Thomas Pohl and Markus Peter

Developing Enterprise Services for SAP®

Master modeling concepts including services metamodel, business objects, data integrity
Learn to develop enterprise services, using tools such as the ES Repository, ES Builder and ES Browser, proxy generation, publishing and testing
Explore service consumers in ABAP and Java, implementation with the Composition Environment, Eclipse, and .NET

Galileo Press
Bonn • Boston
## Contents at a Glance

### PART I  Modeling of Enterprise Services
1. Basic Modeling Principles for Enterprise Services .......... 21
2. Modeling A2X Services ................................................. 73
3. Process Integration and Integration Scenarios ............... 115

### PART II  Development of Enterprise Services
4. Enterprise Services Repository ...................................... 129
5. Technical Principles and Standards for Services ............. 145
6. Development of Enterprise Services in ABAP ............... 167
7. Development of Enterprise Services in Java ................. 233

### PART III  Sample Implementations for Enterprise Services
8. Sample Implementations of Enterprise Services in ABAP ........................................................................ 251
9. Sample Implementations for Service Consumers .......... 311

### Appendices
A. Literature ...................................................................... 383
B. The Authors .................................................................. 385
## Contents

### Part I  Development of Enterprise Services

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1.2 Preparations in the Enterprise Services Repository</td>
<td>75</td>
</tr>
<tr>
<td>2.1.3 Identification of the Business Objects</td>
<td>76</td>
</tr>
<tr>
<td>2.1.4 Layout of the Process Components and Deployment Units</td>
<td>79</td>
</tr>
<tr>
<td>2.1.5 Creation of the Architecture Model</td>
<td>81</td>
</tr>
<tr>
<td>2.1.6 Conclusion</td>
<td>90</td>
</tr>
<tr>
<td>2.2 Consistent Signatures and Their Significance</td>
<td>90</td>
</tr>
<tr>
<td>2.3 Modeling of Business Objects</td>
<td>92</td>
</tr>
<tr>
<td>2.3.1 Structure of the Integrated Business Object Model</td>
<td>93</td>
</tr>
<tr>
<td>2.3.2 Business Object Properties</td>
<td>97</td>
</tr>
<tr>
<td>2.3.3 SAP Global Data Types and Core Data Types</td>
<td>104</td>
</tr>
<tr>
<td>2.3.4 Completing the Business Object Modeling</td>
<td>106</td>
</tr>
<tr>
<td>2.4 Derivation of the Enterprise Services Signatures</td>
<td>107</td>
</tr>
<tr>
<td>2.4.1 Business Document and Business Document Object</td>
<td>107</td>
</tr>
<tr>
<td>2.4.2 Derivation of the Business Document Object Structure</td>
<td>109</td>
</tr>
<tr>
<td>2.4.3 Construction of the Business Document Object Structure</td>
<td>111</td>
</tr>
<tr>
<td>2.5 Summary</td>
<td>114</td>
</tr>
</tbody>
</table>

### Part II  Development of Enterprise Services

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1 Integration Scenario Models</td>
<td>116</td>
</tr>
<tr>
<td>3.1.1 Presenting Integration Scenario Models</td>
<td>117</td>
</tr>
<tr>
<td>3.1.2 Creating Integration Scenario Models</td>
<td>119</td>
</tr>
<tr>
<td>3.2 Process Components Interaction Models</td>
<td>122</td>
</tr>
<tr>
<td>3.2.1 A2A and B2B Services in Contrast to A2X Services</td>
<td>123</td>
</tr>
<tr>
<td>3.2.2 Creating Process Components Interaction Models</td>
<td>124</td>
</tr>
<tr>
<td>3.3 Integration Scenario Catalog</td>
<td>124</td>
</tr>
<tr>
<td>3.4 Summary</td>
<td>125</td>
</tr>
</tbody>
</table>

### PART III Development of Enterprise Services

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1 Structure of the Enterprise Services Repository Content</td>
<td>130</td>
</tr>
<tr>
<td>4.2 Overview of the Required Design Objects</td>
<td>131</td>
</tr>
<tr>
<td>4.2.1 Service Interfaces</td>
<td>131</td>
</tr>
<tr>
<td>4.2.2 Message Types</td>
<td>134</td>
</tr>
<tr>
<td>4.2.3 Data Types</td>
<td>134</td>
</tr>
<tr>
<td>4.2.4 Fault Message Types</td>
<td>139</td>
</tr>
<tr>
<td>4.2.5 Class Model</td>
<td>140</td>
</tr>
</tbody>
</table>
4.3 Identifying Industry-Specific Fields ........................................ 141
4.4 Namespaces in the Enterprise Services Repository ............. 141
4.5 Naming Conventions for Design Objects in the
Enterprise Services Repository ............................................. 142
4.6 Summary ........................................................................ 144

5 Technical Principles and Standards for Services .......... 145
5.1 Web Services .................................................................... 146
5.2 Extensible Markup Language ............................................. 147
  5.2.1 Structure of an XML Document .................................. 147
  5.2.2 Example of an XML Document ............................... 148
  5.2.3 Namespaces ........................................................... 149
  5.2.4 Processing Instruction ............................................ 149
  5.2.5 Syntactic and Semantic Correctness ........................ 149
5.3 XML Schema Definition .................................................... 150
  5.3.1 Primitive Data Types .............................................. 150
  5.3.2 Value Space, Lexical Space, Facet ........................... 150
  5.3.3 Derived Data Type ................................................. 151
  5.3.4 Simple Data Types ............................................... 151
  5.3.5 Complex Data Types ............................................ 153
5.4 Web Services Description Language .................................. 155
  5.4.1 Structure of a WSDL Document ............................. 155
  5.4.2 Definitions ............................................................. 157
  5.4.3 Data Types ............................................................. 157
  5.4.4 Message ............................................................... 157
  5.4.5 Port Type ............................................................... 158
  5.4.6 Binding ................................................................. 159
  5.4.7 Service ................................................................. 160
5.5 SOAP ................................................................................ 160
5.6 Additional W3C Standards ................................................ 161
  5.6.1 XML Parser ............................................................ 161
  5.6.2 Extensible Stylesheet Language for Transformation ... 162
  5.6.3 XML Path Language ............................................... 164
5.7 Summary .......................................................................... 165

6 Development of Enterprise Services in ABAP ............. 167
6.1 Background and Basic Properties ....................................... 167
  6.1.1 Decoupling ............................................................. 168
  6.1.2 Model-Based Development .................................... 168
  6.1.3 Transactional Behavior .......................................... 168
  6.1.4 Stateless and Stateful Communication .................... 168
  6.1.5 Unidirectional and Bidirectional Communication ... 169
Contents

6.1.6 Synchronous and Asynchronous Communication ................................................... 169
6.1.7 Inbound and Outbound Operations and Service Interfaces .............................................................. 170
6.1.8 Communication Patterns ...................................................................................... 171
6.1.9 Quality of Service ................................................................................................. 179
6.1.10 Reliable Messaging ............................................................................................. 180

6.2 Generating Proxies in the Backend System ............................................................... 182
6.2.1 ABAP Provider Proxy .......................................................................................... 183
6.2.2 ABAP Consumer Proxy ....................................................................................... 184

6.3 ABAP Proxy Runtime and Configuration ................................................................. 186
6.3.1 Overview of Message Processing at Runtime ...................................................... 187
6.3.2 Configuration of Provider Proxies ........................................................................ 190
6.3.3 Configuration of Consumer Proxies ....................................................................... 192
6.3.4 Configuration of the Message Processing via the Integration Server ...................... 194
6.3.5 Serialization and Deserialization .......................................................................... 195

6.4 Implementation of Inbound Service Interfaces ......................................................... 197
6.4.1 General Implementation Considerations ................................................................ 197
6.4.2 Basic Programming Model of Enterprise Services .............................................. 198
6.4.3 Implementation of Change Services ..................................................................... 202
6.4.4 Code Lists ............................................................................................................. 210
6.4.5 Reliable Messaging .............................................................................................. 214
6.4.6 Extension Concept ............................................................................................... 217
6.4.7 Error Handling ..................................................................................................... 218
6.4.8 Industry-Specific Extensions ................................................................................ 220
6.4.9 Mass Data Services ............................................................................................... 224

6.5 Testing Services in the Web Services Navigator .......................................................... 225
6.6 Evaluating Services in the Enterprise Services Workplace ........................................ 226

6.7 Publishing Services to the Services Registry ............................................................ 227
6.7.1 Structuring Services in the Services Registry ...................................................... 228
6.7.2 Consuming Web Services in the Services Registry .............................................. 230
6.7.3 Publishing Services to the Services Registry ....................................................... 231

6.8 Summary .................................................................................................................. 231

7 Development of Enterprise Services in Java ......................................................... 233

7.1 Inside-Out Implementation ......................................................................................... 234
7.2 Outside-In Implementation ......................................................................................... 234
7.2.1 Prerequisites ........................................................................................................ 235
7.2.2 Importing a WSDL Document ............................................................................. 235
7.2.3 Proxy Generation ................................................................................................ 239
7.2.4 Displaying the Generated Web Service Artifacts ................................................. 241

7.3 Configuration of Web Services at Design Time ....................................................... 242
7.3.1 Configuration of the Authentication Level ......................................................... 242
### Contents

- **7.3.2** Configuration of the Transport Guarantee ............... 243
- **7.3.3** Configuration of Web Services
  - Reliable Messaging (WS-RM) ........................................ 244
- **7.3.4** Configuration of Stateful Communication ............... 244
- **7.3.5** Implementation of the JavaBean Skeleton .............. 244
- **7.4** Configuration of Java Web Services with the
  SAP NetWeaver Administrator .......................................... 244
- **7.5** Publication of Services from SAP NetWeaver
  Administrator ..................................................................... 245
- **7.6** Testing Services in the Web Services Navigator .................. 247
- **7.7** Summary .......................................................................... 247

### PART III  Sample Implementations for Enterprise Services

<table>
<thead>
<tr>
<th>8</th>
<th>Sample Implementations of Enterprise Services in ABAP</th>
<th>251</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.1</td>
<td>Overview of the Implementation Project .................. 251</td>
<td></td>
</tr>
<tr>
<td>8.2</td>
<td>Preparations in Enterprise Services Repository .......... 253</td>
<td></td>
</tr>
<tr>
<td>8.3</td>
<td>Preparations in the ABAP System ............................ 256</td>
<td></td>
</tr>
<tr>
<td>8.4</td>
<td>Patterns for the Design of Service Implementations ...... 259</td>
<td></td>
</tr>
<tr>
<td>8.5</td>
<td>Reserving, Booking, and Canceling a Flight ............... 262</td>
<td></td>
</tr>
<tr>
<td>8.5.1</td>
<td>Specifications of Services .................................... 262</td>
<td></td>
</tr>
<tr>
<td>8.5.2</td>
<td>Implementing the Service Interface Proxy Classes ...... 265</td>
<td></td>
</tr>
<tr>
<td>8.5.3</td>
<td>Template for Service Implementation Classes ............ 267</td>
<td></td>
</tr>
</tbody>
</table>
| 8.5.4 | Completing the Create Flight Sales Order Service
  Implementation Class ..................................................... 269 |
| 8.5.5 | Helper Classes ..................................................... 272 |
| 8.5.6 | Completing the Confirm and Cancel Services .......... 274 |
| 8.6 | Service for Reading a Flight Sales Order .................... 278 |
| 8.6.1 | Specifying the Read Flight Sales Order Service .......... 278 |
| 8.6.2 | Read Flight Sales Order Service
  Implementation Class ..................................................... 279 |
| 8.7 | Input Help Services .................................................. 284 |
| 8.7.1 | Implementing the Find Flight Customer Service .......... 285 |
| 8.7.2 | Implementing the Find Flight Service ..................... 291 |
| 8.7.3 | Input Helps for the Core Data Types of the
  Code Type ................................................................... 296 |
| 8.8 | Service for Checking the Availability of a Flight .......... 299 |
| 8.9 | Changes to Flight Sales Orders .................................. 300 |
| 8.10 | Testing the Service Implementations ........................... 307 |
| 8.11 | Summary .......................................................................... 309 |
## 9 Sample Implementations for Service Consumers .... 311

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.1</td>
<td>Development of a Consumer Application in ABAP</td>
</tr>
<tr>
<td>9.1.1</td>
<td>Sample Application</td>
</tr>
<tr>
<td>9.1.2</td>
<td>Framework of the Reservation Report</td>
</tr>
<tr>
<td>9.1.3</td>
<td>Filling the List Boxes for the Airport and Booking Class</td>
</tr>
<tr>
<td>9.1.4</td>
<td>Determining the Flights</td>
</tr>
<tr>
<td>9.1.5</td>
<td>Interactive Display of the Flights with an ABAP List Viewer</td>
</tr>
<tr>
<td>9.1.6</td>
<td>Calling the Availability Check</td>
</tr>
<tr>
<td>9.1.7</td>
<td>Service Call for Reserving a Flight</td>
</tr>
<tr>
<td>9.1.8</td>
<td>Conclusion</td>
</tr>
</tbody>
</table>

| 9.2 | Application Development with SAP NetWeaver Composition Environment | 332 |
| 9.2.1 | SAP NetWeaver Visual Composer | 333 |
| 9.2.2 | Web Dynpro Java | 343 |

| 9.3 | Application Development with Eclipse | 356 |
| 9.3.1 | Eclipse | 357 |
| 9.3.2 | JSP Sample Application | 359 |
| 9.3.3 | Generating a Dynamic Web Project in the Eclipse IDE | 359 |
| 9.3.4 | Importing the WSDL Document | 360 |
| 9.3.5 | Creating the JavaServer Page | 362 |
| 9.3.6 | Editing the Deployment Descriptor | 363 |
| 9.3.7 | FlightDemoServlet Servlet | 364 |
| 9.3.8 | Implementing the FlightDemoBean JavaBean | 367 |
| 9.3.9 | Result | 370 |

| 9.4 | Application Development with Microsoft .NET | 371 |
| 9.4.1 | Generating the Consumer Proxy | 372 |
| 9.4.2 | Creating the Consumer Application | 374 |
| 9.4.3 | Result | 381 |

| 9.5 | Summary | 382 |

## Appendices ........................................... 383

<table>
<thead>
<tr>
<th>Letter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Literature</td>
</tr>
<tr>
<td>B</td>
<td>The Authors</td>
</tr>
</tbody>
</table>

Index .................................................. 387
Enterprise services are standardized in multiple ways. The model-specific and design-specific principles for a uniform definition of services and the technical standards for the message transfer have already been discussed. Furthermore, services have a common programming model to ensure a standardized behavior at runtime. This is illustrated in this chapter.

6 Development of Enterprise Services in ABAP

The previous chapters explained how to model enterprise services and specify them in a standardized format in Enterprise Services Repository (ES Repository). This chapter details the rules according to which the services are developed in ABAP.

The implementation is based on proxies that are generated from the metadata of the services. A common architecture ensures that the services behave in a uniform and consistent manner at runtime. This chapter analyzes this architecture in detail. First of all, however, it describes some of the basic properties of an enterprise service, which are essential to understanding the subsequent descriptions. Instead of the term “enterprise services,” this chapter also uses “Web services” or “services” depending on whether it is referring to the overall concept, the technical implementation, or the functionality itself.

To keep the description as simple and clear as possible, this chapter refers to the current release of SAP Business Suite or SAP NetWeaver Process Integration 7.1 EhP1, even though the functions deployed here have already been available in earlier releases.

6.1 Background and Basic Properties

SOA is different from conventional programming models for business applications in many respects. The differences are indicated in the development process of business applications but also in the way the applications communicate with each other at runtime and their integration with
Development of Enterprise Services in ABAP

This chapter discusses the special features of a service-based architecture and its benefits in more detail.

6.1.1 Decoupling

A reason for using enterprise services is the decoupling of the applications by means of intermediate services. Consumer and provider can communicate with each other without any restrictions on the platform on which they are implemented. For example, a Java EE-based business application can access services that are implemented in ABAP. This is enabled by the standardization of the technology that is used, which is at a very advanced level today. Consumer and provider communicate with each other via an open protocol (SOAP) and use standardized interface information only that can be published in a Services Registry. Therefore, technical decoupling allows reusing services on different platforms.

6.1.2 Model-Based Development

Metadata assumes a significant role in the development of enterprise services. It defines the contract between the consumer and the provider at the technical level. The consumer uses the metadata to determine the format of the messages and further information that it requires to call the service from its application. For the provider, the metadata is the default data that it has to implement. Consumer and provider can generate the framework or skeletons of the required development objects using the metadata.

6.1.3 Transactional Behavior

Enterprise services are required to always leave the database in a consistent status. Calling additional services to ensure data consistency should not be necessary. Instead, each service call transforms the database from one consistent status into another consistent status.

6.1.4 Stateless and Stateful Communication

Basically, enterprise service can be either stateless or stateful. In this context, stateful means that specific context information is stored in the provider over multiple subsequent calls for the current interaction with a consumer.

The context can be provided at several levels:
- At the level of the technical communication via a stateful protocol
- At the service level by combining subsequent calls of the same consumer in a session (e.g., by exchanging a session ID)
At the application level which, in turn, has to provide the corresponding mechanisms.

All enterprise services delivered in SAP Business Suite are stateless. The following sections also use stateless enterprise services without explicitly mentioning this every time.

### 6.1.5 Unidirectional and Bidirectional Communication

Services can communicate with each other in unidirectional and bidirectional ways. Unidirectional means that the initiator of the communication sends a message to a receiver. For the bidirectional communication, it is additionally required that the receiver of the message also sends a response message to the sender.

Typical scenarios for a unidirectional communication are messages of an application that indicate the result of a process. However, in most cases, a bidirectional communication is required because the initiator is interested in the result of its message.

### 6.1.6 Synchronous and Asynchronous Communication

The synchronous communication and the asynchronous communication are two generally different modes for exchanging information:

- **Synchronous communication**
  
  Synchronous communication is generally bidirectional because it requires a communication channel between the initiator of the message and the receiver through which the response message can also be directly sent to the initiator. You can compare the synchronous communication with a telephone call where the caller submits a request message to the receiver (e.g., “Book the flight with the number LH454 from Frankfurt to San Francisco on May 1, 2009”), and the receiver directly confirms or rejects this request. The initiator doesn’t proceed with processing until it has received a response from the receiver.

  This behavior, which characterizes the synchronous communication, is also referred to as *blocking* because it deploys critical system resources exclusively, such as database blocks or communication channels.

- **Asynchronous communication**
  
  At first glance, the asynchronous communication is unidirectional. However, this doesn’t mean that no response can be sent. You can compare it with a communication by email. In this case, the initiator sends an electronic message to the receiver. The receiver may send a
response later on but doesn’t have to. The major benefit of the asynchronous communication is that the initiator can continue processing without blocking system resources until it receives a response. Therefore, applications that are based on asynchronous communication feature a more efficient scaling than those that use synchronous communication. Generally, when the input sets increase, the resource requirements of an asynchronous application increase considerably less than the resource requirements of a synchronous application. Of course, this requires that the response message is not needed immediately. Another advantage over synchronous communication is that the receiver doesn’t have to be available when the initiator sends the message.

Everybody knows from real life that phone calls may not be answered, or lines may be busy. On the other hand, asynchronous communication sets considerably higher demands on the technical infrastructure, which has to correlate the response message with the request message and send it to the corresponding receiver. Additionally, special mechanisms are required to manage the errors that occur.

### 6.1.7 Inbound and Outbound Operations and Service Interfaces

An inbound operation is an operation of an inbound service interface. Analogous, an outbound operation is an operation of an outbound service interface. The proxy object that is generated from an inbound (outbound) service interface is also referred to as inbound (outbound) proxy.

An inbound operation receives messages from a sender, and an outbound operation sends messages to a receiver.

An outbound operation is used by a calling application for the following reasons:

- The system is supposed to send a request message to a provider. In this case, a service of the provider is called. The call is made from a consumer application in a consumer proxy. A confirmation is principally expected.

- Another application is to be informed about a process. In this case, a message is transferred to the other application. A confirmation is not expected.

- The service provider implements the inbound operation. It contains the functions that are provided to the consumers as services (in the sense of technical services). The functions are implemented in a provider proxy.
Figure 6.1 illustrates the relationship between the consumer application and the provider application as well as between the consumer proxy and the provider proxy.

![Figure 6.1 Relationship Between Consumer and Provider Proxy](image)

### 6.1.8 Communication Patterns

Analogous to the interpersonal communication where you distinguish between the different types of communication, such as monolog, dialog, and discussion, you can classify the communication between applications according to various criteria. The service-based communication differentiates between the following communication patterns, which are characterized from the perspective of the implementation here:

- **Request/confirmation pattern**
  This pattern is characterized by the fact that the calling program requests an activity from the provider (e.g., creating or changing a business object), and the provider confirms the activity after the execution.

- **Query/response pattern**
  Here, a query is sent to the provider (e.g., which business objects belong to my organizational unit?) and answered in a response message by the provider.

- **Notification pattern**
  A message is sent to a receiver. However, no response is expected. The calling program assumes that the receiver processes the message accordingly so that the business process can be completed consistently.

- **Information pattern**
  Like in the notification pattern, a message is sent to a receiver in this pattern; however, no expectations are linked to the processing of the message. The receiver can take note of the message anytime, and the further processing is completely left up to it.

In contrast to the first two patterns, the notification and information patterns involve a unidirectional communication, which can be modeled via
asynchronous outbound service interfaces. The request/confirmation pattern and query/response pattern are bidirectional and can be either synchronous or asynchronous.

The following sections describe these patterns from the development perspective and also explain the differences between synchronous and asynchronous communication.

**Synchronous Request/Confirmation Pattern**

The process of the request/confirmation pattern is defined as follows (see Figure 6.2):

1. The consumer application sends a request message to the provider application by calling the corresponding method of the outbound proxy.
2. This causes the corresponding method of the inbound proxy to be called on the provider side, which implements the desired business logic.
3. The result of the call (the return parameters) is immediately transferred to the calling application.
4. After the call has been executed, the application can proceed with processing.

![Sequence Diagram for the Synchronous Request/Confirmation Pattern](image)

*Figure 6.2 Sequence Diagram for the Synchronous Request/Confirmation Pattern*
In ES Repository, a synchronous request/confirmation pattern is mapped by an inbound service interface, which contains a synchronous operation with the following messages:

- Request
- Response
- Fault

Figure 6.3 shows an example of such a service interface.

Asynchronous Request/Confirmation Pattern

The asynchronous request/confirmation pattern differs from the synchronous pattern in that consumer and provider communicate asynchronously with each other. The benefit of the asynchronous communication is that the processing of the messages can be decoupled, which results in a higher throughput.

However, for the runtime, the problem is now to send the response message to the original calling program. Basically, there are two options for delivering the messages. The following discusses first the direct communication between the consumer and provider, which is also referred to as point-to-point communication (P2P) and then the communication via a broker.
Direct (P2P) Communication

The process of the asynchronous request/confirmation pattern for a direct communication between the consumer and provider is as follows (and in a sequence diagram in Figure 6.4):

1. The consumer sends a request message to the provider. For this purpose, it calls the corresponding method of the outbound proxy in its application.
2. This causes the corresponding method of the inbound proxy to be called on the provider side.
3. In the implementation of the inbound proxy, the desired business logic is executed.
4. After the business logic has been executed, an outbound proxy of the provider application is called.
5. The outbound proxy sends the confirmation message to the calling application.

Figure 6.4  Sequence Diagram for the Asynchronous Request/Confirmation Pattern (P2P)
The consumer receives the message in the inbound proxy provided for this purpose.

This communication requires that the provider remembers the address information of the consumer until it has delivered the response message to the consumer. This example also illustrates that inbound and outbound proxies can be called on both the consumer and the provider side.

**Communication via an Integration Broker**

Another communication option is that the consumer and provider application communicate with each other via an integration broker. The process is as follows (and is shown in the sequence diagram in Figure 6.5):

1. The consumer sends a request to the broker. For this purpose, it calls the corresponding method of the outbound proxy in its application.
2. The broker determines the desired provider and forwards the request to the corresponding provider.

![Sequence Diagram](image-url)

**Figure 6.5** Sequence Diagram for the Asynchronous Request/Confirmation Pattern with an Integration Broker
3. The provider calls the desired business logic.
4. The provider instances its outbound proxy for the response.
5. The outbound proxy sends the confirmation message to the broker.
6. The broker receives the confirmation message, determines the receiver, and forwards the message to the receiver.
7. The consumer receives the confirmation message in the inbound proxy provided for this purpose.

Service Interface in Enterprise Services Repository

The asynchronous request/confirmation pattern is mapped by two service interfaces in ES Repository. The inbound service interface contains an asynchronous operation for the request, and the corresponding outbound service interface contains an asynchronous operation for the confirmation message. The operation for the inbound service interface additionally contains a fault message.

Figure 6.6 shows an example of an outbound service interface, and Figure 6.7 shows an example of an inbound service interface. Together, they map the asynchronous request/confirmation pattern.

![Example of an Asynchronous Outbound Service Interface in ES Repository](image)
Notification Pattern

The process of the notification pattern is defined as follows:

1. The application triggers the sending of the message to a receiver.
2. The receiver receives and processes the message.
3. The receipt and processing of the message are required to complete the business process consistently.

The notification pattern is mapped by an outbound stateless service interface in ES Repository. It contains an asynchronous operation in the Request role. Figure 6.8 shows an example.
Information Pattern

The information pattern process is as follows:

1. The application triggers the sending of the message to a receiver.
2. The receiver receives the message and can process it.
3. In contrast to the notification pattern, the processing of the message on the receiver side is not required to complete the business process consistently. The optional receiver may trigger follow-up actions.

Analogous to the notification pattern, the information pattern is mapped by an outbound stateless service interface. It contains an asynchronous operation in the Request role. Figure 6.9 shows an example.

![Figure 6.9 Example of an Information Pattern](image)

TU&C/C Pattern (Tentative Update & Confirm/Compensation)

In business applications, a process is often first preliminarily executed before it is ultimately processed. An example of this is a reservation, which blocks specific resources from further access for a certain period of time (e.g., items from a warehouse) before the actual booking takes place. Between the reservation and the booking, the reservation can still be changed several times.

For the example of the flight sales order, the airline would be obliged to not assign the seat otherwise (for a certain period of time) in case of a res-
ervation. Within this period, you can still change the reservation several times. Only when the airline receives a confirmation from the sold-to party does the provider, that is, the airline, implement a hard booking. However, the sold-to party might also cancel the reservation. In this case, the airline would make the seat available to other sold-to parties again.

Technically, this is a protocol, which can be mapped by the TU&C/C pattern. The following steps are performed:

1. The consumer calls a service on the provider side via a synchronous tentative update operation. But this service doesn’t ultimately execute the process. Instead it marks the changes as preliminary. The changes are visible for other consumers; that is, no isolation in the sense of an SQL transaction isolation is given here.

2. During the process, this requirement can still be changed several times. The calling program uses a transaction ID to establish a reference to the process.

3. Finally, the calling program lets the service provider know whether the process is supposed to be ultimately executed or to be canceled. For this purpose, it calls an asynchronous confirm or compensation operation.

Prerequisites for this protocol are the guaranteed processing of messages at the technical level and the fulfillment of a specific contract between consumer and provider at the semantic level. The contract obligates the consumer to always send a confirm or compensation message at the end of the processing and the provider to process these messages accurately. The fulfillment of the contract must also be ensured in the case of technical problems (e.g., when the provider system fails).

In ES Repository, a TU&C/C pattern is mapped by a specific service interface that contains at least one synchronous operation and two asynchronous operations. Because service interfaces of this type are not SAP NetWeaver XI 3.0-compatible, this pattern is not yet used in SAP Business Suite.

6.1.9 Quality of Service

The quality of service (QoS) is a critical attribute of a communication service that describes the reliability of messaging. In the context of Web services, you distinguish between the different quality requirements described in the following:
Best Effort
Best Effort refers to a QoS that ensures that the accumulating messages are transferred in the best and fastest possible way. However, the network doesn’t guarantee that the messages are delivered.
Mail is delivered according to this principle, for example: The mailman does his best to deliver the letter. It is still possible that letters are delivered with delay or get lost.
Standard SOAP also doesn’t guarantee reliable messaging and doesn’t include mechanisms to secure the communication against lost messages or multiple deliveries of the same message.

At Most Once
At Most Once guarantees that the same message is delivered to the receiver at most once. It is not allowed to deliver the same message several times. It may happen that not all messages are delivered.

At Least Once
At Least Once guarantees that each message is delivered to the receiver at least once or that an error message is generated. It may happen that a message is delivered several times.

Exactly Once
Exactly Once guarantees that each message is delivered exactly once or that an error message is generated if this is impossible.

In Order
In Order guarantees that the messages are delivered in the sequence in which they were sent.

Exactly Once In Order
Exactly Once In Order is a combination from Exactly Once and In Order and refers to a QoS that ensures that the receiver receives the messages exactly once and in the sequence in which they were sent.

6.1.10 Reliable Messaging
For business applications, it is considerably important that messages are delivered reliably. In this context, the basic requirement is the delivery of messages with the Exactly Once QoS. Because SOAP doesn’t provide any information on the delivery guarantee, you need additional utilities or standards to achieve this goal.
For synchronous services, the application can achieve the Exactly Once quality level by using the *Idempotency Framework (IDP)*, which is a component of SAP NetWeaver. Section 6.4.5, Reliable Messaging, discusses the underlying programming model in more detail. In the case of asynchronous services, the procedure depends on whether the delivery is implemented via a broker (SAP NetWeaver PI Integration Server) or via P2P. These cases are described in greater detail in the following sections.

**Reliable Messaging When Using the PI Integration Server**

The SAP NetWeaver XI 3.0 protocol for exchanging messages, which is used when the SAP NetWeaver PI Integration Server is deployed, already contains the required mechanisms to deliver the messages at the Exactly Once quality level.

**Reliable Messaging for P2P Communication**

To also enable reliable messaging without using the infrastructure of SAP NetWeaver PI, SAP NetWeaver supports the open standard, *Web Services Reliable Messaging (WS-RM)*. WS-RM is a protocol that can be used to provide the Exactly Once or even Exactly Once In Order QoS. For this purpose, the WS-RM protocol uses sequences. Within a sequence, the messages are numbered in ascending order, starting with 1, so that the receiver can determine whether a message hasn’t been delivered and in which sequence the messages were sent.

Figure 6.10 shows a typical WS-RM scenario. The consumer (ENDPOINT A) first sends a request for the creation of a sequence to the provider (ENDPOINT B). The provider then confirms this request by returning an identifier for this sequence. Referring to this sequence, the consumer now sends three messages, which are numbered sequentially, starting with 1 (MESSAGENUMBER). The last message has the LASTMESSAGE attribute. After having received this message, the provider confirms the receipt of the messages with sequence numbers 1 and 3. Because the message with sequence number 2 has been lost, the consumer resends it. Then the sequence is complete.

WS-RM is supported by SAP NetWeaver at the protocol level. To support P2P interactions without using an integration server, SAP plans to have asynchronous services successively support this protocol in SAP Business Suite.
The following section now describes how you implement services in the backend system.

### 6.2 Generating Proxies in the Backend System

Whereas the design objects in ES Repository are platform-independent, proxy objects are platform-specific runtime objects that the system generates according to the requirements of the respective runtime environment (e.g., ABAP or Java). A prerequisite for the generation of the proxy objects is that the related design objects are modeled in ES Repository or are at least available as an external WSDL description. In the ABAP backend system, a package structure needs to be created that defines in which package the proxies are created.

Figure 6.11 shows an excerpt of the navigation tree in the Enterprise Services Browser in the ABAP development environment. In the detail screen, you can view the name of the generated proxy interface and the implemented provider proxy class for the `CostCentreLineItemBudgetMonitoringRuleByIdQueryResponse_In` service interface.
The following description is based on a scenario with a provider proxy and a consumer proxy. The provider proxy is generated from an inbound service interface. For the inbound service interface, you can create the corresponding outbound interface from which the consumer proxy is generated.

### 6.2.1 ABAP Provider Proxy

For the generation of proxies, start the Enterprise Services Browser using Transaction SPROXY.

1. Expand the nodes of the respective software component version (SWCV) and the namespace, and then select the relevant service interface.

2. Double-click or use the context menu to generate the necessary proxy objects. The system then requests additionally required data, such as the ABAP package and transport request to which the proxy objects are supposed to be added.
3. As a result, you obtain an ABAP interface and ABAP class. The ABAP class is also called a proxy class or implementing class and uses the ABAP interface. Furthermore, the system generates ABAP Data Dictionary objects (provided they don't exist yet) as proxies for the data and message type definitions that are used by the service interface.

**ABAP Interface**

The ABAP class of a service provider contains the implementation of the service operations. It uses an ABAP interface that contains a method for each operation that has been modeled in ES Repository. For compatibility reasons with the SAP NetWeaver XI 3.0 protocol, the service interfaces of SAP Business Suite currently exist of only one operation each. The application developers finally implement the methods.

A provider proxy is based on an inbound service interface and has to be generated in the ABAP backend component by which the service is provided. Here, you can specify a prefix that is included in the proposed names for the ABAP objects.

**Example**

For the synchronous, inbound service interface, `PurchaseOrderCreateRequestConfirmation_In`, the following objects were generated from the `http://sap.com/xi/APPL/SE/Global` namespace, which belongs to the ESA-ECC-SE 604 SWCV. `PUR` was entered as the prefix, and the names were shortened:

- The `_PUR_PURCHASEORDERCRTRC` interface with the `EXECUTE_SYNCHRONOUS` method that is implemented
- The implementing provider class, `CL_PUR_PURCHASEORDERCRTRC`
- The `ECC_PURCHASEORDERCRTRC` service definition

If you generate the proxies yourself, you can overwrite their default names. If the service interface consists of several operations, the generated ABAP interface, of course, contains the respective method for each operation. The name of the method corresponds to the name of the operation.

You can use the service definition to create a runtime configuration. Section 6.3, ABAP Proxy Runtime and Configuration, provides more information on the runtime and configuration.

**6.2.2 ABAP Consumer Proxy**

An application uses the ABAP consumer proxy to call, that is, to consume, a Web service. An ABAP consumer proxy contains object classes and is basically based on an outbound service interface. To consume a service
that is described by an inbound service interface, create the corresponding outbound interface in ES Repository — provided it doesn’t exist yet. Alternatively, you can also generate the outbound service interface from a WSDL document of the inbound service.

In contrast to provider proxies for which you must implement a class in the ABAP backend system for each, the application can directly use consumer proxies. You can generate ABAP consumer proxies in any ABAP system, because a WSDL description is the only requirement.

If the respective outbound service interface is available in ES Repository, the process of generating the consumer proxy is identical to the generation of the provider proxy using the Enterprise Services Browser, which you can call with Transaction SPROXY. If only (external) WSDL documents are available, you generate the consumer proxy via the Repository Browser, which you can access via Transaction SE80.

To configure the (outbound) consumer proxy, you also require a logical port. This is an SAP-specific concept for the configuration of the runtime properties of a consumer proxy. The logical port contains, for example, the URL address under which the service is supposed to be called. This aspect is described in more detail in Section 6.3, ABAP Proxy Runtime and Configuration.

Figure 6.12 illustrates the relationship between the consumer proxy, the provider proxy, and the corresponding service interfaces.
6.3 ABAP Proxy Runtime and Configuration

This section deals with the ABAP proxy runtime and its configuration. These concepts describe how you process Web services at runtime.

For the sake of clarity, the description is restricted to synchronous Web services; SAP NetWeaver Application Server ABAP (SAP NetWeaver AS ABAP) also supports asynchronous services. This, however, requires an infrastructure that enables a reliable delivery of messages, which is currently basically implemented via the infrastructure of SAP NetWeaver PI.

The communication via Web services is based on SOAP. Currently, SOAP is only supported by HTTP(S). SOAP requests are processed via the Internet Communication Framework (ICF). For this purpose, SAP NetWeaver AS ABAP uses HTTP in the ICF for the communication between consumer and provider.

SAP NetWeaver AS ABAP can be used as a provider for Web services and as a consumer of Web services. The ABAP proxy runtime supports Web services for which an integration server is used as well as P2P connections via SOAP. In both cases, a consumer proxy is required to send the message to the receiver or a provider proxy that implements the desired function.

By configuring an ABAP consumer, you can define whether the connection is a SOAP-based P2P connection or whether the message is supposed to be sent via the SAP NetWeaver XI protocol. These two options are illustrated in Figure 6.13.

![Diagram](image)

**Figure 6.13** Communication via the Integration Server with the SAP NetWeaver XI Protocol or Point-to-Point Communication via SOAP
Here, the following rules apply:

- The communication via an integration server can either be synchronous or asynchronous. In the synchronous case, Best Effort is the corresponding QoS; in the asynchronous case, this is Exactly Once or Exactly Once In Order.
- The P2P communication via SOAP is currently synchronous, too, but an asynchronous communication via WS-RM is feasible in the future, which supports Exactly Once In Order as the QoS.

### 6.3.1 Overview of Message Processing at Runtime

The following sections use the example of a synchronous Web service to illustrate what happens when a Web service is called. It is assumed that the service is called via HTTP.

#### Internet Communication Manager

The Internet Communication Manager (ICM) enables communication between the SAP NetWeaver AS and the outside world via the HTTP, HTTPS, and SMTP protocols. ICM manages HTTP requests and responses and provides services for further processing, which are briefly described in the following:

- To obtain an overview of the ICM load, you can log accesses from or to the Internet or intranet. You can export the log file to another file and then use external evaluation programs to analyze it.
- Incoming HTTP requests can be modified before they reach the application server. This includes the following operations:
  - Adding, deleting, and modifying HTTP header fields
  - Filtering requests
  - Redirecting requests to another page
  - Rewriting URLs

- The ICM server cache stores objects before they are sent to the client. If the object is later requested with a request again, the content can be read from the server cache if the expiry time hasn’t expired yet. This increases the processing performance of HTTP requests considerably.
- ICM is managed and monitored via profile parameters, the ICM Monitor (Transaction SMICM), or the web administration interface.

For further processing, the HTTP request is forwarded to ICF.
Internet Communication Framework

ICF is the layer between ICM, which sends or receives the HTTP requests, and the processing in the work process of the SAP NetWeaver AS. ICF has both server and client functionality. Incoming calls in the SAP system are forwarded to the HTTP request handler. An HTTP request handler is an ABAP objects class that implements the IF_HTTP_EXTENSION interface. The HTTP request handler is determined by means of the request URL at runtime.

To have the system call the HTTP request handler when a specific URL is entered in the browser, the handler needs to be integrated with an ICF service. Transaction SICF enables you to create and configure ICF services. ICF services are arranged in a hierarchical structure. In this hierarchy, you can derive the URL path for calling the ICF service from the path to the service.

Example

SAP NetWeaver AS ABAP runs on the saphost host at port 8080. The ping ICF service was created in the sap/bc node, and the CL_HTTP_MYPING handler was assigned to the ICF service. When you enter the http://saphost:8080/sap/bc/ping URL, the system calls the handle_request() method, which was implemented in the handler.

For Web services, SAP provides the sap/bc/srt/xip/sap ICF service. The Web services that are available in ABAP systems can be found under this node and also inherit the corresponding request handler.

Processing in the Web Service Enabling Layer

The Web Service Enabling Layer checks whether the message uses the SAP NetWeaver XI protocol or SOAP. Accordingly, the system transfers the message either to the Web service runtime or to the local SAP NetWeaver XI runtime. The further processing now takes place in the ABAP proxy framework.

Processing in the Application Service Enabling Layer

During the message processing, the system calls the proxy implementation. Section 6.4, Implementation of Inbound Service Interfaces, discusses the related individual steps.
The service implementation then calls the business logic. This call can be made via the method of an ABAP objects class, a BAPI, or an API, which the application provides for this purpose. Figure 6.14 displays an overview that illustrates the processing of the messages at runtime.

Figure 6.14  Processing of the Messages at Runtime
6.3.2 Configuration of Provider Proxies

Service endpoint

A service definition itself is no unit that can be called. To consume a Web service, you first have to create a runtime representation of the service definition, which is also referred to as service endpoint. The service endpoint contains the configuration settings for the Web service definition and is located on the provider system at a specific location, the so-called service endpoint URL. The consuming application uses this URL to call the configured Web service.

SOAMANAGER

To create service endpoints, you can use the SOA Management tool, which can be called via Transaction SOAMANAGER. The service endpoints allow for the following configuration settings:

- **Provider Security**
  You can implement the settings for the transport guarantee (e.g., without transport guarantee, HTTPS, signature and encryption, etc.) and for the authentication method (e.g., no authentication, HTTP authentication via the user ID/password, X.509 SSL client certificate, logon ticket, or message authentication with SAML 1.1).

- **Web Service Addressing**
  Here you can select the protocol for the Web service addressing.

- **Messaging**
  You can define the protocol for reliable messaging and the duration of the confirmation interval (in seconds) here. The confirmation interval is the period within which the provider must confirm to the consumer the receipt of a message.

- **Transport settings**
  In addition to the determined access URL, you can specify an alternative access URL. If the service is not locally available (e.g., behind a firewall), you must specify the alternative path.

- **Message attachments**
  You can define whether message attachments are supposed to be processed. In this case, several files can be sent with one message.

- **Operation specific**
  Here you can specify a SOAP action for an operation. The SOAP action is defined as a URI and transferred as an HTTP header if the Web service is called via HTTP.

Figure 6.15 displays the screen for the configuration settings.
You can assign multiple service endpoints with different configuration settings to a Web service definition. This enables you to provide identical Web service definitions with different configuration settings to consumers.

Services define groups of service endpoints. A service definition may include several services, which in turn may consist of several service endpoints. This relationship is shown in Figure 6.16.
WSDL document  You can generate a WSDL document for each service endpoint. In contrast to the port type WSDL, which doesn’t contain configuration information yet, this WSDL document already contains the binding information.

6.3.3 Configuration of Consumer Proxies

Logical ports  The configuration of consumer proxies is implemented via logical ports. Consequently, a logical port is created based on the requirements of the provider. A logical port refers to a service endpoint that is available under a unique location in the provider system. A logical port additionally contains a runtime configuration for accessing the service endpoint. Furthermore, the port also contains the logon data that is required for calling the service methods.

You can create multiple logical ports for each consumer proxy, but a logical port can only refer to one endpoint. You manage and configure consumer proxies via the SOA Management tool (Transaction SOAMANAGER).

Figure 6.17  Relationship Between the Logical Port and Service Endpoints

Figure 6.17 illustrates the relationship between logical ports and service endpoints. A service consumer establishes a connection by sending a call via a logical port. A logical port can send a call only to one service endpoint, but a service endpoint can be called via various logical ports.

The following options are available to maintain the logical ports:

- **Consumer security**
  Here you can implement settings for the transport security and authentication.

- **Web service addressing**
  You can specify the protocol for the Web service addressing.
Messaging
You can define the protocol for the secure data exchange and the time interval for the confirmation. The lifetime of a sequence is the time interval in seconds in which the sequence is active. 0 stands for endless. The inactivity timeout is the maximum time period for which the sequence is kept open without any confirmation.

Transport settings
Here you can implement the messaging settings. You should select Execute Local Call, if the consumer and provider are located on the same ABAP system. This has the result that no new logon is required for the service call. Maximum Wait Time WS Consumer defines the HTTP timeout, that is, how long the consumer waits for the response message.

Message attachments
You can define whether message attachments are allowed or not.

Operation specific
Here you can specify a SOAP action for an operation. The SOAP action is defined as a URI and generated as an HTTP header if the Web service is called via HTTP.

Configuration Scenarios
You can select configuration scenarios in the SOA Management tool under BUSINESS ADMINISTRATION by choosing the MASS CONFIGURATION entry. In a configuration scenario, you can group several Web service definitions and assign them to profiles. Figure 6.18 shows the initial screen for the maintenance of a mass configuration in the SOA Management tool.
6.3.4 Configuration of the Message Processing via the Integration Server

If the message processing is not supposed to be implemented from point to point but via an integration broker, you must configure some settings in the integration directory. This section provides a brief overview. For more information, refer to the corresponding SAP documentation of the integration directory.

You can implement the following settings in the integration directory:

- **Sender agreement**
  - Which adapter is used for the inbound processing?

- **Receiver determination**
  - Where is the message received?

- **Interface determination**
  - Which interface is used by the receiver?

Figure 6.19 provides an overview of the message processing on the integration server. The message processing is configured in the integration directory.

![Diagram of message processing](image_url)

**Figure 6.19** Configuration in the Integration Directory
6.3.5 Serialization and Deserialization

The data that an inbound proxy receives or an outbound proxy sends is converted in two steps. For incoming messages, the first step involves a technical conversion of the data of the message whose type description is provided in an XML Schema Definition (XSD) into the data structures of the ABAP runtime environment. This process is also called deserialization. It is automatically implemented via the simple transformations (ST), which have been generated by the ABAP proxy framework.

ST describes transformations from ABAP data into XML (serialization) and from XML into ABAP data (deserialization). To call ST in an ABAP program, you can use the CALL TRANSFORMATION language command, which also supports XSL transformations in addition to STs.

ST implicitly leads to a canonical serialization of ABAP data or a canonical deserialization to ABAP data before the actual transformation is executed. The SAP XSLT processor complies to a large extent with the specification for Version XSLT 1.0. These conversions are shown in Figure 6.20.

![Figure 6.20 Serialization and Deserialization](image)

In most cases, however, a technical conversion is not sufficient to format incoming messages, for example, in such a way that they can be transferred to the application, that is, to the existing internal programming interfaces. For example, it is possible that external ISO codes or units of measure must be converted into the SAP-specific format.

You can therefore also implement a conversion at the application level. In addition to the standard conversion of the application, the implementation of the Web service also provides Business Add-Ins (BAdIs), which enable further customer-specific conversions. Figure 6.21 illustrates the two-level conversion process.

![Figure 6.21 Two-Level Conversion](image)
Type systems

The type systems, XSD and ABAP, however, are not completely compatible. Consequently, losses may occur when an XSD data type is mapped to the corresponding ABAP data type (and vice versa). It is therefore important to document these situations and define rules of how different value ranges are supposed to be handled. The core data types can be used as the basis because they form manageable sets, and all other data types are based on them. The following example illustrates this process.

Example

The Amount core data type is defined in ES Repository as shown in Table 6.1.

<table>
<thead>
<tr>
<th>Element</th>
<th>Attribute</th>
<th>XSD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amount</td>
<td></td>
<td>xsd:decimal totalDigits=&quot;28&quot;; fractionDigits=&quot;6&quot;</td>
</tr>
<tr>
<td></td>
<td>currencyCode</td>
<td>xsd:token minLength=&quot;3&quot;; maxLength=&quot;3&quot; use=&quot;required&quot;</td>
</tr>
</tbody>
</table>

Table 6.1 Definition of the Amount Data Type in Enterprise Services Repository

The corresponding proxy data type, SAPPLSEF_AMOUNT, in the ABAP Dictionary is defined as shown in Table 6.2.

<table>
<thead>
<tr>
<th>Structure</th>
<th>Components</th>
<th>Component Type</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAPPLSEF_AMOUNT</td>
<td>CONTROLLER</td>
<td>PRXCTRLTAB</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CURRENCY_CODE</td>
<td>SAPPLSEF_CURRENCY_CODE</td>
<td>CHAR(3)</td>
</tr>
<tr>
<td></td>
<td>VALUE</td>
<td>SAPPLSEF_AMOUNT_CONTENT1</td>
<td>DEC(28,6)</td>
</tr>
</tbody>
</table>

Table 6.2 Definition of the Corresponding Proxy Data Type in the ABAP Dictionary

For a consistent mapping between the XSD and ABAP data type, the statistical methods, AMOUNT_INBOUND and AMOUNT_OUTBOUND, of the CL_GDT_CONVERSION class are available. Besides type-appropriate checks and formatting, the data is also checked against the Customizing settings in the SAP backend system.

The CL_GDT_CONVERSION class contains further methods that are available for the conversion of other core data types. The entire process of the conversion is outlined in Figure 6.22.
Now that you know the technical principles of Web services, the following sections discuss the implementation of service providers.

6.4 Implementation of Inbound Service Interfaces

For the implementation of enterprise services, the goal is a standardized programming model to ensure that the services behave consistently at the technical level. This section introduces the corresponding rules.

6.4.1 General Implementation Considerations

Enterprise services semantics is not only standardized, it is also implemented according to uniform guidelines. These guidelines also define the transaction behavior of the service.

Enterprise services are stateless and behave like atomic transactions. This restricts the usage of potential ABAP language elements, such as the usage of SET/GET parameters or the usage of the global SAP memory. They also define how the commit logic has to be implemented.

Numerous rules that have been specified for the BAPI implementation also apply to services. For example, no UI elements, such as dialog boxes, are allowed to be called during processing, and the business logic must be completed before a database update can be registered. Authorization checks have to be performed according to the ABAP authorization concept to protect data from unauthorized access via services. For transaction control reasons, language elements, such as CALL TRANSACTION or SUBMIT REPORT,
Index

.NET Framework, 371

A

A2A, 40
  Communication, 123
  PCIM, 122
  Service, 42, 115
A2X, 40
  Service, 41, 123
  Service design, 73
ABAP
  Consumer proxy, 184
  Package, 256
  Prefix, 314
  Proxy runtime and configuration, 186
  UI field, 316
  Unit test, 259
  Web Dynpro, 312
ABAP List Viewer (ALV), 312
ABAP proxy object
  Generation, 257
  Naming convention, 257
  Prefix, 257
Abstraction, 78
ActionCode, 255, 301
Adaptive RFC 2 Model, 345
Adaptive Web Service Model, 345
Aggregated data type, 136
Aggregation, 94
  Track relationship backwards, 109
Agility, 21
AirlineCode, 255
AirportCode, 255
ALE, 80
ALV, 312
  Object-oriented, 324
  Table control, 312
Amount, 254
Analysis
  Object-oriented, 30, 76
Annotation, 241, 242
Apache Tomcat, 357
Application
  Error messages, 321
Application integration, 22, 116
Application link enabling (ALE), 80
Application Service Enabling Layer, 188
Application-to-Application (A2A), 40
Application-to-Unknown (A2X), 40
Architectural model, 117
Architecture
  Service-oriented (SOA), 24
Assignment, 85, 124
Association, 48, 94
Asynchronous communication, 43
Asynchronous Request/Confirmation Pattern, 173
At Least Once, 180
At Most Once, 180
Atomicity requirement, 265
Attribute, 147
Authentication level, 242
Authentication method, 319
Automation
  Of standard processes, 40

B

B2B, 40
  Communication, 123
  PCIM, 123
  Service, 44, 115, 123
BAdI
  For inbound processing, 199
  For outbound processing, 200
BAPI, 36
Based-on relationship, 120
BasicBusinessDocumentMessageHeader, 216, 255
Best Effort, 180
Bidirectional, 169
Blocking, 169
Broker, 173
Bulk processing, 225
Bundle processing, 225
Business application, 168
Business Application Programming Interface (BAPI), 36
Business document, 107, 123
  Message Header, 108, 111
  Structure, 111
Business document object, 107, 123
  Derivation, 109
Business function, 222
Business function set, 222
Business language, 29
Business logic, 199
Business network transformation, 22
Business object, 29, 47, 74, 115, 143, 229, 237
  Business partner, 78
    Create, 83
  Flight, 76, 95, 100
    Flight Connection, 95, 98
    Flight Customer, 77, 95
    Flight Sales Order, 77, 96, 101
  Identification, 76
  Independence, 48
  Integrated model, 92, 93
  Integrated object model, 47
  Leading, 107
  Maintain attribute, 85
    Model, 115
    Modeling, 92
    Node, 48, 93
    Node model, 103
Business object map, 67, 79
  Create, 82
Business object node, 96
Business object type, 85
Business partner application, 123
Business process, 74
Business Process Modeling Notation, 116
Business process object, 60, 85
Business process platform, 21, 125
Business Server Pages, 312
Business-to-Business (B2B), 40

CALL TRANSFORMATION, 195
Cancel, 60
Cancellation
  Deletion, 60
Cardinality, 93, 94
Category, 132
CCTS, 36, 98, 137, 145
CDATA, 147
Core data type (CDT), 137
Certificate, 243
Change list, 121
Change operation, 60
Change service, 203
ChangeStateID, 255
Change state identifier (CSI), 205
Check Flight Availability, 299
Classification, 237, 246
  System, 228, 237, 246
CL_CENG_GEN_CODE_LIST_PROVIDER, 296
CL_FEH_REGISTRATION, 220
CL_GDT_CONVERSION, 196, 200
Client proxy
  Generation, 312
CL_PROXY_FAULT, 220
CL_SALV_TABLE, 324
CL_SOAP_COMMIT_ROLLBACK, 202
Code, 134
Code list, 210
Coherence, 34
Comment, 147
Communication, 117
  Asynchronous, 43, 169
  Bidirectional, 169
  Pattern, 171
  Stateful, 244
  Synchronous, 43, 169
  Unidirectional, 169
  Using enterprise service, 117
Communication channel template, 66
Communication pattern, 53, 143
  Information, 56
  Notification, 55
  Query/response, 55
  Request/Confirmation, 54
  Request/response, 56
Compensation operation, 179
Complete Transmission Indicator, 301
Component, 24
  Split, 25
Component controller, 343, 345, 352
Composite application, 21, 74, 228
Composite Application Framework, 333
Composition, 48, 94
Compound key, 100
Configuration, 186, 190
  Scenario, 193
Connector, 121
Consistency
  Of service signatures, 91
  Runtime behavior, 91
Consumer
  Security, 192
Consumer proxy
External name, 318
Generation, 313
Contains, 84, 96
Context category, 223
Context mapping, 349
Controller, 345
CONTROLLER, 209, 261, 304, 305
Conversion
FlightConnectionID, 270
Coordinated Universal Time (UTC), 283
Core Component Technical Specification (CCTS), 145
Core data type, 35, 134, 137
Correct
Semantically correct, 149
Coupling
Loose, 25
Credentials, 380
CRUD-Operation, 58
CSI, 205
Calculation, 276
Check, 275
Custom controller, 345
CustomerID, 254

D
Database
Cursor, 288, 290
Database block, 169
Data consistency, 168
Data integrity, 28
Data link, 349
Data modeler, 349
Data type, 131, 134
Aggregated, 136
Amount core, 104
Code, 105
Complex, 153
Copy, 255
Core, 35, 104, 134, 137
Derivation rule, 151
Derived, 151
Facet, 150
Freely modeled, 136
Global, 35, 138, 233, 253
Identifier, 105
Indicator, 105
Lexical, 150
Maximum length, 255
Measure, 105
Minimum length, 256
Primitive, 150
Quantity, 105
Simple, 151
Specification, 255
Text, 105
Value space, 150
Date, 254
DateTime
Number format, 281
Declaration
Create Flight Sales Order, 267
Decoupling, 168
Delete
Logically, 302
Deployment descriptor, 360
Deployment unit, 52, 115, 116, 229, 237
Create, 83
Flight Sales Order Processing, 81
Foundation layer, 81
Design by contract, 25
Design object, 129, 182
Interface object, 66
Modeling, 66
Process component, 66
Process integration, 66
Specification, 66
Destination, 355
Destination template, 337
Management, 337
Document Object Model (DOM), 161
Document Style, 159
Document Type Definition (DTD), 149
doGet, 364
DOM, 161
Object, 161
doPost, 364
DTD, 149
Duration, 254

E
Eclipse, 357
EJB, 234
Model, 346
Project, 234, 235
Session bean, 234
Stateful, 241
Stateless, 241
Element, 147
Endpoint, 307, 308
End-to-end business process, 122
Index

Enhancement package, 227
Enhancement spot, 217
Enterprise application project, 234, 235
Enterprise JavaBean (EJB), 234
Enterprise model
   Harmonized, 29
Enterprise service, 28, 31
   ABAP, 167
   Category, 115
   Layout, 28, 32
   Metamodel, 44
   Programming model, 198
   Signature, 33
   Stateless, 168
Enterprise Services Browser, 182, 183
Enterprise Services Builder, 64, 65, 76, 129
   Start, 81
Enterprise services bundle, 227
Enterprise Services Registry, 236
Enterprise Services Repository, 32, 63, 75, 81, 129, 176, 182, 236
   Content, 129, 130
   Folder, 76, 253
   Implementation, 251
   Namespace, 253
Enterprise Services Workplace, 226
Enterprise SOA, 27
Entity relationship modeling, 94
Enumeration, 211
Error handling, 200, 218
Error message, 219
ESM ERP, 69
ESM INTEGRATION, 69, 120
ES Repository content, 68
EverybodyWins, 203
Exactly Once, 180, 214
Exactly Once In Order, 180
Exception, 218
   Technical, 323
Executable GUI Language (XGL), 341
Export conversion, 200
Extensible Application Markup Language (XAML), 371
Extensible Markup Language (XML), 146
Extensible Stylesheet Language for Transformation <Pfeil>R<Norm, 162
Extension, 199
   Concept, 217
   Industry-specific, 220

F

Facet, 136
Fault message type, 51, 131, 139, 218
FEH, 200, 220
Finalize method, 201
FirstOneWins, 204
Fixed value, 212
FlightConnectionID, 255
FlightID, 255
FlightSalesCounterID, 255
FlightSeatClassCode, 255
Folder
   In the Enterprise Services Repository, 76
Formatting
   Of values, 36
Forward Error Handling (FEH), 200
Freely modeled data type, 136

G

Global data type (GDT), 35, 138
Generalization, 78
Generic Modeling Language (GML), 341
Global data type, 35, 138, 233, 253
GML, 341
Governance process, 129

H

Harmonized enterprise model, 29
Hash algorithm, 277
Hash value, 277
Helper class
   ZMPTP_CL_PROXY_IMPL_HELPER, 269, 272
Hit list
   Browse, 284, 285, 288, 292
   Limitation, 284, 285, 288
Homonym
   Avoidance, 49
HTTP, 186
   Request handler, 188

I

ICF, 186, 187, 188
ICM, 187
Idempotency, 215
   framework, 180, 260
   requirement, 264
Identification
Change-relevant field, 304
IDP, 180, 260
Implementation
   ABAP system, 251
   Cancel Flight Sales Order, 274
   Check service, 260
   Confirm Flight Sales Order, 274
   Create Flight Sales Order, 265, 269
   Enterprise Services Repository, 251
   Find Flight, 291
   Find Flight Customer, 285
   Read Flight Sales Order, 279
   Update Flight Sales Order, 302
Implicit connection
   Generate, 84
Import conversion, 199
Indicator, 254
Industry classification, 141
   Industry classification context, 224
Information pattern, 178
In Order, 180
Inside-Out, 233
Integrated business object model, 92
Integration
   Business partner system, 22
   SAP and non-SAP applications, 23
Integration broker, 175
Integration Builder, 64
Integration object, 130
Integration Repository, 52
Integration scenario, 115, 117
   Catalog, 124
   Create a model, 119
   Model, 116, 117
Integrity condition
   Check, 269
Interaction type, 117
Interface determination, 194
Interface object
   Use, 75
Interface pattern, 58, 132
   Action, 58
   Manage, 58, 60
   Notification, 62
   Request/confirmation, 61
Internet Communication Framework (ICF), 186
Internet Communication Manager (ICM), 187
Interoperability, 145, 357
   Semantic, 26
   Syntactic, 26
Technical, 26

J
Java, 233
   Artifact, 239
JavaBean, 359
   Model, 346
Java connector, 37
JavaServer Pages (JSP), 357
JAX-RPC, 358
JAX-WS, 241, 358
JSP, 357

K
Key
   Compound, 100

L
Label, 351
LastOneWins, 203
Lifecycle status, 229, 237
List
   Handling, 301
List box, 316
Literal, 151
Log, 112, 255
   Element, 271
Logically delete, 302
Loose coupling, 25
LPCONFIG, 318

M
Maintenance view
   V_CODE_REGISTRY, 297
Mapping, 124
Mapping program, 66
Mass Configuration, 193
Mass data service, 224
Master data, 80
   Distribution, 80
Master data object, 60, 85
maxOccurs, 153
Message attachment, 190
Message category, 53
   Confirmation, 55
   Information, 56
   Notification, 55
   Query, 55
   Request, 55

391
Index

Response, 56
Message Header, 108, 111
Message interface, 52
Message transformation, 124
Message type, 50, 88, 131, 134
Definition, 253
Messaging
Reliable, 180
Metadata, 168
Method
DO_CONVERT_REQUEST, 270
DO_CONVERT_RESPONSE, 271
DO_PERFORM_BUS_LOGIC, 271
DO_VALIDATE, 269
EXECUTE, 265
Metro, 358
minOccurs, 153
Model, 345
Integration scenario, 116
Model-based development, 168
Model binding, 349
Modeling
Enterprise service, 21
Object-oriented, 94
Modeling name, 57
Model type, 67
Model view controller, 359

N
Namespace, 131
Package Mapping, 239
Naming convention
Message type, 57
Navigation icon, 70, 122, 125
Navigation link, 344
Net data rule, 300
Next practice business process, 115
Node data type, 103
Notification
Communication pattern, 55
Interface pattern, 61
Notification pattern, 177
NumberValue, 254
NWDI, 343

O
Object class, 99
Object-oriented analysis, 30, 76
Object-oriented modeling, 94
Open source, 357
Operation, 133
Change versus update, 60
Inbound, 170
Outbound, 170
Operation-specific, 190
Optimistic locking, 204
Outside-In, 233

P
P2P
Communication, 173
Package, 364
Pattern, 171
Information, 171
Notification, 171
Query/response, 171
Request/confirmation, 171
Tentative Update & Confirm/Compensation (TU&C/C), 178
Payload protocol, 208
PCIM, 117, 122
PIC, 145
PI (SAP NetWeaver Process Integration), 64
Placeholder, 118, 121
Platform independence, 25
Plug, 344
Plug-in, 357
Point-to-Point (P2P), 173
Port
Logical, 185, 192, 318
Prepare method, 200
Process component, 29, 45, 74, 115, 226, 229, 237
Architectural model, 66
Business Partner Data Processing, 79, 89
Create, 83
Create model, 85
Flight Data Processing, 79, 89
Flight Sales Order Processing, 79, 87
Identification, 30
Layout, 46
Model, 85
Partner, 118
Third party, 118
Type, 118
Process component model (SAP)
ProComp model, 67
Process components interaction models (PCIM), 117

392
<table>
<thead>
<tr>
<th>Term</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process Composer</td>
<td>333</td>
</tr>
<tr>
<td>Process design class</td>
<td>30, 48</td>
</tr>
<tr>
<td>Processing Condition</td>
<td>112</td>
</tr>
<tr>
<td>Processing instruction</td>
<td>147</td>
</tr>
<tr>
<td>Process integration</td>
<td>115</td>
</tr>
<tr>
<td><strong>Scenario</strong></td>
<td>66</td>
</tr>
<tr>
<td>Process Integration Content Council (PIC)</td>
<td>145</td>
</tr>
<tr>
<td>Process integrity</td>
<td>28</td>
</tr>
<tr>
<td>Process orientation</td>
<td>115</td>
</tr>
<tr>
<td>Property</td>
<td>99</td>
</tr>
<tr>
<td>Proxy</td>
<td></td>
</tr>
<tr>
<td><em>Inbound</em>, 170</td>
<td></td>
</tr>
<tr>
<td><em>Outbound</em>, 170</td>
<td></td>
</tr>
<tr>
<td>PRXCTRL</td>
<td>209</td>
</tr>
<tr>
<td>Publication</td>
<td>231</td>
</tr>
<tr>
<td>Publication rule</td>
<td>245</td>
</tr>
<tr>
<td>PurchaseOrderCreate</td>
<td></td>
</tr>
<tr>
<td>RequestConfirmation_In</td>
<td>155</td>
</tr>
<tr>
<td><strong>Q</strong></td>
<td></td>
</tr>
<tr>
<td>Quality</td>
<td>32</td>
</tr>
<tr>
<td>Quality of service</td>
<td>179, 187</td>
</tr>
<tr>
<td>QueryCodeList</td>
<td>211, 296</td>
</tr>
<tr>
<td><em>Metadata</em>, 297</td>
<td></td>
</tr>
<tr>
<td><em>Provider class</em>, 296</td>
<td></td>
</tr>
<tr>
<td>QueryHitsMaximumNumberValue</td>
<td>285</td>
</tr>
<tr>
<td><strong>R</strong></td>
<td></td>
</tr>
<tr>
<td>Receiver</td>
<td>42, 123</td>
</tr>
<tr>
<td>Receiver determination</td>
<td>194</td>
</tr>
<tr>
<td>Registry service</td>
<td>147</td>
</tr>
<tr>
<td>Relationship</td>
<td>93</td>
</tr>
<tr>
<td><em>Contains</em>, 84, 96</td>
<td></td>
</tr>
<tr>
<td><em>Type</em>, 94</td>
<td></td>
</tr>
<tr>
<td>Reliable messaging</td>
<td>181</td>
</tr>
<tr>
<td>Remote Function Call (RFC)</td>
<td>36</td>
</tr>
<tr>
<td>Repository namespace</td>
<td>141</td>
</tr>
<tr>
<td>Representation</td>
<td>99</td>
</tr>
<tr>
<td>Representation term</td>
<td>134, 210</td>
</tr>
<tr>
<td>Request/confirmation</td>
<td></td>
</tr>
<tr>
<td><em>Communication pattern</em>, 54</td>
<td></td>
</tr>
<tr>
<td>Request/confirmation pattern</td>
<td></td>
</tr>
<tr>
<td><em>Asynchronous</em>, 173</td>
<td></td>
</tr>
<tr>
<td><em>Synchronous</em>, 172, 200</td>
<td></td>
</tr>
<tr>
<td>RequestDispatcher</td>
<td>359</td>
</tr>
<tr>
<td>Reusability</td>
<td>168</td>
</tr>
<tr>
<td>Reuse</td>
<td>22</td>
</tr>
<tr>
<td>RFC, 36</td>
<td></td>
</tr>
<tr>
<td><em>Library</em>, 37</td>
<td></td>
</tr>
<tr>
<td>Root element</td>
<td>148</td>
</tr>
<tr>
<td>Root node</td>
<td>95</td>
</tr>
<tr>
<td>rpc style</td>
<td>159</td>
</tr>
<tr>
<td>Runtime governance</td>
<td>29</td>
</tr>
</tbody>
</table>

**S**

- **SAM 1.1**, 190
- Sample application case, 74
- SAP Business Object Model, 96
- SAP Business Object Node, 103
- SAP Business Suite, 21
- SAP Data Type Model, 103
- SAP entity map, 67, 82
- SAP ERP
  - Foundation, 52
  - Module, 30
  - Without HCM, 53
- SAP ERP HCM, 53
- SAP Exchange Infrastructure (SAP)
- NetWeaver Process Integration, 64
- SAPGLOBAL, 69, 71
- SAPGLOBAL 2.0, 130, 224, 253
- SAP global data type, 104
  - Aggregated, 104
  - Basic, 104
  - Model, 106
- SAPGLOBAL MODEL, 69, 106
- SAP Integration Scenario Model, 121
- SAP Integration Server, 117
- SAPLink, 252
- SAP List Viewer (ALV), 312
- SAP logon ticket, 319
- SAP NetWeaver Administrator, 244
- SAP NetWeaver Application Server
  - Java, 233
- SAP NetWeaver Composition
- Environment, 64, 332
  - *Developer Edition*, 333
- SAP NetWeaver Developer Studio, 233
- SAP NetWeaver Development Infrastructure (NWDI), 343
- SAP NetWeaver Portal, 343
- SAP NetWeaver Process Integration, 23, 52, 64, 117, 186
  - Metamodel, 52
- PI Integration Server, 181
- SAP Exchange Infrastructure, 117
- XI content, 68
- SAP NetWeaver Visual Composer, 41, 333
Index

SAP NetWeaver XI content, 68
SAP ProComp, 86
   Interaction Model, 124
   Model, 67
SAP Scenario Catalog, 124
SAP SERM, 94
SAP Solution Map, 227
SAX, 161
Scaling, 170
SEI, 239
Semantically correct, 149
Semantic interoperability, 26
Sender, 42, 123
Sender agreement, 194
Separation
   Interface and implementation, 25
Sequence, 181
Serialization
   Deserialization, 195
Service, 24
   At advanced level, 34
   At entry level, 34
   Cancel Flight Sales Order, 86
   Category, 40
   Check Flight Availability, 89
   Confirm Flight Sales Order, 87
   Contract, 32
   Create Flight Sales Order, 86
   Endpoint, 190
   Find Flight, 89
   Find Flight Customer, 89
   Idempotent, 215, 260, 265, 266, 329
   Implementation class, 259
   Lifecycle, 29
   Mass data service, 224
   Operation, 50
   Pattern, 53
   QueryCodeList, 316
   Read Flight Sales Order, 86
   Signature, 115
   Synchronous, 307
   Update Flight Sales Order, 86
Service call
   Check Flight Availability, 327
   Create Flight Sales Order, 328
   Find Flight, 320
   Result, 321
Service consumer, 24, 123
Service endpoint interface (SEI), 239
Service endpoint URL, 190
Service interface, 51, 131, 229
   Abstract, 132
   Definition, 253
   Difference, 51
   Inbound, 43, 132, 170
   Maintain attribute, 88
   Model, 86
   Outbound, 43, 132, 170
   Stateful, 132
   Stateless, 132
   Stateless (XI 3.0-compatible), 132
   TU&C/C, 133
Service operation, 74, 229
   Attribute, 88
   Model, 86
Service-oriented architecture (SOA), 24
Service provider, 24, 123
Service signature, 90
   Derivation, 107
Services Registry, 26, 64, 227
Servlet, 357, 359
Session-afflicted, 38
SICF, 188
Simple API for XML (SAX), 161
Simple transformations, 195
Skeleton, 168, 364
SLD, 75, 120, 130
SOA, 24, 167
   Architectural pattern, 27
   By Design, 30, 80
   By Evolution, 30, 47, 52, 73, 80
   Enterprise SOA, 27
   Goal, 21
   Governance, 29
SOAMANAGER, 190, 307, 372
   Logical port, 318
soap\n   body, 159
SOAP, 146, 147, 160, 168, 186
Software catalog
   Entry, 120
Software component, 130
   SAP GLOBAL MODEL, 106
Software component version, 130, 229, 237
   Property, 75
Software component version (SWCV), 68
Source
   Code, 298
   Fixed values for domains, 298
   IMG table, 298
Specialization, 78
Specification
  Create Flight Sales Order, 262
  Find Flight, 291
  Find Flight Customer, 284
  Read Flight Sales Order, 278
  Update Flight Sales Order, 300
SPROXY, 183, 257
Service test, 308
Stability, 32
Standard element
  ProcessingCondition, 284
  StandardMessageFault, 218, 219
Starting point, 288
Statelessness, 290
Strategy
  FirstOne Wins, 300
  FirstOneWins, 264
Supplementary component, 104
  Removal, 256
Switch, 222
Switch Framework, 221
Synchronous communication, 43
Synchronous request/confirmation pattern, 172
Synchronous service, 307
Synonym
  Avoidance, 49
Syntactic interoperability, 26
System Landscape Directory (SLD), 130
T
targetNamespace, 157
Technical interoperability, 26
Template, 162
Test
  Synchronous service, 307
Time, 254
  Coordinated, 283
Transactional behavior, 168
Transport guarantee, 243
Transport setting, 190, 193
Type system, 196
U
UDDI, 26, 146, 228
UML
  Activity diagram, 124
  Class chart, 94
  Sequence diagram, 124
UN/CEFACT, 35
Unicode, 157
Unidirectional, 169
United Nations Center for Trade Facilitation and Electronic Bus, 35
Universal Description, Discovery and Integration (UDDI), 26
UnlimitedQueryHitsIndicator, 285
Update
  Local, 199
Update operation, 61
Update service, 204
Use
  Interface object, 75
UTC, 283
UTF-8, 148, 157
V
Value table, 212
View, 344
Visual Studio, 371
W
W3C, 145
Web application, 360
Web Dynpro, 41, 333, 343
  explorer, 347
  Web Dynpro component, 343, 348
  Web Dynpro development component, 346
  Web Dynpro Flex, 342
  Web Dynpro HTML, 342
Web project
  Dynamic, 359
Web service, 25, 146
  Addressing, 192
Web Service Creation wizard, 239
Web Service Enabling Layer, 188
Web Services Description Language (WSDL), 25
Web Services Navigator, 247, 307
Web Services Reliable Messaging (WS-RM), 244
web.xml, 363
Window, 343, 345
Windows Presentation Foundation, 371
World Wide Web Consortium (W3C), 145
WSADMIN, 307
WSCONFIG, 307
wsdl
  binding, 155
  definitions, 155
  message, 155
  port, 155
  portType, 155
  service, 155
  types, 155
WSDL, 25, 45, 146, 155
  Document, 192
  Import wizard, 235
WS Navigator, 214, 225
WS-RM, 181, 244

X

X.509
  SSL client certificate, 190
XAML, 371
XGL, 341
XI protocol, 186, 215
XI (SAP NetWeaver Process Integration), 64
XML, 146, 147
  Message, 50
  Namespace, 130, 141
XML document
  Valid, 150
  Well-formed, 149

XML handling
  Extended, 208, 303, 304
  xmlns, 149
XML parser, 161
XML Path Language (XPath), 164
XML schema definition (XSD), 149
XPath, 164
xsd
  all, 154
  choice, 154
  extension, 154
  list, 153
  normalizedString, 151
  restriction, 151
  sequence, 153
  string, 151
  token, 151
  union, 153
XSD, 149, 195
  Editor, 134
xsl
  stylesheet, 162
  template, 162
  transform, 162
XSLT, 162
  Processor, 163