As the graphic below illustrates, the process of transferring a message involves various components:
The graphic shows a connection between two application systems using proxies. However, the proxies alone are not able to establish a connection between a sender and a receiver. Messages can only be exchanged once the logical business process has been configured (also when using adapters):

- The connection must be set up in logical routing (LR).
- The connection must have an end point (TR: Technical Routing), and logon data, if applicable.
- If you have used a mapping, you must have saved and assigned it (M). If you are using adapters you must execute a mapping between the XML format that the adapter gets or expects and the XML format that generates or expects a proxy.

You want to send a message to an RFC module using an ABAP outbound proxy. Therefore, you require a mapping from proxy XML to RFC XML.

The contents are defined partly during development and are then configured later by a consultant.

There are also settings to be made that are purely administrative in nature. To be able to forward the message, the proxy runtime must know the address of the Integration Engine. The runtime acquires this information as follows:

1. The proxy runtime fetches the System Landscape Directory (SLD) address from the exchange profile. The SLD contains all business systems that belong to the Integration Server.
2. The proxy runtime sends the SLD a technical ID for the sender system. Once this ID is entered in the SLD and the corresponding associations are available, the proxy runtime receives the name of the sender business system and the address of the corresponding Integration Server as a response.
3. The proxy runtime can now specify the logical business system as the sender in the message and forward it to the Integration Server.

This process is similar when using an adapter.
There should be a contact person for any queries that the development team might have regarding administrative settings. See the Configuration Guide for a description on how to configure the servers within the Exchange Infrastructure for runtime. The configuration guide is available on the Service Marketplace. See http://service.sap.com/instguides.

Transactions for troubleshooting are located in the area menu Integration Engine - Monitoring (transaction SXMB_MONI on the Integration Server). Also see: Monitoring XML Messages [Seite 290]

Integration Engine

Purpose

The Integration Engine [Extern] enables you to process XML messages [Seite 278] exchanged between applications in heterogeneous system landscapes. Both applications of SAP components and external applications can communicate with the Integration Engine.

Using adapters [Seite 340], the Integration Engine can also process IDocs [Extern] (Intermediate Documents) and RFCs [Extern] (Remote Function Calls). This option is available for applications that have already defined IDocs or RFCs, as well as for new applications of SAP components that do not yet have access to the proxy generation [Seite 186] functions.

Introductory Notes

To ensure that XML messages are processed consistently, you first need to configure [Seite 267] the Integration Engine correspondingly and then monitor [Seite 290] message processing.

Integration

The Integration Engine is the runtime environment of the SAP Exchange Infrastructure [Seite 1]. The following are examples of other important parts of this infrastructure:

- Proxy Generation
- Integration Repository
- Integration Directory

The business processes involved can take place between heterogeneous system components within a company, as well as between business partners in the Internet, for example, using a marketplace.

Features

The Integration Engine as a runtime component has the task of receiving, processing and forwarding XML messages. During message processing, routing information [Seite 214] is evaluated, the physical receiver system [Seite 231] is determined, and mapping activities [Seite 143] are executed.

The individual processing steps are called pipeline services [Seite 267] and are defined in pipelines [Seite 267] that are configured and shipped by SAP. Pipelines cannot be modified. It is the Integration Engine’s task to process these pipelines correctly and consistently.
Pipelines and Pipeline Services

Definition

Pipeline [Extern] is the term used to refer to all steps that are performed during the processing of an XML message. The individual processing steps are called pipeline elements.

Pipeline services [Extern] provide functions for the processing of XML messages and are represented by the individual pipeline elements. Each pipeline element is linked to a pipeline service. Pipeline services can be used by more than one pipeline element.

Use

Pipelines represent a fixed processing run for particular XML messages and differentiate themselves depending on the role of the Integration Engine [Seite 266]. There are pipelines for a local Integration Engine in the sender or receiver system, or for a central Integration Engine that is configured as the Integration Server [Seite 269], for example.

Structure

A pipeline comprises individual pipeline elements that an XML message must perform when it is processed. Each pipeline element calls a pipeline service.

Pipeline services read and change inbound XML messages, like for example, mapping [Seite 143] or routing [Seite 214]. They can also call processing processes, for example functions to create business data in an SAP component. The service type, number, and sequence are specified for each pipeline and cannot be changed.

Integration

The pipeline services to be used are either part of the Integration Engine itself or originate from other areas of the SAP Exchange Infrastructure [Seite 1]. You call pipeline services from one standard interface.

Configuring the Integration Engine

Purpose

Before you can use the Integration Engine [Seite 266], you must first configure it for the existing system landscape, according to the business processes involved.

Prerequisites

You have called the transaction Integration Engine – Administration (SXMB_ADM) and the system displays the screen Integration Administration.

Process Flow

The screen Integration Engine: Administration contains a menu that comprises two areas: Administration and Configuration.

Both areas mainly apply to system administrators. They are used to set up the system to process XML messages [Seite 278]. This enables you to make changes to SAP default settings or to configure test settings.
There is an online checklist available for each area, which you can call from the menu by choosing Checklist: Administration or Checklist: Configuration. Use this list to check whether you have carried out the necessary steps to ensure that the Integration Engine is configured and administered correctly.

The following functions are available to help you configure an Integration Engine:

- Setting global and specific configuration data [Seite 269]
- Defining interfaces for archiving [Seite 271] XML messages
- Making pipeline settings [Seite 275]

You also have the following administration options:

- Registering, de-registering, and activating queues [Seite 279]
- Scheduling jobs for the archiving and deletion [Seite 273] of XML messages
- Making settings for troubleshooting [Seite 276]

This function must only be used by experts who are familiar with the technology of the Integration Engine.

The authorizations for the individual functions are based on the authorization object S_XMB_AUTH. They can be derived from the role SAP_XI_ADMINISTRATOR and must be allocated as follows:

**Configuration**

- Activity
  - 01 (create)
- Operation
  - Not defined

**Administration**

- Activity
  - 02 (change)
- Operation
  - Not defined

You cannot execute options in the menu that have the locked symbol (🔒), because you do not have the required authorization.

**Result**

When you have configured the Integration Engine, it can be called by an XML-enabled application, for example using an outbound interface of proxy generation [Seite 186].

If you have configured your Integration Engine as the central Integration Server [Seite 269], using the IDoc Adapter [Seite 340] you can also process IDocs (Intermediate Documents).
Displaying/Changing Configuration Data

Use

The configuration data for the Integration Engine is client-specific. This means that you can configure multiple business systems in different ways in one SAP system. The following is differentiated between:

- **Global configuration data**
  
  You use the global configuration data to define the role of your current business system.
  
  You can also load the global configuration data from your System Landscape Directory, if it is already maintained there. You can also define a configuration locally, for example for test purposes, which is different to the data of the System Landscape Directory.

- **Specific configuration data**
  
  Use the specific configuration data to define parameters for different categories, with which you can configure your business system.

Prerequisites

In the Integration Engine menu you chose Configuration → Integration Engine Configuration, and are now on the screen Integration Engine Configuration Data.

Loading Global Configuration Data

To load your global configuration data from the System Landscape Directory, follow the procedure below:

1. Choose Edit → Change Global Configuration Data.
2. Choose System Landscape – Load Configuration.
   
   The system displays a message if no data is maintained in the System Landscape Directory.
3. Save the loaded data.
4. To undo the loading of global configuration data from the System Landscape Directory, choose Undo Global Changes.

Changing Global Configuration Data

To change the global configuration data displayed, proceed as follows:

1. Choose Edit → Change Global Configuration Data.
2. To change the usage type of your business system, in the field Role of Business System, select an option using the input help.
   
   A distinction is made between the following roles of a business system:
   
   - Integration Server
     
     The Integration Server (Extern) receives XML messages, determines the receiver, executes mappings, and routes the XML messages on to the corresponding receiver system. This is a centrally configured Integration Engine that is not able to execute application logic.

   ![Note](Image)

   There can only be one Integration Server in an SAP system. This means that only one client (business system) can be defined as the Integration Server.
Application system with a local Integration Engine that requires the Integration Server to execute the integration logic (that can run in another client of the same SAP system)

The system can act as a sender or receiver of XML messages, but it does not contain any integration logic. Therefore, neither routing nor mapping can be executed locally. Instead they must be executed by the Integration Server, which is called by the application system.

Although integration logic is only offered in ABAP-based business systems, the sender/receiver role of an Integration Engine can also be made available in other business systems.

3. If you want to call an Integration Server, but do not want to configure your business system as an Integration Server, in the field Corresponding Integration Server, enter the address for the particular Integration Server.

There are two ways of specifying an address:

- **http://<host>:<port>/sap/xi/engine?type=entry**
  
  `<host>` is the host name of the Integration Server and `<port>` is the HTTP port.

  If you have already maintained this entry in your System Landscape Directory, you can copy the entry by choosing System Landscape – Load Configuration.

  The name of the user and the password must be defined with the HTTP service (/sap/xi/engine) in the target system. The user is the person who processes the XML messages on the Integration Server.

- **dest://<IntegrationServer-Destination>**
  
  `<IntegrationServer-Destination>` is an RFC destination of type HTTP Connection (type H) to the Integration Server. In this case, host name, port, and path prefix are stored in the RFC destination.

  First create a type H RFC destination with the host name and the HTTP port of the Integration Server. Enter the following path prefix:
  
  `/sap/xi/engine?type=entry`

  In this case, the name of the user and the password are stored in the RFC destination in the current system; you do not need to specify them in the HTTP service in the target system.

  You must place the prefix **dest://** before the name of the RFC destination so that the system knows that this is not a direct specification of the address, but a logical definition.

4. Save your changes. The system displays a message to confirm that the configuration was changed.

**Changing Specific Configuration Data**

To change the specific configuration data, proceed as follows:

1. Using the input help for the field **Category**, choose the categories of parameters in which you want to make changes.

2. Choose Change Specific Configuration Data.

   The system navigates to the view **Configuration of the Integration Engine**. Here you have the same options as for displaying a view [Extern] with transaction SM30.
The view displayed contains a table with configuration parameters that have already been defined for the selected category. For further information about the individual parameters, select the icon for the corresponding line.

If you did not select a category of parameters for the field Category in step 1, then the table contains all configuration parameters that have already been defined.

In change mode you can now do the following:

- Change the current value of an entry
  The current value overrides the default value
- Create a new entry
  Use the input help in each case here. You can either accept the default value or specify a current value, which then overrides the default value at runtime.
  Select the column Prefix to define the corresponding subparameter as a prefix. This enables you to predefine a configuration value generically for a set of subparameters.
- Select an entry and delete it from the table

Modified or new entries are not automatically transported since the majority of parameters are local parameters.

If you do want to transport your entries, choose Utilities → Compare or Utilities → Adjust and proceed as described in Comparing Customizing Objects [Extern].

You can also transport your entries manually by calling transaction SE09 and including the transport object R3TR TABU SXMSCONFVL with the corresponding table keys (configuration parameters) in your transport request.

---

**Defining Interfaces for Archiving and Retention Periods**

**Use**

This is where you define interfaces [Seite 117] so that you can archive their XML messages. You can also specify for how long XML messages are retained in the database before they are deleted or archived.

To periodically schedule jobs for archiving [Seite 273], in the Integration Engine menu, choose Administration → Schedule Archiving Jobs.

**Defining Interfaces for Archiving**

To define interfaces for archiving, proceed as follows:

1. In the Integration Engine menu, choose Define Interfaces for Archiving and Retention Periods.

   The system navigates to the screen Define Interfaces for Archiving, where you can enter the interfaces you want to define. The system differentiates between sender and receiver interfaces at this point.

2. To copy an interface to the list of interfaces displayed, enter the interface name and namespace [Seite 59] in the respective fields, and choose Flag Interface for Archiving.
You can either enter the interfaces you want to archive manually, or select them from the input help. However, the input help can only offer interfaces and namespaces that are known to your Integration Directory [Extern]. If you make an entry in either Name or Namespace, then input help takes this into account for both fields.

If you enter FLIGHT* in the Name field, only those entries whose interface name begins with FLIGHT will be shown when you select the input help for Name or Namespace.

Which interfaces you want to archive is generally customer-specific, since it depends on the respective applications. For this reason, SAP does not supply any default values.

To select multiple interfaces simultaneously, choose Multiple Selection. The system displays a list in the form of a table with all interfaces known by your Integration Directory.

The table contains the interface type (Inbound/Outbound Interface), the name and the namespace. In the Inbound/Outbound Interface column, O stands for a sender (outbound) interface and I for a receiver (inbound) interface.

You can select any number of entries in the table and add them to the list of interfaces to be archived by choosing Copy. Use the filter function to restrict the number of interfaces displayed.

It is also possible to copy generic entries such as FLIGHT* to the list. You must enter the namespace in this case. The interface name is only determined for this variant when the interfaces you want to archive have been determined at runtime.

Try to select as few interfaces as possible using the input help. Instead, use the generic entry option. Only specify fully qualified interfaces if lots of interfaces exist that have the same prefix, and you only want to archive one of them.

3. To delete an interface from the list, select the corresponding line and choose Delete Interface from List.

4. Save your changes.

All changes made are automatically entered in a customizing request. The system displays a corresponding dialog box, in which you must specify a request to transport the changes.

5. Specify or create a customizing request, and choose Continue.

Your changes are transported.

The interface type (I or O) is not included in the customizing request. In other words, all selected interfaces are transported independently of the interface type.

In the customizing request, all underscores (_) in interface names are replaced by an asterisk (*) and all subsequent characters are rejected (for example, xyz-_abc becomes xyz*).

**Defining Retention Periods**

To define retention periods for messages in the database, proceed as follows:

The system navigates to the Define Retention Periods screen, where you can enter in the corresponding fields the number of days that XML messages marked for deletion or archiving are to be retained in the database.

If you want processed synchronous XML messages without errors to be deleted immediately, enter 0.

2. Save your changes.
3. To transport your changes, in the Integration Engine menu, choose Integration Engine Configuration [Seite 269] and then choose Change Specific Configuration Data.

The system navigates to a screen where the retention periods you specified are represented by the corresponding configuration data as per the following table:

<table>
<thead>
<tr>
<th>Message Type</th>
<th>Category</th>
<th>Parameter</th>
<th>Subparameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Asynchronous messages without</td>
<td>DELETION</td>
<td>PERSIST_DURATION</td>
<td>ASYNC</td>
</tr>
<tr>
<td>errors to be deleted</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Asynchronous messages without</td>
<td>ARCHIVE</td>
<td>PERSIST_DURATION</td>
<td>ASYNC</td>
</tr>
<tr>
<td>errors to be archived</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Synchronous messages with errors to</td>
<td>DELETION</td>
<td>PERSIST_DURATION</td>
<td>SYNC</td>
</tr>
<tr>
<td>be deleted</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Synchronous messages without errors</td>
<td>DELETION</td>
<td>PERSIST_DURATION</td>
<td>SYNC</td>
</tr>
<tr>
<td>to be deleted</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Choose Utilities → Compare or Utilities → Adjust and proceed as described in Comparing Customizing Objects [Extern].

**Example**

Asynchronous XML messages to be deleted are retained in the database for 3 days. Asynchronous XML messages to be archived are retained in the database for 2 days.

The sender interface FlightBooking.Create is the only interface entered in the list.

Therefore, all XML messages that have the interface FlightBooking.Create as their sender are written to the archive and deleted from the database two days after they have been successfully processed.

All other XML messages are deleted from the database three days after they have been successfully processed.

XML messages that do not have the status Processed Successfully, remain in the database.

**Archiving and Deleting XML Messages**

**Use**

You can archive messages that have been processed correctly as well as messages that were processed with errors and that were canceled manually. The default setting is to delete all such messages. All messages that are not to be deleted must be archived.

To archive XML messages, you must first define their corresponding interfaces [Seite 271] and then schedule the archiving jobs.

You require multiple archiving jobs to archive XML messages:

- A job to determine the messages to be archived
• A job to write the messages to the archive
• A job to delete the archived messages from the database

You require two jobs to delete XML messages:
• A job to determine the messages to be deleted
• A job to delete the archived messages from the database

You can reschedule all jobs periodically, but you must maintain the job sequence.
You can select [Seite 301] and display [Seite 302] archived XML messages.

Prerequisites
You have defined the interfaces for the XML messages to be archived using Configuration → Define Interfaces for Archiving and Retention Periods in the Integration Engine menu.

To archive XML messages, use the archiving object BC_XMB and use SAP_BC_XMB as the archiving infostructure. The archiving infostructure must be activated.

Schedule Archiving Jobs
To schedule archiving jobs, proceed as follows:

1. From the Integration Engine → Administration menu, choose Schedule Archiving Jobs.
   The system displays the SAP Archive Administration [Extern] screen that you can use to schedule your archiving jobs.
2. In the field Object Name, enter the archiving object (BC_XMB) for the Integration Engine [Seite 266].
3. Schedule your preprocessing [Extern], write [Extern], and delete [Extern] jobs by choosing the corresponding action.
4. Choose Read [Extern] to start the read program for your archiving object to read already archived XML messages.
   You can either start the read program in the background or in dialog mode. For the latter, choose Archived XML Messages (Search Using Archive) [Seite 301] in the Integration Engine → Monitoring menu.
5. To display an overview of archiving jobs that have already run for your archiving object, choose Manage [Extern].
6. To display statistics for the individual archiving jobs, choose Statistics [Extern].

   !
   For performance reasons, the statistics values are generally imprecise.

Schedule Delete Jobs
To schedule deletion jobs, proceed as follows:

1. From the Integration Engine → Administration menu, choose Schedule Delete Jobs.
2. Select the corresponding check box of the job(s) to be scheduled.
3. Specify the start time and date.
4. Specify the period you want to use and choose Schedule.

For an overview of all jobs, choose Jobs. For each preparation job there must be a deletion job.
Making Pipeline Settings

Use
For each pipeline of the Integration Engine, you can make various settings that are analyzed at runtime. You can check and change these settings as required.

Prerequisites
In the Integration Engine menu you chose Configuration → Pipeline Settings, and are now on the screen Pipeline Settings Display.

Procedure
To display the settings for a pipeline, proceed as follows:

1. On the tab page Pipeline Settings, use the input help for the field Pipeline ID to select a pipeline. Choose Enter to display the selected pipeline.
2. Choose Display <-> Change (to enter change mode and change the pipeline settings in the group box with the same name, if required.
   You can change the settings for logging and trace level for each pipeline here.
   - You can activate or deactivate logging for the entire pipeline.
     1 means that logging is activated and 0 that it is deactivated. An empty input field means that the setting for logging is to be defined at another location.
   - You can activate or deactivate the trace function for the entire pipeline using the trace level (0-3).
     Level 0 means that no trace is activated, while level 3 means that maximum trace is activated.
   - You can set pipeline execution to automatic (A) or manual (M). The setting for the execution mode cannot be changed.
3. Save the pipeline after you have made your changes.
4. Choose Other Pipeline if you want to select another pipeline and check its logging and tracing settings.

Displaying/Changing Pipeline Elements

Use
A pipeline comprises multiple processing steps (pipeline elements) that an XML message must pass through. The pipeline elements define the individual actions of a pipeline to be executed. Each pipeline element is linked to a pipeline service.

Existing pipeline elements can be displayed for each pipeline and changed if necessary.

Prerequisites
In the Integration Engine menu you have chosen Configuration → Pipeline Settings and have then selected the tab page Pipeline Elements. The system now displays the screen Pipeline Elements Display.
**Procedure**

You can access existing pipelines from the input help for the field *Pipeline ID*.

1. To display a particular pipeline, select it and choose *Enter*.
   
   The system displays the individual elements of the selected pipeline in the *Pipeline Element List*. The first element in the list is displayed in the group frame *Pipeline Element*.

2. To display another element, select the element in question from the *Pipeline Element List* by using the ‹ and › pushbuttons, or mark it in the list and choose *Detail*.
   
   The selected pipeline element is displayed in the group frame *Pipeline Element*.

3. To change an element of the selected pipeline, display it in the group frame *Pipeline Element* and choose *Display <-> Change*.
   
   You can only change settings for *logging* [Seite 291] in change mode. Specify for the pipeline element displayed whether you want to activate logging (1), deactivate logging (0), or define the setting for logging at another location (input field remains empty).

4. Make the change to the pipeline element and choose *Add to List* to move the changed element back to the pipeline element list.

5. Save the pipeline after you have made your changes.
   
   When you save the pipeline you must assign it a valid and transportable package [Extern]. The pipeline together with its pipeline elements and the related pipeline services are then written in a transport request. When you assign the package, the original pipeline entry is written.

   If you want to, you can also define a pipeline as *Local* when saving. The pipeline cannot then be transported.

6. Choose *Other Pipeline* if you want to select another pipeline and display or change its elements.

---

**Defining Conditions and Actions for Troubleshooting**

**Use**

In the Integration Engine [Seite 266], you can specify conditions for troubleshooting in inbound messages that trigger particular actions. Conditions and actions each refer to attributes of the message header.

For each conditioned action, you can define one or more conditions that you then link conjunctively (using a logical AND relation). You can assign more than one action to a condition.

Only create a limited number of conditioned actions to maintain a clear overview of their dependencies and the network of their actions. In production operation, do not create any conditioned actions, otherwise each inbound message is checked by the system to see whether it fits any of the conditions.
Prerequisites

In the menu Integration Engine, you chose Administration → Troubleshooting Settings and the system now displays a screen of the same name.

Creating Conditioned Actions

The procedure for creating a conditioned action is divided into the creation of one or more conditions and one or more actions. To create a conditioned action, proceed as follows:

1. Choose Create.

   The system displays a dialog box with two tab pages. Choose the tab page Conditions. The first line in the table displayed is ready for you to enter data. Create a new condition.

   The condition you want to create is activated automatically (卞). The user and the expiration data and time of the condition are predefined. If the validity period expires, the rule is automatically deactivated (卞).

2. Using the input help, select a rule, for example USERNAME.

3. Enter an argument (a value) for the selected condition, for example SMITH.

4. If necessary, change the expiration date and time of the condition.

5. To create additional conditions, choose Create for each further condition and make the corresponding entries in each of the selected lines.

6. To deactivate a condition that you created, choose Activate <-> Deactivate (卞).

7. To delete a condition from the table, select the corresponding line and choose Delete.

8. Once you have finished creating your conditions, choose the tab page Actions.

   The first line in the table displayed is ready for you to enter data. Create a new action. The user and the date of the last change are predefined.

9. Using the input help, select an action. You can choose from the following actions:

   o Tracelevel

      Allows you to log the individual processing steps of a message in the trace segment of this message. You can then display the trace information for messages that were persisted in the database using monitoring [Page 290].

      The action Tracelevel overrides trace settings that you made for the pipeline [Page 275].

   o Tracetofile

      Allows you to also write the trace information for the message to a file. In the SAP directory /global, the system creates the file TRC_<msgid>.txt, where <msgid> contains the respective message ID.

   o Logging

      Allows you to control whether a message is logged in the database following certain processing steps. You can then display the log using monitoring.

      Only set the action Logging if you have not already activated or deactivated logging [Page 291] explicitly at pipeline or pipeline element level.

   o Logtofile
Allows you to also write the message log to a file. In the SAP directory /global, the system creates the file MSG_<msgid>_<pid>_<msgvs>.txt, where <msgid> contains the respective message ID, <pid> contains the ID of the pipeline involved, and <msgvs> contains each logged version of the message.

- Debugstring
  Allows you to activate a breakpoint set in code that then starts the debugger at this location. You must activate HTTP debugging for this purpose for messages that enter the Integration Engine through the HTTP inbound interface.

10. Using the input help, select an argument (a value) for the action.

11. If necessary, create further actions by choosing Create for each additional action and making the corresponding entries.

  Each action may only occur once for each conditioned action.

12. To delete an action from the table, select the corresponding line and choose Delete.

13. Save the actions and conditions you have created.

The system displays the conditioned action you created in the screen Troubleshooting Settings; you can change or delete the conditioned action here.

If you have created multiple conditioned actions, the system automatically links them together disjunctively (using a logical OR relation). If several of the conditioned actions apply, the system executes them in the same sequence that they are displayed on the screen Troubleshooting Settings. For multiple identical actions, this means that the results of the last executed action take precedence each time.

Deleting Conditioned Actions
To delete a conditioned action, choose Delete in the corresponding block on the screen Troubleshooting Settings. The system removes the conditioned action from view.

Changing Conditioned Actions
To change a conditioned action, choose Change in the corresponding block on the screen Troubleshooting Settings. The system displays a dialog window with the tab pages Conditions and Actions, where you can make changes to the conditioned action.

Processing XML Messages

Purpose
The Exchange Infrastructure [Seite 1] is based on the sending of XML messages between individual application components [Extern]. The processing of XML messages is therefore an essential part of this infrastructure. The infrastructure is based on information located in a central repository and directory that is called at runtime. The runtime environment required to process this information is known as the Integration Engine [Seite 1].

Prerequisites
To be able to process XML messages successfully, you must first have set up the entire integration environment and configured [Seite 267] the runtime environment correspondingly.
You also require the role SAP_XI_APPL_SERV_USER to be authorized to process XML messages. The authorizations contained within this role are required for the communication between the user who is logged on who calls the Integration Server using an application, and the user under which message processing takes place on the Integration Server.

**Process Flow**

When the Integration Engine receives an XML message, it processes it using its pipeline ([Seite 267]); this pipeline mainly comprises the following processing steps:

- Inbound processing with receiver and interface identification ([logical routing [Seite 214]], [technical routing [Seite 231]], and [message branching [Seite 284]])
- Outbound processing with [mapping [Seite 143]] and the receiver call

Routing refers to the receiver or receivers of an XML message that must be localized within the existing system landscape.

While the name of the receiver and the receiver interface are determined during logical routing, technical routing queries the end point (the physical address) in the Integration Directory ([Extern]) that is required to call this interface.

Mapping means the mapping of values and structures of the sender application to the corresponding values and structures of the receiver application.

For more information about message processing, see [Configuration Time [Seite 21]].

The following aspects must also be considered when processing XML messages using the Integration Engine:

- [Quality of Service [Seite 279]]
- [Commit Handling [Seite 282]]
- [Receiver Determination [Seite 284]]
- [Displaying System Errors [Seite 285]]
- [Processing Status [Seite 286]]
- [Processing of Acknowledgment Messages [Seite 288]]

---

**Quality of Service**

The sender of a message uses the attribute *Quality of Service* (QoS) to determine how a message is delivered. The following types of quality of service are supported:

- **BE (Best Effort):**
  
The message is sent synchronously; this means that the sender system waits for a response before it continues processing.

- **EO (Exactly Once):**
  
The message is sent asynchronously in this case; this means that the sender system does not wait for a response before continuing processing. The Integration Engine ([Seite 266]) guarantees that the message is sent and processed exactly once.

- **EOIO (Exactly Once In Order):**
  
In addition to *Exactly Once*, messages with the same queue names (supplied by the application) are delivered in the same sequence that they were sent from the sender system. Message processing is asynchronous in this case.
In the case of quality of service BE, an error occurs if more than one receiver is determined for a message. In the case of delivery types EO and EOIO, the message is copied correspondingly and sent to the individual receivers.

**Synchronous Message Processing (BE)**

Messages are not persisted by the Integration Engine in synchronous message processing. Once a message has been processed in the target system it performs an implicit database commit [Seite 282].

**Asynchronous Message Processing (EO, EOIO)**

In asynchronous message processing, messages are persisted by the Integration Engine at a predefined time and scheduled for processing using the queued RFC [Extern] (qRFC).

The qRFC requires the technical names of the inbound and outbound Integration Engine queues for this. These names are different depending on the role of the Integration Engine [Seite 269]:

- **Central Integration Server [Extern]**
  
  In this case the Integration Engine has an inbound and outbound queue.

- **Local Integration Engine without integration logic**
  
  In this case the Integration Engine has a sender and receiver queue.

In addition to these queues, two further queues are required for the processing of acknowledgment messages [Seite 288].

So that all XML messages (including acknowledgment messages) can be processed automatically, you must register queues that are used by the Integration Engine with transaction SMQR. Choose *Administration → Manage Queues* from the Integration Engine menu for this purpose.

Adhere to the following conventions for prefixes when naming these queues.

**Queue Name (maximum 24 characters)**

<table>
<thead>
<tr>
<th></th>
<th>Sender</th>
<th>Inbound</th>
<th>Outbound</th>
<th>Receiver</th>
<th>Acknowledgment</th>
</tr>
</thead>
<tbody>
<tr>
<td>EO</td>
<td>XBTSx</td>
<td>XBTlx</td>
<td>XBTOx</td>
<td>XBTRx</td>
<td>XBTBx</td>
</tr>
<tr>
<td>EOIO</td>
<td>XBQSx</td>
<td>XBQlx</td>
<td>XBQOx</td>
<td>XBQRx</td>
<td>XBQBx</td>
</tr>
</tbody>
</table>

The suffix \( x \) in the queue name is determined differently depending on the quality of service; the suffix for the sender, receiver and inbound queues is determined in the same way.

**Quality of Service EO**

For quality of service EO, a four-digit number is generated with a random algorithm to determine the suffix of the sender, receiver and inbound queues (in other words the queues in which an inbound message is placed for inbound processing) to complete the queue name. This enables you to process multiple messages simultaneously in different queues.

Random number: 38

Queue name: XBT10038

You can determine the area valid for the random algorithm using the configuration parameter [Seite 269] EO_INBOUND_PARALLEL from the category Tuning. Using the subparameters SENDER, CENTRAL and RECEIVER you can define separate areas for the sender queue, inbound queue, and receiver queue.

On the central Integration Server, messages are persisted again after inbound processing and scheduled for sending to the receiver determined in technical routing [Extern]. To avoid
them being blocked, messages are bundled according to receiver and then placed in outbound queues assigned to the respective receivers. The corresponding queue is determined using a four-digit encryption (a hash value) of the receiver name.

Receiver name: CLIENT_XYZ
Encrypted name: E8B9
Queue name: XBT0E8B9

If more than one message is sent to the same receiver, use the configuration parameter [Seite 269] EO_OUTBOUND_PARALLEL of the category Tuning to process messages simultaneously in different queues. A random algorithm is also used here to generate a four-digit number to complete the queue name. You must enter the receiver in the column Subparameter and the area valid for the random algorithm in the column Value for this purpose.

Receiver name: CLIENT_XYZ
Encrypted name: E8B9
Random number: 17
Queue name: XBT0E8B90017

If all messages are to be processed in parallel outbound queues, using the configuration parameter specify the area for the random algorithm only, without entering a special receiver.

The message is persisted again once it has been sent successfully. Once the message has been processed successfully in the target system, the Integration Engine executes an implicit database commit [Seite 282]. If an error occurs during message processing, the Integration Engine executes an implicit database rollback.

**Quality of Service EOIO**

In contrast to quality of service EO, with EOIO the queue name supplied by the application is used to determine the suffix of the sender, receiver and inbound queues. This name must not be more than 16 characters long and it is added to the prefix of the queue name. You can use the following characters in the name: numbers (0-9); upper case letters (A-Z); underscores (_); and forward slashes (/).

Queue name from the application: APPQUEUEUNAME (maximum 16 characters)
Inbound queue name: XBQIAPPQUEUEUNAME

The name of the corresponding outbound queue is then determined on the central Integration Server to bundle the messages according to receiver. In the same way as for quality of service EO, the name of the receiver taken from technical routing [Extern] is used. In this case the name of the application queue is used as well.

Receiver name: CLIENT_XYZ
Encrypted name: E8B9
Outbound queue name: XBQOE8B9APPQUEUEUNAME (maximum 24 characters)

Quality of service EOIO guarantees that inbound messages with the same receiver are processed in the same sequence that they entered the Integration Engine. The messages are
also sent in this sequence to their respective receivers; because they have the same queue name that was supplied by the application, they are placed in the same outbound queue.

If the receiver is not the same, then this sequence can no longer be guaranteed.

The message is persisted again once it has been sent successfully. Once the message has been processed successfully in the target system, the Integration Engine executes an implicit database commit [Seite 282]. Otherwise, it executes an implicit database rollback.

Commit Handling

When using the Integration Engine [Seite 266] transactions are basically processed as follows:

1. The application temporarily writes application data to its database tables.
2. The Integration Engine is called in the same transaction to inform the communication partners about the changes with a message. The message is temporarily written to the Integration Engine database. The Integration Engine is called with QualityOfService = ExactlyOnce [Seite 279] (corresponds to tRFC [Extern]) or QualityOfService = ExactlyOnceInOrder [Seite 279] (corresponds to qRFC [Extern]).
3. The application ends the transaction with an explicit COMMIT WORK (or ROLLBACK WORK if an error has occurred).

This saves all temporary data (application data, recorded messages of the Integration Engine) to the database. Furthermore, the message data for sending and processing by the Integration Engine is released (separated asynchronously). In the case of rollback, all temporary database entries are discarded.

Comparison with RFC-Based System Response

How does communication using the Integration Engine compare with the RFC [Extern]-based system response?

Database Commit with Synchronous Calls

Synchronous Integration Engine calls are flagged using QualityOfService = BestEffort [Seite 279].

The Integration Engine behaves in the same way as RFC communication [Extern] in an SAP system:

If a synchronous RFC is executed during a transaction, the system executes database commits (DB_COMMIT). A DB_COMMIT saves all previously temporary data to the database (just like for a COMMIT WORK). This data can no longer be reset by a ROLLBACK WORK later on.

Transport of Logical Unit of Work with for Asynchronous Calls

Asynchronous Integration Engine calls are characterized by QualityOfService = ExactlyOnce (corresponds to tRFC) or QualityOfService = ExactlyOnceInOrder (corresponds to qRFC).

The Integration Engine behaves in much (but not entirely) the same way as RFC communication in an SAP System.

As with tRFC or qRFC, both temporary application data and Integration Engine calls are written together in the database (as an atomic transaction) with a COMMIT WORK.
However, unlike tRFC or qRFC, different Integration Engine calls within a transaction are sent in different messages. Each Integration Engine call generates its own, independent message.

This means that the logical unit of work comprising the individual calls is not transported to the target system of the call. This contrasts with tRFC and qRFC, because if the target system of these calls is identical, then the calls that took place within a transaction are also processed in one transaction by the receiver.

a. The application writes application data to its database tables.

b. Integration Engine is called for the first time:
   (QualityOfService = ExactlyOnce / ExactlyOnceInOrder)
   Message M1 generated

c. Integration Engine is called for the second time:
   (QualityOfService = ExactlyOnce / ExactlyOnceInOrder)
   Message M2 generated

d. Application ends transaction with an explicit COMMIT WORK.

The COMMIT WORK writes the application data (from step a) and the message data (from step b and step c), in other words the messages M1 and M2, to the local database. At the time of execution or sending (asynchronous call), M1 and M2 are always sent independently of each other and are also processed independently in the target system (with a separate COMMIT WORK for M1 and M2).

- QualityOfService = ExactlyOnce
  M1 and M2 are sent by the sender Integration Engine independently of each other and in parallel. In the target system, M1 is processed with a COMMIT WORK. Later (or even beforehand because of parallel processing), M2 is processed with its own COMMIT WORK. The processing sequence of M1 and M2 at the receiver cannot be determined in advance.

- QualityOfService = ExactlyOnceInOrder
  M1 is sent by the sender Integration Engine. Only when message M1 has been successfully persisted (for example, in the receiver Integration Engine) is M2 sent by the sender. This sequence is maintained, including application calls.

  This means that message M1 is sent to the application in the target system, and is completed with a COMMIT WORK.

  If an error occurs in the application for M1, then the processing of further messages (M2 and all subsequent messages in the queue) is blocked. When M1 has been saved, M2 is transferred to the application and is saved with its own COMMIT WORK.

Transactions and Synchronous Communication Using the Integration Engine

Synchronous calls that write as well as synchronous calls within transactions (both in the Integration Engine and also in RFCs) can lead to data inconsistencies.

Synchronous Integration Engine Calls Within a Transaction

What happens if synchronous Integration Engine calls are executed within a transaction?

The principles shown above clearly define the behavior of synchronous Integration Engine calls (QualityOfService = BestEffort) within transactions. The way the Integration Engine
responds corresponds exactly to the way we know synchronous RFCs respond when used within transactions. In other words, inconsistencies may occur.

   a. The application writes application data to its database tables.
   b. The Integration Engine is called with \textit{QualityOfService = BestEffort}.
      The message M1 is generated and sent synchronously, in other words, an answer is expected.
   c. The application ends the transaction with an explicit COMMIT WORK (or ROLLBACK WORK in case of error).

This action means that the implicit DB\_COMMIT saves the application data from step a in the database after step b (see \textit{Database Commit for Synchronous Calls} above).

This means that if the synchronous Integration Engine call should fail (due to a \textit{Return} with application or system errors), the application data from step a can no longer be rolled back with ROLLBACK WORK. Consequently, this causes inconsistencies.

\textbf{Synchronous Integration Engine Call with Actions at Receiver}

What happens if you execute database inserts or updates at the receiver within a synchronous Integration Engine call?

Inconsistencies can also occur in this case. Synchronous calls (both RFC and Integration Engine calls) cannot guarantee that the database inserts or updates are unique. If the synchronous call is not returned (for example, due to a communication error), the caller does not know whether the call was successfully completed at the receiver end or not.

A repeat call (if successful) will make the change again in the database in each case. This could possibly cause DUPREC errors (\textit{Duplicate Record}), in other words, when the system attempts to write a second database record with an identical key. This could then cause system crashes (RABAX), or data inconsistencies.

Asynchronous protocols (\textit{QualityOfService = ExactlyOnce} corresponding to tRFC or \textit{QualityOfService = ExactlyOnceInOrder} corresponding to qRFC) avoid data inconsistencies. Furthermore, these protocols independently retry communication and execution as long as the receiver does not confirm that the call has been successful. Therefore, write calls and calls within transactions must only be executed with asynchronous Integration Engine logs.

\textbf{Receiver Determination}

The receiver of an XML message is determined during inbound processing in logical and technical routing [Seite 214]. The number of receivers found then determines how processing will continue.

- If the system cannot find a receiver, it displays an error and terminates processing.
- If only one receiver is found, the message is placed in the corresponding outbound queue for outbound processing.

The \textit{quality of service} [Seite 279] \textit{Exactly Once} (EO) has the following distinctive feature:

Using the \textit{configuration parameter} [Seite 269] EO\_INBOUND\_TO\_OUTBOUND, you can specify the category TUNING so that a message is not placed in the outbound
queue as usual; instead the message is processed and sent entirely in the inbound queue.

This improves performance, but there is the disadvantage that the inbound queue is blocked if the receiver cannot be reached. However, in the outbound case, the only queue that is blocked is the one where the messages for this receiver are located.

- If the system finds more than one receiver, then for quality of service *Exactly Once In Order (EOIO)* and *Exactly Once (EO)* the message is branched, as described below.

For quality of service *Best Effort (BE)* an error occurs in these cases.

### Message Branching for EOIO

For quality of service EOIO, a new message with a new message ID is created from the original message for each interface/receiver pair in the receiver list that was determined during logical routing.

So that the message branch remains unambiguous, the message ID of the original message is saved with each new message.

To guarantee that the processing sequence from the receiver list is maintained for messages with the same receiver, the queue name specified by the application is included in each new message.

### Message Branching for EO

Likewise for quality of service EO, a new message with a new message ID is created from the original message for each interface/receiver pair in the receiver list that was determined during logical routing.

If more than one message is going to the same receiver, an internal queue name is generated from the message ID of the original message, to ensure that the correct processing sequence is maintained. As a result, the quality of service switches from EO to EOIO.

---

**Displaying System Error Descriptions**

There are two ways to display descriptions (long texts) of system errors in the Integration Engine [Page 266](#) (including all system errors in pipeline services [Page 267](#)).

### Querying Long Texts in the Integration Engine

To query long texts for system errors in the Integration Engine, first select [Page 294](#) XML messages with errors, then select the message you want from the list displayed and choose *Error Information*.

The error text is displayed in the performance assistant, together with the error code and the error short text. If no error text exists for an error, the system displays the following message:

*No description exists for this system error.*

### Querying Long Texts Using a Web Browser

To query long texts on system errors using a web browser, for example, from a Java tool, use the special HTTP service ERRORINFO. You access this service using an HTTP address with corresponding parameters.

The system then displays the error long text in HTML format in the Web browser.

Use the following HTTP address to access the HTTP service for displaying error texts:
Enter the following for the individual parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>code=</td>
<td>Category and ID (columns in the table SXMSSYERR in the Integration Engine)</td>
</tr>
<tr>
<td>p1=</td>
<td>Value of error parameter 1</td>
</tr>
<tr>
<td>p2=</td>
<td>Value of error parameter 2</td>
</tr>
<tr>
<td>p3=</td>
<td>Value of error parameter 3</td>
</tr>
<tr>
<td>p4=</td>
<td>Value of error parameter 4</td>
</tr>
<tr>
<td>language=</td>
<td>Language, for example EN for English</td>
</tr>
</tbody>
</table>

The values for the individual parameters and the language are located in the error header of the message that contains errors.

**Example**

http://us0050:1081/sap/xi/errorinfo?code=PROTOCOL.IDOC_ADAPTER.GENERIC
&p1=aa
&p2=bb
&p3=cc
&p4=dd
&language=EN

**Web Access Error Messages**

If no long text exists for a pair consisting of category and ID (entry in the table SXMSSYSERR) that exists in the system, the system displays the following error message in HTML:

No description exists for this system error.

If no corresponding pair consisting of category and ID exists at all (entry in the table SXMSSYSERR missing entirely), the system displays the following error message in HTML:

Error code does not exist

**Processing Status**

During message processing, each XML message passes through multiple statuses. These statuses are represented by a corresponding icon in the Status or Acknowledgment Status column in the list of processed XML messages. You can display such a list of XML messages in the monitor for processed XML messages.

- If you have not determined any versions of the XML message, the system displays the overall processing status of the message. This corresponds to the status following the last processing step executed.
- If you have already determined the versions of the message, the system displays the status of the respective processing step.
Explanation of the Different Statuses

The following statuses refer to synchronously and asynchronously processed XML messages.

The statuses in the first table refer to a status whereby processing is complete, while the statuses in the second table refer to an error status.

Completed Processing of Synchronous and Asynchronous Messages

<table>
<thead>
<tr>
<th>Status</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Checkmark]</td>
<td>Message processed correctly. This might only apply for the current system for asynchronous processing.</td>
</tr>
<tr>
<td>![Alert]</td>
<td>A message with this message ID already exists in the receiver system. Processing is therefore completed for the sender system.</td>
</tr>
<tr>
<td>![Alert]</td>
<td>Processing of a message with errors was ended. The message can then be deleted with the next delete job.</td>
</tr>
<tr>
<td>![Alert]</td>
<td>The message has already been recorded but has not yet been released for processing.</td>
</tr>
<tr>
<td>![Alert]</td>
<td>Receiver identification is complete and multiple receivers have been identified. As a result, the message will be branched. Message processing is complete for this message. The new messages produced following branching are processed further, however.</td>
</tr>
</tbody>
</table>

Error Statuses of Synchronous and Asynchronous Messages

<table>
<thead>
<tr>
<th>Status</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Alert]</td>
<td>A system error occurred during message processing in the SAP Exchange Infrastructure [Seite 1]. In synchronous processing the error is returned to the sender.</td>
</tr>
<tr>
<td>![Alert]</td>
<td>An application error occurred during message processing. For asynchronous processing, this status only occurs in the receiver system, for synchronous processing, it occurs in all systems involved.</td>
</tr>
<tr>
<td>![Alert]</td>
<td>Receiver identification is complete and no receivers have been identified. Message processing is therefore complete for this message.</td>
</tr>
</tbody>
</table>

The following statuses refer to intermediate statuses in message processing. They only apply to asynchronously processed messages.

Intermediate Processing Statuses of Asynchronous Messages

<table>
<thead>
<tr>
<th>Status</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Checkmark]</td>
<td>Message has been released for processing. This status indicates the inbound message, which is the original version of a message.</td>
</tr>
<tr>
<td>![Alert]</td>
<td>Receiver identification [Seite 284] is complete. Exactly one receiver was identified and the message was rescheduled for further processing.</td>
</tr>
<tr>
<td>![Alert]</td>
<td>The message is in retry mode. This means that there was an automatic restart following errors in the HTTP connection.</td>
</tr>
<tr>
<td>![Alert]</td>
<td>The message has been scheduled again. This is the status of the message when you restart a particular message version manually.</td>
</tr>
<tr>
<td>![Alert]</td>
<td>The message contained errors, was corrected manually and saved.</td>
</tr>
<tr>
<td>![Alert]</td>
<td>The acknowledgment message [Seite 288] was stopped.</td>
</tr>
<tr>
<td>![Alert]</td>
<td>The message was sent to the Process Engine for further processing.</td>
</tr>
</tbody>
</table>
The message version was merely generated by logging and does not have a particular status.

The following statuses display the contents of acknowledgments of synchronously processed messages in the **Acknowledgment Status** column.

**Statuses for Acknowledgments of Asynchronous Messages**

<table>
<thead>
<tr>
<th>Status</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="Green.png" alt="Green" /></td>
<td>No acknowledgment message has arrived yet.</td>
</tr>
<tr>
<td><img src="Yellow.png" alt="Yellow" /></td>
<td>The acknowledgment message does not contain any errors. Therefore, the message was received from the receiver without any errors, or it was processed by the application correctly, or both.</td>
</tr>
<tr>
<td><img src="Red.png" alt="Red" /></td>
<td>The acknowledgment message contains a system or application error. This means that message processing was not successful.</td>
</tr>
<tr>
<td>![Double Arrow](Double Arrow.png)</td>
<td>More than one receiver was identified during logical routing so the acknowledgment message has to be branched. Message processing is complete for this acknowledgment message. The two new acknowledgment messages produced following branching are processed further, however.</td>
</tr>
<tr>
<td><img src="Green.png" alt="Green" /></td>
<td>Processing of the message was ended and it can now be deleted if required.</td>
</tr>
<tr>
<td><img src="Red.png" alt="Red" /></td>
<td>The acknowledgment message cannot be sent.</td>
</tr>
</tbody>
</table>

**Processing Acknowledgment Messages**

An acknowledgment message informs the sender of an asynchronous (request) message about the result of the processing of the message within SAP Exchange Infrastructure. The reported result is referred to as an asynchronous acknowledgment.

**Request Messages**

Within the request message you must define whether an acknowledgment message is required, and if so, what kind of acknowledgment message. The acknowledgment message can refer to two events:

- The arrival of the request message at the final receiver
- The execution of the application in the receiver system

For this purpose, the request message contains an acknowledgment tag that specifies the event for which an acknowledgment is expected. The sender (or an inbound adapter that supports acknowledgment messages) must provide the data for this tag.

Message processing in SAP Exchange Infrastructure ensures that the acknowledgment message follows the same route from the receiver to the sender as the request message did from the sender to the receiver. For this purpose, the request message contains a hoplist that logs the route of the request message. All Integration Engines and Integration Servers that are involved in the processing of the request message provide the data for this log.

**Acknowledgment Message**

As soon as the request message encounters an event that requires an acknowledgment message, a corresponding acknowledgment message is created and sent to the sender.
An acknowledgment message obtains a new message ID, but also has a reference to the message ID of the request message as well as the value Response in the Directory tag of the message header. The request message hoplist is copied to the acknowledgment message header, thus enabling backward routing of the acknowledgment message.

If a message is branched following receiver identification [Seite 284], a separate acknowledgment message is returned for each newly created request message. These acknowledgments enable the sender to recognize that its original message has been branched.

Acknowledgment messages are processed using backward pipelines. They are defined for each pipeline [Seite 267] involved in the processing of request messages:

- SENDER_BACK for the pipeline SENDER
- CENTRAL_BACK for the pipeline CENTRAL
- RECEIVER_BACK for the pipeline RECEIVER

The main task of these backward pipelines is to return an acknowledgment message to the previous systems, therefore:

- The receiver system or an outbound adapter that supports acknowledgment messages sends the result back to the Integration Server (RECEIVER_BACK)
- The Integration Server sends the result back to the sender system or to an inbound adapter that supports acknowledgment messages (CENTRAL_BACK)

**Transport from the Receiver System to the Integration Server**

The receiver system sends an acknowledgment message back to the corresponding Integration Server, as defined [Seite 269] for the local Integration Engine.

If an outbound adapter is involved in message processing, it sends the acknowledgment message back to the Integration Server, provided that the respective adapter supports this procedure. Otherwise, a final acknowledgment message is returned with a comment that an acknowledgment is not possible.

**Transport from the Integration Server to the Sender System**

The Integration Server sends an acknowledgment message back to the sender. The name of the sender system is used for logical and technical backward routing.

If an inbound adapter is involved in processing the message, the Integration Server sends an acknowledgment message to the inbound adapter that is then responsible for returning the acknowledgment to the sender.

Provided the adapter supports this procedure, it sends the acknowledgment message back to the sender system. Otherwise, a final acknowledgment message is returned with a comment that an acknowledgment is not possible.

**Acknowledgments of Errors and Correcting Errors**

If the sender system has requested an acknowledgment message, when the response message is processed successfully, the acknowledgment message contains a corresponding comment without any additional data. However, the acknowledgment message can also inform the sender about an application or system error that occurred during processing of an asynchronous request message. The fault message is then appended as a binary attachment in the payload of the acknowledgment message.

Message processing is terminated if either request messages or acknowledgment messages contain errors. Once the error has been corrected, such messages can be restarted in the system in which the error occurred from the monitor for processed XML messages [Seite 293]. Processing of the message can also be ended manually here.

If a system error has occurred, only an acknowledgment message is generated if the configuration parameter [Seite 269] ACK_SYSTEM_ERROR, category RUNTIME is set to 1 in the system in which the error occurred.
A system error can of course also occur when processing acknowledgment messages. If this is the case, no further acknowledgment messages are created since an acknowledgment message must refer to a request message and not to another acknowledgment message.

If processing of a request message with errors is ended manually, an final acknowledgement message is generated with an appropriate comment.

Due to this procedure, multiple error acknowledgments can reach the sender before a final acknowledgment about a message being successfully processed or ended manually. For this reason, the sender must always wait for the final acknowledgment before taking active steps to correct any errors.

**Notes for the IDoc Adapter**

IDoc technology supports its own acknowledgment technology, namely the ALE Audit [Extern]. Here there are no acknowledgment requests from the sender. Instead, the receiver decides whether an acknowledgment IDoc should be sent back to the sender based on the ALE configuration. These acknowledgment IDocs are only used for monitoring on the sender side.

IDoc technology is the only technology in which positive acknowledgments from the application contain data. This data is sent by the IDoc adapter [Seite 340] in an acknowledgment IDoc if the sender is an IDoc system. Otherwise, the data is discarded.

For the inbound IDoc adapter to be able to send an acknowledgment IDoc to the sender, an IDoc end point [Seite 235] with the interface ALEAUD.ALEAUD01 must be maintained for the sender.

---

**Monitoring XML Messages**

**Purpose**

Monitoring enables system administrators to monitor their system. They can monitor the processing of XML messages [Seite 278] as well as track errors and statuses. It is also possible to monitor message throughput.

**Prerequisites**

You have called the transaction Integration Engine – Monitoring (SXMB_MONI) and the system displays the screen Integration Monitoring.

**Process Flow**

Monitoring comprises a monitor for processed XML messages [Seite 293]. It identifies and evaluates all XML messages processed by the Integration Engine [Seite 266] and produces processing statistics [Seite 299] for them. The processed XML messages can be selected according to particular criteria and displayed for evaluation purposes.

Monitoring is also used to access archived XML messages [Seite 301]. These can also be selected according to particular criteria and displayed for evaluation purposes. namely by using:

- An archive
- Message ID

In both cases the system displays a list of archived XML messages.
Monitoring also offers the following:

- An overview with corresponding information about archiving and delete jobs [Seite 273] in the Integration Engine
- Analysis of the persistence layer
- Troubleshooting methods

Choose the required function. The authorizations for the individual functions are based on the authorization object S_XMB_AUTH. They can be derived from the role SAP_XI_MONITOR and must be assigned as follows:

Monitoring
- Activity
  - 03 (display)
- Operation
  - Not defined

You require the authorization object S_XMB_DSP for monitoring because the displaying of message contents must be controlled individually. Unless the user has administrator authorization (S_XMB_ADM), the system only displays XML messages for the current client.

You cannot execute options in the menu that have the locked symbol (🔒), because you do not have the required authorization.

Logging

Use

Logging is used to log the processing steps of XML messages in the Integration Engine [Seite 266].

Integration

The logging mechanism is implemented during configuration of the Integration Engine [Seite 267]. However, you can also set it externally (using the message header or in the settings for troubleshooting [Seite 276]), as long as no setting was made during configuration.

Features

Logging logs the status of XML messages to be processed prior to the first processing step (inbound message) and after each pipeline service [Seite 267] call. The entire message, including information regarding the processing status, is saved (persisted). Persisted XML messages can then be used for evaluation purposes as part of monitoring [Seite 290].

Activities

There are multiple levels of logging available in the Integration Engine. The following logging variants are possible; the sequence of the variants displayed corresponds to the overriding hierarchy of the logging mechanism:

- You can switch logging on or off for a particular pipeline [Seite 267], or you can select Definition in Another Location, that is, at a lower hierarchy level.
  
  This is setting for the pipeline [Seite 275] itself. Logging can be activated or deactivated for the entire pipeline, which means that all service calls are either subject to logging or not.
• You can switch logging on or off for a particular pipeline element, or you can select *Definition in Another Location*, that is, at a lower hierarchy level.

This makes sense if just the individual execution steps of a pipeline are to be logged. Make a selection in the field *Logging for PL Element*.

If you activate or deactivate logging for the entire pipeline at a later point, the settings at pipeline element level have no effect.

If you choose *Definition in Another Location*, you can activate or deactivate logging using a switch in the message header.

• Logging is set externally using the message header or using the settings for error handling.

This is the lowest hierarchy level. This setting is only relevant if logging is not activated or deactivated explicitly at pipeline, or pipeline element level.

This means, for example, that no logging takes place if the logging is activated within the message, but is deactivated explicitly for the pipeline in the configuration.

If logging is set by the caller, the logging information is part of the XML message header.

  o `<SAP:Logging>1</SAP:Logging>`

  Activates logging explicitly.

  o `<SAP:Logging>0</SAP:Logging>`

  Deactivates logging explicitly.

If logging is set in the settings for troubleshooting, it is part of a conditioned action defined there.

The logging hierarchy is clarified using the following table:

<table>
<thead>
<tr>
<th>Pipeline</th>
<th>PL Element</th>
<th>Message</th>
<th>Logging</th>
</tr>
</thead>
<tbody>
<tr>
<td>yes</td>
<td>any</td>
<td>any</td>
<td>yes (for the entire pipeline)</td>
</tr>
<tr>
<td>no</td>
<td>any</td>
<td>any</td>
<td>no (for the entire pipeline)</td>
</tr>
<tr>
<td>--</td>
<td>yes</td>
<td>any</td>
<td>yes (for a selected pipeline element)</td>
</tr>
<tr>
<td>--</td>
<td>no</td>
<td>any</td>
<td>no (for a selected pipeline element)</td>
</tr>
<tr>
<td>--</td>
<td>--</td>
<td>yes</td>
<td>yes (for the entire pipeline)</td>
</tr>
<tr>
<td>--</td>
<td>--</td>
<td>no</td>
<td>no (for the entire pipeline)</td>
</tr>
<tr>
<td>--</td>
<td>--</td>
<td>--</td>
<td>no (for the entire pipeline)</td>
</tr>
</tbody>
</table>

The first three columns represent the respective logging variants with the possible settings.

• *Pipeline or PL Element* means that the setting you made at configuration time was made for the entire pipeline or for the respective pipeline element.

• *Message* means that the setting in the message header was made by the sender or using the settings for troubleshooting.

The last column specifies if logging takes place and in what form.

The following are possible settings:

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>yes</td>
<td>Logging activated explicitly</td>
</tr>
<tr>
<td>no</td>
<td>Logging deactivated explicitly</td>
</tr>
<tr>
<td>any</td>
<td>Any setting (yes or no, depending on the configuration)</td>
</tr>
<tr>
<td>--</td>
<td>No explicit settings (definition in another location)</td>
</tr>
</tbody>
</table>
Monitor for Processed XML Messages

Use
You can use the monitor for processed XML messages to display information about previously persisted XML messages that you select beforehand according to particular criteria.

You can display messages from the list individually and compare existing versions with each other.

Integration
To display information on XML messages, the messages must first have been logged. An XML message is logged when it is processed if:

- It is an asynchronous message (persisted implicitly)
- Logging for the underlying pipeline is activated (general logging of all pipeline steps or logging of a particular pipeline step)

The original message and any versions exist for each message.

Versions of an original message only exist if message processing including logging was executed, or if the message was implicitly persisted following a particular pipeline step.

- If logging is activated at pipeline level, a new version of the message is saved each time a pipeline service is called.
- If logging is activated at pipeline element level, a new version of the message is saved following processing by the pipeline element concerned.

In this way, changes to messages called by pipeline services can be monitored. Therefore, depending on the way that logging is configured (for pipeline or pipeline element), you either monitor all services or just critical ones.

Prerequisites
You have called the transaction Integration Engine – Monitoring (SXMB_MONI) and the system displays the screen Integration Engine: Monitoring.

Activities
From the menu option Monitor for Processed XML Messages, you have three different ways of selecting processed XML messages. You can do the following:

- Evaluate information for all processed XML messages
- Search for XML messages with errors only
  This enables you to quickly access all XML messages processed by the Integration Engine for which an error occurred during processing. Using the list that the system then displays, you can restart XML messages with errors.
- Identify XML messages with critical time delays
  XML messages are normally processed immediately by the Integration Engine. If this is not the case, then you can select the XML messages that were delayed during processing. The following rules apply here:
  - Synchronous XML messages are selected if they were processed 60 seconds (or longer) after they were sent.
Asynchronous XML messages are selected if they were processed 5 minutes (or longer) after they were sent.

In each of the three cases you also have the option of selecting acknowledgment messages as well. The system then displays a list of persisted XML messages.

If you do not have administrator authorization (S_XMB_ADM), the system only displays XML messages from the current client.

---

### Selecting Processed XML Messages

**Use**

You select particular messages from the list of available processed XML messages for evaluation purposes.

To condense the number of messages to be selected, use particular selection criteria, only select messages with errors, or only select messages for which the delay was relatively large between when the message was sent, and when it was executed.

**Prerequisites**

In the Monitoring [Seite 290] menu you chose the entry Monitor for Processed XML Messages [Seite 293] and the system now displays the screen with the same name.

**Procedure**

Depending on which XML messages you want to select, proceed as follows:

- To select messages according to criteria that apply to all messages, choose *All Messages* (default).
  
  Define your selection criteria and choose *Execute*. All messages that match the selection criteria are listed. If the system does not find any messages, it displays a corresponding message.

- To select messages according to criteria that only apply to messages with errors, choose *Only Messages with Errors*.
  
  Define your selection criteria and choose *Execute*. All messages with errors that match the selection criteria are listed. If the system does not find any messages with errors, it displays a corresponding message.

- To select messages according to criteria that only apply to messages with critical time delays during processing, choose *Delayed Messages*.
  
  Define your selection criteria and choose *Execute*. All messages with critical time delays that match the selection criteria are listed. If the system does not find any messages, it displays a corresponding message.

For more information about time delays, choose F1.

In each of the three cases you also have the option of selecting acknowledgment messages as well. In addition, for all three options you can choose from various standard selection criteria and advanced selection criteria. These can also be combined in any way you require.
Result
The system displays a list with the selected persisted XML messages. The list contains the
following information about the messages displayed by default:

To switch between different views, choose *Layout Settings* ( ). You can define
your preferred view as the default setting and you also have the option of
defining your own view.

- **XML message status**
  Each XML message has a status.
  If the system has not determined any versions for the XML message you selected, it
displays the overall processing status. This is the status after the last executed
processing step.
  If the system has already found the versions, it displays the status of the respective
processing step.
  Choose *Legend* ( ) to display a list of the different message statuses. For more
information see the *Processing Status* [Seite 286] section.

- **Acknowledgment message status**
  The status of the acknowledgment message is displayed for asynchronous messages
here.
  Choose *Legend* ( ) to display a list of the different acknowledgment statuses. For
more information see the *Processing Status* [Seite 286] section.

- **Date executed**
  Message processing is usually started and completed on the same day. For this
reason, only the start date is shown in the list (*Executed On*). If the end date is different
to the start date, *End Time* (see below) is set to **:**:**:**. This situation could arise if
an XML message with errors is rescheduled for the following day, after its errors have
been corrected, for example.

- **Execution Time**
  If the system has already determined versions for the selected XML message, the
display changes. *Start Time* is only displayed for inbound processing. For all other
versions the end date is displayed as the date (*Executed On*). The *Start Time* is
initialized and *End Time* is shown.
  To display the complete history of the message, click the time of the end time or
**:**:**:**.

- **Sender**
  The sender business system.

- **Namespace of the sender interface**
  The namespace of the outbound interface with which the message from the sender
was received.

- **Name of the sender interface**
  The name of the outbound interface with which the message from the sender was
received.

- **Receiver**
  The receiver business system (if one exists).

- **Namespace of the receiver interface**
The namespace of the inbound interface with which the message is sent to the receiver (if one exists).

- **Name of the receiver interface**
  The name of the inbound interface with which the message is sent to the receiver (if one exists).

- **Message ID**

- **Executed Pipeline** [Seite 267]

- **Displayed version**
  If the system has not determined any versions for the selected XML message, it displays the most recent version of the XML message with the text *Current Status*. If the system has already determined versions, it displays the corresponding processing step.

- **Message Type (Synchronous or Asynchronous)**

- **Quality of service** [Seite 279] of a message

- **Technical inbound channel of the message**
  The technical inbound channel refers to the origin of a message. The following are possible entries:
  - IENGINE
    The message originates from the Integration Engine.
  - IDOC
    The message originates from the IDoc adapter.
  - RFC
    The message originates from the RFC adapter.
  - PLAINHTTP
    The message was sent without control information and originates from a plain HTTP adapter.
  - PROXY
    The message originates from the proxy runtime.
  - AENGINE
    The message was sent in internal SAP-message format and originates from an Adapter Engine.
  - PE
    The message originates from the Process Engine.

  ![Tip](image)

  In the PE inbound channel you can go directly to the corresponding workflow display.

  In the IDOC and RFC inbound channels you can switch directly to the screen for displaying XML messages in the IDoc/RFC adapter [Seite 354].

- **Technical outbound channel of the message**
  The technical outbound channel refers to where the message is sent. The following types are possible:
  - IENGINE
    The message is going to the Integration Engine.
- **IDOC**
  The message is going to the IDoc adapter.

- **RFC**
  The message is going to the RFC adapter.

- **PLAINHTTP**
  The message is sent without control information and is going to a plain HTTP adapter.

- **PROXY**
  The message is going to the proxy runtime.

- **AENGINE**
  The message is sent in internal SAP-message format and is going to an Adapter Engine.

- **PE**
  The message is going to the Process Engine.

In the PE outbound channel you can go directly to the corresponding workflow display.

In the IDOC and RFC outbound channels you can switch directly to the screen for displaying XML messages in the IDoc/RFC adapter [Seite 354].

- **Status of the message in the technical outbound channel**
  If the message is going to an IDoc or RFC adapter (outbound channel IDOC or RFC), this column displays the status of message processing in the corresponding adapter. The display remains empty for messages with errors.

  Choose Legend (button) to display a list of the different message statuses.

- **Queue name (ID), in which the message is currently located (only in asynchronous processing).**
  To navigate to the qRFC monitor for the corresponding queue, click a queue ID.

- **Message status in displayed queue**
  Choose Legend (button) to display a list of the different message statuses.

- **The reorganization status of the message**
  The reorganization status of a message specifies whether the message was marked for deletion or whether it has already been archived. If the display is empty, the message status is initial.

  Choose Legend (button) to display a list of the different message statuses.

- **The message ID of the original message if it is an acknowledgment message - the request message of which was subject to message branching following receiver identification [Seite 284] (only for acknowledgment messages).**

- **The message ID of the request message that lead to the acknowledgment message (only for acknowledgment messages).**

- **The message ID of the original message when the message is the result of message branching following receiver identification.**

The system lists XML messages in chronological order. You can display [Seite 302] each of the listed XML messages in the following ways:

- **Click on the corresponding message ID**
• Double click on the contents of another column provided that the contents has no link.

You also have the following options:

• To compare and display the different versions of an XML message, select the line with the corresponding message ID and choose Display [Seite 302].

• To display the error long text for an XML message with errors, select the line with the message in question and choose Error Information, or choose the symbol in the status column. The error long text contains possible error causes as well as information for troubleshooting.

  ![Error Icon]

  This option is only relevant for XML messages with errors.

• To update the display, choose Refresh.

  The system determines the current status of the persistency layer and any new XML messages written to the persistency layer are sorted according to the time they were sent.

  The versions are updated for XML messages with version display.

• To check and update the status of messages in an outbound adapter or in a queue, choose Refresh Status.

  You can check and update the status of messages of this type until they have been processed.

• To reschedule a message with errors having corrected the error, select the corresponding line and choose Restart. The system displays a confirmation prompt.

  To reschedule numerous messages of this kind, use the program RSXMB_RESTART_MESSAGES for restarting messages with errors.

  ![Confirm Icon]

  These options are only relevant for asynchronous XML messages with errors.

• To cancel further processing of an XML message with errors, select the corresponding line and choose . The message can then be either archived or deleted.

  The system displays a confirmation prompt.

  ![Confirm Icon]

  This option is only relevant for asynchronous XML messages with errors.

• To display the versions of a particular XML message, select the corresponding line and choose Expand.

  The versions available including version number and action executed are expanded beneath the selected line. If no versions exist, the system displays a corresponding message.

  To display any of the XML messages listed, click the corresponding message ID.

  ![Confirm Icon]

  Versions of an XML message only exist if message processing, including logging, was executed.

• To find versions for all XML messages in the list, choose Expand All Messages.

  The system displays existing persisted messages and sorts them according to their message ID and version.
To hide the versions again, choose **Hide Versions** (/tcpell/).
To display the history of a particular message, select the corresponding line and choose **Message History** (/tcpell/).
The system displays information on the life cycle of the XML message in a separate dialog box.

### Processing Statistics

The processing statistics provide you with the following:

- Up-to-date information about the number and status of XML messages processed by the Integration Engine [Seite 266].
- Performance analysis for message processing for each pipeline [Seite 267] processing step.

### Message Status and Number of Messages

An XML message can have one of the following statuses:

- **XML messages with errors**
  Number of XML messages for which an application error or system error occurred during message processing.
  Choose **Goto → Display Error** to switch to the screen for displaying messages with errors [Seite 294] in the technical monitor. Here you can access information about the errors that have occurred.

- **Not yet processed XML messages**
  Number of listed XML messages that are waiting to be processed by the Integration Engine. These XML messages are saved in their original version in the database and still have their original status.

- **XML messages being processed**
  Number of XML messages that are currently being processed by the Integration Engine (current workload).

- **Correctly processed XML messages**
  Number of error-free, entire XML messages processed by the Integration Engine

- **XML messages flagged for archiving**
  Number of correctly processed XML messages that have reached the end of their retention period and which must be archived because the corresponding **interfaces are defined for archiving** [Seite 271]. These XML messages are archived [Seite 273] with the next archiving run.

- **Archived XML messages**
  Number of XML messages that have already been archived. You can read [Seite 301] and display [Seite 302] these XML messages from the archive.

- **XML messages flagged for deletion**
  Number of correctly processed XML messages that are not to be archived due to interfaces determined for archiving and that have reached the end of their retention period. These XML messages are deleted [Seite 273] from the database with the next deletion job.
• Deleted XML messages
  Number of XML messages that have already been deleted. You can no longer access these XML messages.

The system also calculates the average, shortest and the longest processing time of an XML message. The processing time is seen as the period between when the original message was written to the database and when the last status (Correctly Processed or Processed with Errors) was set.

The system also calculates the average, smallest and largest number of XML messages processed by the Integration Engine each day, as well as the number of XML messages processed on the current date.

**Performance Analysis**

In addition to the message status, time required for processing, and number of messages, the processing statistics also provide you with performance data for the individual pipeline processing steps.

To be able to display performance data, you must first have scheduled [Extern] the evaluation jobs SAP_XMB_PERF_COPY and SAP_XMB_PERF_REORG, and [Seite 269] set the corresponding configuration parameters.

The performance data appears in a list and corresponds to the values 30 minutes ago.

**Performance Data Table**

<table>
<thead>
<tr>
<th>Column</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pipeline ID</td>
<td>Logical ID for the pipeline, for example <em>SENDER</em></td>
</tr>
<tr>
<td>Processing Step</td>
<td>Name of the processing step concerned in the pipeline displayed.</td>
</tr>
<tr>
<td>Minimum (s)</td>
<td>Shortest processing time for a message in seconds.</td>
</tr>
<tr>
<td>Average (s)</td>
<td>Average processing time for a message in seconds.</td>
</tr>
<tr>
<td>Maximum (s)</td>
<td>Longest processing time for a message in seconds.</td>
</tr>
<tr>
<td>Throughput (msg/s)</td>
<td>Number of messages processed per second.</td>
</tr>
<tr>
<td>Throughput (KB/s)</td>
<td>Throughput in kilobytes per second.</td>
</tr>
</tbody>
</table>

- To update the data displayed as required, choose **Statistics → Refresh**.
- To display further details about a particular processing step, choose **Details** in the corresponding line.

The system displays a screen where you can specify the time and components for the performance analysis. Here you can change the settings for the time selection as well as the pipeline set as a result of your specifications.

- Make your entries and choose **Execute**. The system displays the screen **Integration Engine: Performance Statistics**, which displays the values that correspond to your selections.

In addition to the columns listed above, data is also displayed for the processing period (from/to) and the number of messages processed during this period. To refresh the values displayed, choose **Statistics → Refresh**.
Selecting Archived XML Messages

Use
You search for archived XML messages to display the contents of the messages found and to compare the different message versions.

Prerequisites
To search for archived XML messages, you must have already archived some XML messages. To archive XML messages, you must first define the interfaces [Seite 271] whose messages you want to archive. You then need to schedule corresponding archiving jobs [Seite 273].

You have called the transaction Integration Engine – Monitoring (SXMB_MONI) and the system displays the screen Integration Engine: Monitoring.

Procedure
Depending on how you want to select the archived XML messages, proceed as follows:

- To display archived XML messages from an archive, choose Archived XML Messages (Search Using Archive).
  The system displays a list of all executed archiving runs. If no archived XML messages exist, the system displays a corresponding message.
  - To display detailed information for a particular archiving run, select the corresponding archiving run by double clicking it. The system displays details about the run in question in another window.
  - To display archive files created in a particular archiving run, expand the archiving run in question. The system displays a list of all archive files created in that archiving run.
  - To display detailed information for a particular archive file, select the corresponding archive file by double clicking it. The system displays details about the archive file in question in another window.
  - If you want to display XML messages archived in a particular archiving run or saved in a particular archive file, select the corresponding archiving run or the corresponding archive file and choose Continue.
    On the screen XML Message Processing, the system displays a list of all XML messages archived during the selected archiving run or in the selected archive file (including the pipeline [Seite 267] responsible for processing), sorted according to date and time of processing.

- To display archived XML messages using the message ID, choose Archived XML Messages (Search Using ID).
  On the screen XML Message Processing, the system displays a list of all archived XML messages, (including the pipeline responsible for processing), sorted according to date and time of processing. If no archived XML messages exist, the system displays a corresponding message.

- To display the listed archived XML messages and to compare the existing versions, select the relevant line and choose Display Archived Versions [Seite 302] or click the relevant message ID.
If you do not have administrator authorization (S_XMB_ADM), the system only displays XML messages from the current client.

**Displaying XML Message Versions**

**Use**

You can display individual XML messages for evaluation purposes from a list of processed [Seite 294] or archived [Seite 301] XML messages and compare existing versions with each other.

By comparing versions of individual XML messages you can see what changes to the XML message were made by the individual processing steps of a pipeline [Seite 267]. You can compare entire XML messages and parts of a message.

**Prerequisites**

You chose the corresponding options in the monitoring [Seite 290] menu and the system now displays a screen with a list of processed or archived XML messages.

**Procedure**

1. Click the message ID of the XML message for which you want to display the message versions, or select the corresponding line and choose Display or Display Archived Versions.

   The system navigates to the screen Display XML Message Versions or Display Archived XML Message Versions. It contains a list of all existing message versions, including their status in the navigation area and the current message version in the split display area.

   Choose Legend to display a list of the different message statuses. For more information see the Processing Status [Seite 286] section.

   The upper display area (window 1) contains the message header and body. The lower display area (window 2) contains the message payload (or the error header for messages with errors).

   If the payload is too large to be displayed, the lower display area remains empty and a corresponding message is displayed.

   The individual parts of the XML message are grouped together and displayed as one document in window 1. <SOAP:Envelope> is inserted as the surrounding tag. This makes the XML document syntactically correct but it is not part of the actual XML message.

   Depending on whether you are displaying an original message or a message version, the system displays Inbound Message or the description of the pipeline processing step that generates the version, as a comment in the XML message. This information is also not part of the actual XML message.

   The Inbound Message identifies the first processing step in the respective Integration Engine [Seite 266] at this point.
2. To compare complete XML message versions, select the line *Inbound Message* or the name of the respective processing step, for example *Response* and specify for each selection whether the system should display the message in the upper part of the screen (*Display Contents in Window 1*) or in the lower part of the screen (*Display Contents in Window 2*).

The display containers are not mutually exclusive. This means that you can display parts of messages in the containers that cannot be compared.

3. To compare the header, body or payload of a message, double click the corresponding line under *SOAP Header*, *SOAP Body* or *Binary Payloads* in the navigation area. Alternatively, select the line and choose *Display Contents in Window 1* or *Display Contents in Window 2*, depending on whether you want to display the parts at the top or the bottom of the screen.

4. To display a complete XML message, double click the corresponding line in the navigation area or select the line and choose *Display Contents in Window 1* or *Display Contents in Window 2*.

5. To save the contents of a window to your local file system as a file, choose *Download Window 1* or *Download Window 2*.

To display the contents of a window in unformatted XML format in your notepad, choose *View Source* from the context menu.

6. To save the entire XML message (including the payload) to your local file system, choose *Save Complete XML Message to File*.

7. To display the error long text for an XML message with errors, choose *Further Information on Error*. The error long text contains possible error causes as well as information for troubleshooting.

This option is only available for XML messages with errors.

8. To edit a message with errors to correct the error, choose *Edit Message Part*.

This option is only available for asynchronous XML messages with errors.

9. To reschedule a message with errors having corrected the error, choose *Reschedule XML Message After Error*.

This option is only available for asynchronous XML messages with errors.

10. To cancel the further processing of an XML message with errors, choose 🗑️. The message can then be either archived or deleted [Seite 273].

This option is only available for asynchronous XML messages with errors.

You require the authorization object S_XMB_DSP here because the displaying of message contents must be controlled individually. Unless the user has administrator authorization (S_XMB_ADM), the system only displays XML messages for the current client.
Proxy Runtime

Use

Using the proxy generation [Seite 186] function you can generate proxies for communicating across SAP Exchange Infrastructure. The proxies encapsulate the communication with the Integration Engine. Using a proxy that you have generated either for the outbound or the inbound side, you can exchange messages between various communication partners [Seite 121]:

<table>
<thead>
<tr>
<th>Runtime Component: Sender Side</th>
<th>Runtime Component: Receiver Side</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABAP or Java Outbound Proxy</td>
<td>ABAP or Java Inbound Proxy (J2EE)</td>
</tr>
<tr>
<td>ABAP or Java Outbound Proxy</td>
<td>Adapters</td>
</tr>
<tr>
<td>Adapters</td>
<td>ABAP or Java Inbound Proxy (J2EE)</td>
</tr>
</tbody>
</table>

You can also exchange messages between the various adapters. However, this section only contains information about the proxy runtime.

Prerequisites

To communicate with the Integration Engine, the generated proxy objects call proxy runtime methods. Therefore, the task of the proxy objects is simply to forward the data transferred during the call (for example, received data) to the proxy runtime. This component is available as part of SAP Exchange Infrastructure from Release 2.0. It must be installed on the system that is connected to the Integration Engine by the means of proxies. This means that for Java applications the corresponding libraries for the Java proxy runtime (JPR) must be compiled together with the application. For more information, see the SAP Exchange Infrastructure installation guide.

Features

The proxy runtime encapsulates all functions of the generated proxies at runtime. This includes the following:

- Generating an outbound XML message or parsing an inbound XML message and calling the implementing class in the inbound case.

- Forwarding the XML message to the Integration Engine or receiving the inbound XML message.

Further connection properties are configured in the Integration Directory first (for example, logical receiver systems, mappings, technical end points). Also see: Runtime [Seite 264].

For more information see:

| ABAP Runtime [Seite 305] | Java Runtime [Seite 322] |
ABAP Proxy Runtime

Use

To send a message using a proxy [Seite 305], you simply need to call an outbound proxy. On the inbound side, you can receive the message [Seite 306] using an ABAP object interface.

Features

You can do the following:

- Trigger exceptions for application errors in synchronous and asynchronous cases (see: Error Handling [Seite 306]).
- Determine whether the default values are to be used for parameters or whether initial fields are to be indicated as such (see: Processing Default and Initial Values [Seite 313]).
- Use special proxy runtime services [Seite 311] to append attachments to a message, to query the payload, or to identify receivers in advance.

Sending a Message

Use

To send a message using the Exchange Infrastructure, call the corresponding outbound proxy.

Procedure

1. Declare variables for error handling, the outbound message, and (in the synchronous case) the inbound message:

   Data:
   - l_sys_exception type ref to cx_ai_system_fault,
   - l_output type <Output Messagetype>,
   - l_input type <Input Messagetype>.

2. Complete structure l_output for the output message.

3. Call your outbound proxy and catch at least the exception cx_ai_system_fault. This is sensible in both synchronous and asynchronous cases:

   try.
   call method <Generated Outbound Class> => EXECUTE_SYNCHRONOUS
       exporting
       output = l_output
       importing
       input  = l_input.
   catch cx_ai_system_fault into l_sys_exception.
   write: 'System fault: '.
   write: l_sys_exception->errortext.
   exit.
   endtry.
4. In the asynchronous case, a message is first sent with the statement `COMMIT WORK`:

```
commit work.
```

In the synchronous case, the call is executed immediately. To avoid inconsistencies, use asynchronous communication for updates.

---

**Receiving a Message**

**Implementing the Inbound Side**

The application implements a class for the generated ABAP object interface that you must enter in transaction `SPROXY` on the tab page `Properties`. If the Integration Engine receives an XML message, it forwards it to the proxy runtime. The proxy runtime imports the message and fills the ABAP data types of the generated ABAP object interface with the message data. The Proxy Framework then calls the inbound interface method.

Therefore, the application must simply implement the class for the inbound interface. In addition to the ABAP object interface, the generation function also offers to generate a class. The name of this class is located on the tab page `Properties`. The application can, however, also enter any class with a suitable signature.

```
On the receiver side, the Integration Engine triggers a commit work.
```

---

**Error Handling**

<table>
<thead>
<tr>
<th>ABAP</th>
<th>Java</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[Seite 330]</td>
</tr>
</tbody>
</table>

**Use**

The following errors can be handled by the Exchange Infrastructure:

- **System errors**, which are triggered by the Exchange Infrastructure. These might be errors during transfer, due to a failed server, for example.

- **Application errors**, which are triggered or handled by the application alone. An example would be a request for data about a material that is unknown in the receiver system.
Error handling is essentially of interest in the synchronous case. In this case, an application can report application errors to the caller application. In the asynchronous case, you can catch an error that has occurred during transfer and, on the inbound side, forward an error to monitoring.

### Features

#### Handling System Errors

You can catch errors that occur when transferring a message, using the exception class `CX_AI_SYSTEM_FAULT`. You distinguish system errors using error codes; these are managed and documented centrally for all systems. Using the attributes `CODE` and `ERRORTEXT` of the exception class, the application can display an error message.

#### Handling Application Errors

In the Integration Repository, you can define fault messages to handle application errors. The proxy generation functions then generate the exception class (prefix `CX_`). Using fault messages, you can handle errors triggered by the called application or persist them for monitoring purposes:

<table>
<thead>
<tr>
<th>Message Interface</th>
<th>Options for Error Handling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synchronous Communication (Inbound/Outbound)</td>
<td>Triggering of an error at the receiver that is providing a service; handling of the error at the sender that called the service.</td>
</tr>
<tr>
<td>Asynchronous Inbound Interface</td>
<td>Fault messages are not used here to handle a error at the sender. The fault message generated by the triggered error is persisted for monitoring instead.</td>
</tr>
</tbody>
</table>

Exception classes for application errors are derived from the basis exception class `CX_AI_APPLICATION_FAULT`. You can use this exception class to determine an error situation, independently of the exact error, or in a `CASE` branch, you can use it to catch all the errors that have not yet been handled.

### Activities

1. To handle an application error, create fault messages for your message in the Integration Repository.
2. In each case, handle system errors, and, if necessary, application errors (see: [Reporting and Handling Errors](Seite 309)).

### Fault Messages

#### Definition

A fault message is a message that describes an application-specific error situation.
Use

The proxy generation functions generate exception classes for fault messages in the Integration Repository. If an application triggers an exception using an exception class such as this, the proxy runtime automatically converts it to a fault message. Using the fault message, the application records an application error (for example, Requested Customer Profile Unknown).

Structure

The exception class generated from the fault message can give you information about the error cause using the following structure:

- **standard**
  - faultText: Brief description of error
  - faultURL: Link to additional documentation
  - faultDetail (Table): Detailed information (table)
  - severity: Error class
  - text: Short text
  - id: Unique error code
  - url: Link to long text

- **addition**
  - Additional Application-Specific Information

The defined structure **STANDARD** contains fields essential for forwarding the error. The structure **ADDITION** can be used freely by the application.

Example

To send a fault message, complete the generated structure for fault messages and trigger the exception with the corresponding exception class:

```plaintext
data: l_standard_data     type <Standard Fault Data Type>,
       l_detail_data       type <Detailed Structure for Standard Fault Data Type>,
       l_additional_data   type <Application Error Data Type>.

l_standard_data-fault_text   = <Error Text>.
l_standard_data-fault_url    = <Error URL>.
l_detail_data-severity       = <System Error Category>.
l_detail_data-text           = <System Error Text>.
l_detail_data-id             = <System Error Id>.
l_detail_data-url            = <System Error URL>.
append l_detail_data to l_standard_data-fault_detail.
l_additional_data  = <Application Error Information >
RAISE EXCEPTION TYPE <Name of Exception Class>
        EXPORTING
            standard = l_standard_data
            addition = l_additional_data.
```

If you use the application log, or have access to error information from a BAPI return table, you can also use these directly to complete the fault message. In this case, you only need to call the method `CL_PROXY_FAULT=>RAISE` of the proxy runtime to trigger the error:
CALL METHOD cl_proxy_fault=>raise
EXPORTING
  exception_class_name = <Name of Exception Class>
  application_log_handle = < Application Log: Handle of a Protocol>
  bapireturn_tab = < Table with BAPI Return Information>.

If you transfer neither a handle of an application log, nor a table with BAPI return information, the method completes the standard data of a fault message as follows:

<table>
<thead>
<tr>
<th>faultText</th>
<th>Fault message text in the Integration Repository (shortened to 60 characters).</th>
</tr>
</thead>
<tbody>
<tr>
<td>faultUrl</td>
<td>URL on the long text of this message in the Integration Repository.</td>
</tr>
</tbody>
</table>

The following additional fields are completed for the data from the application log, or the BAPI return table:

<table>
<thead>
<tr>
<th>faultDetail-severity</th>
<th>The ABAP message type is mapped to this field as follows:</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABAP Feld MSGTY</td>
<td>Fault Message Field</td>
</tr>
<tr>
<td>faultDetail-severity</td>
<td>faultDetail-severity</td>
</tr>
<tr>
<td>A (Abend)</td>
<td>error</td>
</tr>
<tr>
<td>E (Error)</td>
<td>error</td>
</tr>
<tr>
<td>W (Warning)</td>
<td>warning</td>
</tr>
<tr>
<td>S (Success)</td>
<td>information</td>
</tr>
<tr>
<td>I (Information)</td>
<td>information</td>
</tr>
</tbody>
</table>

| faultDetail-text        | Formatted T100 message text (in the language in which the service was executed). |
| faultDetail-url         | URL on the T100 long text (if available).                                        |
| faultDetail-id          | Connection from work area and message number (for example, “XY(007)”).          |

**Representing the Data**

In the asynchronous case, you can see a fault message in the monitoring of the Exchange Infrastructure. For example, in technical monitoring you can display the fault message in a more readable format by choosing Further Information on Error. In the synchronous case, you can integrate a format of this type into your application on the caller side by using the method CL_PROXY_FAULT=>SHOW.

---

**Reporting and Handling Errors**

**Purpose**

The options for error handling depend on the type of communication chosen:

- *Asynchronous Communication*: On the outbound side, the application can only handle errors that occur when the message is being sent (for example, the outbound queue is...
full). These errors can be caught and persisted using the exception class `CX_AI_SYSTEM_FAULT`. On the inbound side, as in the synchronous case, you can define fault messages [Seite 307], which are then only forwarded to monitoring (and not to the sender).

- **Synchronous Communication:** Using fault messages you can handle errors that have occurred on the inbound side. To confirm an error, the proxy runtime sends the fault message from the receiver back to the sender, which can then respond to it.

### Process Flow

In the graphic below, an error from the receiver system is reported to the sender system in synchronous communication:

1. If an application on the inbound side triggers an error, the proxy runtime converts the corresponding exception class to a fault message and transfers it to the sender system using the Integration Server.
2. The proxy runtime in the sender system interprets the fault message and triggers the exception for the outbound interface.
3. In the sender system, the application can catch and handle the exception using a Try/Catch block. Using the super exception class `CX_AI_APPLICATION_FAULT` you can – in a similar way to the statement `WHEN OTHERS` in a CASE statement – catch all remaining application errors.
Special Services of the Proxy Runtime

Use

Using the documentation above, and a correspondingly configured connection, you are now able to exchange messages using the Exchange Infrastructure from the perspective of the proxy runtime (the interfaces used to exchange messages are maintained in the Integration Directory [Seite 21] and not in the application program). However, there are other services that are required in particular application scenarios. Services of this type are described below.

Features

There is a standard procedure for default and initial values in the message, which you can override. See: Processing Default and Initial Values [Seite 313].

Moreover, you can have the system return objects using the factory class [Seite 311] of the proxy runtime; you can also use these objects to influence the runtime response in the application program:

- You can call payloads [Seite 15] in the proxy runtime after an outbound proxy has been called, or while an inbound proxy is being called, for example, because you want to archive the payloads. See: Calling the Payload [Seite 314].
- Instead of determining the logical receiver system using routing, you can specify the receiver business system for the outbound proxy. See: Setting the Receiver System [Seite 315].
- You can append an attachment to the message at the sender [Seite 316], which you can then call at the receiver [Seite 318] (and the other way around).
- For asynchronous messages, you can ensure that messages sent with a commit work arrive at the receiver in the same sequence that they were sent from the sender (compare with qRFC). See: Guaranteeing the Sequence of Asynchronous Messages [Seite 319].

The Factory Class

Use

Assume that you want to append an attachment to your message. The proxy runtime must take this into account when constructing the message. However, if you want to identify the receiver of a message in advance, you can only find this information if you execute logical routing for the maintained data. You must direct any queries to the respective components shortly before the message is sent.

The proxy runtime provides you with the factory class CL_AI_FACTORY, which you can use to generate objects that enable you to use the special services of the proxy runtime [Seite 311].

You do not need a factory class to process default and initial values [Seite 313].
Prerequisites
To be able to use the factory class, you must first load its class definition:

```plaintext
class cl_ai_factory definition load.
```

Features
Using the factory class, you can generate the following objects:

<table>
<thead>
<tr>
<th>Object</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Controller Object</td>
<td>Using this object, you can do the following: Set the receiver [Seite 315]; Call the payload [Seite 314] (after a message is sent or received); Append attachments [Seite 316] to the message or remove them when the message is received [Seite 318]; guarantee the sequence of asynchronous messages [Seite 319]. You can transfer the controller object to your outbound proxy using the parameter CONTROLLER, or access it in the inbound proxy, using the same parameter.</td>
</tr>
<tr>
<td>Controller Object (implements the interface IF_AI_POSTING_CONTROLLER)</td>
<td></td>
</tr>
<tr>
<td>Routing API</td>
<td>Using this API you can determine the logical receiver in logical routing in advance. For more information see Receiver Pre-Identification [Seite 320].</td>
</tr>
<tr>
<td>Routing API (implements the interface IF_ROUTING_LOCAL_RUNTIME)</td>
<td></td>
</tr>
<tr>
<td>Attachment Object</td>
<td>An attachment object that you can append to the message using a controller object.</td>
</tr>
<tr>
<td>Attachment Object (implements the interface IF_AI_ATTACHMENT)</td>
<td></td>
</tr>
<tr>
<td>'Preliminary Payload'</td>
<td>Return of a payload after a simulated proxy call so that you can access the payload before a message is sent. See: Calling the Payload [Seite 314].</td>
</tr>
</tbody>
</table>

Activities

Creating a Controller Object
```plaintext
data: i_controller type ref to if_aiPosting_controller.
i_controller = cl_ai_factory=>create_controller( ).
```

Creating a Routing API
```plaintext
data: i_routing_runtime type ref to IF_ROUTING_LOCAL_RUNTIME.
i_routing_runtime = cl_ai_factory=>create_routing_runtime( ).
```

Creating an Attachment Object
To generate an attachment object, use the method `create_attachment_from_binary()` of the factory class. (See: Appending Attachments at the Sender [Seite 316].)

Creating the Payload Before Sending
To create the payload of a message, use the method `create_parameter2xml()` of the factory class. (See: Creating a Payload [Seite 314].)
Processing Default and Initial Values

Use

For an initial interface parameter in ABAP, you do not know whether the parameter was given a value or not. In addition, there are XSD standards that cover default values. The following standard procedure exists for proxy generation:

Standard Procedure for Default Handling

<table>
<thead>
<tr>
<th>Outbound Interface</th>
<th>When generating the XML data stream, the default value is set in all initial fields.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inbound Interface</td>
<td>Inbound processing of the proxy framework sets empty and missing tags to the default value before the application is called.</td>
</tr>
</tbody>
</table>

To improve performance, optional, initial, scalable elements with a default value are suppressed in the message.

You can override this standard procedure.

Prerequisites

If you choose to override the standard procedure this will affect system performance. This is because the additional information must be evaluated when you construct the message or it must be supplied during parsing. For this reason, both outbound and inbound proxies have the attribute `extended_xml_handling`:

- In the outbound case, to inform the proxy runtime you want to override the standard procedure, set the attribute before you call the outbound method (`extended_xml_handling = 'X'`). For fields where you want to override the standard procedure, complete a Control Table (as explained in the following section). Using the control table, you can determine whether a value should occur in the message payload, for example.

  ```
  extended_xml_handling = 'X'
  ```

- Regardless of whether you set the attribute `extended_xml_handling` at the sender or not, at the receiver, you can set the proxy runtime to complete the control table for fields. For this, set the initial value of the attribute `extended_xml_handling` in the class interface of the inbound proxy statically to `X`. The proxy runtime then determines when parsing messages to this proxy, whether the field was transferred to the message, whether the field contained an initial value, or whether it had no value at all. The proxy runtime correspondingly completes a control table that can be evaluated by the application.

The attribute is not set in both cases in the default setting (`extended_xml_handling = space`).

Controlling at the Sender

The ABAP proxy generation functions generate control tables of type PRXCTRLTAB (line type PRXCTRL) for each complex type. This component has the name `CONTROLLER` (not to be confused with the parameter `CONTROLLER` of the method `EXECUTE_SYNCHRONOUS` or `EXECUTEASYNCRONOUS`). Using this table you can define for each individual field whether
it is to occur in the message, and how. For a table line, set the field FIELD to the ABAP field names, the field VALUE to a value from the type pool sai, and attach the line to the table. The field VALUE can have the following values:

**Value Range for Field VALUE of Table CONTROLLER**

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>sai_ctrl_initial</td>
<td>Field appears with type-correct initial value in the message; it controls default handling and performance optimization. Value is ignored for structures or complex types.</td>
</tr>
<tr>
<td>sai_ctrl_nil</td>
<td>The value xsi:nil is sent in the message for this field.</td>
</tr>
<tr>
<td>sai_ctrl_none</td>
<td>Field does not occur in message, but independently of this value and the default.</td>
</tr>
</tbody>
</table>

**Interpretation of the Control Table on the Receiver Side**

On the inbound side as well, the control table is generated in the same way as every other structure. In this case, the proxy runtime completes the control table during parsing with values that the application can then evaluate.

**Possible Values of the Field VALUE or the CONTROLLER table on the Inbound Side**

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>sai_ctrl_initial</td>
<td>Initial value is sent for the field.</td>
</tr>
<tr>
<td>sai_ctrl_nil</td>
<td>The value xsi:nil was sent in the message for this field.</td>
</tr>
<tr>
<td>sai_ctrl_none</td>
<td>Field not part of the message.</td>
</tr>
</tbody>
</table>


**Creating a Payload**

**Use**

In both the inbound and outbound cases it is useful to be able to access the payload. Note the following two cases:

- You only need the payload after you have sent the message, or once you have received it (for example, to archive the sent message).
- Before you send the message, you need to know what the payload looks like (for example, because you want to identify the receiver in advance and the receiver depends on data in the payload).

**Activities**

**Calling the payload after you have sent or received a message**

You can have the system return the payload of the message on both the inbound and outbound side. Use the controller object methods to do this:
1. Using the factory class [Seite 311] **CL_AI_FACTORY**, generate a controller object.

2. Once you have called an outbound proxy, use the methods **GET_OUTPUT_PAYLOAD** and **GET_INPUT_PAYLOAD** of this object, to make the system return the payload of the output and the input message (the latter is only relevant for synchronous outbound interfaces).

3. You can access the payload of the input message as part of inbound processing. Use the interface controller parameter to fetch the payload using the method **GET_INPUT_PAYLOAD**. To call the payload for the outbound message, use the method **create_parameter2xml()** of the factory class (see the next section).

**Calling the payload before sending a message**

You can create the payload of a message using the method **create_parameter2xml()** of the factory class **CL_AI_FACTORY** (also see report **SEI_PROXY_01**).

---

**Setting the Receiver System**

**Use**

When you create outbound and inbound interfaces in the Integration Builder, do not assign them to each other. You configure which outbound interfaces send messages to which inbound interfaces in the Integration Builder (see: **Configuration [Seite 207]**).

However, in some cases, you know the receiver of a message because of the application data. For this reason, the proxy runtime gives you the option of setting the logical receiver system before the message is sent. This means that the entire routing logic does not have to run. Since all connections have to be completed during configuration, you still have to configure a connection in the Integration Directory even if you set the receiver.

If this were not the case, connections would exist that were only visible in program code. You must be able to access information of this type from a central point, however.

**Procedure**

Each generated outbound method has a component **CONTROLLER** of type **IF_AI_POSTING_CONTROLLER** as a parameter so that you can set the receiver directly. An instance of this interface is also referred to as a Controller Object (see: **Runtime APIs [Seite 311]**).

This parameter is not visible on the tab page Structure; call transaction **SPROXY** to access the parameter. To see the exact signature of the outbound method and the parameter **CONTROLLER**, navigate in the Class Builder (**SE24**).

**Example**

The following example illustrates how to set a receiver:
* Factory class to create controller instance
class cl_ai_factory definition load.
* Variables to identify receiver systems
data: l_receiver   type ait_rcvr,
   lt_receivers type rmt_recsys.
* Define 'l_input' and 'l_output' according to
* the generated Dictionary types for your message types
[...]
* Controller class
data: l_controller type ref to if_ai_posting_controller.
* Superclass for all global exceptions
data: l_exception  type ref to cx_root.
try.
* Create instance of controller
   l_controller = cl_ai_factory=>create_controller( ).
* In this example, we only specify one receiver system as the destination:
   l_receiver = 'TravelAgencyBCE'.
   append l_receiver to lt_receivers.
* Pass receiver system table to proxy runtime
   l_controller->set_to_parties( lt_receivers ).
* Set OUTPUT parameter with data for body of message
[...]
* Call outbound proxy
   call method co_flight_booking_reserve_out=>EXECUTEASYNCHRONOUS
      exporting
           controller   = l_controller
           output       = l_output
      importing
           input       = l_input.
* For asynchronous calls the message is sent after commit work:
  commit work.
* Error handling
  catch cx_ai_system_fault into l_exception.
* Handle your error
endtry.

Appending Attachments at the Sender

Use
An attachment comprises any data (for example, text or graphics) that can be appended to a message. In principle, you can append an unlimited number of attachments to an output message of an outbound or inbound interface (in the synchronous case) using the controller object [Seite 311].
Note that not all receivers of a message can understand attachments and therefore some receivers will ignore them. The proxy runtime (ABAP/Java) can process attachments, however the majority of the adapters cannot.

**Procedure**

1. Declare a variable for the attachment object, a table for the transfer of one or more attachments to the controller object, and variables that specify the name, contents, and the type of attachment. To create the attachment object you require the factory class

   ```
   data: l_attachment type ref to if_ai_attachment,
   lt_attach type prx_attach,
   l_name type string,
   l_xstring type xstring,
   l_type type string.
   ```

2. Assign `l_name` to the name, `l_xstring` to the contents of the attachments in binary format, and `l_type` to the type (for example, "text/html"). The interface `IF_AI_ATTACHMENT` provides a selection of constants for this purpose. If the type in question is not yet available as a constant in the interface, you can also assign `l_type` to any other string.

3. Using the factory class `CL_AI_FACTORY`, you can now generate an attachment instance and append it to the attachment table `lt_attach`:

   ```
   l_attachment = cl_ai_factory=>create_attachment_from_binary(
     p_data = l_xstring
     p_type = l_type
     p_name = l_name ).
   append l_attachment to lt_attach.
   ```

   The factory class also provides the method `create_attachment_from_text()` for pure text attachments; you can append the attachment to this method as `STRING`.

4. To generate additional attachments, repeat the procedure described in the last two steps.

5. Transfer your attachments using the method `set_attachments()` of the controller object:

   ```
   l_controller = cl_ai_factory=>create_controller( ).
   l_controller->set_attachments( lt_attach ).
   ```

6. Transfer the controller object when you call your proxy.

   In the synchronous case, you can also set attachments at the receiver. Once you have fetched input message attachments, you can set attachments for the output message, using `set_attachments()`. Since only one controller object exists for the inbound interface, you overwrite the attachments of the input message, however.
Querying Attachments at the Receiver

Procedure

Using the controller object method `get_attachments()` you can fetch attachments on the receiver side:

```plaintext
data: lt_attach    type prx_attach,
  l_name       type string,
  l_xstring    type xstring,
  l_string     type string,
  l_type       type string,
  l_attachment type ref to if_ai_attachment.

lt_attach = controller->get_attachments( ).
loop at lt_attach into l_attachment.
  l_type = l_attachment->get_kind( ).
  if l_type = IF_AI_ATTACHMENT=>C_ATTACH_TYPE_BINARY.
    l_name = l_attachment->get_document_name( ).
    l_xstring = l_attachment->get_binary_data( ).
  * Do whatever you want with your binary attachment
  else.  " l_type eq IF_AI_ATTACHMENT=>C_ATTACH_TYPE_TEXT.
    l_name = l_attachment->get_document_name( ).
    l_string = l_attachment->get_text_data( ).
  * Do whatever you want with your text attachment
  endif.
endloop.
```

Once you have fetched the input message attachments, you can set attachments for the output message in the synchronous case, using `set_attachments()` . Since only one controller object exists for the inbound interface, you overwrite the attachments of the input message. If you do not want to return any attachments, call the method `reset_attachments()` , so that the system deletes received attachments and does not return them.
Guaranteeing the Sequence of Asynchronous Messages

Use
To ensure that asynchronous messages arrive at the receiver in the same sequence that they were sent, you can give the outbound proxy a String. All messages with the same string then arrive at the receiver in the sequence in which they were sent.

At present there are no conventions for this string and so nothing can stop different components from using the same string.

Procedure
You also set the string using the controller object [Seite 311]:

1. Set the string using the controller object.
   * Serialization context to bundle asynchronous messages that shall be received in the same order as sent
   data: queuid type prx_scnt.
   data: l_controller type ref to if_ai_posting_controller.
   * Create instance of controller
     l_controller = cl_ai_factory=>create_controller( ).
     queueid = 'PLM0000007_ORDER'.
   * set serialization context
     l_controller->set_serialization_context( queueid ).
   2. Transfer the controller object when you call your outbound proxy.

Accessing the Message GUID

Use
The message GUID is a unique indicator that identifies a message. In the synchronous case, both request and response messages have identical GUIDs. The application can use the GUID if it needs to access the messages again at a later point.

Procedure
Use the method GET_MESSAGE_ID of the controller object to access the message GUID.

- You must generate a controller object before the proxy call for outbound proxies (see: Factory Class [Seite 311]) and transfer it to the parameter CONTROLLER of the proxy. Following the call you can query the GUID of the request message or the response message by using the method GET_MESSAGE_ID of the controller object.
- You do not need to create a proxy object for inbound proxies. To query the message GUID, simply call the method GET_MESSAGE_ID of the parameter CONTROLLER of the inbound proxy.
Receiver Pre-Identification

Purpose

You determine the receiver of a message using the settings in logical routing. You can call this information in the Integration Builder and change it there without having to check or adjust the code used to send a message. It can, however, be important for an application to know whether a receiver actually exists at runtime. For example, it is possible that the message to be sent takes too long to construct because you must select a lot of application data. If the application is able to query whether a receiver exists before it sends the message, then the data does not necessarily need to be selected at all.

Logical routing provides an API to support Receiver Pre-Identification.

Prerequisites

The routing API must access the Integration Server to get the relations and routing rules for an outbound interface. The Integration Server is determined from the specific configuration data of the Integration Engine (category RUNTIME, parameter IS_RFC_DESTINATION), see the section Change Specific Configuration Data in: Displaying/Changing Configuration Data [Seite 269]. Also note that specific configuration data is client-specific. Therefore, you can query different Integration Servers in different clients.

Alternatively, you can specify the Integration Server by using the RFC destination AI_INTEGRATION_SERVER. This entry then applies for all clients in an SAP system.

Process Flow

Receiver pre-identification is performed by using the factory class [Seite 311] CL_AI_FACTORY. You can return an object at routing runtime from this class, which you can use to identify the receivers in advance:

1. First declare the required variables and generate the object at routing runtime:
   
   ```
   class cl_ai_factory definition load.
   data: l_routing_runtime type ref to IF_ROUTING_LOCAL_RUNTIME.
   Data: ls_outbound_interface type PRX_IFRINT,
   L_ROUTING_VIA_PAYLOAD type RM_BOOLEAN,
   L_RECEIVERS_EXIST type RM_BOOLEAN.
   l_routing_runtime = cl_ai_factory=>create_routing_runtime( ).
   ```

2. Determine whether any assigned receivers exist; if yes, check whether the receiver determination was assigned conditions. Call the method RECEIVER_EXIST and transfer the names and the namespace of your outbound proxy.

   ```
   ls_outbound_interface-name = '<Name of Outbound Interface in Integration Repository>'
   ls_outbound_interface-namespace = '<Namespace of Outbound Interface in Integration Repository>'
   call method l_routing_runtime->receivers_exist
   EXPORTING
   OUTBOUND_INTERFACE = ls_outbound_interface
   ```
IMPORTING
ROUTING_VIA_PAYLOAD = L_ROUTING_VIA_PAYLOAD
RECEIVERS_EXIST = L_RECEIVERS_EXIST.

3. Evaluate the return values as follows:
   o If RECEIVERS_EXIST = SPACE, no assigned receivers exist for the outbound interface. In this case, you do not need to evaluate the remaining parameters.
   o If RECEIVERS_EXIST = 'X', one or more receiver determinations exist for the specified outbound interface. Whether the corresponding receivers receive the message depends on whether there are conditions to be evaluated in addition to the receiver determination. The parameter ROUTING_VIA_PAYLOAD shows that there is a receiver determination that is based on the evaluation of the payload with XPath. If the parameter is set, it is a receiver determination without a condition and one or more receivers have been determined.

4. If there are still routing rules to be evaluated, or if you are interested in the receivers of the message, transfer the payload of the message to the method get_receivers of the runtime object. (see: Creating a Payload [Seite 314]).

   The parameter ROUTING_VIA_ROUTING_OBJECTS can be ignored.

Result
The method GET_RECEIVERS returns a table with receivers for which the conditions (where applicable) are completed.

Troubleshooting
Test Report
To display the initial payload of the output or the input message for an inbound or an outbound proxy, call the report SEI_PROXY_01.

Troubleshooting
There are two categories of errors, namely those that occur during generation and those that occur at runtime. The proxy generation function must be able to access the interface description in the Integration Repository to be able to generate proxies. To send a message at runtime involves a whole series of Exchange Infrastructure components. If a transfer problem occurs, it is either because configuration is not yet complete or the Exchange Infrastructure is not yet fully established (Integration Server not recognized, for example). Compared with errors that occur during generation, runtime errors are more difficult to localize. For more information see: Runtime [Seite 264].

The following table summarizes the different types of error and their possible cause:

<table>
<thead>
<tr>
<th>Type of Error</th>
<th>Possible Causes</th>
</tr>
</thead>
</table>

SAP Exchange Infrastructure (BC-XI)  623  321
Unable to generate a proxy

- Server for the Integration Repository not loaded or not entered (check by choosing Goto → Test Connection in transaction SPROXY).
- The WSDL format that was read from the Integration Repository contains errors

Unable to send a message to a receiver

A transferal error can have many causes, for example:

- The Integration Server address is not recognized by the proxy runtime
- No inbound interface could be assigned to the outbound interface during interface determination. Interface determination has possibly not been configured.
- The name of the sender or receiver system could not be found in the Integration Directory or the corresponding server could not be reached
- You have not configured the Integration Server correctly

It is beyond the scope of this documentation to describe the entire infrastructure. For more information, see the SAP Exchange Infrastructure configuration guide (http://service.sap.com/instguides [Seite 321]). There is a monitor for message processing in the Integration Engine. See: Monitoring XML Messages [Seite 290].

Java Proxy Runtime

Use

Using the Java proxy runtime you can receive messages or send messages to the Integration Server (also see: Runtime [Seite 264]).

Features

The Java proxy runtime supports the following:

- **J2EE applications [Seite 323]** on the SAP J2EE Engine using Enterprise Java Beans 1.1. Synchronous and asynchronous outbound and inbound communication is possible with proxy beans.

  You can also call a proxy bean for outbound communication from a Java standalone application on the SAP J2EE Engine.

- Direct synchronous outbound communication between a **Java standalone application [Seite 326]** and the Integration Server, by using proxy classes.

Unless stated otherwise, the following sections apply for both application types.
**J2EE Applications**

**Introductory Notes**

J2EE applications are executed on a J2EE server (for the Java proxy runtime this is the SAP J2EE Engine). The components for the Java proxy runtime must be installed on the J2EE server before they can be used by a J2EE application:

- The libraries for the Java proxy runtime (JPR) and the messaging system for J2EE applications that send messages.
- The libraries for the Java proxy runtime (JPR), the messaging system, and the JPR proxy server for J2EE applications that receive messages.

For more information about the installation procedure, see the SAP Exchange Infrastructure installation guide (http://service.sap.com/instguides).

**Proxy Classes and Proxy Beans**

To send messages from a J2EE application, the application must be programmed against beans that encapsulate all details about the Java runtime. You must have selected the option *J2EE Beans* in Java proxy generation for this. The following classes are generated:

- Proxy classes that send or receive messages using the Java proxy framework.
- Bean classes as an outer shell that conform to the J2EE standard. The beans call the proxy classes for communication.

The classes must be deployed on the J2EE server together with their J2EE application. It is assumed that you are familiar with the programming of beans.

**Features**

In the J2EE environment of the SAP J2EE Engine, the proxy runtime can process outbound and inbound calls both synchronously and asynchronously. The Java proxy runtime uses the functions of the *Messaging System* for this purpose.

⚠️

A Java business system that sends messages by using the messaging system cannot also be the receiver of these messages.
J2EE Sender

To send a message from a J2EE application, the application generates a remote proxy bean and uses it to call the generated outbound method. The bean then calls the corresponding method of the outbound proxy class. This method generates a message from the call and forwards it to the Integration Server by using the messaging system:

For more information about configuration and monitoring of the messaging system, refer to the configuration guide SAP MarketSet Adapter and Messaging System under [http://service.sap.com/instguides](http://service.sap.com/instguides), SAP Exchange Infrastructure.

Indirect Outbound Communication

In this case the standalone application calls a proxy bean on the SAP J2EE Engine. The J2EE server only finds the called proxy bean if the file `jndi.properties` can be found in the classpath and is configured correspondingly. See: Configuration Files [Seite 327].

Otherwise there are no differences with programming a proxy bean from a J2EE application.

J2EE Receiver

On the receiver side, the JPR proxy server listens for inbound messages from the messaging system. The JPR proxy server forwards the message to the corresponding inbound bean using the Java proxy runtime. This bean must be registered in the jpf.registry file. It calls the implementing class on the J2EE server using the generated inbound proxy interface (the generated classes are shown in light gray).
Note the following for the implementing class:

- It must implement the generated Java proxy interface.
- The name of the implementing class is fixed. It comprises the name of the Java proxy interface and Impl.
- The class must be derived from the class AbstractProxy.

If the message interface has the name MyInbound, then the Java interface MyInbound_PortType is generated from it that you must implement using the class MyInbound_PortTypeImpl:

```java
public class MyInbound_PortTypeImpl
    extends AbstractProxy
    implements MyInbound_PortType {...}
```

When the application is deployed, the proxy server is started automatically and it reads the file jpf.registry. This file must be located in the work directory of the SAP J2EE engine [Seite 327]. Here you need to specify which message interface from the Integration Repository belongs to inbound bean. Once the proxy server has been started, it knows automatically if entries in the file have been changed and updates its registrations accordingly. A line in this file is structured as follows:

```
<Interface Namespace>#<Inbound Interface>=<Bean Name>:<Bean Method>
```

<Bean Name> corresponds to the JNDI name of the bean.

<Inbound Interface> is the name of the interface in the Integration Repository. If the character '#', '=' or ' occurs in one of the names then this must be indicated using the escape character backslash (\), as in the following example:

```
http://com/sap/aii/proxy/example#Listening=ExampleService:receive
```

Using this registration, the proxy server would instantiate the bean ExampleService for an incoming message with the inbound interface Listening in the namespace http://com.sap.aii.proxy.example, and call its method receive.

**Deployment**

When deployed, the J2EE applications must ensure that any dependencies on Java proxy runtime libraries in the file reference.txt in the subdirectory managers of the SAP J2EE
Engine are specified. The following entries would be necessary for the application MyOutAPP:

```
reference MyOutApp library:com.sap.aii.proxy.xiruntime
reference MyOutApp library:com.sap.aii.messaging.runtime
reference MyOutApp library:com.sap.aii.util.misc
reference MyOutApp library:inqmyxml
```

Java Standalone Applications

Introductory Notes

The Java proxy runtime supports the direct synchronous outbound communication between a Java standalone application and the Integration Server. Therefore, using a Java proxy, you can call another business system synchronously via the Integration Server. Inbound processing for Java standalone applications is not supported.

Direct Synchronous Outbound Communication

Prerequisites

The components of the Java Proxy Runtime (JPR) must be installed on the computer that functions as the business system.

Process Flow

To send messages from a standalone application, the application must be programmed against proxy classes that encapsulate all details about the Java runtime. The same proxy classes are generated for J2EE applications without bean classes. Also see: Java Proxy Objects [Seite 200].

To send a message using SAP Exchange Infrastructure, simply call the corresponding outbound proxy by transferring the object for the output message:
The Java proxy runtime (JPR) converts the proxy parameters to an XML message and sends it to the Integration Engine, where it is processed further according to the configuration (Routing, Mapping, and so on).

### Configuration Files

**Use**

The Java proxy runtime (JPR) fetches server parameters from the exchange profile at runtime. Using these parameters, the JPR can determine server names from the SAP System Landscape Directory, for example. In addition to the exchange profile, the JPR reads the following configuration files:

- **jpf.properties**: A configuration file for test purposes (for example, you can divert copies of incoming and outgoing messages to a local file).
- You require the **jpf.registry** to generate the implementing class and to register inbound beans on the proxy server [Seite 323].
- **jndi.properties**: Required to localize the proxy beans if a standalone application calls an outbound proxy bean on the J2EE server.
- **logging.properties**: Defines the settings for the logging of the JPR. The system gathers the initial settings from the exchange profile.

For more information about the exchange profile, see the *Configuration Guide*.

The files **jpf.properties**, **jpf.registry** and **logging.properties** are expected in the work directory of the standalone application or of the J2EE server. If the root directory of
the SAP J2EE Engine is \texttt{<SERVER\_HOME>}, then the work directory of the J2EE server is located under the following path:

\textbf{J2EE Work Directory}

<table>
<thead>
<tr>
<th>Server Type</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standalone</td>
<td>\texttt{&lt;SERVER_HOME&gt;\alone}</td>
</tr>
<tr>
<td>Cluster</td>
<td>\texttt{&lt;SERVER_HOME&gt;\cluster\server}</td>
</tr>
</tbody>
</table>

\textbf{Features}

\textit{jpf.properties}

Using the exchange profile, you can administrate the server landscape of the Exchange Infrastructure from a central point. If you change a server address, you can make the necessary changes for all components involved centrally. Therefore, you may only use the configuration file \texttt{jpf.properties} for test purposes, otherwise you will have to make these changes locally each time.

Each line in this file is structured as follows:

c\texttt{om.sap.aii.proxy.framework.\textless\textit{Parameter}\textgreater\texttt{==\textless\textit{Value}\textgreater}}

\textbf{Parameters in the jpf.properties file}

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>lcrHostname</td>
<td>\texttt{&lt;Host&gt;}</td>
<td>System Landscape Directory host. Only takes effect if information cannot be taken from the exchange profile.</td>
</tr>
<tr>
<td>lcrPortnumber</td>
<td>\texttt{&lt;Port&gt;}</td>
<td>System Landscape Directory port. Only takes effect if information cannot be taken from the exchange profile.</td>
</tr>
<tr>
<td>connection.timeoutInMSec</td>
<td>\texttt{&lt;Timeout in ms&gt;}</td>
<td>Specifies how long a message is valid after it has been transferred to the messaging system.</td>
</tr>
<tr>
<td>connection.timeoutInMSec</td>
<td>Default: 30000</td>
<td></td>
</tr>
<tr>
<td>destination</td>
<td>http://\texttt{&lt;Host&gt;:&lt;Port&gt;}/sap/xi/engine/entry?action=execute&amp;pipelineid=central (only relevant for Java standalone applications)</td>
<td>Overrides the address of the Integration Server inbound pipeline. \texttt{&lt;host&gt;} and \texttt{&lt;port&gt;} must be replaced by the name and port number of the server on which the central Integration Engine is running. In other words, the physical address of the corresponding R/3 system. You must specify the client that was assigned the role of the Integration Server using the parameter \texttt{destination.client}.</td>
</tr>
<tr>
<td>destination.client</td>
<td>\texttt{&lt;Client&gt;} (only relevant for Java standalone applications)</td>
<td>Overrides the client that was assigned the role of the Integration Server. Example: \texttt{010}.</td>
</tr>
<tr>
<td>requestFilename</td>
<td>\texttt{&lt;Filename.txt&gt;}</td>
<td>Diverts a copy of the SOAP message to the text output file \texttt{&lt;Filename.txt&gt;} for outbound messages. The system saves the message payload as an XML file.</td>
</tr>
<tr>
<td>responseFilename</td>
<td>\texttt{&lt;Filename.txt&gt;}</td>
<td>Diverts a copy of the SOAP message to the text output file \texttt{&lt;Filename.txt&gt;} for inbound messages. The system saves the message payload as an XML file.</td>
</tr>
<tr>
<td>senderName</td>
<td>&lt;Name of a business system&gt;</td>
<td>If you cannot reach the server specified using <code>lcrHost</code> and <code>lcrPortnumber</code> for the System Landscape Director or if the Java application is not entered on the server, enter the name of the sender business system using the parameter <code>senderName</code>. The Java proxy runtime does then not try to find out the name of the business system using <code>TechnicalId</code> and <code>Hostname</code>.</td>
</tr>
</tbody>
</table>

**jndi.properties**

This file is only required in the outbound case, if a Java standalone application wants to use a proxy bean on the J2EE server. The file enables the application to locate the proxy bean by using the naming service of the J2EE server. It must be located in the work directory of the standalone application and contain the following entries:

```java
java.naming.factory.initial=com.inqmy.services.jndi.InitialContextFactoryImpl
java.naming.provider.url=localhost
java.naming.security.principal=Administrator
java.naming.security.credentials=
```

Alternatively, you can also set these attributes directly in your application program.

**logging.properties**

Using this file, you configure logging at runtime. In the file `logging.properties` in the work directory, you can set different detailed loggings, using parameters. If no file of this type exists in the work directory, the system creates it and enters default settings when you first call the JPF. The default setting for logging information is `NONE`; this means that the system does not generate any logging information. In `logging.properties`, set the parameter `severity` to `INFO`, if required. As a result, the system writes a log in a file called `.log/default.trc` in the work directory.

⚠️ The file `.log/default.trc` can become extremely large over time, because new entries are constantly being added to it. SAP recommends that you delete the contents of this file periodically.

**Example of logging.properties**

```properties
#Created by Java Proxy Runtime
#Wed Feb 20 13:40:14 GMT+01:00 2002
.logs=log[defaultTrace]
.severity=INFO
/Applications.logs=log[applicationsLog]
/Applications.severity=INFO
/System.logs=log[systemLog]
/System.severity=INFO
formatter[DefaultTrace]=TraceFormatter
formatter[FullTrace].pattern=%24d (%p) [%t] %-44l %s: %m
formatter[FullTrace]=TraceFormatter
log[applicationsLog].formatter=ListFormatter
log[applicationsLog].pattern=./log/applications.log
log[applicationsLog]=FileLog
log[defaultTrace].formatter=formatter[DefaultTrace]
log[defaultTrace].pattern=./log/default.trc
log[defaultTrace]=FileLog
```
Error Handling

Use

The following errors can be handled by the Exchange Infrastructure:

- **System errors**, which are triggered by the Exchange Infrastructure. These might be errors during transfer, due to a failed server, for example.

- **Application errors**, which are triggered or handled by the application alone. An example would be a request for data about a material that is unknown in the receiver system.

Error handling is essentially of interest in the synchronous case. In this case, an application can report application errors to the caller application. In the asynchronous case, you can catch an error that has occurred during transfer and, on the inbound side, forward an error to monitoring.

Features

Handling System Errors

You can catch errors that occur when a message is being transferred, using the exception class `SystemFaultException`, so that the system displays an error message.

The full name of the exception class for system errors is:

```
com.sap.aii.proxy.framework.core.SystemFaultException
```

Handling Application Errors

In the Integration Repository, you can define **fault messages** [Seite 331] to handle application errors. The proxy generation functions then generate exception classes. Using fault messages, you can handle errors triggered by the called application or persist them for monitoring purposes:

<table>
<thead>
<tr>
<th>Message Interface</th>
<th>Options for Error Handling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synchronous Communication (Inbound/Outbound)</td>
<td>Triggering of an error at the receiver that is providing a service; handling of the error at the sender that called the service.</td>
</tr>
</tbody>
</table>
Asynchronous Inbound Interface

Fault messages are not used here to handle a error at the sender. The fault message generated by the triggered error is persisted for monitoring instead.

Exception classes for application errors are derived from the basis exception class ApplicationFaultException. You can use this exception class to determine an error situation, independently of the exact error, or in an else branch, you can use it to catch all the errors that have not yet been handled.

The full name of the exception class for application errors is: com.sap.aii.proxy.framework.core.ApplicationFaultException.

Activities

5. To handle an application error, create fault messages for your message in the Integration Repository.

6. In each case, handle system errors, and, if necessary, application errors (see: Reporting and Handling Errors [Seite 333]).

Fault Messages

Definition

A fault message is a message that describes an application-specific error situation.

Use

The proxy generation functions generate exception classes for fault messages in the Integration Repository. If an application triggers an exception using an exception class such as this, the proxy runtime automatically converts it to a fault message. Using the fault message, the application records an application error (for example, Requested Customer Profile Unknown).

Structure

The exception class generated from the fault message can give you information about the error cause using the following structure:
The defined structure **STANDARD** contains fields essential for forwarding the error. The structure **ADDITION** can be used freely by the application.

**Example**

The following UML diagram shows the class hierarchy of the generated classes for the fault message **Fault**. The data type **Application** in the Integration Repository acts as the application-specific part of the fault message:

To send the fault message, complete the objects for the classes of the fault message on the inbound side. Transfer the fault message to an instance of the exchange class (in the
example, Fault_Message_Exception) and trigger the exception throw <Exception Class>.

### Reporting and Handling Errors

#### Purpose

The options for error handling depend on the type of communication chosen:

- **Asynchronous Communication:** On the outbound side, the application can only handle errors that occur when the message is being sent (for example, the outbound queue is full). These errors can be caught and persisted using the exception class `SystemFaultException`. On the inbound side, as in the synchronous case, you can define fault messages [Seite 331], which are then only forwarded to monitoring (and not to the sender).

- **Synchronous Communication:** Using fault messages you can handle errors that have occurred on the inbound side. To confirm an error, the proxy runtime sends a message from the receiver back to the sender, which can then respond to it.

#### Process Flow

In the graphic below, an error from the receiver system is reported to the sender system in synchronous communication:

1. If an application on the inbound side triggers an error, the proxy runtime converts the corresponding exception class to a fault message and transfers it to the sender system using the Integration Server.
2. The proxy runtime in the sender system interprets the fault message and triggers the exception for the outbound interface.
3. In the sender system, the application can catch and handle the exception using a
try/catch block. Using the super exception class `ApplicationFaultException`,
you can catch all remaining application errors.

---

**Default and Initial Values in Java**

The XSD standards tell you how to handle default values. The following standard procedure
exists for proxy generation:

### Standard Procedure for Default Handling

<table>
<thead>
<tr>
<th>Outbound Interface</th>
<th>When generating the XML data stream, the default value is set in all initial fields.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inbound Interface</td>
<td>Inbound processing of the proxy runtime sets empty and missing tags to the default value before the application is called.</td>
</tr>
</tbody>
</table>

To improve performance, optional, initial, scalable elements with a default value are
suppressed in the message. The zero pointer in Java is converted to `xsd:nil`.

---

**Special Services of the Proxy Runtime**

### Use

Using the documentation above, and a correspondingly configured connection, you are now
able to exchange messages using the Exchange Infrastructure from the perspective of the proxy runtime (the interfaces used to exchange messages are maintained in the Integration Directory [Seite 21] and not in the application program). However, there are other services that are required in particular application scenarios. Services of this type are described below.

### Features

You can also influence the runtime response in the application program, using methods of the class `messageSpecifier` [Seite 335]:

- You can call payloads [Seite 15] in the proxy runtime after an outbound proxy has been
called, or while an inbound proxy is being called, for example, because you want to
archive the payloads. See: Calling the Payload [Seite 338].

- Instead of determining the receiver business system using routing, you can specify the
receiver system for the outbound proxy. See: Setting the Receiver System [Seite 338].

- You can append attachments to the message at the sender, which you can then call at
the receiver (and the other way around). See: Attachments [Seite 339].

- You can query the message GUID of sent and received messages. See:
MessageSpecifier Object [Seite 335].
Furthermore, administrators on the inbound side can allocate authorizations for receiving messages by assigning the corresponding security roles to a J2EE user and then specifying this user in the Integration Directory in the logon data [Seite 247]. For further information about authorization checking and creating users on the SAP J2EE engine, see the configuration guide.

MessageSpecifier Object

Use

Assume that you want to append an attachment to your message. The proxy runtime must take this into account when constructing the message. The proxy runtime provides an object MessageSpecifier to be able to use special services of the proxy runtime [Seite 334].

Integration

The object MessageSpecifier is an attribute of your generated outbound or inbound proxy; its full name is com.sap.aii.proxy.framework.core.MessageSpecifier.

Access to Methods of MessageSpecifier

J2EE Applications

You cannot access this object directly in J2EE applications, because access to the MessageSpecifier object is encapsulated using the proxy bean. Instead, use the proxy bean methods

- MessageSpecifier $messageSpecifier()
- void $messageSpecifier(MessageSpecifier ms)

to fetch the MessageSpecifier object, call its methods (see below), and then return it to the proxy class again.

Standalone Applications

Java standalone applications directly access the proxy class instead of the bean class. This class has an attribute that you can reference directly to access the MessageSpecifier object. Let us assume that you generate an instance with the name flightBooking for your outbound proxy class. You can, for example, call the method addAttachment() directly from the instance flightBooking when appending an attachment:

flightBooking.messageSpecifier.addAttachment(...) 

In the same way that you give the proxy object information before you send a message, you can also query information about methods of the MessageSpecifier object when a message is received.

Note that no inbound processing is supported for Java standalone applications.

Features

Messaging-System (MS) for J2EE Application
<table>
<thead>
<tr>
<th>Method</th>
<th>Meaning</th>
</tr>
</thead>
</table>
| `public void setMessagingTimeoutInMSec (long timeout);` | Determines the validity period of a message:  
- If an asynchronous message cannot be sent to the Integration Server within the period of validity it is persisted for MS monitoring.  
- If the messaging system does not receive a response message for a synchronous message within the period of validity then an error is returned to the application. If the response message is received thereafter, the messaging system also persist this message for MS monitoring. |
| `public void setSerializationContext (String name)` | This method enables you to assign a serialization context to multiple messages to guarantee the quality of service **exactly once in order**. The string `name` can be a maximum of 16 characters long and must be in upper case. All asynchronous messages with the same serialization context are received in the same order at the receiver as they were put in the outbound queue during the **COMMIT WORK** on the sender side. |

**Payload (see also: Calling the Payload [Seite 338])**

<table>
<thead>
<tr>
<th>Method</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>public String getInputPayload()</code></td>
<td>Returns the payload of the input message (after the system has called an outbound proxy or during inbound processing).</td>
</tr>
<tr>
<td><code>public String getOutputPayload()</code></td>
<td>Returns the payload of the output message (after the system has called an outbound proxy).</td>
</tr>
<tr>
<td><code>public void setPayload(String payload)</code></td>
<td>This method enables you to set the payload yourself instead of generating the data that was transferred to the proxy. In this case, it is the application itself that is responsible for the correct format of the payload. The proxy runtime completes the message header and forwards the message with the transferred payload.</td>
</tr>
</tbody>
</table>

**Setting the Receiver [Seite 338]**

<table>
<thead>
<tr>
<th>Method</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>public void addReceiver (com.sap.aili.messaging.mo.xmb.XMBParty receiver)</code></td>
<td>Inserts a receiver business system for all receiver systems of the outbound proxy.</td>
</tr>
<tr>
<td><code>public boolean removeReceiver (com.sap.aili.messaging.mo.xmb.XMBParty receiver)</code></td>
<td>Deletes a specific business system for all receiver systems of the outbound proxy.</td>
</tr>
<tr>
<td><code>public void clearReceivers()</code></td>
<td>Deletes all specific receiver business systems.</td>
</tr>
</tbody>
</table>

**Attachments [Seite 339]**

<table>
<thead>
<tr>
<th>Method</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>public MessageAttachment createAttachment(String name, String type, Object data)</code></td>
<td>Creates a new attachment object. For <code>type</code> you must specify the MIME type (for example, <code>'text/html'</code>).</td>
</tr>
</tbody>
</table>
public MessageAttachment addAttachment(String name, String type, Object data)
Generates a new attachment object, adds this to the set of attachments and returns the attachment object.

public void addAttachment(MessageAttachment attachment)
Adds a new attachment to the set of attachments.

public boolean removeAttachment(MessageAttachment attachment);
Deletes the specified attachment.

public void clearAttachments();
Deletes all attachments. You must call this method on the receiver side so that the system does not return received attachments.

public MessageAttachment getAttachment(String name);
Returns the attachment with the name name.

public java.util.Enumeration getAttachments();
Returns a list of all attachments.

Setting/Determining the Sender (Business System)

<table>
<thead>
<tr>
<th>Method</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>public String getSenderName()</td>
<td>Determines the sender name (business system) of a received message at the receiver.</td>
</tr>
<tr>
<td>public void setSenderName(String name)</td>
<td>Sets the sender name manually (business system). Corresponds to the parameter senderName of the configuration file [Seite 327] jpf.properties. Normally the Java proxy runtime fetches the sender name from the SAP System Landscape Directory.</td>
</tr>
</tbody>
</table>

Querying a Message GUID

<table>
<thead>
<tr>
<th>Method</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>public com.sap.aii.util.misc.api.GUID getMessageID()</td>
<td>Query the message GUID when a message arrives. The method toString() provides the string representation of the GUID.</td>
</tr>
<tr>
<td>public com.sap.aii.util.misc.api.GUID getMessageIDLastSent()</td>
<td>Query the message GUID when a message has been sent. The method toString() provides the string representation of the GUID.</td>
</tr>
</tbody>
</table>

In synchronous communication the message GUID of the request is the same as that for the response. getMessageID() is therefore only required in the inbound case.
**Calling the Payload**

**Use**

Both in the inbound and outbound cases, it is useful to be able to access the payload of a message. You can only access the payload once the message has been sent or once it has been received (you might want to archive the sent message, for example).

For test purposes, you can also configure the Java Proxy runtime using the configuration file `jpf.properties` so that the request or response message is written to a local file. The payload is located after the SOAP envelope in the message.

**Activities**

You can have the system return the payload of the message on both the inbound and outbound side. Use methods of the messageSpecifier object to do this:

1. Access the `MessageSpecifier` object after the message has been sent (see The `MessageSpecifier` Object).
2. You can return the corresponding payload as `String` with the `MessageSpecifier` methods `getOutputPayload()` and `getInputPayload()` (`getInputPayload()` does not return anything in asynchronous communication).

**Setting the Receiver System**

**Use**

When you create outbound and inbound interfaces in the Integration Builder, do not assign them to each other. Instead, the configuration determines which receivers receive a message.

However, in some cases, you know the receiver system of a message because of the application data. For this reason, the proxy runtime gives you the option of setting the receiver yourself, before the message is sent; this means that the entire routing logic does not have to run. Since you must maintain all connections centrally in logical routing, setting the receiver in advance does not save you from having to create a connection in logical routing.

If this were not the case, connections would exist that were only visible in program code. You must be able to access information of this type from a central point, however.

**Procedure**

1. Import the class `XMBParty` to your source file:
   ```java
   import com.sap.aii.messaging.mo.xmb.XMBParty;
   ```
2. Set the name of the business system by using the method `setName`:
   ```java
   XMBParty receiverParty = new XMBParty();
   ```
receiverParty.setName("<Receiver Business System>");

The receiver interface in the object XMBParty cannot be set.

3. Access the MessageSpecifier object [Seite 335] to transfer receiverParty prior to your proxy call by using the MessageSpecifier method addReceiver.

You can delete a receiver that you have added by using the messageSpecifier object method removeReceiver.

Attachments

Use

An attachment comprises any data (for example, text or graphics) that can be appended to a message. In principle, you can append an unlimited number of attachments to an output message of an outbound or inbound interface (in the synchronous case), using the messageSpecifier object.

Note that not all receivers of a message can understand attachments and therefore some receivers will ignore them. The proxy runtime (ABAP/Java) can process attachments, however the majority of the adapters cannot.

Features

You can generate an attachment object using methods of the messageSpecifier object. Attachment objects are instances of the class MessageAttachment, which has the full name com.sap.aii.proxy.framework.MessageAttachment.

Activities

- Access the MessageSpecifier object [Seite 335] prior to the proxy call to generate attachments and to append them to the output message. Use the method createAttachment() to create attachment objects and then use the method addAttachment() to append them to the proxy object before the message is sent.

- To query attachments at the receiver, fetch the MessageSpecifier object and query the attachments with the method getAttachment() or getAttachments().

Note for synchronous communication that received attachments are sent back to the sender if they are not deleted. If this should not happen, you can delete the individual attachments with the method removeAttachment(), or all attachments with the method clearAttachments().
Adapters

Purpose
Adapters enable the Integration Engine [Seite 266] to communicate with different application components [Extern].

Introductory Notes
You only require an adapter to communicate with SAP systems older than Release 6.20 and with external systems. A direct system connection using proxies and without additional adapters is supported for SAP systems that are based on the SAP Web Application Server 6.20.

Features
Adapters connect the Integration Engine to SAP legacy systems, as well as to external systems.

In this way, SAP adapters integrate existing SAP components with SAP Exchange Infrastructure [Seite 1], for example. In the process, XML and HTTP-based documents are converted to IDocs [Extern] (IDoc adapter [Seite 340]) and RFCs [Extern] (RFC adapter [Seite 349]) and the other way around. This enables you to integrate your existing SAP infrastructure with the new SAP infrastructure that is based on system integration and the exchange of XML messages.

The plain HTTP adapter [Seite 357] gives application systems the option of communicating with the Integration Engine and exchanging business data in a simple format, using an HTTP connection.

Furthermore, the Adapter Engine [Seite 362] provides you with various adapters that you can use to connect external systems to your Integration Engine. You can use these adapters to convert XML and HTTP-based messages to the specific logs and formats of the respective external systems and the other way around.

IDoc Adapter

Definition
The IDoc adapter [Extern] enables you to process IDocs [Seite 343] (Intermediate Documents) using the Integration Engine [Seite 266]. IDocs are supported from SAP systems Release 3.1x or higher.

The IDoc adapter is used by SAP components to connect to a centrally configured [Seite 267] Integration Engine using IDocs. A system that has an Integration Engine of this type is referred to as the Integration Server [Extern]. External systems can also use the IDoc adapter to connect to an Integration Server.

Use
You require the IDoc adapter to connect SAP components (or external systems) to the Integration Server using IDoc communication.
You need the IDoc adapter if you want to use the Integration Server to process [IDoc [Extern]]. The IDoc Adapter converts the IDocs to IDoc XML [Extern] format so that they can be processed by the Integration server pipeline [Seite 267].

It is also possible to transfer XML received by the Integration Server in IDoc format to connected systems. The Integration Server must convert the received XML to IDoc XML, and then transfer this IDoc XML to the IDoc adapter. The IDoc adapter converts the IDoc XML to native IDoc format and sends the IDoc to the receiver system identified in technical routing [Extern] using the standard IDoc interface. The IDoc adapter can also call external subsystems.

The IDoc adapter only evaluates the payload of a received XML message. Additional attachments (such as trace attachments) are ignored and not forwarded.

**Structure**

Essentially, the IDoc adapter comprises two parts, namely an inbound adapter at the Integration Server inbound channel, and an outbound adapter at the Integration Server outbound channel. The meta data for the IDoc types [Extern] involved is shared.

- The inbound adapter converts an IDoc to IDoc XML, starts the Integration Server and then transfers the IDoc XML message.
- The outbound adapter converts IDoc XML to IDoc format and transfers the IDoc to an SAP component or an external system (subsystem).

If none of the pipeline services [Seite 267] require the IDoc XML, then you can set a corresponding configuration parameter [Seite 269] so that IDocs are not converted to IDoc XML, but are transported as tables instead. This only makes sense if IDocs are to be received on the Integration Server without changes to the data records and sent again as IDocs. By avoiding unnecessary conversions from and to XML can lead to improvements in system performance.

You also have transactions to do the following:

- **Maintain ports in the IDoc adapter** [Seite 346] (IDX1)
- **Load, display and delete meta data** [Seite 348] (IDX2)
- **Display IDoc XML messages** [Seite 354] (IDX5)

The authorizations for the individual transactions are all based on the authorization object S_XMB_AUTH or S_XMB_DSP. Individual authorizations for transactions must be granted as follows using the corresponding roles:

IDX1 (role SAP_XI_ADMINISTRATOR) and IDX2 (role SAP_XI_CONFIGURATOR)

- **Activity**
  - 01 (create)
  - 02 (change)
  - 03 (display)
  - 06 (delete)

- **Operation**
  - Not defined

IDX5 (role SAP_XI_MONITOR)

- **Activity**
You cannot execute transactions in the menu that have the locked symbol (🔒), because you do not have the required authorization.

**Integration**

The IDoc adapter is part of an Integration Server. The inbound IDoc adapter is located before the Integration Server pipeline and calls this pipeline. Whereas the outbound IDoc Adapter is called by the pipeline; it can be regarded as a pipeline service [Seite 267] and is therefore part of the pipeline itself.

The connected systems transfer IDocs to standard IDoc tRFC outbound processing or receive IDocs from standard IDoc tRFC inbound processing.

Only use the IDoc adapter to integrate SAP components with the Integration Server if a scenario of this kind suits your requirements. For example, if you want to make IDoc data in the form of XML messages available to additional receivers, or if you want to integrate different components or business processes, which were previously not integrated. Only stop using existing functioning IDoc scenarios after serious consideration.
IDoc Processing with the Integration Server

Purpose

The IDoc adapter [Seite 340] enables you to process and send IDocs [Extern] (Intermediate Documents) using the Integration Engine [Seite 266]. You can use all existing IDoc types [Extern] that have been released. ALE Audit [Extern] is supported.

Processing IDocs using the Integration Engine pipeline [Seite 267] is an alternative to processing XML messages that are generated using the proxy interface. This alternative is considered for all SAP applications and external systems (subsystems) that have already defined IDocs, as well as for new applications of SAP components that do not yet have access to the proxy generation [Seite 186] functions.

For more information on IDocs, see IDoc Interface/Electronic Data Interchange (BC-SRV-EDI) [Extern].

Prerequisites

Integration Server

If the system on which the IDoc adapter is located is configured [Seite 267] as the Integration Server, then inbound IDocs are processed using the IDoc adapter and not using the normal IDoc interface.

On an Integration Server [Extern] you must therefore enter IDocs that are to be processed using the IDoc interface in an exception table using the report IDX_SELECT_IDOCTYP_WITHOUT_IS. You can only include IDoc types in this table that are already defined in the system. These IDoc types are not processed using the Integration Engine.

To load IDoc metadata [Seite 348] you must establish an RFC connection to the connected system using the port maintenance in the IDoc adapter [Seite 346]. This system is defined by the sender port and the client in the IDoc control record.

Application Components

Existing applications do not need to be changed.

In the sender system you merely need to change the target address of the RFC destination of a tRFC port for the Integration Server. If no tRFC port exists, you must create one for the Integration Server.

To change an existing tRFC port, proceed as follows:

1. Call the transaction Display and Maintain RFC Destinations (SM59).
2. Select the corresponding RFC destination by double clicking it.
3. Enter the server address of the Integration Server as the Target host.

For more information see, Displaying, Maintaining and Testing Destinations [Extern].

To create a new tRFC port, proceed as follows:

1. Call the transaction Display and Maintain RFC Destinations (SM59) to create a connection type 3 RFC destination and then specify the server address of the Integration Server as the Target host.

For more information see, Displaying, Maintaining and Testing Destinations [Extern].

2. Call the transaction Ports in IDoc Processing (WE21) and then with this destination define a new port [Extern] of type Transactional RFC (tRFC).

For more information about configuring ports, see Configuring Ports [Extern].

3. Call the transaction Partner Profiles (WE20) to change the partner profiles. Enter the receiver system as the partner number, select the type of message to be sent and the
basis type in the outbound channel parameters, and enter the new port as the receiver port.

For more information, see Creating Outbound Partner Profiles [Extern].

You must also change the partner profiles in the receiver system.

1. Call the transaction Partner Profiles (WE20) and enter the sender system as the partner number.

2. Also specify the inbound message type and the basis type for the inbound parameters.

**Process Flow**

The connected systems generate the corresponding IDocs in the applications and send them to the new tRFC port; the RFC destination of this port addresses the Integration Server.

The IDoc adapter that generates [IDoc XML [Extern]] from the native IDoc format is called on the Integration Server. This IDoc XML is transferred to the Integration Engine pipeline for the purposes of routing [Seite 214], mapping [Seite 143] and sending.

If the received IDoc is to be sent as an IDoc again without any changes to the data records, it is possible to deactivate the conversion to IDoc XML by using a corresponding configuration parameter [Seite 269].

**Inbound IDoc Adapter**

Once the IDoc has left the application through the RFC interface and has reached the inbound IDoc adapter, the system converts it from IDoc format to IDoc XML format. A message GUID is generated for each IDoc.

IDocs can also be transferred in packages. A package of IDocs is only accepted if all IDocs within the package can be converted to IDoc XML format. If one or more of the IDocs cannot be converted, the whole package is refused.

The conversion comprises a implicit 1:1 mapping of segments and fields to XML tags. The necessary metadata (IDoc structures for the corresponding IDoc types) is not located on the Integration Server, but in the connected SAP system (or in the SAP reference system in which the meta data is stored if the sender system is a subsystem).

You can either call this metadata directly at runtime or you can load [Seite 348] it onto the Integration Server beforehand. To be able to do this you must have already created [Seite 346] an RFC connection (a port) to the connected system.

To display already loaded metadata and connected systems, use the metadata display in the [IDoc adapter [Seite 348]]. You can also reset [Seite 348] and then reload metadata.

**Integration Server: Inbound Channel**

A message comprising a header and body with payload (IDoc XML) is sent to the Integration Server for each IDoc XML.

Among other things, the header contains the sender business system. This is identified from the values for the sender port and the client from the IDoc control record. The sender port comprises the prefix SAP and the system ID (for example, SAPBCE). These values are assigned to business systems in the System Landscape Directory [Seite 255].

In addition, the sender/receiver IDoc partners are made available to logical routing as context objects. For more information, see the description of the conditions editor [Seite 220] for logical routing.

The IDoc XML message must only contain one IDoc. Inbound IDocs are not persisted.
**Logical Routing**

The receiver can be determined either by using context objects from the IDoc control record or by using an XPath rule on the IDoc XML, depending on the sender business system in the IDoc XML message. To be able to identify the receiver successfully, the XPath rule must have the status `true`.

If context objects are not used and if no XPath rule is defined, then the business system that is assigned to the sender business system is selected as the receiver.

**Technical Routing**

In technical routing the logical receiver is assigned a physical target. For the message, the outbound IDoc adapter requires one IDoc end point (an RFC destination) from technical routing for each logical receiver.

This IDoc end point is analyzed and provides the technical information for the tRFC call. Note the following for the version of the technical IDoc interface: The function module

- `INBOUND_IDOC_PROCESS` (Version 2) must be used for IDoc types with short names, for example eight characters
- `IDOC_INBOUNDASYNCHRONOUS` (Version 3) must be used for IDoc types with long names, for example thirty characters

An IDoc end point is also required in sender systems that expect acknowledgment messages from an IDoc adapter.

**Mapping**

The IDoc adapter does not make any special demands on mapping. The outbound IDoc adapter must simply be provided with an IDoc XML structure. This either already exists or must be generated by using a mapping. The data in the IDoc control record does not necessarily need to be completed by the mapping here since any data will be overwritten by the outbound IDoc adapter at a later stage anyway.

**Integration Server: Outbound Channel**

Before the message leaves the Integration Server, the message header contains values that are exported and used later to complete the IDoc control record.

The system calls the outbound IDoc adapter and transfers the XML IDoc, IDoc end point data, and the control record data.

**Outbound IDoc Adapter**

The task of the outbound IDoc adapter is to convert XML IDocs to native IDoc format and to transfer the IDocs to the receiver system (an SAP component or subsystem) using the standard tRFC IDoc interface. The IDoc control record is completed by the outbound IDoc adapter.

The outbound IDoc also identifies the IDoc sender partner from the assigned sender business system and the IDoc receiver partner from the assigned receiver business system.

Sender and receiver business systems are assigned to IDoc partners in the System Landscape Directory. An IDoc partner of type LS (logical system) is assigned to each business system by using a logical system name.

External systems can also be maintained in the System Landscape Directory as SAP systems.

To convert IDoc XML to IDoc format, the outbound IDoc adapter requires the metadata for the respective IDoc type. If this data is not available, you must load it from the receiver system (or from the corresponding SAP reference system, if the receiver system is a...
subsystem). The IDoc adapter requires the current release of the receiver system to obtain the metadata. The current release is located in the IDoc end point [Seite 235].

The conversion to IDoc format then takes places using the metadata.

If the IDoc you want to send has already been received as an IDoc, and if the corresponding configuration parameters [Seite 269] that deactivate the conversion of IDoc XML have been set, then conversion from IDoc XML to IDoc format does not occur at this point.

The system then calls a function module of the tRFC IDoc inbound channel using the destination (from the IDoc end point) and transfers the data in the corresponding format. The IDoc interface version from the IDoc end point determines which function module is called.

- In the case of version 2, the function module INBOUND_IDOC_PROCESS is called that requires the native IDoc exchange format 3.x (for IDoc types with short names, for example eight characters).
- In the case of version 3, the function module IDOC_INBOUNDASYNCHRONOUS is called (for IDoc types with long names, for example thirty characters).

Both function modules are called in the receiver system that is addressed using the RFC destination that is defined in the IDoc end point.

The function module call is placed in the tRFC queue [Extern] and then the IDoc is finally sent to the receiver application. The Integration Server checks if the message was actually sent or not.

To enable you to select IDocs in the target system later on, the system transfers the message GUID and the IDoc number in the field ARCKEY of the IDoc control record.

Maintaining Ports in the IDoc Adapter

Use

Using the port maintenance function in the IDoc adapter [Seite 340] you establish an RFC connection to the system that contains the metadata that the IDoc adapter requires to convert received IDocs [Extern] (Intermediate Documents) to XML IDoc [Extern] format. This system is defined by the sender port and the client in the IDoc control record. It is either the sender SAP system or an SAP reference system.

The metadata is the IDoc structures for the corresponding IDoc types [Extern]. Using this port you can call this data either directly at runtime or you can load [Seite 348] it to the Integration Server [Extern] (the system with the IDoc adapter) beforehand.

You must maintain a port in the IDoc adapter for all clients in each SAP system that you want to connect to the Integration Server using IDocs.

This is also the case for receiver systems if they are subsystems (non-SAP systems). Systems of this kind are defined using the receiver port in the respective IDoc end point [Seite 235].

Creating a Port

To establish an RFC connection using the port maintenance in the IDoc adapter proceed as follows:

1. Call the transaction IDoc Adapter Port Maintenance (IDX1).
On the left-hand side the system displays a list of ports that are already defined, for example SAPABC_123, and a description.

2. Choose Create (_create) to create a new port (a new RFC connection) for the selected system.

   The system converts the display area on the right-hand side of the screen into an input area where you can enter the specifications for the new port.

**Specifications for Port to Be Created**

<table>
<thead>
<tr>
<th>Specification</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Port</td>
<td>Port name. For SAP systems this comprises the type and ID of the connected system, for example, SAPABC.</td>
</tr>
<tr>
<td>Client</td>
<td>Three figure number of the respective client [Extern], for example, 123.</td>
</tr>
<tr>
<td>Description</td>
<td>A short description of the port.</td>
</tr>
<tr>
<td>RFC Destination</td>
<td>The RFC destination for the connected system. This comprises, for example, the system ID and the client, for example, ABC_123.</td>
</tr>
<tr>
<td>Receiver of Status Messages</td>
<td>Partner number and partner type of receiver.</td>
</tr>
</tbody>
</table>

The specifications for Port and Client correspond to the sender port and the client in the IDoc control record.

3. Enter the required data for the new port and save the data.

   The system creates the new entry and places it alphabetically in the list of already defined ports. The system displays a corresponding message.

**Changing a Port**

To change specifications for a port, proceed as follows:

1. Select the port from the port list and choose Change (_changes).

   The system converts the display area on the right-hand side of the screen into an input area where you can make your changes. The port name and the client cannot be changed.

2. Make your changes to the selected port and save the data.

   The system returns the input area on the right-hand side of the screen to display mode and displays the selected port with your changes.

**Deleting a Port**

To delete a port, proceed as follows:

1. Select the port from the port list and choose Delete (_delete).

   The system displays a dialog box asking if you really want to delete the port.

2. To delete the port, choose Yes.

   The system deletes the entry and displays a corresponding message.
Loading, Displaying, and Deleting Metadata

Use


Using an RFC connection [Extern], metadata of this type can be either called directly at runtime or loaded to the Integration Server [Extern] (the system with the IDoc adapter) beforehand.

The system where the metadata is located is either the sender SAP system or the SAP reference system where the meta data is saved if the sender system is a subsystem.

You can display metadata that has already been loaded, or if you are upgrading the application system then you can delete the meta data and reload it.

Prerequisites

To access the metadata in the sender system, you must establish an RFC connection to this system using the port maintenance function in the IDoc adapter [Seite 346].

Procedure

1. To find out what metadata has already been loaded, call the transaction Metadata Overview for IDoc Adapter (IDX2).

   The system displays a screen with the directory of all systems connected with the IDoc adapter (including a description) for which metadata has already been loaded. Choose Port Maintenance in IDoc Adapter (edoresop) to go to the corresponding transaction and to create additional ports [Seite 346].

2. Expand the individual systems to display the IDoc types and clients including a description for each system. The system displays the metadata for each connected system for which metadata has already been loaded.

3. To copy the metadata structure loaded from a particular system to another system (for example, to an SAP reference system), select the link to the corresponding IDoc type and choose Copy (Copy).

   The system displays a dialog box in which you can copy the IDoc type description to another system (Target Port).

4. To delete metadata that has already been loaded, select the link to the corresponding IDoc type and choose Delete (Delete).

   The system asks you if you really want to delete the selected structure.

5. To delete the metadata structure, choose Yes.

6. To load additional metadata, choose Create (Create).

   The system displays a dialog box where you can enter the IDoc Type including Enhancement and the system (Source Port).

7. Make the required entries and choose Continue.

   The new structure is inserted in the tree structure as follows:
   - If the structure originates from a system from which metadata has already been loaded, it is inserted below the structures already loaded from this system.
   - If the structure originates from a system from which no metadata has already been loaded, it is inserted together with the system below the already listed systems.
8. To display details about a metadata structure that has already been loaded, choose the link to the corresponding IDoc type. To display this structure in detail, choose Display (Display). The system displays the structure in a hierarchy tree.

9. To display the corresponding system (port [Extern]) and the basis type from the detailed display, choose Header Information (Header Information).

10. To display the segment versions of all segment types in the structure, choose All Segment Versions (All Segment Versions).

11. Select a segment to display all its fields.

RFC Adapter

Definition

The RFC adapter [Extern] enables you to use the functions of the Integration Engine [Seite 266] in existing SAP landscapes. It supports SAP systems from Release 3.1x. The RFC adapter is used by SAP components to connect to an Integration Engine using remotely called function modules (RFCs [Extern]).

Use

The RFC adapter maps RFC calls to XML messages and the other way around. You require the RFC Adapter to connect SAP components to the Integration Engine using the RFC interface [Extern].

On the one hand you require the RFC adapter if you want to process RFCs [Seite 351] using the Integration Engine. The RFC adapter converts the incoming RFC data to XML message format so that it can be processed by the Integration Server pipeline [Seite 267].

On the other hand, the receiver system determined by technical routing [Extern] may expect the data from the Integration Engine to arrive in RFC format. For this purpose, the Integration Engine pipeline must transfer the received XML message to the RFC adapter. The RFC Adapter converts the XML message to a valid RFC call and executes the call.

Structure

Essentially, the RFC adapter comprises two parts, namely an inbound adapter at the Integration Engine inbound channel, and an outbound adapter at the Integration Engine outbound channel. The metadata for the RFC calls involved is shared.

The inbound adapter converts an RFC call to XML format, starts the Integration Engine and then transfers the XML message.

The outbound adapter converts the XML message to RFC format and executes the RFC call.

Integration

The RFC adapter is part of the Integration Engine. The inbound RFC adapter is located before the Integration Engine pipeline, and calls this pipeline. Whereas the outbound RFC adapter is called by the pipeline; it can be regarded as a pipeline service [Seite 267] and is therefore part of the pipeline itself.

Both adapters use SAP Java Connector (JCo); the inbound and outbound adapters are implemented as RFC server programs. This approach enables you to use the functions of JCo to manage metadata which in turn enables RFC data to be mapped generically to XML and the other way around.
Features

You have the following three possible scenarios when using the RFC Adapter:

**SAP Legacy System to SAP Web AS**

```
R/3  Transform RFC Data to RFC XML  Mapping & Routing  SAP Web AS
```

**SAP Web AS to SAP Legacy System**

```
SAP Web AS  Mapping & Routing  Transform RFC XML to RFC Data  R/3
```

**SAP Legacy System to SAP Legacy System**

```
R/3  Transform RFC Data to RFC XML  Mapping & Routing  Transform RFC XML to RFC Data  R/3
```
SAP Legacy System to SAP Web AS

- Register the RFC server program of the inbound RFC Adapter at the R/3 gateway of the SAP legacy system.
- Change the RFC destination in the SAP legacy system to reference the registered RFC adapter.
- Provide mappings for request and response messages.
- Establish logical and technical routing.

SAP Web AS to SAP Legacy System

- Provide mappings for request and response messages.
- Establish logical and technical routing.

SAP Legacy System to SAP Legacy System

- Register the RFC server program of the inbound RFC Adapter at the R/3 gateway of the SAP legacy system.
- Change the RFC destination in the SAP legacy system to reference the registered RFC adapter.
- Provide mappings for request and response messages.
- Establish logical and technical routing.

Restrictions

Note the following restrictions:

- Guaranteed assignment for transactional RFCs (tRFC) only.
- No transactional context for more than one call / one message for each LUW (Logical Unit of Work).
- Not provided: SAPgui, Callbacks, qRFC, or RFC library clients
- Context in the called system is not kept open over multiple calls (calls are stateless; there is no session).
- Only the payload of a received message is evaluated. Additional attachments (such as trace attachments) are ignored and not forwarded.

RFC Processing with the Integration Engine

Purpose

The RFC adapter [Seite 349] enables you to process and execute RFCs [Extern] (Remote Function Calls) using the Integration Engine [Seite 266].

Processing RFC calls using the Integration Engine pipeline [Seite 267] is an alternative to processing XML messages that are generated using the proxy interface. This alternative is considered for all SAP applications that already use RFC calls, as well as for new applications of SAP components that do not yet have access to the proxy generation [Seite 186] functions.

For more information about RFCs see RFC Basics [Extern].
**Prerequisites**

**Sender Application Component**

A user for SAP Java Connector (JCo) must be maintained in the sender system to access the required metadata for the called function modules in the sender system.

The RFC adapter cannot be used for external programs since programs of this type that use the RFC SDK (Software Development Kit) do not have any metadata.

Existing applications do not need to be changed. However, all application systems that send RFCs require an RFC connection to the respective inbound RFC adapter. You must register the program ID for the inbound adapter with this RFC destination. The program ID is located in the RFC Adapter configuration file.

You cannot send IDocs using this RFC connection. You must sent IDocs to the RFC destination of the IDoc adapter [Seite 340].

To establish an RFC destination to the inbound RFC Adapter, proceed as follows:

1. Call the transaction **Display and Maintain RFC Destinations** (SM59).
2. Select an existing RFC destination of type R/3 Connections (for example, R/3_Old) by double clicking it.
3. Choose **Destination → Copy** and copy the RFC destination using a new name (for example, R/3_New).
4. Save the new RFC destination (R/3_New) and delete the old RFC destination (R/3_Old).
5. Create a new RFC destination (for example, RFC_Adapter) of type T (TCP/IP connection) as the RFC destination for the RFC Adapter.
6. Select the registration type **Registered Server Program** and in the field **Program ID** enter the corresponding program ID from the RFC Adapter configuration file.
7. If the sender system is a Unicode system, define the RFC destination as Unicode.
8. Using the name of the old R/3 connection (R/3_Old), create a new RFC destination of type L (logical destination with reference entry).
9. In the field **Reference Entry** enter the RFC destination (RFC_Adapter) for the RFC adapter.

Under the name of the old RFC destination (R/3_Old) – formerly an R/3 Connection, now a Logical Destination – the system calls the new RFC connection (RFC_Adapter) of type TCP/IP Connection for the RFC adapter.

The change to the target address (RFC destination) is transparent to the application but it alters the semantics of the RFC connection.

To return to the old destination, simply change the reference entry in the logical destination (R/3_Old), for example from RFC_Adapter to R/3_New.

**System with RFC Adapter and Integration Engine**

Each RFC inbound adapter requires an RFC connection to the Integration Engine. The RFC Adapter configuration file contains the required connection data for this.

Integration Engine outbound processing calls the outbound RFC Adapter with the fixed RFC destination AI_RFCADAPTER_JCOSERVER. The program ID for the outbound adapter must
be registered to this RFC destination. The program ID is located in the RFC Adapter configuration file.

Since the system with the Integration Engine is a Unicode system, the RFC destination of the registered RFC Adapter must also be defined as Unicode.

**Receiver Application Component**

The end points [Seite 231] in the Integration Directory [Extern] determine the respective receiver systems. If you know which receiver systems are called by using an RFC, specify them in the RFC adapter configuration file for optimization purposes.

**Process Flow**

The connected systems generate the corresponding RFC calls in the applications and send them to the RFC destination of the Integration Engine.

The RFC adapter is then called in the Integration Engine; the RFC adapter generates an XML RFC document (XML message) from the RFC data and transfers this document to the pipeline for routing [Seite 214] and mapping [Seite 143].

If necessary, the system also converts (in the opposite direction) and maps the results of RFC calls of this type.

**Inbound RFC Adapter**

Once the RFC call has progressed from the application to the inbound RFC Adapter through the RFC interface, SAP Java Connector converts the metadata to XML RFC format.

**Integration Engine: Inbound Channel**

A message ID is created for each XML RFC document and a message comprising a header and payload is sent to the Integration Engine. The header contains the technical connection data for the sender and receiver.

The message only contains one RFC call.

Incoming XML RFC documents are only persisted if they are based on a tRFC call (asynchronous message processing). Synchronous RFC calls are only persisted if logging [Seite 291] is activated.

**Logical Routing**

Once it has entered the Integration Engine, the message header contains the technical connection data (business systems) required to assign the sender and receiver of a message.

Depending on the message contents, the receiver can be identified using an XPath rule in logical routing [Extern]; it must have the status True for the receiver to be identified successfully. If no XPath rule is defined, the receiver business system assigned to the sender business system is selected.

**Technical Routing**

In technical routing [Extern] the logical receiver is assigned a physical target. For the message, the outbound RFC Adapter requires one RFC end point [Seite 236] (an RFC destination) from technical routing for each logical receiver.

This RFC end point originates from the Integration Directory. It is analyzed and provides the technical information for the RFC call:

- SAP logon client in the field *Client*.
- SAP logon user in the field *User*. 
- SAP logon password in the field **Passwd** (optional).
- SAP logon language in the field **Lang**.
- Message server host name (when using load balancing) in the field **Mshost**.
- Message server service (with load balancing) in the field **Msserv**.
- SAP system name (with load balancing) in the field **R3name**.
- Application server group (with load balancing) in the field **Group**.
- Host name of a particular SAP application server (without load balancing) in the field **Ashost**.
- SAP system number (without load balancing) in the field **Sysnr**.
- SNC mode (Secure Network Communication) in the field **Snc_mode**.
  - 0 without SNC (Default)
  - 1 with SNC
- SNC quality of service in the field **Snc_qop**.
- Own SNC name (if the default name is not to be used) in the field **Snc_myname**.
- SNC name of the SNC partner (RFC server) or SNC name of the message server (with load balancing) in the field **Scn_partnername**.
- Directory and name of the SNC library in the field **Snc_lib**.

**Mapping**
The RFC adapter does not make any special demand on mapping. The outbound RFC adapter must simply be provided with an RFC XML document. This either already exists or must be generated by using a mapping.

**Integration Engine: Outbound Channel**
The system calls the outbound RFC Adapter and transfers the XML RFC document (XML message) with the RFC end point data.

**Outbound RFC Adapter**
The task of the outbound RFC Adapter is to convert XML RFC to a valid RFC call and to execute this call.

The result of the RFC call is accepted by the outbound RFC Adapter, converted to XML (XML RFC response), if necessary, and returned to the Integration Engine for mapping.

---

**Displaying XML Messages in the IDoc/RFC Adapter**

**Use**
Use this function to track inbound and outbound XML messages in the IDoc or RFC adapter. The system displays an overview of which messages were sent or received by the respective adapter.

**Prerequisites**
You have called the transaction **SAP Adapter – Monitoring** (IDX5). The system navigates to the screen **XML Messages in IDoc/RFC Adapter**.
Features

On the screen XML Messages in IDoc/RFC Adapter you can use various selection criteria to select and list XML messages separately for the IDoc adapter [Side 340] and the RFC adapter [Side 349].

XML Messages in the IDoc Adapter

To display XML messages in the IDoc adapter, proceed as follows:

2. Select XML messages by specifying a time and date the messages were created, or enter a specific message GUID to determine a specific message to be selected.
3. Specify your selection criteria in the group box IDoc.

Selection Criteria:

<table>
<thead>
<tr>
<th>Selection Criterion</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>System ID</td>
<td>Name of port in IDoc adapter [Side 346]</td>
</tr>
<tr>
<td>IDoc Object</td>
<td>Name of a customer enhancement or IDoc basic type [Extern]</td>
</tr>
<tr>
<td>Direction</td>
<td>Direction of IDoc transfer</td>
</tr>
</tbody>
</table>

4. Choose Execute.

The system sorts and lists the XML messages according to the time that they were created. Messages that were routed through as IDocs are listed together and are identified by the same message ID (GUID).

Besides the specified selection criteria, the system also displays the following information:

<table>
<thead>
<tr>
<th>Display</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client</td>
<td>Client [Extern] in sender or receiver SAP component</td>
</tr>
<tr>
<td>Reference Number</td>
<td>IDoc reference number in IDoc adapter</td>
</tr>
<tr>
<td>Transaction ID</td>
<td>Transaction ID in IDoc adapter</td>
</tr>
</tbody>
</table>

The system also uses the list of messages to summarize the number of messages displayed for each direction.

To display a summary of the displayed information for each XML message, select the corresponding message in the list and choose Details (⋯).

To display additional information about a message, switch to monitoring [Side 290] of the Integration Engine [Side 266], by double clicking one of the messages listed. On this screen you can display the contents of the message and existing message versions, compare [Side 302] individual versions and reschedule messages with errors.

XML Messages in the RFC Adapter

You have three options if you want to display XML messages in the RFC adapter. You can do the following:

- Inbound and outbound messages
- Outbound messages only
- Inbound messages only
Displaying Inbound and Outbound Messages

Choose this option if you want to determine which XML messages entered or exited the RFC adapter according to particular selection criteria. Proceed as follows:

1. Choose **RFC Monitor**.
2. Select XML messages by specifying a time and date the messages were created, or enter a specific message GUID to determine a specific message to be selected.
3. Choose **Inbound and Outbound** in the group box **RFC** (default).
4. Specify your selection criteria in the group box **RFC**.

### Selection Criteria:

<table>
<thead>
<tr>
<th>Selection Criterion</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transaction ID</td>
<td>Transaction ID in RFC adapter</td>
</tr>
<tr>
<td>System ID</td>
<td>Name of sender or receiver SAP component (for example, BCE)</td>
</tr>
<tr>
<td>RFC Destination</td>
<td>Logical destination of original RFC call from sender</td>
</tr>
<tr>
<td>Function module</td>
<td>Name of function module called</td>
</tr>
<tr>
<td>Client</td>
<td>Client [Extern] in sender or receiver SAP component</td>
</tr>
</tbody>
</table>

5. Choose **Execute**.

The system sorts and lists the XML messages according to the time that they were created. Messages that were routed through as RFC calls are listed together and are identified by the same message ID (GUID).

Besides the specified selection criteria, the system also displays the function calls for each LUW (Logical Unit of Work).

**Displaying Inbound Messages Only**

Choose this option if you want to check whether an inbound asynchronous message actually arrived at the receiver, for example. You require the ID of the transaction where the message is located for this. Proceed as follows:

1. Choose **RFC Monitor**.
2. Choose **Inbound -> Outbound** in the group box **RFC**.
3. Enter the required transaction ID in the group box **RFC**.
4. Choose **Execute**.

The system displays all messages that arrived during the specified transaction.

All messages for which the system also displays an outbound transaction ID were sent successfully to the receiver.

Messages that do not have an outbound transaction ID are synchronous messages, messages that left the Integration Engine as proxy objects, or messages with errors that were therefore not sent.

**Displaying Outbound messages Only**

Choose this option if, for example, in the receiver system you want to locate the sender of the message, using a transaction ID. You require a transaction ID for this. Proceed as follows:

1. Choose **RFC Monitor**.
2. Choose **Outbound -> Inbound** in the group box **RFC**.
3. Enter the required transaction ID in the group box **RFC**.
4. Choose **Execute**.
The system displays all messages belonging to this transaction ID, including existing inbound transaction IDs.

### Plain HTTP Adapter

#### Definition

The plain HTTP adapter gives application systems the option of communicating with the Integration Engine and exchanging business data using a plain HTTP connection. Depending on the receiver system, outbound messages can be enhanced with certain information.

#### Use

The plain HTTP adapter is used by external systems to connect to the Integration Engine using the native HTTP interface (HTTP payload without SOAP envelope). These systems are connected using the internet communication framework of the SAP Web Application Server. For this purpose, the Integration Engine HTTP inbound channel contains an HTTP service delivered by SAP, called `/sap/xi/adapter_plain`.

There are two usage types for the plain HTTP adapter:

- **Synchronous processing**
  
  Ordering on the Internet with an order number as a response.

- **Asynchronous processing**
  
  Ordering on the Internet without confirmation of the purchase.

In principle, both scenarios are possible at the Integration Engine HTTP inbound and outbound channels.

In special cases for particular external servers with CGI scripts, for example marketplaces, it is possible to send the payload as an HTML form using HTTP post. The payload is enhanced correspondingly.

The plain HTTP adapter only evaluates the payload of a received XML message. Additional attachments (such as trace attachments) are ignored and not forwarded.

#### Structure

The plain HTTP adapter comprises two parts, namely an inbound adapter by the Integration Engine inbound channel and an outbound adapter by the Integration Engine outbound channel. The outbound adapter requires a corresponding endpoint from technical routing for each logical receiver of the message.

#### Plain HTTP: Inbound Channel

You communicate with the plain HTTP inbound channel of the Integration Engine using an HTTP address. The HTTP body is used as the payload for the message. The payload does not contain an envelope with control data for the message you want to create. You must code
the control data in the HTTP address as parameters; these parameters are obligatory to be able to generate a message.

Queries entering the plain HTTP adapter must have the following syntax:

http://<hostname:port>/<path>?<query-string>

The query string contains the following data:

- Sender Namespace
- Sender Interface
- Sender business system
- Quality of Service (BE or EO)
- Message ID (optional)

The message ID is 32 characters long and must only comprise the characters 0 to 9 and A to F. It is persisted and can therefore be checked for duplication. If the HTTP client (the sender system) cannot provide a message ID, the plain HTTP adapter generates a message ID and forwards it to the application through the Integration Engine. You can then carry out a duplication check at document level, for example, by using the order number.

**Plain HTTP: Outbound Channel**

The plain HTTP outbound channel is a service provided by the Integration Engine. The system is addressed when technical routing determines a corresponding end point for the logical receiver from logical routing. The system sends the payload of the message to an HTTP address using HTTP, without being enhanced with an envelope.

The plain HTTP adapter gives the receiver system the following specifications:

- Specifications regarding the end point for technical routing
  - Content type (default text/XML as type/subtype)
    Specifies the format that the message contents (the payload) should have.
  - Encoding (default UTF-8)
    Specifies the character set.
  - URL Encoding
    Presents the XML in a URL-enabled format; special characters that could be interpreted as control characters are replaced with escape characters.
  - Prolog
    Enhances the payload for particular servers (optional, see below).
  - Epilog
    Enhances the payload for particular servers (optional, see below).
  - Header fields for user name and password (&lt;name:value, &lt;password:value&gt;)

- Specifications regarding an HTTP address or an HTTP destination, see the transaction Display and Maintain RFC Destinations (SM59)
  - Host, port and path including query string
    That is: http://<hostname:port>/<path>?<query-string>
    You can identify the HTTP port of a SAP Web Application Server using the ICM monitor [Extern].
  - Proxy host and proxy service (optional if there is a firewall between the plain HTTP adapter and the receiver system)
The following are specifications that are only possible using an HTTP destination:

- User and password (optional for an authentication in the receiver system)
- SSL ID (Secure Socket Layer), if you want to use SSL encoding

**Enhancing the Payload**

Some external systems, for example, Web servers in marketplaces, can only process data if it is sent as an HTML form using HTTP.

A typical HTML form comprises named fields. When transferring a completed form to the server or a CGI program, the data must be transferred in such a way that the CGI script can recognize the fields that make up the form, and which data was entered in which field.

The plain HTTP adapter constructs this format using a prolog and an epilog. Therefore, there is a particular code method that separates form fields and their data from each other. This code method uses the following rules:

- Individual form elements, including their data, are separated from each other by the character `&`.
- Name and data of a form element are separated from each other by an equals sign (`=`).
- Blanks in the entered data (for example, in multiple words) are replaced by a plus sign (`+`).
- All characters with the (enhanced) ASCII values 128 to 255 (hexadecimal 80 to FF) are transcribed using a hexadecimal sequence, beginning with a percentage sign (`%`) followed by the hexadecimal value of the character (for example, the German umlaut ö in the character set ISO-8859-1 is transcribed as `%F6`).
- All characters that occur in these rules as control characters (`&`, `+`, `=` and `%`) are also transcribed hexadecimal in the same way as high value ASCII characters.

**Example**

```
&cmd=T
&loaderXML=
<?xml version="1.0" encoding="UTF-8">
<STARTTAG>...
</STARTTAG>
&end=yes
```

**Integration**

The plain HTTP adapter is part of the Integration Engine. The inbound adapter is fed by the Integration Engine [pipeline](#), and calls this pipeline. Whereas the outbound adapter is called by the pipeline; it can be regarded as a [pipeline service](#) and is therefore part of the pipeline itself.

**Processing Business Data Using the Plain HTTP**
Adapter

Purpose

The plain HTTP adapter enables you to receive from and to send business data to external systems, using the plain HTTP inbound/outbound channels of the Integration Engine. The standard exchange format for data in the Integration Engine is XML. However, using the plain HTTP adapter you can also send and receive data in HTML and ASCII. In a case such as this, you must configure the mapping correspondingly; this means that you must use a Java mapping.

Prerequisites

The HTTP service supplied by SAP must be assigned to the client in which you run the Integration Engine.

Process Flow

An HTTP client sends business documents to the Integration Engine using HTTP. A message is generated from the business document and transferred to the Integration Engine. If, on the other hand, the Integration Engine identifies a plain HTTP receiver, a business document is generated from the message and sent to the receiver. This communication is either synchronous (the client awaits a response to its HTTP request) or asynchronous.

Integration Engine: HTTP Inbound Channel

The Integration Engine is called by external systems using HTTP and an HTTP address. For this purpose, the HTTP inbound channel of the Integration Engine contains the HTTP service /sap/xi/adapter_plain, delivered by SAP.

The class CL_HTTP.PLAIN_INBOUND is provided as a request handler that processes an HTTP request synchronously (client awaits a response) or asynchronously.

Multi-part documents and HTTP requests with an empty HTTP body are not accepted and receive a response with status code 500 or 204.

The obligatory parameters of the HTTP address are analyzed and used to construct the message.

Mandatory Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>bs</td>
<td>Business system partner to identify the sender.</td>
</tr>
<tr>
<td>namespace</td>
<td>Namespace of the sender interface.</td>
</tr>
<tr>
<td>interface</td>
<td>Interface of the HTTP payload.</td>
</tr>
</tbody>
</table>

p-ag.com@interface=/sap/orders

The optional parameters of the HTTP address are also analyzed. They affect the runtime response of the message in the Integration Engine.

Optional Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Meaning</th>
</tr>
</thead>
</table>
| qos     | Quality of service:  
|         | Synchronous (BE) or asynchronous (EO) processing; the default setting is synchronous processing. |
| msgguid | 32 character message ID; uniquely identifies a message in the Integration Engine. If no message ID is specified, the system generates it internally. |

```
```

You must code special characters such as forward slash (/), hyphen (-), or colon (:) with escape characters (for example, `%2F` for `/`, `%2D` for `-`, and `%2A` for `:`).

Optional HTTP header fields are ignored.

The payload in the HTTP body is appended as a binary document to the message. The payload should be sent as an XML document with code page UTF 8, so that all services of the Exchange Infrastructure can process the document.

The instantiated message is transferred to the Integration Engine. In mode BE (synchronous processing) the system waits for the response from the Integration Engine. If processing is successful, the payload of the message is sent as an HTTP response. If not, the system returns an error.

If processing is successful, the HTTP client has the return code 200. Otherwise, it has the return code 500, or 409. If an error occurs, the error object of the message contains an error message. In the synchronous case, this is placed in the HTTP body of the HTTP response; in the asynchronous case, it is given as the error cause to the HTTP request.

**Integration Engine: HTTP Outbound Channel**

The plain HTTP adapter is called by the Integration Engine if technical routing identifies a plain HTTP end point [Seite 240] for the logical receiver. This end point originates from the Integration Directory [Extern]. It is analyzed and provides the technical information for the HTTP request.

The main payload of the message is sent as HTTP post to an HTTP address. The technical data for the HTTP post is determined from the end point of the message. An HTTP client is instantiataed for a logical destination, in which the HTTP address, proxy data, user/password, and SSL are maintained. If no destination is specified, the system instantiates an HTTP client using the specified HTTP address and the proxy data. A basic authentication is entered in the HTTP header fields with a user and password in BASE64 code.

The optional header fields maintained in the end point are set in the HTTP header.

The header fields **Content Type** (if not already explicitly set), **Content Length** and the host are set by default.

In the HTTP address, the sender business system is set as parameter `bs` and the interface used as parameters `namespace` and `interface`. Depending on the current quality of service (qos) of the message, the parameter `qos` is also added to the HTTP address.

The HTTP body comprises the prolog, message payload, and epilog. Prolog and epilog are optional and **enhance the payload** [Seite 357] for particular servers (for example CGI servers). If the payload is to be sent as an HTML form using HTTP post, it is URL-encoded beforehand.

If the specified code page for the HTTP body does not correspond to the default setting UTF 8, the system converts the HTTP body to the corresponding code page; in the XML.
document, the system replaces the tag encoding=UTF-8 with the corresponding code page tag.

If the quality of service is set to *Best Effort*, the system awaits a response from the HTTP server. The system appends this response to the message as a new payload, while it deletes the old main payload.

The system logs any errors that occur in the message.

---

**Adapter Engine**

**Purpose**

The Adapter Engine is used to connect the Integration Engine [Seite 266] to external systems (for example, legacy systems). Using the various adapters in the Adapter Engine, XML and HTTP-based messages can be converted to the specific protocol and format required by such systems, and the other way around.

The individual adapters can be installed [Seite 363] either separately or together, and then configured [Seite 367] using the configuration screen of the Adapter Engine.

**Introductory Notes**

The Adapter Engine is a separate software component that is automatically installed on the Integration Server. However, you can also install it separately on another host if you wish.

**Features**

Depending on the configuration, the adapters can be used to do the following:

- Exchange data with external systems using a file interface or an FTP server (file/FTP adapter)
- Communicate with messaging systems using the JMS API (JMS adapter)
- Access databases using JDBC (JDBC adapter)
- Communicate with clients and providers of Web Services using SOAP (SOAP adapter)

An important distinction to be made here is whether an adapter is an inbound adapter (converts data from external systems) or an outbound adapter (converts data from the Integration Engine).

To begin functioning, each inbound or outbound adapter must be appropriately installed and configured, and finally started.

You can use different adapters simultaneously or different configurations of the same adapter. For this purpose, it may be necessary for you to change the installation parameters to enhance the installation [Seite 363].

You can also use the Adapter Engine configuration screen to configure the Java part of the RFC adapter. For more information, see the *Configuration Guide - SAP Exchange Infrastructure 2.0*. 
Installation

Purpose
You install the Adapter Engine [Page 362], including available adapters, so that you can then configure [Page 367] and use them to convert the respective message formats. You can configure adapters individually or together as required, but always separately for inbound processing (inbound adapter) and outbound processing (outbound adapter) by the Integration Engine [Page 266].

Prerequisites
To install adapters, you require the following:

- A Java development kit (JDK) 1.3
- The optional package Java Servlet Version 2.3 or above (servlet.jar). This can be located, for example, on the SAP Web Application Server or directly on the Internet at http://java.sun.com/products/servlet/index.html (download from servlet.jar). You can copy the package directly to the adapter installation directory (tech_adapter); otherwise it must be located in the Java CLASSPATH.
- Additionally for the JMS adapter
  - The optional package Java Messaging Service (jms.jar). This can be located, for example, on the SAP Web Application Server or directly on the Internet at http://java.sun.com/products/jms/index.html (download from jms.jar).
  - A JMS driver installation for the queue system to be connected to (for example, WebSphereMQ). The required Java libraries are product-specific and can be obtained from either the product vendor or other providers. The Java libraries must be located in the Java CLASSPATH following installation so that the JMS adapter can access the required Java classes at runtime.
- Additionally for the JDBC adapter
  - A JDBC driver installation (version 2.0) for the database product to be connected to. The required Java libraries are product-specific and can be obtained from either the database vendor or other providers. The Java libraries must be located in the Java CLASSPATH following installation so that the JDBC adapter can access the required Java classes at runtime.
- Additionally for HTTPS protocol
  - If the adapters are to communicate using HTTPS, additional libraries to implement the SSL protocol are required. You can download the Java libraries iaik_jce.jar and iaik_ssl.jar from the SAP Service Marketplace (http://service.sap.com) Software Distribution Center, under SAP Cryptographic Software, but ensure that you follow the regulations outlined there.

You must copy both libraries to the Java CLASSPATH after installation.
  - If the inbound adapter is to communicate with the Integration Server using HTTPS, you can use the HTTPS protocol instead of HTTP for XMB.TargetURL (https:// instead of http://).
  - If the outbound adapters from the Integration Server are to be communicated with using HTTPS protocol, you must configure the HTTP server correspondingly for the adapters. This is achieved using the HttpServer service [Page 376].
  - If the browser is to be connected to the Adapter Engine configuration screen using HTTPS protocol, you must configure the GuiBrowserEngine service [Page 376] correspondingly.
The Java runtime (java.exe) is located in the search path, if installation was performed correctly.

The packed adapter-installation file has been unpacked in a directory of your choice and you have retained the path structure.

**Process Flow**

**New Installation**

Within the directory you used to unpack the installation file you will find the sub-directory tech_adapter with further sub-directories such as Administration, Configuration and LogFiles.

To install an adapter, switch to the directory tech_adapter.

- To install the file/FTP adapter, execute the following:
  - Windows: install_fileadapter.bat
  - UNIX: install_fileadapter.cmd

- To install the JDBC adapter, execute the following:
  - Windows: install_jdbcadapter.bat
  - UNIX: install_jdbcadapter.cmd

- To install the JMS adapter, execute the following:
  - Windows: install.jmsadapter.bat
  - UNIX: install.jmsadapter.cmd

- To install the SOAP adapter, execute the following:
  - Windows: install_soapadapter.bat
  - UNIX: install_soapadapter.cmd

These scripts generate the default configuration in the adapter menu [Page 369] for the respective adapter type. If you do not call any of the scripts then the default configurations for all adapters are found in the adapter menu. However, you can modify the configuration of the adapter menu in each case by using the functions available there.

- To start the Adapter Engine, execute the following:
  - Windows: run_adapter.bat
  - UNIX: run_adapter.cmd

When you have executed run_adapter.bat or run_adapter.cmd, you can call the Adapter Engine using a browser. The corresponding default address is http://<hostname>:81; when logging on for the first time, enter the user SAP and the password init. When you have created new users [Page 372], you must delete this initial user as soon as possible.

If the standard port 8200 of the Adapter Engine is already in use, you must enter a free port under port= in the file GuiBrowserEngine.properties.

⚠️

If this port is also already being used, the Adapter Engine displays the following error message in the window in which it was started and then it closes itself.

ERROR starting Adapter engine base service ‘GUI’

Installation is now complete. On the Adapter Engine configuration screen you find two adapters (inbound and outbound) for each of the installed adapter types (inbound and outbound) that you can configure there. Note that only one SOAP adapter is required.
You define the difference between inbound and outbound for the SOAP adapter during configuration. This means that following installation, there is only one SOAP adapter for which you must then configure two different instances, if necessary.

To run multiple adapters of the same adapter type or different adapter types simultaneously, or to start adapters automatically, go to the configuration screen [Seite 367] of the Adapter Engine.

**Upgrading the Installation**

If you want to install a more recent adapter version, or want to repeat the installation, you have two options:

- The existing adapter configuration is not to be saved
  
  In this case, delete the existing installation beneath the installation directory completely and install the adapter as described above; the Adapter Engine does not contain any other data apart from this directory structure.

- The existing adapter configuration and user management are not to be saved

  a. Switch to the directory tech_adapter beneath the installation directory and execute the following file:
     
     **Windows:** save_configuration.bat
     
     **UNIX:** save_configuration.cmd
     
     Besides the directory tech_adapter, the system creates a new directory adapter_configuration.backup that contains all relevant files for adapter configuration and user management.
     
     If you have also changed the configuration of services and if these configurations also need to be saved, execute the following file as well:
     
     **Windows:** save_baseconfiguration.bat
     
     **UNIX:** save_baseconfiguration.cmd
     
     b. Delete the directory tech_adapter beneath the installation directory.
     
     c. Install the adapter as described above. Instead of the installation script, execute the following file in the directory tech_adapter:
     
     **Windows:** restore_configuration.bat
     
     **UNIX:** restore_configuration.cmd
     
     This file reproduces the adapter configurations, including user management. The services configurations are also reproduced if you have also saved them. Otherwise the services are available in their original configuration, as is the case following a new installation.

**Continuous Operation**

The following operating system-specific installation options are available to enable continuous operation of the Adapter Engine:

- **WindowsNT**
  
  The program SAPAdapterService.exe, located in the directory tech_adapter in the Adapter Engine installation directory, is used to start the Adapter Engine as an NTService. To install or de-install the NTService, use the script install_service.bat or uninstall_service.bat. The call conventions are as follows:
  
  o install_service.bat [ServiceName] [JavaVM]
The `uninstall_service.bat` [ServiceName] is a logical name that the service is created under, and `JavaVM` is the name of the DLL of the Java Virtual Machine with a complete path specification. Corresponding examples can be found in the script.

All jar files (for example the JDBC or JMS driver) that may additionally be required by the adapters must be located in the Java classpath when `install_service.bat` is called. Subsequent changes to the classpath have no affect on the created NTService.

Once installation is complete, you must start the Adapter Engine as an NTService in the Windows services menu or configure it the same as each NTService (startup type, logon information, and so on).

The Windows results-display displays information and error messages of the service under the corresponding service names in the application log. Adapter Engine console and error output is placed in the files `stdout.log` or `stderr.log` respectively in the sub directory `LogFiles`.

- **UNIX**
  
  The script `run_adapter.sh` is located in directory `tech_adapter` for Unix operating systems. You must adjust the variable `ADAPTER_HOME` to the installation path in this script. In the variable `EXTLIBS` you can specify all Jar files that are actually required by the Adapter Engine, for example to be able to load a JDBC or a JMS driver. For further details, refer to the `USAGE` section of the script. You can then insert the script in the boot procedure of the operating system.

**Accessing the SAP System Landscape Directory**

You can access the SAP System Landscape Directory [Seite 442] (SLD) from the Adapter Engine. When the Adapter Engine starts, it can log on to the SLD, whereby the Runtime Workbench [Seite 436] recognizes it as an SAP Exchange Infrastructure component, thus making it globally accessible.

Optionally, associations can also be created between individual outbound adapters and business systems if a business system is specified in the configuration of an outbound adapter. The addresses of the associated adapters are then automatically adopted as default values by the Integration Directory [Extern] when end points are created for the business system.

Moreover, this enables information connected with business systems to be exported. For more information about configuring this service, see the section `SLDaccessor` in Services [Seite 376].

Note that this service is not mandatory and the Adapter Engine can function without being connected to the SLD.

**Monitoring with the Runtime Workbench**

The Runtime Workbench can access the Adapter Engine for monitoring purposes. The same HTTP port as for configuration is used for this purpose. To log on you require the technical user `XIRWBUSER` and the password `XIPASS`, but no other authorizations. This user is created automatically when the Adapter Engine is installed. If this user is deleted (for example, when an older configuration is implemented following installation), or if the Runtime Workbench requires another user, you must create this user manually. The user to be created only requires the authorization group `Locked`. For more information about creating users, see user management [Seite 372].

You can also call the Adapter Engine administration screen from the Runtime Workbench. To be able to do so, both administration and monitoring settings must be maintained in the SAP System Landscape Directory. For more information, see the Runtime Workbench [Seite 436] documentation.
Configuration

Purpose
You need to configure the available adapters so that you can use them to convert the respective message formats. Using the intended Adapter Engine [Seite 362], you can configure adapters individually or together as required, but always separately for inbound processing (inbound adapter) and outbound processing (outbound adapter) by the Integration Engine [Seite 266].

You can configure the following adapters:

- Inbound File/FTP Adapter [Seite 381]
- Outbound File/FTP Adapter [Seite 394]
- Inbound JDBC Adapter [Seite 401]
- Outbound JDBC Adapter [Seite 406]
- Inbound JMS Adapter [Seite 416]
- Outbound JMS Adapter [Seite 423]
- SOAP Adapter [Seite 431] (Inbound or Outbound)

The Adapter Engine also provides you with services [Seite 376] that can be used by all adapters and that are also partially configurable. This includes the following:

- An HTTP(S) server for browser-based configuration
- Central user management [Seite 372]
- A central storage location for passwords [Seite 374] that are required in adapter configurations
- Option of displaying additional log data when searching for errors
- A test environment [Seite 378]
- Access to the SAP System Landscape Directory [Seite 442]
- An HTTP(S) server for the outbound adapter

Prerequisites
You have successfully installed [Seite 363] the Adapter Engine and the required adapters, started the Adapter Engine and called it using your web browser and are now in the Overview section of the Adapter Engine configuration screen.

Process Flow
The Main Menu of the Adapter Engine configuration screen comprises the following menu options that you can call with a single mouse-click:

- Adapters [Seite 369]
- Documentation (see below)
- User Management [Seite 372]
- Password Management [Seite 374]
- Services [Seite 376]
- Traces (see below)
- Test environment [Seite 378]
- About the Adapter Engine
• Log Off Current User (see below)
• Shutdown Adapter Engine (see below)

Documentation
Here you find the available language versions of the Adapter Engine documentation in PDF format. To open a document, ensure that you have a suitable viewer installed then click the mouse button once.

Traces
This menu option is only available for users with the Administrator role.

Choose Traces to display the internal traces of the adapter engine. The contents of these files are essentially only required by the SAP Hotline for the detailed analysis of problems and are not necessary for the day-to-day operation of the Adapter Engine. For this reason, this section is not documented.

In the standard configuration, five files are written, each 2MB in size. The current file can be identified from the time stamp. If required, the most recent 10 MB of trace information for the Adapter Engine is available here. The size or number of these files and the scope of the information is controlled by the file logging.properties in the directory tech_adapter. You must only change this file if requested to do so by the SAP Hotline.

To display a trace file, click the file name. The file is displayed as a Text/Plain document. It is displayed in a text viewer or directly in the browser, depending on the configuration of the browser.

About the Adapter Engine
Here information about the installed version of the Adapter Engine and the Java runtime environment used is displayed.

Users with the role Administrator can set the default setting for the way that the graphical user interface of the Adapter Engine is displayed. You can choose between SAP Web and SAP GUI. The setting you make here becomes the standard view for all other users provided that a different user interface has not been selected in the user settings.

Changes to the graphical user interface take immediate effect. However, depending on the browser cache settings, certain parts of the old display may remain visible until they are loaded again, or until the browser is restarted.

Log Off Current User
Choose this menu option to log off the current user. Note that this action merely logs the user off the configuration screen; the adapter status remains unchanged.

Shutdown Adapter Engine
This menu option is only available for users with the Administrator role.

Choose this menu option to shutdown the Adapter Engine completely. To restart the Adapter Engine you must run the script run_adapter, as described in the installation [Seite 363] procedure. Since you need to log on to the Adapter Engine host locally to do this, you can only shutdown the Adapter Engine from a browser started on the same host. To confirm that the Adapter Engine has been shutdown, the system displays the following text:

Adapter Engine shutdown, restart with script XXX

XXX is the script name including the complete path.

Once you have shutdown the Adapter Engine, you can no longer perform any actions in the menu options because the respective HTTP server no longer exists. Any attempt to do so triggers the following HTTP error:

The page cannot be displayed
**Result**

You have configured the selected adapters. To use the existing test environment [Seite 378] to test the conversion of the respective message formats, choose Test Environment in the Overview screen.

---

**Adapters Menu**

**Use**

The Adapters menu of the Adapter Engine [Seite 362] enables you to add new adapters, deactivate existing adapters, and reactivate or remove deactivated adapters. Furthermore, you can manage existing adapters in the detailed view.

**Prerequisites**

You have installed [Seite 363] the Adapter Engine and the adapters you require, started the Adapter Engine and called it using your web browser.

**Features**

The Adapters menu is displayed on the right-hand side when you call the configuration screen [Seite 367]. It displays the available (installed) adapters with their current status, as well as a traffic light that displays the condition of this status. A green traffic light means that the adapter is free of errors, red means errors have occurred, and amber means that the condition of the status is unknown.

You have the following options from the menu:

- **Rename Adapter**
  
  Use this function to rename an existing adapter. To do so, proceed as follows:
  
  a. Choose the function by clicking it.
  
  A selection list appears containing all the adapters that are currently activated.
  
  b. Select an adapter from the list.
  
  c. Enter a new name for the adapter.
  
  d. If you want the adapter to be started automatically when the Adapter Engine is started, check the checkbox **Automatic Startup**. If you do not check the checkbox, the adapter remains in the status **INITIALIZED** following a restart and must be started manually from the adapters menu.
  
  e. Choose **Rename** to complete the renaming process.
  
  Once the adapter has been renamed, the configuration is displayed in the selection list of the adapter menu under its new name.

- **Add New Adapter**
  
  Choose this function to generate a new adapter configuration from an existing adapter. To do so, proceed as follows:
  
  a. Choose the function by clicking it.
  
  A selection list appears containing all the adapters that are currently activated.
  
  b. Select an adapter from the list.
  
  c. Enter a name for the new adapter.
d. If you want the adapter to be started automatically when the Adapter Engine is started, check the checkbox **Automatic Startup**. If you do not check the checkbox, the adapter remains in the status **INITIALIZED** following a restart and must be started manually from the adapters menu.

e. To create an adapter configuration under the new name that has the same settings as the adapter configuration that you used as a template, choose **Create**.

The configuration is now displayed under the new name in the selection list of the adapters menu and can be edited there to adjust the settings copied.

⚠️

You must not activate identical adapter configurations under different names. This is usually not technically possible and leads to an error message when you attempt to initialize the new adapter. In isolated cases however, it is possible and can lead to unexpected results with messages being duplicated or lost.

- **Deactivate Adapters**

Choose this function to remove (deactivate) existing adapters from the adapters menu. However, the adapter configurations are not deleted and can, if required, be re-included (reactivated) in the adapters menu. Proceed as follows:

  a. Choose the function by clicking it.
      
      A selection list appears containing all the adapters that are currently activated.
  
  b. Select one or more adapters from the list.
  
  c. To remove the selected adapter(s) from the adapters menu, choose **Deactivate**.

- **Reactivate Deactivated Adapter**

Choose this function to re-include existing, but deactivated adapters in the adapters menu. Proceed as follows:

  a. Choose the function by clicking it.
      
      A selection list appears containing all the adapters that are currently deactivated.
  
  b. Select one or more adapters from the list.
  
  c. If you want all the adapters selected to be started automatically when the Adapter Engine is started, check the checkbox **Automatic Startup**. If you do not check the checkbox, the adapters remains in the status **INITIALIZED** following a restart and must be started manually from the adapters menu.
  
  d. To re-include the selected adapters in the adapters menu, choose **Reactivate**.

⚠️

**Copying Adapter Configurations from Other Installations**

Since the name of a deactivated adapter is no longer managed by the Adapter Engine, but a properties file with the same name still exists in the configuration directory of the Adapter Engine, you can use this function to copy adapter configurations from other installations of the Adapter Engine.

To do so, copy the source file `<AdapterName>.properties` from the directory `../tech_adapter/Configuration` to the same directory in the target installation. It is then automatically displayed in the list of deactivated adapters and can be reactivated correspondingly.

💡

**Changing the Start Mode for an Adapter**
If you have defined an adapter so that it starts automatically, you can change the start mode by choosing the function **Deactivate Adapters**.

To do so, first deactivate the adapter(s) in question and the reactivate it/them again with the start mode required.

- **Remove Deactivated Adapters**
  
  Choose this function to delete existing deactivated adapters. Once you have deleted an adapter configuration it can no longer be reactivated. Proceed as follows:
  
  a. Choose the function by clicking it.
     
     A selection list appears containing all the adapters that are currently deactivated.
  
  b. Select one or more adapters from the list.
  
  c. To delete the selected adapter(s), choose **Remove**.

- **Refresh**
  
  To update the displayed status and condition of all adapters, choose **Refresh** to reload the adapter menu.

**Activities**

To display details about the displayed adapter, double-click it.

The detailed view displays the name, type, version, and status of the adapter. As in the adapter menu, a traffic light specifies the condition of the current status. A green traffic light means that the adapter is free of errors, red means errors have occurred, and amber means that the condition of the status is unknown.

The **Log** area displays an adapter-specific log if one exists.

From the detailed view you can do the following:

- **Return to the Main Menu**

- **Update the contents currently displayed, by choosing **Reload**

  The current status of the adapter is displayed in each case (**INITIALIZED**, **STARTED**, **STOPPED**, or **TERMINATED**). Due to the partly asynchronous processing of the corresponding actions described below, the status displayed might also just be an intermediate status. This is particularly true for **Restart**, which involves the statuses **STOPPED**, **TERMINATED**, **INITIALIZED**, and **STARTED**. In this case, you can update the display by choosing **Reload**.

- **Configure, start, create, restart, stop, terminate the adapter**
  
  o **Configure**

  To call the configuration that you require to configure an adapter, choose **Configure**. It then appears in a window in the **Configuration** area.

  The following options are available to those users with the user role **Administrator** or **Configurator**:

  - To switch between edit and view mode, choose **Edit/View**.
  
  - To save configuration data modified in change mode, choose **Store Configuration Data**.
  
  - To delete any changes made in change mode that have not yet been saved, choose **Cancel**.

  The system displays the name of the configuration underneath the window.
Essentially, changes made to the configuration are only adopted when you choose Store Configuration Data. If you exit the configuration screen by using any other means (for example, by choosing one of the buttons on the left of the detailed view), then the original configuration is maintained and is displayed again when you next call the configuration.

To activate an adapter configuration that you have changed, you must always reinitialize the adapter first. To do so, choose either Terminate and Create or Restart, which is the same as the button sequence Terminate, Create, and Start.

It is not sufficient to simply choose Stop and Start since this combination does not reinitialize the adapter and therefore the modified configuration data is not recognized.

- Start
  
  To start an adapter, choose Start. To do so, you must have already created and configured the adapter. Adapters that were stopped by choosing Stop can also be restarted again by choosing Start.

- Create
  
  To regenerate and initialize the adapter, choose Create. This is necessary if you have changed the configuration data or have deleted the current instance of the adapter using Terminate.

- Restart
  
  To restart the adapter, regardless of its status, choose Restart. This is necessary if you have changed its configuration, for example. This action is the same as the button sequence Terminate, Create, and Start.

- Stop
  
  To stop a running adapter, choose Stop. The adapter has now been stopped can either be restarted by choosing Start, or it can be stopped completely by choosing Terminate.

- Terminate
  
  To end an adapter instance, for example, if you no longer require it, or if it is not functioning as you want it to, choose Terminate.

- Display and delete the adapter-specific log

---

**User Management**

**Use**

Using the user management function of the Adapter Engine you can define various users whose authorizations are based on different roles.

**Prerequisites**

You have installed the Adapter Engine and the adapter you require, started the Adapter Engine and called it using your web browser. On the configuration screen, you have chosen the menu option User Management and the system now displays the menu with the same name.
Features
The role-based user concept of the Adapter Engine has the following roles:

- **Adapter Engine Administrator**
  
  Users with this role can display and change all configurations. The user SAP, created initially during the installation, has this role. User management ensures that at least one user has this role. This means that the last user with this authorization cannot be deleted, nor can his or her role be changed.

  Only this role has the authorization to perform the following:
  
  - Create and delete users and to change their roles and passwords
  - Call the configuration of the [services](#)
  - Display the internal [trace files](#) of the Adapter Engine
  - Shut down [#](#) the operating system process of the Adapter Engine

- **Adapter Configurator**

  This role has all authorizations except those of the administrator mentioned above. In particular, a user with this role can configure adapters displayed in the adapters menu [#](#). This also includes the authorization for storing passwords [#](#).

- **Adapter User**

  This role only has the authorization to use existing adapter configurations, in other words, all the functions available in the detailed view screen except that for changing the configuration.

- **HTTP Server User**

  Users can be assigned the role HTTP Service User in addition to any other role they have, or they can be assigned this role only. These users are required for authentication for the outbound adapter if the authentication mode is activated on the HTTP server.

Any functions not specifically mentioned here can be used by all roles.

All users can be assigned the status *Locked*. This status means that users are temporarily locked, without being deleted. As a result, you do not need to create them again when you unlock them.

Activities

User management provides various functions that you can execute by using the following menu options:

- **Display Users**

  Displays all users and the roles they have been assigned. This function can only be executed by Adapter Engine administrators.

- **Create User/Change User Settings**

  Choose this function to create a new user and to assign him or her a role, or to change the password or the role of an existing user, or both. This function can only be executed by Adapter Engine administrators.

  You cannot assign any other role to the last Adapter Engine administrator.

- **Delete User**

  Choose this function to delete an existing user. This function can only be executed by Adapter Engine administrators.
The last Adapter Engine administrator cannot be deleted.

- Change Current User Settings
  Here the user who is currently logged on can change both his password and the way that the graphical user interface (GUI) is displayed.
    - Change Password
      Once you have changed the password, you must log on to the Adapter Engine using the new password.
    - Change Graphical User Interface Style
      You can choose from the following:
      - Default
      - SAP Web
      - SAP GUI
      The default setting is that which was selected as the default setting for the Adapter Engine. You can override this default settings by selecting either SAP Web or SAP GUI.

  Changes to the graphical user interface take immediate effect. However, depending on the browser cache settings, certain parts of the old display may remain visible until they are loaded again, or until the browser is restarted.

Password Management

Use
The password management function of the Adapter Engine [Seite 362] enables you to protect saved passwords that are used in the adapter configurations (for example, those for logging on to the database, queue system, FTP server, or Integration Server).

Prerequisites
You have installed [Seite 363] the Adapter Engine and the adapter you require, started the Adapter Engine and called it using your web browser. On the configuration screen [Seite 367], you have chosen the menu option Password Management and the system now displays the menu Password Management: Tokens.

Features
This menu option is available for users with the Adapter Engine Administrator or Adapter Configurator role.
In the Tokens menu you have the following options:
- Display Tokens
  Choose this option to display all tokens currently registered.
- Create/Change Tokens
Choose this option to create a password for a registered token or to change a password.

- Delete Tokens

Choose this option to delete a token.

To save adapter configuration passwords so that they are concealed, you must activate password management by entering the following:

```
PWD.tokenHandler=PWtokens
```

Replace the password with the token and place it between the separators `<!%` and `%!`.

If the separators `<!%` and `%!` conflict with other parameter values in the adapter configurations they can be configured using the `services [Seite 376]`.

The corresponding password is saved in the Tokens menu on the password management screen.

During the initialization of the adapter, the properties whose values contain a token are shown. If a token is found that does not exist, initialization is terminated and a corresponding error message displayed. If the properties are not shown, although you specified a token, then you either placed the separators incorrectly and therefore the token was not recognized, or you did not activate password management.

**Example**

In the inbound file/FTP adapter [Seite 381] you require the user `XIUSER` with the password `roger` to log on to the Integration Server. In the file/FTP adapter configuration the following two parameters are required:

- `XMB.User=XIUSER`
- `XMB.Password=roger`

To protect the password, a token is created with the name `myPassword`.

Password management is activated by entering the following in the configuration of the file/FTP adapter:

```
PWD.tokenHandler=PWtokens
```

The password is then replaced by the token:

```
XMB.Password=<!%myPassword%!>
```

The separators `<!%` and `%!` are required to identify the token.

The password `roger` is then saved under the name `myPassword` in the token menu (without the separators). If the file/FTP adapter is reinitialized, a message appears in the file/FTP adapter log stating that the token was identified and replaced at runtime.
**Services**

**Use**

This menu option contains the services of the Adapter Engine [Seite 362] that contain configurable parameters. You configure these parameters using the same screen as for the adapters. This menu option is only available for users with the Adapter Engine Administrator role.

Any changes made to the configuration of the services are only activated when the Adapter Engine is restarted. It is not sufficient to simply restart the services (unlike with configuration changes to adapters).

**Prerequisites**

You have installed [Seite 363] the Adapter Engine and the adapter you require, started the Adapter Engine and called it using your web browser. On the configuration screen [Seite 367], you have chosen the menu option Services and the system now displays the menu with the same name.

**Features**

You have the following choice of services:

- **GUIBrowserEngine**
  
  The GUI browser engine represents a separate HTTP server with a configurable port:

  ```
  port=<portNo>
  ```

  This is the HTTP port that the browser can log on to. The default value is 8200 and must not be changed unless it has already been reserved.

  The following settings must not be changed under any circumstances:

  - zones=root
  - rootDirectory=Administration

  ```
  HTTP.transmission=plain|SSL
  ```

  Defines whether the browser and the Adapter Engine are to communicate using HTTP (plain) or HTTPS (SSL). The default is plain.

  The following additional specifications are required for SSL:

  - SSLcertificate=<p12-certificate name>
  - SSLcertificatePassword=<p12-certificate password>

  SSLcertificate specifies the complete file name of a password-protected certificate and SSLcertificatePassword specifies the corresponding password (the password can be protected by using the token concept, described above).

  To make the installation HTTPS-enabled you must install additional Java libraries that are available from the SAP Service Marketplace (http://service.sap.com). These IAIK libraries must be located in the Java CLASSPATH.

- **HttpServer**

  The HTTP server is used by the outbound adapters. You have the option of making the following settings during the configuration of the HTTP server:
http.authentication=none|basic

Define whether user-authentication is performed. In the basic setting, the HTTP client of the Integration Server must log on to the Adapter Engine with a valid user that has been assigned the role HTTP Server User. In the setting none, no authentication takes place.

The default is none.

HTTP.transmission=plain|SSL

Defines whether the Integration Server and the HTTP server of the Adapter Engine are to communicate using HTTP (plain) or HTTPS (SSL). The default is plain.

The following additional specifications are required for SSL:

- HTTP.SSLcertificate=<p12-certificate name>
- HTTP.SSLcertificatePassword=<p12-certificate password>

SSLcertificate specifies the complete file name of a password-protected certificate and SSLcertificatePassword specifies the corresponding password (the password can be protected by using the token concept, described above).

To make the installation HTTPS-enabled you must install additional Java libraries that are available from the SAP Service Marketplace (http://service.sap.com). These IAIK libraries must be located in the Java CLASSPATH.

• PWtokens

This is the service for replacing the password tokens in the adapter configurations. If required, you can change the beginning and end characters of the tokens centrally here.

- PWD.beginSeparator=<beginSign>
- PWD.endSeparator=<endSign>

beginSign and endSign can be any character string. Existing replacements of tokens in the adapter configurations must then be adjusted correspondingly.

• SLDaccessor

This service enables you to log on to the SAP System Landscape Directory and supports the following functions:

- Write access: When the Adapter Engine is started, its current configuration is transferred to the System Landscape Directory. This information is then accessed by the Runtime Workbench [Seite 436] for example, to establish a connection to the monitoring and configuration functions of the Adapter Engine, or it is accessed by the Integration Directory [Extern] to determine default values when creating end points for business systems if these are associated with an adapter.

- Read access: Configuration data can be accessed centrally for each adapter instance at runtime. This service can be activated by the inbound adapter to access the parameter XMB.TargetURL (which describes the address of the Integration Server) dynamically using the parameter XMB.SenderBusinessSystem.

To do so, the corresponding association must be maintained in the System Landscape Directory and the latter must be accessible. You configure the HTTP connection required to access the System Landscape Directory by using the following parameters:
The user must have the authorizations of group SAP_XI_APPL_SERV_USER on the Integration Server.

Read access to the System Landscape Directory can be activated in the inbound adapters by adding the following entry to the configuration:

```plaintext
XMB.SLDConfiguration=SLDaccessor
```

In this case, for the system specified with `XMB.SenderBusinessSystem`, the URL of the corresponding Integration Server in the SLD is read and is used instead of the value specified under `XMB.TargetURL`. Therefore, specify the following:

```plaintext
XMB.TargetURL=<fromSLD>
```

If the System Landscape Directory logon data is not maintained correctly, corresponding error messages can occur in the Adapter Engine console output when the service is initialized (for example, when the Adapter Engine is started). These error messages do not affect the other functions and can be ignored if access to the System Landscape Directory is not required.

---

### Test Environment

#### Use

The main function of the test environment of the Adapter Engine [Seite 362] is to test the settings of the individual adapters (JMS, File/FTP, JDBC, SOAP). It is possible to create test variants and to save these in the Adapter Engine.

#### Prerequisites

You have installed [Seite 363] the Adapter Engine and the adapter you require, started the Adapter Engine and called it using your web browser. On the configuration screen [Seite 367], you have chosen the menu option Test Environment and the system now displays the menu with the same name.

#### Features

There are three ways of sending client-side messages using HTTP in the test environment.

- The first message type is general MIME messages.

  The test interface receives a single part MIME message in text format and sends it to a specified address on the Internet. It comprises two input areas:
In the first input window, enter the parameters for the HTTP header in the same way that you entered the adapter parameters. In particular, you can set the values for XMB.TargetURL and XMB.ContentType, for example:

- XMB.TargetURL=http://p35841:3333/JMS
- XMB.ContentType=text/xml

In the second input window, specify the payload you want to send. The payload and the body of the HTTP request are identical.

With this test, you can easily generate JMS text messages using the mode JMSTunneling, with the JMS adapter, for example.

- The second message type is Integration Engine messages.

The test interface receives messages of this type with a text payload. It comprises two input areas:

- In the first input field, enter the data that you want to put in the message header. You set the parameters of the header data in the same way you set the parameters of the JMS file/FTP adapter or JDBC adapter.

- In the second input field, specify the text payload you want to send.

With this test, you can easily either send a message to the Integration Engine, or simulate a sending Integration Engine (for example, to test the configuration of an adapter), for example.

- The third message type is simple Web Service or SOAP messages.

The test interface receives messages of this type with a text payload. It comprises two input areas.

- In the first input field, enter the data that you want to put in the HTTP header. In particular, you can set the values for WS.targetURL and WS.SOAPAction, for example:

  - WS.SOAPAction=testservice

- In the second input field, specify the sender payload you want to send.

Using this test it is very easy to send a Web Service message to a SOAP adapter (for example to test the configuration of the adapter), for example.

**Activities**

If you have set the data correctly, choose Send Message to generate and send the message for all three text variants.

The system then sends you a review of the send procedure (Send Review). The upper output field contains the generated and sent data; the lower output field contains the received data or a specific error text, in the case of an error.

To load or delete a test variant, choose Load variant. To save a test variant as a default variant, choose Save variant as.

The variant sap_default is shipped by SAP and cannot be deleted or saved under the same name. If you attempt to save sap_default the system creates a new variant with the name default.

**Example**

As an example of the second message type, a message for the Integration Engine is generated from the following header data and payload and sent to the following address:
http://pwdf0436:8082/sap/xi/engine?type=entry

**Header Data**

- XMB.TargetURL=http://pwdf0436:8082/sap/xi/engine?type=entry
- XMB.SenderBusinessSystem=ExtAdapterSender
- XMB.SenderInterfaceNamespace=http://sap.com/xi/xidemo
- XMB.SenderInterfaceName=ExtAdapterSenderIF
- XMB.ReceiverBusinessSystem=ExtAdapterReceiver

**Payload**

```xml
test>Payload</test>
```

**Message**

Content type: Multipart/Related; boundary=4859e1d0-1ef1-11d6-a1c4-6fda44caeb12

Content length: 1495

```xml
--4859e1d0-1ef1-11d6-a1c4-6fda44caeb12
Content type: text/xml
Content length: 1189

<SOAP:Envelope xmlns:SOAP='http://schemas.xmlsoap.org/soap/envelope/'>
  <SOAP:Header>
    <sap:MessageHeader
      xmlns:sap='http://sap.com/exchange/MessageFormat' version='1.0'
      SOAP:mustUnderstand='1'>
      <sap:From><sap:Name>ExtAdapterSender</sap:Name>
        <sap:Interface namespace='http://sap.com/xi/xidemo'>ExtAdapterSenderIF</sap:Interface>
      </sap:From>
      <sap:To><sap:Name>ExtAdapterReceiver</sap:Name>
      </sap:To>
      <sap:MessageId>48585b30-1ef1-11d6-8726-6fda44caeb12</sap:MessageId>
      <sap:QualityOfService>BestEffort</sap:QualityOfService>
      <sap:Direction>Request</sap:Direction>
      <sap:TraceLevel file='mytrace.trc'>1</sap:TraceLevel>
      <sap:Logging file='mylog.log'>1</sap:Logging>
      <sap:Document>MainDocument</sap:Document>
    </sap:MessageHeader>
  </SOAP:Header>
  <SOAP:Body>
    <sap:Manifest xmlns:sap='http://sap.com/exchange/MessageFormat'
      xmlns:xlink='http://www.w3.org/1999/xlink' version='1.0'>
      <xp:Payload xlink:type='simple'
        xlink:href='cid:payload-48585b311ef111d694636fda44caeb12@sap.com'>
        <xp:Document>MainDocument</xp:Document>
      </xp:Payload>
    </sap:Manifest>
  </SOAP:Body>
</SOAP:Envelope>
```

```xml
--4859e1d0-1ef1-11d6-a1c4-6fda44caeb12
Content type: text/plain
Content length: 22
Content ID: payload-48585b311ef111d694636fda44caeb12@sap.com
```

**Result**

The system displays the sent and the received message in the screen *Send Review*. 
Configuring the Inbound File/FTP Adapter

Use

You need to configure the inbound file/FTP adapter to send file contents to the Integration Engine [Page 266].

To configure the adapter you must specify the following:

- A directory and file name with a placeholder (*) to select the files
- A processing rule for successfully sent data
- Simple routines to convert a text file (with columns of fixed length or columns that are separated by particular characters, or both) to an XML document (these specifications are optional).
- A dispatcher class (optional) with the corresponding settings and the user exits and their settings that are to be called by the dispatcher
- Sender and receiver data required by the Integration Engine for routing and mapping

Prerequisites

You have:

1. Installed [Page 363] the corresponding adapter.
3. Called the adapter configuration by choosing Configure.

Procedure

The configuration of the inbound file/FTP adapter comprises five functional sub areas:

1. Java class name for the inbound file/FTP adapter
   Specify the class name as follows:
   ```
   classname=com.sap.aii.messaging.adapter.ModuleFile2XMB
   ```
   This specification is mandatory.

2. File/FTP adapter mode
   Specify the mode of the inbound file/FTP adapter. The following values are permitted:
   - **mode=FILE2XMB**
     The content of the file is sent directly to the Integration Engine.
   - **mode=FILE2XMBWITHROWCONVERSION**
     The system expects a text file that contains identically structured rows that can be converted into an XML document. To make this type of conversion, you must make the necessary specifications in step 5.
   - **mode=FILE2XMBWITHSTRUCTURECONVERSION**
     The system expects a text file with more complex file structures that can be converted into an XML document. A file of this type contains various row formats in logical structures. To make this type of conversion, you must make the necessary specifications in step 6.
   - **mode=FILE2XMBSTREAM**
     A special file format is expected here that represents a completely serialized Integration Engine message. This file format can be generated from the outbound file/FTP adapter [Page 394] in the mode XMBSTREAM2FILE. This
format can therefore be used to temporarily save complete Integration Engine messages, including all Integration Engine-specific parameters as described under point 3. Therefore, only parameters required for logging on to the Integration Engine (namely, XMB.TargetURL, or XMB.User, XMB.Password, XMB.Client, and XMB.Language) are evaluated from the configuration in this mode.

The default value is FILE2XMB.

3. The dispatcher class and the user exits to be called by the dispatcher (optional)

The inbound file/FTP adapter provides you with a dispatcher that you can use to convert messages before they are sent. The settings required for the dispatcher are explained in the example [Seite 435].

4. Specifications for the Integration Engine

Enter the complete address (URL) of the Integration Engine that you want to send the message to:

XMB.TargetURL=\texttt{http://IntegrationEngineHost:port/pipeline-arguments}

This specification is mandatory.

The Integration Engine address can also be accessed dynamically from the SAP System Landscape Directory. The following entry is added to the configuration for this purpose:

XMB.SLDConfiguration=SLDaccessor

In this case, for the system specified with XMB.SenderBusinessSystem, the URL of the corresponding Integration Server in the SLD is read and is used instead of the value specified under XMB.TargetURL. Therefore, specify the following:

XMB.TargetURL=<fromSLD>

To be able to access the System Landscape Directory, the SLDaccessor service [Seite 376] service must be configured correspondingly and the respective entries must be maintained in the System Landscape Directory.

If the Integration Server is accessed by means of an HTTP proxy server rather than directly, the following parameters must be set:

XMB.proxyHost=<proxyHostname>
XMB.proxyPort=<proxyPortnumber>

<proxyHostname> is the host name of the proxy and <proxyPortnumber> is the port where the proxy receives HTTP requests (for example, 8080).

If an authentication was specified for the specified URL (HTTP service) in the Integration Engine, the following specifications are mandatory:

- XMB.User=<user-name>
- XMB.Password=<password>

The specifications must match those that you made in transaction SICF in the Integration Engine. If you do not specify a user and password combination, or the combination is invalid, the system will terminate each attempt to access the
Integration Engine with the message Transport Exception: http-Error 401 – Unauthorized.

The user must have the authorizations of group SAP_XI_APPL_SERV_USER on the Integration Server.

If you want to use a different client or language to the default setting in the Integration Engine when logging on, you can also set the following parameters:

- XMB.Client=<client-no>
- XMB.Language=<language-id>

The following arguments are mandatory, except in FILE2XMBSTREAM mode (for the sender system). They are used to identify the adapter configuration in routing [Seite 214] and in mapping [Seite 143] in the Integration Engine pipeline [Seite 267]. You can also find an explanation about each argument there.

- XMB.SenderBusinessSystem=<configuration name>
- XMB.SenderInterfaceNamespace=<namespace URI>
- XMB.SenderInterfaceName=<name>
- XMB.ReceiverBusinessSystem=<configuration name>

Usually, the receiver system is determined by routing in the Integration Engine. This specification is not mandatory.

- XMB.ContentType=<type>

  Specifies the document type that is sent to the Integration Engine. The following values are permitted:
  - type=text/xml For XML texts
  - type=application/xml For binary data
  - type=text/plain For non XML texts

- XMB.ContentKind=<kind>

  Specifies the document data type. The following values are permitted:
  - kind=B (Binary data type)
  - kind=T (Text data type)

  The default value is T.

A code page can be set for text files by using the parameter file.encoding.

- XMB.QualityOfService=<QualityOfService>

  Specifies how the Integration Engine should process a message. The following values are permitted:
  - XMB.QualityOfService=BE (Best Effort: means synchronous processing)
  - XMB.QualityOfService=EO (Exactly Once: means asynchronous processing with guaranteed execution exactly once)
- XMB.QualityOfService=EOIO (Exactly Once in Order: means asynchronous processing using queues. This means guaranteed execution exactly once following the sequence of successive messages)

You must also define a queue name for EOIO:

- XMB.QueueId=<QueueName>
  This queue name is used in the Integration Engine to process messages in the same sequence that they arrived in.

5. Specifications for file selection and processing mode

The following specifications are mandatory:

- file.sourceDir=<directory_name>
  Specify the directory where the files to be processed are located.
  You can either specify the entire path name or a name relative to the working directory of the Adapter Engine. In all platforms (including Windows), use a forward slash (/) to separate directory names in accordance with Java specification.

- file.sourceFileName=<filename>
  Specify the name of the file that you want to process.
  The name can contain a placeholder (*) at any point, so as to be able to select a list of files for processing.
  The following are valid examples for filename:
  - myFile.txt
  - my*.txt
  - *.txt
  - *File.*
  - *File*.*
  - my*le.*
  Names that comprise more than two parts are also permitted.

- file.pollInterval=
  Specify the number of seconds that the adapter must wait if no files for processing can be found.
  You can also specify the following parameters so as to specify an additional waiting time in milliseconds:
  file.pollIntervalMsecs=
  If file.pollInterval is set to 0, then very short, almost real time processing times can be achieved.
  The default value for this parameter is 0.
  If the call is not to be repeated then you must set both parameters to 0 (or just the parameter file.pollInterval if you have not specified a value for parameter file.pollIntervalMsecs).
  In this case the adapter status remains as STARTED. To initiate a new call, choose Stop/Start or Restart to restart the adapter.

- file.retryInterval=
  Specify the number of seconds that the adapter is to wait before a file processed with errors is processed again.
The value from file.pollInterval is adopted as the default value. If this value is 0 (the file is not processed again), then files processed with errors are not processed again.

If file.retryInterval is also 0, then the adapter is terminated if an error occurs, even when a value greater than 0 was specified for file.pollInterval. In this case the adapter status remains as STARTED. To process the file again, choose Stop/Start or Restart to restart the adapter.

- file.logPollInterval=NO|YES
  All entries to loops that last longer than 5 seconds are shown in the adapter log by default. As a result, this can lead to a lot of entries in the log. In such instances, the log output (but not the loop itself) can be deactivated by using file.logPollInterval=NO.
  For loops that last 5 seconds or less, the output can be activated with file.logPollInterval=YES.

- file.processingMode=
  Once the files’ contents have been sent to the Integration Engine, specify how you want them to be processed.
  There are five possible processing types for processingMode: delete, archive, archiveWithTimestamp, setAttribute or test.
  delete means that successfully processed files should be deleted.
  archive and archiveWithTimestamp mean that successfully processed files are moved to an archive directory.
  In archive mode, the file name is copied unchanged. If a file with the same name already exists in the archive directory, it is overwritten.
  In the archiveWithTimestamp mode, a time stamp including the date and time is added to the file name. The format of the time stamp is yyyyMMdd-hhMMss-SSS_, where yyyy specifies a year and SSS the number of milliseconds. As a result, files can be sorted according to the time they entered the archive directory, irrespective of the file name. This also stops files with the same name being overwritten.
  In the case of both processing types, the archive directory must be specified using the following parameters:
    - file.archiveDir=<archiveDir>
      This parameter is mandatory for these processing types.
  setAttribute means that successfully processed files are given the attribute read only. This means that only files you can write to are processed.
  test means that successfully sent files will not be processed.

⚠️
This means that files are sent again as a new message if the file/FTP adapter is restarted or if the time period specified in file.pollInterval is exceeded. This mode is exclusively of use when testing the configuration of the file/FTP adapter or the Integration Engine, and not for production operation.

The following specifications are not mandatory:

- file.processingLocked=YES|NO
  Specify here whether files that are locked by other applications are to be transferred or not. The default setting is NO, meaning that only files that are not locked are transferred. In particular, this ensures that the adapter does not
transfer incomplete files that are generated or transferred by other applications. Only change the setting to **YES** if you can guarantee that such data losses will not occur.

- **file.processingOrder=<processingOrder>=byDate|byName**

  Specify here the sequence in which files are to be processed if you used placeholders when you specified file.sourceFileName, and the system found multiple files to be processed.

  The default value is **byName**. This value means that files are processed in alphabetical order. Alternatively, you can specify **byDate**. In this case, files are processed according to their time stamp in the file system, starting with the oldest file.

- **file.execute=<operating system command>**

  An operating system command specified here is executed after a file has been successfully processed. The default value is an empty character string (no command).

- **file.executeAfterAll=<operating system command>**

  An operating system command specified here is executed once all files have been successfully processed that were found in a run. The default value is an empty character string (no command).

- **file.encoding=<codepage>**

  This parameter is used to import text files. The default setting is to use the system code page that is specific to the configuration of the installed operating system. You can find the system code page under file.encoding in the general system information of the About Adapter Engine [Seite 367] menu option.

  You must specify the value UTF 8 for XML text documents that are coded in UTF 8. Text files that are neither in the system code page nor are coded in UTF 8 can only be read if the corresponding code page is available in the Java runtime environment.

If you select files from an FTP server and not from the file system, you need to make the following additional specifications:

- **ftp.host=<ftp-server>**

  The host name or IP address of the FTP server. If this specification is available, it is assumed that you are accessing an FTP server. The specifications file.sourceDir and file.sourceFileName refer to the FTP server. The following are then not possible specifications:

  - **setAttribute** for parameter file.processingMode.
  - **byDate** for parameter file.processingOrder.

- **ftp.port=<port-no.>**

  The port number of the FTP server. It is not obligatory that you specify the port number here. The default is the standard port for the FTP server (21).

- **ftp.user=<user name>**

- **ftp.password=<password>**

  A valid user name for the FTP server. It is not obligatory that you specify a user name. The default user is **anonymous** with the password **anonymous**.

- **ftp.connection=perFileTransfer|permanently**

  Use this specification to define whether a new connection needs to be established each time a file is transferred to the FTP server (value **perFileTransfer**), or whether a permanent connection should be used.
(value permanently). In this case, the connection is re-established automatically if it closed on the server side (for example, due to a timeout). The default value is permanently.

- ftp.mode=<Binary | Text>
  This specification enables you to set the transfer mode of the FTP connection to either text or data transfer. The default value is the payload type of the received document from the Integration Server.

6. Specifications for the conversion to an XML document in mode FILE2XMBWITHROWCONVERSION

If you set the inbound file/FTP adapter mode to FILE2XMBWITHROWCONVERSION, a text file with identical structures of a type is expected that is arranged line-by-line by default. However, it is possible to specify other separators to separate structures. The structure can be converted to an XML document and sent to the Integration Engine in place of the original file. This conversion occurs in addition to all other processing described in previous sections.

The rows of the text file must either contain separators, or have columns of fixed length, or both. You can either specify column names explicitly in this segment, or they can be read as header lines from the file itself.

You cannot specify any additional metadata (such as column types or conversion rules for columns). Information of this type must be made available during mapping in the Integration Engine pipeline. When the inbound file/FTP adapter converts non-XML to XML format, only an XML document for the mapping in the Integration Engine should be created.

The following mandatory arguments refer to the mode FILE2XMBWITHROWCONVERSION. In mode FILE2XMB, none of the following arguments should be set; existing arguments are ignored.

Make the following specifications:

- xml.processFieldNames=
  Specify the field names. The following are possible specifications:
    - notAvailable means that no field name information is available.
      notAvailable means that no field name information is assumed to be in the configuration or in the file to be converted. In this case, the columns in the XML document are identified using a simple counter tag (columnX>, X=0,1,2...).
    - fromFile means that the field name information is located in the header line of the file to be converted.
    - fromFileWithBlankLine corresponds to fromFile. After the header line there also follows a blank line or separator that is skipped.
    - fromConfiguration means that no header information exits in the file to be converted, but it will be delivered by the present configuration. You must set the value for xml.fieldNames= correspondingly for this (see below).

This specification is mandatory.

- xml.fieldFixedLengths=
  If you make a specification here, the system expects a character string that contains the lengths of the file columns as arguments separated by commas. If you also specify a separator for the columns, you must not add its length to the length of the columns.
If you do not make any specifications for xml.fieldSeparator=, the specification xml.fieldFixedLengths= is mandatory.

- xml.fieldSeparator=

  If you make a specification here, the system expects that the file contains the specified character string (one or more characters) as a separator between the individual columns.

  If you made no specification for fieldFixedLengths=, this is the only specification to identify the individual columns in a row.

  If you made a specification for fieldFixedLengths=, the extra length of the separator is taken into account, but no further consistency checks are performed.

You must specify at least either xml.fieldFixedLengths or xml.fieldSeparator.

- xml.enclosureSign=

  You can specify a string here that acts as a text delimiter. Text enclosed by such delimiters is transferred to the target structure unmodified although the default setting is to remove all text delimiters. Separators within such texts are ignored.

  This parameter is optional. The default setting is an empty value (no text delimiter).

- xml.enclosureSignEnd=

  If there are different text delimiters for the beginning and the end of a text, here you can specify the text delimiter for the end of the text. If you do not enter a value, the value from xml.enclosureSign is used instead.

- xml.enclosureSignEscape=

  Here you can specify a string that replaces the text delimiter if it occurs within a text that it delimits. When the text is transferred the string is replaced by the value specified in xml.enclosureSign. The default setting is no value.

- xml.enclosureSignEndEscape=

  Here you can specify a string that replaces the text delimiter for the end of the text if it occurs within a text that it delimits. When the text is transferred the string is replaced by the value specified in xml.enclosureSignEnd. The default setting is no value.

- xml.enclosureConversione=YES|NO

  This value specifies whether the text delimiter is to be removed or whether the escape character is to be replaced when the files are transferred. The default setting is YES; if you change the setting to NO, all characters are transferred unchanged.

  If you specify xml.enclosureSign=" and xml.enclosureSignEsc="", text enclosed in quotation marks is transferred unchanged and the quotation marks are removed.

  If the escape character for a quotation mark (""), itself occurs in the text it is replaced by the quotation mark during the transfer.

- xml.endSeparator=

  If you want to define an additional character string as a separator after the last column in a row, make a specification here. The system skips this separator
when it processes the last column (otherwise the system would treat it as part of
the last column).

The default value is a line break (no explicit separator after the last column).
Instead, the structures are arranged line-by-line.

- **xml.beginSeparator=**

If you want to define an additional character string as a separator before the first
column in a row, make a specification here. The system skips this separator
when it processes this column (otherwise the system would treat it as part of the
first column).

The default value is an empty character string (no separator before the first
column).

**Special Characters in the String for Separators:**

In all strings for separators (`xml.fieldSeparator`, `xml.beginSeparator`
and `xml.endSeparator`), you can specify non-printable ASCII characters.
These characters can be each be inserted individually in the string in the form
`\'0xHH\'` (including the quotation mark), where `HH` represents the character
coded as a hexadecimal value.

**Inserting Strings for Separators in the XML Document:**

The separators specified with `xml.beginSeparator` and
`xml.endSeparator` can also be inserted as fields in the structure of the
generated XML document. To do so, specify field names with the following
specifications:

```java
xml.addBeginSeparatorAsField=<fieldname>
```

and/or

```java
xml.addEndSeparatorAsField=<fieldname>
```

The strings, together with the specified field name, are then inserted either at
the start or the end of the structure, as they were specified in
`xml.beginSeparator` and `xml.endSeparator`. The definition of special
characters also needs to be included. Special characters cannot be converted
since characters of this type are not permitted in XML documents.

- **xml.fieldNames=**

If you specified `fromConfiguration` for `processFieldNames=`, this
specification is mandatory. The system expects a list of the column names. The
format of these column names varies depending on the following specifications:

If you specify a value for `xml.fieldFixedLengths=`, the system expects a
string that contains the names of the file columns as arguments separated by
commas (this is also true if you specify a value for `xml.columnSeparator=`).

If you only specify a value for `xml.fieldSeparator=`, the system expects a
string that contains the names of the file columns in the same format as the file
rows. This means that the same separator string and any additional strings you
specify for `xml.endSeparator` and/or `xml.beginSeparator` are expected.

- **xml.fieldContentFormatting=**

If you specify `trim` (the default setting), all blanks that proceed and follow a
found value are removed.

If you specify `nothing`, the value remains unchanged.
o xml.structureTitle=<anyName>
   The value you specify here is used as the name of the structure in the XML schema. The default is row.

o xml.documentName=<name>
   If you specify a document name, it is inserted in the message as the main XML tag. The default setting is resultset.

o xml.documentNamespace=<namespace>
   If specified, the namespace is added to the name of the document.

The document constructed from the specifications described looks as follows:

```xml
<documentName>
   <structureTitle>
      <field-name1>field-value</field-name1>
      <field-name2>field-value</field-name2>
      <field-name3>field-value</field-name3>
   </structureTitle>
   <structureTitle>
      <field-name1>field-value</field-name1>
      <field-name2>field-value</field-name2>
      <field-name3>field-value</field-name3>
   </structureTitle>
</documentName>
```

A document of this type cannot be processed further by the Integration Engine.

7. Specifications for the conversion to an XML document in mode FILE2XMLWITHSTRUCTURECONVERSION

In this mode, you can convert text files with more complex file structures to an XML output format. The files contain various row formats in logical structures.

The system expects a file with one or more logical structures (Recordsets). An unlimited number of recordsets (from one to all recordsets in the file) can be sent to the Integration Engine as separate messages.

A recordset can contain multiple types of substructures identified by logical names. There can be a fixed or variable number of substructures in a recordset. The format of a substructure of this type is fixed and corresponds to the description of the line format in step 5. A substructure must always be represented in exactly one line of the text document.

o Specifications for Recordsets:
  - xml.recordsetStructure=<NameA,nA,nameB,nB,>
    This string specifies the sequence and number of substructures (identified by logical names).
    Therefore, nA=1,2,3,... or * (for a variable, unlimited number, including 0). The same is the case for nB, and so on.
  - xml.recordsetName=<tagName>
    The value you specify here is used as the name of the structure in the XML schema. The default is Recordset.
  - xml.recordsetNameSpace=uri
If specified, this is added to the name of the recordset.

- **xml.recordsetsPerMessage=<noOfRecordsets>**
  1,..., * for an entire document in a message.
  Specifies the number of recordsets to be grouped together for one message. If the number of recordsets in a document is greater than the number specified, the adapter creates multiple messages from a document. The last of these messages might then contain fewer recordsets than specified.

  If you specify Exactly Once as the Quality-of-Service, each of these messages (that is to say, each part of a document from which a message is created) is sent to the Integration Engine exactly once. This is also true if the application is interrupted while creating the messages and restarted again later.

  If you change the file containing the document before the application is restarted, the file is treated as new. Any outstanding Exactly Once messages can no longer be sent for this file.

- **xml.documentName=<name>**
  This specification is mandatory if you set recordsetsPerMessage to a value greater than 1 or *
  If you specify a document name, it is inserted in the message as the main XML tag. If the generated message comprises more than one recordset, then this specification is mandatory since an XML document must contain exactly one main tag.

- **xml.documentNamespace=<namespace>**
  If specified, the namespace is added to the name of the document.

- **xml.documentSkipFirstRows=<noOfRows>**
  Specifies how many rows are to be skipped at the start of the document. The default is 0.

- **xml.keyFieldName=<fieldname>**
  If you specified a variable number of substructures for xml.recordsetStructure (at least one substructure with the number '*', the substructures must be identified by the parser from their content. This means that a key field must be set with different constants for the different substructures.
  In this case, you must specify a key field and the field name must occur in all substructures.
  If a structure has a fixed number of substructures, this specification is ignored.

- **xml.keyFieldType=CaseSensitiveString|CaseInsensitveString|Integer|Float**
  If you specify a key field, then you must also specify its type. The comparison with the specified values is based on this key field type (see xml.NameA.keyFieldValue= below).

- Specifications for Substructures:
Use the same specifications you made in step 5 and add the name of the substructure to the name of the respective property, separated by a period after the XML prefix.

xml.NameA.fieldNames instead of xml.fieldNames to specify the field names of substructure NameA.

Changes to the properties in step 5:

- xml.NameA.structureTitle=<<anyName>
  The default is not row here, but the structure name from xml.recordsetStructure. So the default is NameA.

- xml.NameA.processFieldNames=
  You do not need to make any specifications here since this property is defined by fromConfiguration in this mode. This means that the definition of the substructure must be contained in the configuration file. No structure information can be transferred in the file itself.

- xml.NameA.fieldNames=<listOfFieldnames>
  This specification is mandatory for the aforementioned reasons.

- xml.NameA.keyFieldValue=<valueInSubstructureName>
  This specification is also mandatory if you set xml.keyFieldName= in the recordset definition.

- xml.NameA.keyFieldInStructure=add|ignore
  Here you can specify whether the key field of the substructure should be added to the XML document (add) or ignored (ignore). The default is add.

- xml.NameA.endSeparator=
  Even if no specification is made here, a line break must follow since substructures are always expected as a line of the document.

The document constructed from the specifications described looks as follows:

```
<documentName>...
<recordset>
  <NameA>
    <field-nameA1>field-value</field-nameA1>
    <field-nameA2>field-value</field-nameA2>
    <field-nameA3>field-value</field-nameA3>
  </NameA>
  <NameB>
    <field-nameB1>column-value</field-nameB1>
    <field-nameB2>column-value</field-nameB2>
    <field-nameB3>column-value</field-nameB3>
  </NameB>
</recordset>
...```
The tag `<documentName>` is completed if you made a specification for `xml.documentName=`. If the message contains more than one recordset, this specification is mandatory. The number of recordsets is in turn dependent on the specification made for `xml.recordsetsPerMessage`.

A document of this type cannot be processed further by the Integration Engine.

8. Transport of further documents within a message to the Integration Engine

You have the option of transporting additional file contents within a message to the Integration Engine. The following rules apply for this procedure:

- The files must all be in the same directory that is defined in `file.sourceDir`.
- The files that are to be transported are grouped together using their file name, which must be identical except for one defined part (for example, the suffix).
- Additional file contents can only be transported unchanged, in other words, without the conversion options described in points 5 and 6.
- Messages of this type with multiple payloads require a corresponding receiver that can process them.

To add additional file contents, you must make the following specifications:

- `XMB.AdditionalPayloads=<NameA, NameB, ...>`
  This specification defines a logical name for each additional file that is then used in the subsequent parameters.

- `file.NameA.namePart="Main-NamePart"="Additional-NamePart"`
  This specification specifies which part of the original file name is to be replaced to create the name of the additional file.

The following example explains how three files of type `text`, `pdf` and `doc` are included in a message:

```
file.sourceFileName=x*.txt
```

The files `x1.txt` and `x2.txt` were found in the specified directory following this specification.

```
XMB.AdditionalPayloads=ddaPDF,ddaDOC
File.ddaPDF.namePart=".txt"=".pdf"
File.ddaDOC.namePart=".txt"=".doc"
```

The files `x1.pdf` and `x1.doc` were also searched for in the same directory as the file `x1.txt`. The contents of these three files are included in a message to the Integration Engine. Likewise, the files `x2.pdf` and `x2.doc` are searched for and processed like file `x2.txt`.

- `file.NameA.optional=YES|NO`
  Specify here whether the additional file is required or whether it is optional. If the file is not found for `YES`, the action is terminated with an error message. For `NO`, the specification is ignored and the message sent to the Integration Engine without the additional file contents. The default is `YES`.

- `XMB.NameA.ContentType=<type>`
Specifies the document type that is sent to the Integration Engine. The following values are permitted:

- `type=text/xml` (for XML texts)
- `type=application/xml` (for binary data)
- `type=text/plain` (for non-XML texts)

   - XMB.NameA.ContentKind=<kind>

   Specifies the document data type. The following values are permitted:

   - `kind=B` (Binary data type)
   - `kind=T` (Text data type)

   The default value is `T`.

---

### Configuring the Outbound File/FTP Adapter

**Use**

You need to configure the outbound file/FTP adapter to save XML messages from the Integration Engine [Seite 266] in files.

To configure the adapter you must specify the following:

- A directory and rule to determine the file names under which inbound messages are saved.
- A dispatcher class (optional) with the corresponding settings and the user exits and their settings that are to be called by the dispatcher.
- The address (HTTP port and URL) where the adapter can be contacted from the Integration Engine.
- Rules to convert an XML text file that represents a simple table structure, to a text document with columns of fixed length or columns that are separated by particular characters, or both (these specifications are optional).

**Prerequisites**

You have:

1. [Installed [Seite 363]] the corresponding adapter.
2. [Selected [Seite 367]] the adapter on the configuration screen.
3. Called the adapter configuration by choosing Configure.

**Procedure**

The configuration of the outbound file/FTP adapter comprises five functional sub areas:

1. Java class name for the outbound file/FTP adapter
   
   Specify the class name as follows:
   
   `classname=com.sap.aii.messaging.adapter.ModuleXMB2FILE`
   
   This specification is mandatory.

2. File/FTP adapter mode
Specify the mode of the outbound file/FTP adapter. The following values are permitted:

- **mode=XMB2FILE**
  The system saves the document sent from the Integration Engine as a file.

- **mode=XMB2FILEWITHCONVERSION**
  A text file is expected that contains an XML document that represents a table. This document can be converted to a text file that stores the contents of the document as a list, with elements separated by commas or with fixed column lengths. The expected XML format corresponds to the result of the conversion of the inbound JDBC or file/FTP adapter. You must make the necessary specifications for the conversion in step 5.

- **mode=XMBSTREAM2FILE**
  A special file format is generated here that represents a completely serialized Integration Engine message. This file format can be processed by the inbound file/FTP adapter [Seite 381] in the mode XMBSTREAM2FILE. This format can be used to temporarily save complete Integration Engine messages, including all Integration Engine-specific parameters that then no longer need to be specified in the inbound file/FTP adapter.

The default value is XMB2FILE.

3. The dispatcher class and the user exits to be called by the dispatcher (optional)

The outbound file/FTP adapter provides you with a dispatcher that you can use to convert messages after they have been received. The settings required for the dispatcher are explained in the example [Seite 435].

4. Specifications for addressing by the Integration Engine

- **XMB.httpPort=<port_no>**
  
  `<port_no>` specifies the HTTP server port that receives the messages from the Integration Engine.

- **XMB.httpService=<service>**
  
  `<service>` describes the service part of the address where the Integration Engine must send its messages.

These specifications are mandatory.

For example, if you have specified `XMB.httpPort=1234` and `XMB.httpService=/file/Receiver`, the end point address of the file/FTP adapter in the Integration Engine must be specified as follows:

http://<fileadapterhost>:1234/file/Receiver

The end point address must be extended as follows for the Integration Engine in Release 1.0:


If the Integration Engine message is sent to a non-specified adapter service, the system displays the following error message:

No registered listener for <Service> found

The system displays the same message if the adapter is initialized, but has not been started (status STOPPED or INITIALIZED).

- **XMB.ReceiverAdapterForBusinessSystem=<Business System>**
This specification is optional.

<Business System> is the logical name of a business system in the System Landscape Directory (SLD). If the Adapter Engine connection to the SLD is configured, an association for the outbound file/FTP adapter is created for this business system. When you create an end point for this business system in the Integration Directory [Extern], the address of the adapter is determined from this association as the default value.

If no connection can be established with the SLD, or if the business system does not exist in the SLD, the specification is meaningless.

5. Specifications for file processing:
   - file.createDir=<n>
     Specify whether a target directory (targetDir) should be created (1) or not (0), if one does not already exist. The default value is 1 (create directory).

The following specifications are mandatory:
   - file.targetDir=<directoryName>
     Specify the target directory where incoming documents are to be saved. The system creates this directory automatically if you do not make any specifications for file.createDir.
   - file.targetFilename=<fileName>
     Specify the file name where the system is to save the first incoming document.
   - file.writeMode=<mode>
     Specify whether the system is to save all inbound documents under the file name that you specified in file.targetFilename. The following values are permitted:
       - file.writeMode=overwrite
         In mode overwrite, the file is overwritten so that the system always only saves the last received document.
       - file.writeMode=append
         In append mode the file is updated by appending the incoming documents.
       - file.writeMode=addTimeStamp
         In mode addTimestamp the system generates a new file for each document. The file name comprises the name specified under file.targetFilename, in each case extended by a timestamp in the format yyyyMMdd-HHmmss-SSS. This corresponds to the time that the document was processed, from the year (yyyy) to the millisecond (SSS), which is inserted before the file name extension. This mode guarantees that no data is overwritten. In this way, files that originate from the same adapter configuration can simply be sorted according to the date that they entered processing.
       - file.writeMode=addMessageId
         In mode addMessageId the system generates a new file for each document. The file name comprises the name specified under file.targetFilename and the message ID (GUID) used by the Integration Server in the format xxxxxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx.
         This mode guarantees that no data is overwritten.
**file.writeMode=addCounter**

In mode addCounter the system generates a new file for each document. The file name comprises the name specified in file.targetFilename, extended by one counter that is added before the file name extension (for example, default002.file). The counter is constructed from the subsequent specifications.

The mode addCounter guarantees that no data is overwritten. If necessary, the system finds the next free counter from which it can construct a new file name.

In mode addCounter the exact format of the file name can be controlled using the following parameters; each parameter has a default setting.

- **file.counterSeparator=</SeparatorString>**
  Specify one or more characters that you want to add before the numerator in the file name. The default value is an empty character sequence (that is, no SeparatorString).

- **file.counterFormat=</FormatString>**
  Specify the first counter to be used. It must be a valid integer number, though leading zeros are permitted. The length of the FormatString is used for all counters (including existing leading zeros) until overflow forces the format to be enlarged.
  The default value is 000.

- **file.counterStep=</StepNo>**
  Enter a number that specifies the increment of the counter that successive counters are to be increased by. The default value is 1.

- **file.counterMode=</counterMode>**
  Specify when a counter is to be added. The following values are permitted:

  - **file.counterMode=afterFirst**
    Only add the counter after the file name specified under file.targetFilename was used unaltered.

  - **file.counterMode=immediately**
    The counter should be added after the first incoming document.

  The default value is afterFirst.

If you do not set a parameter, the system generates the file names <default.file>, <default000.file>, <default001.file>, <default002.file> from the specified default values for incoming documents.

If instead of the default values, you set the following explicit values

- file.targetFilename=test.dat
- file.counterMode=immediately
- file.counterSeparator=_
- file.counterFormat=00005
- file.counterStep=2
the system generates the file names test_00005.dat, test_00007.dat, test_00009.dat and so on.

This list is also continued once the adapter has been stopped and restarted, without overwriting existing files.

- **file.execute=**<operating system command>

  An operating system command specified here is executed after a file has been successfully processed. The default value is an empty character string (no command).

- **file.exactlyOnceHandlingIDsExpiration=**<n Days>

  Specify the number of days that the management information for received messages with the service attribute Exactly Once (In Order) should be kept. This information is required to avoid messages being duplicated, if when handling an error the Integration Engine sends the same message to an adapter more than once. The default value is 60 days.

! IMPORTANT

If the value is set to 0 or to a negative value, all management information is deleted each time the adapter is initialized. This can be useful for test purposes but must not be used for production operation.

If a message is received with the service attribute Exactly Once, a temporary file with the name of the adapter configuration is generated first in the target directory. This file is then automatically renamed during the Exactly Once handling. It must not be renamed or edited manually in any way, or deleted since this can lead to the message being lost.

If you want to save files on an FTP server and not in the file system, you need to make the following additional specifications:

- **ftp.host=**<ftp-server>

  The host name or IP address of the FTP server. If this specification is available, it is assumed that you are accessing an FTP server. The specifications file.targetDir and file.targetFileName refer to the FTP server.

- **ftp.port=**<port-no.>

  The port number of the FTP server. It is not obligatory that you specify the port number here. The default is the standard port for the FTP server (21).

- **ftp.user=**<user name>

- **ftp.password=**<password>

  A valid user name for the FTP server. It is not obligatory that you specify a user name. The default user is anonymous with the password anonymous.

- **ftp.connection=**perFileTransfer|permanently

  Use this specification to define whether a new connection needs to be established each time a file is transferred to the FTP server (value perFileTransfer), or whether a permanent connection should be used (value permanently). In this case, the connection is re-established automatically if it closed on the server side (for example, due to a timeout). The default value is permanently.

6. Specifications to convert an XML document (mode XMB2FILEWITHCONVERSION)
The following arguments refer to the mode `XMB2FILEWITHCONVERSION`. In another mode the arguments subsequently executed are ignored.

To convert a table in XML format to pure text format, the document format must be the same as that which is generated by the inbound file/FTP adapter in mode `FILE2XMBWITHROWCONVERSION` (with the default value `<row>` for the structure) and the JDBC adapter. This means that the document looks as follows:

```xml
<resultset>
  <row>
    <column-name1>column-value</column-name1>
    <column-name2>column-value</column-name2>
    <column-name3>column-value</column-name3>
  </row>
  <row>
    <column-name1>column-value</column-name1>
    <column-name2>column-value</column-name2>
    <column-name3>column-value</column-name3>
  </row>
</resultset>
```

This is an example for 3 columns and 2 lines. The number of lines and columns is, of course, unlimited. The XML elements in italics can be defined by the user and the elements of a line must be grouped within an element of type `<row>`...

Using the following specifications, the system constructs a text file from this XML document.

- `xml.addHeaderLine=<n>`
  Specify whether the text file should have a header line with columns names. The following values are permitted:
  - 0 – No header line
  - 1 – Header line with column names from the XML document
  - 2 – as for 1, followed by a blank line
  - 3 – Header line is stored as `xml.headerLine` in the configuration and is copied
  - 4 – as for 3, followed by a blank line

  This specification is mandatory.

- `xml.headerLine=<headerLineString>`
  Specify the header line that is generated in the text file if `xml.addHeaderLine` has the value 3 or 4. This specification is mandatory.

- `xml.fieldFixedLengths=<String>`
  Specify a character string that contains a list of fixed column widths that are separated by commas and which determine the number and the length of columns generated in the text file.

  If you do not make any specifications for `xml.fieldSeparator`, the specification `xml.fieldFixedLengths` is mandatory.
Specify how you want the system to respond when column widths in the actual document exceed those defined in `xml.fieldFixedLengths`. The following values are permitted:

- `xml.fixedLengthTooShortHandling=Error`
  - Error means that processing of the document is terminated.

- `xml.fixedLengthTooShortHandling=Cut`
  - Cut means that a value is shortened to the maximum permitted length.

- `xml.fixedLengthTooShortHandling=Ignore`
  - Ignore means that the system copies the value completely, regardless of it being too long. Subsequent columns are moved correspondingly.

The default value is **Error**.

If you do not specify `xml.fieldFixedLengths`, this specification is obsolete.

- `xml.fieldSeparator=<SeparatorString>`
  - If you specify a character string here, it is added as a separator to all columns except the last. You can also specify this string in addition to `xml.fieldFixedLengths`.
  - If you made no specification for `xml.fieldFixedLengths`, this is the only specification to identify the individual columns in a row.
  - If you made a specification for `xml.fieldFixedLengths`, the length of the separator is not taken into account.

💡 You must specify at least either `xml.fieldFixedLengths` or `xml.fieldSeparator`.

If you have only specified `xml.fieldSeparator`, the structures of the XML document can have differing numbers of elements that are added to each other separately in the text file by the value from `xml.fieldSeparator`. If you specify `xml.fieldFixedLengths` this is not possible since the number of columns is defined when the column widths are given.

- `xml.endSeparator=<lastSeparatorString>`
  - If you enter a character string here, the system adds it to the last column as a closing character. You can also make this specification in addition to `xml.fieldFixedLengths`. To include a line break following the closing character, you must explicitly define it by attaching “\n” (including the quotation marks) to the string.

  The default value is a line break (no explicit separator after the last column, instead the structures are arranged line-by-line).

- `xml.beginSeparator=<firstSeparatorString>`
  - If you specify a character string here, the system places it before the first column. You can also make this specification in addition to `xml.fieldFixedLengths`.

  The default value is an empty character string (no separator before the first column).

💡 Special Characters in the String for Separators:
In all strings for separators (xml.fieldSeparator, xml.beginSeparator and xml.endSeparator), you can specify non-printable ASCII characters. These characters can be each be inserted individually in the string in the form "0xHH" (including the quotation mark), where HH represents the character coded as a hexadecimal value. A line break can be entered using the character "\n1" (including the quotation marks); \n1 stands for new line here.

**Fixed Column Width of the Text Document**

\texttt{xml.absoluteRowWidth=\langle\text{noOfColumns}\rangle}

If the separators do not contain any line breaks, the text document is generated as a single line text. To restrict the width of the text, use this parameter. \langle\text{noOfColumns}\rangle represents the maximum number of columns in this case. This parameter also functions together with the specification of a line break for xml.endSeparator.

This value requires \texttt{xml.addHeaderLine=0} to be specified.

---

**Configuring the Inbound JDBC Adapter**

**Use**

You need to configure the inbound JDBC adapter to send database contents to the Integration Engine [Seite 266].

To configure the adapter you must specify the following:

- A JDBC database driver to access the database
  This driver is not supplied with the adapter but must be supplied by the database provider or third-parties.
- Logon data for the database to be read from
- An SQL statement to determine the table contents you want to read
- An SQL statement to process successfully sent table contents
- A dispatcher class (optional) with the corresponding settings and the user exits and their settings that are to be called by the dispatcher
- Sender and receiver data required by the Integration Engine for routing and mapping

**Prerequisites**

You have:

1. Installed a JDBC driver (Version 2.0) for the database.
2. [Installed][Seite 363] the corresponding adapter.
3. [Selected][Seite 367] the adapter on the configuration screen.
4. Called the adapter configuration by choosing Configure.

**Procedure**

The configuration of the inbound JDBC adapter comprises four functional sub areas:

1. Java class name for the inbound JDBC adapter
Specify the class name as follows:

classname=com.sap.aii.messaging.adapter.ModuleDB2XMB

This specification is mandatory.

2. JDBC adapter mode

Specify the mode of the inbound JDBC adapter. The following is the only permitted value:

mode=DB2XMB

Subsequent versions may permit additional values.

3. The dispatcher class and the user exits to be called by the dispatcher (optional)

The inbound JDBC adapter provides you with a dispatcher that you can use to convert messages before they are sent. The settings required for the dispatcher are explained in the example [Seite 435].

4. Specifications for the Integration Engine

Enter the complete address (URL) of the Integration Engine that you want to send the message to:

XMB.TargetURL=http://IntegrationEngineHost:port/pipeline-arguments

This specification is mandatory.

The Integration Engine address can also be accessed dynamically from the SAP System Landscape Directory. The following entry is added to the configuration for this purpose:

XMB.SLDConfiguration=SLDaccessor

In this case, for the system specified with XMB.SenderBusinessSystem, the URL of the corresponding Integration Server in the SLD is read and is used instead of the value specified under XMB.TargetURL. Therefore, specify the following:

XMB.TargetURL=<fromSLD>

To be able to access the System Landscape Directory, the SLDaccessor service [Seite 376] service must be configured correspondingly and the respective entries must be maintained in the System Landscape Directory.

If the Integration Server is accessed by means of an HTTP proxy server rather than directly, the following parameters must be set:

XMB.proxyHost=<proxyHostname>
XMB.proxyPort=<proxyPortnumber>

<proxyHostname> is the host name of the proxy and <proxyPortnumber> is the port where the proxy receives HTTP requests (for example, 8080).

If an authentication was specified for the specified URL (HTTP service) in the Integration Engine, the following specifications are mandatory:

- XMB.User=<user-name>
- XMB.Password=<password>
The specifications must match those that you made in transaction SICF in the Integration Engine. If you do not specify a user and password combination, or the combination is invalid, the system will terminate each attempt to access the Integration Engine with the message "Transport Exception: http-Error 401 – Unauthorized.”

The user must have the authorizations of group SAP_XI_APPL_SERV_USER on the Integration Server.

If you want to use a different client or language to the default setting in the Integration Engine when logging on, you can also set the following parameters:

- XMB.Client=<client-no>
- XMB.Language=<language-id>

The following address arguments for the sender system (the JDBC adapter) are mandatory: They are used to identify the adapter configuration in routing [Seite 214] and in mapping [Seite 143] in the Integration Engine pipeline [Seite 267]. You can also find an explanation about each argument there.

- XMB.SenderBusinessSystem=<configuration name>
- XMB.SenderInterfaceNamespace=<namespace URI>
- XMB.SenderInterfaceName=<name>
- XMB.ReceiverBusinessSystem=<configuration name>

You only need to specify a receiver system if one is not determined during routing.

- XMB.QualityOfService=<QualityOfService>

Specifies how the Integration Engine should process a message. The following values are permitted:

- XMB.QualityOfService=BE (Best Effort, means synchronous processing)
- XMB.QualityOfService=EO (Exactly Once, means asynchronous processing)
- XMB.qualityOfService=EOIO (Exactly Once in Order, means asynchronous processing using queues, in other words guaranteed execution exactly once and maintaining the sequence of successive messages)

You must also define a queue name for EOIO:

- XMB.QueueId=<QueueName>

This queue name is used in the Integration Engine to process messages in the same sequence that they arrived in.

5. Specifications for database access

- db.jdbcDriver=<java_class_ofjdbcdriver>

Specify the Java class of the JDBC driver that the JDBC adapter must load to be able to access the driver. The exact specification varies according to the JDBC driver; see material from the respective provider for exact details. This specification is mandatory.

- db.connectionURL=<jdbc_driver_connection_url>
Specify the address with which you can establish a database connection using the JDBC driver. The exact format of the address can vary; see material from the respective provider for more details. This specification is mandatory.

- **db.reconnect=**YES|NO

  When a database connection error occurs, this parameter enables you to specify whether you want reconnect to the database via the JDBC driver each time the database is accessed. The default setting is YES. If you specify NO, when an error occurs the JDBC adapter terminates the connection to the database and it must then be restarted from the administration screen.

- **db.processDBSQLStatement=**<SQL-Select-Statement>

  Either specify a valid SQL SELECT statement to select the data to be sent from the specified database, or specify an SQL EXECUTE statement to execute a stored procedure that contains exactly one SELECT statement. The expression only needs to correspond to the SQL variant supported by the respective JDBC driver; so it can also contain table JOINs, for example. This specification is mandatory.

- **db.confirmDBSQLStatement=**<SQL-Update-Statement>

  Specify a valid SQL statement that can be applied to the database once the data (determined with db.processDBSQLStatement) has been successfully sent to the Integration Engine. This must be an INSERT, UPDATE, or DELETE statement. Examples of typical scenarios would be deleting the sent data from the database with a DELETE statement or selecting the data with an UPDATE statement. Instead of the SQL statement, you could also specify <TEST> (including the pointed parenthesis). In this case, once the data has been successfully sent, the system does not execute a database operation and the data remains unaltered in the database.

  The default value is <TEST>. This may be useful (except for test purposes) when, for example, the data has not only been read by a stored procedure defined in the statement db.processDBSQLStatement, but has already been changed.

  !

  This means that data is sent again as a new message if the adapter is restarted or if the time period specified in de.pollInterval is exceeded. This mode is exclusively of use when testing the configuration of the adapter or the Integration Engine, and is not for production operation.

  💡

  In Exactly Once mode, the statements specified under db.processDBSQLStatement and db.confirmDBSQLStatement, as well as the formatting of the message are all performed within one database transaction. The message is then sent afterwards. Until the message has been sent successfully to the Integration Server, it is temporarily persisted as a file in the Adapter Engine directory /Data.

- **db.execute=**<operating system command>

  If you specify an operation system command here, the system executes it following successful database operations (PROCESS and CONFIRM statements). The default value is an empty character string (no command).

- **db.pollInterval=**

  Specify the number of seconds that the adapter must wait before it calls the db.processDBSQLStatement again.
This specification is mandatory. You can also specify the following parameters so as to specify an additional waiting time in milliseconds:

```
db.pollIntervalMsecs=
```

If `db.pollInterval` is set to 0, then very short, almost real time processing times can be achieved.

The default value for this parameter is 0.

If the call is not to be repeated then you must set both parameters to 0 (or just the parameter `db.pollInterval` if you have not specified a value for parameter `db.pollIntervalMsecs`).

You can set the value to 0, if you do not want to repeat the call. In this case the adapter status remains as STOPPED. To initiate a new call, choose Stop/Start or Restart to restart the adapter.

- `db.retryInterval=`

Specify the number of seconds that the adapter is to wait before a SQL statement processed with errors is processed again.

The value from `db.pollInterval` is adopted as the default value. If this value is 0 (processing is not repeated), then SQL statements processed with errors are not processed again.

If `db.retryInterval` is also 0, then the adapter is terminated if an error occurs, even when a value greater than 0 was specified for `db.pollInterval`. In this case the adapter status remains as STARTED. To process the file again, choose Stop/Start or Restart to restart the adapter.

- `db.logPollInterval=NO|YES`

All entries to loops that last longer than 5 seconds are shown in the adapter log by default. As a result, this can lead to a lot of entries in the log. In such instances, the log output (but not the loop itself) can be deactivated by using `db.logPollInterval=NO`.

For loops that last 5 seconds or less, the output can be activated with `file.logPollInterval=YES`.

**Result**

The system converts the table resulting from the `db.processDBSQLStatement` into a valid XML document and sends it to the Integration Engine. The document looks as follows:

```
<resultset>
  <row>
    <column-name1>column-value</column-name1>
    <column-name2>column-value</column-name2>
    <column-name3>column-value</column-name3>
  </row>
  <row>
    <column-name1>column-value</column-name1>
    <column-name2>column-value</column-name2>
    <column-name3>column-value</column-name3>
  </row>
</resultset>
```

This document can then be processed further by the Integration Engine.
Configuring the Outbound JDBC Adapter

Use

You need to configure the outbound JDBC adapter so that you can convert XML messages from the Integration Engine [Seite 266] into the contents of database tables.

To configure the adapter you must specify the following:

- A JDBC database driver to access the database
  This driver is not supplied with the adapter but must be supplied by the database provider or third-parties.
- Logon data for the database to be written to
- The address (HTTP port and URL) where the adapter can be contacted from the Integration Engine.

Prerequisites

You have:

1. Installed a JDBC driver (Version 2.0) for the database.
2. Installed [Seite 363] the corresponding adapter.
4. Chosen Configure to call the configuration of the adapter module.

Procedure

The configuration of the outbound JDBC adapter comprises five functional sub areas:

1. Java class name for the outbound JDBC adapter
   Specify the class name as follows:
   \texttt{classname=com.sap.aii.messaging.adapter.ModuleXMB2DB}
   This specification is mandatory.

2. JDBC adapter mode
   Specify the mode of the outbound JDBC adapter. The following values are permitted:
   - \texttt{mode=XMB2DB}
   - \texttt{mode=XMB2DB\_XML}
   - \texttt{mode=XMB2DB\_RAWSQL}

   Depending on the mode you select, the adapter has different functions and expects the message received from the Integration Server to have a particular document format in each case. Note the following:
   - In mode \texttt{XMB2DB}, table values are inserted in a table that is specified in the adapter configuration. This mode has been kept for reasons of compatibility with older versions of the JDBC adapter, but should principally no longer be used.
   - In mode \texttt{XMB2DB\_XML}, table values can be inserted, updated, or deleted in one or more tables. Furthermore, stored procedures in the database can be called using transfer parameters. In the case of synchronous queries, results from database queries or return values of stored procedures can also be transferred.
This function is described by the transmission of predefined XML schema, which is explained in more detail below. The type and number of actions that can be bundled in a message is not restricted. All actions are then executed within a database transaction – either completely or not at all.

- In mode XMB2DB_RAWSQL, an SQL statement is expected that is transferred to the database for processing unchanged. This mode is primarily for test purposes only.

Subsequent versions may permit additional processing modes.

3. Specifications for addressing by the Integration Engine

- **XMB.httpPort=<port_no>**
  
  `<port_no>` specifies the HTTP server port that receives the messages from the Integration Engine.

- **XMB.httpService=<service>**
  
  `<service>` describes the service part of the address where the Integration Engine must send its messages.

These specifications are mandatory.

For example, if you have specified **XMB.httpPort=1234** and **XMB.httpService=/db/Receiver**, the end point address of the JDBC adapter in the Integration Engine must be specified as follows:

http://<Datenbankadapterhost>:1234/db/Receiver

The end point address must be extended as follows for the Integration Engine in Release 1.0:


If the Integration Engine message is sent to a non-specified adapter service, the system displays the following error message:

No registered listener for `<Service>` found

The system displays the same message if the adapter is initialized, but has not been started (status STOPPED or INITIALIZED).

- **XMB.ReceiverAdapterForBusinessSystem=<Business System>**
  
  This specification is optional.

  `<Business System>` is the logical name of a business system in the System Landscape Directory (SLD). If the Adapter Engine connection to the SLD is configured, an association for the outbound JDBC adapter is created for this business system. When you create an end point for this business system in the Integration Directory [Extern], the address of the adapter is determined from this association as the default value.

  If no connection can be established with the SLD, or if the business system does not exist in the SLD, the specification is meaningless.

4. Specifications for database access

These specifications are mandatory and have no default values.

- **db.jdbcDriver=<java_class_of_jdbcdriver>**
Specify the Java class of the JDBC driver that the JDBC adapter must load to be able to access the driver. The exact specification varies according to the JDBC driver; see material from the respective provider for exact details.

- `db.connectionURL=<jdbc_driver_connection_url>`

Specify the address with which you can establish a database connection using the JDBC driver. The exact format of the address can vary; see material from the respective provider for more details.

The JDBC adapter automatically reconnects to the database when a database connection error occurs.

- `db.table=<tablename>`

Specify the database table where the incoming data is to be written to. In the current version of the JDBC adapter it is not possible to distribute the contents of an incoming document to different database tables.

- `db.execute=<operating_system_command>`

An operating system command specified here is executed following successful database operations. The default value is an empty character string (no command).

5. Additional specifications to handle messages with Quality of Service = Exactly Once (In Order)

- `db.exactlyOnceErrorInPendingState=IGNORE|ERROR`

As in the other adapters, Exactly Once messages are handled by default using status information in the file system. All adapter error statutes and program terminations initialized externally (shutdown of the adapter operating system process) are also handled in this mode.

However, this does not include external program terminations during a database commit. In this case, the status of message processing is unclear since it can only be changed in the management file once the database commit has been completed.

However, this kind of situation is identified when the application is started and editing of the message terminated during processing can be controlled using `db.exactlyOnceErrorInPendingState`.

This specification is only effective for handling errors that occur when a message is being processed for a second time after initial processing remained in the unclear status described above.

If an error now occurs during this second processing attempt, this is returned by default to the sender system as an error (or if the value ERROR is set explicitly).

However, if the error occurs because the message was saved in the database when it was first processed and it is still located there, processing can be completed for the sender system using the value IGNORE (note that the database interface normally triggers the error duplicate insert if at least one of the table fields was defined as a primary key). Otherwise the sender system will continue to send the message and the error will continue to occur.

Nevertheless, it is possible that a message can become duplicated during Exactly Once handling. For example, if there is no primary key field in the database table or if the data is already being processed by another application and it is then deleted, when the first attempt at message processing is interrupted by an external termination of the adapter process immediately after the database commit, a message can be duplicated. This problem can only be
solved if message processing and status information management take place in
the same database so that the processing steps have the same commit cycle.

In the database where the write-to tables are located you must create an
additional table with two columns for this purpose. Column 1 must be type
character with length 36 (or more) and have the name XIMessage_ID. Column 2
must be of type integer and have the name XIMessage_Ts. This table is made
known to the adapter as follows:

- db.tableForExactlyOnceHandling=<table name>
  Set this value to use the specified database table instead of file-based Exactly-
  Once processing. If the table does not exist or the columns cannot be found or
  are of the wrong type, then a corresponding error is triggered and the adapter
  will not be started.

- db.tableEOColumnNameId=<column name>
  If you use a different column name to XIMessage_Id (for example, because you
  want to use a table that already exists), you can specify this value for the type
  char(36) column. If a greater field length is specified the table can nevertheless
  be used. However, the minimum field length is 36. This value is only evaluated if
  you set db.tableForExactlyOnceHandling.

- db.tableEOColumnNameTs=<column name>
  If you use a different column name to XIMessage_Is (for example, because you
  want to use a table that already exists), you can specify this value for the type
  integer) column. This value is only evaluated if you set
  db.tableForExactlyOnceHandling.

- db.exactlyOnceHandlingIDsExpiration=<n Days>
  Specify the number of days that the management information for received
  messages with the service attribute Exactly Once (In Order) should be kept. This
  information is required to avoid messages being duplicated, if when handling an
  error the Integration Engine sends the same message to an adapter more than
  once. The default value is 60 days. This value applies to both file-based and
database-based status management.

  If the value is set to 0 or to a negative value, all management information is
  deleted each time the adapter is initialized. This can be useful for test purposes
  but must not be used for production operation.

6. The required XML document format [Seite 409]

As already explained, the XML document format depends on the mode of the JDBC
adapter.

![Document Format for the Outbound JDBC Adapter](image)

Depending on the processing mode selected for the outbound JDBC adapter [Seite 406], the
adapter expects particular XML document formats in the message received from the
Integration Server.

The following processing types are possible:

- XMB2DB
- XMB2DB_XML
• XMB2DB_RAWSQL
Additional processing modes may be possible with future versions of the JDBC outbound adapter.

**XML Document Format for the Mode XMB2DB**

An additional parameter is required in this mode:

- **db.table=** `<tablename>`

Specify the database table where the incoming data is to be written to. In the current version of the JDBC adapter it is not possible to distribute the contents of an incoming document to different database tables.

You can also specify the following parameters:

  - **db.apostropheEsc=** `<apostropheEscapeString>`
    
    The apostrophe character (‘) is a reserved character in SQL syntax and is therefore replaced by an escape character if it occurs within value strings. This replacement character can be database-specific. Typical replacement characters are \’ or ′. If a character occurs that is invalid for the database being used, the adapter triggers an error message (an SQL exception) concerning the SQL syntax that is generated by the database.

  - **db.columnNameDelimiter=** `<DelimiterSign>`
    
    Depending on the database being used, column names can be enclosed by a special delimiter character, for example, if the names can contain special characters (such as a *). This character can be specified at this point. The default setting is no delimiter character. If a character occurs that is invalid for the database being used, the adapter triggers an error message (an SQL exception) concerning the SQL syntax that is generated by the database.

The incoming message must represent a table in XML format; its contents are written to the database table specified in `db.table`. The same document format is required as is generated by the inbound file/FTP and JDBC adapters. This means that the document looks as follows:

```xml
<resultset>
  <row>
    <column-name1>column-value</column-name1>
    <column-name2>column-value</column-name2>
    <column-name3>column-value</column-name3>
  </row>
  <row>
    <column-name1>column-value</column-name1>
    <column-name2>column-value</column-name2>
    <column-name3>column-value</column-name3>
  </row>
</resultset>
```

This is an example for 3 columns and 2 lines. The number of lines and columns is, of course, unlimited.

The column names of the database tables are taken from the element names in the XML document (for example, `column-name1`, `column-name2`, `column-name3`), and the corresponding values are inserted for each line.
The document does not have to contain all columns of a database table as long as the
database definition accepts a null value in place of the missing columns. Element
names of the XML document that do not exist as table column names are ignored.

**XML Document Format for the Mode XMB2DB_XML**

In this mode you can modify one or more database tables by means of a message.
Depending on the content of the message, you can either insert (INSERT), update (UPDATE)
or delete (DELETE) the data. Results from queries (SELECT) can also be included in the
response in XML format for synchronous messages. The XML document must have the
following schema in this case:

```xml
<root>
  <StatementName1>
    <dbTableName action="UPDATE" | "UPDATE_INSERT">
      <access>
        <col1>val1</col1>
        <col2>val2new</col2>
      </access>
      <key1>
        <col2>val2old</col2>
        <col4>val4</col4>
      </key1>
      <key2>
        <col2>val2old2</col2>
      </key2>
    </dbTableName>
  </StatementName1>

  <StatementName2>
    <dbTableName action="INSERT">
      <access>
        <col1>val1</col1>
        <col2>val2</col2>
      </access>
      <access>
        <col1>val11</col1>
      </access>
    </dbTableName>
  </StatementName2>

  <StatementName3>
    <dbTableName action="DELETE">
      <key1>
      </key1>
    </dbTableName>
  </StatementName3>
</root>
```
<col2>val2old</col2>
<col4>val4</col4>
</key1>
<key2>
<col2>val2old2</col2>
</key2>
</dbTableName>
</StatementName3>

<StatementName4>
<dbTableName action="SELECT">
<access>
<col1/>
<col2/>
<col3/>
</access>
<key1>
<col2>val2old</col2>
<col4>val4</col4>
</key1>
<key2>
<col2>val2old2</col2>
</key2>
</dbTableName>
</StatementName4>

<StatementName5>
<storedProcedureName action="EXECUTE">
<param1 [type=NUMERIC | "STRING"]>val1</param1>
<param2 isOutput="true" type=SQL-Datatype></param2>
</storedProcedureName>
</StatementName5>
</root>

7. Comments:

- The document contains a tag with the arbitrary name <root>. Within this tag there are one or more statement elements that also have arbitrary names. Each of these statements contains the description of a database action. With the exception of the execute description for a stored procedure (shown in the example under the element <StatementName5>), all statements have the same structure:
The name of the element beneath the statement element specifies the name of the database table and contains the attribute action with the values INSERT, UPDATE, UPDATE_INSERT, DELETE or SELECT.

Within this element there is (except for DELETE) an element with the name access and one or more elements with the name key. Both have arbitrary names. access contains the table columns that are to be accessed and must be specified as the first element. key describes a condition for access. If no such elements are specified, access proceeds without any conditions. In the case of UPDATE and DELETE, this can lead to the entire table being updated and deleted respectively. Such consequences can be avoided by setting the following parameter:

db.conditionRequired=YES

The response document described below can only be evaluated by the Integration Server if the call is synchronous because the content of the response document is not accessible if the call is asynchronous. The response is put in a separate element <StatementName_response> for each statement element.

- **action=UPDATE**

  Statements with this action cause existing table values to be updated. Therefore, the statement corresponds to an SQL UPDATE statement.

  The block <access> contains the new column values and a <key> element contains the columns whose values must be identical with the specified value to get the new column values. The name of the <key> element is arbitrary. Column values within a <key> element are combined with a logical AND; different <key> elements are combined with a logical OR.

  A statement with the action UPDATE must have exactly one <access> element. The number of <key> elements with arbitrary names is not restricted.

  The corresponding SQL statement for StatementName1 in the example above is as follows:

  ```
  UPDATE dbTableName  SET col1='val1', col2='val2new' WHERE ((col2='val2old' AND col4='val4') OR (col2='val2old2'))
  ```

  As in the remaining examples, the column type String is used for all columns. The character " may be missing in other column types.

  The response document contains the following element as well as the number of updated table lines, including 0.

  `<update_count>count</update_count>`

  If there is no <key> element, or if there is a <key> element but it is empty, then no condition is specified and the entire table is to be updated. This may not be permitted by the configuration of the JDBC adapter for security reasons and will therefore result in an error during message processing and an appropriate error message.

- **action=INSERT**

  Statements with this action cause existing table values to be inserted. Therefore, the statement corresponds to an SQL INSERT statement.

  The block <access> contains the new column values.

  A statement with the action INSERT must have exactly one <access> element. It cannot have a <key> element.
The corresponding SQL statement for StatementName2 in the example above is as follows:

```
" INSERT INTO dbTableName  (col1, col2) VALUES('val1', 'val2')
INSERT INTO dbTableName  (col1) VALUES('val11') "
```

The response document contains the following element as well as the number of inserted table lines, including 0.

<insert_count>count</insert_count>

- action=UPDATE_INSERT

The statement has the same format as for the UPDATE action. Initially, the same action is executed as for UPDATE. If no update to the database table can be made for this action (the condition does not apply to any table entry), values of the table described in the <access> element are inserted in accordance with the description of the action INSERT. <key> elements are ignored in this case.

The response document has the following format; one of the two values is 0 because either an UPDATE or an INSERT action is always executed:

<update_count>count</update_count>

<insert_count>count</insert_count>

- action=DELETE

Statements with this action cause existing table values to be deleted. One or more <key> elements formulate the condition that deletes the table values. The names of <key> elements are arbitrary. Column values within a <key> element are combined with a logical AND; different <key> elements are combined with a logical OR.

The corresponding SQL statement for StatementName3 in the example above is as follows:

```
" DELETE FROM dbTableName  WHERE ((col2='val2old' AND col4='val4') OR (col2='val2old2')) "
```

The response document contains the following element:

<delete_count>count</delete_count>

If there is no <key> element, or if there is a <key> element but it is empty, then no condition is specified and the entire table is to be deleted. This may not be permitted by the configuration of the JDBC adapter for security reasons and will therefore result in an error during message processing and an appropriate error message.

- action=SELECT

Statements with this action cause existing table values to be selected. Therefore, the statement corresponds to an SQL SELECT statement.

The block <access> contains the column names to be selected, a <key> element contains the columns whose values must be identical with the specified value to get the new column values. The name of the <key> element is arbitrary. Column values within a <key> element are combined with a logical AND; different <key> elements are combined with a logical OR.

A statement with the action SELECT must have exactly one <access> element. The number of <key> elements with arbitrary names is not restricted.
The corresponding SQL statement for `StatementName4` in the example above is as follows:
```
SELECT col1, col2, col3 FROM dbTableName WHERE ((col2='val2old' AND col4='val4') OR (col2='val2old2'))
```

If there is no `<key>` element, or if there is a `<key>` element but it is empty, then no condition is specified and the entire table is to be selected. This may not be permitted by the configuration of the JDBC adapter for security reasons and will therefore result in an error during message processing and an appropriate error message.

The response document contains the result of the action in XML format as follows:
```
  <row>
    <column1>value11</column1>
    <column2>value12</column2>
    ...
  </row>
  ...
  <row>
    <column1>valueN1</column1>
    <column2>valueN2</column2>
    ...
  </row>
  ...
```

• `action=EXECUTE`

Statements with this action result in a stored procedure being executed. The name of the element is interpreted as the name of the stored procedure in the database. The elements within the stored procedure are interpreted as parameters.

Output parameters of a stored procedure must have the attribute `isOutput=true` and an attribute `type=<SQL-Datatype>` that describes the valid SQL data type (for example, `type="INTEGER"` or `type="CHAR(30)"`).

Input parameters can have the attribute `type` with the values `NUMERIC` or `STRING`. Numerical values are automatically interpreted as numerical data types. All remaining values are interpreted as text data types.

There are two ways that the parameters can be transferred:

- The default setting is to use the sequence of the parameters.
- If the following parameter is set then the element names are used as parameter names instead:

```
db.useParamNames=true
```

All return values are returned in an XML structure. The results within the stored procedure are returned either as a table or as the element `<update_count>`. This depends on the SQL statements executed within the stored procedure. The return parameters of a stored procedure are attached in a separate structure.

Additionally, the following parameters can be specified for the SQL syntax that are used for all actions:
The apostrophe character ('') is a reserved character in SQL syntax and is therefore replaced by an escape character if it occurs within value strings. This replacement character may be database-specific. Typical characters are '\' or '". The default character is '".

If a character occurs that is invalid for the database being used, the adapter triggers an error message (an SQL exception) concerning the SQL syntax that is generated by the database.

XML Document Format for the Mode XMB2DB_RAWSQL

This mode is primarily for test purposes only. Instead of an XML document format, a text is expected that represents any valid SQL statement.

When inserting a line into a table the corresponding document looks as follows:

```
"INSERT INTO tableName (column-name1, column-name2, column-name3)
VALUES('column-value1', 'column-value2', 'column-value3')
"
```

![Configuring the Inbound JMS Adapter](image)

**Use**

You need to configure the inbound JMS adapter to send JMS messages to the Integration Engine [Seite 266].

To configure the adapter you must specify the following:

- The corresponding Java class
- General adapter settings
- The message type to be received
- The transformation class and the bindings implemented in the transformation class
- A dispatcher class (optional) with the corresponding settings and the user exits and their settings that are to be called by the dispatcher
- How to address the Integration Engine
- Settings specific for the respective JMS provider [Seite 430]

**Prerequisites**

You have:
1. Installed [Seite 363] the corresponding adapter.
2. Selected [Seite 367] the adapter on the configuration screen.
3. Called the adapter configuration by choosing *Configure*.

**Procedure**

The configuration of the inbound JMS adapter comprises six functional sub areas:

1. Java class name for the inbound JMS adapter

   Specify the class name as follows:
   ```
   classname=com.sap.aii.messaging.adapter.ModuleJMS2Transport
   ```

   This specification is mandatory.

2. General adapter settings

   You can set the following general parameters for the inbound JMS adapter:
   - `JMSSession.sessionTransacted=true` or `false`
     
     The default value is `true`.
     
     This parameter determines whether a transactional JMS session should be used or not. Following processing of a message, a transactional session ends either with a COMMIT, or in the case of an error, with a ROLLBACK.
   - `errorDelay`
     
     The default value for this parameter is `0`.
     
     This parameter determines how long the system waits before processing further messages if an error occurs. In transactional processing, it can make sense to specify a certain time period, for example, because messages with errors remain in the queue following a ROLLBACK.
   - `reconnectDelay`
     
     The default value for this parameter is `-1`.
     
     This parameter controls whether and after what time period an attempt is made to reestablish the connection following a connection error on the JMS side.
     
     The time period must be specified in milliseconds. No attempt is made if the value is set to `-1`.

     Your JMS provider must support the registration of an ExceptionListener for this.

3. The message type to be received or sent

   On the JMS side, the JMS adapter knows the following messages:
   - `JMSMessage`
   - `JMSTextMessage`
   - `JMSBytesMessage`
   - `JMSObjectMessage`
   - `JMSSStreamMessage`
   - `JMSMapMessage`

   See the corresponding [documentation on JMS](#) for more information.

   On the HTTP side, the JMS adapter knows the following message types:
   - `TransportMessage`
This is essentially a (single or multipart) MIME message with a connection to the transport protocol (HTTP). It transports an XMBMessage.

- **XMBMessage**
  This is an SAP-defined SOAP message with attachments that is logically based on a multipart-MIME message.

The architecture of the JMS adapter is designed so that every JMS message can be mapped to a HTTP message.

In the configuration, specify which message type you want to send:

- `TransportMessage.type=TransportMessage` or `XMBMessage`
  The default value is `XMBMessage`.

The following parameter is set at runtime:

- `JMSMessage.type`

4. The transformation class in which the bindings for the conversion of different message formats are implemented, and the definition of the particular bindings for the message formats.

In the configuration, specify the transformation class to be used under `transformClass`. The following classes are supplied:

- `com.sap.aii.messaging.adapter.trans.JMSTunneling`
- `com.sap.aii.messaging.adapter.trans.JMSSinglePayloadBinding`

You can define your own bindings at any point and apply them to the JMS adapter by implementing a class that enhances the class `com.sap.aii.messaging.adapter.trans.Transform` and entering this class as the transformation class. Use the classes `JMSTunneling` or `JMSSinglePayloadBinding` as templates; their functions are explained in the following step:

In the class `com.sap.aii.messaging.adapter.trans.JMSTunneling`, a binding is implemented for each of the following cases:

- **JMSTextMessage** → **TransportMessage**
- **JMSBytesMessage** → **TransportMessage**

As you can see from the class name, each binding is a kind of JMS tunneling. This means that the transport message is transported to the JMS payload.

In the class `com.sap.aii.messaging.adapter.trans.JMSSinglePayloadBinding`, a binding is implemented for each of the following cases:

- **JMSTextMessage** → **TransportMessage**
  The payload of a JMSTextMessage is used as the payload of a TransportMessage. The HTTP content type is determined from the value of the parameter `XMB.ContentType` (default value: `text/plain`).

- **JMSBytesMessage** → **TransportMessage**
  The payload of a JMSBytesMessage is used as the payload of a TransportMessage. The HTTP content type is determined from the value of the parameter `XMB.ContentType` (default value: `application/octet-stream`).

- **JMSTextMessage** → **XMBMessage**
  The payload of a JMSTextMessage is used as the payload of a XMBMessage. The header data of the XMBMessage is taken from the configuration, for example:

  ```
  XMB.SenderBusinessSystem=ExtAdapterSender
  ```
XMB.SenderInterfaceNamespace=http://sap.com/xi/xidemo
XMB.SenderInterfaceName=ExtAdapterSenderIF

This address argument for the sender system is mandatory. They are used to identify the adapter configuration in routing [Seite 214] and in mapping [Seite 143] in the Integration Engine pipeline [Seite 267]. You can also find an explanation about each argument there.

You can also specify the receiver system by using the following parameter.
XMB.ReceiverBusinessSystem=ExtAdapterReceiver

In this case you do not need to determine the receiver in routing.

- JMSBytesMessage → XMBMessage
  The payload of a JMSBytesMessage is used as the payload of a XMBMessage. The header data of the XMBMessage is taken from the configuration.

While JMS is used by the bindings in the class JMSTunneling as a kind of means of transport, bindings in the class JMSSinglePayloadBinding are much more specific. The message types specified in step 3 control which binding is used at runtime.

5. The dispatcher class and the user exits to be called by the dispatcher (optional)

The inbound JMS adapter provides you with a dispatcher that you can use to convert messages before they are sent. The settings required for the dispatcher are explained in the example [Seite 435].

6. How to address the Integration Engine

Enter the complete address (URL) of the Integration Engine that you want to send the message to:
XMB.TargetURL=http://IntegrationEngineHost:port/pipeline-arguments

This specification is mandatory.

💡

The Integration Engine address can also be accessed dynamically from the SAP System Landscape Directory. The following entry is added to the configuration for this purpose:
XMB.SLDConfiguration=SLDaccessor

In this case, for the system specified with XMB.SenderBusinessSystem, the URL of the corresponding Integration Server in the SLD is read and is used instead of the value specified under XMB.TargetURL. Therefore, specify the following:
XMB.TargetURL=<fromSLD>

To be able to access the System Landscape Directory, the SLDaccessor service [Seite 376] service must be configured correspondingly and the respective entries must be maintained in the System Landscape Directory.

💡

If the Integration Server is accessed by means of an HTTP proxy server rather than directly, the following parameters must be set:
XMB.proxyHost=<proxyHostname>
XMB.proxyPort=<proxyPortnumber>

<proxyHostname> is the host name of the proxy and <proxyPortnumber> is the port where the proxy receives HTTP requests (for example, 8080).
If an authentication was specified for the specified URL (HTTP service) in the Integration Engine, the following specifications are mandatory:

- XMB.User=<user-name>
- XMB.Password=<password>

The specifications must match those that you made in transaction SICF in the Integration Engine. If you do not specify a user and password combination, or the combination is invalid, the system will terminate each attempt to access the Integration Engine with the message **Transport Exception: http-Error 401 – Unauthorized.**

The user must have the authorizations of group SAP_XI_APPL_SERV_USER on the Integration Server.

If you want to use a different client or language to the default setting in the Integration Engine when logging on, you can also set the following parameters:

- XMB.Client=<client-no>
- XMB.Language=<language-id>

### 7. Provider-specific settings

For more information about provider-specific settings, see [Settings for the JMS Provider](Seite 430).

**Example**

Below is an example configuration for the inbound JMS adapter:

```
# JMS Adapter Configuration
#
# This configuration can be used as a template to configure the
# JMS inbound adapter
#
#*****************************************************************************
# uncomment for receiving JMS messages
classname=com.sap.aii.messaging.adapter.ModuleJMS2Transport
# uncomment for sending JMS messages
#*****************************************************************************
#
# set if a transacted JMS session should be used (default: true)
#JMSSession.sessionTransacted=false
#
# set delay in milliseconds before processing next message after an error
# (default: 0)
```
#errorDelay=10000
#
# set delay in milliseconds before trying to reestablish the JMS connection
# after a connection error (default: -1 (do not try to reconnect))
# reconnectDelay=10000
#
#******************************************************************************
# set the message formats you want to process
# formats are only needed if they cannot be evaluated at runtime
#
# TransportMessage Type can be set to TransportMessage or XMBMessage
# it is used for sending and receiving TransportMessages and XMBMessages
#TransportMessage.type=TransportMessage
TransportMessage.type=XMBMessage
#******************************************************************************
#
# uncomment and adjust parameters for Transformation Type
# (binding of JMS TransportMessage); insert your class here for an
# individual binding between JMS and TransportMessage
#
# uncomment the following line for using bindings that belong to Tunneling
#transformClass=com.sap.aii.messaging.adapter.trans.JMSTunneling
# uncomment the following line for using bindings that belong to
# SinglePayloadBinding
transformClass=com.sap.aii.messaging.adapter.trans.JMSSinglePayloadBi
nding
#
# the following parameters are used in bindings that belong to
# SinglePayloadBinding and require XMBHeader information
#
# uncomment and adjust parameters according to your requirements
XMB.SenderBusinessSystem=ExtAdapterSender
XMB.SenderInterfaceNamespace=http://sap.com/xi/xidemo
XMB.SenderInterfaceName=ExtAdapterSenderIF
XMB.ReceiverBusinessSystem=ExtAdapterReceiver
XMB.TraceLevel=1
XMB.LoggingLevel=1
XMB.QualityOfService=EO
XMB.QueueId=ABCDEFABCDEF
XMB.ContentType=text/xml
#
# uncomment and adjust parameters for XMB.Sender
XMB.TargetURL=http://<host>:<port>/sap/xi/engine?type=entry
#
# uncomment and adjust parameters for loading JMS administrated objects
#JMS.FileNameQueueConnectionFactory=c:/temp/QueueConnectionFactory.sjob
#JMS.FileNameQueue=c:/temp/Queue.sjo
#
# uncomment and adjust parameters for creating SonicMQ
# JMS administrated objects
JMS.QueueConnectionFactoryImpl.classname=progress.message.jclient.QueueConnectionFactory
JMS.QueueConnectionFactoryImpl.constructor=java.lang.String p49512:2506
JMS.QueueImpl.classname=progress.message.jclient.Queue
JMS.QueueImpl.constructor=java.lang.String SampleQ1
#
# uncomment and adjust parameters for creating WebSphereMQ
# JMS administrated objects
#JMS.QueueConnectionFactoryImpl.classname=com.ibm.mq.jms.MQQueueConnectionFactory
#JMS.QueueConnectionFactoryImpl.method.setHostName=java.lang.String p24537
#JMS.QueueConnectionFactoryImpl.method.setChannel=java.lang.String JAVA.CHANNEL
#JMS.QueueConnectionFactoryImpl.method.setTransportType=java.lang.Integer {com.ibm.mq.jms.JMSC.MQJMS_TP_CLIENT_MQ_TCPIP}
#JMS.QueueConnectionFactoryImpl.method.setQueueManager=java.lang.String QM_p24537
#JMS.QueueImpl.classname= com.ibm.mq.jms.MQQueue
#JMS.QueueImpl.constructor=java.lang.String postcard
#JMS.QueueImpl.method.setTargetClient=java.lang.Integer
{com.ibm.mq.jms.JMSC.MQJMS_CLIENT_NONJMS_MQ}

#********************************************************************
******
#
# uncomment and adjust the parameters for using a dispatcher
#Dispatcher.class=com.sap.aii.messaging.adapter.ConversionDispatcher
#Dispatcher.namespace=namespace1
#namespace1.ConversionDispatcher.logPayload=true
#namespace1.Service.1=Plain2XMLService
#namespace1.Plain2XMLService.class=com.sap.aii.messaging.adapter.Conversion
#namespace1.Plain2XMLService.xml.conversionType=SimplePlain2XML
#namespace1.Plain2XMLService.xml.processFieldNames=fromConfiguration
#namespace1.Plain2XMLService.xml.fieldNames=a,b,c
#namespace1.Plain2XMLService.xml.fieldSeparator=;
#namespace1.Service.2=XSLTService
#namespace1.XSLTService.class=com.sap.aii.messaging.adapter.XSLTConversion
#namespace1.XSLTService.XSLTConversion.XSLTFileName=Data/DemoConversion.xsl

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++

### Configuring the Outbound JMS Adapter

**Use**

You need to configure the outbound JMS adapter to convert XML messages from the Integration Engine [Seite 266] into JMS messages.

To configure the adapter you must specify the following:

- The corresponding Java class
- General adapter settings
- The message type to be received
- The transformation class and the bindings implemented in the transformation class
- A dispatcher class (optional) with the corresponding settings and the user exits and their settings that are to be called by the dispatcher
- How to address the Integration Engine
- Settings specific for the respective JMS provider [Seite 430]
Prerequisites
You have:
1. Installed [Seite 363] the corresponding adapter.
2. Selected [Seite 367] the adapter on the configuration screen.
3. Chosen Configure to call the configuration of the adapter module.

Procedure
The configuration of the outbound JMS adapter comprises six functional sub areas:

1. Java class name for the outbound JMS adapter
   Specify the class name as follows:
   \[ \text{classname=\texttt{com.sap.aii.messaging.adapter.ModuleTransport2JMS}} \]
   This specification is mandatory.
2. General adapter settings
   You can set the following general parameters for the outbound JMS adapter:
   - JMS\texttt{Session.sessionTransacted=\texttt{true or false}}
     The default value is \texttt{true}.
     This parameter determines whether a transactional JMS session should be used or not. Following processing of a message, a transactional session ends either with a COMMIT, or in the case of an error, with a ROLLBACK.
   - JMS\texttt{.MessageProducer.DeliveryMode=\texttt{PERSISTENT or NON_PERSISTENT}}
     The default value is \texttt{PERSISTENT}.
     This parameter controls the delivery mode of the JMS messages to be sent.
   - errorDelay
     The default value for this parameter is 0.
     This parameter determines how long the system waits before processing further messages if an error occurs. In transactional processing, it can make sense to specify a certain time period, for example, because messages with errors remain in the queue following a ROLLBACK.
   - reconnectDelay
     The default value for this parameter is -1.
     This parameter controls whether and after what time period an attempt is made to reestablish the connection following a connection error on the JMS side. The time period must be specified in milliseconds. No attempt is made if the value is set to -1.
     ! Your JMS provider must support the registration of an ExceptionListener for this.
3. The message type to be received or sent
   On the HTTP side, the JMS adapter knows the following message types:
   - TransportMessage
     This is essentially a (single or multipart) MIME message with a connection to the transport protocol (HTTP). It transports an XMBMessage.
   - XMBMessage
This is an SAP-defined SOAP message with attachments that is logically based on a multipart-MIME message.

On the JMS side, the JMS adapter knows the following messages:
- JMSMessage
- JMSTextMessage
- JMSBytesMessage
- JMSObjectMessage
- JMSStreamMessage
- JMSMapMessage

See the corresponding documentation on JMS for more information.

The architecture of the JMS adapter is designed so that every HTTP message can be mapped to a JMS message.

In the configuration, specify which message type is to be received:
- TransportMessage.type= \texttt{TransportMessage} or \texttt{XMBMessage}
  The default value is \texttt{XMBMessage}.

Also specify the JMS message type to be sent:
- JMSMessage.type= \texttt{TextMessage} or \texttt{BytesMessage}
  The default value is \texttt{JMSTextMessage}.

4. The transformation class in which the bindings for the conversion of different message formats are implemented, and the definition of the particular bindings for the message formats.

In the configuration, specify the transformation class to be used under \texttt{transformClass}. The following classes are supplied:
- \texttt{com.sap.aii.messaging.adapter.trans.JMSTunneling}
- \texttt{com.sap.aii.messaging.adapter.trans.JMSSinglePayloadBinding}

You can define your own binding at any point and apply it to the JMS adapter by implementing a class that enhances the class \texttt{com.sap.aii.messaging.adapter.trans.Transform} and entering this class as the transformation class. Use the classes JMSTunneling or JMSSinglePayloadBinding as templates; their functions are explained briefly below:

In the class \texttt{com.sap.aii.messaging.adapter.trans.JMSTunneling}, a binding is implemented for each of the following cases:
- TransportMessage \rightarrow JMSTextMessage
- TransportMessage \rightarrow JMSBytesMessage
- XMBMessage \rightarrow JMSTextMessage
- XMBMessage \rightarrow JMSBytesMessage

As you can see from the class name, each binding is a kind of JMS tunneling. This means that the TransportMessage or XMBMessage is transported in the JMS payload.

You do not need to set any further configuration parameters. The message types specified in step 3 control which binding is used at runtime.

In the class \texttt{com.sap.aii.messaging.adapter.trans.JMSSinglePayloadBinding}, a binding is implemented for each of the following cases:
- TransportMessage \rightarrow JMSTextMessage
The payload of a TransportMessage is used as the payload of a JMSTextMessage. The value in the HTTP header field is used to determine whether the payload of the TransportMessage is text-based:

\texttt{content-Type = text/...}

- **TransportMessage** → **JMSBytesMessage**
  The payload of a TransportMessage is used as the payload of a JMSBytesMessage. The value in the HTTP header field is used to determine whether the payload of the TransportMessage is not text-based:

\texttt{content-Type \neq text/...}

- **XMBMessage** → **JMSTextMessage**
  The payload of a XMBMessage is used as the payload of a JMSTextMessage. The payload type is used to determine whether the payload of the XMBMessage is text-based: The header data of the XMBMessage is ignored.

- **XMBMessage** → **JMSBytesMessage**
  The payload of a XMBMessage is used as the payload of a JMSBytesMessage. The payload type is used to determine whether the payload of the XMBMessage is not text-based: The header data of the XMBMessage is ignored.

While JMS is used by the bindings in the class JMSTunneling as a kind of means of transport, bindings in the class JMSSinglePayloadBinding are much more specific.

5. The dispatcher class and the user exits to be called by the dispatcher (optional)

The outbound JMS adapter provides you with a dispatcher that you can use to convert messages before they are sent. The settings required for the dispatcher are explained in the example [Seite 435].

6. Specifications for Integration Engine addressing

- **XMB.httpPort=\texttt{<port_no>}**
  \texttt{<port_no>} specifies the HTTP server port that receives the messages from the Integration Engine.

- **XMB.httpService=\texttt{<service>}**
  \texttt{<service>} describes the service part of the address where the Integration Engine must send its messages.

These specifications are mandatory.

For example, if you have specified \texttt{XMB.httpPort=1234} and \texttt{XMB.httpService=/JMS}, the end point address of the JMS adapter in the Integration Engine must be specified as follows:

\texttt{http://<JMSAdapterhost>:1234/JMS}

The end point address must be extended as follows for the Integration Engine in Release 1.0:

\texttt{http://<JMSAdapterhost>:1234/JMS?action=execute&pipelineid=Receiver}

If the Integration Engine message is sent to a non-specified adapter service, the system displays the following error message:

\texttt{No registered listener for <Service> found}
The system displays the same message if the adapter is initialized, but has not been started (status STOPPED or INITIALIZED).

7. Provider-specific settings
   For more information about provider-specific settings, see Settings for the JMS Provider [Seite 430].

Example
Below is an example configuration for the outbound JMS adapter:

```
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++
# JMS Adapter Configuration
#
# This configuration can be used as a template to configure the
# JMS outbound adapter
#
#******************************************************************************
# uncomment for sending JMS messages
classname=com.sap.aii.messaging.adapter.ModuleTransport2JMS
#******************************************************************************
#
# set if a transacted JMS Session should be used (default: true)
#JMSSession.sessionTransacted=false
#
# set the delivery mode of the JMS MessageProducer (QueueSender, default: PERSISTENT)
#JMS.MessageProducer.DeliveryMode=NON_PERSISTENT
#
# set delay in milliseconds before processing next message after an error
# (default: 0)
#errorDelay=10000
#
# set delay in milliseconds before trying to reestablish the JMS connection
# after a connection error (default: -1 (do not try to reconnect))
#reconnectDelay=10000
#
#******************************************************************************
#
# set the message formats you want to process
```
# formats are only needed if they cannot be evaluated at runtime
#
# JMSMessage Type can be set to TextMessage or BytesMessage
# it is only used for sending JMS messages
#JMSMessage.type=BytesMessage
JMSMessage.type=TextMessage
#
# TransportMessage Type can be set to TransportMessage or XMBMessage
# it is used for sending and receiving TransportMessages and
XMBMessages
#TransportMessage.type=TransportMessage
TransportMessage.type=XMBMessage
#
# uncomment and adjust parameters for Transformation Type
# (binding of JMS TransportMessage); insert your class here for an
# individual binding between JMS and TransportMessage
#
# uncomment the following line for using bindings that belong to
Tunneling
#transformClass=com.sap.aii.messaging.adapter.trans.JMSTunneling
# uncomment the following line for using bindings that belong to
# SinglePayloadBinding
transformClass=com.sap.aii.messaging.adapter.trans.JMSSinglePayloadBi
nding
#
# uncomment and adjust parameters for XMB.Receiver
XMB.httpPort=3333
XMB.httpService=/JMS
#
# uncomment and adjust parameters for loading JMS administrated
objects
#JMS.FileNameQueueConnectionFactory=c:/temp/QueueConnectionFactory.sj
#JMS.FileNameQueue=c:/temp/Queue.sjo
# uncomment and adjust parameters for creating SonicMQ
# JMS administrated objects
JMS.QueueConnectionFactoryImpl.classname=progress.message.jclient.QueueConnectionFactory
JMS.QueueConnectionFactoryImpl.constructor=java.lang.String p49512:2506
JMS.QueueImpl.classname= progress.message.jclient.Queue
JMS.QueueImpl.constructor=java.lang.String SampleQ1

# uncomment and adjust parameters for creating WebSphereMQ
# JMS administrated objects
#JMS.QueueConnectionFactoryImpl.classname=com.ibm.mq.jms.MQQueueConnectionFactory
#JMS.QueueConnectionFactoryImpl.method.setHostName=java.lang.String p24537
#JMS.QueueConnectionFactoryImpl.method.setChannel=java.lang.String JAVA.CHANNEL
#JMS.QueueConnectionFactoryImpl.method.setTransportType=java.lang.Integer {com.ibm.mq.jms.JMSC.MQJMS_TP_CLIENT_MQ_TCP}
#JMS.QueueConnectionFactoryImpl.method.setQueueManager=java.lang.String QM_p24537
#JMS.QueueImpl.classname= com.ibm.mq.jms.MQQueue
#JMS.QueueImpl.constructor=java.lang.String postcard
#JMS.QueueImpl.method.setTargetClient=java.lang.Integer {com.ibm.mq.jms.JMSC.MQJMS_CLIENT_NONJMS_MQ}

# uncomment and adjust parameters for using a dispatcher ***
#Dispatcher.class=com.sap.aii.messaging.adapter.ConversionDispatcher
#Dispatcher.namespace=namespace1
#namespace1.ConversionDispatcher.logPayload=true
#namespace1.Service.1=Plain2XMLService
#namespace1.Plain2XMLService.class=com.sap.aii.messaging.adapter.Conversion
#namespace1.Plain2XMLService.xml.conversionType=SimplePlain2XML
#namespace1.Plain2XMLService.xml.processFieldNames=fromConfiguration
#namespace1.Plain2XMLService.xml.fieldNames=a,b,c
#namespace1.Plain2XMLService.xml.fieldSeparator;
Settings for the JMS Provider

In JMS, provider-specific information is contained in administrated objects. QueueConnectionFactory and Queue are the objects relevant for the JMS adapter. Administrated objects such as these are either loaded dynamically (for example, from the local file system or from LDAP servers by using JNDI), or generated directly.

Loading Objects Dynamically

Existing administrable objects can be loaded from the local file system. The following two parameters can be used for this purpose:

- JMS.FileNameQueueConnectionFactory=<path/filename>
- JMS.FileNameQueue=<path/filename>

Generating Objects Directly

Since on the one hand, administrable objects do not always already exist, and on the other, no provider-specific characteristics should exist in the adapter (JMS interfaces are provider-independent, however, administrable objects are extremely provider-dependent), there is a additional, simple way to specify provider-specific data in the configuration of the JMS adapter. This makes the overall configuration easier and more flexible.

Below is a brief example of the general administration of QueueConnectionFactory and Queue using the example of SonicMQ and WebSphereMQ.

You select the JMS provider by specifying the implementing class of QueueConnectionFactory, for example:

JMS.QueueConnectionFactoryImpl.classname=progress.message.jclient.QueueConnectionFactory (for SonicMQ)
JMS.QueueConnectionFactoryImpl.classname=com.ibm.mq.jms.MQQueueConnectionFactory (for WebSphereMQ)

The provider determines which messaging parameter is set, and how. For example, the provider address is set in the constructor of the QueueConnectionFactory for SonicMQ, while this is performed using a SetMethod for WebSphereMQ:

JMS.QueueConnectionFactoryImpl.constructor=java.lang.String p49512:2506 (for SonicMQ)
JMS.QueueConnectionFactoryImpl.method.setHostName=java.lang.String p49512:2506 (for WebSphereMQ)
In addition, you need to set a number of provider-dependent parameters or flags. For example, for WebSphereMQ the information whether the provider should be contacted using JNI or TCP/IC (the former is the default setting, but is only possible on local machines). You set the connection type TCPIP using a constant, for example:

```java
JMS.QueueConnectionFactoryImpl.method.setTransportType=java.lang.Integer{com.ibm.mq.jms.JMSC.MQJMS_TP_CLIENT_MQ_TCPIP}
```

For Queues the same applies as for QueueConnectionFactory. For example, if you want to send a JMS message using WebSphereMQ, but the receiver only supports native WebSphereMQ messages, then the following configuration entry is recommended:

```java
JMS.QueueImpl.method.setTargetClient=java.lang.Integer{com.ibm.mq.jms.JMSC.MQJMS_CLIENT_NONJMS_MQ}
```

See the documentation of your JMS provider for more information about mandatory/optional settings.

---

### Configuring the SOAP Adapter

#### Use

You need to configure the SOAP adapter so that you can exchange SOAP messages between the Integration Engine [Seite 266] and remote clients or servers of Web Services.

The SOAP adapter provides a runtime environment that includes various SOAP components for the processing of SOAP messages. You can combine these SOAP components with separate components to satisfy your needs and requirements.

The SOAP adapter uses a helper class to instantiate and control SOAP components. If you want to use your own SOAP processing logic you must make your helper class known to the SOAP adapter.

To configure the SOAP adapter you must specify the following:

- The helper class that implements the following interface:
  ```java
  com.sap.aii.messaging.adapter.ModuleBubbleHelper
  ```

- The parameter values for the specified helper class

For example, you must specify the following parameters for the helper class ModuleBubbleHelperXMBWSImpl (instantiates a BubbleBag to integrate remote Web Services with the Integration Engine):

- Information about the Integration Engine destination (when the Integration Engine is acting as a service provider and therefore the SOAP adapter must be configured as an inbound adapter).
- Information about the Web Service provider destination (when the Integration Engine is acting as a service client and therefore the SOAP adapter must be configured as an outbound adapter).
- Various options for controlling the conversion of multi-part Integration Engine SOAP messages and Web Service SOAP messages.
**Prerequisites**

You have:

1. Installed [Seite 363](#) the corresponding adapter.
2. Selected [Seite 367](#) the adapter on the configuration screen.
3. Called the adapter configuration by choosing Configure.

**Procedure**

The configuration of the SOAP adapter comprises two functional sub areas:

1. The Java class name for the SOAP adapter
   Specify the class name as follows:
   
   ```
   classname=com.sap.aii.messaging.adapter.ModuleBubble
   ```

   This specification is mandatory.

2. The configuration parameters for the specified helper class
   Therefore, you must specify the following parameters for the shipped helper class ModuleBubbleHelperXMBWSImpl, for example:
   
   - Specifications for the Integration Engine as a service provider for queries from Web Service clients (inbound SOAP adapter).
   - Specifications for the Web Service provider for queries from the Integration Engine as a Web Service client (SOAP outbound adapter).

   You can configure an inbound adapter, or an outbound adapter, or both, if required.

**Configuring the Helper Class for Inbound Processing**

1. Enter the complete address (URL) of the Integration Engine that you want to send the message to:
   
   ```
   XMB.TargetURL=http://IntegrationEngineHost:port/pipeline-arguments
   ```

   This specification is mandatory when the Integration Engine is acting as a service provider.

  💡

   The Integration Engine address can also be accessed dynamically from the SAP System Landscape Directory. The following entry is added to the configuration for this purpose:
   
   ```
   XMB.SLDConfiguration=SLDaccessor
   ```

   In this case, for the system specified with XMB.SenderBusinessSystem, the URL of the corresponding Server in the SLD is exported and the value specified under XMB.TargetURL is used instead. Therefore, specify the following:
   
   ```
   XMB.TargetURL=<fromSLD>
   ```

   To be able to access the System Landscape Directory, the SLDaccessor service [Seite 376](#) service must be configured correspondingly and the respective entries must be maintained in the System Landscape Directory.

   If an authentication was specified for the URL (HTTP service) entered in the Integration Engine, use the following address:
   
   ```
   XMB.TargetURL=http://<username>:<password>@IntegrationEngineHost:port/pipeline-arguments
   ```
As an alternative, or if the URL is taken from the System Landscape Directory, you can also specify the logon parameters as follows:

- XMB.User=<user-name>
- XMB.Password=<password>

The specifications must match those that you made in transaction SICF in the Integration Engine. If you do not specify a user and password combination, or the combination is invalid, the system will terminate each attempt to access the Integration Engine with the message Transport Exception: http-Error 401 – Unauthorized.

The user must have the authorizations of group SAP_XI_APPL_SERV_USER on the Integration Server.

If you want to use a different client or language to the default setting in the Integration Engine when logging on, you can also set the following parameters:

- XMB.Client=<client-no>
- XMB.Language=<language-id>

The following address arguments are optional. If this information is not contained in the request message of a Web Service client, the specifications made here are used to identify the adapter configuration in routing [Seite 214] and mapping [Seite 143] in the Integration Engine pipeline [Seite 267]. You can also find an explanation about each argument there.

- XMB.SenderBusinessSystem=<configuration name>
- XMB.SenderInterfaceNamespace=<namespace URI>
- XMB.SenderInterfaceName=<name>
- XMB.ReceiverBusinessSystem=<configuration name>

If you specify this parameter you do not need to determine a receiver in routing.

The following arguments are mandatory:

- XMB.QualityOfService=<QualityOfService>
  Specifies how the Integration Engine should process a message. The following values are permitted:
  - XMB.QualityOfService=BE (Best Effort, means synchronous processing)
  - XMB.QualityOfService=EO (Exactly Once, means asynchronous processing)
  - XMB.QualityOfService=EOIO (Exactly Once in Order, means asynchronous processing using queues)

  You must also define a queue name for EOIO:

- XMB.QueueId=<QueueName>
  This queue name is used in the Integration Engine to process messages in the same sequence that they arrived in.

2. Specify the port number and the path for the adapter port for Web Service clients:

- XMBWS.WSPort=<port_no>
  The port number (<port_no>) specifies the HTTP server port of the adapter that contains Web Service messages.

- XMBWS.WSPath=<path>
The path `<path>` describes the service part of the adapter URL that contains Web Service messages.

**Configuring the Helper Class for Outbound Processing**

1. Enter the complete address (URL) of the Web Service provider that you want to send the message to:
   
   ```
   XMBWS.TargetURL=http://WebServiceHost:port/service-arguments
   ```
   
   This specification is mandatory when the Web Service provider is acting as a service provider.

   If an authentication was specified for the URL (HTTP service) entered in the Web Service provider, use the following address:
   
   ```
   XMBWS.TargetURL=http://<username>:<password>@WebServiceHost:port/service-arguments
   ```
   
   Alternatively, you can also specify the logon parameters as follows:
   
   - XMB.User=<username>
   - XMB.Password=<password>

2. Specify the value for SOAPAction:
   
   ```
   XMBWS.DefaultSOAPAction=<default_soap_action>
   ```
   
   `<default_soap_action>` specifies the default value for the SOAPAction in Web Service messages.

   ```
   XMBWS.UseSOAPAction=<use_soap_action_opt>
   ```
   
   `<use_soap_action_opt>` specifies the method with which the value of the SOAPAction is determined on the basis of the message instance.

   The only valid value is `receiver`.

3. Specify how the format of Web Service messages is to be controlled
   
   ```
   XMBWS.KeepHeaders=<boolean>
   ```
   
   If you specify `true` here, all Integration Engine message headers are copied to Web Service messages. Otherwise, all headers are deleted.

   Web Service message headers are always copied to Integration Engine messages.

   ```
   XMBWS.UseEncoded=<boolean>
   ```
   
   If you specify `true` here, Integration Engine message headers are coded in the http header `X-XMB_WS_ENCODED`.

4. Specify the port number and the path for the adapter port for the Integration Engine:
   
   ```
   XMBWS.XMBPort=<port_no>
   ```
   
   The port number `<port_no>` specifies the HTTP server port of the adapter that contains messages from the Integration Engine.

   ```
   XMBWS.XMBPath=<path>
   ```
   
   The path `<path>` describes the service part of the adapter URL that contains messages from the Integration Engine.
Using the Dispatcher: Example

Most of the adapters in the Adapter Engine include a dispatcher that can process a series of user exits for converting messages. For example, this dispatcher enables you to carry out file/FTP adapter conversions, XSLT transformations, or Java-based transformations either individually, or combinations of all three.

The example below describes how the dispatcher can be used.

Example

A JMS message is sent to the Integration Engine by using the inbound JMS adapter. The data appears in a list separated by semicolons. It is converted to an XML document in the JMS adapter using the functions already available and used in the file/FTP adapter. This XML document can then be processed further by an individual XSLT stylesheet.

The following parameters must be set in the inbound JMS adapter:

- `transformClass=com.sap.aii.messaging.adapter.trans.JMSSinglePayloadBinding`
  
  You must specify this transformation class because the JMS payload only contains the data and no XI message.

  This parameter is only required for the JMS adapter and is therefore specific to this example.

- `Dispatcher.class=com.sap.aii.messaging.adapter.ConversionDispatcher`
- `Dispatcher.namespace=namespace1`

  Both these parameters activate the dispatcher function and set the namespace used to `namespace1`.

You can display the results of the individual conversion steps for test purposes by entering the following in the log file:

- `namespace1.ConversionDispatcher.logPayload=true`

  `namespace1` is the namespace specified above (the same applies for the parameters that follow)

The file/FTP adapter conversion routines and their settings are used because of the following entries (for further information about the individual settings refer to Configuring the Inbound File/FTP Adapter [Seite 381]):

- `namespace1.Service.1=Plain2XMLService`
  
  `Plain2XMLService` is an arbitrary name.

- `namespace1.Plain2XMLService.class=com.sap.aii.messaging.adapter.Conversion`
- `namespace1.Plain2XMLService.xml.conversionType=SimplePlain2XML`
  
  Depending on the processing mode, you can specify `SimplePlain2XML`, `StructPlain2XML` or `SimpleXML2Plain` as the value for this parameter.

- `namespace1.Plain2XMLService.xml.processFieldNames=fromConfiguration`
- `namespace1.Plain2XMLService.xml.fieldNames=a,b,c`
The following entries in the conversion file then produce the subsequent XSLT conversion:

- namespace1.Service.2=XSLTService
  XSLTService is also an arbitrary name.
- namespace1.XSLTService.class=com.sap.aii.messaging.adapter.XSLTC
  onversion
- namespace1.XSLTService.XSLTConversion.XSLTFileName=Data/DemoConv
  ersion.xsl

In this example it is assumed that the stylesheet to be used has the name DemoConversion.xsl and that it is located in the subdirectory Data.

---

**Runtime Workbench**

**Purpose**

You use the Runtime Workbench to test and monitor the following Exchange Infrastructure components.

- System Landscape Directory [Seite 255]
- Integration Directory [Seite 21]
- Integration Repository [Seite 19]
- Mapping [Seite 173] runtime
- Runtime Workbench [Extern]
- Available Integration Engines [Seite 266]
- Available Adapter Engines [Seite 362]

Furthermore, you can display the configuration of the connections from the Integration Servers to the business systems [Extern] defined in the system landscape, provided that they are business systems that are based on SAP Web Application Server 6.20 or higher. You can also apply security settings for the communication with the respective Integration Server.

**Features**

The Runtime Workbench comprises a browser-based interface that is divided in two and that you call from the Exchange Infrastructure start page that is also browser-based. From here you have an overview of the entire system landscape.

The Runtime Workbench gathers the information required from the System Landscape Directory, the exchange profile and from the components themselves. For this reason, it is only possible to display the SAP Exchange Infrastructure components that are correctly maintained in the SAP System Landscape Directory.

You can select the individual components of your system landscape (excluding the business systems) from the Component Test tab page on the left-hand side of the screen. Depending on the selection made, the following information is displayed in the window on the right:

- Selected component type for components (see list below) from the Integration Engines and Adapter Engine nodes
  - Integration Engines
  - Integration Servers [Extern]
  - Pipelines [Seite 267]
You can then do the following for the individual components:

- **Perform simple component tests** [Seite 437]
- **Call configuration information** [Seite 439]
- **Access component-specific administration tools** [Seite 439]

You can access the business systems [Seite 441] from the **Connection Configuration** tab page.

You can load the landscape descriptions from the System Landscape Directory from the menu on the start page. Here you can also test the availability of the individual components, call the online documentation, and return to the start page.

If an error occurs when reloading the landscape descriptions, once the loading process is complete you can display information about the error by choosing **Show/Hide Details**.

The availability test performs a status query (Ping) for all existing components. The current status of all components on the **Component Test** tab page is then displayed. A green traffic light means that the component is available, a red traffic light means that it is not available.

![Both the Integration Engine and Adapter Engine nodes show a red traffic light if just one of the components they include is not available.](image)

More information about the status of a component is displayed on the right-hand side of the screen once you select a component.

![To ensure that the components can be communicated with, the service user XIRWBUSER with the password XIPASS and the role SAP_XI_RWB_SERV_USER must be available in all of the components.](image)

### Performing Component Tests

#### Use

This function enables you to perform simple component tests. Note that a distinction is made between a Ping and a self-test.

#### Prerequisites

You have started the **Runtime Workbench** [Seite 436] from the **Exchange Infrastructure** [Seite 1] start page and have chosen one of the following components on the **Component Test** tab page on the left-hand side of the screen.

- **System Landscape Directory** [Seite 255]
- **Integration Directory** [Seite 21]
- **Integration Repository** [Seite 19]
You cannot perform component tests for business systems. The same applies for Integration Servers, pipelines, pipeline services, and adapters under the nodes Integration Engines or Adapter Engines.

The system displays a window on the right-hand side of the screen with the name of the selected component.

### Activities

You can query the status of the selected components and, with the exception of the System Landscape Directory, perform a self-test. To do so, perform the following activities in the window on the right-hand side of the screen:

- **To query the current status of the component**, select the Ping tab page and then choose Start Ping (or choose Start Ping directly if in the System Landscape Directory). The status of the component is displayed in the form of a traffic light. The individual statuses (red, amber, and green) are explained by a corresponding short text.

  - If the status is OK (green), the following additional information is displayed:
    - Integration Engines: Name of the business system (the Integration Engine), vendor, role of the Integration Engine, and client.
    - Adapter Engines: Name of the component, vendor, host name, and port number.
    - All other components: Name of the component and vendor.

  - To repeat the status query, choose Repeat Ping.

- **To execute an Integration Directory self-test**, select the tab page Self-Test and then choose Start Self-Test. If the self-test fails, the system displays a corresponding error message.

  - If the self-test is executed, the overall status of the components (Status) as well as the various individual statuses are displayed as traffic lights (red, amber, or green) and are explained by a corresponding short text.

  - The status is only green (OK) if all other traffic lights are likewise green.

  - The Self-Test tab page is only displayed for Adapter Engines if the component supports a self-test. However, it is always displayed if the component is not correctly maintained in the System Landscape Directory.

  - To repeat the self-test, choose Repeat Self-Test.
**Calling Configuration Information**

**Use**

Use this function to call configuration information for individual components.

**Prerequisites**

You have started the Runtime Workbench [Seite 436] from the Exchange Infrastructure [Seite 1] start page and have chosen one of the following components on the Component Test tab page on the left-hand side of the screen.

- Integration Directory [Seite 21]
- Integration Repository [Seite 19]
- Mapping [Seite 173] runtime
- Runtime Workbench [Extern]
- A component from the Integration Engine node
- One of the available Adapter Engines [Seite 362]

It is not possible to call configuration information in the System Landscape Directory [Seite 255] and in business systems [Seite 441].

The system displays a window on the right-hand side of the screen with the name of the selected component.

**Activities**

You can check the configuration parameters [Seite 269] for the components displayed. To do so, on the Configuration Parameters tab page select a category of configuration parameters that you wish to check from the dropdown list box.

The Configuration Parameter tab page is only displayed if the relevant component is correctly maintained in the System Landscape Directory (Adapter Engines and components from the Integration Engines node) or in the exchange profile (all other components). If this is not the case, only the Ping and Self-Test tab pages are displayed. If you then choose either Start Ping or Start Self-Test, the system displays a red status traffic light and the message Component not available.

All available Integration Engines including the defined (and highlighted in bold) Integration Servers [Extern] are displayed under the node Integration Engines. The corresponding pipelines [Seite 267], pipeline services [Seite 267] and adapters [Seite 367] are located beneath each Integration Engine or Integration Server. You can select and display each of these components individually.

**Accessing Administration Tools**

**Use**

This function enables you to do the following:
- Navigate to the System Landscape Directory [Page 255] administration screen
- Call the configuration screen for the selected Adapter Engine [Page 362]

**Prerequisites**

You have started the Runtime Workbench [Page 436] from the Exchange Infrastructure [Page 1] start page and have selected the following on the Component Test tab page on the left-hand side of the screen:

- The System Landscape Directory
  The System Landscape Directory window is displayed on the right-hand side of the screen with the host name and the port number of the System Landscape Directory.
- An Adapter Engine
  The type and a description of the selected Adapter Engine, as well as the respective application system is displayed on the right-hand side of the screen.

All available Adapter Engines are displayed under the Adapter Engines node. Located under each Adapter Engine are the corresponding adapters. You can select and display each of these components individually.

For information about individual adapters, each adapter instance must be maintained in the System Landscape Directory. This takes places automatically when you start the Adapter Engine.

**Activities**

**System Landscape Directory**

You can access the System Landscape Directory administration tool from the System Landscape Directory window. To navigate to the administration screen of the System Landscape Directory, choose Administration Tool.

The system displays the initial screen of the System Landscape Directory.

**Adapter Engine**

To call the configuration screen [Page 367] for the relevant Adapter Engine, choose Administrations Tool.

The Runtime Workbench can access an Adapter Engine for monitoring purposes. To log on you require the technical user XIRWBUSER and the password XIPASS, but no other authorizations. This user is created automatically when the Adapter Engine is installed. If this user is deleted, or if the Runtime Workbench requires an additional user, create a corresponding user on the Adapter Engine configuration screen. The user to be created only requires the authorization group Locked. For more information, see the Adapter Engine documentation under User Management [Page 372].
Connections to Business Systems

Use

Use this function to access data from available business systems that are based on SAP Web Application Server 6.20 or higher.

Business systems are logical systems used within Exchange Infrastructure as either sender or receiver systems. These can be either SAP or non-SAP systems. For example, a business system can be a client in an SAP system or a Java standalone application.

Prerequisites

You have started the Runtime Workbench from the Exchange Infrastructure initial screen, have selected the tab page Connection Configuration, and the system is now displaying all the available business systems.

For business systems based on SAP Web Application Server 6.20 or higher you must at least have maintained the role of the business system (Integration Server or Application System). If it is an application system, the corresponding Integration Server must be assigned.

Features

All Integration Servers that exist in the system landscape are listed on the Connection Configuration tab page. The business systems that are assigned to each of the Integration Servers are listed in turn below. You can select and display each of the business systems individually.

When you select a business system, the type and the role of the business system is displayed in the window on the right-hand side of the screen. These two pieces of information are specific to Exchange Infrastructure. A description (if available), the client, the system name, the message server, the application server (optional), and the system number are also displayed.

Furthermore, the Connections tab page displays the name and the version of the business system, as well as the corresponding Integration Server.

Activities

On the Connections tab page, you have the option of choosing Details and then logging on to the systems displayed to display detailed information about security settings, for example.

To do so, you require a corresponding user for the systems concerned.

Enter the user name and password and choose Log On.

- If a connection already exists between the two systems, detailed information about the connection between the business system (BS) and the Integration Server (IS) is displayed. You can also delete the connection again (Delete Connection).
  - If you want to delete an existing connection, the system calls a corresponding wizard. To delete the connection, follow the wizard’s instructions.
- If no connection already exists between the systems, the system calls a wizard with which you can establish a connection of this type. Follow the wizard’s instructions.
SAP System Landscape Directory

Purpose
A modern computing environment consists of a number of hardware and software components that depend on each other with regard to installation, software updates, and demands on interfaces. To administrate system landscapes better, you require the SAP System Landscape Directory (SLD).

SLD is a server application that communicates with a client application by using Hypertext Transfer Protocol (HTTP). The SLD server contains component information and a description of the system landscape, which are based on the standard Common Information Model (CIM). The CIM standard is a general schema for describing the system. This standard is independent of any implementation.

Features
The component description provides information about all available SAP software modules. This includes version numbers, current patch level, and dependencies between system components. SAP makes this information available to its customers. You can request the current component description from SAP, which updates your local component description. It is also possible to add instances for third-party components to the component description.

The system landscape description represents the exact model of an actual system landscape. Together with the current component description, the system description provides important information for different processes (the system administration and implementation, for example).

The example below shows a possible scenario that illustrates how the component and system landscape description functions.

Example
On the left-hand side of the following graphic is the master description for all existing SAP software modules. SAP maintains this information. The local component description on the right-hand side (client side) can be updated in accordance with the master description.

An installed mySAP.com component is registered in the System Landscape Directory. The component description contains information about the installed components. If, for example, a new Support Package is available for this component, SAP publishes this information using the master description. In this way, the customers receive all the latest information relevant for their system landscape promptly.
CIM Concept

The Common Information Model (CIM) is a standard of the Distributed Management Task Force (DMTF) and is based on the object-oriented modeling approach. This standard provides a neutral implementation schema to describe management information within a computing environment.

Object-oriented modeling is a means of representing the real world. CIM is designed to model hardware and software elements. The following are basic terms associated with this concept:

- Class and inheritance
- Instance
- Association and reference
- Schema

Class and Inheritance

A class is a collection of objects that have similar properties. It is a template for a type of object. CIM supports the inheritance concept so that classes in the context of CIM are organized hierarchically.
Inheritance is an object-oriented concept. A class automatically has all the properties of the class from which it is derived. The class that passes on properties is called the superior class. The class that inherits properties from a superior class is called the subclass. A subclass represents a specialization of the superior class by enhancing it with additional properties.

There are also abstract classes. These are classes that do not have a particular form, but which represent the superior class for one or more subclasses.

**Example**

The CIM class `SAP_Product` defines the number of SAP products that have joint properties such as the same product name, version, and so on.

**Instance**

An instance is a form of a class, or an object of the class. All properties defined in the class have fixed values in an instance.

**Example**

The CIM class `SAP_Product` can have instances such as `mySAP CRM` and `R/3 Enterprise`.

**Property and Key Property**

An instance has certain properties. A class declares some of these properties as key properties. The CIM concept uses key properties to clearly identify instances of a class. This identification is also known as the name of an instance. This means that there are no two instances of one class whose key properties are completely identical. Key properties are mandatory for any given instance.

**Example**

The CIM class `SAP_ProductLine` declares `Vendor` and `Name` as key properties. In addition, this class also has properties such as `Caption` and `Description`. The product `CRM IPC` differs completely from other products with the name `sap_productline.name="CRM IPC",vendor="sap.com"`.

**Qualifier**

Qualifiers contain metainformation that describes classes, instances, and properties in more detail.

**Example**

Many CIM classes or their instances have a `Caption` property. This property is a short description and has, among other things, a `maxlen` qualifier, which restricts the length of the description.

**Association and Reference**

Associations are a type of class that represent relations between classes or their instances. An association has one or more references as properties, which refer to corresponding CIM instances. In this way you can define relations between classes or their instances, without affecting the definitions of the classes concerned.
Example
The association class `SAP_ApplicationSystemHost` represents the connection between a computer (hardware) and the SAP system (software) running on this computer.

Schema
A schema in CIM is a collection of classes and has a name. You use a schema to name a class. A CIM class can belong to one schema only. You name a schema, class, and property in CIM by using the syntax `schemaname_classname.propertyname`.

The CIM standard contains a number of class definitions that represent the common model. This common model is called the CIM schema. You can extend this common model by adding technology-specific class definitions. These extensions are referred to as extension schemas.

Example
All CIM classes in the examples above belong to the SAP extension schema, which represents an enhancement of the CIM schema.

Component Information

Purpose
The SLD server contains component information about all existing software modules, as well as their combination options and dependencies. This information represents the basis for the description of the system landscape [Seite 446].

SAP provides component information about all SAP software modules. You can get the current model data from the master component information provided by SAP.

Features
The component description provides you with the following information:
- Software product and software component attributes
mySAP.com components
- Software components
- Releases and Support Packages

- Dependencies:
  - Supported platforms, versions of operation systems, databases, and so on
  - Combination options and integration matrix

**Example**

*SAP Basis 6.20* is a software component. It is registered in the component information as a CIM instance of the class *SAP_SoftwareComponent*. The entry contains important information about this component (date of implementation and end of support, for example). In addition, all existing Support Packages for the components are registered as associations.

---

**System Landscape Description**

**Purpose**

The system landscape description is an exact model of the installed system landscape. The classes in the system landscape description are linked to those classes in the component information by using associations. In this way, an installed component that is entered in the system landscape description can quickly find current, relevant information by using its corresponding instances in the component information.

**Features**

Each system landscape has one system landscape description. It provides information about the following:

- System topology (network addresses and links, for example)
- Component structures
- Component information of the installed software
- ...

**Example**

A system in the system landscape has the component SAP Web Application Server 6.20 installed. This installed software component has an entry in the system landscape description as an instance of the CIM class *SAP_InstalledSoftwareComponent*. Using the association *SAP_SoftwareComponentType*, this instance establishes a connection with its counterpart in the component description, namely the instance of the CIM class *SAP_SoftwareComponent*. In this way, this instance gets all the relevant information from the component information.

---

**Administrative Activities**

**Purpose**

You must perform the following administrative tasks to ensure that the SLD service functions correctly.

- Server administration:
Start and stop the SLD service [Seite 447] [Seite 447]
Display system information [Seite 448] [Seite 448]
Configure the system [Seite 449] [Seite 449]
Configure data persistence [Seite 450] [Seite 450]
Monitor the system [Seite 451] [Seite 451]

Data Management:
Maintain data [Seite 453] [Seite 453]
Navigate in the structure view [Seite 455] [Seite 455]
Import [Seite 456] and export [Seite 457] data models
Trace model modifications [Seite 457] [Seite 457]
Administer namespaces [Seite 458] [Seite 458]

Prerequisites
To administer the SLD, you require the authorizations contained in the user role LcrAdministrator.

If you do not have the authorizations required, contact your system administrator.

Logging On as Administrator

Prerequisites
To access the administration area, you require a user ID with the user role LcrAdministrator. If you do not have a user ID with this authorization, contact your system administrator.

Procedure
When you call a server function for the first time, the system prompts you to log on so that it can check your authorizations. If you do not have the necessary authorizations for the function that you want to call, the system prompts you to log on again.
Enter your user name and password, and then choose OK.

Result
The system checks your user name and password. If both are valid, the browser displays the administration area.

Starting and Stopping the SLD Service

Use
To use the SLD service, you must start the SLD server. You can exit the service regardless of the application server it runs on.
The browser displays a status bar that indicates the current status of the service. When the service is running, the status bar displays *Running*. If the service is not active, the status bar displays *Stopped*.

**Prerequisites**

You have navigated to the *Administration* screen.

**Starting the SLD Service**

Choose *Start Server*.

The system starts the SLD service.

**Stopping the SLD Service**

Choose *Stop Server*.

The system stops the SLD service.

**Result**

When the service has been started or stopped successfully, the status bar indicates the corresponding status. If this is not the case, check the [system log](#) to localize possible error causes.

---

**Displaying System Information**

**Use**

To perform administrative activities, you need an overview of the client-server application that is running. You can use the function described here to display information about the following:

- Server system
- Data models on the server
- Client-server connection
- Browser

**Prerequisites**

You have navigated to the *Administration* screen.

**Procedure**

1. Choose *Server → Details*.
   
   The browser displays the *Details* screen.
2. Select the tab page that you want.
Configuring a System

Use
You can configure the SLD service by using parameter settings to influence how the server acts, and to influence its performance. This configuration is called a system profile.

For more information about configuration, see the Installation and Configuration Guides.

Prerequisites
To modify configuration data, you require the user role LcrAdministrator.

Setting Parameters
To modify a system profile, proceed as follows:

1. On the Administration screen, choose Server → Profile.
   The browser displays the Profile screen.
2. Make the modifications and then choose Save.
   The modifications are saved.

If you want to discard modifications that you made, choose Reset to Defaults. The parameter values recommended by SAP take effect again.

Some parameters require you to restart the SLD server after they have been changed so that the changes take effect. This is why the system automatically stops after you have changed these parameters.

Downloading a Configuration File from the Server
You can also download the parameter settings as a compressed XML file (zip) to process them locally.

1. On the Profile screen, choose Download.
2. Choose Save This File to Disk, and then enter the file name and the directory where you want to save the file.
   The configuration file is now available for processing in the specified directory.

Transferring a Configuration File to the Server
After you have processed the configuration file as described in the previous section, you have to transfer this file back to the server. You must also do this after the SLD server has been installed.

1. On the Profile screen, choose Upload.
   The browser displays the Profile Upload screen.
2. Enter the file name and then choose Upload.
   The browser transfers the file to the server.

   You can transfer the configuration file to the server directly as an XML or zip file.
The changes to the configuration take effect only when you restart the SLD service [Seite 447].

Restoring the Default Settings
To restore the default settings, go to the Profile screen and choose Reset to Defaults.

Configuring the Data Persistence

Use
The SLD service supports two ways of storing data: file system persistence and database persistence. In a distributed computing environment we recommend that you save data centrally in a database.

In the local system, file system persistence is useful for test purposes only. This section informs you how to set the parameters for database persistence.

Prerequisites
You have navigated to the Administration screen.
The SLD service has been stopped.

Configuring Database Persistence
To set the parameters for database persistence, proceed as follows:

The browser displays the screen Persistence Settings.
2. Under Database Connection Parameters, enter the authentication and connection data for the database server, and choose Set.
The SLD server saves the data.

For security reasons, the browser displays the password with a placeholder (*).

You can switch to file system persistence by choosing Persistence → File System → Set.

Result
The SLD server establishes a connection to the specified database server.
System Monitoring

Use
During a system process, the SLD server logs relevant information for monitoring purposes. This process is called a Trace. The trace levels specify which message classes the system records. You can set logging on one of the following trace levels:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>None</td>
</tr>
<tr>
<td>1</td>
<td>Warn</td>
</tr>
<tr>
<td>2</td>
<td>Info</td>
</tr>
<tr>
<td>3</td>
<td>Debug</td>
</tr>
<tr>
<td>4</td>
<td>Fine</td>
</tr>
<tr>
<td>5</td>
<td>Finer</td>
</tr>
<tr>
<td>6</td>
<td>Finest</td>
</tr>
</tbody>
</table>

A trace level includes all levels that are lower. (In the table above they are numerically lower.) This means that the system records all messages of the set classes and of the ones lower than them. Additionally, the two message classes Error and Fatal are always recorded. Error displays application and system errors that do not prevent the application from continuing. Fatal, on the other hand, flags errors that cause the application to stop.

The higher the trace level, the more extensive and detailed the log, and the larger the log file.

Prerequisites
You have navigated to the Administration screen.

Setting a Trace Level
To set the global trace level, proceed as follows:

1. On the Administration screen, choose Server → Server Log.
   The browser displays the screen Server Log.

2. In the drop-down list Server Log Level, select the trace level that you want to implement.
   The global trace level that you selected is now implemented.

To optimize system performance, you can choose lower trace levels.

Viewing the System Log

1. On the Administration screen, choose Server → Server Log.
   The browser displays the screen Server Log.

2. To set the filter category and the filter level, make corresponding entries in the drop-down lists Filter Category and Filter Level.
   The filter category specifies in the Java package hierarchy the package whose log data you want to view (including the log data for the subpackages).
The filter levels correspond to the trace levels at the start of this section.

3. Choose **Refresh**.

The browser displays the log data you selected.

To update the display automatically, select the **Auto-Refresh** option.

### Deleting the System Log

On the **Server Log** screen, choose **Clear**.

⚠️ The log data is deleted permanently.

### Downloading the System Log from the Server

You can download the system log from the server to view it locally or archive it. Proceed as follows:

1. On the **Server Log** screen, choose **Download Log**.
2. Choose **Save This File to Disk**, and then enter the file name and the directory where you want to save the file.

The system log is now located in the specified directory.

### Downloading the Configuration File for Logging from the Server

You can also download the configuration for logging in the form of a file from the server to process the file locally. Proceed as follows:

1. On the **Server Log** screen, choose **Download Settings**.
2. Choose **Save This File to Disk**, and then enter the file name and the directory where you want to save the file.

The configuration file is now available for processing locally.

### Transferring the Configuration File for Logging to the Server

After you have modified the configuration file, you have to transfer it back to the server. Proceed as follows:

1. On the **Server Log** screen, choose **Upload Settings**.
2. Enter the file name and then choose **Upload**.

The file is transferred to the server.

⚠️ The changes to the configuration take effect only when you **restart the SLD service** [Seite 447].
Maintaining Data

Use
You can use the generic data maintenance editor to view and manipulate the content of the SLD directly at model level. This is useful for objects that have no specific maintenance interface.

Maintenance interfaces exist for most SAP applications, in particular for the SAP Exchange Infrastructure. Use these specific interfaces for maintaining data so that the applications run smoothly.

If you use the generic maintenance editor, it could destroy the data integrity, which means that a smooth operation of the applications is no longer guaranteed. Use the generic editor only if you are familiar with the CIM concept and the data models of the specific applications.

Prerequisites
To call up the maintenance editor and edit the data, you require the user role LcrAdministrator.

Procedure
There are two ways of calling up the generic data maintenance editor. Choose Content Maintenance:

- Directly on the initial screen of the SLD
  or
- On the Administration screen

The browser displays the Content Maintenance screen.

There are two drop-down lists on this screen:

- Subset: Determines the subset of the data that you want to display. You can choose Component Information or Landscape Description.
  - Class: Determines the class whose instances you want to view and process. The number in parentheses after the class name indicates the number of instances.

To display a particular CIM class or CIM instance, choose the corresponding list.

The browser displays an overview screen with all the instances of the selected CIM class.

On the overview screen there is a Display Filter input field to the right of the selection lists. You can enter strings in this field that you want the return values to contain.

If you want to display all products, choose Component Information in the Subset selection list, or Product in the Class selection list.

The browser then displays all registered products.

Example: If you enter the string CRM in the Display Filter field, and confirm this with Enter, the browser displays only the rows that contain this string.
You can use the links in the Assoc's column to display all associations of the relevant instances.

If you select an instance on the overview screen, the browser displays the detailed view of this instance.

**Activities on the Overview Screen**

This section describes activities that you can perform on the overview screen.

To create an instance of the selected CIM class, choose **New <Class Name>**. (<Class Name> stands for the selected class.)

The browser displays the detail screen for the new instance. Enter the instance data and then choose **Create**.

The system creates the new instance.

The entries for key properties are mandatory. The browser displays key properties on the detail screen in bold font. If the key properties are missing, the system triggers an exception.

To delete multiple instances simultaneously, proceed as follows:

1. Select all the instances that you want to delete.
2. Choose **Remove Selected**.

The system deletes all selected instances.

To view an instance in more detail, or to process it, select the instance directly by clicking the instance name.

The browser displays the detail screen for the selected instance. This screen displays detailed information about the instance.

**Activities on the Detail Screen**

This section describes activities that you can perform on the detail screen.

To change the instance, proceed as follows:

1. Make modifications on the detail screen.
2. To save the changes, choose **Update**.

The system saves the changes.

To create a copy of an instance, proceed as follows:

1. On the detail screen of the instance that you want to copy, choose **Copy**.

The browser displays the detail screen for a new instance. All fields on this screen are identical to the original instance.

2. Make modifications to the key properties.
3. To save the copy, choose **Create**.

The system saves the copy.

The CIM concept does not permit completely identical instances. At least one of the key properties must be different. In this case, copying means only that the data of the original instance is copied as a template for the new instance. The
new instance must have at least one key property that is different from the original instance.

To rename an instance, proceed as follows:

1. On the detail screen of the relevant instance, choose Move.
2. Change the key properties of the instance.

   You also change other property values here.
3. Save the instance by choosing Create.

   The system saves the renamed instance. This instance replaces the original instance.

To delete an instance, choose Remove.

To view the associations of an instance, choose <num> Associated Instances. (<num> is the number of existing associations.)

The browser displays the Association screen.

**Activities on the Association Screen**

An association screen displays all associations of a specified CIM instance and its associated CIM instances. This section describes the activities on the association screen.

To delete associations, select the relevant rows, and then choose Remove Selected.

The system deletes the selected association instances. The instances that are connected with the original instance by these associations remain.

To create associations that start with the specified instance, proceed as follows:

   The browser displays a dialog box with the drop-down list Association Type.
2. In this selection list, select the relevant association type.
   The browser then displays all the relevant instances.
3. Select the instances that you want to link to the specified instance by using the selected association type.
4. Choose Associate Selected.

The new associations are created.

---

**Navigating in the Structure View**

**Use**

CIM is based on the object-oriented concept. Classes and instances are ordered hierarchically. SLD provides a structure view for the component information and system landscape description, which reflects this hierarchical order. You can navigate in this hierarchy to access particular class and instance definitions.

**Procedure**

To display the structure view, go to the Administration screen and choose Class Browser.

The Web browser displays the structure overview.
On the left-hand side of the structure view there is a tree-like overview. On the right-hand side there is a detailed view for individually selected objects. The tree view has three tab pages: system landscape description (LD), component information (CR), and all models (All). To switch between the three structure overviews, choose the corresponding tab page.

To open the entire structure view, choose **Expand All**.

To update the structure view, choose **Refresh**.

When you select an object in the tree view, the right-hand side of the screen displays detailed information about the object. In the detailed view, you can also display the XML definition for each object.

**Importing CIM Models**

**Use**

The component description and the system landscape description are based on the standard Common Information Model (CIM) [Seite 443]. To insert these descriptions into a system, or to update a description that already exists in a system, import the model from a master file.

**Prerequisites**

You have navigated to the *Administration* screen.

The model descriptions must be either XML files or compressed XML files in ZIP format.

**Procedure**

1. Choose **Content → Export**.
   
   The browser displays the **Import** screen.

2. Specify the name of the local file where the component or system landscape description is located.

3. To select the file name interactively in the dialog, choose **Browse**.

4. If you want to overwrite existing objects in the system, choose **Modify Existing Objects**.

5. Choose **Import Data**.

   SAP has made changes to the CIM models for SAP Exchange Infrastructure 2.0. If you want to import CIM data from SAP Exchange Infrastructure 1.0, choose **Upgrade Content from XI 1.0** to make the system perform a migration.

   If you do not want to import the models into the current namespace, you can change the namespace [Seite 458].

   The CIM standard is based on the object-orientated approach. Ensure that you maintain the import sequence when importing models from multiple files, since this has an effect on the inheritance hierarchy.
Result
After you have triggered the import, the browser returns to the Administration screen. The status bar on this screen indicates the status of the import procedure.

Exporting CIM Instances

Use
Before you maintain your system, we recommend that you create a backup copy for the instances in the component and system landscape descriptions. The system creates an XML file when exporting instances. This file contains all relevant data regarding the instances.

We also recommend that you transfer instances from one system to another system. The simplest method is to export the instances from the source system and then to import them into the target system.

Prerequisites
You are logged on with the user role LcrAdministrator, and have navigated to the Administration screen.

Procedure
1. Choose Content → Export.
   The browser displays the Export screen. The browser displays the namespace to be exported.
2. To export CIM classes as well, choose Include Classes.
3. Choose Export. Follow the instructions in the dialog to save the XML file locally.

Result
The system compresses the exported XML file in ZIP format. You can import this ZIP file as required.

Tracing Model Modifications

Use
The system logs every change to the model. This includes generating, modifying, and deleting models. You can display the log data according to particular criteria.

Prerequisites
You have navigated to the Administration screen.

You can perform this action only if database persistence is activated.

Procedure
1. Choose Change Log.
The browser displays the Change Log screen.

2. To restrict the search, enter search criteria here.

You have the following choice of search criteria:

- **User Name**: User who made the changes.
- **Class or Qualifier Name**: Name of a CIM class or a CIM qualifier. If it is a class name, the system returns modifications to the declaration of the class and modifications of its instances. If it is a qualifier name, the system returns the modifications to the qualifier declaration.
- **Time Interval**: Time range in which the changes were made.
- **Object Types**: Types of changed objects (classes, instances, and associations).
- **Event Types**: Type of change (generation, change, and deletion).

3. Choose *Search*.

The browser displays the results screen.

**Result**

The browser shows all changes in the current namespace that meet the search criteria. In the case of a modification, the browser also shows the old object.

**Administering Namespaces**

**Use**

The component and system landscape description contains current information about your system landscape. Using SLD simulations, you can also plan the future system landscape. We recommend that you test simulations in a different area to the current system landscape.

The SLD takes this consideration into account with the namespace concept. This means that you can create various namespaces as logical areas. The namespace `sld/active` mirrors the real system landscape. You can copy data from the standard namespace to other namespaces, and then modify and test the data there.

You have the option of switching between different namespaces.

**Creating Namespaces**

To create a namespace, proceed as follows:

1. On the *Administration* screen, choose *Content* → *Namespaces*.
   
The browser displays the *Namespaces* screen.

2. Choose *Add*.
   
The browser displays a dialog box for entering the new namespace.

3. Enter the name of the new namespace in the input field and choose *OK*.
   
The system creates a new namespace.

**Switching Between Namespaces**

To switch between namespaces, proceed as follows:

1. On the *Administration* screen, choose *Content* → *Namespaces*.
   
The browser displays the *Namespaces* screen.
2. In the field *Database Namespaces*, select the namespace to which you want to switch.

The system switches to the namespace you selected.

You can switch between namespaces on all the screens where a namespace switch is relevant. On these screens, the current namespace is displayed at the bottom of each screen. To switch the namespace from there, proceed as follows:

1. Double-click the current namespace.
   
   The browser displays the *Change Namespace* screen.

2. Select the new namespace from the list, and then choose *Set Current*.

The browser switches to the selected namespace and displays the previous screen.

**Copying Model Data**

When you create a new namespace, it is initially empty. To create a duplicate of another namespace for test purposes, copy the model data from the namespace concerned to the new namespace.

Go to the *Administration* screen and proceed as follows:

1. Choose *Content* → *Namespaces*.
   
   The browser displays the *Namespaces* screen.

2. Under *Namespace Import*, select the source namespace in the drop-down list (*From Namespace*).

3. Choose *Import into <namespace>*. (*<Namespace>* is the current namespace.)

**Deleting Namespaces**

To delete a namespace entirely, proceed as follows:

1. On the *Administration* screen, choose *Content* → *Namespaces*.
   
   The browser displays the *Namespaces* screen. The name of the current namespace is highlighted under *Database Namespaces*.

   ![Warning]
   
   If the browser does not display the namespace you want to delete, first switch to the correct namespace.

2. Choose *Remove <namespace>*. (*<Namespace>* is the current namespace.)

   ![Warning]
   
   This procedure deletes the namespace, including all its data.

**Deleting Instances**

The system deletes these instances permanently. Make a backup copy of the instances using the export function [Seite 457], before you delete any instances permanently.

To delete all instances in a namespace, proceed as follows:

1. On the *Administration* screen, choose *Content* → *Namespaces*.
   
   The browser displays the *Namespaces* screen. The name of the current namespace is highlighted under *Database Namespaces*.

2. Select the namespace that contains the instances that you want to delete.
3. Choose Remove Instances.

Software Catalog

Purpose

The SAP System Landscape Directory contains component information about all the available software modules, including their combination options and dependencies. This information represents the basis for the description of the system landscape. You can get the current model data from the master component information provided by SAP.

You can also enter information about third-party products in the SLD for managing your system landscape.

The software catalog displays this information and enables you to maintain data for third-party products. This section contains information about the following:

- Modeling logic of software components and software products [Seite 460] [Seite 460]
- Displaying the software catalog [Seite 461] [Seite 461]
- Maintaining data for third-party products [Seite 462] [Seite 462]

Software Components and Software Products

The classes of the SAP schema, the software components and their versions, and the software products and their versions are the basis for the component information.

This section deals with the connectivity of these terms as shown in the following graphic:

- *Product*: Represents a collection of all versions of a product. A product is a unit that can be delivered, is visible to the customer, and that is installable and renewable. In an SAP environment, a product corresponds to a SAP technical component.
• **Product Version**: Represents a particular version of a product.

• **Software Component**: Represents a collection of all versions of a software component. Software components represent the reusable modules of a product. They can be upgraded or have patches installed.

• **Software Component Version**: Represents a particular version of a software component.

• **Software Feature**: Represents a particular aspect of the product functions. It is a logical link between *Product Version* and *Software Component Version*.

As shown in the graphic above, a product version consists of multiple software features. These features are implemented by various software components (or versions).

The product *SAP R/3 Enterprise*, which represents an SAP technical component, has, among other things, a product version *SAP R/3 Enterprise Core 4.7*. The software feature that represents the functions of this product version has the same description as the product version. Various software components (or versions) fill the functions of this software feature. These software components (or versions) are, among others, *SAP Basis 6.20*, *SAP HR 4.70*, and *SAP Enterprise Retail 1.10*.

The data for software components and software products for SAP products is provided by the internal SAP *Product and Production Management System* (PPMS). SAP provides this master data for updating your local component information. You can add current data to your SLD by using a data import [Seite 456].

Also, you can enter third-party products in the SLD by using the function third-party product maintenance [Seite 462].

---

### Displaying the Software Catalog

**Use**

You can use the software catalog to display all registered software products and their versions, as well as all registered software components and their versions.

The component information is provided by SAP. You cannot change this information. Third-party products that you enter on the screen *Maintaining Third-Party Products* [Seite 462] are also in the software catalog.

**Prerequisites**

You have navigated to the initial screen of the SAP System Landscape Directory.

**Procedure**

To open the software catalog, choose *Software → Software Catalog*.

The browser displays the *Software Catalog* screen.

In the drop-down list *Software Type*, you can choose whether you want to display products or software components.

Depending on your selection, the browser displays a list of all available products or software components and their versions.

To restrict the display, you can enter a corresponding string in the field *Display Filter*. The browser then displays only the rows that contain this string.
Maintaining Third-Party Products

Use

The *SAP System Landscape Directory* is the central information point for your system landscape. Management and business applications access the information stored there to perform tasks in a collaborative computing environment.

Third-party products are often implemented in a system landscape. These third-party products can be standalone applications or additional modules for SAP applications.

To register third-party products in the SLD, you have to enter the product information manually.

Prerequisites

You have the user role *LcrAdministrator*.

Entering a Product (Version)

Go to the initial screen of the SLD and proceed as follows:

1. In the *Software* column, choose *Third-Party Software*.
   
   The browser displays the screen *Third-Party Software*.

2. Choose *Define Product*.

   The browser displays the screen *Define Product*.

3. Enter the vendor name, product name, and the product version.

4. Choose *Create*.

   The SLD server creates instances for the classes *Product*, *Product Version*, and *Software Feature* as well as the association instances. The browser displays the screen *Define Software Component*.

   If you do not want to create the software components (immediately), you can stop the process by choosing *Cancel*. The system then creates only the product.

   You can create various versions for a registered third-party product by specifying exactly the same vendor and product name. Assign the version numbers accordingly.

Entering a Component (Version)

Go to the initial screen of the SLD and proceed as follows:

1. In the *Software* column, choose *Third-Party Software*.

   The browser displays the screen *Third-Party Software*.

2. Choose *Define Software Component*.

   The browser displays the screen *Define Software Component*.

3. In the *Product* drop-down list, select the product that the defined component belongs to.

4. Enter the vendor name, component name, and the component version.

5. Choose *Create*.
The SLD server creates instances for the CIM classes *Software Component* and *Software Component Version*, as well as the association instances.

You can create various versions for a registered third-party product by choosing exactly the same product, and specifying exactly the same vendor and component name. Assign the version numbers accordingly.

### Implementing an SLD Client

#### Purpose

The SAP System Landscape Directory provides information about the available software products and the system landscape. Applications can access this information on the SLD server. These applications represent clients of the SLD server.

SLD provides programming interfaces (APIs) for Java and ABAP that implement the *specification for CIM operations over HTTP* of the *Distributed Management Task Force (DMTF)*. This specification enables the SLD server to communicate with other systems using an HTTP connection. To guarantee that a software system is able to work with other systems, the XML format is used to represent data, and HTTP is used for data transport.

Applications can use these Java or ABAP APIs to send queries to and receive replies from the server. These applications can be management applications or instrumented business applications. Instrumented applications are applications that have an API.

In addition, SAP has implemented a CIM proxy layer for easy access to the classes of the *CIM Core Schema* and the *SAP Extension Schema*.

The following provides you with the basics for implementing:

- A Java client
- An ABAP client
Java Client for the SLD Service

Concept
To execute a particular CIM operation, a Java client calls the corresponding Java methods of the API. The API packs the operation and its parameters in an XML message. The API then sends the request to the SLD server by using HTTP. The SLD server executes the operation and sends the result, or the error message, in an XML message as an HTTP response to the API on the client side. The API forwards the results as the return value of the method call to the application.

Prerequisites
To implement an SLD client you require the following Java archives:

- `lcrclient.jar`: Implementing the specification for CIM operations over HTTP
- `aii_util_log.jar`: Logging
- `tc_sec_core.jar`: Security
- `inqmyxml.jar`: XML/XSLT processors

All the JAR files named above are included in the SAP Web Application Server.

The next sections show you how to:
- Create a CIM client [Seite 464]
- Process CIM objects [Seite 465]
- List CIM objects [Seite 468]
- Handle exceptions [Seite 469]
- Control logging [Seite 469]
- Use namespaces [Seite 470]
- Use the SAP proxy layer [Seite 470]

For more information, see the JavaDoc package.

Create a CIM Client
To execute CIM operations from the client, proceed as follows:

1. Generate an instance of the class `HttpRequestSender`.
2. Create a CIM client – a client of the class `CIMClient` – with the instance of the class `HttpRequestSender` as a parameter.
3. Call the corresponding methods of the CIM client to call the server operations that you want.

These steps are explained below.

Creating an Instance of the Class `HttpRequestSender`
The class `com.sap.lcr.api.cimclient.HttpRequestSender` takes over the client-server communication by using HTTP. This class implements the interface `com.sap.lcr.api.cimclient.IRequestSender`, which you require later to create a CIM client.
Specify the connection data when creating an instance of this class. On the server side, a servlet – the CIM Object Manager – edits the requests of the clients. The URL of the CIM Object Manager is as follows: http://<HOST>:<PORT>/sld/cimom; you must specify the host and the port individually. /sld is the preset name where the SLD server is installed as J2EE components on the application server.

Use the following constructors to get an instance for HTTP connections:

1. `public HttpRequestSender(java.net.URL cimomUrl, java.lang.String user, java.lang.String password)`
   The first parameter represents the URL of the CIM Object Manager.

2. `public HttpRequestSender(java.lang.String host, int port, java.lang.String user, java.lang.String password)`
   The first two parameters specify the server names and the port where the CIM Object Manager waits for requests. The API fits together the standard URL by using the template mentioned above.

⚠️

The J2EE standard prescribes that the name of a component is determined when it is installed. That is why the second constructor works only if the SLD server is installed under the name /sld.

You must also specify the user ID and the password for authentication.

**Creating a CIM Client**

The class `com.sap.lcr.api.cimclient.CIMClient` is a main component that implements the Specification for CIM Operations over HTTP. To create an instance of this class, call the constructor `public CIMClient (IRequestSender aSender)`.

Specify the instance of the class `HttpRequestSender` as a parameter. Using this constructor you can access the default namespace. You can change the namespace at any time.

With the other variants of the constructors, you can specify the namespace directly when you create the CIM client.

**Calling Methods of the Class CIMClient**

The class `CIMClient` implements the Specification for CIM Operations over HTTP. You can call the methods of the class with the generated instance of this class.

**Process CIM Objects**

CIM objects are units of data that are managed by the SLD server. They are CIM classes and CIM instances. The following sections give you an overview of how to create and process a CIM object.

For more information, see the JavaDoc package.

**Creating a New CIM Object**

To create a new CIM object, proceed as follows:

1. Create a local CIM object.
2. Process the new CIM object locally.
3. Transfer the new CIM object to the SLD server. The SLD server creates the CIM object.

**Creating a Local CIM Object**

SLD is a client-server application. For this reason, the client cannot create any new server objects directly by calling the constructors. Using the class methods of the class `com.sap.lcr.api.cim.CIMFactory`, you can create new CIM objects locally on the client side. You can process these local CIM objects and then transfer them to the server.

To create a new object of a CIM class, call up the class method `cimclass(String aClassname, String aSuperclassname)` of `CIMFactory`. The first parameter specifies the name of the new CIM class, whereas the second parameter represents the name of the superclass. Leave out the second parameter if the new class has no superclass.

To create a new instance of a CIM class, call up the class method `instance(String aClassname)` of `CIMFactory`. The parameter specifies the CIM class for which you are creating an instance.

For the class methods of `CIMFactory`, you can create local objects only. To transfer them to the SLD server, call up the corresponding instance methods of `com.sap.lcr.api.cimclient.CIMClient`.

**Processing CIM Objects Locally**

After you have created a new CIM object locally, you can set values for properties and qualifiers. Properties are attributes of a CIM object. The key properties are important for the identity of the CIM instances. You cannot change key properties after the CIM object has been saved on the server. Qualifiers are metainformation that characterizes CIM classes and properties.

An instance of `SAP_Product` has a number of properties such as `Name`, `Version`, and `WarrantyDuration`. The warranty duration has the qualifier `Units`, which specifies the unit of time. In this case, the unit is `Day`. This means that the warranty duration is specified in days.

Using the class methods `property()` and `qualifier()` of `CIMFactory`, you can create Java objects of the metaclasses `CIMProperty` and `CIMQualifier`. Like `CIMFactory`, these classes also come from the package `com.sap.lcr.api.cim`. You can set values for these metaobjects by using the extensive instance methods of both classes.

To add these properties and qualifiers to the CIM object mentioned above, call up the instance methods `addProperty()` and `addQualifier()` of the classes `CIMClass` or `CIMInstance`. If the values of the properties or qualifiers have already been set in the relevant CIM object, the method call overwrites the old values.

**Transferring the CIM Object to the Server**

To transfer the CIM object to the SLD server, call up the instance methods `createClass()` or `createInstance()` of the class `CIMClient`. By doing this, you save the CIM object on the server permanently.

The following program section creates a new CIM class called `SAP_SWProduct`. The superclass is called `SAP_Product`. The qualifier `DISPLAYNAME` of the new class also has the value `Software Product`. The code snippet is as follows:

```
// Create a new CIM class called SAP_SWProduct
CIMClass CIMClass = CIMFactory.cimclass("SAP_SWProduct", "SAP_Product");
// 设置DISPLAYNAME qualifier
CIMQualifier CIMQualifier = CIMFactory.qualifier("DISPLAYNAME", "Software Product");
// 添加DISPLAYNAME qualifier
CIMClass.addProperty("DISPLAYNAME", CIMQualifier);
// 创建一个新实例
CIMInstance CIMInstance = CIMFactory.instance("SAP_SWProduct");
// 将新实例添加到SAP_SWProduct类
CIMInstance.addProperty("NAME", "Example Product");
CIMInstance.addProperty("VERSION", "1.0");
CIMInstance.addProperty("WARRANTYDURATION", "3000 Days");
CIMInstance.addProperty("DISPLAYNAME", "Software Product");
```
private CIMClient client;
...
try {
    CIMClass cc = CIMFactory.cimclass("SAP_SWProduct", "SAP_Product");
    CIMQualifier cq =
        CIMFactory.qualifier("DISPLAYNAME", "string", "Software Product");
    cc.addQualifier(cq);
    client.createClass(cc);
} catch (Exception ex) {...}
finally {...}
...

Modifying an Existing CIM Object
To modify an existing CIM object, proceed as follows:

1. Get a local copy of the CIM object from the SLD server.
2. Make the modifications locally.
3. Transfer the modified CIM object to the SLD server.

The SLD server saves the modification permanently.

Getting a Local Copy of a CIM Object
SLD is a server-client application. To process an existing CIM object, you have to get a local copy of this object. The Java class CIMClient provides two instance methods that you can use to make copies of existing CIM classes or CIM instances from the SLD server. These instance methods are getCIMClass() and getInstance(). The first parameter of these two methods specifies the name of the CIM object. The return value is of the type CIMClass or CIMInstance.

According to the Specification for CIM Operations over HTTP (version 1.0), the second parameter of the boolean type specifies whether the properties and qualifiers that were transferred by the relevant superclasses are included. Usually they are not included. To make sure, set the value to false. However, doing this increases the amount of data on the communication channel. Read the JavaDoc, which contains detailed information.

Modifying CIM Objects Locally
The Java classes CIMClass and CIMInstance provide a large number of instance methods (getPropertyByName() and getQualifierByName(), for example) that return properties or qualifiers of the CIM objects. You can change the values of the properties or qualifiers.

Transferring the Modified CIM Object to the Server
To save the changes permanently, transfer the modified CIM object with the instance method modifyInstance() of the class CIMClient back to the server. The modification then takes effect.

The following program section shows the steps for modifying an existing CIM instance.
try {
    CIMInstancename cin;
    ...
    // Get a local copy of instance
    CIMInstance ci =
        client.getInstance(cin, false, true, false, null);
    // Get property
    CIMProperty prop = ci.getPropertyByName("Description");
    // Modify property
    prop.setValue("Changed on " + DateFormat.getInstance().format(new Date()));
    // Upload modified instance to server
    client.modifyInstance(cin, ci);
} catch (Exception ex) {...}

Finally {...}

The instance method `setProperty()` of the class `CIMClient` changes a
property of an existing CIM instance directly. This means that no local copy of
the CIM instance is required.

---

**List CIM Objects**

**CIM Classes**

To list all subclasses of any given class, call up the instance method `enumerateClasses()`
or `enumerateClassNames()` of the Java class `CIMClient`. The first parameter `ClassName`
specifies the class whose subclasses are to be listed. The second parameter `DeepInheritance` is of the `boolean`
type and determines whether only direct (`false`) or all (`true`) subclasses are returned.

**CIM Instances**

To list all existing instances of a particular CIM class, call up the instance methods `enumerateInstances()`.
The first parameter `ClassName` specifies the class whose
instances are to be listed.

**Associations and References**

The CIM standard uses association objects to model relations between CIM objects. An
association object contains references that point to the objects involved in the relation. These
are references. This procedure has the advantage that you can model relations between
objects, without directly affecting the definitions of the objects involved.

The specification for `CIM Operations over HTTP` defines two pairs of operations for listing
associations and references that are linked with a CIM object - the outbound object:

1. `associators()` and `references()`.

Both methods return an array of linked objects of the outbound object. The first method
`associators()` lists all objects that are linked with the outbound object through an
association. The second method `references()` returns all association objects that have a reference to the outbound object.

2. `associatorNames()` and `referenceNames()`.

Unlike the first method pair, this method pair returns the names of the associated objects instead of these objects themselves.

Handle Exceptions

The CIM operations trigger two runtime exceptions if an error occurs:

1. `com.sap.lcr.api.cimclient.CIMClientException`: Displays errors on the client side (failed connections, for example).
2. `com.sap.lcr.api.cimclient.LcrException`: Displays processing errors from the SLD server, for example, invalid parameters. The Specification for CIM Operations over HTTP defines 17 status codes that indicate different error types.
3. You can identify the numerical value and the symbolic names of the error by using the instance methods of `LcrException` `getStatusCode()` and `getStatusName()`.

When class methods of `CIMFactory` are executed, the exception of the type `com.sap.lcr.api.cim.ItemConstructionException` is triggered, which displays the errors that occurred when the CIM objects were generated.

Control the Logging Process

The API logs the process flow of client-server communication. To specify the level of logging, call the instance method `setTraceLevel()` of `CIMClient`, using the level of your choice. There are seven logging levels, which are defined in the Java interface `com.sap.lcr.api.cimclient.CIMConst`:

<table>
<thead>
<tr>
<th>Level</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIMConst.NONE</td>
<td>0</td>
</tr>
</tbody>
</table>
Namespace

SLD supports the namespace concept that implements namespaces as logical areas for different purposes. The default namespace for SLD is `sld/active`. You can determine the namespace that you want to use in the following ways:

- Specify the namespace when you create the CIM client.
- Change the namespace at any time after you have created the CIM client by using the instance method `setLocalNamespacePath()`.

To identify the current namespace, call the instance method `getLocalNamespacePath()` or `getLocalNamespacePathString()` of the CIM client.

SAP Proxy Layer

To simplify the programming and the handling of CIM objects, SAP introduced an SAP proxy layer. Using this approach, each CIM class of the CIM schema and the SAP extension schema generates a representation class, an access class, and a filter class in Java.

All generated Java classes are grouped in the package `com.sap.lcr.api.sapmodel`.

Representation Class

A representation class represents the corresponding CIM class on the client side. It provides access methods (Getter/Setter) for all attributes of the CIM class concerned. Using the corresponding access class, you can transfer instances from the server to the client side, and modify them. You can also create new CIM instances by adding them to the server.

Representation classes have the same names as the corresponding CIM classes.

The representation class of the CIM class `SAP_SoftwareComponent` has the same name.

Representation classes for abstract CIM classes are also abstract.

Access Class

Access classes are used for client-server communication. The name of the access class consists of the name of the corresponding CIM class with the suffix `Accessor`.
The access class of the CIM class *SAP_SoftwareComponent* is called *SAP_SoftwareComponentAccessor*.

The constructor of the access class has a parameter that represents the CIM client.

You have created a CIM client *aClient*. For an instance of the class *SAP_SoftwareComponentAccessor*, call the constructor *SAP_SoftwareComponentAccessor(aClient)*.

Using the listing methods of the access class, you can access all instances of the corresponding classes, as well as all objects (associations and references) that are linked to these instances. You can also delete instances from the server or add new instances to the server.

Abstract CIM classes do not have generated access classes.

**Filter Class**

To select CIM instances from a large number of CIM instances that return the listing methods of the access classes, define selection criteria using the filter classes. You can specify the value range for numerical attributes and a search string for attributes of the string type. You can construct search strings with the following special characters:

<table>
<thead>
<tr>
<th>Special Character</th>
<th>Meaning</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>^</td>
<td>Start of a string</td>
<td>^string</td>
</tr>
<tr>
<td>$</td>
<td>End of a string</td>
<td>string$</td>
</tr>
<tr>
<td>.</td>
<td>A single character</td>
<td>str.ng</td>
</tr>
<tr>
<td>*</td>
<td>Any sequence of characters</td>
<td>St*</td>
</tr>
<tr>
<td>[... ]</td>
<td>One of the specified characters</td>
<td>[abc]</td>
</tr>
<tr>
<td>\</td>
<td>Escape</td>
<td>*</td>
</tr>
</tbody>
</table>

Note: An escape character cancels out the special character meaning of the following character.

**ABAP Client for the SLD Service**

SAP provides a programming interface for ABAP (ABAP API), which enables ABAP applications to access services from the SLD server. The following graphic illustrates the software architecture of an ABAP client.
On the right-hand side of the graphic, there is an SAP Web Application Server on which the SLD service runs. On the left-hand side on the SAP Web Application Server, there is the ABAP program package SLD_API, which represents the programming interface for ABAP (ABAP API). On the same server there is a Java layer, which communicates with the SLD server by using the Java API. The ABAP API sends calls from the ABAP client to the Java layer. The Java layer forwards the calls to the SLD server by using HTTP. In the same way, the Java layer forwards replies from the SLD server to the ABAP API.

The ABAP API consists of three ABAP classes:

- **CL_SLD_ACCESSOR**: Represents the connection between the SLD server and a client.
- **CL_SLD_CIM_CLASS**: Represents a CIM class.
- **CL_SLD_CIM_INSTANCE**: Represents a CIM instance.

The ABAP API also contains some help classes:

- **CL_SLD_DATETIME**: Encapsulates the date and time format according to the CIM standard.
- **CL_SLD_FILTER**: Provides support for search criteria.
- **CL_SLD_UTIL**: Offers a collection of static methods.
- **CX_SLD_API_EXCEPTION**: Represents exception conditions during execution.

**Basis for Server Access**

**Establishing a Connection**

The ABAP class **CL_SLD_ACCESSOR** manages the connection between the client and the SLD server. You can use this class to call server functions. To create a connection to the SLD server, you have to specify the parameters **server** and **port** of the SLD service, and **user** and **password** for the authentication.
Setting a Trace Level

You can use the instance method `set_tracelevel` of the class `CL_SLD_ACCESSOR` to set the trace level to control the logging. The parameter specifies the trace level. To find out the values of the trace levels, see System Monitoring [Seite 451].

Checking the Connection

You can use the instance method `ping` of the class `CL_SLD_ACCESSOR` to test the state of a connection. The method runs, and, if an error occurs, triggers an exception called `cx_sld_api_exception`.

The following program section creates a connection to the specified SLD server, and sets the trace level to 1 (Warning). The program then tests whether the connection was successful.

```
report  sldapi_read_example.

data: my_sld_host     type string,
     my_sld_port     type i,
     my_namespace    type string,
     my_user         type string,
     my_password     type string,
     accessor        type ref to cl_sld_accessor,
     ex              type ref to cx_sld_api_exception.
...

* Set SLD access parameters:
my_sld_host  = 'iwdf7059'.
my_sld_port  = 55800.
my_namespace = 'sld/active'.
my_user      = 'STANLEY'.
my_password  = 'KUBRICK'.

* Create accessor object to SLD server:
create object accessor
exporting
  host           = my_sld_host
  port           = my_sld_port
  namespace      = my_namespace
  user           = my_user
  password       = my_password
.
* set trace level for java logging.
accessor->set_tracelevel( 1 ).

* Test the connection to the SLD server:
try.
  accessor->ping( ).
  catch cx_sld_api_exception into ex.
  value = ex->get_text( ).
  write: / 'PING failed!' color 6,
           at / value.
  uline.
  return.
endtry.
...
```
Read Access

General
The ABAP API offers a number of methods that enable read access to the SLD server. To get
information about existing CIM objects, call the following methods from the ABAP class
cl_sld_accessor:

- **get_class**: Gets the definition of a CIM class.
- **get_instance**: Gets information about a CIM instance.
- **get_property**: Returns the property value of a CIM instance.
- **get_property_array**: Returns a list of property values of a CIM instance.
- **enumerate_instances**: Lists existing instances of a CIM class.
- **enumerate_instnames**: Lists the names of existing instances of a CIM class.
- **associators**: Returns all associated CIM objects that are connected with the
  outbound object by an association.
- **associatornames**: Returns the names of all associated CIM objects that are
  connected with the outbound object by an association.
- **references**: Returns all association objects that contain a reference to the outbound
  object.
- **referencenames**: Returns the names of all association objects that contain a
  reference to the outbound object.

Listing Search Criteria
You can optionally use certain search criteria to call the methods **enumerate_instances**
and **enumerate_instnames**. To do so, when you call the methods you have to transfer an
object of the class **cl_sld_filter** as a parameter. You can use the filter class to determine the
value areas or values for numeric property values, and search templates for property values
of the type **String** in such a way that the list methods mentioned above return only the
instances that meet these search criteria.

The class **cl_sld_filter** provides the following instance methods, which you can use to set the
search criteria:

- **add_min_dt** and **add_max_dt**: Specify the lowest or the highest value of an attribute
  of the ABAP class **cl_sld_datetime**.
- **add_min_i** and **add_max_i**: Specify the lowest or the highest value of an attribute
  of the ABAP type **I**.
- **add_min_f** and **add_max_f**: Specify the lowest or the highest value of an attribute of
  the ABAP type **F**.
- **add_value**: Specifies the value of an attribute of the ABAP type **String**.
- **add_value_c**: Specifies the value of an attribute of the ABAP class **sldi_char**.
- **add_value_dt**: Specifies the value of an attribute of the ABAP class **cl_sld_datetime**.
- **add_value_i**: Specifies the value of an attribute of the ABAP type **I**.
- **add_value_f**: Specifies the value of an attribute of the ABAP type **F**.
- **add_pattern**: Specifies the search template of an attribute of the ABAP type **String**.
  All the search templates that are defined with the filter classes of the SAP proxy layer
  [Seite 470] are valid.
The following program section returns a list of the CIM instances of the CIM class `SAP_SoftwareFeature`, whose property value for `Name` contains the string R/3.

```abap
report  sldapi_read_example.

data: accessor        type ref to cl_sld_accessor,
ex              type ref to cx_sld_api_exception,
filter          type ref to cl_sld_filter,
t_irefs         type sld_t_instref,
instref         type ref to cl_sld_cim_instance,
...

try.
create object filter.
filter->add_pattern( pname = 'Name'
value = '*R/3*' ).
t_irefs =
  accessor->enumerate_instances( cname = 'SAP_SoftwareFeature'
filter = filter ).
...
loop at t_irefs into instref.
...
endloop.
catch cx_sld_api_exception into ex.
...
endtry.
```

### Write Access

#### Creating a New CIM Instance

To create a new CIM instance, proceed as follows:

1. Create a local object of the ABAP class `cl_sld_cim_instance`.
2. Set property values of the local object.
3. Call the instance method `create_instance` of the ABAP class `cl_sld_accessor` to transfer the new CIM instance to the SLD server.

The SLD server adds the new CIM instance.

The following program creates a CIM instance of the CIM class `SAP_BCSystem`. 

```abap
```
report sldapi_write_example.

data: accessor type ref to cl_sld_accessor,
      systref type ref to cl_sld_cim_instance,
      systhandle type string,
...

try.
  create object systref
    exporting
      classname = 'SAP_BCSysen'
      accessor = accessor.
  systref->set_property({
    pname = 'CreationClassName'
    value = 'SAP_BCSysen'
  }).
  ...  
systhandle = accessor->create_instance( iref = systref ).
catch cx_sld_api_exception into ex.
  ...
endtry.

Modifying an Existing CIM Instance

There are two ways of modifying an existing CIM instance:

1. Process the CIM instance locally and then transfer it to the server.
2. Change the property values of the CIM instance directly.

Processing a CIM Instance Locally

This method is suitable if you want to change multiple property values of a CIM instance. In this case, we recommend that you create a local copy from the server and then process it locally. You can then transfer it back to the SLD server. Proceed as follows:

1. Call the instance method get_instance of the ABAP class cl_sld_accessor to get a local copy of the CIM instance that you want to modify.
2. Call the instance method set_property or set_property_array of the ABAP class cl_sld_cim_instance to change the property values.
3. Call the instance method modify_instance of the ABAP class cl_sld_accessor to transfer the modified CIM instance to the server.
   
The changes to the CIM instance take effect on the server side.

Setting Property Values Directly

If you want to change only one or a few property values of a CIM instance, you can call the instance method set_property or set_property_array of the ABAP class cl_sld_accessor. These methods cause direct changes to the CIM instance on the server.

Removing a CIM Instance from the Server

To remove CIM instances from the server, call the instance method delete_instance or delete_instances of the ABAP class cl_sld_accessor.

- delete_instance: Deletes a particular CIM instance.
- delete_instances: Deletes all CIM instances of a particular CIM class that fulfil the specified filter criteria.
Exception Handling

If an error occurs, the method calls of the ABAP API trigger the exception of the ABAP class cx_sld_api_exception. The attribute field textid of the exception can be one of the constants that are defined in the class itself. These constants display various error conditions. You can view values of the constants in the class builder (SE24).

The following program section illustrates the exception handling after the method get_property has been called.

```abap
... try.
    value = accessor->get_property( ihandle = insthandle
                                  pname   = 'Slartibartfast' ).

    catch cx_sld_api_exception into ex.
      case ex->textid.
        when cx_sld_api_exception=>no_such_property.
          write: / 'Caught expected exception NO_SUCH_PROPERTY.'.
        when others.
          write: / 'Caught some unexpected exception!' color 6.
      endcase.
      value = ex->get_text( ).
      write: / value.
    uline.
  endtry.
```