BC416

ABAP Web Services

SAP NetWeaver

Date _____

Training Center _____

Instructors _____

_____

Education Website _____

Internal Use SAP Partner Only

Participant Handbook

Course Version: 2006 Q3
Course Duration: 2 Day(s)
Material Number: 50085035

An SAP course - use it to learn, reference it for work

Copyright

Trademarks

Disclaimer

Internal Use SAP Partner Only

About This Handbook

This handbook is intended to complement the instructor-led presentation of this course, and serve as a source of reference. It is not suitable for self-study.

Typographic Conventions

American English is the standard used in this handbook. The following typographic conventions are also used.

Type Style	Description
Example text	Words or characters that appear on the screen. These include field names, screen titles, pushbuttons as well as menu names, paths, and options. Also used for cross-references to other documentation both internal (in this documentation) and external (in other locations, such as SAPNet).
Example text	Emphasized words or phrases in body text, titles of graphics, and tables
EXAMPLE TEXT	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.
`Example text`	Screen output. This includes file and directory names and their paths, messages, names of variables and parameters, and passages of the source text of a program.
`Example text`	Exact user entry. These are words and characters that you enter in the system exactly as they appear in the documentation.
`<Example text>`	Variable user entry. Pointed brackets indicate that you replace these words and characters with appropriate entries.

Internal Use SAP Partner Only

Icons in Body Text

The following icons are used in this handbook.

Icon	Meaning
	For more information, tips, or background
	Note or further explanation of previous point
	Exception or caution
	Procedures
	Indicates that the item is displayed in the instructor's presentation.

Contents

Internal Use SAP Partner Only

Internal Use SAP Partner Only

Course Overview

This course provides an overview of ABAP Web services.

Target Audience

This course is intended for the following audiences:

- Project team leaders and consultants who are interested in obtaining a overview of ABAP Web services.
- Project managers and project team members who need to decide whether ABAP Web services should be used within applications
- Developers who want to develop and use ABAP Web services

Course Prerequisites

Required Knowledge

- ABAP programming experience (SAP course BC400)

Recommended Knowledge

- Basic knowledge of Internet technology
- ABAP OO programming experience (SAP course BC401)

Course Goals

This course will prepare you to:

- Understand the Web service paradigm
- Provide ABAP Web services on a SAP NetWeaver Web Application Server
- Integrate ABAP Web services into an ABAP application

Course Objectives

After completing this course, you will be able to:

- Describe the Web service paradigm
- Create an ABAP Web service using the *Service Definition Wizard*
- Create and configure a *service interface* and *service variant*
- Publish ABAP Web services to a UDDI repository

- Generate an Web service client proxy for consuming a Web service

SAP Software Component Information

The information in this course pertains to the following SAP Software Components and releases:

Internal Use SAP Partner Only

Unit 1

Overview

Unit Overview

This unit provides an introduction to Web services. It introduces the Web service paradigm and describes the relationship between Web services and the Enterprise Services Architecture (ESA).

Unit Objectives

After completing this unit, you will be able to:

- Explain what a Web service is
- Explain how Web service providers and Web service requesters interact
- Describe the evolution of the *SAP Web Application Server*
- Explain what is meant by Enterprise Services Architecture (ESA)
- Explain the objectives of SAP NetWeaver

Unit Contents

Internal Use SAP Partner Only

Lesson: Introduction: What Is a Web Service?

Lesson Overview

This lesson provides an introduction to Web services.

Lesson Objectives

After completing this lesson, you will be able to:

- Explain what a Web service is
- Explain how Web service providers and Web service requesters interact

Business Example

You want to describe the Web service paradigm and the standard Internet technologies used in combination with Web services.

What is a Web service?

Business processes are divided into a certain number of process steps. You can assign one or more functions to each of these steps and an executing software component to each of these functions. If you look at a typical heterogeneous system landscape in an organization, it is quickly apparent that the necessary functions in a business process are not all implemented using the same technology and the same components. In particular, the integration of an ever increasing number of business partners further complicates this problem. A modern software infrastructure must therefore be capable of integrating functions that are implemented on very different software components into an efficient global process.

Internet technology already provides the basis for communicating with distributed services. Superimposed onto this simple, globally accepted communication standard, XML (eXtensible Markup Language) provides the basis for defining additional necessary standards. Only when we move away from proprietary definitions and toward generally accepted standards can there can be any guarantee of smoothly integrating all the functions and partners involved in the process. The result is Web services.

Figure 1: Web Service Definition

A Web service is an independent, modular, self-describing application function or service. Based on XML standards, this application function can be described, made available, located or called using standard Internet protocols. Each Web service therefore encapsulates a piece of functionality that can be used, for example, to forward a price query to a provider, check the availability of an item in an enterprise resource planning system, locate a telephone number, or even to run credit card checks, convert currencies, or implement payroll functionality.

The Web Service Paradigm

The provider of a service is generally called a **Service provider**. If the service is a Web service, the service provider will have a corresponding XML-based description (WSDL document). In principle, any programming language can be used to implement this service. Based on the HTTP transport protocol, Simple Object Access Protocol (SOAP) is now established as the quasi-standard access protocol. In a client/server relationship, the service provider can be regarded as the server.

When publishing a service, the service provider transmits information about itself and a description of the service it is offering and transfers this to the **service registry**. A service registry can be described as a type of "Yellow Pages" for Web services. In addition to other data, it also provides information on calling the Web service, for example. The service registry therefore provides a description of the Web service only. This description forms an abstraction layer and is therefore not dependent on the corresponding implementation. The Web service itself is hosted by the service provider.

The user of a Web service is called a **service requester**. A service requester can be, for example, someone who locates a Web service using a Web browser and then uses this service. In most cases however, the service requester is an application

that accesses the Web service. The application can also “bind” to the service on request, that is, the application can dynamically create a Web service client proxy at runtime in order to access the Web service. The application obtains the necessary information for this from the service description, which is in turn stored in the service registry. However, if the application recognizes the provider and the call details, it can obviously use the Web service without having to access the service registry. In a client/server relationship, the service requester is the application client.

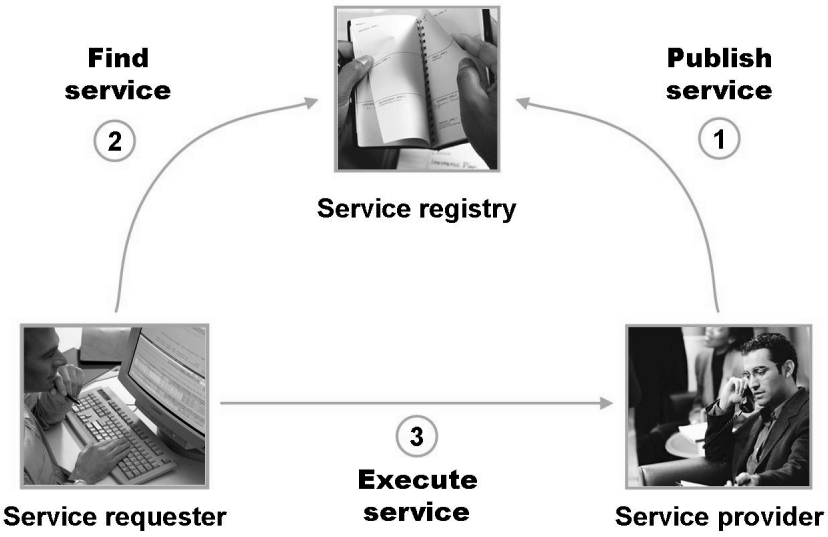


Figure 2: Web Service Paradigm

Central Internet Standard for Web Services

Web services can exist in any implementation. If Web services are to be called from any application, a standardized description is required. Web Services Description Language (WSDL) has been shown to best meet this demand.

A Web services description in WSDL alone, however, is not sufficient. To find the right business partner and corresponding service quotation, you will need a “register of companies” to help you to find the service you need. The Web service provider must also be able to make its offer publicly available as easily as possible. Universal Description, Discovery and Integration (UDDI) offers a solution. See http://www.uddi.org.

UDDI provides the necessary tools with its UDDI Business Registry and UDDI specification. The specification provides a detailed description of how to locate and register services. The UDDI Business Registry contains a list of registered companies

with their service offerings. The UDDI Business Registry is accessed either manually via Internet pages or via XML-based messages, which are described in the UDDI specification.

Many companies provide public UDDI servers. Examples include:

- SAP: http://uddi.sap.com/
- IBM: http://uddi.ibm.com/
- Microsoft: http://uddi.microsoft.com/

An appropriate protocol definition is required to call Web services based on Internet technologies. SOAP provides a more straightforward standard that allows you to call Web services in decentralized, distributed landscapes. Similar to the standards discussed earlier, SOAP is also based on an XML language definition. SOAP defines what is known as an **Envelope**. This Envelope contains the actual XML-based message and additional information on how the message is to be processed, for example. A further series of conventions was also adopted for describing the technical constraints.

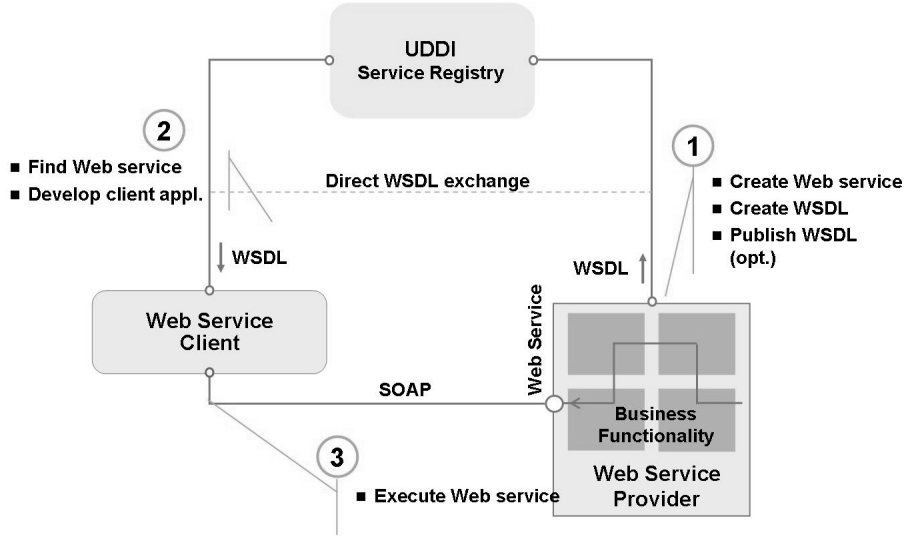


Figure 3: Web Services and Internet Standards

XML (eXtensible Markup Language)

XML is an extensible markup language for exchanging structured documents over the Internet. XML documents are increasingly used to support the exchange of business documents and messages, and thereby strengthen cooperation between companies.

SOAP

SOAP specifies a package of XML documents for transport via Internet protocols such HTTP(S), SMTP, or FTP. This protocol is used to call Web services in distributed system landscapes. A SOAP message has a header (additional information concerning security and transaction) and a body (content of the message).

WSDL (Web Service Description Language)

WSDL is an XML-based description language for Web services. WSDL documents are broken down into the names of the services, messages that are exchanged to use these services, links to specific transport protocols, and addresses at which a Web service is available. WSDL is an integral part of, and is used by, UDDI.

UDDI (Universal Description, Discovery and Integration)

UDDI is a Web-based registry that can be accessed via the Internet. The registry consists of a list of Web services in WSDL format and is used to locate these services. UDDI is different from other registry services insofar as it does not store documents or specifications, but only references them.

Example: Credit card check

Web services have many different uses: sending price queries to product providers, checking the availability of items in an enterprise resource planning system, finding telephone numbers, querying share prices, accessing current meteorological data, and performing currency conversions. The following figure shows how a Web service can be used to integrate a credit card check into a process landscape.

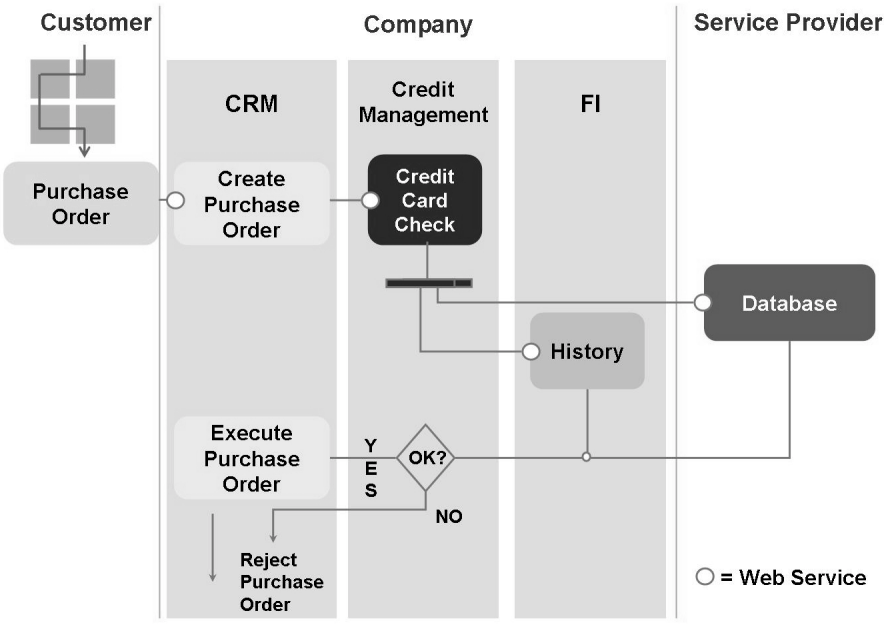


Figure 4: Example: Credit card check

Web Service Support in SAP NetWeaver AS 7.0

SAP provides a standardized architecture and set of tools for creating Web services. These include the SAP development environments SAP NetWeaver Developer Studio and ABAP Workbench, which are geared towards the Java and ABAP programming languages. This means that SAP users can provide Web services with a minimum of effort. Existing BAPIs, remote-enabled function modules, Enterprise JavaBeans (EJBs), Java classes, and SAP XI server proxies can be used for setting up Web services.

SAP NetWeaver AS 7.0 implements the basic standards for Web services: XML, SOAP, WSDL, and UDDI. *ABAP Workbench* provides an environment for creating, publishing, searching, and calling Web services. *SAP NetWeaver AS 7.0* can act both as a **server** for Web services and a **client** for consuming Web services.

SAP provides the *Web Service Framework* for *SAP NetWeaver AS 7.0* as a separate infrastructure for the development, publication, and use of Web services. The Web services framework consists of the following components:

- A distributed and interoperable SOAP runtime (*NetWeaver AS ABAP* or *NetWeaver AS JAVA*)
- Development environment of *NetWeaver AS ABAP*: Web services can be created or included in the *Object Navigator* (SE80).
- Development environment of *NetWeaver AS JAVA*: Web services can be created or included in the *SAP NetWeaver Developer Studio*.
- Tools for supporting UDDI registration

SOAP requests for Web services implemented in ABAP are processed using the Internet Communication Framework (ICF).

SAP NetWeaver Application Server as a Web Service Provider

SAP NetWeaver Application Server can function as a *service provider*. This allows SAP users to use existing BAPIs, remote-enabled function modules, EJBs, Java classes, and SAP XI server proxies for setting up Web services.

With the *Service Definition Wizard* you can make functions in the development environment available as Web services with just a few clicks of the mouse. The wizard contains predefined profiles with predefined settings for securing Web services or the transport protocol being used.

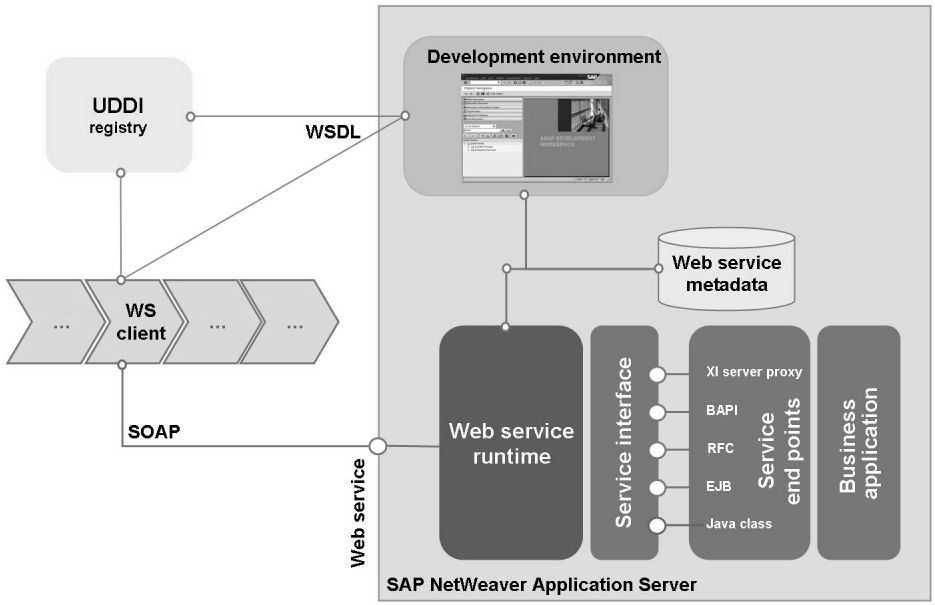


Figure 5: SAP NetWeaver Application Server as a Web Service Provider

The *Service Interface* is the interface that the Web service provides for the clients. The Service interface is abstracted from the “actual”, original application interface. A 1:1 mapping between the original, implemented interface and the interface provided for the client is applied to the Service interface as standard. Users have the option of customizing this standard interface to their own requirements. For example, they can rename and hide parameters, or provide default values. This means that based on the original, implemented interface, a view can be defined easily to provide an individually adapted, platform-independent interface for the Web service client.

SAP NetWeaver AS as a Web Service Client

The SAP NetWeaver Application Server (SAP NetWeaver AS) doesn't just support Web services on the server side. It is also capable of calling Web services as a Web service client. This is supported by the development environment again.

A *Web service client proxy* must be generated before a Web service can be implemented in an application. Web service client proxies are generated using the Web service's WSDL document. Every standard-compliant WSDL description can be used as an entry for proxy generation. WSDL documents are found either in the UDDI business registry using a URL/HTTP destination, or in a local file.

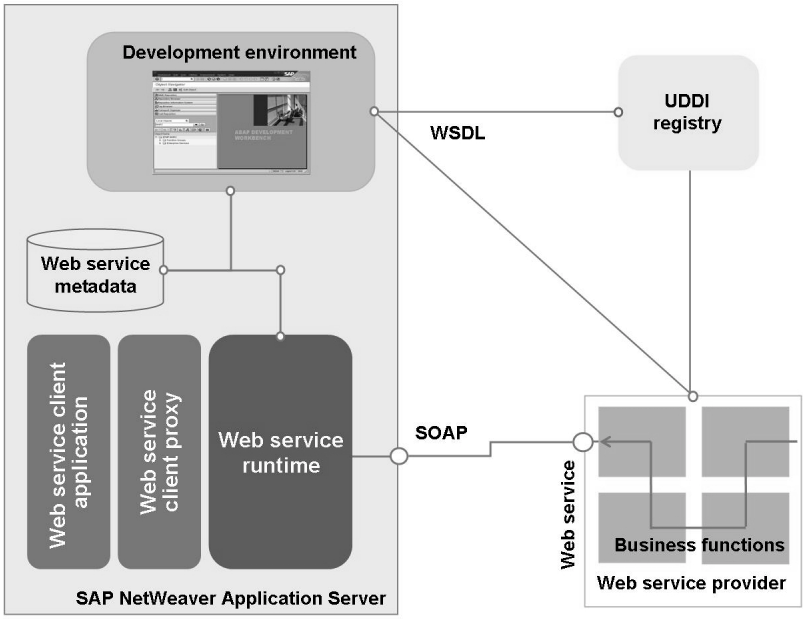


Figure 6: SAP NetWeaver Application Server as a Web Service Client

A Web service client proxy acts as a transfer program. A client proxy is used to connect to the server of the required Web service. The developer can concentrate on the business application while the technical aspects, such as the automatic packaging of calls to a SOAP message or the evaluation of received responses, are carried out using the proxy.

SAP's Web Service Solution

A complete Web service solution presupposes the use of open technology standards for Web services, which must be supported by an underlying platform. In response to this, SAP provides the SAP NetWeaver and, in particular, the SAP NetWeaver Application Server within SAP NetWeaver, which supports these technologies. However, this is not the only feature that distinguishes SAP from other providers who also offer corresponding platforms. The SAP system now integrates a multitude of business processes. The best way to achieve a complete Web service solution is to provide these business processes as Web services now.

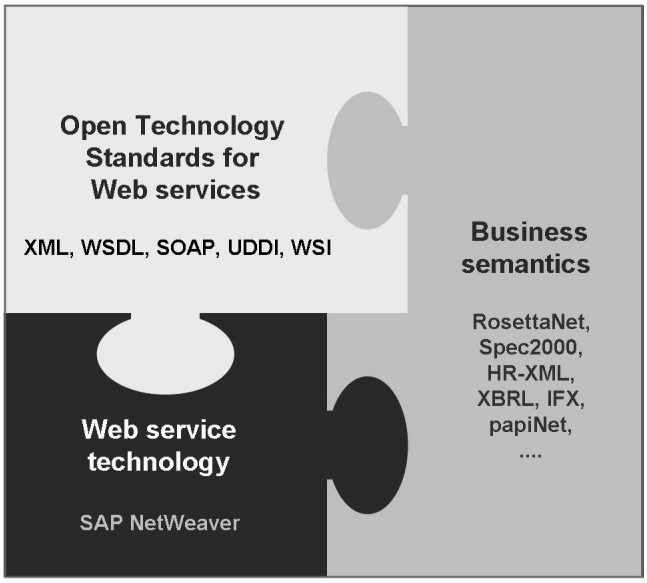


Figure 7: The SAP Web Service Solution

Lesson Summary

You should now be able to:

- Explain what a Web service is
- Explain how Web service providers and Web service requesters interact

Internal Use SAP Partner Only

Lesson: ESA and Web Services

Lesson Overview

This lesson serves as an overview of SAP integration technologies. It also explains the core terms in the SAP NetWeaver environment, such as composite applications, enterprise services, and Web services.

Lesson Objectives

After completing this lesson, you will be able to:

- Describe the evolution of the *SAP Web Application Server*
- Explain what is meant by Enterprise Services Architecture (ESA)
- Explain the objectives of SAP NetWeaver

Business Example

You need to explain the interaction between Web services and the Enterprise Services Architecture.

Evolution of SAP Basis to NetWeaver AS

If you look at the development of the technology basis of SAP systems over the last releases, the trend towards open standards and more flexible system architectures - away from monolithic applications with proprietary protocols and programming languages - is clearly discernible. In the days of Basis 4.6, the Internet capability of the platform for user interfaces (HTML) was implemented solely through the SAP Internet Transaction Server (SAP ITS) and, for process interfaces (XML), through the SAP Business Connector (SAP BC).

Internal Use SAP Partner Only

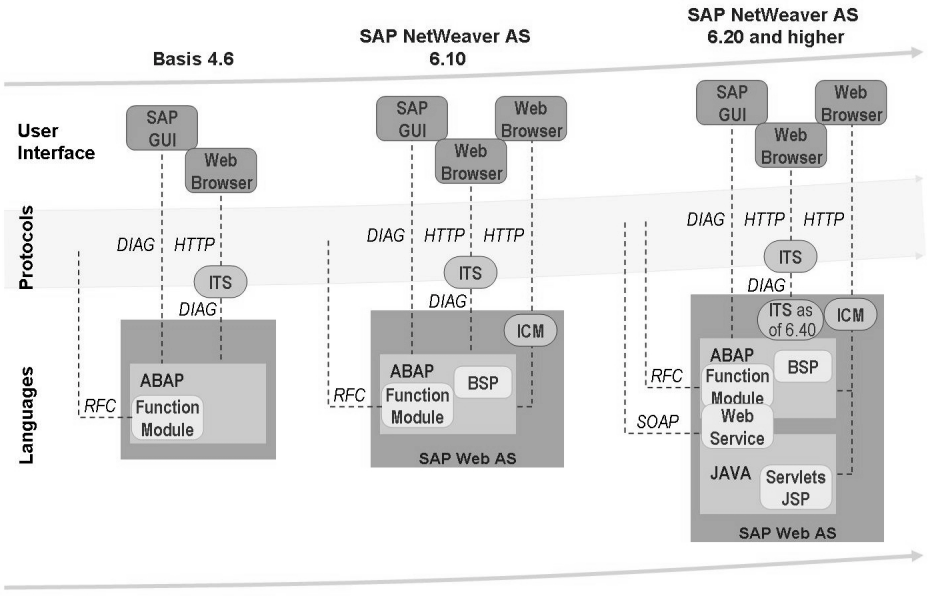


Figure 8: Basis release 4.6 -> SAP Web AS 6.40

SAP ITS was delivered for the first time in release 3.1G. SAP ITS is a software component that serves as a gateway between a Web server and an SAP system. SAP ITS serves as a connection between the protocols and formats of the Internet (HTTP(s), HTML) and those of the SAP system (DIAG, RFC, and screens). Web applications that were developed specially for SAP ITS are called Internet Application Components (IACs). These include Employee Self Services (ESS) or the SAP Online Store. *SAP ITS* is an integral part of the kernel in *SAP Web Application Server (SAP Web AS 6.40 for short)*. In this way, for example, the SAP GUI for HTML implementation model can be used from SAP Web AS 6.40 without a dedicated SAP ITS. You can find more information about ITS on SAP Service Marketplace under the quick link */sap-its*.

Based on the highly scalable infrastructure, new technologies are available as of SAP Web Application Server (SAP Web AS) 6.10 for processing HTTP requests (for example, from a browser) directly from the Internet, or for sending them to the Internet as HTTP client requests. To achieve this, the SAP kernel was enhanced to include the Internet Communication Manager (ICM). This ICM renders the classic SAP Application Server as an SAP Web Application Server. Through the ICM, the kernel can “speak” HTTP directly. Thus, the SAP Web Application Server constitutes the technical platform for new applications at SAP, customers, and partners. The Business Server Pages (BSPs) are a programming model for such applications.

Similar to a transaction in the classic SAP R/3 sense, a Business Server Page is an application that is functionally complete within itself. However, this application is not executed in the SAP GUI but in a Web browser or another mobile device browser.

Access through the network is carried out using the HTTP or the HTTPS protocol; other standard products, such as firewalls or proxy servers, can also be implemented. BSP applications can access all elements on the application server (function modules, BAPIs, database tables, and so on).

With SAP Web AS 6.20, Java as a complete programming language has become an equal partner to ABAP. With Web AS Java, SAP has a complete J2EE-compatible application server in the product portfolio. It provides developers with a development and runtime environment for new applications based on Java. Examples of SAP software components that use the J2EE engine are SAP CRM 3.1, SAP Enterprise Portal, and SAP Exchange Infrastructure.

From SAP Web AS 6.20, Web services technology can also be used. Web services are based on open and generally accepted standards implemented by the SAP Web Application Server. These include Web Services Description Language (WSDL), Simple Object Access Protocol (SOAP), and Universal Description, Discovery and Integration (UDDI). Web services are independent, executable entities that can be published, searched for, and accessed network-wide.

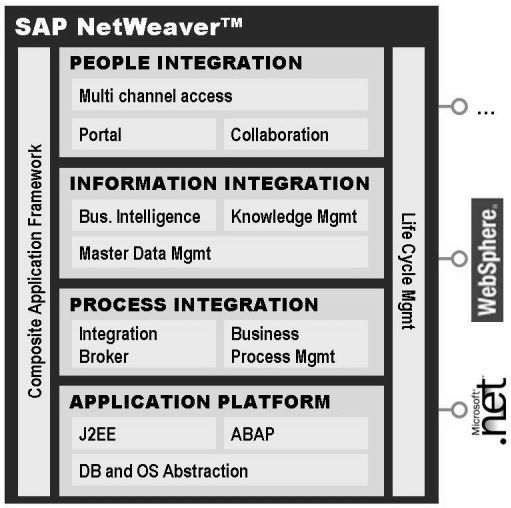


Figure 9: Application Platform as Basis of SAP NetWeaver

The Application Platform is the basis of SAP NetWeaver. Thus, for example, in SAP NetWeaver 04 generally available since September 2004 the Application Platform is implemented using SAP Web Application Server 6.40. SAP NetWeaver as an open integration and Application Platform is, in turn, the basis for all SAP solutions on given hardware.

Current Challenges in Companies with Expanding IT Landscapes

In today's climate, reducing costs, finding new ways to increase revenue and profitability, and being able to react flexibly to all kinds of changes are all items at the top of a typical company's wish list. In this respect, companies must pay special attention to the question of how to adjust and integrate new and existing applications and implement new applications flexibly. Optimum use should be made of existing investments; at the same time, companies want to - and must - support new business processes with greater intelligence and speed.

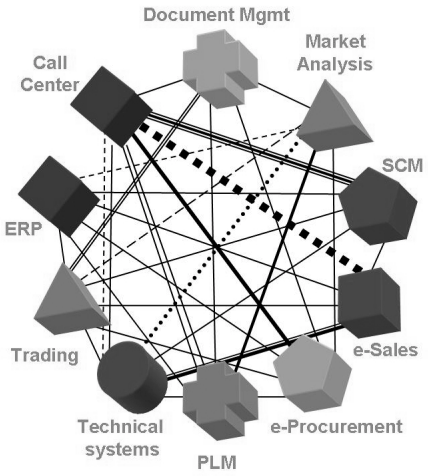


Figure 10: The Challenge of Today's IT Landscapes

Use the example of a company like Cisco Systems, which has experienced more than 60 company mergers. It needs to be able to provide all its customers with consistent information about order statuses across all of its business units' product lines and structures. The company must therefore use information contained in numerous new and old applications. If only business structures and customer requirements change in the course of time, the costs for maintaining and expanding such a service can increase exponentially.

Another challenge is posed by the legal requirements that increasingly demand traceability of business processes and force companies to adapt their processes for legal reasons.

Studies of the time required to change existing business processes or implement new business processes indicate that this still takes between several months and several years for each process. An inflexible IT landscape was cited as the reason for this in approximately one third of cases.

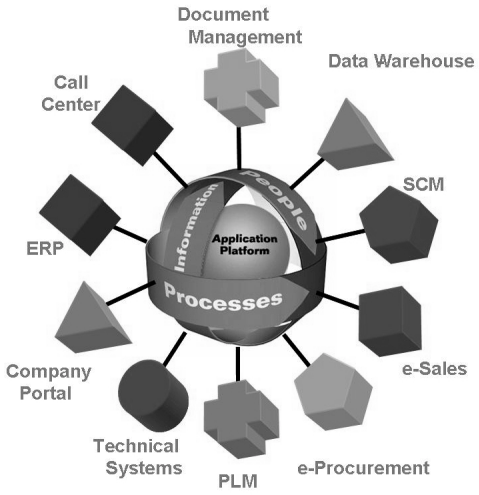


Figure 11: Integrated System Landscape

SAP NetWeaver helps you to meet the challenges listed above. For example, it reduces the complexity of system landscapes in the following ways:

- SAP NetWeaver is a complete package of integrated components to provide solutions for customers' integration requirements.
- A **single** platform is used to integrate information and systems.
- SAP NetWeaver provides functions that eliminate the need for time-consuming and costly integration projects.
- Compatibility with .NET and J2EE is guaranteed.
- Enterprise Services Architecture can make business processes more flexible.

SAP NetWeaver comprises a range of modules that can be examined individually. Essentially, however, the solution as a whole is more than the sum of its parts. As an open integration and application platform, SAP NetWeaver offers a multitude of solutions. As a technology platform, it integrates the different applications, information, and systems, and affords users common access to the entire system landscape without the need to abandon existing investments.

Enterprise Services Architecture: New Options for Integration

What is SAP Enterprise Services Architecture (ESA) and how can this new infrastructure allow IT to be viewed not only as a cost factor, but also as playing a lasting, positive role in integrating processes and making them more flexible? In brief: ESA is not a stand-alone product. Rather, it is an SAP concept for converting a services-oriented architecture for customizable business solutions. It provides a basis

for the development of new solutions and for the integration of existing products. Through an open structure, with the help of Web services and other standards, ESA affords new types of architecture that are easily modifiable, user-friendly, and supportive of innovation. Enterprise Services Architecture will be used as a basis for all SAP solutions in the future.

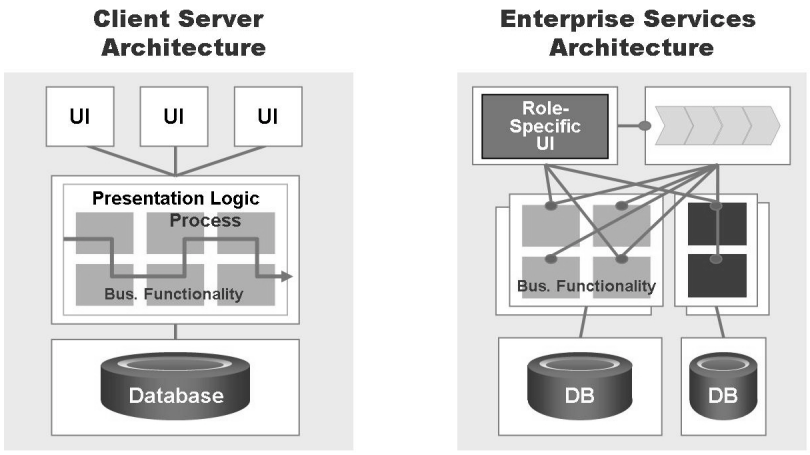


Figure 12: Client-Server Architecture Versus ESA

In conventional client-server architecture, the business process data is contained in the system database, the application processes run on application servers, and retrieval is based on a specific, predefined interface. Business processes that are not part of the standard SAP environment can be connected using interfaces. The work involved varies and may be considerable in some cases. Processes are very often integrated by "human integrators" because the company employees know when they need to access which systems to maintain data in the company business processes.

The idea behind Enterprise Services Architecture is to eliminate monolithic, complex architectures, and replace them with loosely coupled components that are smaller and reusable. These components, referred to as **enterprise services**, can then be incorporated into your company's business process environment in accordance with general standards and with minimum effort. We call these **composite applications**, and they use enterprise services to represent a more extensive process. Enterprise Services Architecture thus represents a new approach for the future development of applications.

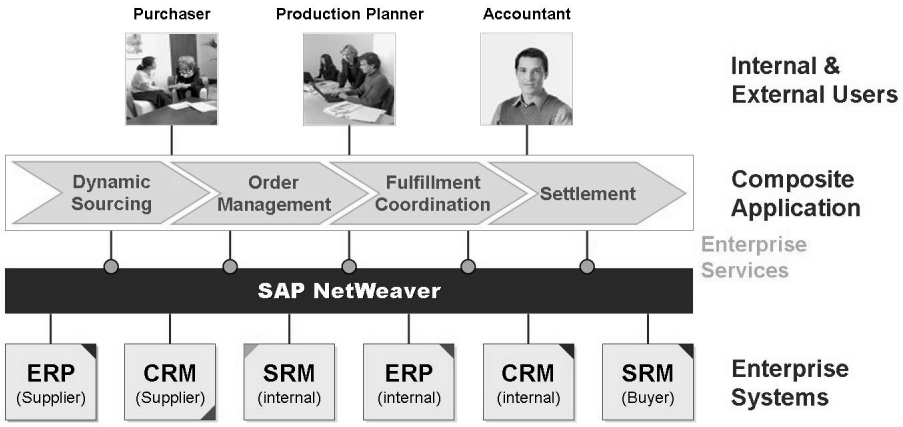


Figure 13: Enterprise Services Architecture (ESA)

An **enterprise service** does not focus on detailed functions, but rather on a complete, industry-specific process. For example, you could use an enterprise service to cancel a purchase order. From a business perspective, an enterprise service may represent various actions in various systems:

- Send a confirmation to the customer
- Remove the order from production
- Cancel material requirements
- Change the order status
- and so on.

An enterprise service comprises all of the individual actions and therefore provides context-based business process logic. The contextual nature of an enterprise service is crucial because the individual functions in an order cancellation service in the automobile industry will differ from those in a similar service in the media sector, for example.

However, if you have decided on a context-specific definition of the “cancel purchase order” service, all providers of the service can implement the service in their system, which means the systems will ultimately become interchangeable, as long as the process does not change at company level.

The individual steps within an enterprise service can then be processed using Web services. How does a **Web service** differ from an **enterprise service**?

Enterprise services describe the broader business process logic. **Web services** are small, modular applications that use Internet technologies and are usually accessed as detailed functions in applications or enterprise services. There are

agreed standards for describing and accessing Web services. These include Web Service Description Language (WSDL), Simple Object Access Protocol (SOAP), and Universal Description, Discovery and Integration (UDDI).

At the core of ESA is the idea that data and application functions can be merged to form reusable enterprise services. To achieve this, ESA is modeled on the lean manufacturing model used in the automotive industry. In lean manufacturing, automobile subsystems (brakes, drive shafts, engines, steering mechanisms) are standardized to such an extent that they can be used and combined by various manufacturers. In other words, the components of a car are no longer exclusively provided by the relevant manufacturers. Lean manufacturing is not used in just the automotive industry; however, the industry has developed it to an advanced level.

Enterprise Services Architecture should place a company in a position similar to an automotive manufacturer. The intricate web of applications that has been implemented corresponds to the thousands of components in conventional automobile production, while the components of an ESA platform mirror the standardized components in the automotive industry.

The IT industry is only starting to develop this model. Components that fit this model roughly (because they have not yet been fully standardized) are largely used for basic technologies, for example, relational databases or Web server and Web browser technology.

Before a company can even begin to consider components and enterprise services, it must understand its own business processes and applications, that is, an analysis of the existing process landscape is crucial. Since increasing numbers of interfaces are used between different companies and different components in a company, the question of standards is now more important than ever before. SAP NetWeaver takes this into account by supporting the use of industry standards.

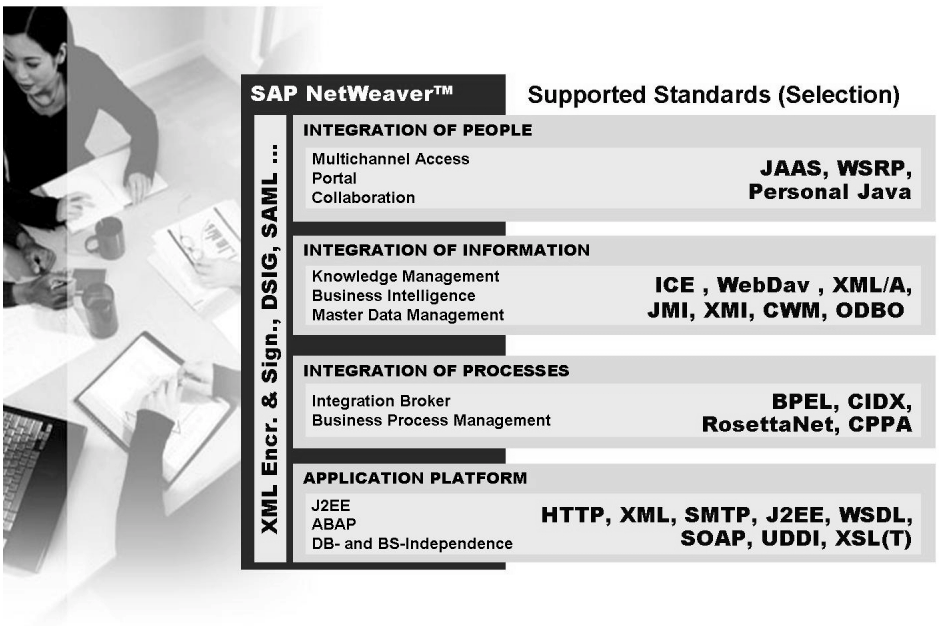


Figure 14: Using Standards

Today, the challenge lies in creating an IT environment where standardized components can work in synchronization, without giving rise once again to rigid, complex structures. What is the main disadvantage of rigid constructions? Tight coupling means that it is difficult and expensive to make changes. Everyone knows the rule “never change a running system.” However, this hinders creative and innovative thinking about ways of restructuring business processes.

This is where SAP NetWeaver comes in, with its coordinated integration package and modified approach in the form of Enterprise Services Architecture. ESA implies the conversion of a rigid architecture into a loosely coupled architecture based on Web services and components. Of course, SAP applications and third-party applications currently fulfill these criteria to varying degrees.

SAP is currently working on the development of an **Enterprise Service Infrastructure**. This infrastructure includes an **Enterprise Services Framework** for creating user interfaces, Web services, and enterprise services, as well as the development of a global repository for ESA objects.

Lesson Summary

You should now be able to:

- Describe the evolution of the *SAP Web Application Server*
- Explain what is meant by Enterprise Services Architecture (ESA)
- Explain the objectives of SAP NetWeaver

Internal Use SAP Partner Only

Unit Summary

You should now be able to:

- Explain what a Web service is
- Explain how Web service providers and Web service requesters interact
- Describe the evolution of the *SAP Web Application Server*
- Explain what is meant by Enterprise Services Architecture (ESA)
- Explain the objectives of SAP NetWeaver

Internal Use SAP Partner Only

Internal Use SAP Partner Only

Unit 2

Fundamentals

Unit Overview

This unit introduces the basics of ABAP Web service technology. It provides an introduction to RFC, BAPI, HTTP, XML, and SOAP topics.

Unit Objectives

After completing this unit, you will be able to:

- Describe the basics of RFC-enabled function modules
- Describe the TCP/IP reference model
- Describe HTTP as an application protocol of TCP/IP
- Describe the structure of an XML file
- Explain the difference between a DTD and an XML schema
- Describe how XSL programs are used
- Describe the structure of a SOAP message
- Describe SOAP exceptions (faults)

Unit Contents

Lesson: RFC Basics

Lesson Overview

Introduction to the function modules: searching, analyzing, testing.

Lesson Objectives

After completing this lesson, you will be able to:

- Describe the basics of RFC-enabled function modules

Business Example

Introduction to Function Modules

Function modules in ABAP are used to encapsulate source text that can be used by several applications. They can be compared to routines in a main program, whereby the function modules in the entire system are available for use, even partly from outside. A function module incorporates the actual source text that is executed at runtime, the interface (also called signature), and the properties of that function module.

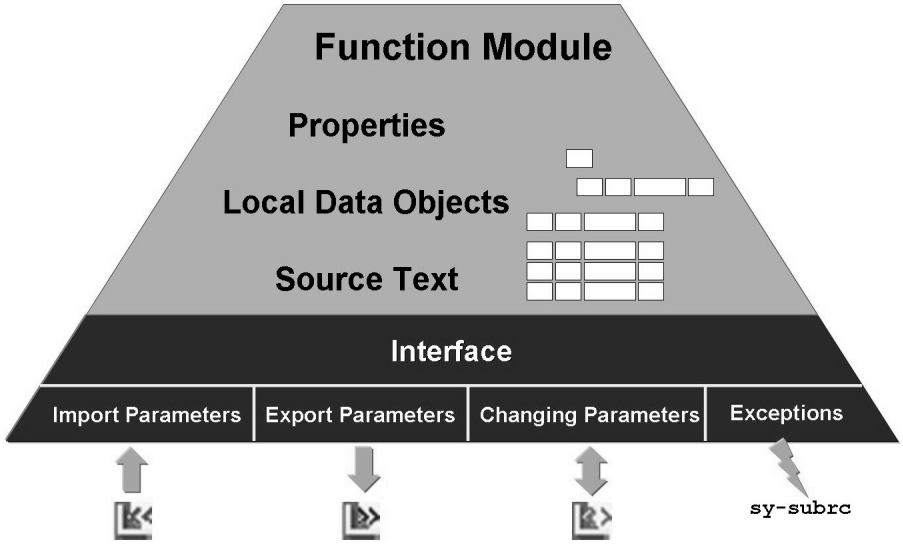


Figure 15: Elements of a Function Module

Internal Use SAP Partner Only

The interface of a function module incorporates:

Import Parameters

These are transferred to the function module, can be optional, and are often standard dictionary types.

Export Parameters

These are returned by the function module and are always optional.

Table Parameters

These are both import and export parameters for tables, though often they are only used for exporting.

Exception Parameters

These provide information about error situations in processing, which is then ended.

Like subroutines, function modules can also have local data types and data object definitions. Local screens are also allowed in a function group, although we will not examine this here because these mainly involve external calls for which dialog boxes are not possible.

In the SAP system, every function module is included in a so-called function group, just as routines belong to a main program. A function group typically combines several function groups that have the same applications in common.

Internal Use SAP Partner Only

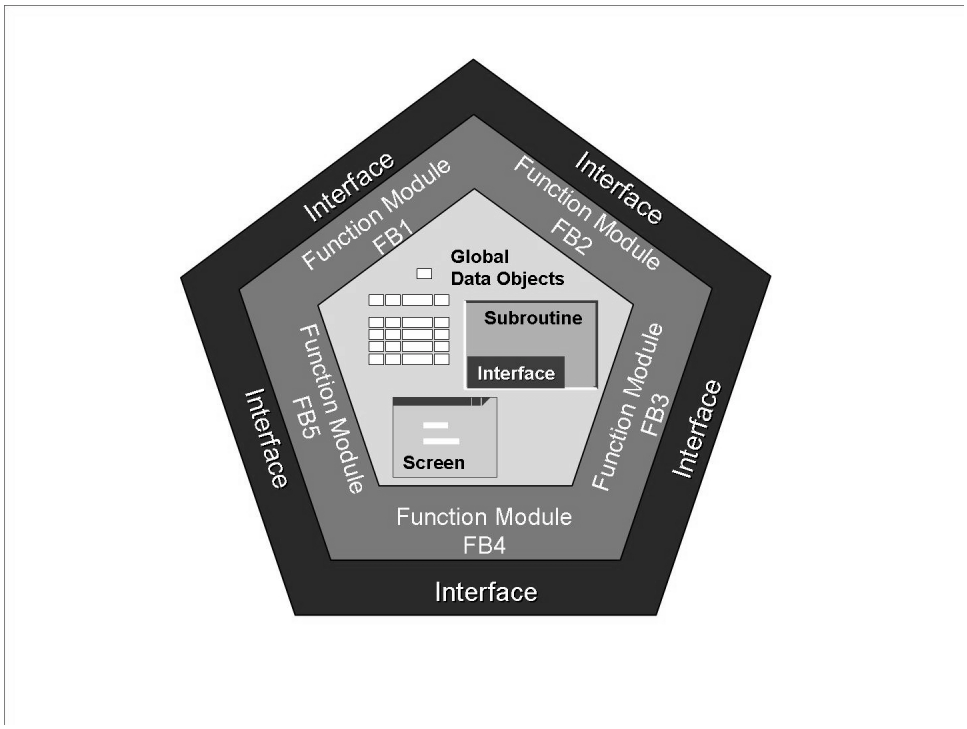


Figure 16: Function Group

Function groups can also have local data types and subroutines that can be used by all of the function modules in the group. When a function module is called, the whole function group is loaded.

The function library (also known as the *Function Builder*) is used for viewing function modules in the SAP system (transaction code SE37). You must either specify the name of the function module on the initial screen, or find one by running a search. The display of function modules is also included in the *Object Navigator* (transaction code SE80).

The properties of a function module are especially important for the external utilization of function modules, as the remote-enabled attribute must be set in the properties.

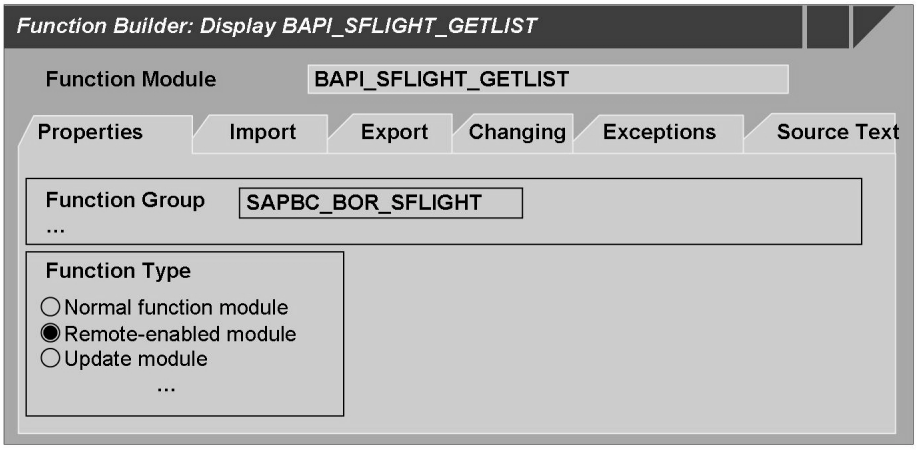


Figure 17: Properties of Function Modules

Hint: Function modules that are remote-enabled are often abbreviated as follows: **RFM** stands for **R**emote-enabled **F**unction **M**odule. Similarly, **RFC** stands for **R**emote **F**unction **C**all, whereby the function module can be called "remotely".

The interface parameters of a remote-enabled function module must be typed with objects of the ABAP Dictionary.

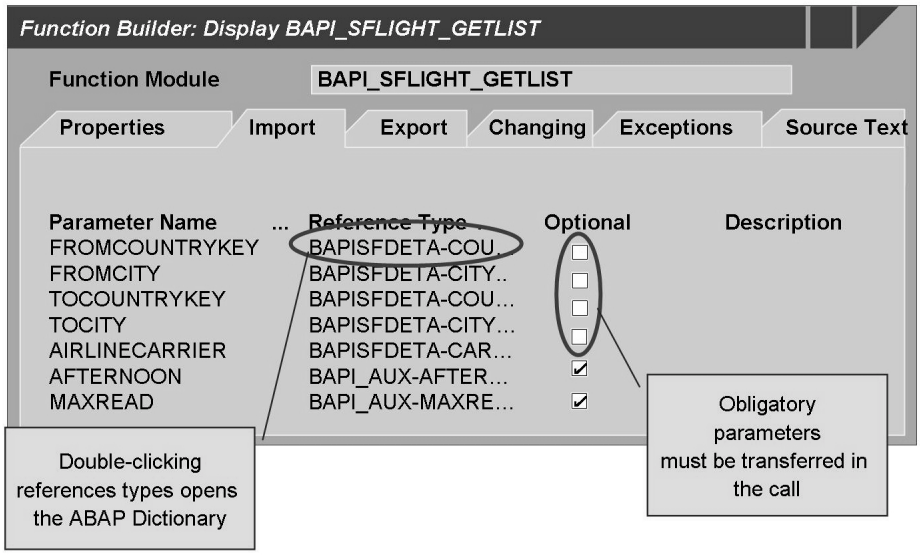


Figure 18: Function Module Interface

Internal Use SAP Partner Only

The import interface provides information about calling important parameters. By double-clicking the reference types provided, you can use forward navigation to view the information in the ABAP Dictionary: the semantic information is contained in the data element, while the technical information is in the domain.

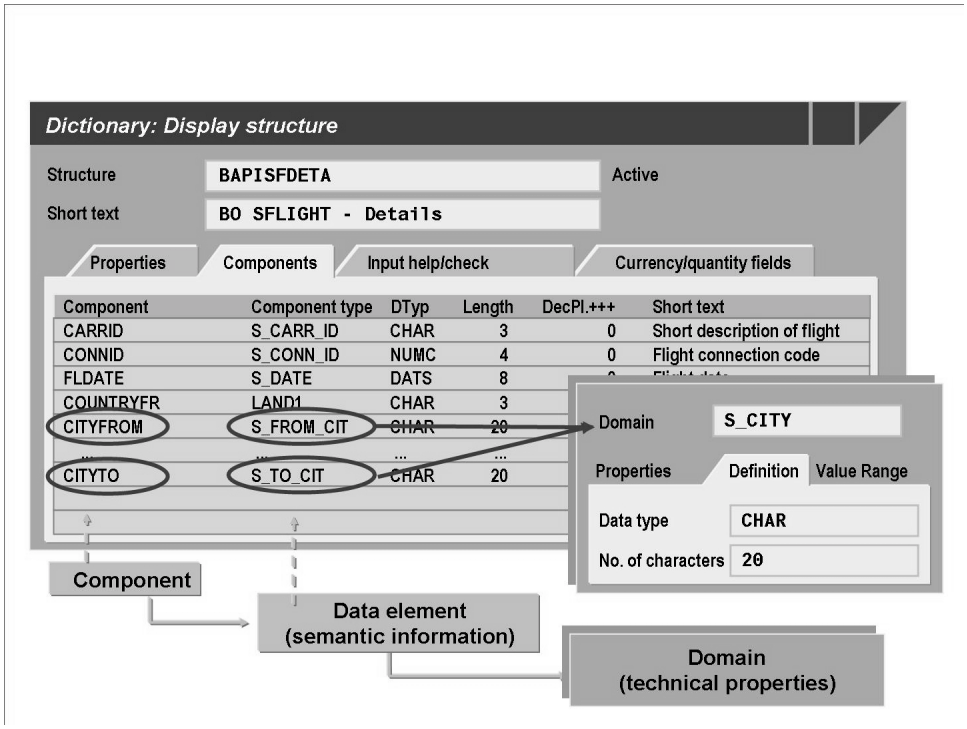


Figure 19: Structures in the ABAP Dictionary

In addition to documentation, the source text of a function module provides important information about processing. For an explanation of individual ABAP statements, you can call the ABAP keyword documentation by selecting the statement with the cursor and pressing F1.

To understand how a function module works, you can test them.

Hint: Testing a function module involves a complete execution of the source text, but no simulation.

During the test, the *Function Builder* generates a screen containing all the import parameters, so that values for calling the function module can be provided. The results, or any exceptions that occurred, are then displayed.

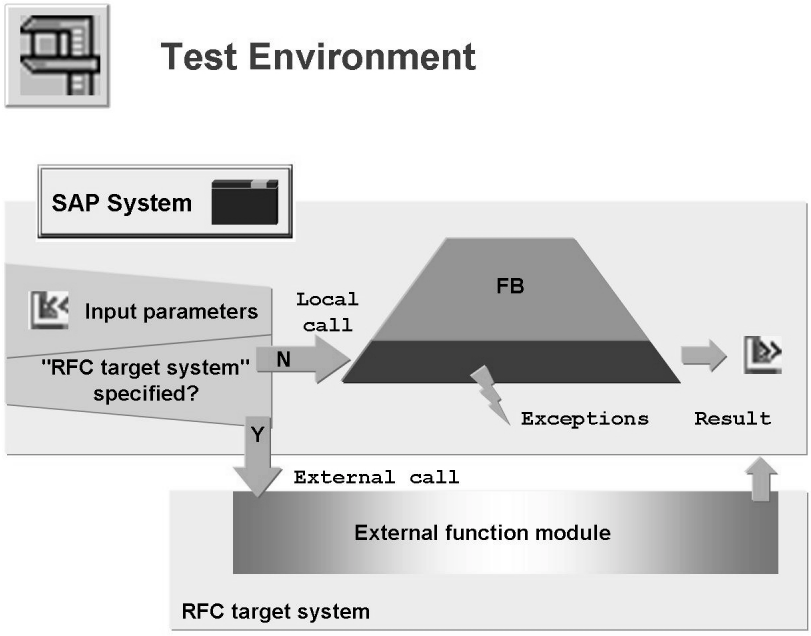


Figure 20: Testing a Function Module

When a function module is tested, a field called RFC target system is also displayed for remote-enabled function modules, so that when an RFC destination containing the connection parameters for the target system is specified, an RFC call can also be made. This will only work if the target system also contains this function module and if this has the same interface as the function module in the local system. The *Function Builder*, however, cannot react appropriately to RFC error situations, for example a connection failure.

Caution: The test environment of the *Function Builder* is a screen for which the usual screen conversions apply. This means that parameters such as date fields must be entered in the user-defined format (for example <Day.Month.Year>), but the function module receives and processes the ABAP-internal display (<YearMonthDay>).

Lesson Summary

You should now be able to:

- Describe the basics of RFC-enabled function modules

Internal Use SAP Partner Only

Lesson: Basics of HTTP

Lesson Overview

The first part of this lesson describes how communication via the Internet works and introduces the TCP/IP reference model. After that, we will focus on the HTTP application protocol, which is based on TCP/IP.

Lesson Objectives

After completing this lesson, you will be able to:

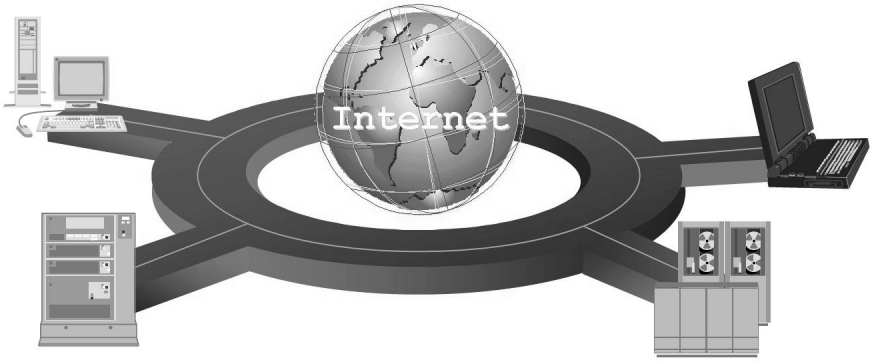
- Describe the TCP/IP reference model
- Describe HTTP as an application protocol of TCP/IP

Business Example

You need to describe the basic relationship between the TCP/IP communication protocol and the HTTP application protocol.

The Arrival of the Internet

The origins of the Internet lie in the ARPANET, developed by the Advanced Research Projects Agency (ARPA), a military network between four U.S. military institutes. It first went online in 1969. The aim was to replace the vulnerable centralized network architecture by a decentralized system with many independent cross connections, which would also continue to transport data should a front-end computer or data connection malfunction in that it would independently find another way through the network. This should then prevent total network failure in the event of a nuclear strike. It should also facilitate communication between different computer types, allowing networking of different military mainframe computers. To meet the different requirements, a new data transfer protocol needed to be developed for the network. Efforts to develop such a protocol eventually gave rise to TCP/IP.



- TCP/IP = Transmission Control Protocol/Internet Protocol
- Developed and standardized in the USA in 1970
- Available for almost all hardware and most operating systems
- Facilitates the connection of different systems to each other

Figure 21: The TCP/IP Communication Protocol

The ARPANET later gave rise to the Internet. All protocols developed for the internet in the 1990s were based on the technical capabilities of TCP/IP.

The TCP/IP Reference Model

The TCP/IP reference model, which was named after the two primary protocols (TCP and IP) of the network architecture, is based on the proposals made during the further development of the ARPANET. It describes the structure of and interaction between the network protocols from the Internet protocol family, and arranges them into four tiers built one on top of the other. This is also referred to as a **protocol stack**. In a layered architecture, it is vital that every layer communicates with the layer directly above it and the one directly below it. As a result, the individual layers can be easily exchanged. This means that it is possible to replace the ethernet architecture with a token ring architecture, for example.

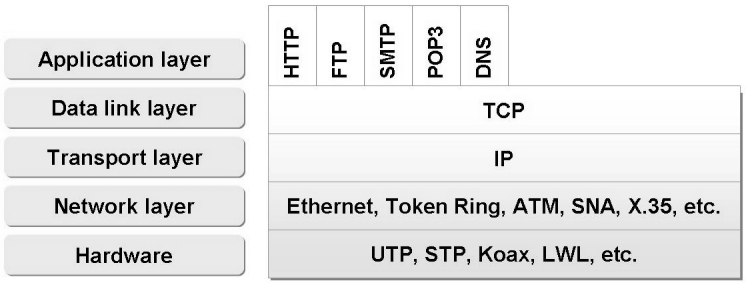


Figure 22: The TCP/IP Reference Model

The individual layers within the TCP/IP reference model perform the following functions:

Hardware and Network Layer: To be networked, computers need to be hooked up to each other with an electrical connection. In principle, this requires a "cable" to be laid, which must then be connected to the computers. The computers, therefore, must have specific hardware at their disposal. In the simplest case, ethernet cards are used as network cards, which are now very common in LANs. The ethernet hardware can exchange electronic signals between these two ethernet cards. Appropriate driver software is required for this to work properly; this enables the exchange of individual bits between the computers with no resulting deeper impact.

Transport Layer: The transport layer is located above the network layer. From a software point of view, this layer is implemented via the Internet Protocol (IP). It bundles the data to be transferred into packets and assigns these packets a sender and the recipient address. The data packets are forwarded to the network layer below for transfer. IP receives the data packets from the network and unpacks them. This renders the data transfer more convenient, as entire data packets can now be exchanged. A mechanism to define whether all data packets have arrived and what sequence these take has yet to be established.

Data Link Layer: The data link layer is located above the transport layer. It is implemented by the Transmission Control Protocol (TCP). This layer monitors the transfer, supplies any missing data packets, and arranges them in the correct sequence. As a rule, both the TCP and IP protocols are integrated into TCP/IP. As these protocols are located one above the other, this is often called a **TCP/IP stack**.

Application Layer: The application layer includes all protocols that interact with the application programs and that use the network infrastructure to exchange application-specific data. The most widely known protocols include HTTP (Hypertext Transfer Protocol); FTP (File Transfer Protocol); SMTP (Simple Mail Transfer Protocol) for sending e-mails; POP3 (Post Office Protocol Version 3) for retrieving e-mails; and DNS (Domain Name System) for conversion between domain names and IP addresses.

HTTP: A Special TCP/IP-Based Protocol

This story of HTTP (Hypertext Transfer Protocol) began in 1990 at the CERN Institute (European Organization for Nuclear Research). This was where Tim Berners-Lee developed the HTTP standard known as Version HTTP/0.9. HTTP, meanwhile, is available in Version 1.1. HTTP is an application protocol. It is used to communicate between a client and a server, and it is one of the most important protocols for the Internet. Typically, the server listens to a port - usually 80 or 8080 - at the request of a client. The communication process essentially consists of an HTTP request from a client to the server, through which the client requests an object from the server,

and an HTTP response that the server sends back to the client after having received the request. The response then contains the information that was requested by the client. This communication between client and server is based on messages written in text format.

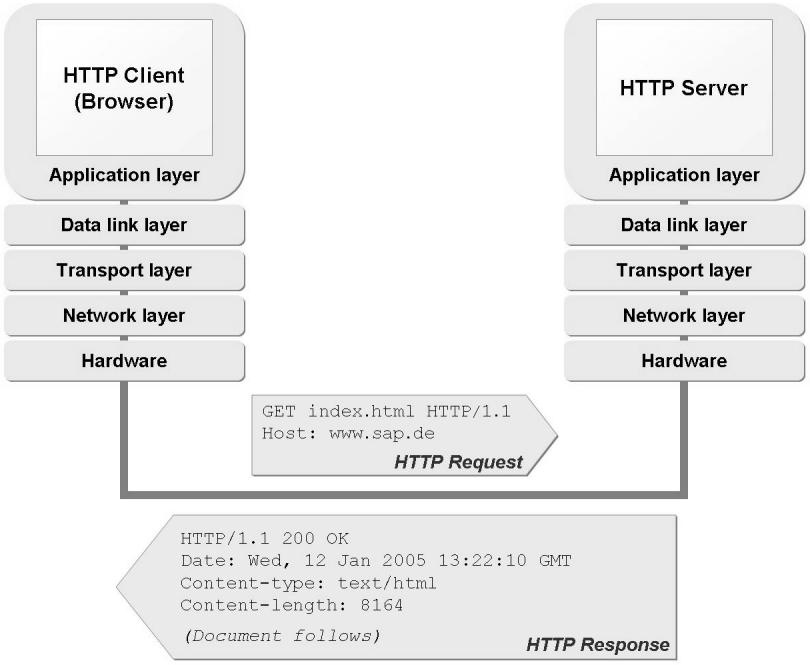


Figure 23: HTTP Request / Response Cycle

HTTP is an application protocol that was created to facilitate the transfer of multimedia data (text, pictures, videos, and audio files) in the simplest way possible. When we talk about the Web, we mean this service, which can be called from an HTTP client using the following address (URL= Uniform Resource Location):

```
http://<host>[:<port>]/...
```

A URL is made up of the transport protocol http://, the host name, and the domain name with the top level domain. The specification for the TCP port is optional and required only if the connection is handled through a port other than standard port 80. Paths and files are separated from one another and from the server address using a forward slash (/). If no other path or file is specified, the server sends a default file back to the client. If paths or files are specified, the HTTP server returns this file.

Format of HTTP Messages

Communication between the client and server occurs through the exchange of messages that transfer requests and responses between the client and server. These messages consist mainly of the HTTP header and the actual data. The HTTP header contains control information and the required URL. The header entries are arranged into four categories: general header, request, response, and entity header entries. The general header entries are contained both in the requests and in the responses. Entity headers describe the data part of the message. The entity body of the message contains the actual data that was requested. This could be an HTML document, for example.

Request/Status Line	`HTTP/1.1 200 OK`
General Header	`Date: Wed, 12 Jan 2005 14:08:11 GMT`
Request/Response Header	`Server: Apache/2.0.52(Win32)PHP/5.0.2` `Location: http://localhost:80/`

Entity Header	`Content-Type: application/xml`
	`Content-Length: 208`
	`Last-Modified: Mon, 10 Jan 2005 13:00:00 GMT`

Entity Body	`<?xml version="1.0" encoding="utf-8"?>`
	`<connection carrid="LH" connid="2402">`
	`<departure>FRANKFURT</departure>`
	`<destination>BERLIN</destination>`
	`<time dep="10:30:00" arr="11:35:00"/>`
	`</connection>`

Figure 24: Format of HTTP Messages

Request Line/Status Line: A distinction must be made between the request from the client and the response from the server. If a request is involved, it will be specified here which object (document or program) is to be accessed with which method. In addition, the protocol version to be used for the transfer (HTTP 1.0 or HTTP 1.1) will also be defined. HTTP defines a number of methods, including the well-known GET and POST methods. If a response from the server is involved, the protocol version and the status code for the result of the request are specified in the status line. This is then followed by text that describes the status code.

General Header: Every transmitted message (request or response) has the following fields that can be queried: Cache-Control, Connection, Date, Pragma, Transfer-Encoding, Upgrade, and Via. The header information also includes the time and date of the transmitted data packet.

Request Header/Response Header: The response header transmits additional information about the server. It uses the following fields: Age, Location, Proxy-Authenticate, Public, Retry-After, Server, Vary, Warning, and WWW-Authenticate.

Entity Header: The entity header transmits information about the length of the document or about the last change made to it. If no entity body is defined, the following fields provide information about the resources without actually sending them in the entity body: Allow, Content-Base, Content-Encoding, Content-Language, Content-Length, Content-Location, Content-MD5, Content-Range, Content-Type, Expires, and Last-Modified.

Entity Body: The entity body is separated from the header by a blank line. The actual data from the message is placed in the body. This can be either the client's user data or the server's response.

HTTP Status Codes

The status code provides information about the result of the request. It consists of a three-digit figure with additional, optional text. The first digit of the status code defines the response class. The five existing response classes are as follows:

1xx - Informational

The response from the server is of a temporary nature. The server received the request and is processing it at present.

2xx - Successful

The request was received, understood, and accepted by the server.

3xx - Redirection

The request could not be processed in full, and the server refers to other servers that the client must contact in order to process the request successfully.

4xx - Client Error

The request could not be processed by the server. This may be due to a syntactic error in the request on the side of the client, for example.

5xx - Server Error

As a result of an error arising on the server, the server was unable to process the request.

Most commonly, the return values involve the values *200 OK* or *404 Not Found*.

HTTP Methods

HTTP 0.9 only allowed the user to retrieve data from the server using the GET method. HTTP 1.0 included the new methods HEAD, POST, PUT, DELETE, LINK, and UNLINK. The LINK and UNLINK methods were removed from the current version, HTTP 1.1, while the OPTIONS, TRACE, and CONNECT methods were added.

GET

The GET method requests the document specified by the URL provided. This method facilitates the transfer of data in the actual request, which is then transmitted to a server program via parameters. The parameters are separated by a question mark added to the URL.

HEAD

The HEAD method functions in the same way as GET, but here only the header is returned as the response. The actual data (entity body) is omitted. The HEAD method can be used, for example, to check the validity of hyperlinks, or to determine the size of a document and work out its estimated download time.

POST

The POST method is used to transfer data to a server program. The data is contained in the body of the client request.

PUT

The PUT method tries to store the data transferred in the entity body under the specified URL on the server. An attempt is made on the server to generate a new object, which of course requires the appropriate permissions.

DELETE

This method can be used to delete the data stored under the specified URL if the appropriate permissions are in place. This and the PUT method are two of the riskiest. If the server was not configured properly, the data on the server can end up being manipulated.

TRACE

This method checks whether a message reaches the server correctly. The server sends back the same message as message/HTTP, unaltered. The TRACE method can thereby check whether the message was altered during transfer.

CONNECT

This method is used to establish a connection to an HTTPS server.

Hint: You can find more information about HTTP on the Internet. See for example http://www.w3.org/protocols.

Lesson Summary

You should now be able to:

- Describe the TCP/IP reference model
- Describe HTTP as an application protocol of TCP/IP

Internal Use SAP Partner Only

Lesson: Basics of XML

Lesson Overview

This lesson provides an introduction to XML. It will also help you to understand DTDs and XSL schemas.

Lesson Objectives

After completing this lesson, you will be able to:

- Describe the structure of an XML file
- Explain the difference between a DTD and an XML schema
- Describe how XSL programs are used

Business Example

Your company plans to use XML. You need to describe the steps that are required to use XML profitably at your company.

Introduction

Over the last few years, a myriad of information has been made available on the Internet in the form of HTML documents. Developed by the founder of the Web, Tim Berners-Lee, HTML became a successful, widely used file format in the course of the Web boom. HTML documents are text files that contain data and associated formatting instructions in the form of what are called HTML **tags**. HTML tags are quoted between angle brackets, and almost all HTML tags are marked with an opening tag and closing tag. Graphics and multimedia content can also be included in HTML documents. The following figure shows how a simple HTML document looks when displayed in a Web browser:

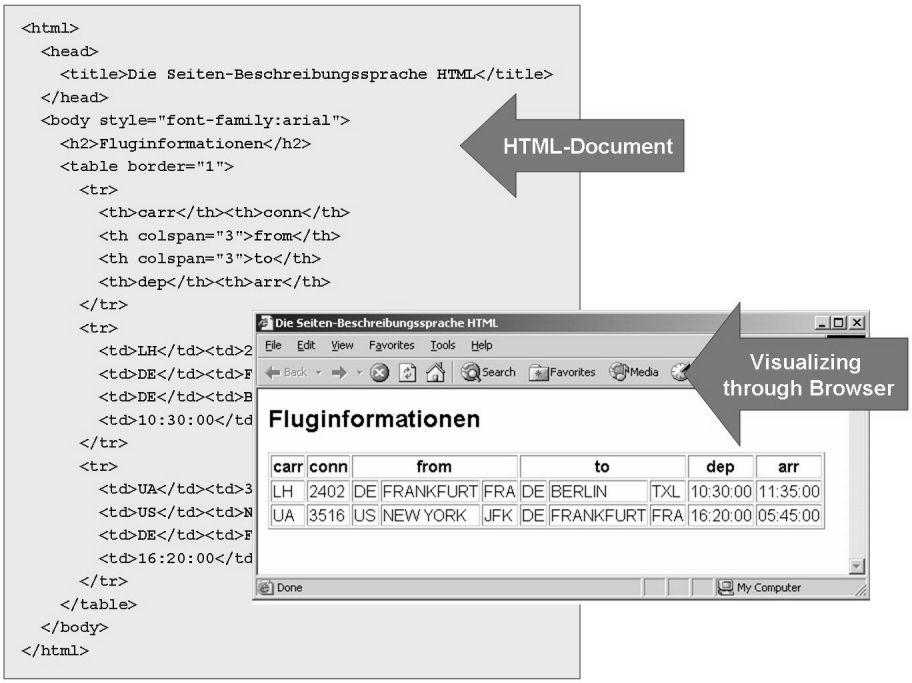
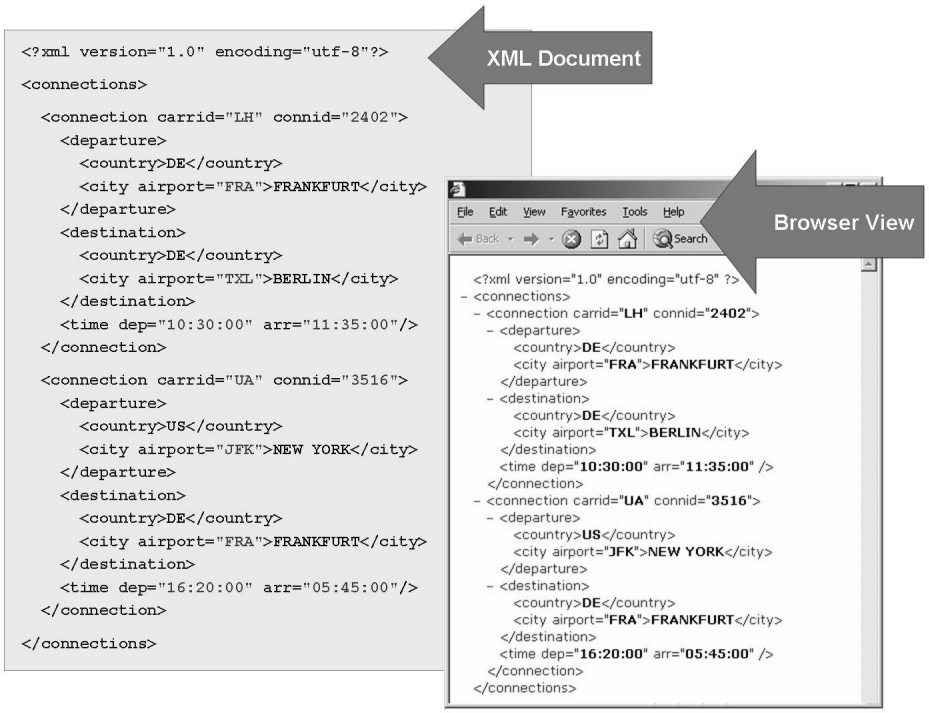


Figure 25: HTML - Hypertext Markup Language

Hint: You can find more information about HTML on the Internet. See for example http://de.selfhtml.org/html/.

Further processing of this information in the different systems is actually very limited. No provisions were made for adding extra tags to the HTML vocabulary. As a result, the World Wide Web Consortium (W3C) developed a new structured and extensible Markup Language, called eXtensible Markup Language (XML). This language facilitates a proper separation of data and associated formatting. XML documents mirror the structured data and can be exchanged using common Internet protocols. XML is platform- and manufacturer-independent. Other description languages are also defined using XML.




```
<?xml version="1.0" encoding="utf-8"?>
<connections>
  <connection carrid="LH" connid="2402">
    <departure>
      <country>DE</country>
      <city airport="FRA">FRANKFURT</city>
    </departure>
    <destination>
      <country>DE</country>
      <city airport="TXL">BERLIN</city>
    </destination>
    <time dep="10:30:00" arr="11:35:00"/>
  </connection>
  <connection carrid="UA" connid="3516">
    <departure>
      <country>US</country>
      <city airport="JFK">NEW YORK</city>
    </departure>
    <destination>
      <country>DE</country>
      <city airport="FRA">FRANKFURT</city>
    </destination>
    <time dep="16:20:00" arr="05:45:00"/>
  </connection>
</connections>
```

Figure 26: XML - eXtensible Markup Language

XML Syntax

The beginning of every XML document includes an XML declaration that identifies the XML version being used. Like an HTML document, an XML document consists of elements that are marked with an opening tag and closing tag. XML, however, does not provide any element formatting. It is a universal data description. In contrast to HTML, the tag names can be chosen freely. These should be chosen in line with the meaning of their content. The tag name in the closing tag must match the one in the opening tag. A distinction is made between capital letters and small letters (tag names are case-sensitive). The closing tag is obligatory. An element with no content (that is, no text between the opening and closing tags) has the following special form: `<tagName/>`. The forward slash before the closing bracket replaces the closing tag.

Any number of attributes can be defined within the preliminary tags. For attribute names, the same rules apply as for tag names. The value of an attribute is separated from its name by “=” and is enclosed in straight quotation marks: `<tagName attrName="attrWert">`

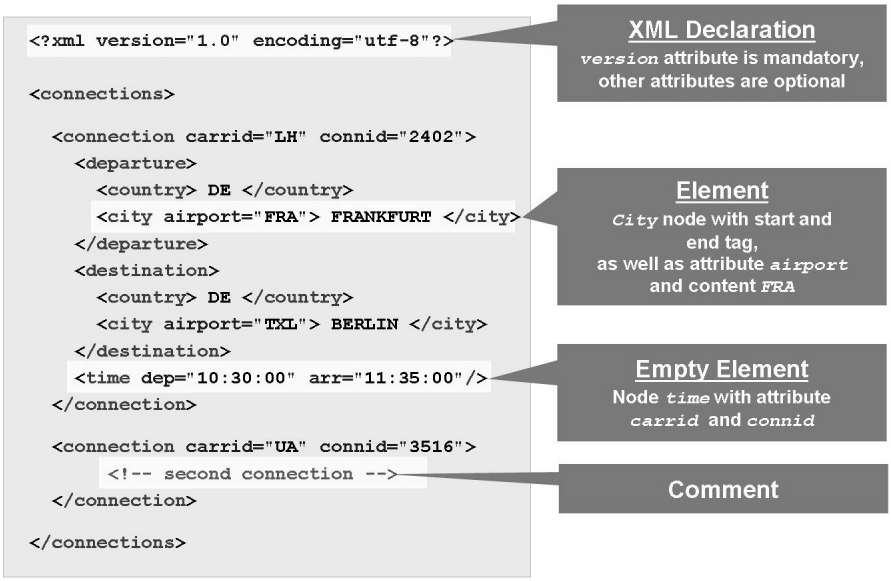


Figure 27: XML Syntax

If the content of an element contains a sign that occurs in the XML syntax (<,>,&, and so on), this sign needs to be substituted by a placeholder (entity).

Entities

Sign	Entity
<	<
>	>
&	&amp;

In line with the tag hierarchy, the elements are known as **root nodes**, **parent nodes**, or **child nodes**. An XML document can also be interpreted as a tree structure. Such a result tree is generated, for example, by parsers.

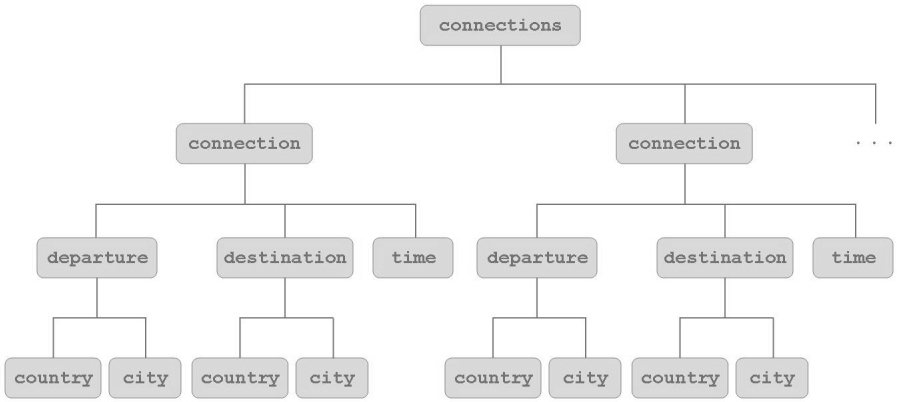


Figure 28: XML Tree Diagram

Document Type Definition (DTD)

If the same structures are to be used repeatedly for different XML documents, or if certain rules pertaining to the sequence and repeatability of elements are to be followed, it makes sense to use a **DTD** to check the validity of the XML structure. The DTD determines where and how often an element can occur in an XML document. The DTD uses a syntax which differs from that in XML documents. The number of keywords is restricted. DTDs can be stored both in the XML document itself or in an external file. These external files usually have the "*.dtd" file extension. The DTD is assigned by the document type declaration at the beginning of an XML document. The root element and the file reference are specified. The addition of the word SYSTEM indicates that the DTD is defined in an external file. PUBLIC enables access to a catalog file in which placeholders are defined for different paths. DTDs can, therefore, be stored on a central server. If the *standalone* attribute has the value *no* in the XML prolog, then a check will be run against the specified external DTD when the XML document is parsed.

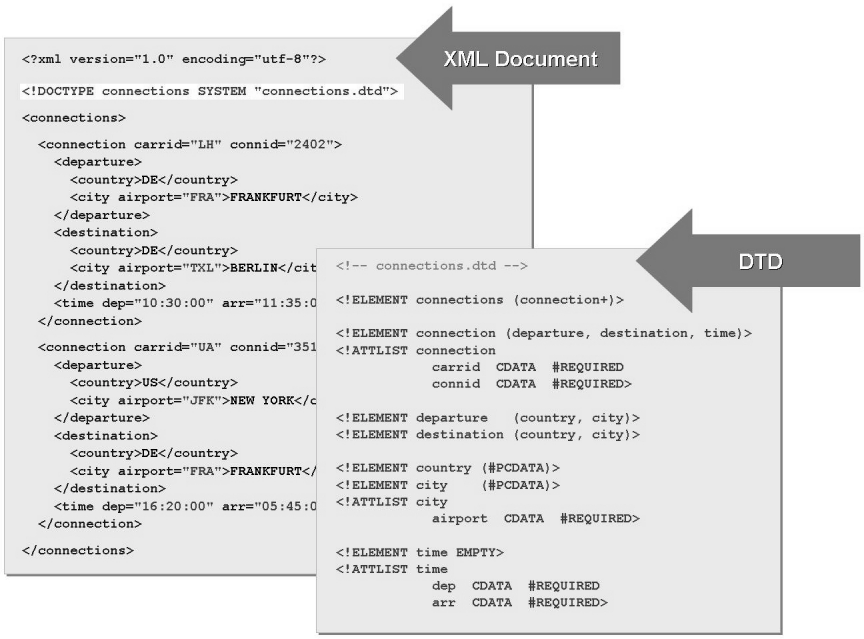


Figure 29: XML Document with DTD

DTDs have some disadvantages.

- Own syntax is used.
- Validity cannot be verified.
- Frequency cannot be specified concretely.
- Few content types are available for elements and attributes.

XML Schema

XML schemas are a more convenient substitution for DTDs. They are XML documents, and they make it possible, for example, to describe the text content and text length of different elements. There are many predefined content types, such as date types.

The syntax within the schema is based on XML and it is called XML Schema Description language (XSD). Accordingly, *.xsd is used as the associated file extension. The “schemaLocation” or “noNamespaceSchemaLocation” attributes are used to establish a link to the schema in the XML root element within the XML document: `<XMLRootNode xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="schema01.xsd">`

Generally, `<xsd:schema>` is used as the root element in the schema document. Here, the elements are defined with `<xsd:element name="elementname" type="datatype">` and the attributes are defined with `<xsd:attribute name="attributename" type="datatype">`.

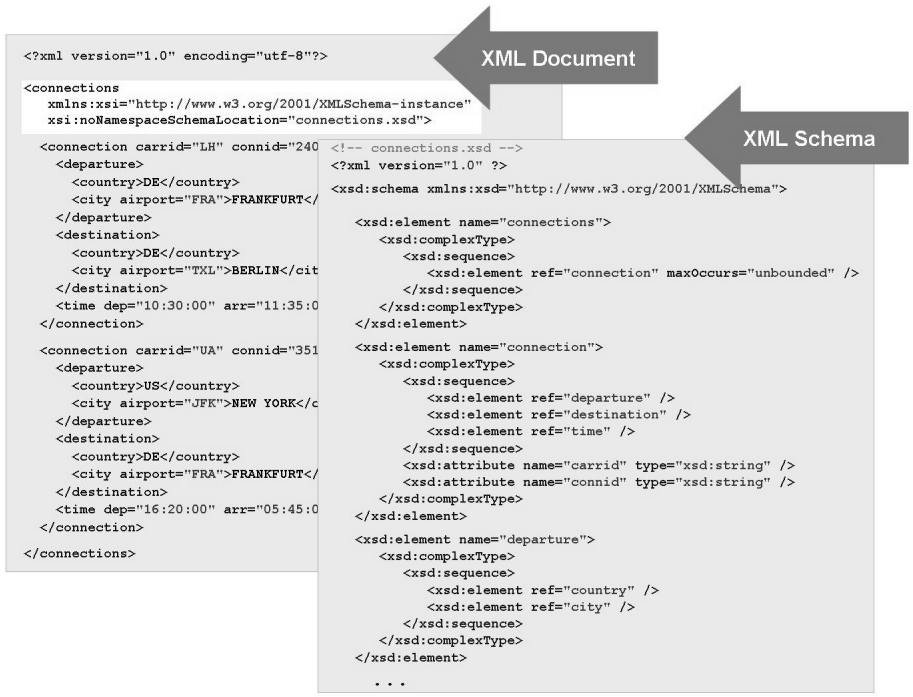


Figure 30: XML Document with XML Schema

Validity and Well-Formed XML Documents

A **well-formed** XML document complies with the following syntax rules:

- The document begins with an XML declaration.
- A root element is defined.
- All elements have an opening and a closing tag.
- All attribute values are placed between quotation marks.
- The element groups are nested correctly.

If these rules are not adhered to, this XML document is not well-formed. Most XML editors check whether a document is well-formed.

A DTD or XML schema that defines the rules for creating the XML document is developed to determine which elements can be used where, and how often, in a document. Parsers can be used to check whether the XML document complies with these rules. An XML document is **valid** if it complies with the following:

- The XML document is well-formed.
- It contains a DTD or an XML schema.
- The rules of the DTD or the XML schema are adhered to.

XML Namespaces

User-defined tags can be used in XML documents. To ensure the uniqueness of these tags, individual tags can be assigned to a specific namespace. Every namespace is uniquely defined by a Uniform Resource Identifier (URI). A URL can be used as a URI, for example. The protocol (http://) does not need to be specified for the URI, and the URI must not point at a document. Namespaces must be unique; therefore, Internet addresses are nearly always used with URIs for XML applications, as these addresses are unique, thus ensuring that the namespace is unique.

If a tag belongs to a particular namespace, this namespace must also be assigned to the tag. The *xmlns* attribute, which contains the namespace as an attribute value, is therefore added to the tag. The namespace is then also valid for all child elements of this tag. If the attribute is added to the root element, the namespace is valid for the entire document.

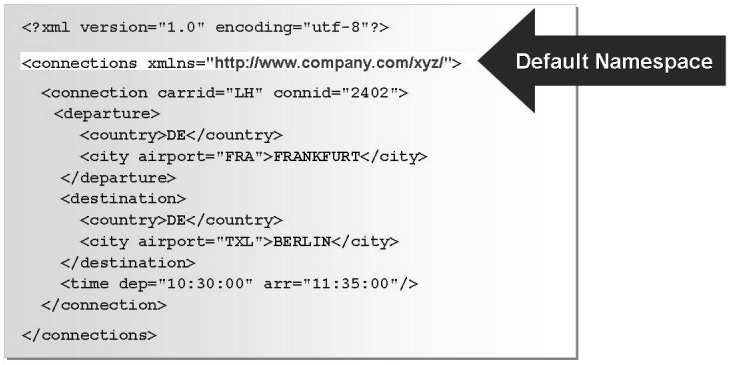


Figure 31: XML Namespaces: Default Namespaces

This type of namespace specification works as long as elements from different namespaces are not mixed with each other. If elements from different namespaces are mixed, it makes more sense to use qualified names, as shown in the example below.

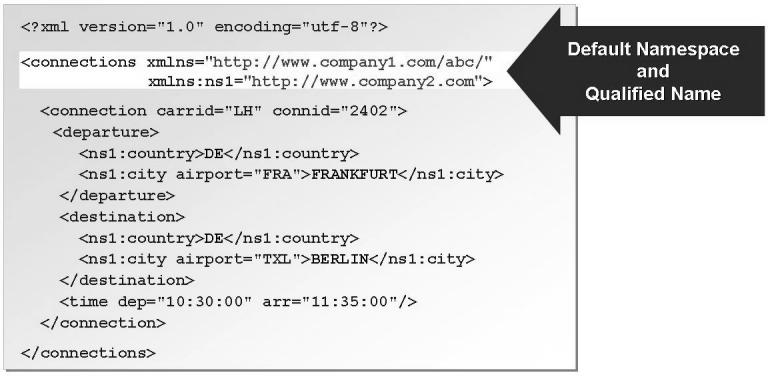


Figure 32: XML Namespaces: Default Namespaces and Qualified Names

Transformation of XML Documents

Stylesheets can be used to make the data of an XML document **visible**. This data can be prepared for the World Wide Web with Cascading Stylesheets (CSS). Better still, however, is the use of the eXtensible Stylesheet Language (XSL). This is based on XML and not only can it be used for displaying information on the Internet, but also for creating the most diverse output formats. For example, an XML structure can be transferred to another structure. It is also possible to convert to HTML or PDF as the target format. Such conversions of XML structures to other structures are considered transformations. Transformation programs (parsers), which can be used for the most diverse XML documents, are an advantage. A control file, such as an XSL stylesheet, is used to define the conversion rules for the elements of the XML file. The process of converting to the intermediate document is referred to as XSL Transform (XSLT).

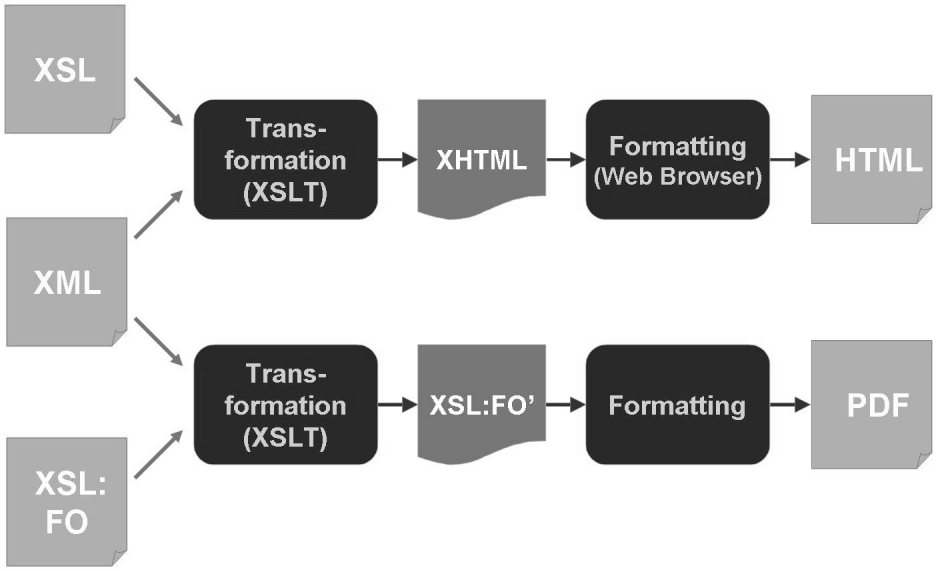


Figure 33: Transformations

Using XML allows you to clearly separate the content from the output form. Accordingly, an unlimited number of output documents can be produced for one and the same outgoing document through the use of different transformations.

The nested tags enable you to quickly describe a path through the document: `connections/connection/departure/country`To access specific elements and their content using the XSL stylesheets during transformations, you use the **XPath** language. XPath is a notation that specifies how to search for parts of a document.Examples of an XML document and an XSL stylesheet are illustrated in the following figure. The XSL stylesheet contains XPath statements for accessing elements of the source XML document. You can also see how the resulting document would appear in a Web browser.

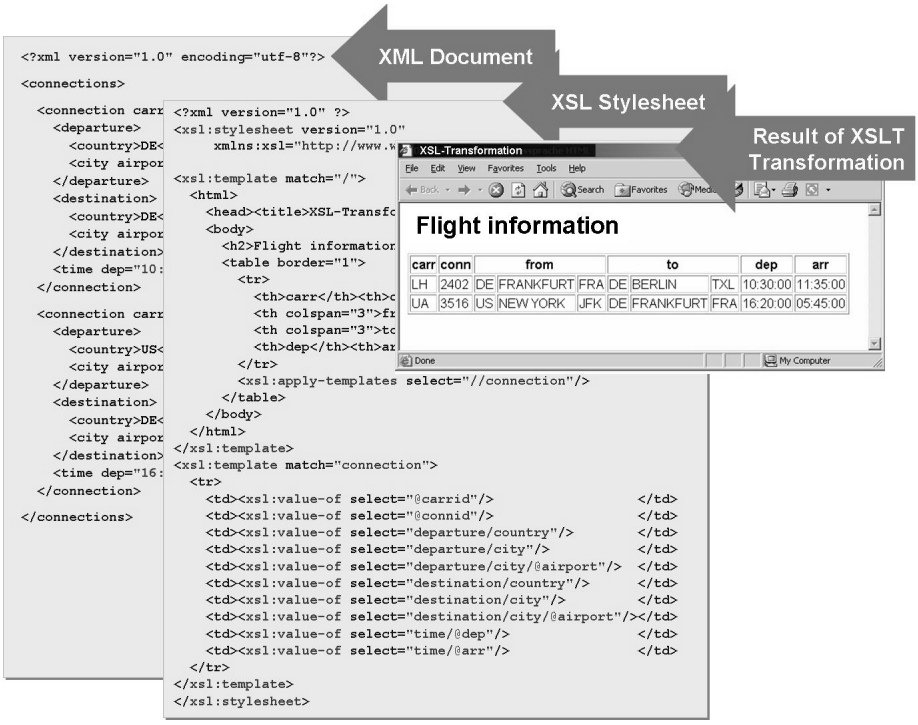


Figure 34: XSLT Transformation

An **XML processor** is a piece of software that allows you to read XML documents. This software provides access to the structure and contents of the XML document.

Hint: You can find more information about XSL and XSLT on the Internet. See for example http://www.w3.org/TR/xsl/ or http://www.w3.org/TR/xslt

Exercise 1: XML (Optional)

Exercise Objectives

After completing this exercise, you will be able to:

- Explain how XML schema definitions are used
- Use the *Cooktop editor* to check the validity of XML documents

Business Example

You receive data in an XML SOAP request. The XML parser terminates document conversion because it finds differences between the XML schema definition and the transferred XML document. Your task is to determine the cause of the problem.

Task:

Check the validity of an XML document.

1. Extract the *XML_Exercise.zip* file contained in the *BC416XML* BSP application to your local temp directory. Open the two XML exercise documents *connections.xml* and *connections.xsd* in the *Cooktop editor*. Familiarize yourself with the XML schema definition *connections.xsd*.
2. Check the settings to see which parser has been set. Use the MSXSL 4.0 parser. (Other parsers may not be suitable for the exercises because they may not query the errors included, for example.)
3. Validate *connections.xml* by clicking the red checkmark. Correct the errors that are displayed.

Solution 1: XML (Optional)

Task:

Check the validity of an XML document.

1. Extract the *XML_Exercise.zip* file contained in the *BC416XML* BSP application to your local temp directory. Open the two XML exercise documents *connections.xml* and *connections.xsd* in the *Cooktop editor*. Familiarize yourself with the XML schema definition *connections.xsd*.

   a) Ask your instructor if you have any questions.

2. Check the settings to see which parser has been set. Use the MSXSL 4.0 parser. (Other parsers may not be suitable for the exercises because they may not query the errors included, for example.)

   a) Ask your instructor if you have any questions.

Continued on next page

3. Validate *connections.xml* by clicking the red checkmark. Correct the errors that are displayed.

   a) Error 1: Add a *time* tag to line 15 of the XML document, for example `<time dep="15:20:00" arr="16:45:00"/>`. A time entry is mandatory.

   Error 2: Delete the duplicate *departure* tag. This is not permitted according to the current XML schema definition.

```
<?xml version="1.0" encoding="utf-8"?>

<connections xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:noNamespaceSchemaLocation="connections.xsd">

  <connection carrid="LH" connid="2402">
    <departure>
      <country>DE</country>
      <city airport="FRA">FRANKFURT</city>
    </departure>
    <destination>
      <country>DE</country>
      <city airport="TXL">BERLIN</city>
    </destination>
    <time dep="10:30:00" arr="11:35:00"/>
  </connection>

  <connection carrid="UA" connid="3516">
    <departure>
      <country>US</country>
      <city airport="JFK">NEW YORK</city>
    </departure>
    <destination>
      <country>DE</country>
      <city airport="FRA">FRANKFURT</city>
    </destination>
    <time dep="16:20:00" arr="05:45:00"/>
  </connection>

</connections>
```

Lesson Summary

You should now be able to:

- Describe the structure of an XML file
- Explain the difference between a DTD and an XML schema
- Describe how XSL programs are used

Lesson: SOAP: Basics

Lesson Overview

This lesson provides an introduction to the subject of SOAP.

Lesson Objectives

After completing this lesson, you will be able to:

- Describe the structure of a SOAP message
- Describe SOAP exceptions (faults)

Business Example

You want to understand how SOAP works with the HTTP transport protocol.

Introduction to SOAP

SOAP is a transmission and packaging protocol standardized by the World Wide Web Consortium (W3C). This protocol is used to exchange structured and typed information and messages based on XML between applications in a decentralized, distributed environment such as the Internet.

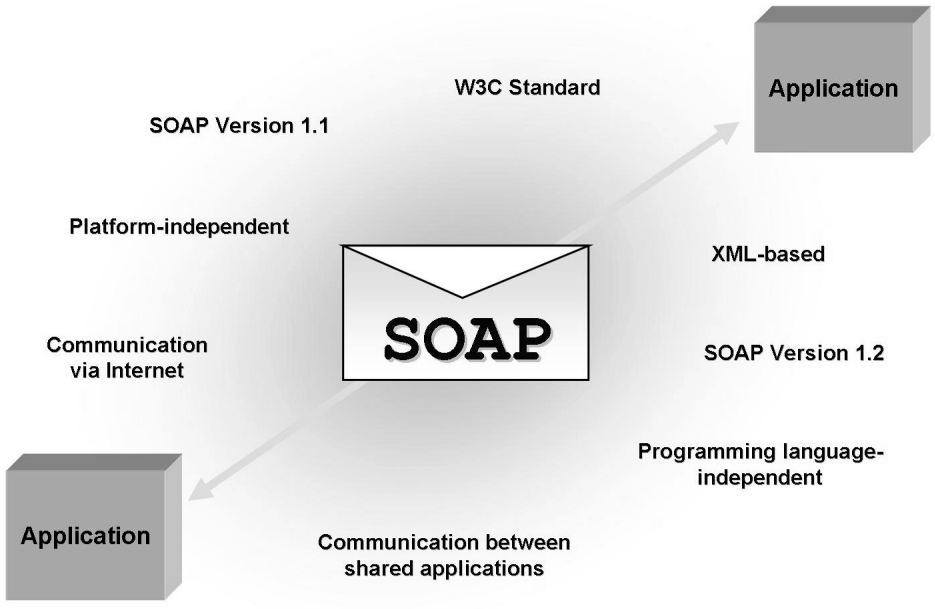


Figure 35: What is SOAP?

No particular transport protocol is specified for SOAP. For the most part, however, SOAP is connected to the HTTP transport protocol. This allows your applications to communicate with remote systems using standard Internet protocols and to execute components on other systems. Another advantage of SOAP is that it is not dependent on a particular platform. SOAP is accepted and supported by all major software manufacturers.

When SOAP is used with HTTP, the content type of the HTTP message must be set to *text/xml* and an additional line containing *SOAPAction* must be present in the HTTP header. The SOAPAction entry enables firewalls to identify SOAP messages and let them through without checking the XML data being transmitted. The name of the service to which the message is to be transmitted is used as the value for SOAPAction.

SOAP now stands for SOAP!The W3C working group decided to adopt the name SOAP for the specification *SOAP v1.2* as well. In the new version, however, SOAP is no longer used as an acronym for *Simple Object Access Protocol*, but as an independent name.

The Structure of a SOAP Message

A SOAP message consists of an optional SOAP header and a mandatory SOAP body. Both elements are incorporated into the SOAP envelope. The information contained within the blocks in the SOAP header provides details about how the SOAP message should be processed. You can find information about the required message routing or about authentication or authorization, for example. The SOAP body contains the actual user data of the message (payload) in XML format.

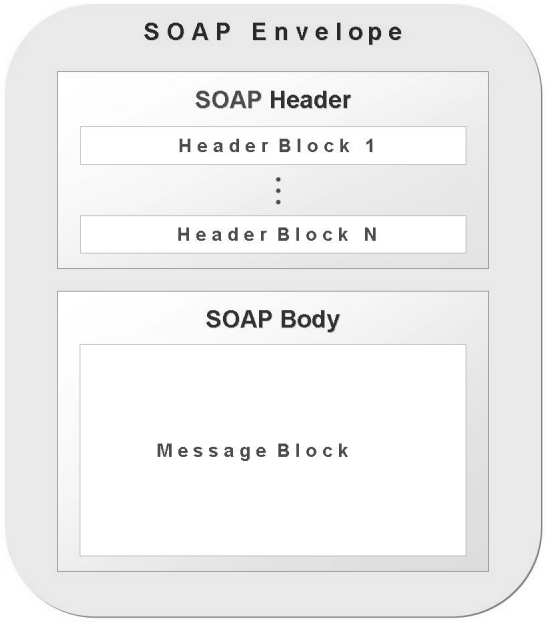


Figure 36: SOAP Message Structure

The following example illustrates the main components of a SOAP message. In the envelope root element, the namespace for formulating a SOAP message is defined using the *xmlns* attribute. The attribute points to the underlying XML schema. If the SOAP Version 1.1 specification is used for the SOAP message, the attribute value is set to *http://schemas.xmlsoap.org/soap/envelope*. A SOAP Version 1.2 message contains the entry *http://www.w3.org/2003/05/soap-envelope* as the attribute value:

```
<!-- SOAP Version 1.1 -->
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
   ...
</s:Envelope>

<!-- SOAP Version 1.2 -->
<s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope/">
   ...
</s:Envelope>
```

XML namespaces are an easy way to assign unique element and attribute names in XML documents. Element and attribute names are linked with namespaces by URI references.

```
<?xml version="1.0" ?>
<soap:Envelope
    xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
    soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">
  <soap:Header>
    <h:transaction xmlns:h="http://some-uri.org/"
                   soap:mustUnderstand="true">
      <h:transactionID> 12345 </h:transactionID>
    </h:Transaction>
  </soap:Header>
  <soap:Body>
    <m:GetOrderList xmlns:m="http://some-uri.org/">
      <m:OrderNumber> order-nr-x </m:OrderNumber>
    </m:GetOrderList>
  </soap:Body>
</soap:Envelope>
```

Figure 37: SOAP Request

In the SOAP header and body, you can insert any valid, well-formed, and namespace-conformant XML code. The SOAP specification merely stipulates that there can be only one SOAP header, which must precede the mandatory SOAP body and must be included in the SOAP envelope with the SOAP body.

mustUnderstand

If a header block contains a *mustUnderstand* attribute and the value of this attribute is set to *true* or *1*, the message receiver must know how the information contained in this header block is to be interpreted. Otherwise, the receiver must reject the whole message and describe the error using a *SOAP Fault*.

encodingStyle

The *encodingStyle* attribute allows for encoding-style rules in every element. These rules apply in this element and in all child elements that do not have their own *encodingStyle* attribute. Encoding style implies a batch of rules that define precisely how data types of an application are to be encoded in a generally binding XML syntax. This set of rules is used to define how different applications can exchange data with each other, even if they do not have any data types or data representations in common.

The SOAP Communication Model

One-way transmission of messages from a sender to a receiver is described using the SOAP communication model. These messages can also pass through different intermediate stations (known as *agents* or *actors*) on different computers or applications that handle the messages themselves and forward them to the next node. Accordingly, the messages also contain information for the different agents to process. The route covered by a message is known as a *message path*.

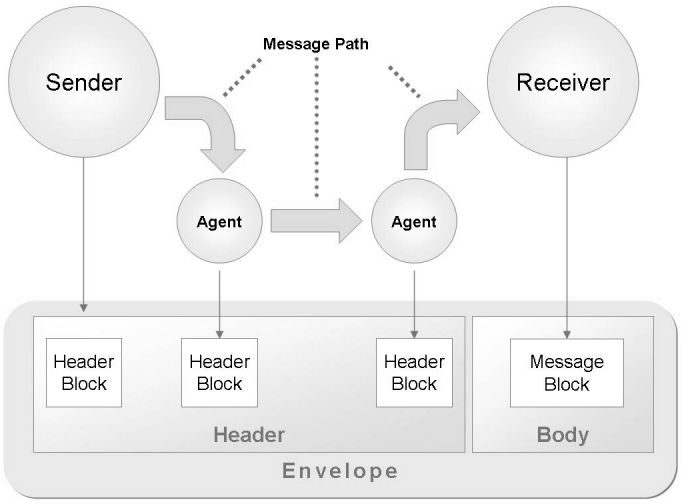


Figure 38: The SOAP Communication Model

Using a special mechanism, SOAP specifies which parts of a SOAP message are to be processed by which agent. This mechanism is also referred to as “targeting” and can be used only on header blocks. The SOAP body is only for the end node within the message path and cannot be assigned to intermediate nodes. Every agent deletes all the header elements intended for that agent from the message before forwarding it.

The SOAP specification does not provide any mechanism for defining the *message path*. You can define only which message is intended for which agent. The processing sequence cannot be defined. This issue can be worked around by using the Microsoft *SOAP Routing Protocol* (WS Routing).

SOAP Faults for Error Handling

If errors occur when a SOAP message is being processed, the errors are shown in the SOAP body. To include this processing error in the SOAP body, the SOAP specification defines the **SOAP fault** element.

```
<?xml version="1.0" ?>
<soap:Envelope
    xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <soap:Body>
    <soap:Fault>
      <faultcode> soap:Server </faultcode>
      <faultstring> Server Error </faultstring>
    </soap:Fault >
  </soap:Body>
</soap:Envelope>
```

Figure 39: Example of a SOAP Fault

If a processing error occurs, a fault element is inserted as a direct child element in the body of the SOAP message. The fault element can be included in the SOAP body only once and can contain the following child elements:

<faultcode>

Namespace-conformant identification of the error that occurred. SOAP defines some standard error codes (SOAP fault codes) that can be used in this element to describe errors:

VersionMismatch: An invalid namespace was found in the SOAP envelope.

MustUnderstand: A header block with the *mustUnderstand="true"* attribute was not understood by the message receiver. Using the optional standard header block *Misunderstood*, the SOAP fault provides concrete information about which header blocks were not understood in the message that was received.

Client: Invalid message or authentication missing, for example.

Server: A processing error occurred on the server side.

<faultstring>

Short, readable description of the processing error in more detail.

<faultactor>

This element contains the unique identifier (URI) of the message processing node at which the error occurred, if it is not the target node.

<detail>

This element is designed to contain information about application-specific errors that occurred during processing of the actual message (payload) in the SOAP body. Errors that are to be assigned to the header must be described in the error message header.

Fault entries that must conform to namespace conventions can also be added.

Remote Procedure Call with SOAP

The Remote Procedure Call (RPC) is a widely used SOAP application. The method call takes place in the body of the SOAP request; the return values of the method call are encoded in the SOAP response.

The method call is encoded as follows:

- The method call is modeled as a structure in the SOAP body, and the structure receives the name of the method. Every interface parameter of this method is a child element of this structure.
- The names and sequence of the interface parameters must match the names and sequence of the parameters for the method called. The individual interface parameters must be of the same type.
- Information that does not belong to the actual method parameters can be encoded in the SOAP header elements. The session ID or authentication details could be encoded here, for example.

```
<soap-env:Envelope
        xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/">
  <soap-env:Header>
    <!-- Session ID                              -->
    ...
    <!- Information for authentication -->
    ...
  </soap-env:Header>
  <soap-env:Body>
    <nr1:BC416_VACATION_CHECKER
          xmlns:nr1="urn:sap-com:document:sap:rfc:functions">
      <EMPLOYED_YEARS> 4 </EMPLOYED_YEARS>
    </nr1:BC416_VACATION_CHECKER>
  </soap-env:Body>
</soap-env:Envelope>
```

Figure 40: RPC with SOAP: Request

The result of the method call is also sent back to the client within a SOAP message:

- The result of the method call, accordingly, is modeled as a structure in the SOAP body, as above.
- If an error occurs during processing of the method, a SOAP fault element is returned instead of the result structure. In addition, the error handling procedure of the transport protocol is used. If, for example, the HTTP protocol is used as the transport protocol, a corresponding HTTP error code is included in the HTTP response.

```
<soap-env:Envelope
        xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/">
  <soap-env:Body>
    <n0:BC416_VACATION_CHECKERResponse
          xmlns:n0="urn:sap-com:document:sap:rfc:functions">
      <VACATION_DAYS> 20 </VACATION_DAYS>
    </n0:BC416_VACATION_CHECKERResponse>
  </soap-env:Body>
</soap-env:Envelope>
```

Figure 41: RPC with SOAP: Response

Categorizing Technologies

This section contrasts the individual technologies and categorizes them accordingly:

Traditional Program-to-Program Communication

The different horizontal layers represent levels of abstraction. The first layer features a binary transport protocol. In the SAP environment, the RFC protocol, for example, is located in the first layer. As additional levels of abstraction, you have data representation by data types such as string or integer that abstract the individual byte streams. The next level features support from programming languages, and the actual program is located on the uppermost level. Programs that want to communicate using this approach must communicate across all levels using the same protocol.

XML

If you are working with XML and want to exchange XML messages with other programs over the Internet, you can use the HTTP protocol on the transport layer. The programs can then access the XML messages by using the DOM (Document Object Model), for example. One advantage of this approach is that the technologies used involve Internet standards that are widely supported by software manufacturers. Using these standards makes technical communication easier and more standardized, and takes things one step closer to achieving interoperability on a technical level.

SOAP

SOAP takes a central position in relation to these technologies. Using the SOAP protocol, an attempt is made to create protocol properties based on XML. The SOAP header can model enhancements that describe the quality of the transport and that extend beyond the actual application data (payload) in the SOAP body. The SOAP request is interpreted semantically, and functions such as a function module, for example, are processed on the server. An XML document is received as a response; this is the SOAP response.

If an interface description (WSDL document) still exists for this server functionality, which describes the actual functionality and the associated data types, this is closer to the first approach but still has the advantages of XML. The interface description facilitates the creation of proxies on both the client and server sides. These proxies then encapsulate the actual call, so that you do not have to handle the SOAP call.

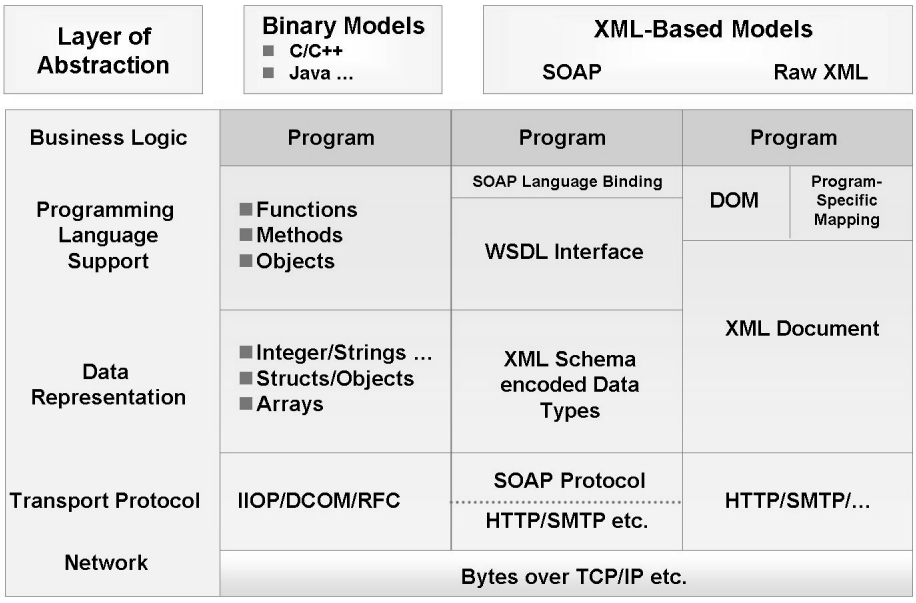


Figure 42: Categorizing Technologies

SOAP @ SAP

On the *SAP NetWeaver Application Server*, the SOAP runtime provides a mechanism that allows RFC-enabled function modules to be accessed or called as Web services using the SOAP protocol across HTTP.

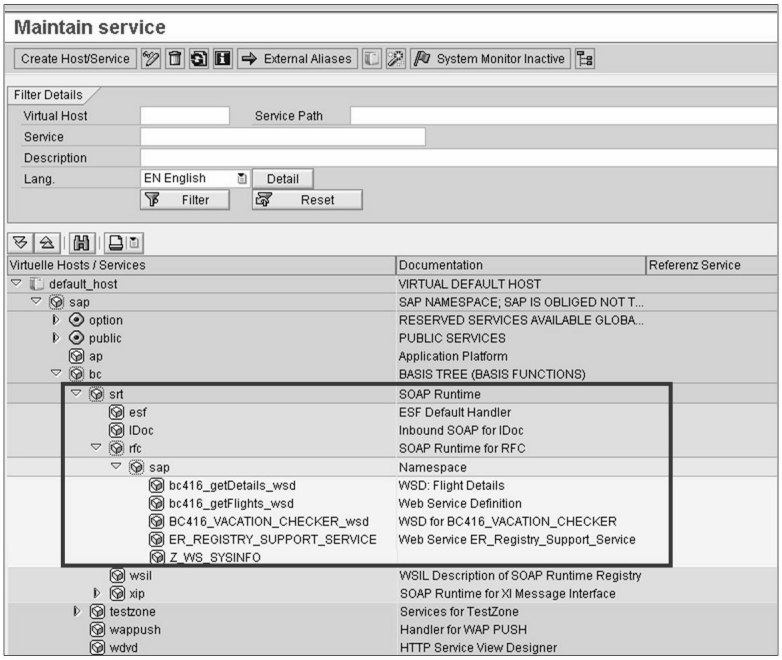


Figure 43: SOAP Runtime in ICF

The SOAP runtime on the *SAP NetWeaver Application Server* does not represent a fully generic framework for SOAP processing. It provides an alternative mechanism for the synchronous RFC instead: Synchronous RFC calls, which are otherwise implemented on the basis of the RFC Engine in the *SAP NetWeaver Application Server* kernel or the external RFC library, can be substituted by Internet mechanisms (XML, SOAP, HTTP). This means that the SAP RFC protocol can be avoided. This facilitates the connection between the *SAP NetWeaver Application Server* and non-SAP systems in particular. Frequently, a SOAP mechanism is available directly or by way of a wrapper. A special RFC-SDK-based program may not apply. The special variants of RFC (asynchronous, transactional, or queued) are actually not supported by SOAP.

The consistent orientation toward RFC call types requires some restrictions, namely those that limit the uses of the SOAP runtime on the *SAP NetWeaver Application Server* as a general Web service platform and, especially, the interoperability with other platforms. For the SOAP runtime restrictions, see the online documentation.

Lesson Summary

You should now be able to:

- Describe the structure of a SOAP message
- Describe SOAP exceptions (faults)

Internal Use SAP Partner Only

Unit Summary

You should now be able to:

- Describe the basics of RFC-enabled function modules
- Describe the TCP/IP reference model
- Describe HTTP as an application protocol of TCP/IP
- Describe the structure of an XML file
- Explain the difference between a DTD and an XML schema
- Describe how XSL programs are used
- Describe the structure of a SOAP message
- Describe SOAP exceptions (faults)

Unit 3

Web Services for SAP NetWeaver 7.0

Unit Overview

This unit introduces the ABAP Web service technology for *SAP NetWeaver 7.0*. You will learn how to create an ABAP Web service based on an RFC-enabled function module using the *Service Definition Wizard*. The publishing of Web service descriptions on a UDDI repository is also dealt with in this unit.

Unit Objectives

After completing this unit, you will be able to:

- Classify the ICF in the SAP Web AS ABAP.
- Use transaction SICF to browse ABAP HTTP services.
- Create an ABAP Web service based on an RFC-enabled function module.
- Describe the functions of the *service interface* and the *variant*.
- Describe the *Web service homepage*.
- Describe the structure of a WSDL document
- Discuss the importance of WSDL in the Web services environment
- Generate a proxy using a WSDL document
- Define a logical port
- Call a Web service from an ABAP program
- Perform a connection test for an RFC destination
- Carry out a trace record to trace an SOAP request
- Use the ICF Recorder to record request and response cycles
- Understand the core meaning of UDDI in the Web service environment.
- Name the possible scenarios in which UDDI can be used in the ABAP Web service environment.
- Name the general technical conditions for using UDDI in local or public environments.

Unit Contents

Internal Use SAP Partner Only

Lesson: Introduction to the Internet Control Framework

Lesson Overview

Introduction to Internet Control Framework (ICF)

Lesson Objectives

After completing this lesson, you will be able to:

- Classify the ICF in the SAP Web AS ABAP.
- Use transaction SICF to browse ABAP HTTP services.

Business Example

You want to understand how HTTP calls to SAP ABAP system are processed.

System Architecture

A classic *SAP R/3* system is implemented as a three layer client/server architecture: Presentation layer, application layer, and database layer. SAP R/3 is scalable in terms of the presentation layer and the application layer. Good scalability is an important prerequisite for enabling a number of users to work simultaneously. The *SAP NetWeaver Application Server* is a further development to the classic client/server technology. The SAP kernel has been extended to include a new process, the *Internet Communication Manager (ICM)*. This enables direct processing of requests that are created using a Web browser and HTTP protocol, for example, from the Internet or intranet.

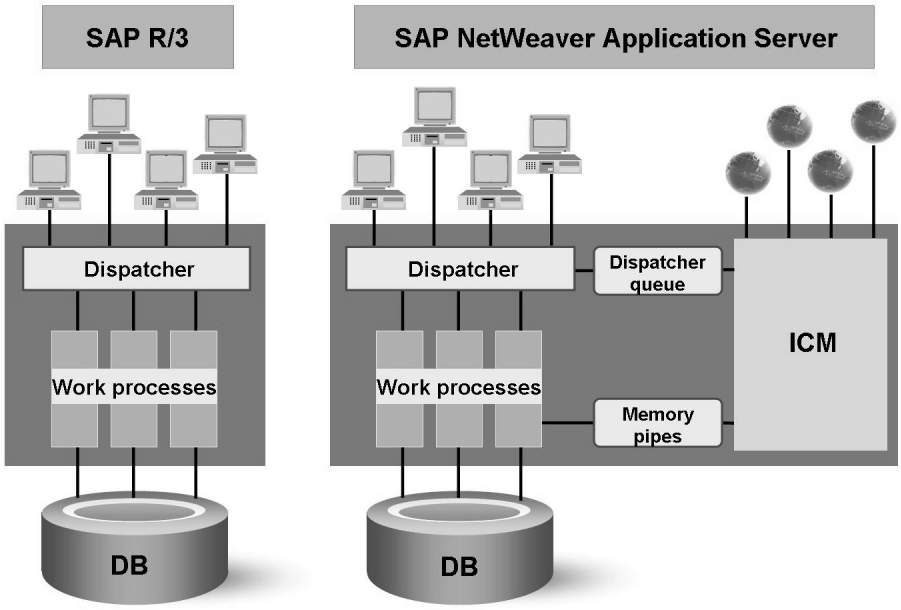


Figure 44: Internet Communication Manager in SAP NetWeaver AS ABAP

Internet Communication Framework

The *Internet Communication Framework (ICF)* provides the environment for handling an HTTP request in a work process of a SAP System (server and client). The ICF consists of ABAP classes and interfaces whose basis objects can be instanced and which, in turn, allow access to the request and response data. The IF_HTTP_SERVER interface for servers and the IF_HTTP_CLIENT interface for clients are of central significance here. The following describes how a simplified HTTP request/response cycle is processed (interaction model).

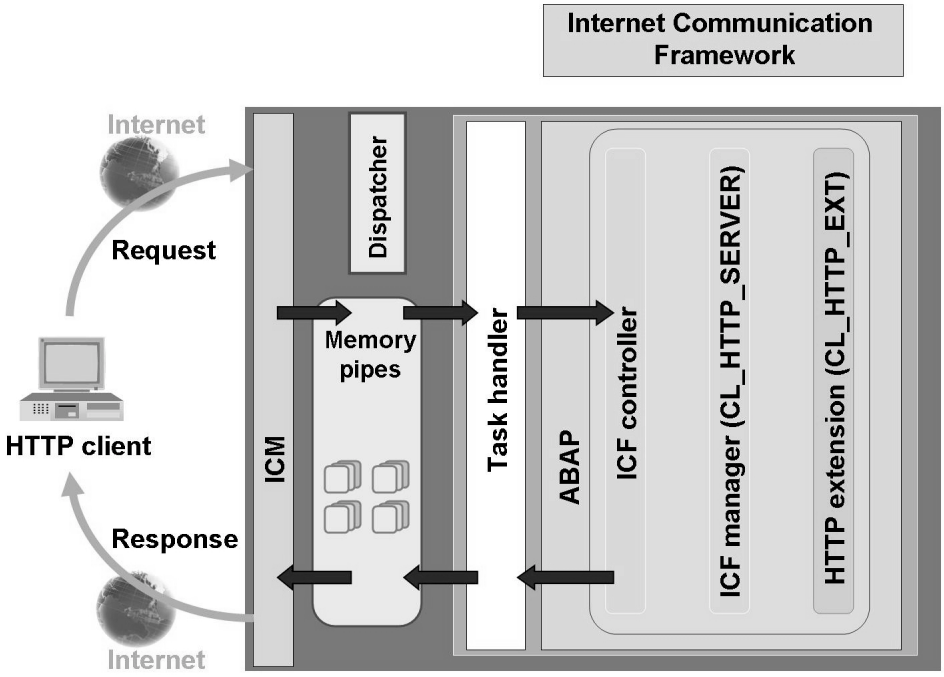


Figure 45: The Internet Communication Framework (ICF)

The above figure shows how a HTTP request/response cycle is processed.

- Accept the HTTP request via the ICM
- Process the request via the SAP system handler (based on the URL prefixes)
- Call the function module HTTP_DISPATCH_REQUEST (ICF Controller)
- Create a server object using the function module (ICF Manager). This involves an instance of the CL_HTTP_SERVER class
- Transfer the request data from the memory pipes to the server object attributes
- Select the HTTP request handler: this is determined by the structure of the URL using the HTTP service tree

  Note: SAP provides the CL_HTTP_EXT_BSP class for starting BSP application servers.

- Client logon
- Processing the HTTP request handler: the request handler can access the request data via the server object and define the response data
- Return control to the ICF controller, which may call further handlers depending on the definitions in the HTTP service tree
- Generate the HTTP response and write the data back to the memory pipes
- Send back the data via the ICM using the SAP system handler

The HTTP Service Tree

HTTP request handlers are defined using the Class Builder in the ABAP Workbench. The handler class must implement the HANDLE_REQUEST method of the IF_HTTP_EXTENSION interface here, as this method is called by the server control block. The server object can then be accessed from the HANDLE_REQUEST method. The URL determines which of the request handlers is called by the server control block. Transaction SICF is used to assign request handlers to URLs. A number of virtual Web servers can then be defined that listen to different IP addresses, alias names (host header names), and/or ports. Requests whose URLs do not match any of the installed virtual Web servers are accepted by the default host. A service tree whose node sequence correlates with the URL prefix can be created for each virtual Web server. Each node can represent either a service for which an HTTP request handler and other service-specific data can be stored (definition of logon procedures for starting the service, explicit response pages if errors occur, and so on), or it can represent a reference to an existing service (alias). The stored log on data is cumulated using the tree. Services (including all subnodes) can be activated and deactivated in ransaction SICF.

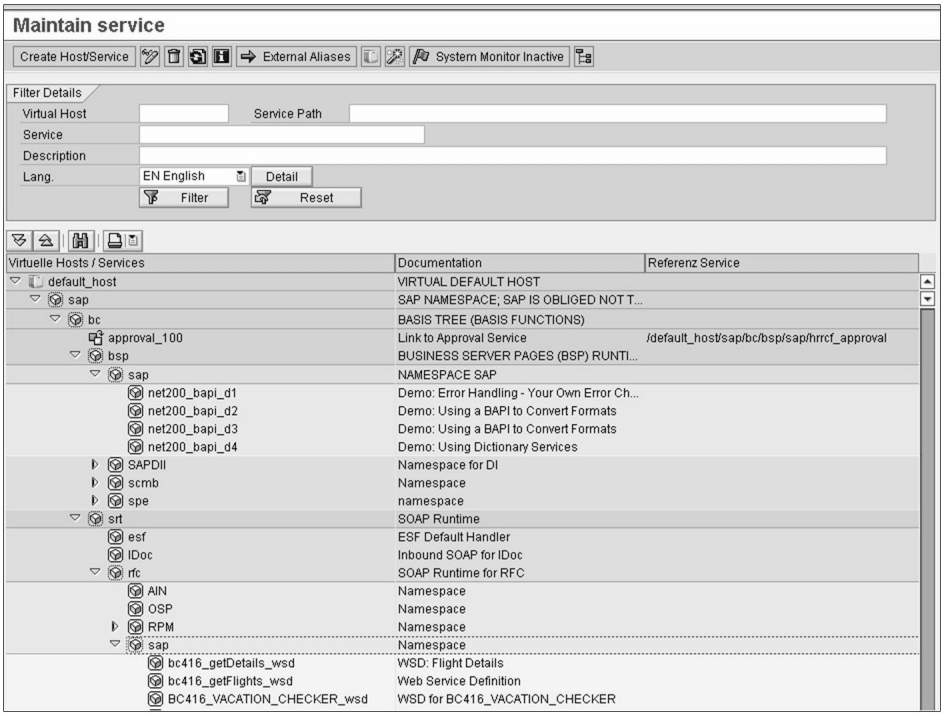


Figure 46: Transaction SICF: HTTP Service Tree

Lesson Summary

You should now be able to:

- Classify the ICF in the SAP Web AS ABAP.
- Use transaction SICF to browse ABAP HTTP services.

Internal Use SAP Partner Only

Lesson: SAP NetWeaver Application Server as a Web Service Server

Lesson Overview

This lesson provides an introduction to the topic of Web services, focusing on the *SAP NetWeaver Application Server* as the Web service server. The aim of this lesson is to create an ABAP Web service using the *ABAP Workbench*.

Lesson Objectives

After completing this lesson, you will be able to:

- Create an ABAP Web service based on an RFC-enabled function module.
- Describe the functions of the *service interface* and the *variant*.
- Describe the *Web service homepage*.

Business Example

You need to create an ABAP Web service using the *ABAP Workbench*.

Provision of Web Services – The Server Side

Any Web service starts with the business process itself and identified, standard implementations of business functionality. These can range from credit card checks and currency translations to payroll functions. Based on the standard interfaces of a business application, any self-contained, modularized functionality that is implemented as a BAPI, RFC-enabled function module, Enterprise JavaBean (session bean), or Java class lends itself to deployment as a Web service with the help of *SAP NetWeaver* technology. This functionality could be a component of a *SAP Business Suite* solution, or could be developed by SAP users, their customers, or their partners This section takes a look at how a Web service is created and registered on the server side.

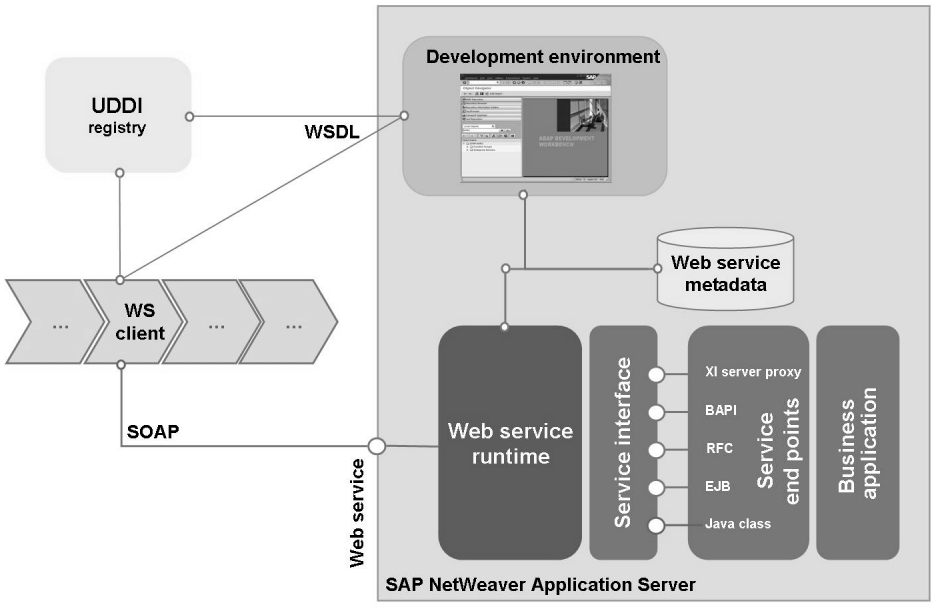


Figure 47: Provision of Web Services – The Server Side

The *ABAP Workbench* provides a wizard (*Service Definition Wizard*) to help you to create an ABAP Web service.

Since UDDI represents a cornerstone of the overall Web services architecture, it is quite natural that UDDI is natively supported by *SAP NetWeaver*. This will help SAP customers to seamlessly connect to any UDDI registry in order to publish Web services that are developed with the *SAP NetWeaver Application Server* and to integrate externally discovered Web services. The *SAP NetWeaver Application Server* can also be used to deploy a local UDDI registry. The *SAP NetWeaver Application Server* has its own UDDI implementation for this.

The *Service Definition Wizard*

Once a process and an interface that are to be made available have been identified, you can then create ABAP Web services with the *Service Creation Wizard* in the *ABAP Workbench*. With this wizard, configuring a Web service is a highly automated process based on predefined configuration profiles developed by SAP that bundle settings used in typical Web services scenarios. With this wizard, even a developer who is less experienced with detailed technical aspects of Web services can still create one simply by clicking through the three steps of the wizard. All of the required objects are generated in the background.

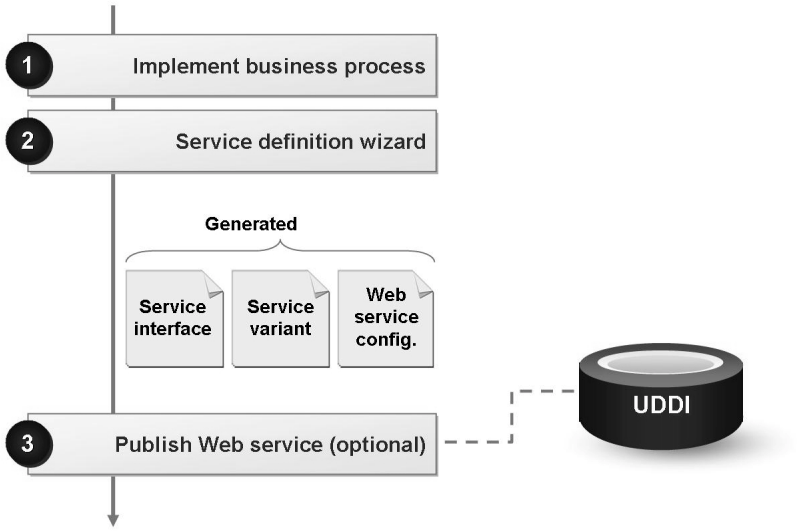


Figure 48: Service Definition Wizard: Overview

The following objects are generated by the *Web service creation wizard*:

***Service Interface* of a Web service**

A Service Interface is the representation of a Web service to the outside world. A Service Interface enables you to define a view of an implementation (Web service end point) and publish this view as a Web service. When you create a Service Interface, you can hide or rename operations and parameters. For example, you can replace technical names with descriptive names. You can also define default values for parameters. A Service Interface can currently be created for the following objects: RFC-enabled function module, function group (with RFC-enabled function modules), BAPI and XI Message Interface.

***Service Variant* and *configuration* of a Web service**

Features such as the communication type, authentication level, and transport properties are assigned in abstract form in the *Service Variant*. The technical details are specified during administration and the release of the Web service for the SOAP runtime. The assignment of abstract features ensures that a *Service Variant* can be used for different application servers that are configured differently. The proxy generation on the client side refers to the *Service Variant*. Technical details specified during configuration of the Web service are configured separately in the Web service client runtime.

Using the Service Definition Wizard

You can start the *Service Definition Wizard* within the *Object Navigator* (SE80) using the context menu for the package (*Create → Enterprise Service / Web Service → Web Service*). You have to maintain a name for the *service definition* in the wizard. You must also specify the end point type. Next, enter the name of the RFC-enabled function module as the Web service end point. Finally, select one of the available profiles for the predefined feature quantity for the service definition. You can choose from the following options:

Basic Auth SOAP Profile: The communication type is stateless, and the caller is verified by a user name and password.

Secure SOAP Profile: Again, the communication type is stateless. Authentication occurs via client certificates, and data is transmitted in encrypted form using the Secure Socket Layer protocol (SSL).

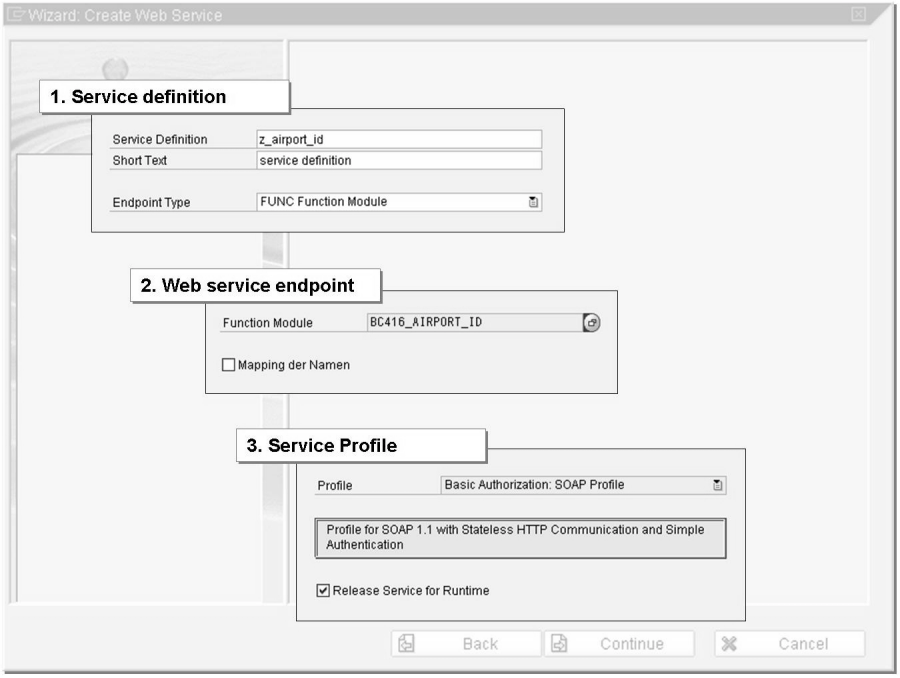


Figure 49: Service Definition Wizard

You can create the *service definition* using the *Service Definition Wizard* with just a few clicks of the mouse. You can release the SOAP runtime if you wish. In the object list in the navigation area of the *Object Navigator*, an entry is included for the generated objects.

For **function groups and function modules**, the *Service Creation Wizard* can also be called from the function library (SE37). Choose the required function module, display it, and from the menu select *Utilities* → *Extra Utilities* → *Create Web Service* → *From the Function Module* or *From the Function Group*. Note that function groups must contain at least one RFC-enabled function module.

Hint: To be able to create and integrate Web services, you need the *SAP_BC_WEBSERVICE_ADMIN* role authorizations.

Web Service Homepage

The *Web service homepage* offers resources for developing and utilizing Web services. You can use the homepage to test Web services. To call the homepage, you need to have a *SAP NetWeaver AS Java*. The *Web service homepage* acts as a Web service client and sends a SOAP document to the application server. The application server receives this SOAP message, extracts the input parameters, and executes the relevant Web service. The result is sent back to the *Web service homepage* via SOAP, and it is displayed there. You can also view the SOAP request and SOAP response on the *Web service homepage*.

You access the *Web service homepage* in transaction WSADMIN. Here, choose the *service definition* you require and start the *Web service homepage* by using the test icon on the application toolbar.

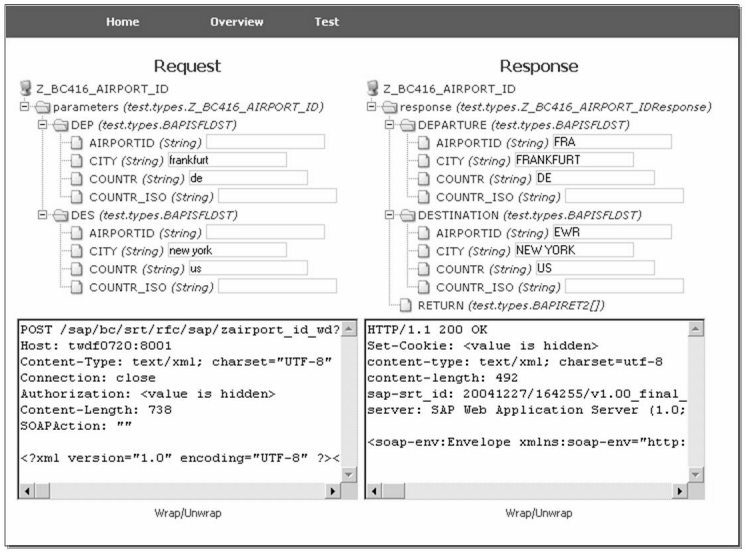


Figure 50: Web Service Homepage

Hint: In transaction WSADMIN, you can enter the address of the *SAP NetWeaver AS Java* by choosing *Goto → Administration Settings*. The format of this address must be as follows: `<http(s)>://<JavaServerHost>:<JavaServerPort>`. The *SAP NetWeaver AS Java* provides access to the *Web service homepage*, with different ways of testing the Web service.

Adapting the Service Interface

The following changes can be made on the ***Service Interface*** **tab page** of the service definition:

- **Change the Operation Name** Select the name of the operation whose name you want to change. In the *Name* field, enter the required operation name. If necessary, change the namespace.
- **Customize Parameters** Select the required parameter of the operation whose parameter name you want to change. Enter the required parameter name in the *Parameter(s)* field. You can also change the data type of a parameter if this is based on a structured type or on a table type. The data type must first be defined on the *Types* tab page.
- **Define Default Values** Parameters that are optional for the original interface can also be optional for the service interface. In this case, the default value of the original interface is used. If you wish, you can assign your own default value instead.
- **Hide Parameters** The *Exposed* checkbox allows you to hide parameters. To do this, remove the indicator in the *exposed* field. If you hide a parameter it will not appear in the WSDL document. In this case, the fixed value or the initial value of the parameter will be used.

You can change a parameter type on the ***Types*** **tab page**. Proceed as follows:

- Select the type you want to copy. Choose *Copy* from the context menu or the application toolbar.
- In the following dialog box, enter a name for the copied type. The copied type will now be displayed in the *Copied Types* category.

You can delete or rename the fields of the copied type using the appropriate pushbuttons or the context menu. If the copied type contains more fields that are based on a structure type or a table type, the context menu also contains the entry *Change Type*. In that case, choose a suitable type in the following dialog box. You may have copied this already. Select the *Interface* tab page to assign the new type.

You have now finished creating a virtual interface. Create the Web service definition next.

The following information is available on the ***Variants*** **tab page** of the service definition:

- Name of the service definition and service operation
- Path prefix in the Internet Communication Framework (ICF)

The following options are available when you select the service operation:

- **Session-Oriented Communication**. Tick the *Select Feature* checkbox if you want to define the Web service communication as stateful.
- **Authentication**. The following options can be selected here: *None*, *Basic* and *Strong*. Choose *Basic* if the execution of a Web service on the client side requires the user to be authenticated via a user name and password. Choose *Strong* if the user is to be authenticated using the Secure Socket Layer protocol (SSL) and X.509 client certificates. If no authentication is required, choose *None*.
- **Transport Guarantee**. The following options can be selected here: *None*, *Integrity*, *Confidentiality* and *Both*. Choose *Both* (Integrity + Confidentiality) if data being sent from Web service clients to the server via SOAP requests is to be encrypted. The Secure Socket Layer protocol (SSL) is used for this.

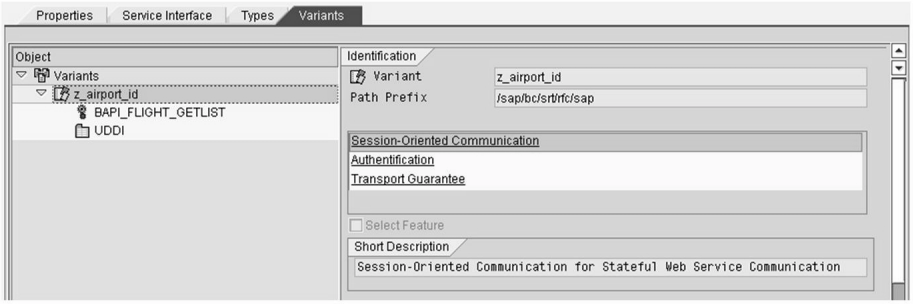


Figure 51: *Variants* Tab Page

You can publish the service definition as a tModel in the UDDI by way of the ***UDDI*** entry. To do this, choose the context menu and then *Publish*. In the dialog box that opens up, specify a UDDI registry, a user name, and a password in order to access the UDDI registry. You only need to provide a user name and password if you did not enter a default user when setting up the UDDI registry. The tModel key of the registered service definition is returned by the UDDI registry and transferred to the service definition. When you select *Change*, the browser for the UDDI client appears.

Release for SOAP runtime

Use transaction WSCONFIG to release service definitions for the SOAP runtime. The WSCONFIG release transaction reads the default settings - for example, for the security of data transmission from the service definition - and provides appropriate configuration options. In this respect, the values of the service definition serve as default values or for value restriction.

A Web service should be released by the Web service configurator. He or she knows the system landscape and the technical requirements of the application server where the Web service is to be called. The configurator assigns specific attributes to the abstract features chosen in the Web service definition.

To release the service definition for the SOAP runtime, start the release transaction WSCONFIG and enter the name of the Web service definition in the *Service Definition* field. Choose *Create*.

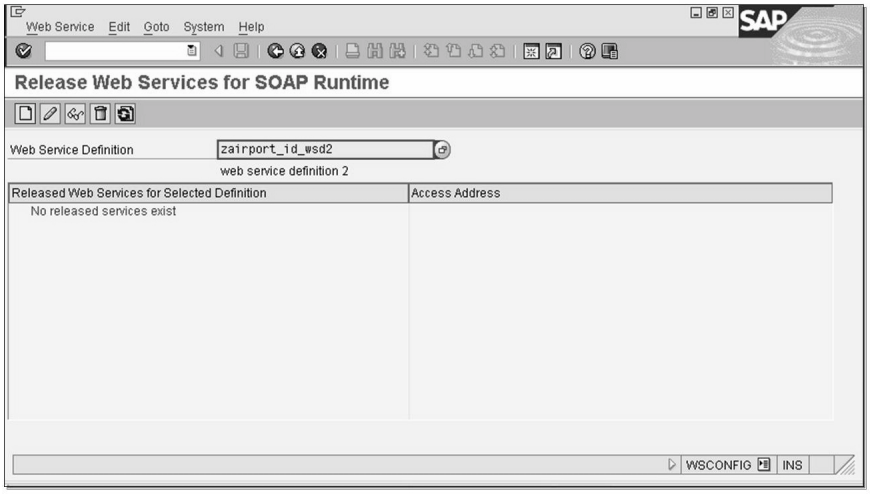


Figure 52: Transaction *WSCONFIG*: Release for SOAP Runtime

Internal Use SAP Partner Only

Internal Use SAP Partner Only

Exercise 2: Service Definition Wizard

Exercise Objectives

After completing this exercise, you will be able to:

- Create an ABAP Web service using the *Service Definition Wizard*
- Test an ABAP Web service via the *Web service homepage*.

Business Example

You need to activate and test a BAPI in an *SAP NetWeaver Application Server 7.0* as an ABAP Web service.

Task 1: Service Definition Wizard

From the *Flight* business object, the *GetList* BAPI should be activated as an ABAP Web service using the *Service Definition Wizard*.

1. Familiarize yourself with the BAPI interface and test the BAPI. Use the *BAPI Explorer* for this (transaction BAPI).

   What information does the BAPI provide you with?

   What application components does the business object belong to?

   _____

   _____

   _____

   _____

2. Create the ***zbc416_##*** package and assign it to the change request given to you by your course instructor.

   You should replace ## with your two-digit group number. This group number is available from your instructor.

   From now on, in all the following exercises, assign your repository objects to this package and this change request.

3. Use the *Service Definition Wizard* to generate a Web service based on the *GetList* BAPI of the *Flight* business object. This Web service should be stateless. The caller must be verified using a user name and password.

   Use the following identifiers for the service definition to be generated:

Continued on next page

Service definition: ***z##_getFlights***

You should replace ## with your two-digit group number. This group number is available from your instructor.

4. Check whether the service definition was generated.

Task 2: Web Service Homepage

The ABAP Web service generated using the *Service Definition Wizard* must now be tested via the *Web service homepage*. To do this, start the transaction WSADMIN (Web Service Administration) and select your ABAP Web service. Use the *Test* icon to go to the *Web service homepage*.

1. Which different WSDL styles are supported?

2. The URL of the Web service is http://<server>:8000/sap/bc/srt/rfc/sap/<web-service>?sap-client=100. What must be added to this URL in order to obtain the WSDL document?

Task 3: Local SOAP Client (Optional)

The generated ABAP Web service should be tested again using a local SOAP client.

1. In the *BC416SOAPClient* BSP application for the package BC416, the *Mimes* folder contains a SOAP Client in a ZIP file. Extract *SOAP_Client.zip* to the local *Temp* directory on your machine. Use *SOAP_Client.exe* to launch the SOAP client.

2. To call your Web service, send a SOAP request to the application server from the local SOAP client. In the SOAP client, specify the host and HTTP port of your application server, as well as the path to your Web service. You must also enter a valid SOAP request.

Continued on next page

Internal Use SAP Partner Only

Task 4: SAP SOAP Client Tool (Optional)

If you have a user in the *SAP Developer Network (SDN)*, you can also use the *SAP SOAP Client Tool* to transmit an SOAP request.

1. Use the hyperlinkhttp://www.sdn.sap.com to access the *SAP Developer Network (SDN)*. Use the search function and enter *SAP SOAP Client Tool* as the search term. The result will be transmitted in the form of a hyperlink that you can use to start the *SAP SOAP Client Tool*.

   Note that this requires Java Virtual Machine (JVM) and Java Web Start technology to be installed on your machine.

Internal Use SAP Partner Only

Solution 2: Service Definition Wizard

Task 1: Service Definition Wizard

From the *Flight* business object, the *GetList* BAPI should be activated as an ABAP Web service using the *Service Definition Wizard*.

1. Familiarize yourself with the BAPI interface and test the BAPI. Use the *BAPI Explorer* for this (transaction BAPI).

   What information does the BAPI provide you with?

   What application components does the business object belong to?

   Answer: Available flights can be determined using the BAPI. The flight selection can be restricted using the import parameters. A list of flights with the most important flight properties is generated.

   The business object is assigned to the CA application component.

2. Create the ***zbc416_##*** package and assign it to the change request given to you by your course instructor.

   You should replace ## with your two-digit group number. This group number is available from your instructor.

   From now on, in all the following exercises, assign your repository objects to this package and this change request.

   a) Ask your instructor if you have any questions.

3. Use the *Service Definition Wizard* to generate a Web service based on the *GetList* BAPI of the *Flight* business object. This Web service should be stateless. The caller must be verified using a user name and password.

   Use the following identifiers for the service definition to be generated:

   Service definition: ***z##_getFlights***

Continued on next page

You should replace ## with your two-digit group number. This group number is available from your instructor.

a) Proceed as illustrated in the figure below:

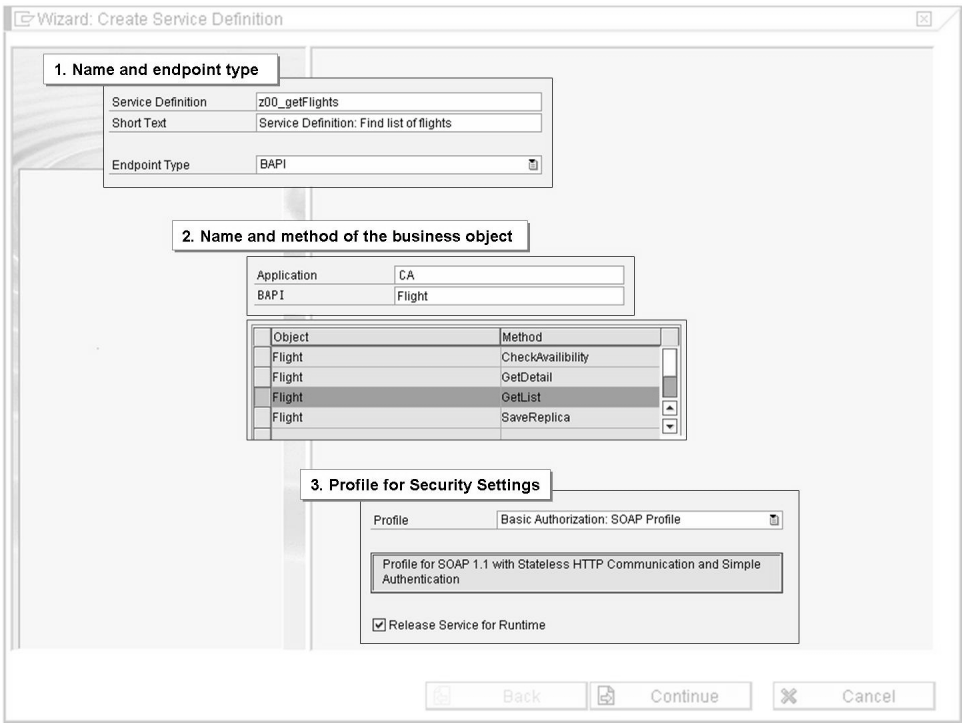


Figure 53: Web Service Creation Wizard

4. Check whether the service definition was generated.

   a) Look through the object list in the *Object Navigator* (SE80).

Task 2: Web Service Homepage

The ABAP Web service generated using the *Service Definition Wizard* must now be tested via the *Web service homepage*. To do this, start the transaction WSADMIN (Web Service Administration) and select your ABAP Web service. Use the *Test* icon to go to the *Web service homepage*.

1. Which different WSDL styles are supported?

   Answer: *Document style* and *RPC style*.

Continued on next page

Internal Use SAP Partner Only

2. The URL of the Web service is http://<server>:8000/sap/bc/srt/rfc/sap/<web-service>?sap-client=100. What must be added to this URL in order to obtain the WSDL document?

   Answer: Use the ?sap-client=100&wsdl=1.1 query string to get the WSDL document.

Task 3: Local SOAP Client (Optional)

The generated ABAP Web service should be tested again using a local SOAP client.

1. In the *BC416SOAPClient* BSP application for the package BC416, the *Mimes* folder contains a SOAP Client in a ZIP file. Extract *SOAP_Client.zip* to the local *Temp* directory on your machine. Use *SOAP_Client.exe* to launch the SOAP client.

   a) If you have questions, contact your instructor.

2. To call your Web service, send a SOAP request to the application server from the local SOAP client. In the SOAP client, specify the host and HTTP port of your application server, as well as the path to your Web service. You must also enter a valid SOAP request.

   a) You can determine the host, HTTP port, and path to your Web service using, for example, the URL of the WSDL document.

   b) You can determine the SOAP request for calling the Web service from, for example, the *Web service homepage*. To do so, test the Web service.

Continued on next page

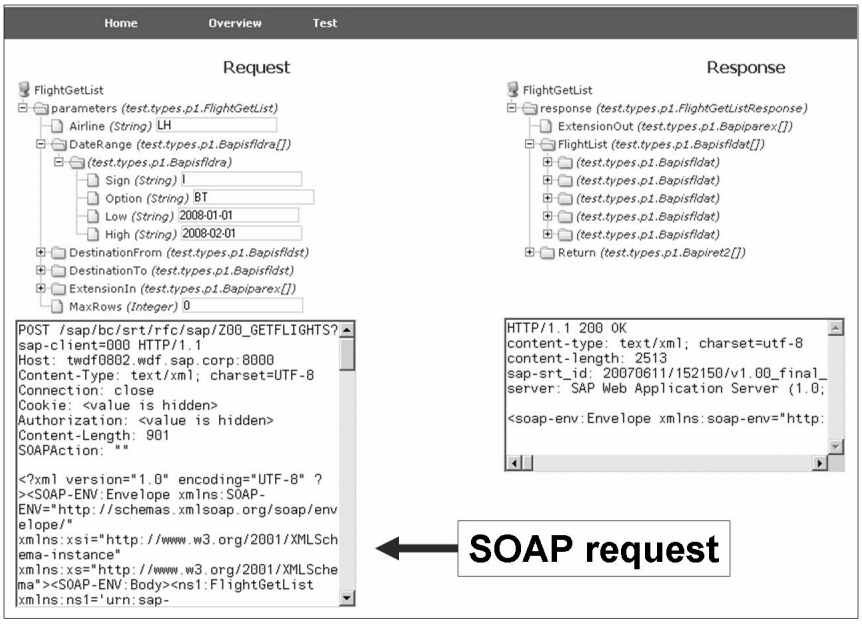


Figure 54: SOAP Request from *Web Service Homepage*

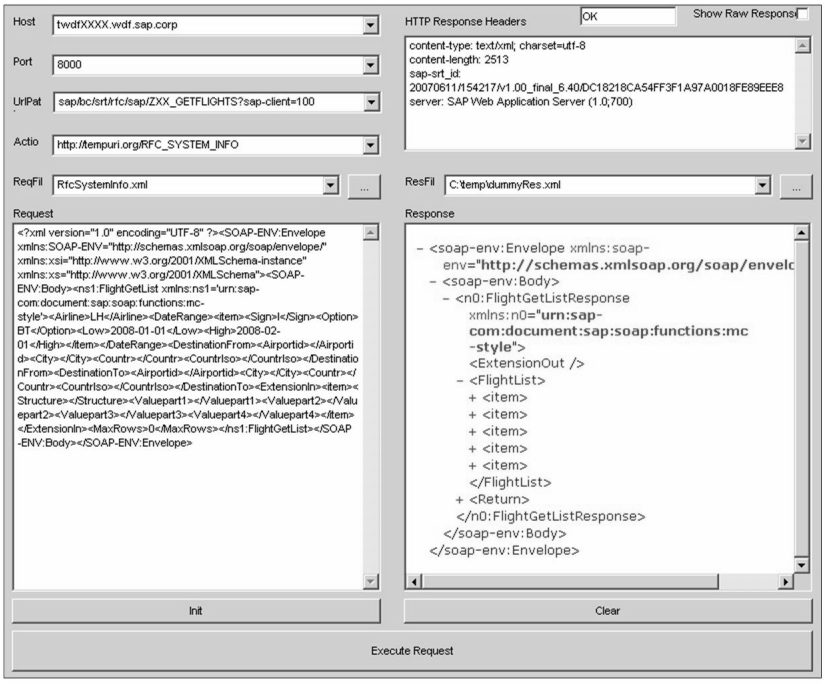


Figure 55: Local SOAP Client

Continued on next page

Task 4: SAP SOAP Client Tool (Optional)

If you have a user in the *SAP Developer Network (SDN)*, you can also use the *SAP SOAP Client Tool* to transmit an SOAP request.

1. Use the hyperlinkhttp://www.sdn.sap.com to access the *SAP Developer Network (SDN)*. Use the search function and enter *SAP SOAP Client Tool* as the search term. The result will be transmitted in the form of a hyperlink that you can use to start the *SAP SOAP Client Tool*.

   Note that this requires Java Virtual Machine (JVM) and Java Web Start technology to be installed on your machine.

   a) If you have questions, contact your instructor.

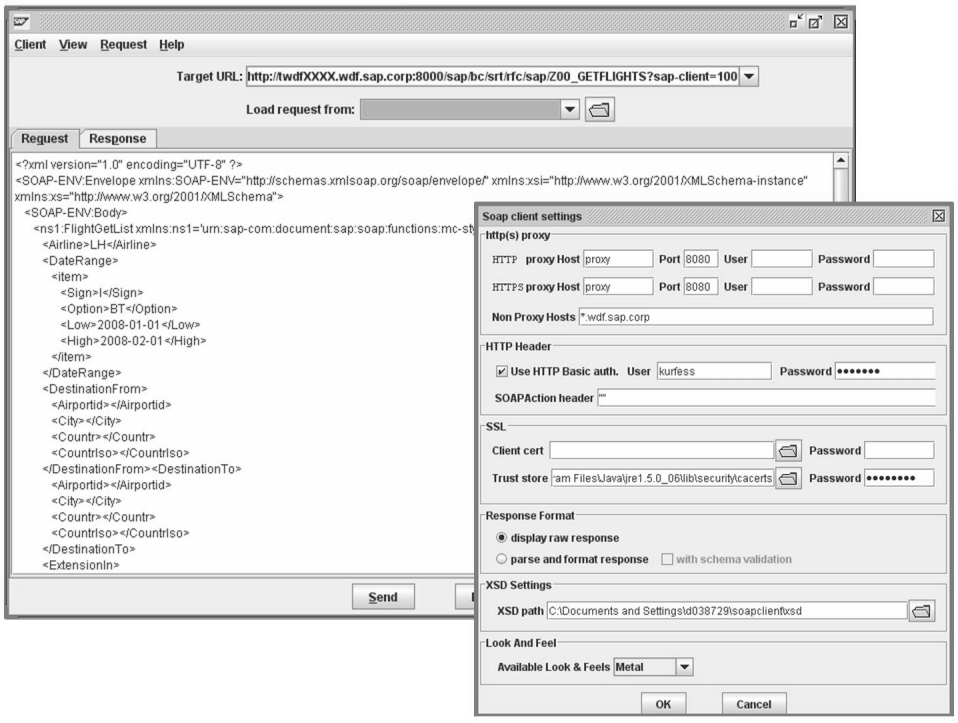


Figure 56: SAP SOAP Client Tool

Exercise 3: Process a Service Definition

Exercise Objectives

After completing this exercise, you will be able to:

- Create an ABAP Web service using the Service Definition Wizard
- Adapt an ABAP Web service interface
- Activate a Web service for the SOAP runtime.

Business Example

You need to activate and test a BAPI in an *SAP NetWeaver Application Server 7.0* as an ABAP Web service.

Task 1: Service Definition Wizard

From the *Flight* business object, the *GetDetail* BAPI should be activated as an ABAP Web service using the *Service Definition Wizard*.

1. Familiarize yourself with the BAPI interface and test the BAPI. Use the *BAPI Explorer* for this (transaction BAPI).

   What information does the BAPI provide you with?

   _____

   _____

   _____

   _____

2. Use the *Service Definition Wizard* to generate a Web service based on the *GetDetail* BAPI of the *Flight* business object. This Web service must be stateless, and the caller must be verified using a user name and password. The service should **not** be activated for the SOAP runtime.

   Use the following identifiers for the service definition to be generated:

   Service definition: ***z##_getDetails***

   You should replace ## with your two-digit group number. This group number is available from your instructor.

3. Check whether the service definition was generated. Verify that the ABAP Web service cannot be called yet.

Continued on next page

Internal Use SAP Partner Only

Task 2: Process a Service Definition

From the *Flight* business object, the *GetDetail* BAPI was activated as a Web service using the *Service Definition Wizard*. In the next step, you adapt the service definition interface:

1. The service definition interface should be adapted as follows:

   Hide the *ExtensionIn* and *ExtensionOut* parameters.

   The exporting parameter *AdditionalInfo* should be adapted as follows: The fields *Distance*, *Unit*, *UnitIso*, *Planetype* and *Flighttype* should be deleted.

   The exporting parameter *Availability* should be adapted as follows: The fields *Economax*, *Businmax* and *Firstmax* should be deleted.

   The exporting parameter *Return* should be adapted as follows: The exporting parameter should be renamed *Error*, since the *Return* indicator represents a reserved keyword in many programming languages. Delete the *Id*, *Number*, *LogNo*, *LogMsgNo*, *Parameter*, *Row*, *Field*, and *System* fields.

2. Now activate the service definition.

Task 3: Activation for SOAP Runtime

You have now created and processed a service definition. In the next step, you need to activate the service definition for the SOAP runtime.

1. Activate your service definition for the SOAP runtime.
2. Test your ABAP Web service via the *Web service homepage*.

Task 4: Modify the Logon Procedure (Optional)

You want to store a user name and password for the Web service.

1. Define your own user (BC416-##) for the logon procedure.
2. Now test the Web service on the *Web service homepage* or with the local SOAP client.

Solution 3: Process a Service Definition

Task 1: Service Definition Wizard

From the *Flight* business object, the *GetDetail* BAPI should be activated as an ABAP Web service using the *Service Definition Wizard*.

1. Familiarize yourself with the BAPI interface and test the BAPI. Use the *BAPI Explorer* for this (transaction BAPI).

   What information does the BAPI provide you with?

   Answer: With this BAPI, you can get all the detailed information that is available for a particular flight.

2. Use the *Service Definition Wizard* to generate a Web service based on the *GetDetail* BAPI of the *Flight* business object. This Web service must be stateless, and the caller must be verified using a user name and password. The service should **not** be activated for the SOAP runtime.

   Use the following identifiers for the service definition to be generated:

   Service definition: ***z##_getDetails***

   You should replace ## with your two-digit group number. This group number is available from your instructor.

   a) Since the service is not to be activated for the SOAP Runtime, the *Release Service for Runtime* checkbox must not be selected.

3. Check whether the service definition was generated. Verify that the ABAP Web service cannot be called yet.

   a) Look through the object list in the *Object Navigator* (SE80).

   b) To test the ABAP Web service, call the transaction WSADMIN (Web Service Administration). You will find no entry for the previously created service definition, since the service was not activated for the SOAP runtime.

Task 2: Process a Service Definition

From the *Flight* business object, the *GetDetail* BAPI was activated as a Web service using the *Service Definition Wizard*. In the next step, you adapt the service definition interface:

1. The service definition interface should be adapted as follows:

   Hide the *ExtensionIn* and *ExtensionOut* parameters.

Continued on next page

The exporting parameter *AdditionalInfo* should be adapted as follows: The fields *Distance*, *Unit*, *UnitIso*, *Planetype* and *Flighttype* should be deleted.

The exporting parameter *Availability* should be adapted as follows: The fields *Economax*, *Businmax* and *Firstmax* should be deleted.

The exporting parameter *Return* should be adapted as follows: The exporting parameter should be renamed *Error*, since the *Return* indicator represents a reserved keyword in many programming languages. Delete the *Id*, *Number*, *LogNo*, *LogMsgNo*, *Parameter*, *Row*, *Field*, and *System* fields.

a) To hide an interface parameter, disable the *Exposed* flag.

b) To delete fields from an interface parameter, you need to adjust the relevant data type. Use the *Types* table tab header to access the copied interface data types.

2. Now activate the service definition.

   a) Speak to your instructor if you have any questions.

Task 3: Activation for SOAP Runtime

You have now created and processed a service definition. In the next step, you need to activate the service definition for the SOAP runtime.

1. Activate your service definition for the SOAP runtime.

   a) To do this, call the transaction WSCONFIG and enter the name of your service definition in the *Definition* field. Enter the name of the service definition again in the *Variant* field. Choose *Create*, followed by Save, to activate the service definition.

2. Test your ABAP Web service via the *Web service homepage*.

   a) To do this, call the transaction WSADMIN.

Task 4: Modify the Logon Procedure (Optional)

You want to store a user name and password for the Web service.

1. Define your own user (BC416-##) for the logon procedure.

   a) To do this, call the transaction WSCONFIG. Double-click on your configuration. On the *Web Service Settings* tab page, choose the *ICF Details* button to access the SICF service node of your Web service. You can define a user here.

Continued on next page

2. Now test the Web service on the *Web service homepage* or with the local SOAP client.

   a) Speak to your instructor if you have any questions.

Lesson Summary

You should now be able to:

- Create an ABAP Web service based on an RFC-enabled function module.
- Describe the functions of the *service interface* and the *variant*.
- Describe the *Web service homepage*.

Lesson: Introduction: WSDL

Lesson Overview

In this lesson, we will explain the structure and use of Web Services Description Language (WSDL). You will gain an understanding of the contents of a WSDL document, and you will learn how to localize information about a Web service.

Lesson Objectives

After completing this lesson, you will be able to:

- Describe the structure of a WSDL document
- Discuss the importance of WSDL in the Web services environment

Business Example

WSDL plays an important part in the development and utilization of Web services. To be able to develop Web services for important business applications, you need to understand the semantics of a WSDL document.

Definition of WSDL

WSDL = Web Service Description Language

The WSDL service definition provides a description of distributed systems and information about the automation of data exchange between applications.

The Web Services Description Language describes Web services or electronic services in XML format. A service is defined as an accumulation of end points (ports) and the messages that these work with.

Using WSDL allows service providers to describe the requirements and functions of their Web services such that a prospective customer can understand these and integrate them correctly with the service.

WSDL was originally developed by Microsoft (SOAP Contract Language (SCL) and IBM (Network Accessible Service Specification Language (NASSL). In 2001, WSDL 1.1 was adopted by W3C with the support of Microsoft, IBM, and SAP, among others. Since 2002, WSDL has been developed further by W3C, and WSDL 2 has been released.

WSDL is not fixed on any special message or network protocol. WSDL 1.1 contains definitions for SOAP 1.1, HTTP GET/POST, and MIME.

Internal Use SAP Partner Only

Although WSDL does not display a definitive standard, it is nonetheless a language that is widely used on a variety of Web services platforms. For example, the Web Services Interoperability Organization (WS-I) was founded to facilitate the rapid implementation of Web services. Basic Profile 1.0 and 1.1 of the WS-I confirm WSDL 1.1 as one of the key specifications. The Java Developer Network supports WSDL, as do leading developers of Web services platforms, for example Microsoft .NET, IBM WebSphere, BEA Web Logic, and SAP NetWeaver.

WSDL is used to describe the operational interface, comparable to a Web Interface Definition Language (WIDL). The structure of WSDL provides for a dichotomy into "abstract" and "concrete" descriptions of the interface.

The abstract description of the interface defines language- and platform-independent messages for exchange. This allows the program structure to remain independent of real communication protocols and other technical factors, right down to the lowest possible level. Prior to the actual communication process, the abstract level must be left behind so that object serialization can be performed based on a concrete example.

Uses of WSDL

The same Web service can be called from different systems, independent of their technical features. The implementation of the Web service client is not dependent on the technical structure of the server. Consumers are generated from an implementation-independent service definition. The technical details are configured in the Web services consumer's runtime environment.

The developer, on the part of the consumer, obtains the description of the Web service in the form of a WSDL document. (1). The developer creates a Web service proxy using a programming language generation tool (2). This encapsulates Web service access, for example, as a class that can be used in the application, at a suitable point, and as often as is required (3). At runtime, the client proxy undertakes communication with the Web service, for example via the SOAP protocol. (4).

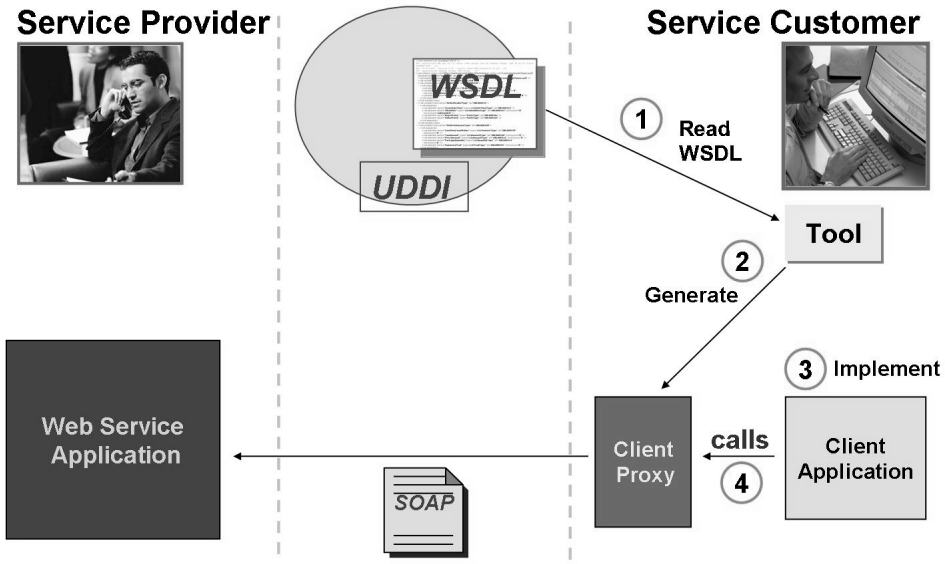


Figure 57: Function of the Web Service Definition

WSDL Elements

Binding links the abstract description of the interface with the concrete part containing the physical protocols and addresses for communication.

The concrete description of the interface takes into account the concrete network addresses and ports of the end points or Web services on the one hand and, on the other, the communication protocols (SOAP, HTTP, MIME etc) for the actual data exchange.

A WSDL document defines services as a grouping of network ports. WSDL separates the abstract definition of the ports and messages from the concrete network and data format link. This allows definitions to be re-used: Messages, the abstract definition of exchange data, and Port Types, the abstract depiction of operations. The actual protocol and data format specification requires a shared connection (binding). A port, in turn, links a network address with a binding, and a group of ports represents a service.

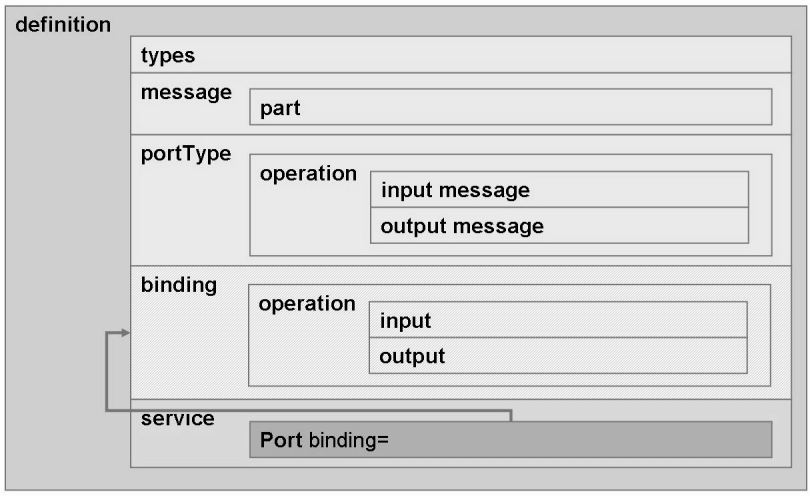


Figure 58: Structure of a WSDL Document

The following elements are outlined in a WSDL:

- **Definition** : The logical root element in which the different namespaces are defined as attributes
- **Types**: A container for data type definitions when using a typing system, for example XSD
- **Messages**: An abstract name for communicated data. A message element consists of a number of part elements. Every part element refers to a type defined in the types element. A part can represent a message parameter or a function call parameter.
- **Operation**: Operation is an abstract name for the possible actions of a service. Every operation consists of its name and one to three messages: an input, output, and possibly a fault message. Depending on whether an operation involves an input and output message, or only one of these two messages, there are different types of operations in a port type (one-way, notification, etc.):- One-way: The operation can contain a message, but it does not return a message.- Request/Response: The operation identifies an input message and returns a message.- Expected response: The operation can send a message, and waits for a response.- Message: A message can be sent, and no response is expected.
- **Port Type**: An abstract grouping of operations from one or more ports
- **Binding**: A concrete protocol and data format definition (SOAP, HTTP-GET, etc.) for a specific port type
- **Port**: A single port as a combination of a binding and a network address
- **Service**: A grouping of used ports or a grouping of related interfaces, consisting of ports for a service
- **Part**: Abstract definition of a parameter

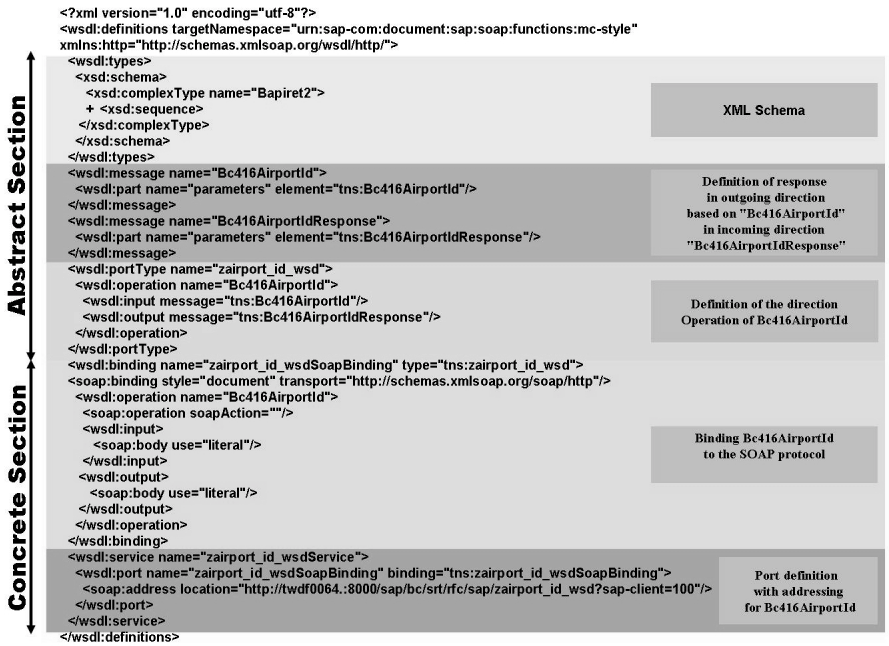


Figure 59: Sample Structure of a WSDL Document

WSDL of SAP NetWeaver AS ABAP

The WSDL of a Web service generated under ABAP can be viewed with the transaction WSADMIN. Use *Web Service → WSDL* or the framed icon (below) to print the WSDL in a browser window.

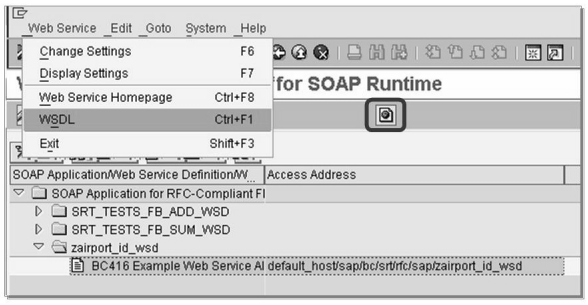


Figure 60: Calling the WSDL of an ABAP Web Service

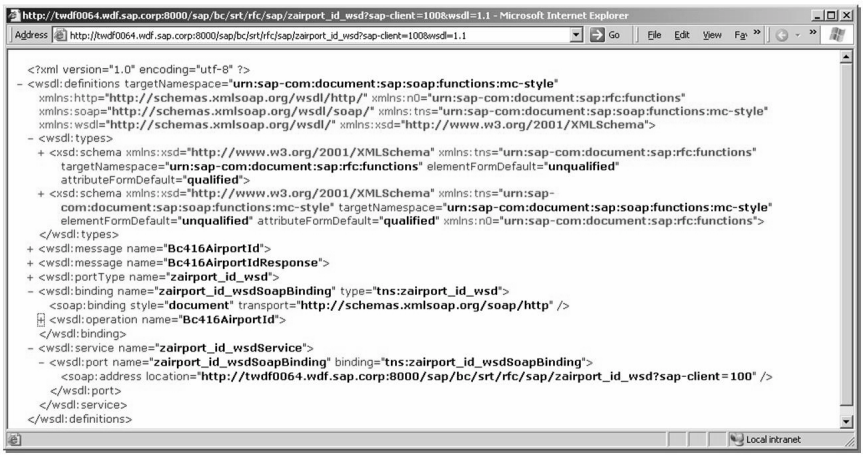


Figure 61: Output of the ABAP Web Service Definition

Hint: Viewing WSDL documents We recommend that you use an XML editor (such as *Microsoft XML Notepad*) to view or edit a WSDL document. If you only want to view the document, you can use a standard Web browser (Microsoft Internet Explorer or Firefox). You should save a WSDL document with the .wsd file extension.

For more information about WSDL, see:

- help.sap.com (keyword "WSDL")
- www.w3schools.org

Internal Use SAP Partner Only

Exercise 4: WSDL

Exercise Objectives

After completing this exercise, you will be able to:

- Analyze a WSDL document

Business Example

You want to integrate a Web service and, therefore, verify the interface parameters using the WSDL document.

Task 1:

Load the WSDL description under the following URL *http://<host>.wdf.sap.corp:<port>/sap/bc/srt/rfc/sap/BC416_VACATION_CHECKER?sap-client=100&wsdl=1.1*and answer the following:

1. Which port can be used to call the service?

2. What is the service called?

3. Which parameters does the service recognize?

Continued on next page

4. What is the incoming message called?

Task 2:

Answer the following:

1. Are the function modules of a function group a Web service, or is each a separate service of its own?

2. How can you forward the WSDL of an ABAP Web service?

Internal Use SAP Partner Only

Solution 4: WSDL

Task 1:

Load the WSDL description under the following URL *http://<host>.wdf.sap.corp:<port>/sap/bc/srt/rfc/sap/BC416_VACATION_CHECKER?sap-client=100&wsdl=1.1*and answer the following:

1. Which port can be used to call the service?

   Answer: BC416_VACATION_CHECKERSoapBinding

2. What is the service called?

   Answer: BC416_VACATION_CHECKERService

3. Which parameters does the service recognize?

   Answer: The following parameters are used for input or output:

   - EMPLOYED_YEARS
   - VACATION_DAYS

4. What is the incoming message called?

   Answer: BC416_VACATION_CHECKER

Task 2:

Answer the following:

1. Are the function modules of a function group a Web service, or is each a separate service of its own?

   Answer: Function modules of a function group can be grouped together as one Web service. Every function module then represents an operation. The developer is free to activate each function module as an individual Web service.

2. How can you forward the WSDL of an ABAP Web service?

   Answer: As a file or URL, or via the UDDI directory.

Lesson Summary

You should now be able to:

- Describe the structure of a WSDL document
- Discuss the importance of WSDL in the Web services environment

Internal Use SAP Partner Only

Lesson: SAP NetWeaver AS as Web Service Client

Lesson Overview

This lesson provides an introduction to the topic of Web services, focusing on the SAP NetWeaver Application Server as the Web service client. The aim of this lesson is to integrate an ABAP Web service into an ABAP application using the ABAP Workbench.

Lesson Objectives

After completing this lesson, you will be able to:

- Generate a proxy using a WSDL document
- Define a logical port
- Call a Web service from an ABAP program

Business Example

You want to integrate a Web service into an ABAP application using the ABAP Workbench.

Web Services: The Client Side

The SAP NetWeaver Application Server doesn't just support the creation of Web services on the server side. It is also capable of calling Web services as a Web service client. As on the server side, the ABAP Workbench also offers support for the developer on the client side.

Internal Use SAP Partner Only

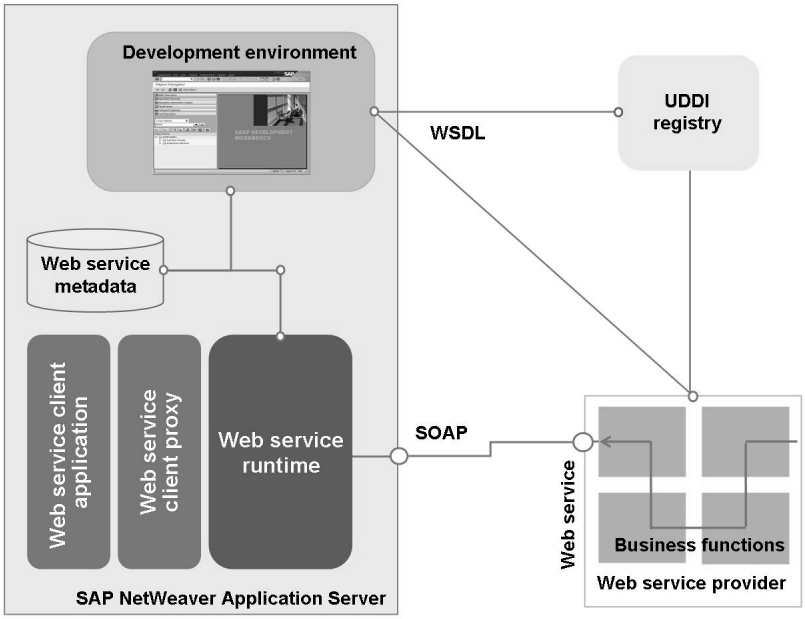


Figure 62: SAP NetWeaver Web Application Server as Web Service Client

Web Service Client Proxies

A **Web service client proxy** must be generated before a Web service can be implemented in an application. Client Proxies are generated in the *Object Navigator* (SE80) using a Web service description (WSDL document). Before the client proxies are generated, you can specify whether you want this description to be loaded from a URL, from a UDDI server, or from the SAP Exchange Infrastructure (SAP XI) Integration Repository. In the last case, you can then use the generated proxy to exchange messages with the SAP XI Integration Server.

A Web service client proxy acts as a transfer program. A Web service client proxy is used to connect to the server of the required Web service. The developer can concentrate on the business application while the technical part, for example the creation of a valid SOAP message or the evaluation of received responses, is carried out automatically using the proxy.

Logical Port

When the proxy is generated, all of the objects required to call a Web service are created. This includes the **logical port**. This is an SAP-specific concept for configuring the runtime features for Web service client proxies.

Runtime features, in contrast to design-time features, are properties that can be configured at the time of activation of the Web service client. An example of such a feature is the URL for calling the Web service, which may need to be customized by a user.

Design-time features are properties that are defined by the application user during development. These features are supplied together with the Web service client proxy and cannot be modified by the user at a later stage. In this way an application user can determine, for example, whether communication between the client and server should be session-oriented. The Web service is thereby assigned specific behaviour.

An example: Problems always occur when the proxy in a system landscape is moved from a test system to the live system, for example. If no customizations are made, the proxy will keep trying to call the Web service on the test system, although it should now be calling the live system. To prevent this from happening, either the proxy must be generated again using the WSDL of the live system, or the source code must be customized manually. As these individual adjustments are extremely error-prone, the configuration data (runtime features) is separated from the implementation (design-time features) in the Web service framework. After the proxy has been transported, you can customize the URL and other important parameters in the logical port so that the proxy can then call the Web service on the live system.

The following steps must be carried out to implement a Web service application on the client side.

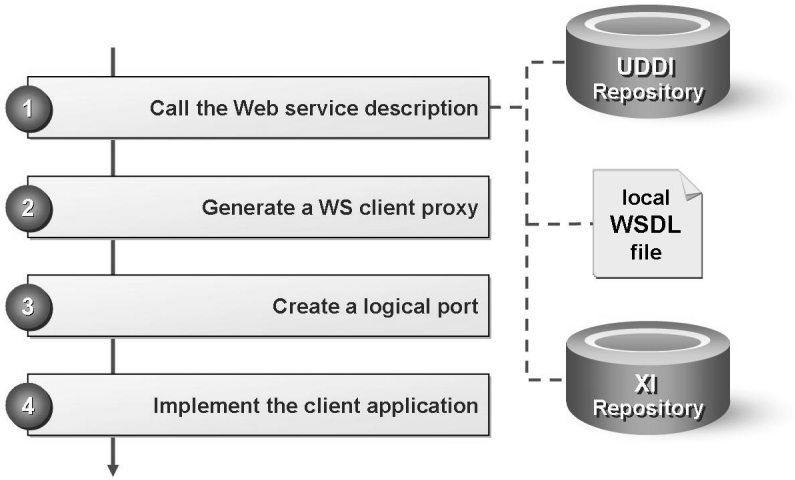


Figure 63: Overview: Implementing the Web Service Client

Step 1: Calling the Web Service Description

The WSDL document is the starting point for implementing a client application. You can find a suitable Web service by going through a UDDI directory, for example.

Step 2: Generating a Web Service Client Proxy

To generate a Web service client proxy on the basis of a WSDL document, call the context menu for your package and choose *Create → Enterprise Service / Web Service → Proxy Object*. In the dialog box, choose the WSDL document source. The WSDL document can be determined using a URL/HTTP destination, from a local file, from the UDDI, or via the SAP XI repository.

Hint: At present, only WSDL documents in *Document style* are supported during generation of the Web service client proxy. A proxy cannot be generated in the Object Navigator for a WSDL document in *RPC style*.

The ABAP server as Web service server, on the other hand, supports both forms, so if the Web service client is suitable, you can generate a Web service client proxy using a WSDL in *Document style* or *RPC style*.

In the dialog box that opens up, enter the name of a package in which the proxy objects are to be created. In addition, you can specify a prefix for the names of all the objects to be created to prevent naming conflicts with objects that already exist in the system.

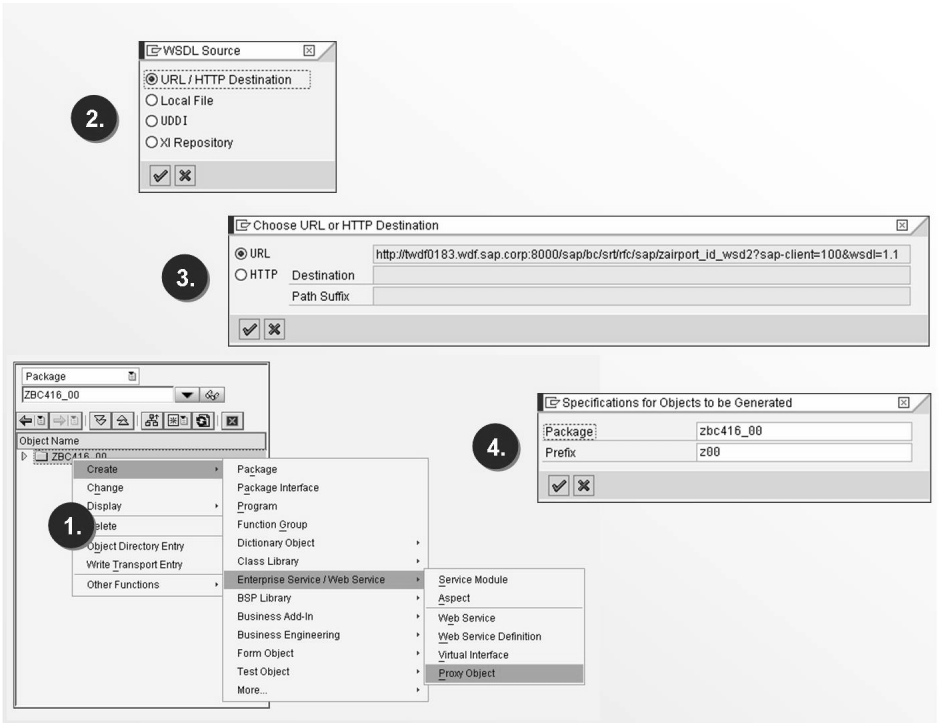


Figure 64: Proxy Generation

Don't miss **Tips on Generating Proxies** at the end of this lesson. Check whether you need to make adjustments following the automatic name assignment for proxy objects. If any subsequent changes are made to the WSDL document, the proxy objects must be regenerated.

Display and Modify Proxy Objects

The work area of the *Object Navigator* displays the following tab pages for the Web service client proxy that was created:

- **Properties** Attributes for the generation, such as name of the proxy class, package, or last user to make a change.
- **Name Conflicts** This tab is only available immediately after the proxy is generated. It allows you to correct names that are truncated during generation, or that need to be changed because a conflict arose.
- **Generation** A list of all the proxy objects that were generated.
- **Structure** Similar to the Generation tab, only that here the objects are arranged in a tree structure in accordance with their usage.
- **Type Mappings** Even if the proxy was generated successfully, there were situations where generation was only possible on the basis of certain implicit assumptions. Example: Restrictions to the value range are checked by the developer. If such situations arise during generation, they will be listed in an application log.
- **Preconfiguration** Preconfiguration involves specifying the properties for the proxy runtime environment that are determined at the time of development (designtime features). Preconfiguration forms the basis for the logical port settings. The proposed values come from the WSDL document used to generate the proxy. This tab is displayed when synchronous client proxies are generated.

The Object Navigator shows the proxy objects created in the system in the navigation tree under *Enterprise Services → Web Service Library* (client proxies, service proxies, and proxy dictionary types). The objects will only be created in the system if you select *Activate*. Up to the time of activation, you can save the metadata managed by the transaction, which contains all the information required for the generation, and continue with the generation at a later stage.

Step 3: Creating a Logical Port

The logical port is used to define the runtime features for the Web service client proxy. Use transaction LPCONFIG in order to create a logical port. Enter the name of the proxy class and a name for the logical port. You can use the value help (**F4 key**) for the *Logical Port* field to display existing ports for a proxy class. You may need to select the *Default Port* checkbox. If another logical port is already set as the default for this proxy class, this setting is overwritten, and the new port becomes the default.

Select *Create* to create the logical port.

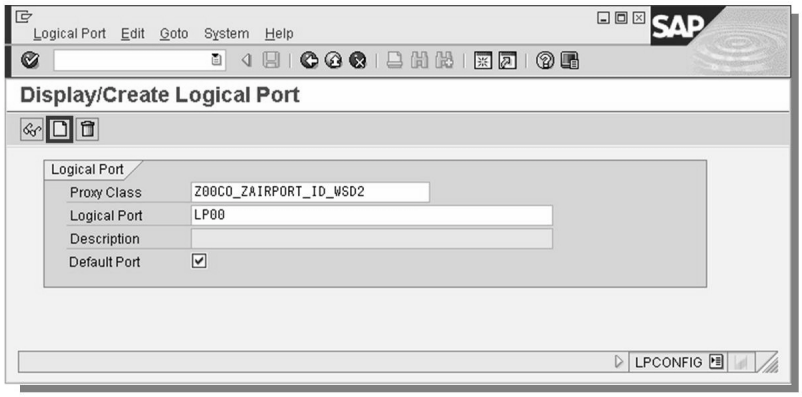


Figure 65: Creating a Logical Port

On the following screen, the configurable runtime features are assigned default values that can be changed if required.

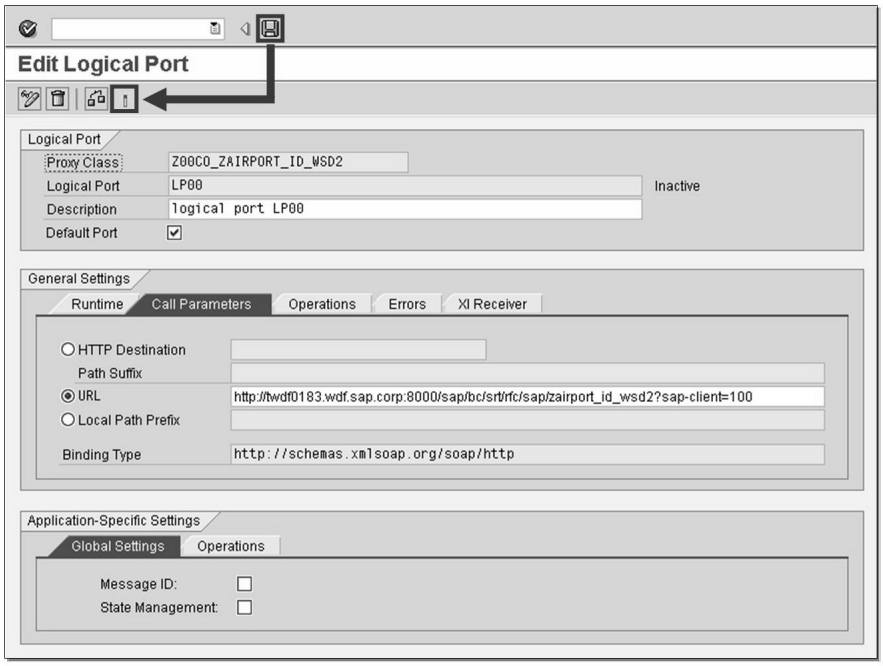


Figure 66: Activating a Logical Port

The following tab pages are available in the *Global Settings* screen area:

- **Runtime** You can use the radio buttons to choose whether the Web service runtime or the SAP NetWeaver XI runtime should be active through the logical port for the exchange of messages.

  If you choose the XI runtime environment, the *Call Parameters* and *Errors* tab pages are hidden in the *General Settings* screen area, because the XI runtime environment does not support these features. The *Application-Specific Settings* screen area is also hidden, because the XI runtime environment does not have any specific features.

- **Call Parameters** There are three ways of configuring the call address of the Web service:

  As an HTTP destination: Select a type G RFC destination (HTTP connection to external server) or a type H RFC destination (HTTP connection to R/3 system) from the transaction SM59. The HTTP destination approves the configuration of the logon procedure, encryption, and status management. This is the preferred access method.

  As a URL: Enter the URL directly. The disadvantage of this method is that with the exception of the URL, no other parameters for logging on, encryption, or status management can be configured. This method is only possible for Web services that do not require a logon procedure, encryption, or status management.

  As a local path prefix: This access method is only intended for accessing your own system. The default RFC destination, *NONE*, is called to address your own server. The specified local path prefix is used to identify the Web service that was called.

  The URI of the transport mechanism used is displayed in the *Binding Type* field. At present, only SOAP through HTTP are used, so no entry is required here.

- **Operations** A value can be specified in the SOAP Action field for the *SOAP Action* of the HTTP header. Servers and firewalls can use this value to filter SOAP messages.
- **Errors** The settings for logging and tracing can be made here. Messages from tracing are stored in the RFC developer trace of the SAP system. Logging writes the messages to the SAP system log.
- **XI Receiver** This tab is intended for applications that want to assign a logical port to an SAP NetWeaver XI receiver.

The following tab pages are available in the *Global Settings* screen area:

- **Global Settings** You can activate the message ID protocol by selecting the *Message ID* checkbox. The runtime environment then sends a unique message ID every time a message is sent.

  Selecting the *State Management* checkbox activates state management by means of HTTP cookies.

- **Operations** The operations of the Web service are displayed. Select an operation and assign security profiles. Using the WSS profiles, you can guarantee security at message level.

Finally, you need to save the logical port. This can then be activated in the next step.

Step 4: Implement the client application

The Web service client proxy that was generated can now be used in an ABAP application to call the Web service. Drag and drop the proxy class to your ABAP program and adapt the source code:

```
REPORT  zbc416_ws_client.

DATA: proxy TYPE REF TO z00co_zairport_id_wsd2,
      in    TYPE z00my_operation_name,
      out   TYPE z00my_operation_name_response.

in-depa-city    = 'frankfurt'.
in-depa-country = 'de'.

TRY.
    CREATE OBJECT proxy
       EXPORTING
         logical_port_name = 'LP00'.
  CATCH cx_ai_system_fault.
ENDTRY.

TRY.
    CALL METHOD proxy->my_operation_name
      EXPORTING
        input  = in
      IMPORTING
        output = out.
  CATCH cx_ai_system_fault.
  CATCH cx_ai_application_fault.
```


Internal Use SAP Partner Only


```
ENDTRY.

WRITE: / out-departure-airportid,
       / out-departure-city,
       / out-departure-country.
```

You can specify a logical port at runtime using the proxy class constructor. This allows you to control whether the Web service runtime or the XI runtime should be processed for sending messages.

- If no logical port was specified via the constructor, and if no port was set as the default, the system assumes that the message is to be sent via the XI runtime.
- If a logical port was specified but it is not available, the system raises an exception. In all other cases, the port setting described above determines the runtime that the system will use to send the message.

There can only be one default port for a Web service client proxy. To ensure that calls can be made without specifying a logical port, a default port should be set for every Web service client proxy. There can be several logical ports for a Web service client proxy.

Tips on Generating Proxies

Name conflicts

Generating proxies gives rise to a number of ABAP Dictionary objects. To avoid name conflicts with existing objects, you can assign a prefix for the objects to be generated. This prefix cannot start with “X”, and an underscore (“_”) cannot be inserted until after the third character at the earliest (example: “ABC_”). You can also assign an SAP R/3 namespace as a prefix (for example, /CRM/). Make sure in these instances that the assigned development class is compatible with the namespace.

Translation and package assignment

When proxy objects are generated, the number of ABAP Dictionary objects, classes, and interfaces created can lead to a considerable volume of translation. This translation is pointless, however, since these proxy objects do not appear in user interfaces. You should therefore ensure that proxy objects are separated at package level. Create a separate package for the proxy objects.

User access

To enable access to proxies and the proxy framework from the application code, the SAI_PROXY_PUBLIC_PIF package interface must have use access. Please note that for a potentially enclosing package, use access must also be created on the SAI_TOOLS_PIF enclosing package interface. For more information, see the online documentation.

Internal Use SAP Partner Only

Exercise 5: Consume a Web Service

Exercise Objectives

After completing this exercise, you will be able to:

- Create an HTTP destination
- Generate a Web service client proxy
- Create a logical port
- Implement a Web service client application

Business Example

The ABAP Web service (*z##_getFlights*) created in exercise 2 should now be integrated into an ABAP application. You need to generate a Web service client proxy for this. You must also define another logical port and HTTP destination.

Task 1: HTTP Destination

You need to call your *z##_getFlights* ABAP Web service on the "remote" SAP system using an HTTP destination.

1. Create the HTTP destination that you will use to call the Web service. Suggested name: ***WS##_DES***

   Replace ## with your two-digit group number.

2. In the technical settings, specify the target host, port number, and path prefix.

3. Save the client and your user in the HTTP destination.

4. Run a connection test. What HTTP status code is returned, and why?

Task 2: Web Service Client Proxy

You need to generate a Web service client proxy for the *z##_getFlights* Web service.

1. Generate a Web service client proxy for the Web service *z##_getFlights*. Use the ***Z##*** prefix for the objects to be generated.

Continued on next page

You should replace ## with your two-digit group number.

Task 3: Logical Port

You have to create a logical port for the generated Web service client proxy.

1. Create a logical port for your Web service client proxy and classify it as the default port. Suggested name: ***LP##_FLIGHTS***.

   Replace ## with your two-digit group number.

2. You still have to specify the Call URL for the Web service via the logical port. Use your HTTP destination for this.

3. Save and then activate your logical port.

Task 4: Implement a Client Application

You need to be able to call your *z##_getFlights* Web service from an ABAP program and display the data on the basic list. The import parameters of the Web service are to be entered via a selection screen.

1. Create an executable program. Suggested name: ***Z##_WS_CLIENT***.

2. Define a selection screen with input parameters for describing the airline, departure location, arrival location, and date in brief.

3. Call your *z##_getFlights* Web service using the Web service client proxy that you generated. Use the constructor to specify your *LP##_FLIGHTS* logical port.

   When calling the Web service operation, complete the relevant interface. Use the data from the selection screen for this.

4. Now print the flight list with the most important flight properties on the basic list.

Solution 5: Consume a Web Service

Task 1: HTTP Destination

You need to call your *z##_getFlights* ABAP Web service on the "remote" SAP system using an HTTP destination.

1. Create the HTTP destination that you will use to call the Web service. Suggested name: ***WS##_DES***

   Replace ## with your two-digit group number.

   a) Go to the transaction SM59 and create a destination of type H (HTTP connection to R/3 system).

2. In the technical settings, specify the target host, port number, and path prefix.

   a) As we will remain on the current system, the current server will be used as the "target host".

   The port number corresponds to the HTTP port of the ICM. You can determine this using, for example, the *Goto → Services* menu path in the transaction SMICM.

   Use */sap/bc/srt/rfc/sap/* as the path prefix.

3. Save the client and your user in the HTTP destination.

   a) To do this, use the *Logon & Security* tab page.

4. Run a connection test. What HTTP status code is returned, and why?

   Answer: The body of the HTTP request does not contain a SOAP document. The SOAP runtime was therefore unable to process the request. An SOAP fault and HTTP status code 500 (Internal Server Error) are returned.

Task 2: Web Service Client Proxy

You need to generate a Web service client proxy for the *z##_getFlights* Web service.

1. Generate a Web service client proxy for the Web service *z##_getFlights*. Use the ***Z##*** prefix for the objects to be generated.

Continued on next page

You should replace ## with your two-digit group number.

a) To do this, go to the *Object Navigator* (SE80). From the context menu for your package *zbc416_##* follow the menu path *Create → Enterprise Service / Web Service → Proxy Object* to create a proxy.

b) The WSDL document is used to generate the proxy. Enter the URL of the WSDL document on the corresponding screen. Use the HTTP destination you created earlier, which points to the following destination: *http://<server>:<port>/sap/bc/srt/rfc/sap/*.

As the path suffix, enter your *z##_getFlights* Web service followed by the *?wsdl=1.1* query string. You do not need to specify the client in the query string, since this was already defined in the HTTP destination. Replace ## with your group number.

c) On the next screen, enter your package (*zbc416_##*). Use *Z##* as the prefix. Replace ## with your group number.

d) Then save and activate your Web service client proxy.

Task 3: Logical Port

You have to create a logical port for the generated Web service client proxy.

1. Create a logical port for your Web service client proxy and classify it as the default port. Suggested name: ***LP##_FLIGHTS***.

   Replace ## with your two-digit group number.

   a) Use the transaction LPCONFIG for this.

   b) On the initial screen, use the input help to select your proxy class and assign a name to your logical port: Suggested name: *LP##_FLIGHTS*.

   c) Classify the logical port as the default port using the corresponding checkbox.

   d) Now choose *Create* to create the logical port.

2. You still have to specify the Call URL for the Web service via the logical port. Use your HTTP destination for this.

   a) You can enter the Web service URL on the *Call Parameters* tab page. Use your *WS##_DES* HTTP destination for this. Specify your Web service *z##_getFlights* as the path suffix. You do not need to specify the client, since this was defined in the HTTP destination. Replace ## with your group number.

Continued on next page

Internal Use SAP Partner Only

3. Save and then activate your logical port.

   a) If you have questions, contact your instructor.

Task 4: Implement a Client Application

You need to be able to call your *z##_getFlights* Web service from an ABAP program and display the data on the basic list. The import parameters of the Web service are to be entered via a selection screen.

1. Create an executable program. Suggested name: ***Z##_WS_CLIENT***.

   a) If you have questions, contact your instructor.

2. Define a selection screen with input parameters for describing the airline, departure location, arrival location, and date in brief.

   a) See the sample solution. Report *BC416S_WS_CLIENT_S1*.

3. Call your *z##_getFlights* Web service using the Web service client proxy that you generated. Use the constructor to specify your *LP##_FLIGHTS* logical port.

   When calling the Web service operation, complete the relevant interface. Use the data from the selection screen for this.

   a) See the sample solution. Report *BC416S_WS_CLIENT_S1*.

4. Now print the flight list with the most important flight properties on the basic list.

   a) See the sample solution. Report *BC416S_WS_CLIENT_S1*.

Result

```
*&---------------------------------------------------------------------*
*& Report  BC416_WS_CLIENT_S1
*&---------------------------------------------------------------------*
*& Musterlösung zur Übung 5: Web Service konsumieren
*&---------------------------------------------------------------------*
REPORT bc416_ws_client_s1.

DATA date TYPE s_date.
*****************************************
* Selektionsbild
*
PARAMETERS: pa_airl TYPE bc00char3  DEFAULT 'LH',
            pa_depa TYPE bc00char20 DEFAULT 'Frankfurt',
            pa_dest TYPE bc00char20 DEFAULT 'New York'.
```

Continued on next page


```
SELECT-OPTIONS pa_date FOR date DEFAULT '20080101' TO '20090101'
                               OPTION BT SIGN I.


*************************************************************
*  START-OF-SELECTION                                       *
*************************************************************
START-OF-SELECTION.

*********************************************************
* Proxy instanziieren (Flugliste)
*
  DATA proxy_flights TYPE REF TO co_bc00bc416_get_flights.

  TRY.
      CREATE OBJECT proxy_flights
        EXPORTING
          logical_port_name = 'LP_FLIGHTS'.

    CATCH cx_ai_system_fault .
  ENDTRY.

*********************************************************
* Schnittstelle versorgen
*
  DATA: in_flights  TYPE bc00flight_get_list,
        out_flights TYPE bc00flight_get_list_response.

  MOVE: pa_airl TO in_flights-airline,
        pa_depa TO in_flights-destination_from-city,
        pa_dest TO in_flights-destination_to-city.

  DATA: wa_date LIKE LINE OF pa_date,
        wa_date2 TYPE bc00bapisfldra.

  LOOP AT pa_date INTO wa_date.
    MOVE-CORRESPONDING wa_date TO wa_date2.
    INSERT wa_date2 INTO TABLE in_flights-date_range-item.
  ENDLOOP.

*********************************************************
```

Continued on next page

```
* Web Service Operation über Proxy rufen (Flugliste)
*
  TRY.
      proxy_flights->flight_get_list( EXPORTING input  = in_flights
                                      IMPORTING output = out_flights ).

    CATCH cx_ai_system_fault .
    CATCH cx_ai_application_fault .
  ENDTRY.

*********************************************************
* Flugliste auf Grundliste ausgeben
*
  DATA wa_flights TYPE bc00bapisfldat.
  LOOP AT out_flights-flight_list-item INTO wa_flights.

    WRITE: /   wa_flights-airlineid,
            5  wa_flights-connectid,
            11 wa_flights-airline,
            25 wa_flights-cityfrom,
            38 wa_flights-cityto,
            50 wa_flights-flightdate,
            63 wa_flights-deptime,
            75 wa_flights-arrdate,
            88 wa_flights-arrtime.
    ENDLOOP.
```


Internal Use SAP Partner Only

Internal Use SAP Partner Only

Exercise 6: Consume a Web Service (Optional)

Exercise Objectives

After completing this exercise, you will be able to:

- Generate a Web service client proxy
- Create a logical port
- Implement a Web service client application

Business Example

The ABAP application created in the preceding exercises needs to be enhanced: You must now integrate the *z##_getDetails* ABAP Web service and provide detailed data for a chosen flight. You need to generate a new proxy and define a logical port.

Task 1: Web Service Client Proxy

You need to generate a Web service client proxy for the *z##_getDetails* Web service.

1. Generate a proxy for the *z##_getDetails* Web service. Use the ***Z##*** prefix for the objects to be generated.

   You should replace ## with your two-digit group number.

Task 2: Logical Port

You have to create a logical port for the generated Web service client proxy.

1. Create a logical port for your Web service client proxy and classify it as the default port. Suggested name: ***LP##_DETAILS***.

   You should replace ## with your two-digit group number.

2. You still have to specify the Call URL for the Web service via the logical port. Use your HTTP destination for this.

3. Save and then activate your logical port.

Continued on next page

Task 3: Implement a Client Application

The ABAP application created in the preceding exercises needs to be enhanced: You must now integrate the *z##_getDetails* ABAP Web service into the application and display detailed data for a chosen flight on the details list.

Enhance your *Z##_WS_CLIENT* program for this.

1. Enhance your *Z##_WS_CLIENT* program with the *AT LINE-SELECTION* event, and make sure that the flight data chosen on the basic list is available again in the *AT LINE-SELECTION* event.
2. In the *AT LINE-SELECTION* ABAP processing block, call your *z##_getDetails* Web service using the Web service client proxy that you generated. Use the constructor to specify the *LP##_DETAILS* logical port.

   When calling the Web service operation, complete the relevant interface. Use the data from the HIDE area for this.
3. Now print the detailed flight data on the details list.

Solution 6: Consume a Web Service (Optional)

Task 1: Web Service Client Proxy

You need to generate a Web service client proxy for the *z##_getDetails* Web service.

1. Generate a proxy for the *z##_getDetails* Web service. Use the ***Z##*** prefix for the objects to be generated.

   You should replace ## with your two-digit group number.

   a) To do this, go to the *Object Navigator* (SE80). From the context menu for your package *zbc416_##* follow the menu path *Create → Enterprise Service / Web Service → Proxy Object* to create a proxy.

   b) The WSDL document is used to generate the proxy. Enter the URL of the WSDL document on the corresponding screen. Use the HTTP destination you created earlier, which points to the following destination: *http://<server>:<port>/sap/bc/srt/rfc/sap/*.

   As the path suffix, enter your *z##_getDetails* Web service followed by the *?wsdl=1.1* query string. You do not need to specify the client in the query string, since this was already defined in the HTTP destination. Replace ## with your group number.

   c) On the next screen, enter your package (*zbc416_##*). Use *Z##* as the prefix. Replace ## with your group number.

   d) Then save and activate your Web service client proxy.

Task 2: Logical Port

You have to create a logical port for the generated Web service client proxy.

1. Create a logical port for your Web service client proxy and classify it as the default port. Suggested name: ***LP##_DETAILS***.

Continued on next page

You should replace ## with your two-digit group number.

a) Use the transaction LPCONFIG for this.

b) On the initial screen, use the input help to select your proxy class and assign a name to your logical port: Suggested name: *LP##_DETAILS*.

c) Classify the logical port as the default port using the corresponding checkbox.

d) Now choose *Create* to create the logical port.

2. You still have to specify the Call URL for the Web service via the logical port. Use your HTTP destination for this.

   a) You can enter the Web service URL on the *Call Parameters* tab page. Use your *WS##_DES* HTTP destination for this. Specify your Web service *z##_getDetails* as the path suffix. You do not need to specify the client, since this was defined in the HTTP destination. Replace ## with your group number.

3. Save and then activate your logical port.

   a) If you have questions, contact your instructor.

Task 3: Implement a Client Application

The ABAP application created in the preceding exercises needs to be enhanced: You must now integrate the *z##_getDetails* ABAP Web service into the application and display detailed data for a chosen flight on the details list.

Enhance your *Z##_WS_CLIENT* program for this.

1. Enhance your *Z##_WS_CLIENT* program with the *AT LINE-SELECTION* event, and make sure that the flight data chosen on the basic list is available again in the *AT LINE-SELECTION* event.

   a) Keyword: HIDE area!

2. In the *AT LINE-SELECTION* ABAP processing block, call your *z##_getDetails* Web service using the Web service client proxy that you generated. Use the constructor to specify the *LP##_DETAILS* logical port.

   When calling the Web service operation, complete the relevant interface. Use the data from the HIDE area for this.

   a) See the sample solution.

3. Now print the detailed flight data on the details list.

Continued on next page

Internal Use SAP Partner Only

a) Refer to the sample solution for details.

Result

```
*&---------------------------------------------------------------------*
*& Report  BC416_WS_CLIENT_S2
*&---------------------------------------------------------------------*
*& Musterlösung zur Übung 6: Web Service konsumieren (optional)
*&---------------------------------------------------------------------*
REPORT bc416_ws_client_s2.

DATA date TYPE s_date.
*****************************************
* Selektionsbild
*
PARAMETERS: pa_airl TYPE bc00char3  DEFAULT 'LH',
            pa_depa TYPE bc00char20 DEFAULT 'Frankfurt',
            pa_dest TYPE bc00char20 DEFAULT 'New York'.

SELECT-OPTIONS pa_date FOR date DEFAULT '20080101' TO '20090101'
                               OPTION BT SIGN I.


************************************************************
*  START-OF-SELECTION                                      *
************************************************************
START-OF-SELECTION.

*******************************************************
* Proxy instanziieren (Flugliste)
*
  DATA proxy_flights TYPE REF TO co_bc00bc416_get_flights.

  TRY.
      CREATE OBJECT proxy_flights
        EXPORTING
          logical_port_name = 'LP_FLIGHTS'.

    CATCH cx_ai_system_fault .
  ENDTRY.
```

Continued on next page


```
*********************************************************
* Schnittstelle versorgen
*
  DATA: in_flights  TYPE bc00flight_get_list,
        out_flights TYPE bc00flight_get_list_response.

  MOVE: pa_airl TO in_flights-airline,
        pa_depa TO in_flights-destination_from-city,
        pa_dest TO in_flights-destination_to-city.

  DATA: wa_date LIKE LINE OF pa_date,
        wa_date2 TYPE bc00bapisfldra.

  LOOP AT pa_date INTO wa_date.
    MOVE-CORRESPONDING wa_date TO wa_date2.
    INSERT wa_date2 INTO TABLE in_flights-date_range-item.
  ENDLOOP.

*********************************************************
* Web Service Operation über Proxy rufen (Flugliste)
*
  TRY.
      proxy_flights->flight_get_list( EXPORTING input  = in_flights
                                      IMPORTING output = out_flights ).

    CATCH cx_ai_system_fault .
    CATCH cx_ai_application_fault .
  ENDTRY.

*********************************************************
* Flugliste auf Grundliste ausgeben
* und HIDE Bereich versorgen
*
  DATA wa_flights TYPE bc00bapisfldat.
  LOOP AT out_flights-flight_list-item INTO wa_flights.

    WRITE: /   wa_flights-airlineid,
            5  wa_flights-connectid,
            11 wa_flights-airline,
            25 wa_flights-cityfrom,
            38 wa_flights-cityto,
            50 wa_flights-flightdate,
```

Continued on next page

Internal Use SAP Partner Only

```
            63 wa_flights-deptime,
            75 wa_flights-arrdate,
            88 wa_flights-arrtime.
    HIDE:      wa_flights-airlineid,
               wa_flights-connectid,
               wa_flights-flightdate.
  ENDLOOP.


************************************************************
*  AT LINE-SELECTION  NEW CODING HERE. DONT FORGET HIDE      *
************************************************************
AT LINE-SELECTION.

*********************************************************
* Proxy instanziieren (Details)
*
  DATA proxy_details TYPE REF TO co_bc00bc416_get_details.

  TRY.
      CREATE OBJECT proxy_details
        EXPORTING
          logical_port_name = 'LP_DETAILS'.

    CATCH cx_ai_system_fault .
  ENDTRY.

*********************************************************
* Schnittstelle versorgen: Verwenden Sie
* hierzu die Daten aus dem HIDE Bereich
*
  DATA: in_details  TYPE bc00flight_get_detail,
        out_details TYPE bc00flight_get_detail_response.

  MOVE: wa_flights-airlineid  TO in_details-airline_id,
        wa_flights-connectid  TO in_details-connection_id,
        wa_flights-flightdate TO in_details-flight_date.

*********************************************************
* WS Operation über Proxy rufen (Details)
*
  TRY.
```

Continued on next page


```
      proxy_details->flight_get_detail( EXPORTING input  = in_details
                                        IMPORTING output = out_details ).

    CATCH cx_ai_system_fault .
    CATCH cx_ai_application_fault .
  ENDTRY.

*********************************************************
* Detaildaten auf Verzweigungsliste ausgeben
*
  SKIP.
  ULINE.
  FORMAT COLOR COL_HEADING.
  WRITE: / 'Ausgewählter Flug'(001), AT sy-linsz space.
  FORMAT COLOR OFF.
  ULINE.
  WRITE: / out_details-flight_data-airlineid,
           out_details-flight_data-connectid,
           out_details-flight_data-airline,
           'von'(002),
           out_details-flight_data-cityfrom,
           'nach'(003),
           out_details-flight_data-cityto.
  ULINE.
  WRITE: / 'Abflug:'(004),
        10 out_details-flight_data-flightdate,
        25 out_details-flight_data-deptime.
  WRITE: / 'Ankunft:'(005),
        10 out_details-flight_data-arrdate,
        25 out_details-flight_data-arrtime.
  WRITE: / 'Preis:'(006),
        10 out_details-flight_data-price LEFT-JUSTIFIED,
        25 out_details-flight_data-curr.
  ULINE.
  FORMAT COLOR COL_HEADING.
  WRITE: / 'Verfügbare Plätze'(007), AT sy-linsz space.
  FORMAT COLOR OFF.
  ULINE.
  WRITE: / 'Economy:'(008),     15 out_details-availibility-econofree,
         / 'Business:'(009),    15 out_details-availibility-businfree,
         / 'First Class:'(010), 15 out_details-availibility-firstfree.
  ULINE.
```

Continued on next page

Internal Use SAP Partner Only

Lesson Summary

You should now be able to:

- Generate a proxy using a WSDL document
- Define a logical port
- Call a Web service from an ABAP program

Internal Use SAP Partner Only

Lesson: Web Service Error Analysis

Lesson Overview

The creation and integration of Web services can always give rise to errors. This lesson suggests ways to isolate these errors in the most effective way possible.

Lesson Objectives

After completing this lesson, you will be able to:

- Perform a connection test for an RFC destination
- Carry out a trace record to trace an SOAP request
- Use the ICF Recorder to record request and response cycles

Business Example

You need to analyze error statuses in a Web service application.

Web Service Error Analysis

Calling Web services can always give rise to errors. The ABAP Workbench provides different tools that you can use to quickly pinpoint the cause so that you can isolate the error as fast as possible. Here we will distinguish between error analysis for the Web service client side and error analysis for the Web service server side.

Internal Use SAP Partner Only

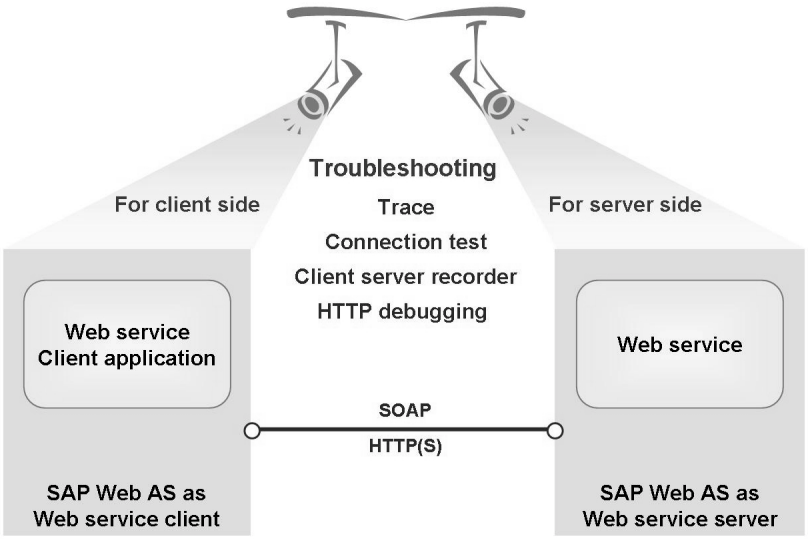


Figure 67: Web Service Error Analysis

Hint: The monitoring methods put forward here for troubleshooting interfere with the performance of your system and should therefore only be activated where this is necessary. We recommend that you monitor the settings and deactivate any settings that are no longer needed.

Error Analysis for the Client Side

If calling a Web service via a Web service client proxy results in errors, you can locate these using the following transactions and tools on the client side:

HTTP destination connection test

Transaction SM59 (Display and Maintenance of RFC Destinations) facilitates a destination connection test. On the initial screen of the transaction, first double-click the required destination to select it, then choose the *Test connection* button.

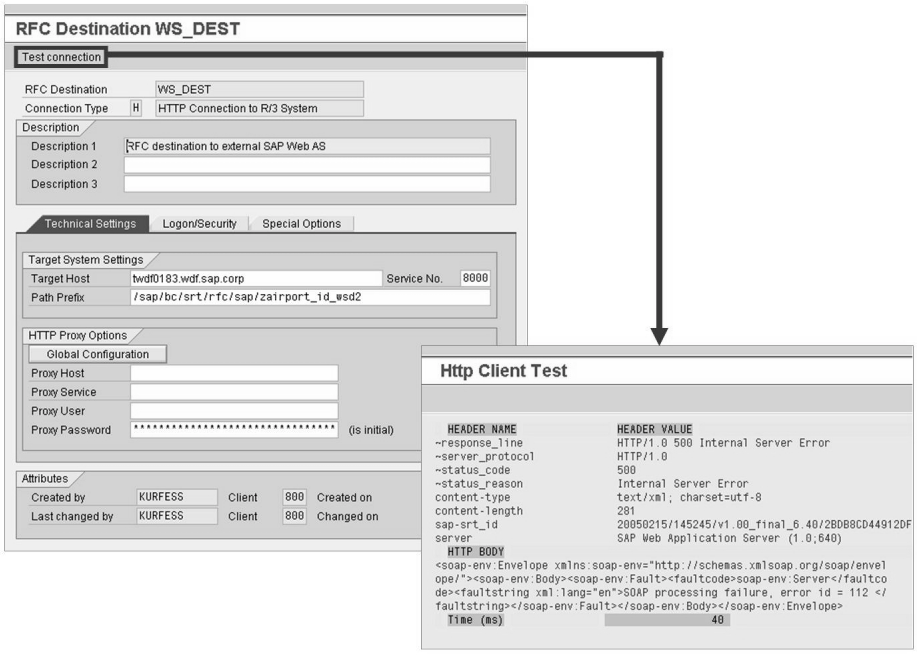


Figure 68: Connection Test in SM59

If you carry out a connection test to an HTTP destination that references the inbound SOAP runtime, you receive a SOAP fault with error code 500. This occurs because the connection test in transaction SM59 using the GET method sends an empty HTTP body from which the inbound SOAP obviously cannot extract a valid SOAP call. However, by performing this test you can guarantee that the server can be reached by using HTTP. You cannot however use this test to check whether the service is functioning correctly.

ICF Client Recorder

The ICF Client Recorder can be used to log client requests. It allows you to see what was sent via HTTP across the network to the remote server, and what this sends back in response. In conjunction with Web services, the SOAP request and the corresponding SOAP response can be analyzed here. For example, SOAP faults processed by the SOAP server can be determined if the SOAP request contains errors. Or, if the server sends back an HTML document as a response instead of an SOAP response, this can also be determined using the recorder.

Choosing *Client → Recorder* within transaction SICF gives you access to the ICF Client Recorder. Three options are available for selection:

- Activate Recording
- Deactivate Recording
- Call ICF Recorder

Activate Recording: Select *Client → Recorder → Activate Recording* to activate the *ICF Client Recorder*. Entering a user name allows you to restrict the recordings to that user. Under *Client URL Path*, enter the request path that you want the *ICF Recorder* to record. You can also set the *Record Time* and *Lifetime* (storage period of the monitoring data on the database). By default, only the requests are recorded here. If you also want to record the responses, you need to select *Request + Response* as the *Recording Level*.

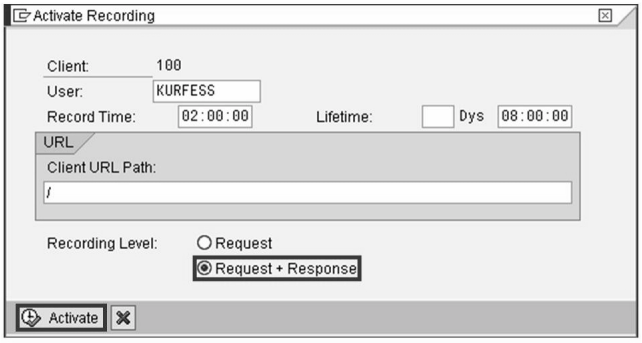


Figure 69: Activate ICF Client Recorder

Deactivate Recording: Select the *Client → Recorder → Deactivate Recording* menu path to fully deactivate the ICF Client Recorder again in order to prevent performance interruptions. On the next screen, you need to select the user and choose *Deactivate*.

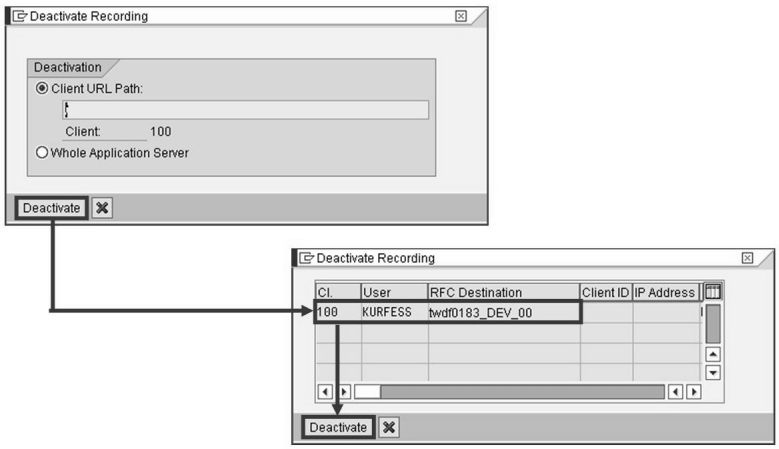


Figure 70: Deactivate ICF Client Recorder

Display Recording: Select the *Client → Recorder → Display Recording* menu path to display recorded ICF communication data. In the following dialog box you have the option of entering restrictive criteria, similar to the recording:

- Logon Date: Sets the date of logon.
- Logon Time: Sets the time of logon.
- Request Path: Path to the recorded service.
- Logon by User: Enter the name of the recorded user.
- Recording by User: Enter the name of the user who is doing the recording.
- Processed Requests: Choose whether you want to display processed requests.
- Logon error: Choose whether you want to display logon errors.

If you want to view your own recorded services, enter only your user name in the *Recording by User* field and leave the *Logon by User* field empty.

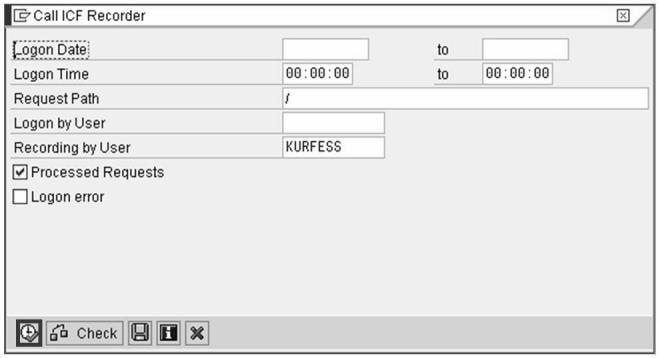


Figure 71: Display Recording

On the following screen, choose the *Client Requests* button from the application toolbar to view the recorded client request entries. Next, select a recorded entry. Then click the *Display* icon to view, for example, the request or the response.

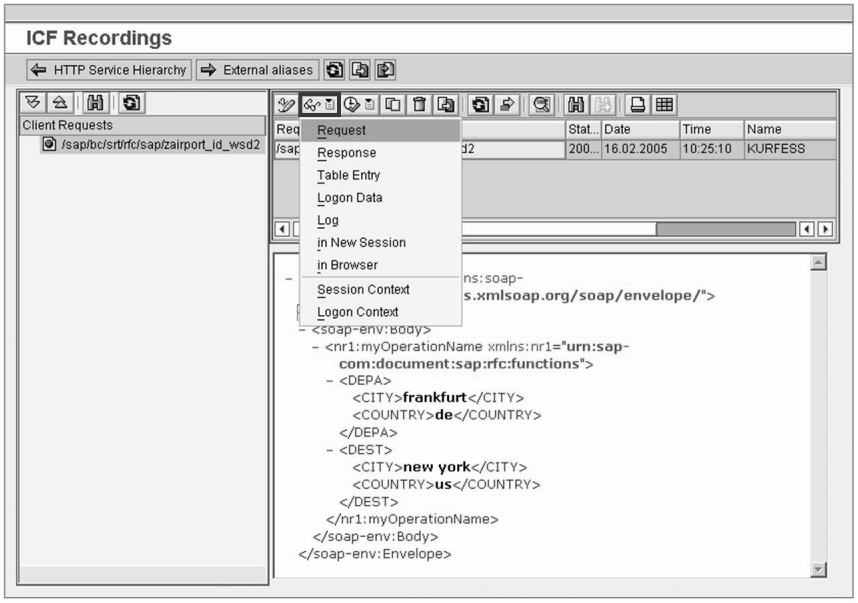


Figure 72: Display Recording

Hint: You can record other users' ICF communication data , but you require additional authorizations to view this data. As a basic principle, you need authorization for the SICFRECORDER transaction before you can edit the recorded data. If you have the appropriate authorizations, you can edit the recorded communication data. You can only edit your own entries in the absence of additional authorizations. Authorization Object: *S_ICFREC*

SOAP runtime trace

The SOAP runtime trace is used to save complete SOAP request and response documents from the caller to the trace file. This trace uses the error log files of the RFC (for example, dev_rfc<NR>). You can view these within transaction SM59 by choosing *RFC → Display Trace*.

To switch on the SOAP runtime trace on the client side, select your logical port in transaction LPCONFIG and switch to the detail view. You can now define the trace level on the *Error* tab page in the *General Settings*. The following values can be selected for tracing:

- No trace: Tracing is switched off
- Errors only: Errors are logged in the trace file
- Payload only: Requests, responses, and errors are logged in the trace file
- Full trace: Full log of trace information

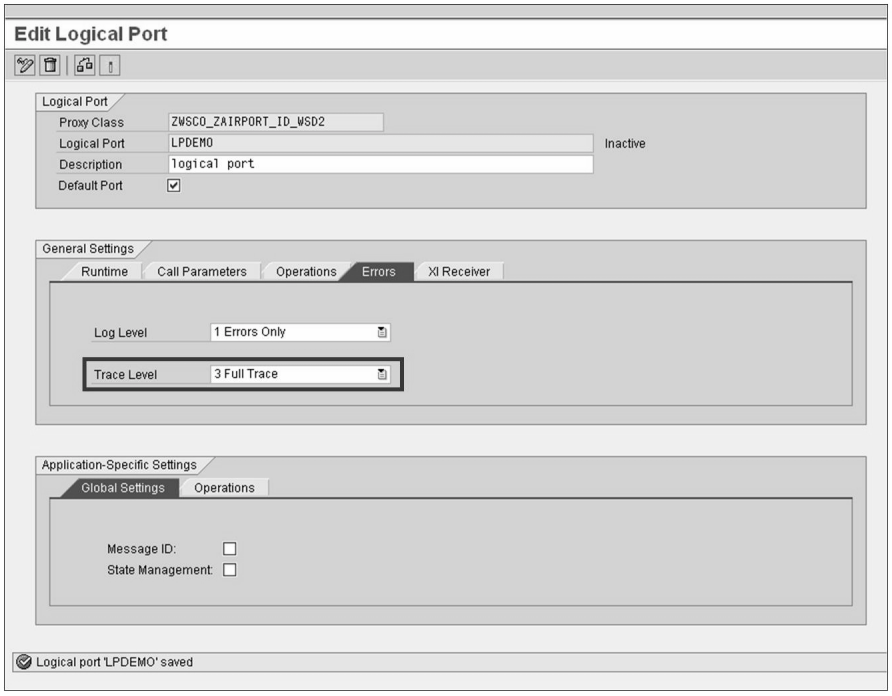


Figure 73: SOAP Runtime: Activate Trace

You can set logging by choosing the *Log Level*. The messages (SysLog entries) are written to the SAP system log and can be read via transaction SM21.

Before you can start gathering the trace information for an error situation, you should first restore the existing trace files. You can delete the trace file in transaction SM59 by choosing *RFC → Delete trace*.

You can view the trace file in transaction SM59 by choosing *RFC → Display Trace*.

Figure 74: SOAP Runtime: Display Trace

Hint: When using the trace you must make sure that you switch the trace off again. If you forget to do this, large trace files will be created.

Error Analysis for the Server Side

Similar to error analysis on the client side, the IFC Recorder can also be used for error analysis on the server side to monitor the incoming SOAP requests. You also have the option of recording SOAP runtime traces. Troubleshooting should also involve a check to see whether the related service node is active in transaction SICF.

ICF Server Recorder

You can use the ICF Server Recorder in transaction SICF to record the incoming SOAP request and the associated response. Proceed as follows:

- To activate the recorder, in transaction SICF choose *Edit* → *Recorder* → *Activate Recording*
- To deactivate the recorder, in transaction SICF choose *Edit* → *Recorder* → *Deactivate Recording*
- To view the recorder, in transaction SICF choose *Edit* → *Recorder* → *Display Recorder*

SOAP Runtime Trace

Activate trace and log: To implement the trace and log settings **across the whole of the system**, select *Goto → SOAP Runtime Settings* on the initial screen of the transaction WSADMIN to access the relevant maintenance screen. Select *Use System Settings for All Services* and implement the required trace and log settings. You can also specify whether you want to allow trace settings to be set by client requests.

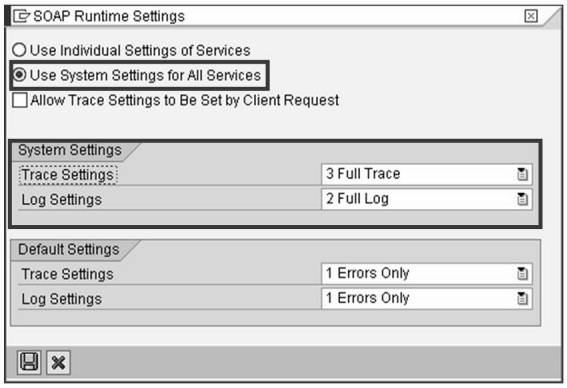


Figure 75: System-Wide Trace and Log Settings

To implement **individual** trace and log settings for a specific Web service, in the initial screen of transaction WSADMIN select your Web service definition by double-clicking it. You can implement individual trace and log settings specially for this Web service from the following screen.

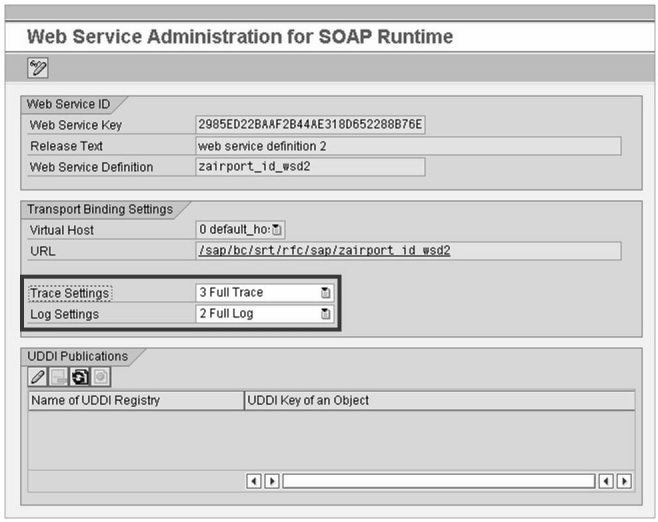


Figure 76: Individual Trace and Log Settings

View Trace and Log Settings: You can display the trace log in the transaction SM59 by selecting *RFC → Display Trace*. Transaction SM21 contains the SysLog entries for canceled Web services.

Lesson Summary

You should now be able to:

- Perform a connection test for an RFC destination
- Carry out a trace record to trace an SOAP request
- Use the ICF Recorder to record request and response cycles

Internal Use SAP Partner Only

Lesson: Introduction: UDDI

Lesson Overview

In this lesson, you will learn about the significance of UDDI as a component of the Enterprise Services Architecture (ESA). The predefined usage scenarios illustrate how UDDI is used. You will be shown the steps required for integration between ABAP Web services and UDDI.

Lesson Objectives

After completing this lesson, you will be able to:

- Understand the core meaning of UDDI in the Web service environment.
- Name the possible scenarios in which UDDI can be used in the ABAP Web service environment.
- Name the general technical conditions for using UDDI in local or public environments.

Business Example

A CIO wants to implement a standardized information platform so that both, himself and his employees, can formally describe and search for existing software components within the company. Rather than accessing Microsoft Office tools or individual tables to do this, he wants to access a freely available repository. He asks you to examine the ways in which Universal Description, Discovery and Integration (UDDI) can be used for the company.

Web Service Catalog

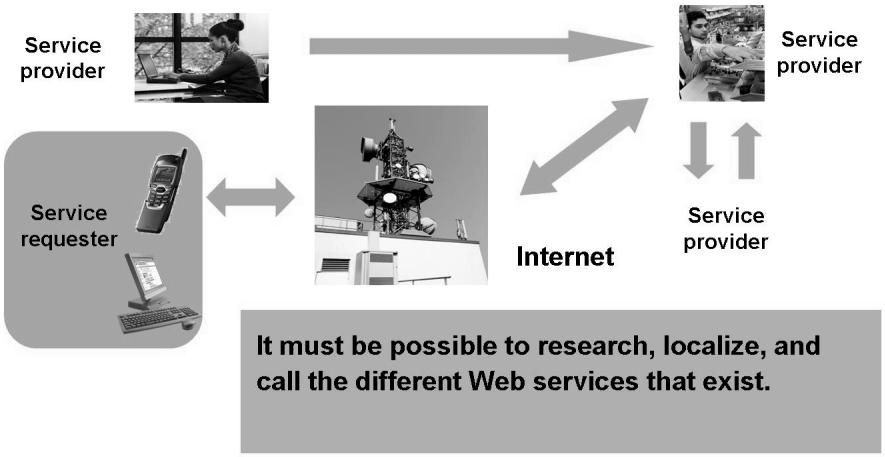


Figure 77: Web Service Requirements

The increasing number of business services within a company leads to an unmanageable number of possible implementation services. In some cases, these services are not used because they cannot be located. The use of a central repository helps employees locate the distributed services in a company. UDDI allows you to classify services according to standards and store all of the necessary information for the technical connection. This information can be transferred either purely within the company or across the Internet. SAP NetWeaver provides this infrastructure as standard in SAP NetWeaver Application Server.

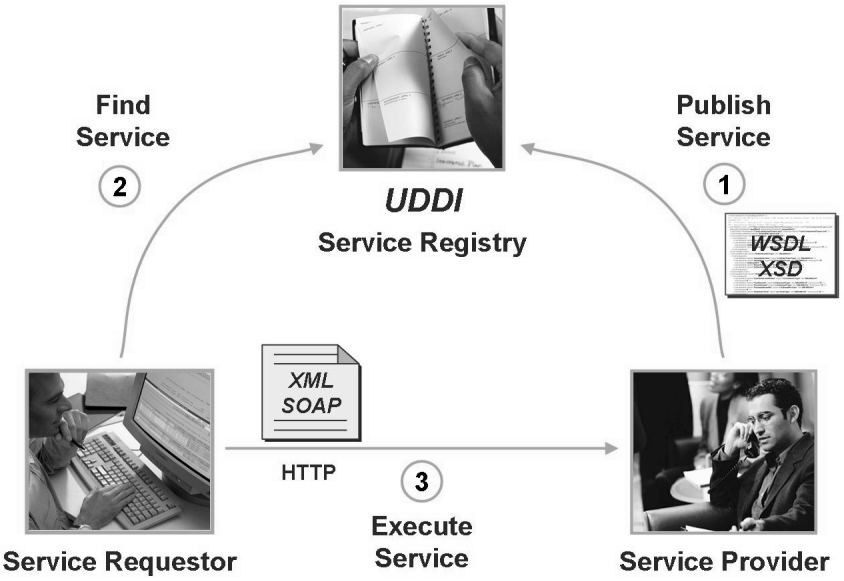


Figure 78: Web Service Paradigm

1. A service provider adds a description of his or her (business) Web services to the UDDI repository.
2. A user searching for a service queries the UDDI repository and finds out what services are offered by the service provider.
3. The service user can now use the WSDL provided to access the service by agreement.

Information is provided and queried via the standardized UDDI API.

UDDI is a single, standardized platform for a service marketplace. A UDDI instance must be used to allow a user searching for a service find a service provider, and vice versa. UDDI provides this as a Web service. The implementation type of the UDDI registry service is not significant as its interfaces are standardized as a Web service.

UDDI Definition

- UDDI = Universal Description, Discovery, and Integration
- Directory for describing companies and their services ("Yellow Pages" for services)
- Provides a standardized description of services
- UDDI itself represents a Web service implementation and provides the functions of the UDDI registry using a SOAP interface.
- NOT: It does not describe the legal expiration of business relations or of temporal contractual commitments.

Figure 79: What is UDDI?

UDDI is a repository service for describing companies and their services ("Yellow Pages" for services). It provides a standardized description of services. UDDI itself represents a Web service implementation and provides the functionality of the UDDI registry via a SOAP interface. UDDI is not a description of the legal expiration of business relations or of temporal contractual commitments.

White pages

Information such as name, address, telephone number, and contact information for a company

Yellow pages

Information that categorizes the company. Based on standard classified directories

Green pages

Technical information about the Web service

Figure 80: UDDI Business Registry Areas

Using the categorized and standardized entries in the Yellow Pages, a user searching for a service finds the provider of a specific service, such as a wine supplier, for example. The White Pages describe additional attributes of the wine supplier, such as location, telephone number, and contact data. If you want to place an order via an available Web service, you can find the technical description, such as the URL of the WSDL, in the Green Pages.

UDDI offers three basic functions:

- Publish
  Register Web services
- Find
  Find Web service
- Bind
  Describes how a connection to a Web service is set up and how this Web service is utilized

Figure 81: Basic UDDI Functions

The *Publish* and *Find* functionalities are processed via separate, independent HTTP ports. Thus, user and provider access the registry via different ports. Binding describes the modes of accessing the Web service without the active input of the UDDI registry.

To provide a rough introduction to UDDI functionality, the SOAP operations in UDDI API 1.0 are listed here by way of example.

Inquiry API

- Find things
  - find_business
  - find_service
  - find_binding
  - find_tModel
- Get details about things
  - get_businessDetail
  - get_serviceDetail
  - get_bindingDetail
  - get_tModelDetail

Publication API

- Save things
  - save_business
  - save_service
  - save_binding
  - save_tModel
- Delete things
  - delete_business
  - delete_service
  - delete_binding
  - delete_tModel
- Security
  - get_authToken
  - discard_authToken

Figure 82: UDDI API 1.0

Configuration of a Local UDDI Registry

UDDI can be used as a local instance in the company. This allows all of the Web services (business services) in the company to be categorized and searched. UDDI can be used as the basis for locating sources of information in a department or subsidiary, for example.

The following requirements apply when using the UDDI repository integrated with *SAP NetWeaver*:

- *SAP NetWeaver AS Java* is installed
- *SAP Java Cryptographic Toolkit* must be installed (optional, only if encryption is used)
- J2EE administration permissions

SAP NetWeaver AS Java allows you to use a local UDDI registry. As in the case with Web services, the next step is to check the purpose of the UDDI registry. If the registry is for internal use only, integrated access control with the associated access permissions is generally sufficient. If the information is transported over a public network and published, it is essential that a security strategy is created and agreed with the IT security managers in the company. The encryption of publishing access can be configured with SSL, for example.

Use the *SAP J2EE Visual Administrator* to activate the UDDI registry and define a connection to *SAP NetWeaver AS Java*. To log on, choose *Connect*:

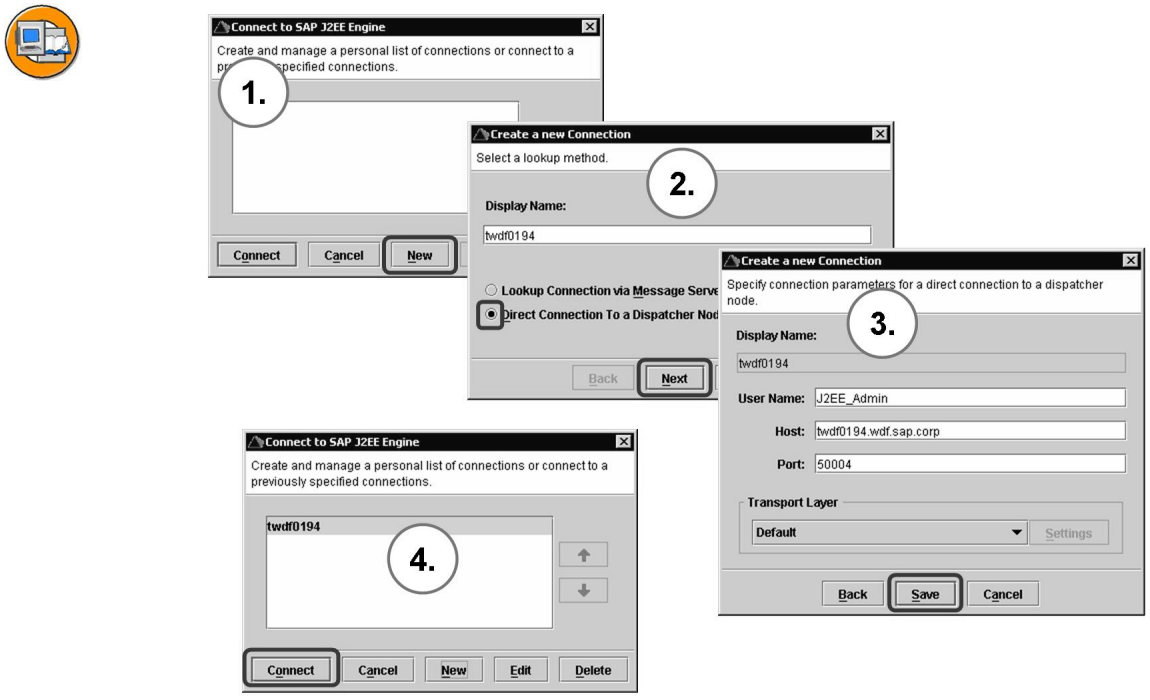


Figure 83: Setting Up the Connection in SAP J2EE Visual Administrator

You are now in the *SAP J2EE Visual Administrator* and can set up the UDDI registry.

SAP NetWeaver AS JAVA already contains an empty UDDI registry. The tables must be populated with initial data before they are used for the first time. This is done by choosing *Reset DB*.

Hint: If the message *Base Models and Taxonomies have not been preloaded* is not displayed, this means that the tables have already been initialized. If you initialize the tables again, existing data will be deleted.

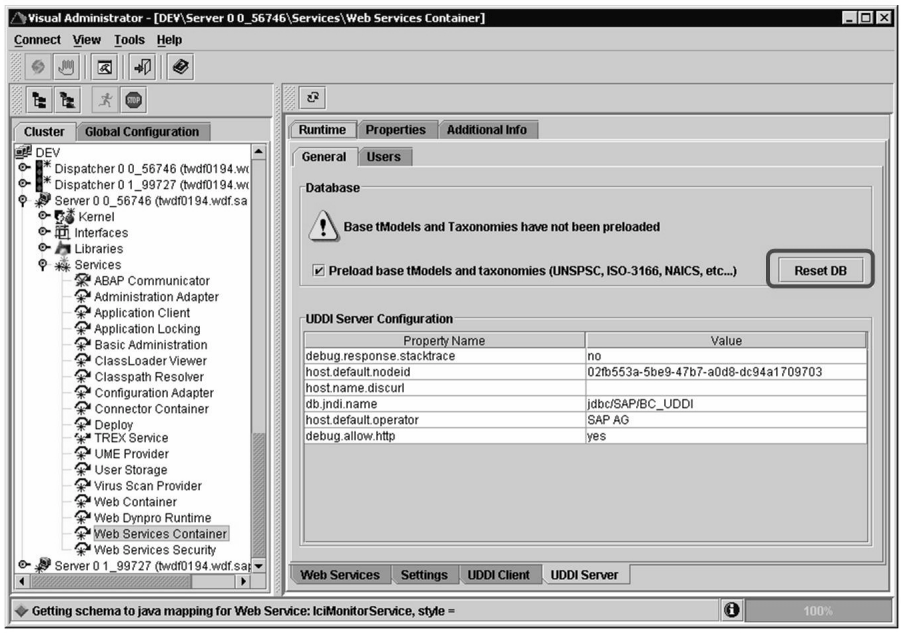


Figure 84: Initializing the Tables

A maximum of one UDDI instance can exist for each SAP system.
You can access the instance by using the URL for the J2EE stack.
Query: http://<hostName>:<portNummer>/uddi/api/inquiryPublishing: http://<hostName>:<portNummer>/uddi/api/publish http://<hostName>:<portNummer>/uddi/api/publish

Setting Up the UDDI User

After basic initialization, the UDDI user has to set up. These users are created for maintaining the UDDI content only. A distinction is made between the following user groups:

UDDI User Group	Authorization
Level Tier 1	The user can create the following: ▪ 1 business entity ▪ 4 business services ▪ 2 binding templates for each business service ▪ 100 tModels
Level Tier N	The user can create an unlimited number of objects
Level Admin	The user can create and delete an unlimited number of objects

Figure 85: UDDI: Access Levels

As specified in the figure above, tier 1 users are only permitted to create a limited number of objects in the registry.

To create a UDDI user, in *SAP J2EE Visual Administrator* choose *UDDI Server* → *Users* and then *New User*:

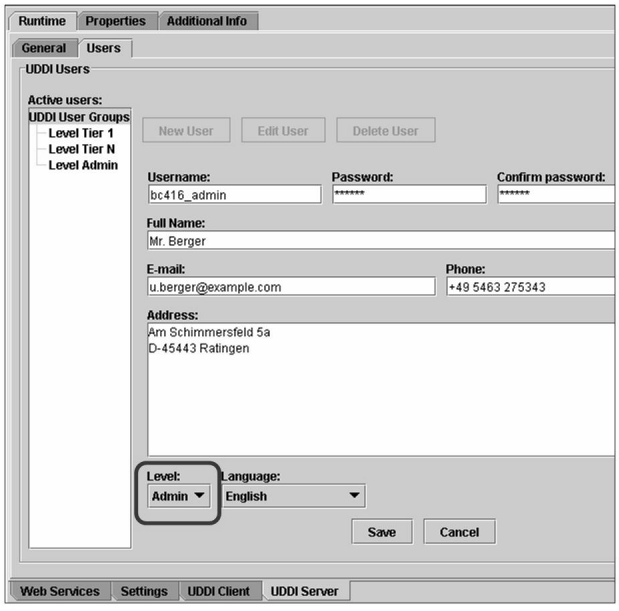


Figure 86: UDDI User Management in J2EE/Visual Administration Console

In addition to the name of the user, the entry screen for creating UDDI users also specifies the authorization level. After you have saved the user, you can find the new users under the specified user group.

Defining UDDI Access

On the *UDDI Client* tab page, choose *New Registry*. Give the UDDI registry a name and maintain the inquiry URL and publish URL. Next, save the configuration by choosing *Save*:

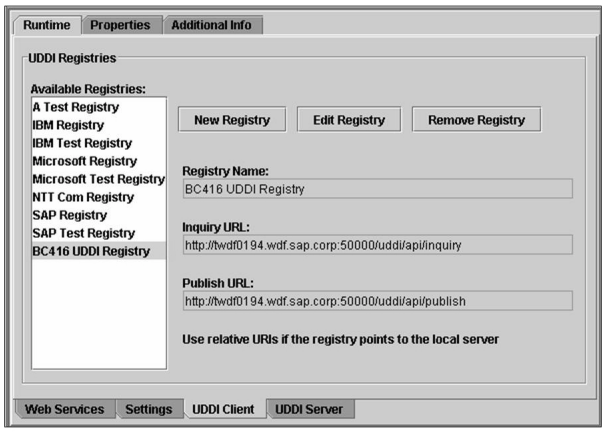


Figure 87: Defining UDDI Access for SAP NetWeaver AS

With this configuration, you can now search in and publish to the UDDI registry.

Browser Access to Local UDDI Registry

You can access the UDDI client via the *SAP NetWeaver AS* initial screen. The URL is http://<hostName>:<portNummer>/index.html.

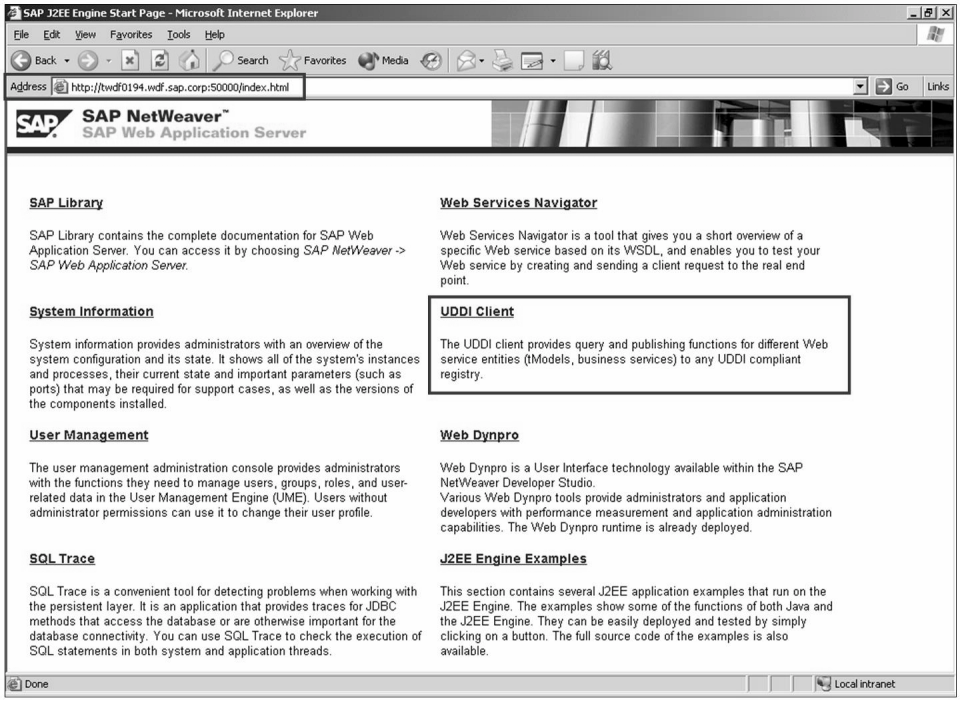


Figure 88: SAP NetWeaver AS Initial Screen

Before connecting to the UDDI registry, the user must select the required action, for example, *Search Registry*, as well as the UDDI instance.

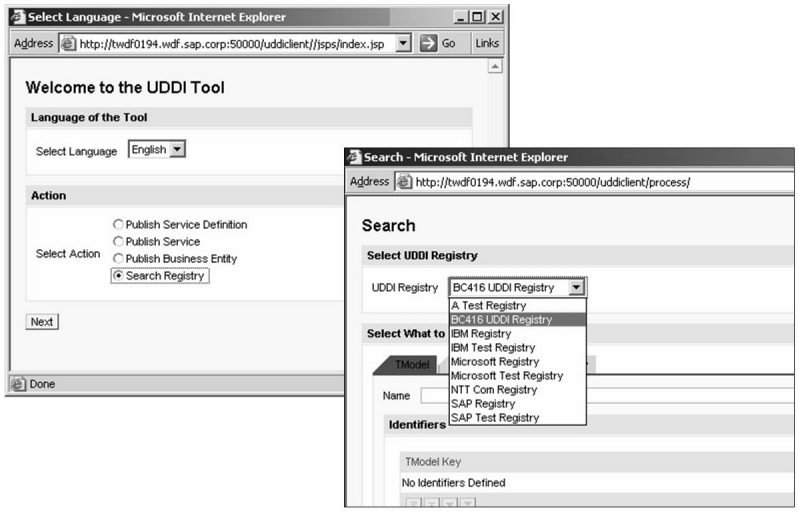


Figure 89: Accessing the UDDI Browser Client

Internal Use SAP Partner Only

Accessing a Local UDDI Registry from the ABAP Stack

To create a Web service WSDL in the UDDI registry from the ABAP stack, a connection must be defined between the ABAP and Java stacks via external HTTP destinations. You must maintain two destinations for UDDI registries: one destination for searching, and one for publishing Web services.

To create an HTTP destination, call transaction SM59 and choose *Create*. Select *G - HTTP Connection to External Server* as the connection type.

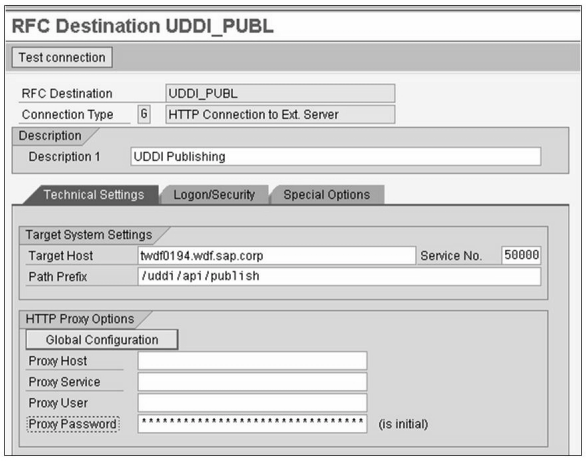


Figure 90: External HTTP Connection for UDDI Publishing Access

The above figure shows the connection for maintaining UDDI information in an external registry. The host name, port number, and path prefix can be different. You configure the search API as follows:

Internal Use SAP Partner Only

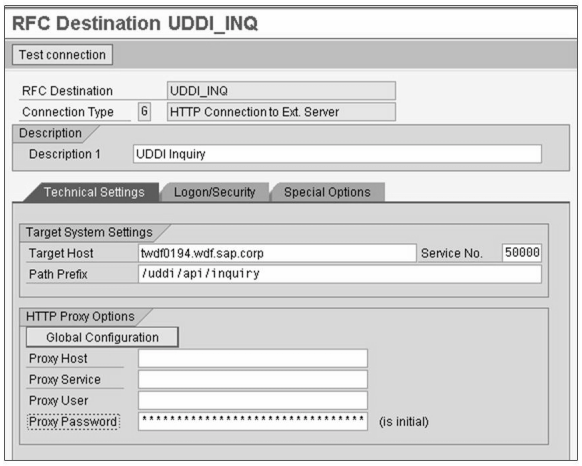


Figure 91: External HTTP Connection for UDDI Inquiry Access

The connection to the UDDI registry is now defined. The next step is to declare the registry for the ABAP development environment using transaction SUDDIREG:

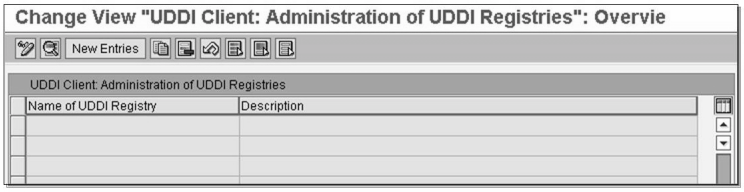


Figure 92: SUDDIREG: Creating a New UDDI Connection

If there have been no UDDI registries used in the system up to this point, the list of registries will be empty. Choose *New Entries* to create a new entry. To do so, switch to change mode. The UDDI registry logon can be preconfigured with the parameters *User*, *Password*, *Default Business Entity*.

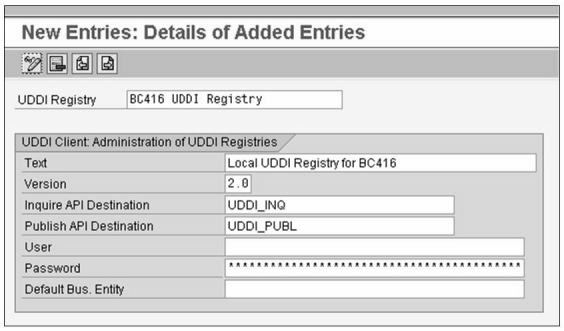


Figure 93: Using External HTTP Destinations for the UDDI Registry Service

You can now register the Web service in the local registry. There are two options available here, one of which can be implemented successfully without prior configuration of the UDDI tModel.

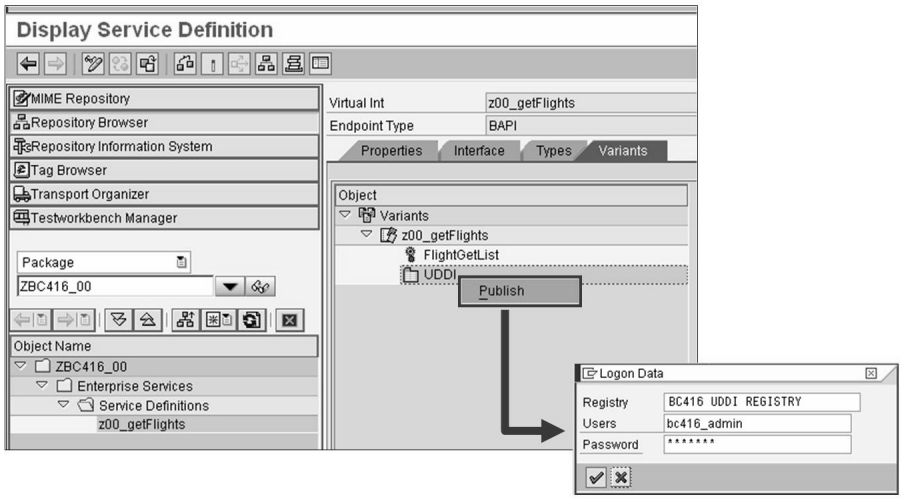


Figure 94: Registering a Web Service in UDDI

After you select the context menu entry *Publish*, you will be prompted to enter a user name and password. Enter the UDDI user you created here.

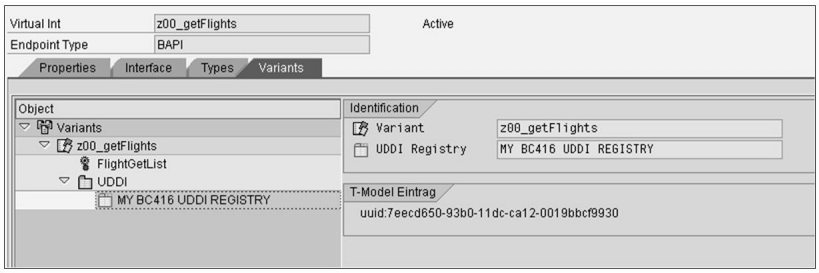
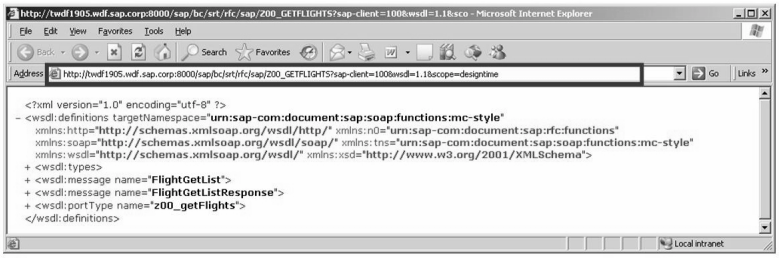


Figure 95: Registering a Web Service in UDDI

You can now define a business entity or detailed description for the service in the UDDI registry. If you choose the *Display* icon, the Web browser will display the properties of the Web service in the UDDI registry.

The WSDL, which is published via UDDI, contains less information. The service name and binding are not passed on: the Web service consumers must transmit this information using another method.

WSDL in the UDDI registry:



WSDL of the Web service home page:

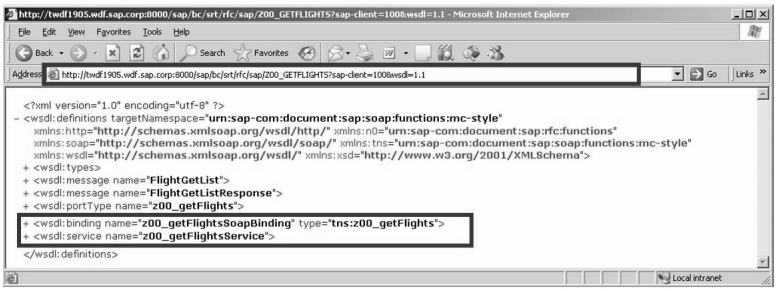


Figure 96: WSDL Components

To maintain the properties, open the UDDI registry via the initial *SAP NetWeaver AS JAVA* page under the URL http://<hostName>:<portNummer>

Creating a Business Entity

To create a business entity in the UDDI registry, start a Web browser and go to the URL http://<hostName>:<portNummer>/uddiclient. Choose *Publish Business Entity* and choose *Next*. Select the UDDI registry that you created earlier, and for which you already created a client entry in the J2EE visual administration console, and enter your UDDI user details.

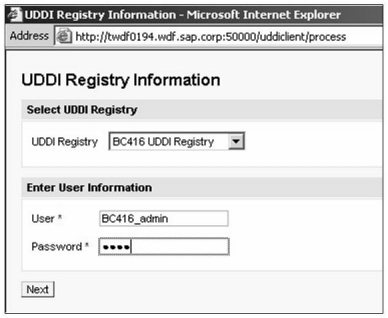


Figure 97: Creating a Business Entity

Choose *Create Business* to create a new business entity.

Figure 98: Creating a Business Entity

Select the *Names* tab and choose *New Name*. Enter the name of the business unit on the following screen.

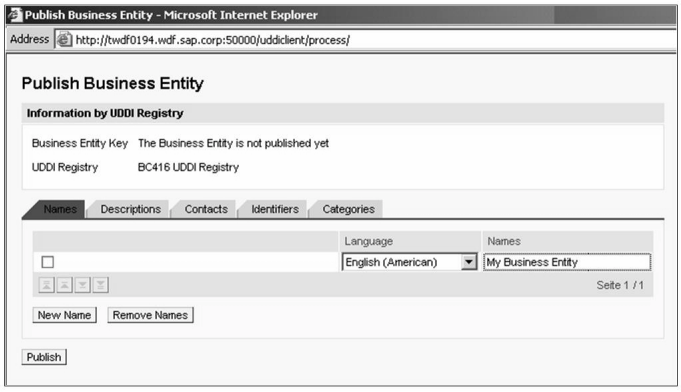


Figure 99: Creating a Business Entity

You can store various languages, contacts, key flags, and categories to the UDDI-specific information for the business entities. This information is then used to identify and classify the business entity.

After you have entered the required data, choose *Publish* to publish the data.

Hint: If you have logged on as a standard user (tier 1), you can only publish one business entity.

Integrating an External UDDI Registry for ABAP

A UDDI registry is accessed via two HTTP ports. An external RFC connection is set up in the basis system for this. The connection is defined in the same way as described for the locally available UDDI registry. As this involves accessing the Internet, technical security issues must be resolved with the IT security manager beforehand. Unlike the internal UDDI registry, you may need to configure a proxy and SSL access for the publish path.

Proxy entries must be defined if the UDDI sources or destinations need to be addressed directly from the Internet and the associated HTTP packets are transferred via a proxy. You will be familiar with similar settings from your Internet browser.

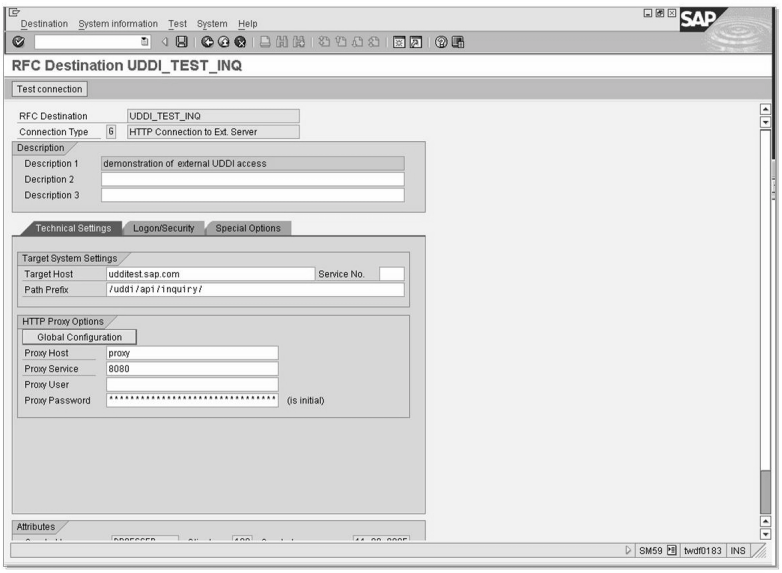


Figure 100: Accessing the Public SAP UDDI Test UDDI Registry

WSDL Transfer from the UDDI Registry

The basic procedure for generating Web service proxies is described in the lesson *SAP NetWeaver Application Server as a Web Service Client*. This section focuses on explaining how a WSDL document is transferred from a UDDI registry.

To be able to transfer a WSDL document from the UDDI, you must first define the registry as a client in transaction SUDDIREG (see*Accessing a Local UDDI Registry from ABAP*).

During Web service proxy generation, UDDI is specified as the source.

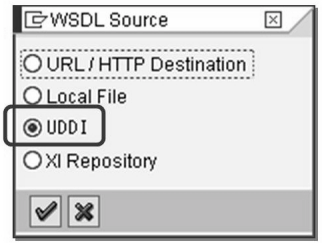


Figure 101: UDDI as WSD Source

The subsequent selection list contains the UDDI client entries that were defined using transaction SUDDIREG.

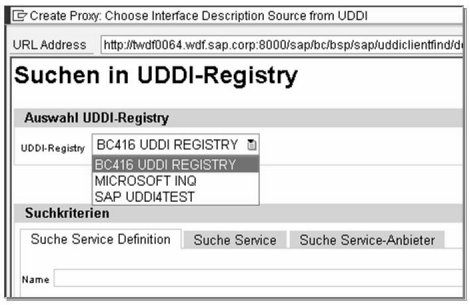


Figure 102: Choosing the Local UDDI Registry

You can use the % sign as a wildcard operator to start your search for a Web service (for example, %AIRPORT%). Select the required service definition from the hit list and familiarize yourself with the description of the service definition.

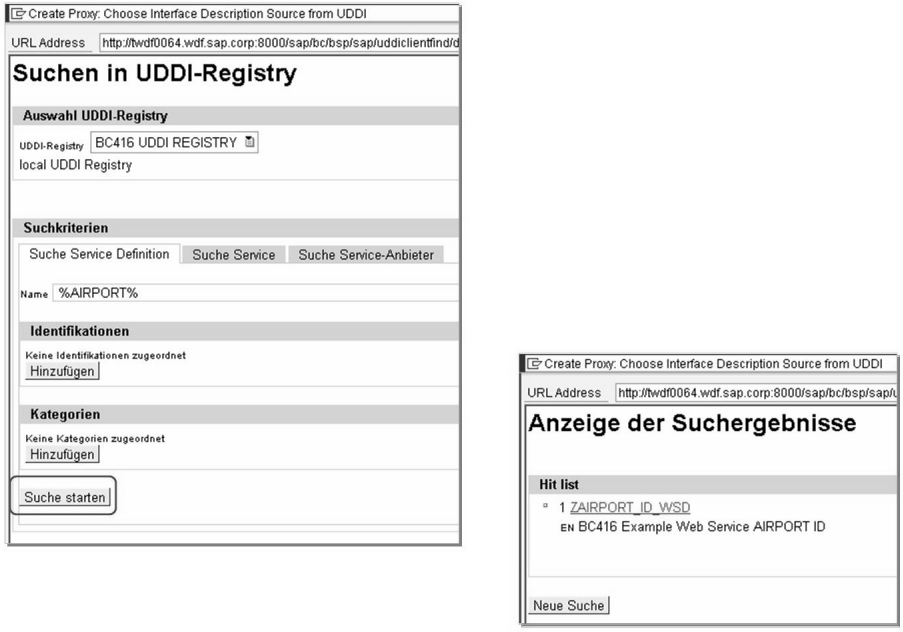


Figure 103: Searching for a Service Definition in the UDDI Registry

Switch the service definition tab to *Service Definition Details* and choose *URL* in order to select the WSD (1). The button may be hidden because of the length of the URL in the window area. Therefore move the scroll bar to the right. After you have transferred the URL, choose *Continue* (2) to exit the dialog box and continue the proxy generation process in transaction SE80.

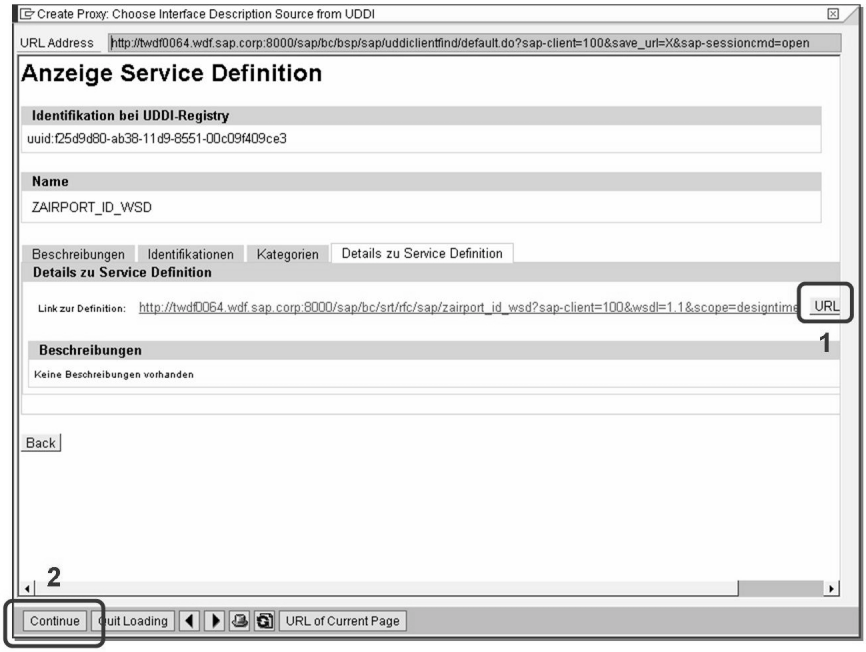


Figure 104: Transferring the WSDL for the Web Service Proxy Generator

Exercise 7: UDDI (Optional)

Exercise Objectives

After completing this exercise, you will be able to:

- Create a connection between Web service providers and Web service consumers using the UDDI registry
- Identify and use a Web service in a UDDI registry

Business Example

The exchange of information in a company needs to be managed in a way that is standardized and bundled. UDDI can be used as a suitable platform here. The following exercise shows how UDDI is utilized in a Web service development cycle.

Task 1: Activate UDDI Functions

It will soon be possible to publish services in a UDDI-based *Services Registry*. The *Services Registry* will contain *Enterprise Services* definitions and references to their metadata. *Enterprise Services* are classified using *classification systems* and can be found using these classification criteria.

You can continue to use the UDDI functions described in this lesson, if required: The prerequisite for this is that users are assigned the parameter *OLD_UDDI* with the parameter value *X* in User Administration.

1. Call the transaction SU01 (user maintenance). On the initial screen, enter your user *BC416-##* and switch to change mode. Go to the *Parameters* tab page and maintain the parameter ***OLD_UDDI*** with the parameter value ***X***.

Task 2: UDDI Publication

The ABAP Web services you created in the exercises for the lesson *SAP NetWeaver Application Server as Web Service Server* have been released. The Web service consumers do not have information about how to use this Web service. In this exercise you will publish the released Web services in the local UDDI registry of the training system.

1. Call the transaction SE80 and register the ABAP Web service ***z##getFlights*** in the local UDDI registry of the training system.

Continued on next page

Task 3: UDDI Drilldown Reporting and Binding

As the service requester, you now need to search for a Web service that can provide your business application with the necessary information.

1. Use the proxy generation function in the SE80 transaction to create a Web service client access point from the previously generated Web service entry in the local UDDI registry of the training system.

Internal Use SAP Partner Only

Solution 7: UDDI (Optional)

Task 1: Activate UDDI Functions

It will soon be possible to publish services in a UDDI-based *Services Registry*. The *Services Registry* will contain *Enterprise Services* definitions and references to their metadata. *Enterprise Services* are classified using *classification systems* and can be found using these classification criteria.

You can continue to use the UDDI functions described in this lesson, if required: The prerequisite for this is that users are assigned the parameter *OLD_UDDI* with the parameter value *X* in User Administration.

1. Call the transaction SU01 (user maintenance). On the initial screen, enter your user *BC416-##* and switch to change mode. Go to the *Parameters* tab page and maintain the parameter ***OLD_UDDI*** with the parameter value ***X***.

   a) If you have questions, ask your instructor.

Task 2: UDDI Publication

The ABAP Web services you created in the exercises for the lesson *SAP NetWeaver Application Server as Web Service Server* have been released. The Web service consumers do not have information about how to use this Web service. In this exercise you will publish the released Web services in the local UDDI registry of the training system.

1. Call the transaction SE80 and register the ABAP Web service ***z##getFlights*** in the local UDDI registry of the training system.

   a) In the *Object Navigator* under Enterprise Services, double-click on the service definition *z##_getFlights* to select it for processing.

   b) Choose the *Variants* tab. Position the cursor below the service definition on *UDDI* and choose *Publish* from the context menu.

   c) In the dialog box, select the local UDDI registry. Use the UDDI user whose ID was provided to you by your course instructor.

Continued on next page

Task 3: UDDI Drilldown Reporting and Binding

As the service requester, you now need to search for a Web service that can provide your business application with the necessary information.

1. Use the proxy generation function in the SE80 transaction to create a Web service client access point from the previously generated Web service entry in the local UDDI registry of the training system.

   a) Call transaction SE80, select your package *ZBC416_##*, and right-click to switch to the Web service proxy generation. Choose UDDI as the source and select the local UDDI registry. Search for your own Web service and select it. To do this, go to Service Definition and choose *Save URL*. Choose *Continue*. The Web service client proxy is displayed in transaction SE80 as an ABAP object class. Since you already created the proxy in the exercise for lesson *SAP NetWeaver Application Server as Web Service Client*, the existing proxy is replaced only. You can therefore dispense with the remaining part of the generation process.

Lesson Summary

You should now be able to:

- Understand the core meaning of UDDI in the Web service environment.
- Name the possible scenarios in which UDDI can be used in the ABAP Web service environment.
- Name the general technical conditions for using UDDI in local or public environments.

Internal Use SAP Partner Only

Unit Summary

You should now be able to:

- Classify the ICF in the SAP Web AS ABAP.
- Use transaction SICF to browse ABAP HTTP services.
- Create an ABAP Web service based on an RFC-enabled function module.
- Describe the functions of the *service interface* and the *variant*.
- Describe the *Web service homepage*.
- Describe the structure of a WSDL document
- Discuss the importance of WSDL in the Web services environment
- Generate a proxy using a WSDL document
- Define a logical port
- Call a Web service from an ABAP program
- Perform a connection test for an RFC destination
- Carry out a trace record to trace an SOAP request
- Use the ICF Recorder to record request and response cycles
- Understand the core meaning of UDDI in the Web service environment.
- Name the possible scenarios in which UDDI can be used in the ABAP Web service environment.
- Name the general technical conditions for using UDDI in local or public environments.

Unit 4

Preview

Unit Overview

This unit provides a preview of the SAP Exchange Infrastructure (SAP XI) and e-business standards.

Unit Objectives

After completing this unit, you will be able to:

- Understand the advantages of *SAP NetWeaver XI* compared with peer-to-peer connections
- Understand why eBusiness standards are required
- Describe how eBusiness standards based on Web services can be used as well as the associated requirements
- Describe the tasks that need to be carried out to allow applications to communicate using eBusiness standards
- Describe which *SAP NetWeaver* technology solutions are available for using different eBusiness standards directly

Unit Contents

Lesson: SAP NetWeaver XI and Web Services

Lesson Overview

Differentiation between ABAP Web services and SAP NetWeaver Exchange Infrastructure (SAP NetWeaver XI)

Lesson Objectives

After completing this lesson, you will be able to:

- Understand the advantages of *SAP NetWeaver XI* compared with peer-to-peer connections

Business Example

You are using an ABAP Web service and are wondering what advantages there are in using the SAP Exchange Infrastructure.

Historical Development of Interfaces

Most companies' system landscapes have expanded over time. Business requirements (partly resulting from globalization) mean that business processes more and more frequently involve multiple different systems, some of which are even owned by other companies. Since new interfaces are usually added to the system landscape one at a time, each individual interface is integrated partly by using appropriate connectors, without giving consideration to a universal structure for all interfaces in the system landscape. This results in numerous individual peer-to-peer connections (P2P), which have to be maintained individually.

In a complex system network, peer-to-peer connections present a real challenge in the event that an interface needs to be changed, or if maintenance or a release upgrade is pending.

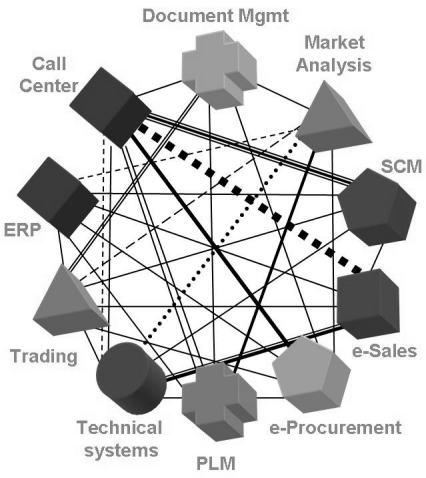


Figure 105: Interfaces with Peer-to-Peer Connections (P2P)

The figure shows that the number of individual interfaces grows exponentially as the number of systems increases. Since individual systems use different protocols and data structures, some kind of mapping is always required. The aim of SAP NetWeaver XI is to channel all of these interfaces using one Integration Server. All the interfaces are to be standardized and documented centrally, and the total number of connections reduced because each individual system now needs to communicate only with the Integration Server. The Integration Server supports many protocols and provides a means of mapping data structures to one another.

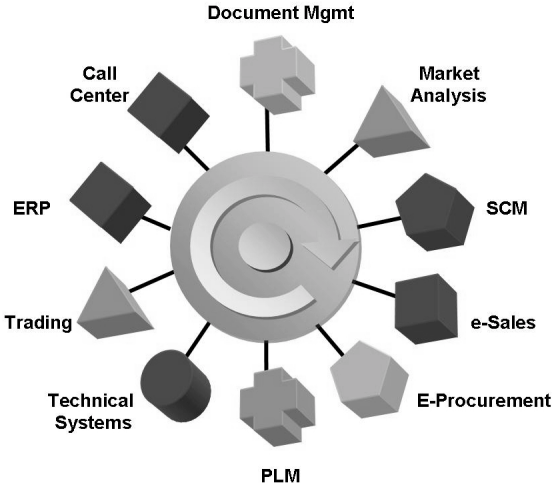


Figure 106: SAP XI as a Message Bus and Central Information Point

SAP NetWeaver XI serves as a message bus for the exchange of electronic messages between systems, receiving messages and forwarding them on, as well as undertaking protocol and format conversions.

SAP NetWeaver is SAP's application and integration platform. *SAP NetWeaver XI* is an integration broker that forms part of the *Process Integration* area of *SAP NetWeaver*.

SAP NetWeaver XI, as part of *SAP NetWeaver*, is based on the *SAP NetWeaver Application Server* and is subdivided into a whole range of components.

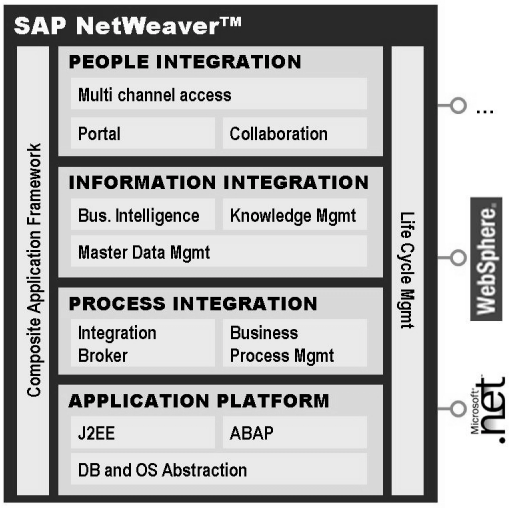


Figure 107: Components of SAP NetWeaver

Components of SAP NetWeaver Exchange Infrastructure

SAP NetWeaver XI is a runtime environment that assigns the inbound messages to the relevant receivers, and, if applicable, maps them to another structure or protocol. To do so, SAP NetWeaver XI requires information about how the messages are to be processed. This information is provided by the *Integration Repository* and *Integration Directory* components of the *Integration Builder*, which contain all the design and configuration objects of the interfaces, respectively. The *System Landscape Directory (SLD)* contains information about which systems are connected to SAP NetWeaver XI for exchanging data. The *Adapter Engine* provides information about the protocols and addresses to be used to access each system. The monitoring tools enable all messages that pass through SAP NetWeaver XI, regardless of their origin, to be monitored centrally.

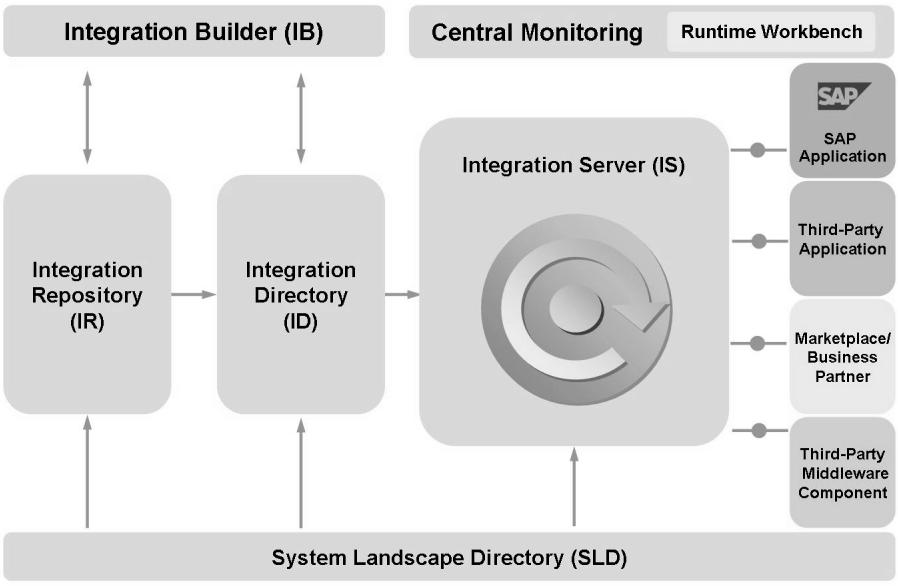


Figure 108: Overview of the SAP XI Components

The *Integration Server* is the central component of SAP NetWeaver XI; it receives messages at runtime and, based on the configuration, sends them to one or more receiver systems. A multitude of *adapters* provide connection options for all current formats and protocols.

The remainder of this lesson focuses on these XI components and their tasks within SAP NetWeaver XI.

The System Landscape in SAP NetWeaver XI

To enable data exchange between systems, all the information about the type of systems and the applications they contain must be available in a central location that the Integration Server can access when required. For XI, this location is the *System Landscape Directory (SLD)*. The SLD distinguishes between two kinds of systems: “technical systems” contain the information about the type of system (SAP, Java, or third-party) and how it can be accessed; “business systems” contain the information about the role of the system in the system landscape, and which of the software versions provided by the technical system are actually used. In the SLD, each technical system is assigned one or more business systems. In the SAP world, a technical system corresponds to an actual SAP installation, whereas a business system corresponds to a client of an SAP installation.Consequently, the SLD also contains all software products, software components, and software component versions, which are in turn provided by the technical systems and used by the business systems.

Integration Builder

The *Integration Builder* is comprised of two components. The *Integration Repository* provides all the objects that are required at design time to describe a system-independent connection. This description is used at configuration time in the *Integration Directory* to describe the actual communication between two existing systems. One reason for keeping the two phases separate is that if the same process is to take place between multiple systems (for example, between a test system and the equivalent productive systems), you need to describe it only once. It can then be accessed as many times as required at configuration time. Another reason is that SAP NetWeaver XI is currently included in the shipment of a large number of other SAP products (such as MDM and BW). By keeping the design phase and the assignment of the real systems separate, SAP is able to ship default scenarios for these products that customers can then apply to their system landscape.

The *Integration Builder* provides the graphical interface for accessing the *Integration Repository* and the *Integration Directory*.

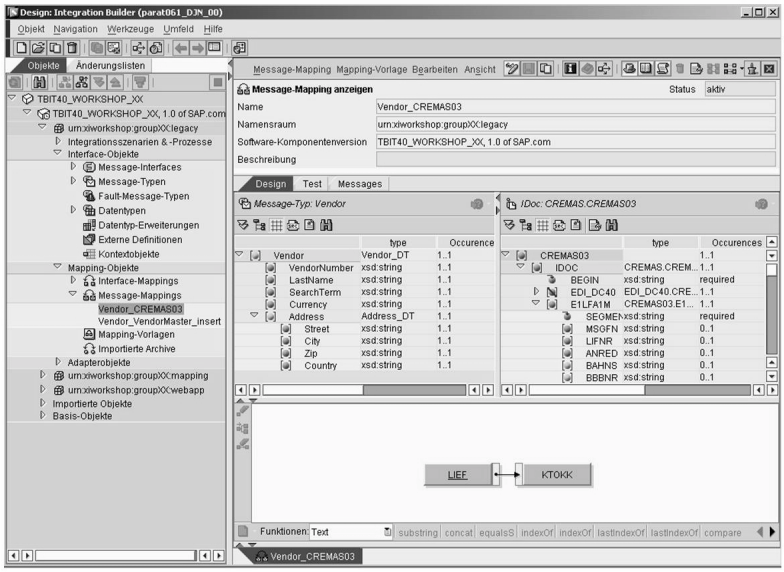


Figure 109: Integration Builder Design

You enter all the necessary interfaces (message interfaces) of the various software components and store mapping programs in the *Integration Repository*. In the Integration Repository, SAP also provides graphical representations of standard business processes, showing which interfaces can be used between which systems and using which mapping. These graphical illustrations can also be created by the customer.

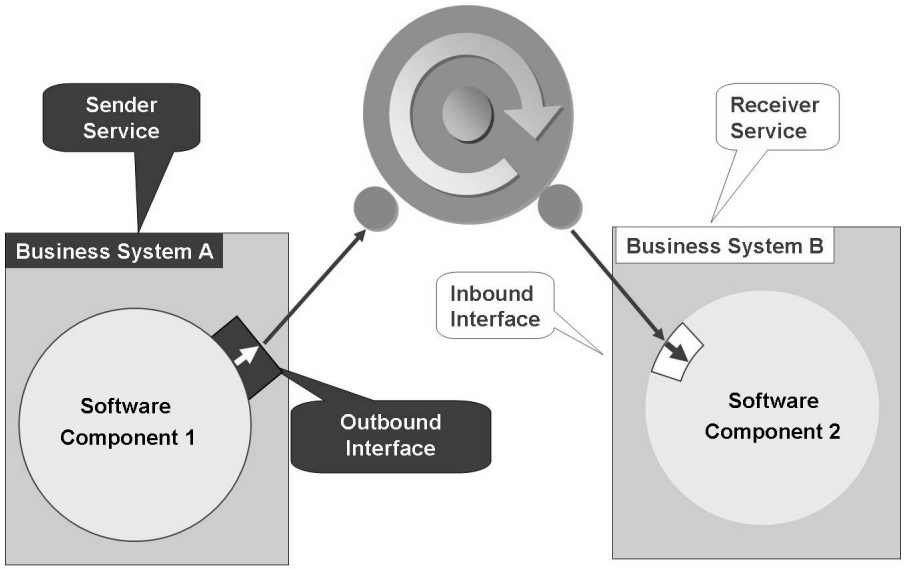


Figure 110: Integration Scenario in the Integration Directory

You use the *Integration Directory* to configure actual scenarios with real business partners. Using a wizard, you can access the scenarios from the Integration Repository so that you need to enter only the system names. You can also design your own scenarios; however, they must reference message interfaces and mappings that already exist in the Integration Repository. You must assign the following to an *outbound interface* (an incoming message): the receiver or receivers of an inbound message (and any conditions) in the receiver determination, and the relevant *inbound interface* (with a mapping, if applicable) in the interface determination. You also need to define how the systems involved communicate with SAP NetWeaver XI. This is achieved using sender communication channels for the sender of the outbound interface, and receiver communication channels at the receiver of the inbound interface. These communication channels define both the protocol and how a system is to be reached (for example, the RFC destination in RFC communication).

Processing Messages in SAP NetWeaver XI

The receipt of a message always triggers the SAP NetWeaver XI runtime. The information in the Integration Directory is used in the pipeline to determine which business system is the receiver of the message, which message interface is expected, and in which communication channel. Technical routing in the business system then determines which technical system is concerned. The system also checks whether a mapping is required and executes it, if applicable.

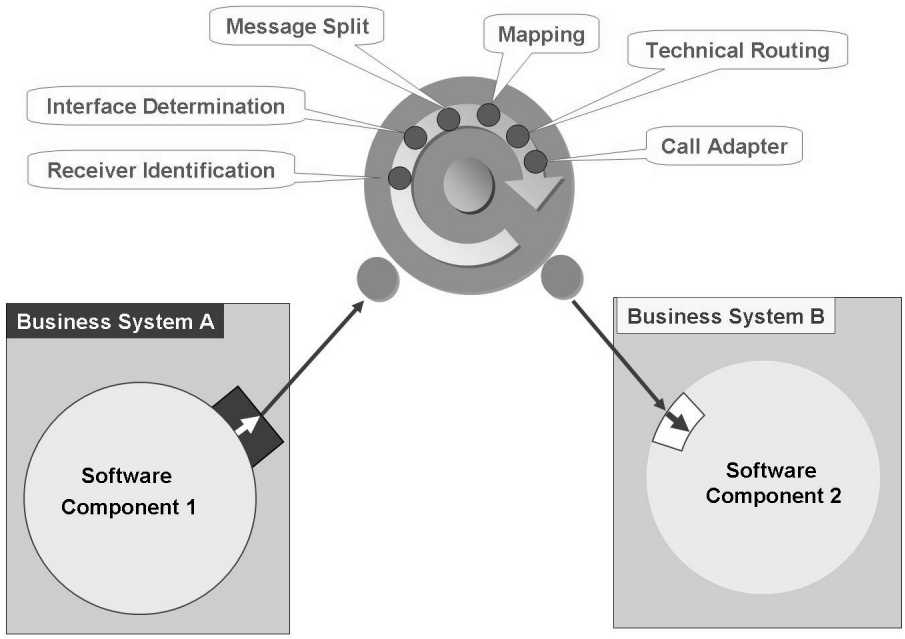


Figure 111: Runtime Behavior in SAP NetWeaver XI

Advantages of Using SAP NetWeaver XI over Peer-to-Peer Web Services

Routing

Dynamic receiver determination means that for an *SAP NetWeaver XI* inbound message, the receiver can be determined dynamically, that is, based on the content of the XML message.

Mapping

The ability to map different interfaces to each other is important, because different systems often need to be connected.

Central information

The System Landscape Directory (SLD) is an SAP XI component in which the relevant systems (with your software component details) can be stored. The Integration Repository contains interfaces, the mapping between these, and graphical representations of the business scenario. SAP supplies contents of the Integration Repository for SAP solutions as XI content, and the customer can use this information directly. Configuration takes place in the Integration Directory.

Inside-out development approach

In *SAP NetWeaver XI*, the development process is driven by the business process: first, the process is modeled with its interfaces (in the *Integration Repository*). From this central interface information, the interface objects are generated in the back end as proxy objects, and this is followed by implementation of the functionality in the back end. For server proxy objects, this means implementing the server method; for client proxy objects, it means calling the client proxy.

Support for asynchronous communication

As far as possible, loose coupling should be used, as this gives rise to fewer dependencies between the systems. Since the sender does not receive any direct business feedback from asynchronous communication, receipt of the message encompasses system stability through guaranteed receipt, including where network problems arise.

Use of *SAP NetWeaver XI* can also be regarded as an enhancement of Web service communication, because ultimately, *SAP NetWeaver XI* also allows every interface to be addressed as a Web service. The *Integration Server* takes charge of communication between the Web service client and the target interface. The WSDL file required for the Web service client can also be centrally generated.

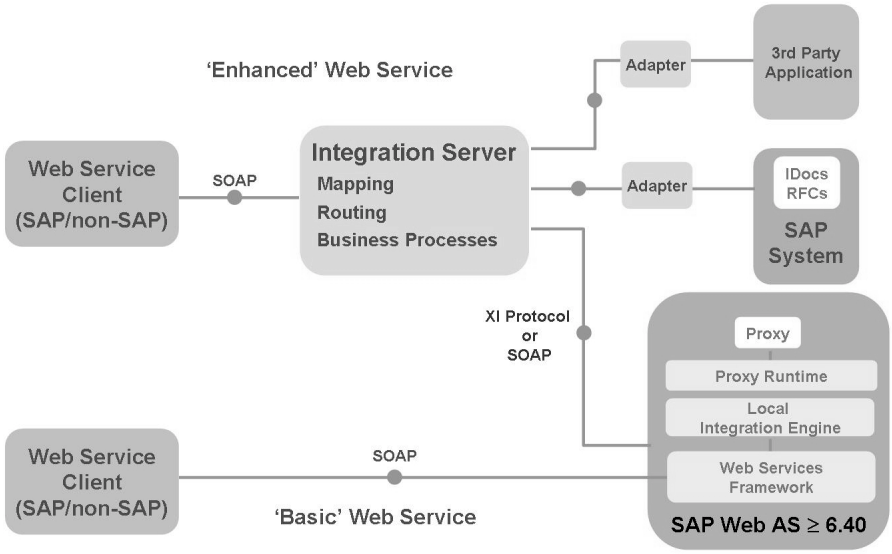


Figure 112: Web Service Enhancement with SAP XI

With the inside-out development approach, interfaces are first modeled in the *Integration Repository*, then the input functionality is generated in the target system (back-end system). We call this the proxy technique. A server proxy is an input

interface and can be compared to a Web service. In actual fact, these server proxies can also be simply defined as a Web service so that the input interface can be used directly in the event that neither mapping nor routing is required.

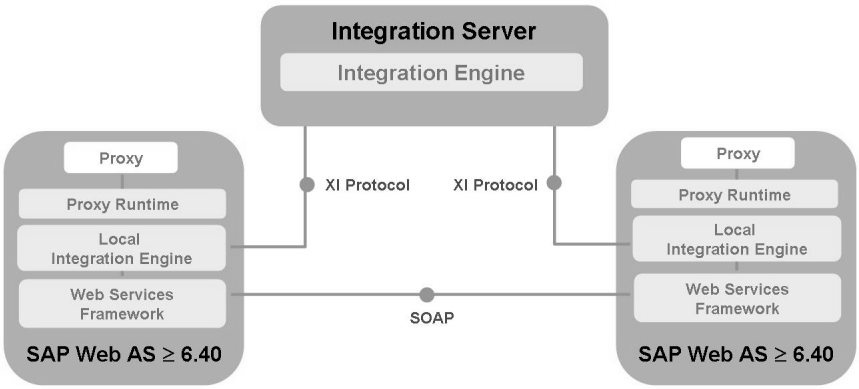


Figure 113: Web Services - P2P Optimization

Lesson Summary

You should now be able to:

- Understand the advantages of *SAP NetWeaver XI* compared with peer-to-peer connections

Related Information

- http://service.sap.com/xi

Internal Use SAP Partner Only

Lesson: eBusiness Standards

Lesson Overview

This lesson conveys a broad picture of the situations in which eBusiness standards are required and the ways in which these standards can be used. It shows how eBusiness standards can be implemented on the basis of Web services, as well the requirements that need to be met by business partners in order to use these eBusiness standards for communication between their applications. The lesson also provides a brief overview of the solutions offered by *SAP NetWeaver* technology that can be implemented using eBusiness standards.

Lesson Objectives

After completing this lesson, you will be able to:

- Understand why eBusiness standards are required
- Describe how eBusiness standards based on Web services can be used as well as the associated requirements
- Describe the tasks that need to be carried out to allow applications to communicate using eBusiness standards
- Describe which *SAP NetWeaver* technology solutions are available for using different eBusiness standards directly

Business Example

Every day, a large department store chain orders several hundred kilos of various types of frozen fish from a number of frozen food suppliers. As each company has an electronic application for orders and deliveries, the most effective scenario would be to process all the necessary steps associated with the purchase and the required business documents (such as the purchase order, order confirmation and so on) electronically. However, the companies involved do not have the same applications. They therefore interpret the individual steps and business documents very differently, depending on the application. However, to allow this purchase order handling process to take place electronically, the companies agree on an internationally accepted eBusiness standard and map their internal application data structures and processes to the eBusiness standard's predefined data structures and processes. Because this eBusiness standard is available to everyone, each of the companies understands exactly which information is exchanged from the standardized purchase order handling process, as well as the sequence in which this ordering process is carried out.

Introduction

A business process consists of a series of interlinked activities between two or more business partners who have reached an agreement to achieve a mutually required result. All of the necessary activities for a service are defined sequentially and can be handled between the business partners either manually or electronically.

A typical electronic business process, such as the ordering process illustrated below, consists of individual activities or process steps as well as the business documents to be exchanged (such as the purchase order, confirmation, and invoice), which are required to trigger status changes in the individual process steps, and thereby achieve the required result.

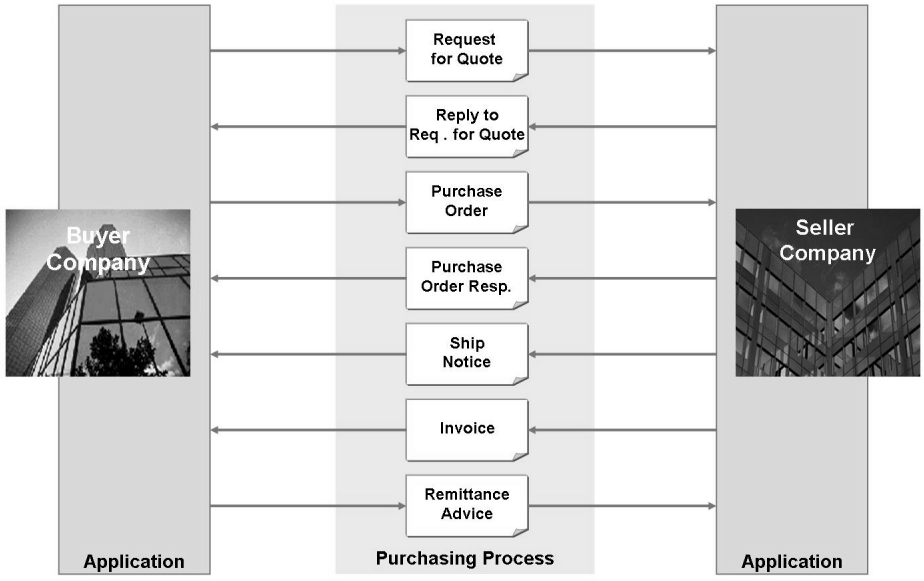


Figure 114: Electronic Business Process: Ordering Process

For these business processes and their process steps to be handled in a way that is fully electronic and interoperable, the business partners not only need to agree on the relevant activities, their sequence and decisions, and the necessary business information on a semantic level, they also need to standardize the technical services and protocols for the electronic exchange.

Breakdown into Levels

It is only by means of agreed, uniformly applied standards in the areas of technical communication and business-oriented semantics that business processes can be handled electronically between applications. The ISO Open-EDI reference model describes the following views for this:

- The **Business Operational View (BOV)** for the semantic and operational levels required for handling business processes, including a list of business-oriented vocabulary, and
- the **Functional Service View (FSV)** for the technical level required for communication between systems using protocols and services.

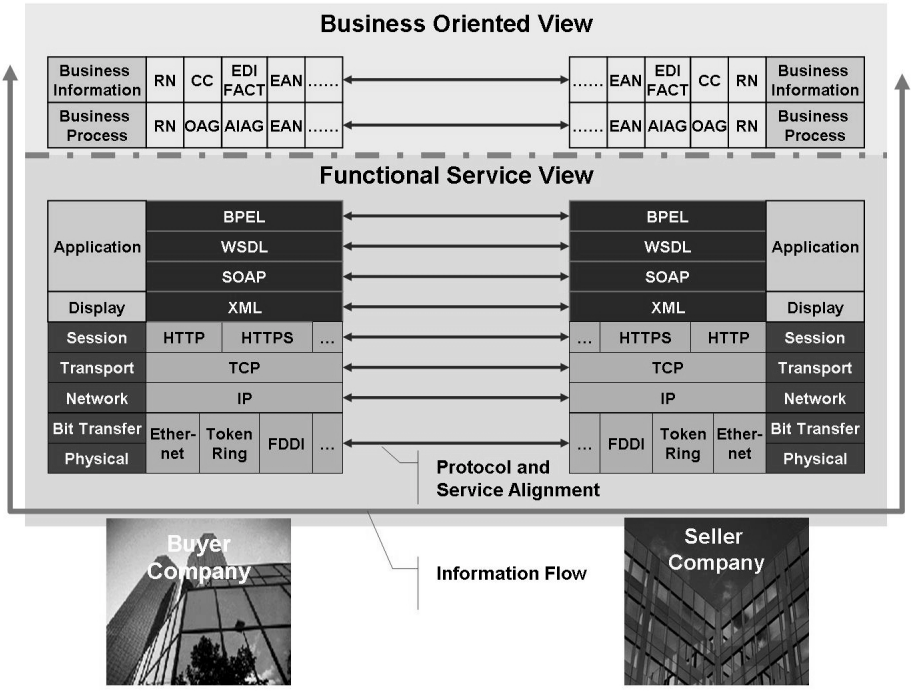


Figure 115: ISO Open-EDI Reference Model

The necessary protocols and services within the FSV that need to be applied and the ways in which these interact are best described by the ISO/OSI (Open Systems Interconnection) reference model. This is an open 7-layer model that is often used as the basis for a number of developer-independent network protocols. We will not go into the individual levels in detail. Suffice it to say that for communication between business partners, it is essential that the same network protocols and services are used on each level within the 7 layers. The above figure lists multiple protocols on a number of levels. In these cases, the partners need to agree on the same protocols and services.

It therefore stands to reason that, provided only one protocol or service exists on the individual levels, it is possible to have a business process carried out electronically. This is where international standards are brought to bear. International standards are required here because they are familiar to everyone and they can be implemented and applied uniformly. Communication across international networks is mostly carried out by TCP/IP today. The HTTP/HTTPS and SMTP protocols therefore play a leading role in the Internet today in terms of the management and communication of business processes between the logical connections. These standardized protocols are therefore essential for ensuring a successful B2B transaction.

Web Services for Technical Implementation

In terms of representing information on the application level and the underlying BOV, XML is indispensable. **Web services**, for example, which on the application level are finding increasing acceptance worldwide, are based on XML. B2B communication between applications can be based on the Web service standards **SOAP, WSDL (Web Service Description Language)**, and **BPEL (Business Process Execution Language)**. The following figure best illustrates how these three standards should be applied.

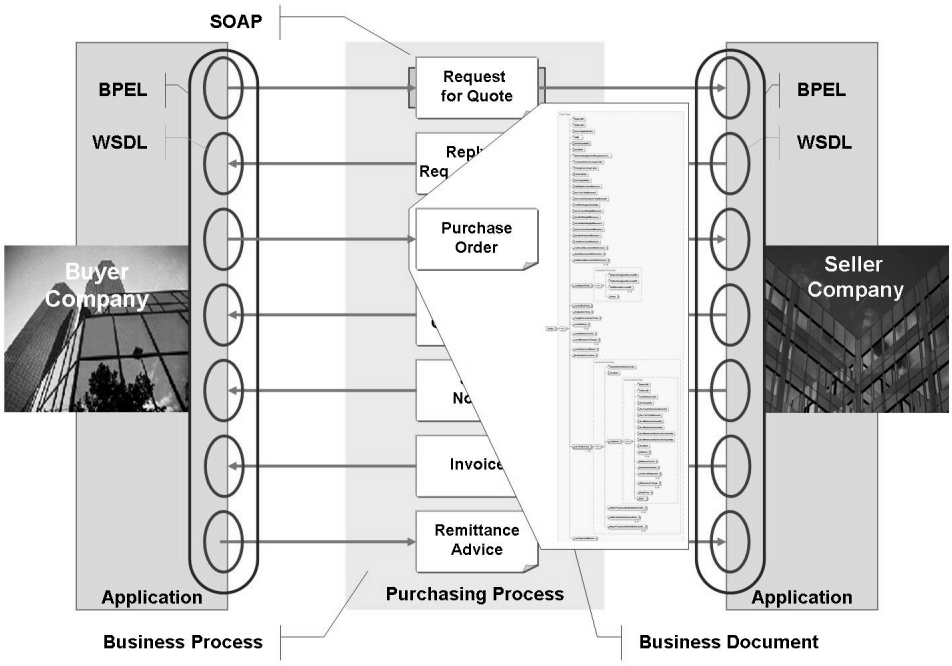


Figure 116: Assignment of Web Services to a Business Process

Internal Use SAP Partner Only

Explained in brief, SOAP is a message protocol used for the secure and standardized transmission of business documents between the actual business partners. WSDL is the bridge between the internal and external communication. The type and method of physical communication to be used as well as the way in which SOAP is to be used for the individual activities is determined here in detail. BPEL is used on the partner side to orchestrate the execution of activities according to a predefined business process sequence.

Semantic Representation of Business Processes and Documents

However, BPEL is not used to predefine any semantics or collaborative interaction between the business partners involved. All of the necessary requirements for a business process in the BOV are either Web services or are defined in other standards within the FSV. These requirements include agreements, instructions, logic, sequence, necessary process steps, content to be exchanged, common semantics, grammar, structuring, naming, and resulting dictionaries. Similarly, these requirements also need to based on an eBusiness standard so that they can be uniformly understood by the business partners involved. The requirements make it clear that this eBusiness standard is very similar to a natural language. This is understandable, because the business experts and users of the business processes in particular must be able to understand this eBusiness standard unequivocally. This also means that it is possible to interpret and process the eBusiness standard using the application program in a way that is uniform and without restriction.

However there is a major problem here: Because fundamentally different requirements exist in the various different sectors, industries, areas, and countries, new and different eBusiness standards are developed to accommodate these requirements. This is because the individual representatives believe that it would be quicker and easier to develop a separate eBusiness standard for their own requirements, rather than taking into account the reusability of existing eBusiness standards. XML syntax, in particular, further complicates this problem. This is because it is now very easy to use XML to quickly model and use a data structure that takes these individual requirements into account. Over the past number of years, this has lead to well over 1000 different industry-specific dialects and actual standards that do not match in terms of either structure or semantics. In contrast to the gradual process of standardization within the FSV and BSV, this process is dominated by uncontrolled growth.

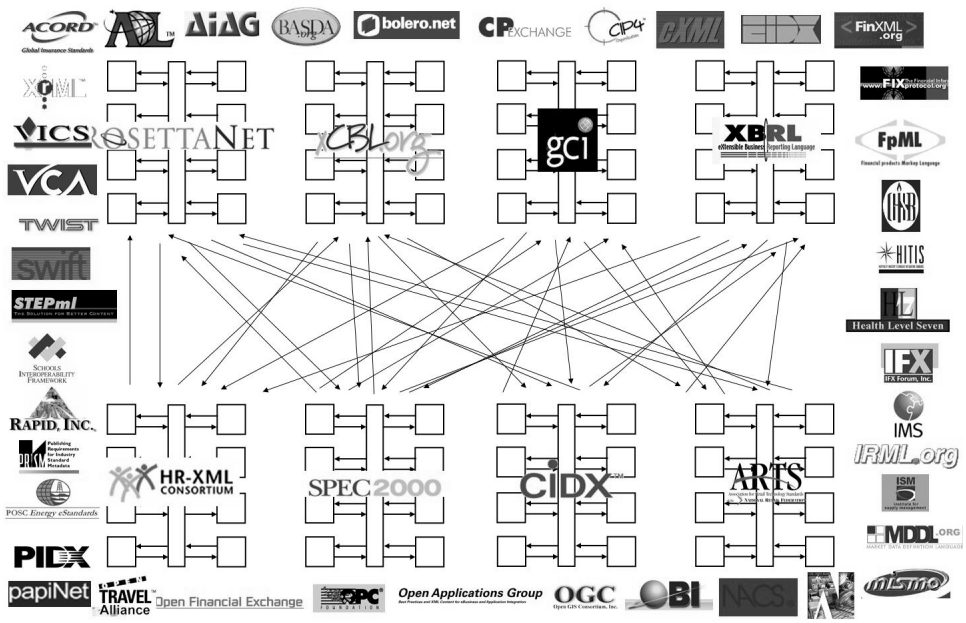


Figure 117: Uncontrolled Growth: eBusiness Standards

It is exactly this uncontrolled growth that makes implementation very problematic and time-consuming, even when XML is used as the standard syntax. If you take a detailed look at these substantial differences, you will immediately understand why a considerable volume of further work is required. The data structures of two different standards (OAG and xCBL) should in fact represent the same information: the address on a business document. However, because of the numerous differences in the structure, names, and semantic meaning, these data structures cannot be applied directly. Mapping is inevitably required. This is because instead of using the eBusiness standard, each application developer uses their own data structures, such as IDoc or BAPIs, which represent the same semantic information in different ways.

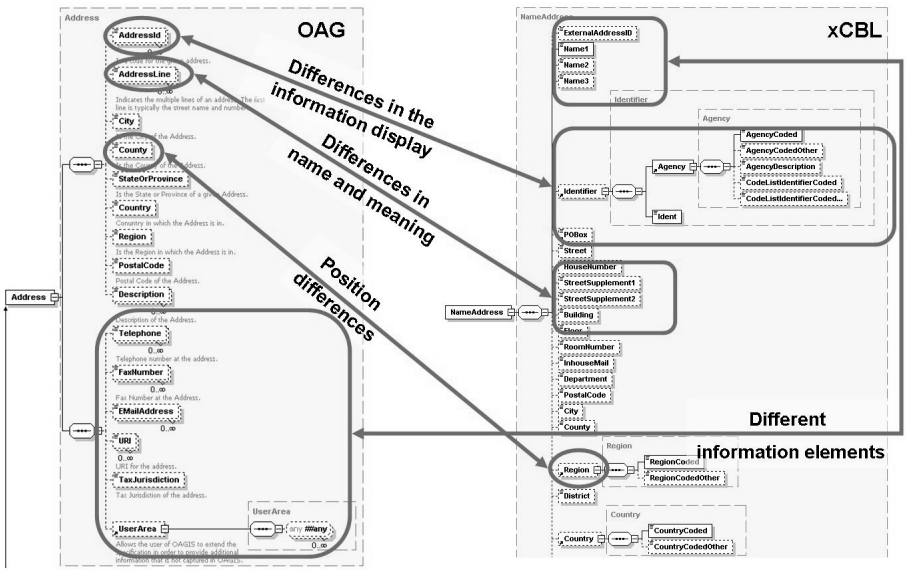


Figure 118: Data Structure Mapping

As illustrated in the figure below, a mapping must be created for each business document to be exchanged within a business process for all business partners involved. Either one outbound mapping is implemented for converting the internal data structure to a standardized or external data structure, or a separate mapping is used for the opposite direction. Because the individual semantic information and corresponding integrity is managed differently depending on whether the direction is inbound or outbound, it is rarely possible to create a mapping that takes into account both directions at the same time. Nor is there any guarantee that the same business process mappings can be used with other business partners. This is because these business partners either use another eBusiness standard or their requirements cannot be represented in the same way. Most eBusiness standards therefore often allow very considerable scope for interpretation.

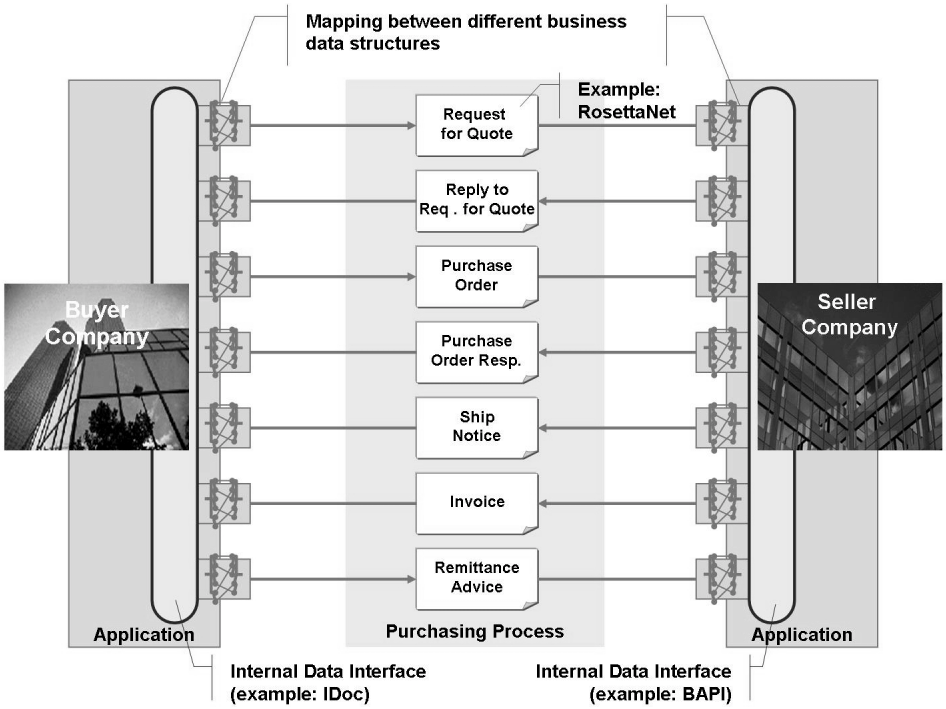


Figure 119: Mapping Between Different Business Data Structures

In the worst case scenario in our example, 14 different mappings would need to be created. If approximately 10 man days are required for a mapping, it quickly becomes apparent why a complete B2B transaction using one or more eBusiness standards is today only implemented for fairly long-term business relations or larger companies. Note that we are only discussing the mapping of data structures here. The internal descriptions of business process activities often differ in exactly the same way, such as the default data for an eBusiness standard, for example.

Unequivocal Semantic Understanding

To find a long-term solution to this very costly problem, we need to go back to the roots of the issue. The first questions to be asked are: Who needs to use these data structures in the first instance? Who models these data structures? Man or machine?

Man of course, and this is often forgotten when modeling data structures. When modeling data, the first thing to be ensured is that the machine understands the data as quickly as possible. Oddities always occur, as illustrated in the figure below. The data contains non-unique abbreviations with information arranged randomly and no standardized definition of the semantics. We always think that for performance reasons everything must be as short as possible, and that it is much more important to have the requirements “fed” into the system directly. It is definitely much, much

quicker to disregard the numerous conventions and standards that exist. Also, the notion still prevails that if something is represented in XML, anyone can interpret it, because this data can be read by man and machine. But how exactly is someone supposed to understand what <NM>Beaujolais Primeur</NM> or <Q>12</Q> means in an XML instance? How can this person's machine understand this information when this person doesn't even know how to interpret it?

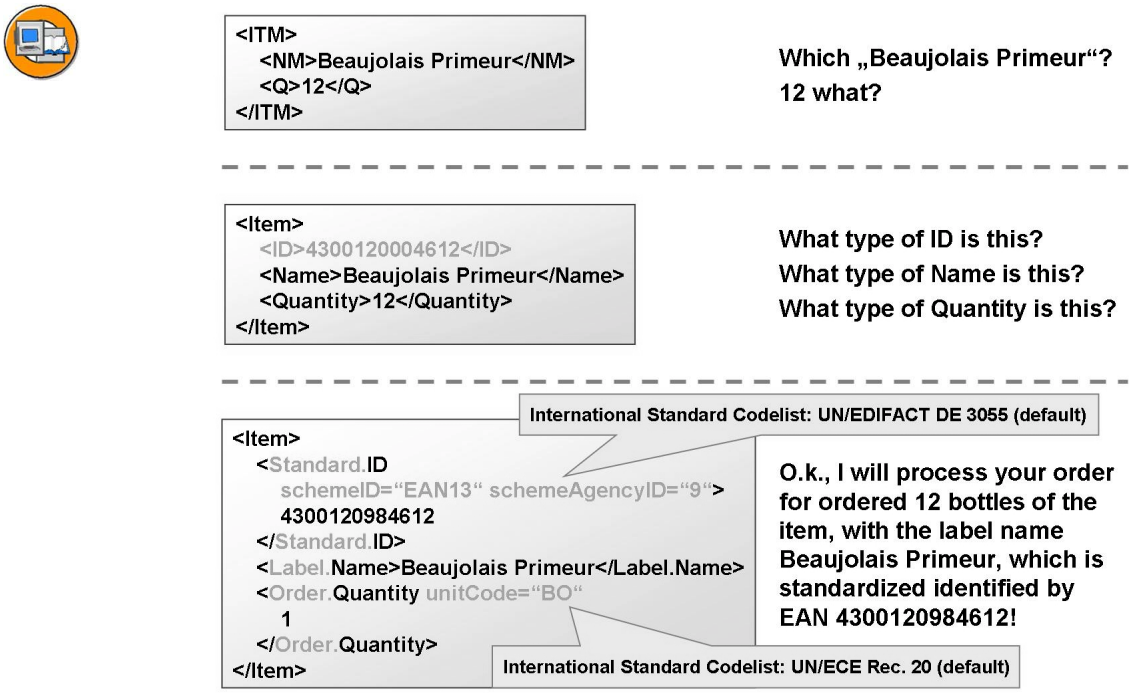


Figure 120: Unique Modeling of Data Structures

This precise problem leads to a very costly but equally thriving business. The integration of different data structures using either mappings or the direct implementation of data structures. However this business is not lucrative for everything; if this were the case, not everyone would be able to afford it, especially small and medium-sized businesses. This sector now represents the largest group with non-existent or very unsatisfactory levels of participation in eBusiness.

A common, one-to-one understanding therefore needs to be created for man and machine alike. We can now exchange international know-how or carry out international trading because we have agreed on an international business language for this purpose with its necessary grammar and terminology, namely English. English dialog can be carried out by telephone, paper, conversation, e-mail and so on, in other words, it is not bound by a technical syntax. We can only understand each other if we adhere to grammar and terminology rules. The same principle needs to be

taken into account when modeling business information. This one-to-one semantic understanding must be expressed in such a way that allows all types of requirements to be mapped and interpreted efficiently by both man and machine. As illustrated in the middle section of the figure above, the communication of random sentences or names is not sufficient because machines are more restricted in how they work. We can perhaps guess what the other person requires, but machines cannot do this. This is why a method is needed that can be used to create reusable information modules on a syntax-free level that can be interpreted unequivocally using a grammar system and underlying terminology. Since it involves machines, it must be more restrictive than the grammar system of a natural language. However, it must be able to cover the different semantic requirement options of the business information.

A method such as UN/CEFACT CCTS (Core Components Technical Specification) addresses the aspects outlined above. It basically takes into account the aspects explained in further detail below:

- The formation of unique names according to semantic requirements
- The standardized and logical structuring of complex business information according to the OO approach in order to enable standardized implementation in machines
- The standardized representation of values such as amounts, dates, measurements, code lists and so on, so that key data can be understood and processed uniformly
- The context-dependent categorization of business information so that only the relevant aspects of a specific requirement are taken into account

If all of these aspects are taken into account, the order information for the “Beaujolais Primeur” can be submitted in such a way that the employee does not need to make further inquiries and the application can immediately process the order for 12 bottles because it can locate this under the EAN 13 identifier very quickly.

This procedure is already implemented in SAP NetWeaver technology to ensure that all new SAP applications have the same unequivocal understanding of the business information.

SAP NetWeaver Technology for eBusiness Standards

Because SAP provides solutions based on NetWeaver technology for 26 different sectors and industries, *SAP NetWeaver Exchange Infrastructure (SAP NetWeaver XI)* offers a number of integration options. It is virtually impossible to integrate all eBusiness standards in *SAP NetWeaver XI* directly. The very substantial number of business documents and their extensive data structures would increase the work involved immeasurably. It is also not clear which eBusiness standards are relevant, or which information in an eBusiness standard is actually required. So there can never be any guarantee that all requirements will be met immediately. However, in order to

ensure that the necessary electronic business processes of an eBusiness standard can be applied as quickly as possible and to ensure a flexible response to the corresponding business requirements, *SAP NetWeaver XI* offers the following alternatives:

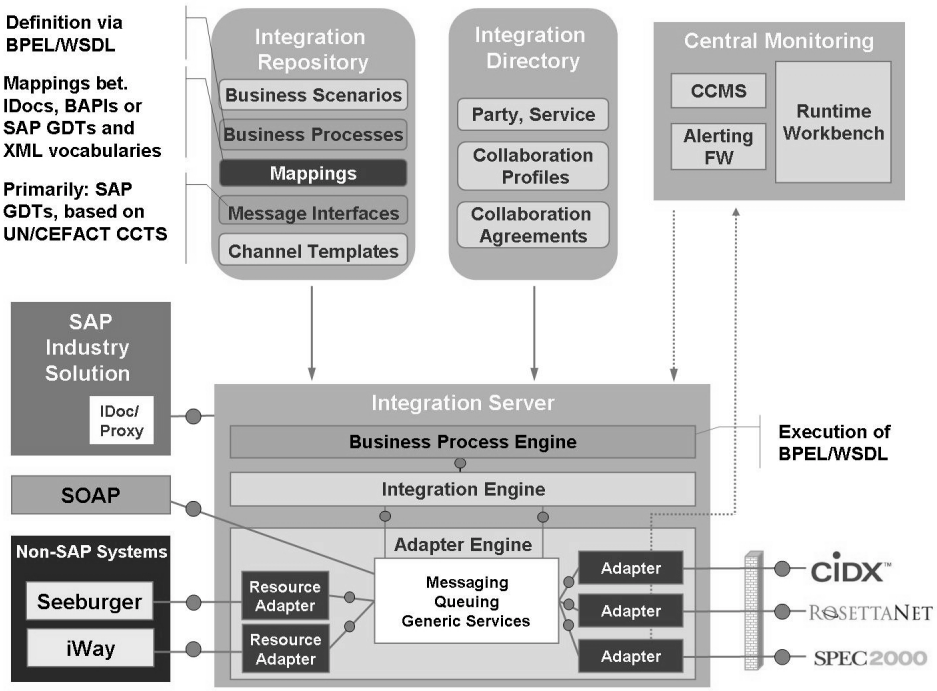


Figure 121: SAP NetWeaver Exchange Infrastructure

Primary representation and implicit support of business processes and business documents based on internationally accepted, multilingual, and generally applicable standards that satisfy all the requirements of SAP applications as well as the relevant sectors and industries. Support is therefore provided for the technical description and implementation of business processes primarily by means of Web services such as BPEL, WSDL, and SOAP, and for the basic representation of the business documents of all future applications based on UN/CEFACT CCTS (Core Component Technical Specification) and UN/CEFACT XML Naming and Design Rules for CCTS, which are known as SAP GDTs (Global Data Types). We will discuss SAP GDTs in more detail later. A primary representation based on internationally successful standards ensures that, in the long term, integration workloads will be reduced and usage will be quicker and more flexible because many implementers and users are supporting this standard.

Mapping component in the *SAP NetWeaver XI Integration Repository*, to allow the bidirectional mapping of IDocs, BAPIs, and SAP-GDTs for example, in all possible eBusiness standards based on XML.

Adapter for specific eBusiness standards, which is directly available to users via the Adapter Framework and contains the following functionality for the corresponding standards:

- Complete adaptation to the corresponding business applications, such as *mySAP CRM* or *mySAP SCM* for example, to enable B2B support within the SAP business packages with minimal effort.
- Mapping of the relevant business documents for the business processes provided in the adapter.
- Integration of the business processes predefined by the eBusiness standard. The bilateral agreements between the business partners involved in a business process can also be set up.

The following table lists the eBusiness standards that are currently supported by adapters or that will be supported in the near future:

Industry:	Vertical standard:	Basic business processes:
Chemicals	CIDX	Order-to-Cash, Supply/Demand Mgt., Logistics
Cons Prods	EANUCC	Order-to-Cash, RFID Enabled Inventory
High Tech	RosettaNet	Order-to-Cash, Data Harmonization
Oil & Gas	PIDX	Field eProcurement, Order Reconciliation
A&D	Spec2000	Order-to-Cash, Supply Chain Integration
Agriculture	RAPID	Supply Chain Integration, eProcurement
Automotive	StarStandard	Design Collaboration, Product Registration
	AIAG	Order-to-Cash, Supplier Integration
	Odette	Order-to-Cash, Supplier Integration
Mill Products	PapiNet	eProcurement, Supplier Integration
(Metals)	Eurofer	Supplier Integration, Distributor Integration

Adapters for non-SAP systems are required to enable the processing of eBusiness standards that are not directly supported by *SAP NetWeaver XI*. A substantial number of large companies still use classic eBusiness standards that are not based on XML. Current standards include: UN/EDIFACT, ODETTE, ANSI X.12, EAN.UCC, VDA, and Tradacomms. These eBusiness standards are supported or mapped using non-SAP systems such as Seeburger, WebMethods, or iWay. They include predefined mappings between IDocs, BAPIs, and the corresponding eBusiness standard for a number of

business documents. Although they have a separate *runtime*, they support all of the functionality provided in the XI adapter such as business process management, central configuration, and monitoring. This ensures that the user has access to central operation and management using *SAP NetWeaver XI*.

SAP GDTs (Global Data Types)

Almost unwittingly, the publication of *XML Recommendation 1.0* by the World Wide Web Consortium (W3C) provided the syntactic basis for worldwide data communication. XML offers a considerable number of advantages here that have been recognized by a large number of industry initiatives, which, as mentioned previously, have developed their own eBusiness standards. Despite the advantages of this eBusiness standard (seamless transfer with an optimally priced technical configuration, for example), it was not possible to close the *electronic loop*, that is, the continuous electronic transfer of business data between all partners involved. One of the most time-intensive problems here is the semantic and structural mapping between the different eBusiness standards and the internal data structures of the individual applications. The greatest potential for interoperability is still not being utilized.

As mentioned previously, the data structures of eBusiness standards can be compared to a natural language. Only when the grammar and terminology can be understood globally and used flexibly can unambiguous interoperability become a possibility. UN/CEFACT recognized this problem and developed, as mentioned earlier, a syntax-free method, the *UN/CEFACT Core Components Technical Specification* (CCTS), for the creation of business documents to enable the semantics and structure of these documents to be uniformly and unambiguously understood by all sectors, areas, industries, and countries. As already specified in the previous section, the expression and description of standardized semantics has nothing at all to do with syntax. Natural languages can also be spoken or written down, for example – the semantics remains the same.

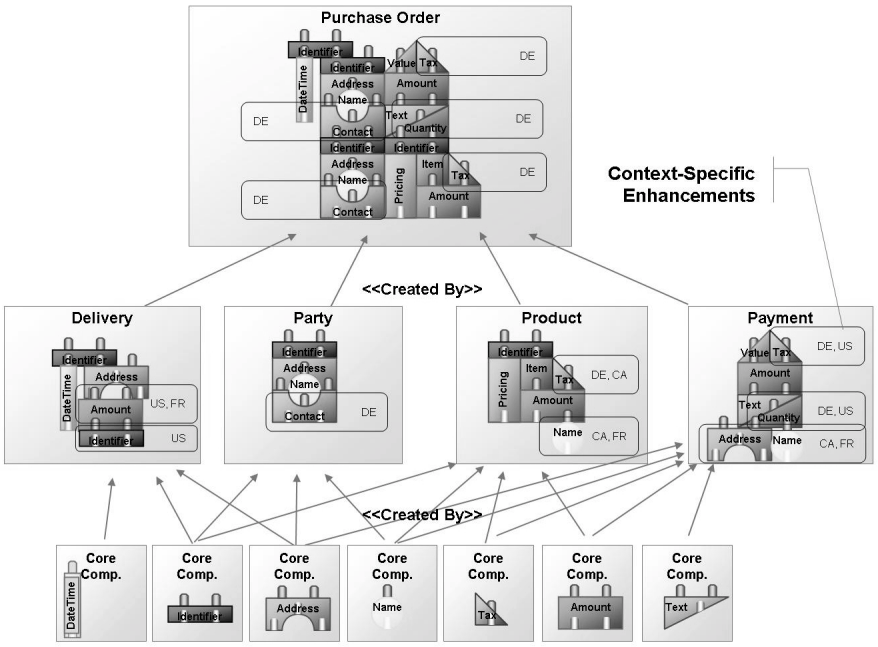


Figure 122: Building Block Principle

The essence of this approach is the building block principle described in the existing body of rules and specifications. The assumption here is that all types of business documents can be formed using generic components, in much the same way as LEGO bricks. These basic components should not contain any business semantics for one or more specific contexts (such as countries, industries, business process types, and products, for example); it must be possible to combine and interchange these components at will. They can represent a postal address, a currency amount, or a specified quantity, for example. If one or more of these components requires additional information for one or more contexts, this information can be easily integrated as a context-specific extension. What is important here is that both the base components and the context-specific extensions use the same methods for creating names and structures in order to ensure that the semantics are represented uniformly. These follow standardized conventions, such as ISO 1119, which determine the uniform structure of names in the same way as a grammar system.

For example, *Product. Tax. Amount* is a generic component that is always used whenever product taxes or duties are involved. If the tax is used for adding value, the *Value Added* qualifier is added to the generic name: *Product. Value Added_ Tax. Amount*. If the tax is also a country-specific sales tax, a further context value is added to this component for the country ID – for example *Product*. Value Added_ Tax. Amount (CountryCode="CA"). This means that this component applies to

Canada only. Additional contexts, such as *Product. Value Added_ Tax. Amount (CountryCode="CA, DE, US, FR")* can be added without difficulty. This makes it clear that this component applies to other countries.

If all component developers comply with these methods, the harmonization principle described in the CCTS will allow identical parts or context-specific extensions to be established and consolidated easily. Numerous initiatives such as OAG, AIAG, SWIFT, and EAN.UCC have recognized the basic advantages and are now using this method to develop their context-specific components, which can then be centrally harmonized for UN/CEFACT very easily.

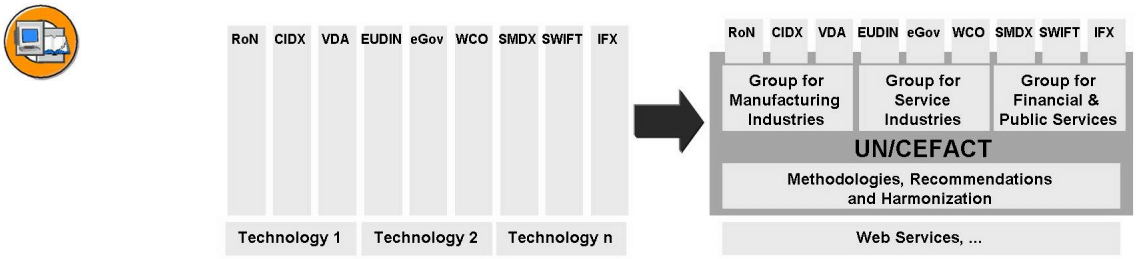


Figure 123: Harmonization of Components for UN/CEFACT

We will soon have a very comprehensive library that will provide individual, customized business document components for all requirements. When using these components, only the context-specific extensions should ever be reimplemented, provided they are not yet known in the application. Ultimately, this considerably minimizes the effort required for implementation or any necessary mapping.

Because all components are syntax-free in accordance with UN/CEFACT, they can be applied uniformly in completely different applications that require different syntax. In any event, XML will play the major role in the actual electronic data exchange between business partners. This is why UN/CEFACT developed the *XML Naming and Design Rules for CCTS*, which are used for transmitting the individual components in the XML syntax.

Because SAP provides a solution for almost every industry and sector, and because the effort required for the implementation of proprietary and industry-specific standards is increasing immeasurably, SAP is developing the primary interfaces for business documents for all applications for *NetWeaver* - SAP GDTs (Global Data Types), which are based on exactly these *UN/CEFACT CCTS* and *XML Naming and Design Rules for CCTS* standards.

Lesson Summary

You should now be able to:

- Understand why eBusiness standards are required
- Describe how eBusiness standards based on Web services can be used as well as the associated requirements
- Describe the tasks that need to be carried out to allow applications to communicate using eBusiness standards
- Describe which *SAP NetWeaver* technology solutions are available for using different eBusiness standards directly

Internal Use SAP Partner Only

Unit Summary

You should now be able to:

- Understand the advantages of *SAP NetWeaver XI* compared with peer-to-peer connections
- Understand why eBusiness standards are required
- Describe how eBusiness standards based on Web services can be used as well as the associated requirements
- Describe the tasks that need to be carried out to allow applications to communicate using eBusiness standards
- Describe which *SAP NetWeaver* technology solutions are available for using different eBusiness standards directly

Course Summary

You should now be able to:

- Describe the Web service paradigm
- Create an ABAP Web service using the *Service Definition Wizard*
- Create and configure a *service interface* and *service variant*
- Publish ABAP Web services to a UDDI repository
- Generate an Web service client proxy for consuming a Web service

Internal Use SAP Partner Only

Internal Use SAP Partner Only

Appendix 1

Web Service Security

Internal Use SAP Partner Only

Overview

The measures described in this unit for securing Web services only apply to the functionality provided with the ABAP stack *(SAP Web AS ABAP)*. The more advanced security features of *SAP NetWeaver XI* and *Web AS Java* are not dealt with in this unit.

Most Internet users are familiar with the standard security mechanisms used to transfer sensitive data such as when banking online or ordering goods over the Internet. The padlock icon in the browser indicates to the end user that the data exchange with the server will be encrypted. This is the only information the user needs before he or she is now ready to pass on confidential data. Millions of Internet users place their trust in the technology behind this process. This is the same technology that is also used to encrypt Web service data. This unit therefore covers these encryption techniques in detail.

Business Scenario

A company wants to provide customers with the option to order its products via a portal. A partner company creates the marketplace for this. However, customers are given individual prices, depending on the order quantity and other criteria. The online order pricing request therefore needs to be transmitted to the SAP system, completed, and returned. The IT manager wants this link to be implemented on the basis of Web services. The IT department now needs to plan the technical development of this implementation process. No one in the company has had any previous experience working with Web services. The pricing process within a company is surely one of the most closely guarded secrets, and the measures used to secure this transaction are of great interest to all concerned. The IT department needs to familiarize itself with the options for securing Web services.

Security-Relevant Aspects of Exchanging Data

The work involved in implementing technical security measures for exchanging data is largely shaped by the process of securing external or internal communication. The protection measures are determined by the size of the company and the significance of the data, for example.

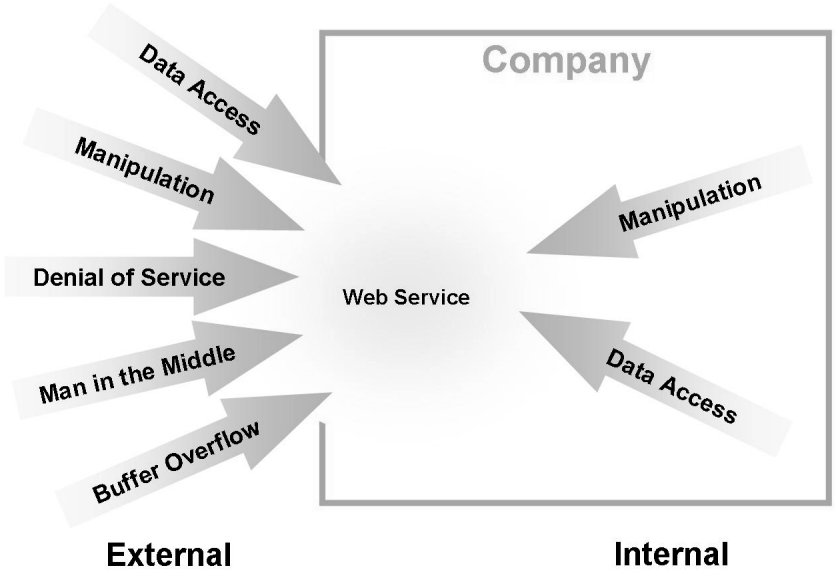


Figure 124: Data Exchange Security Risks

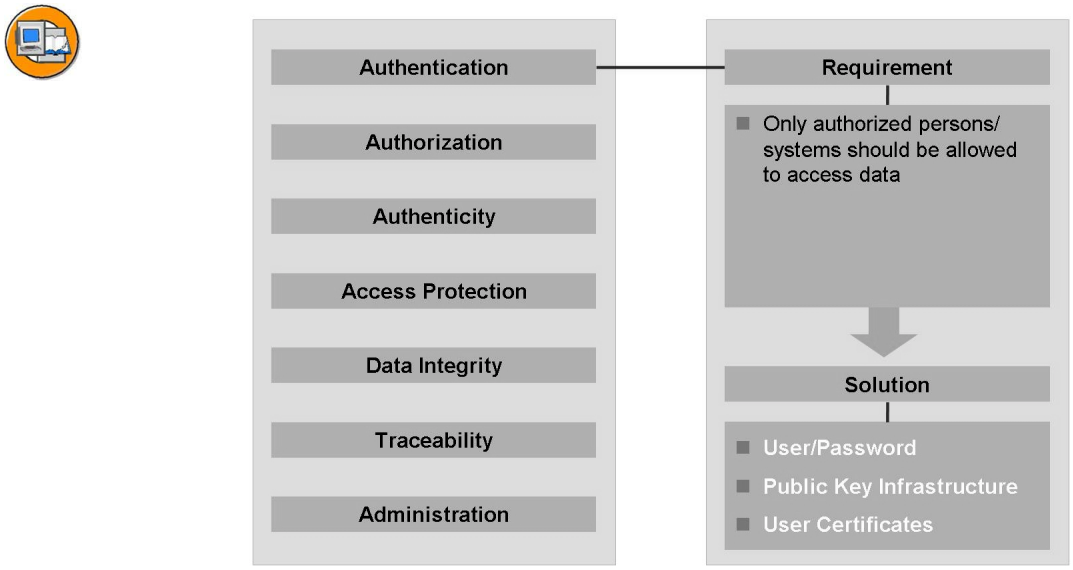


Figure 125: Security Requirement: Authentication

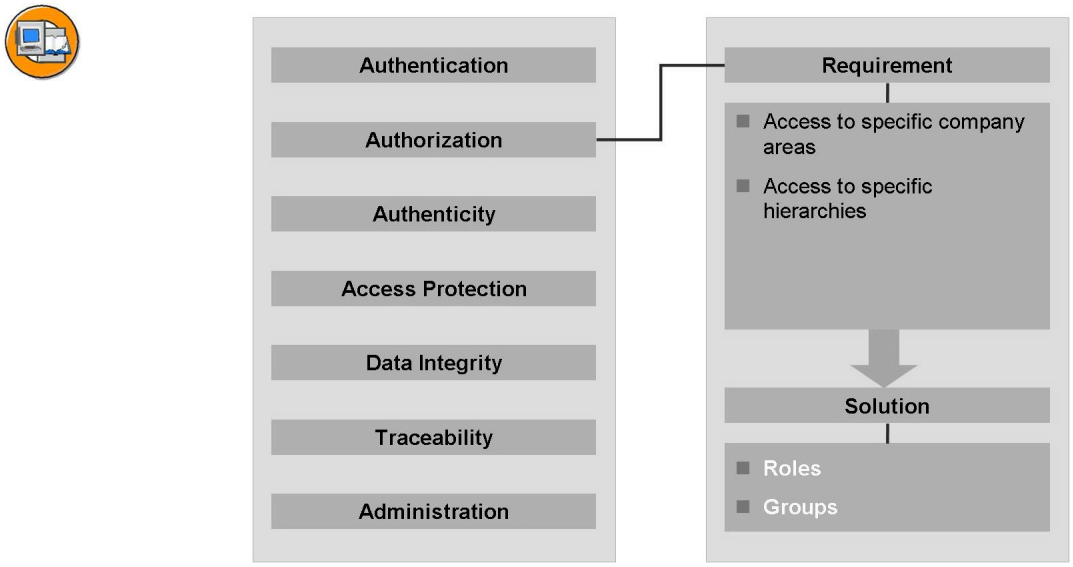


Figure 126: Security Requirement: Authorization

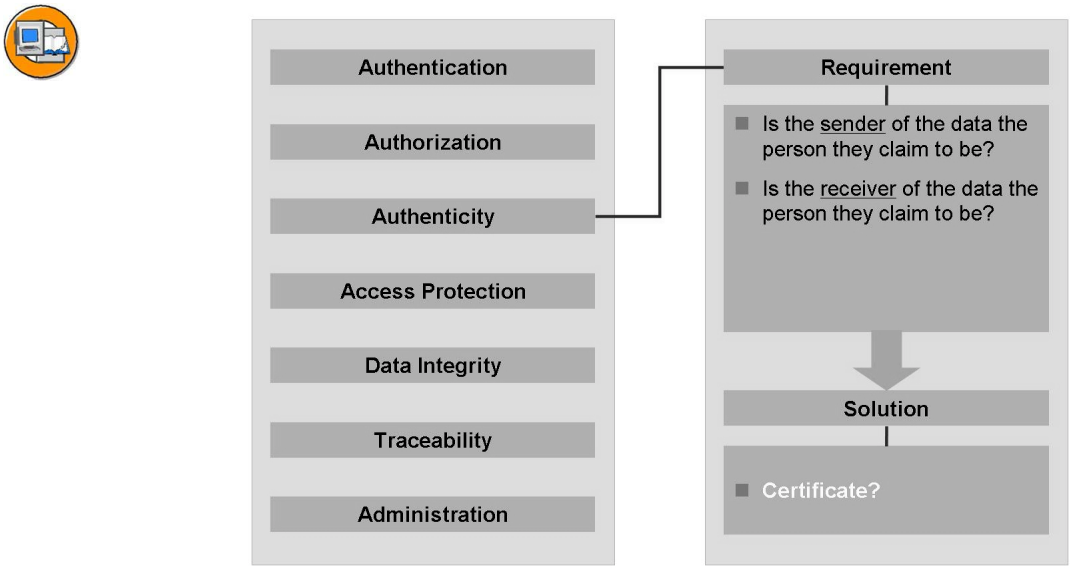


Figure 127: Security Requirement: Authenticity

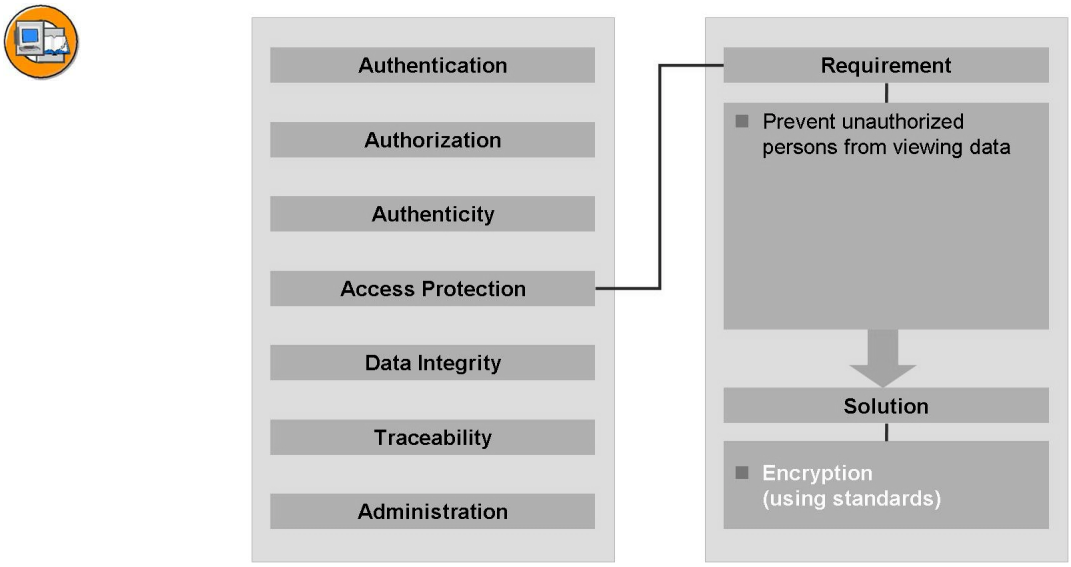


Figure 128: Security Requirement: Access Protection

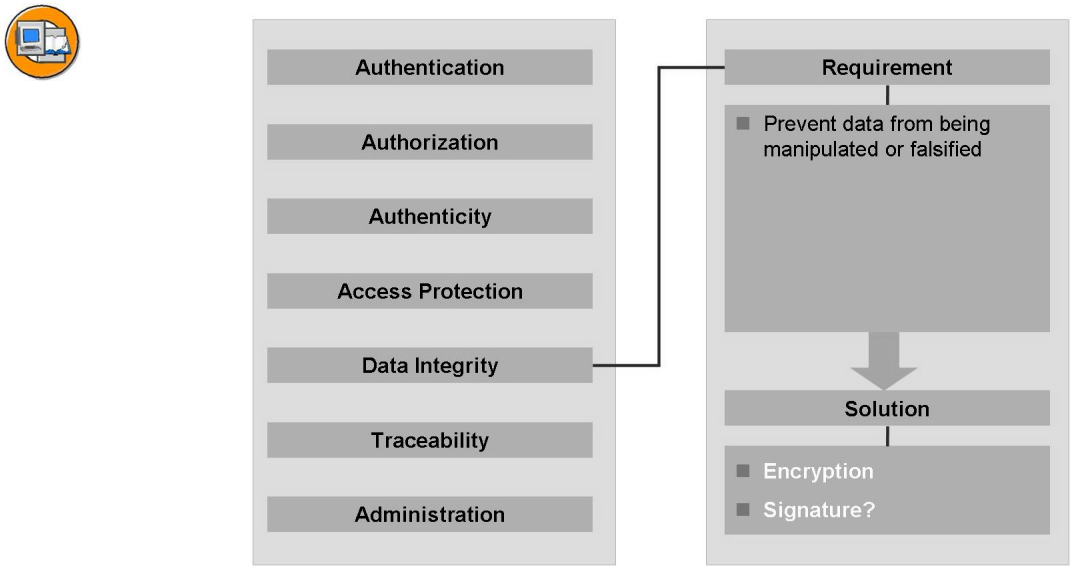


Figure 129: Security Requirement: Data Integrity

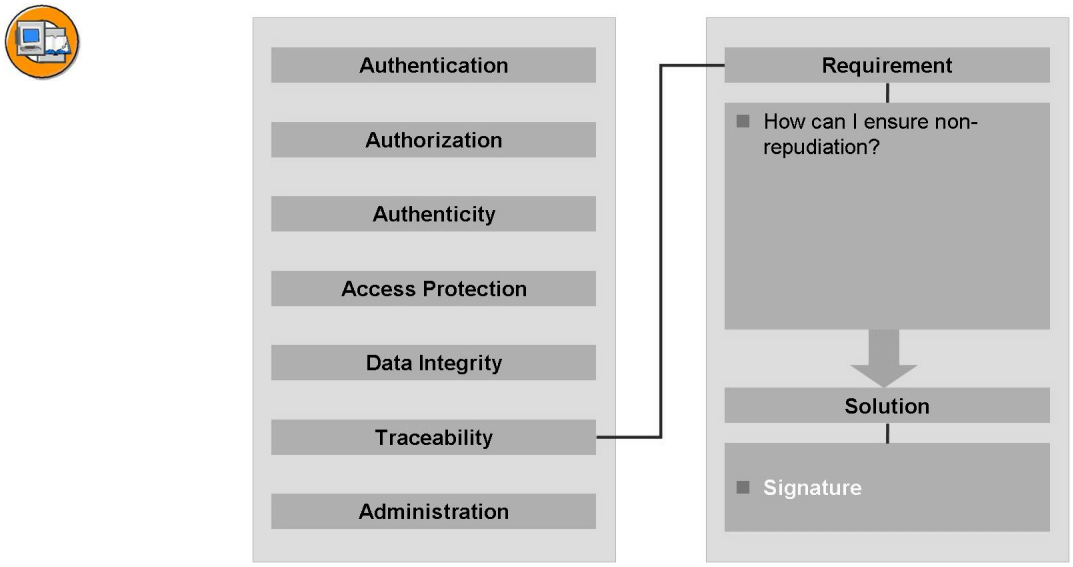


Figure 130: Security Requirement: Traceability

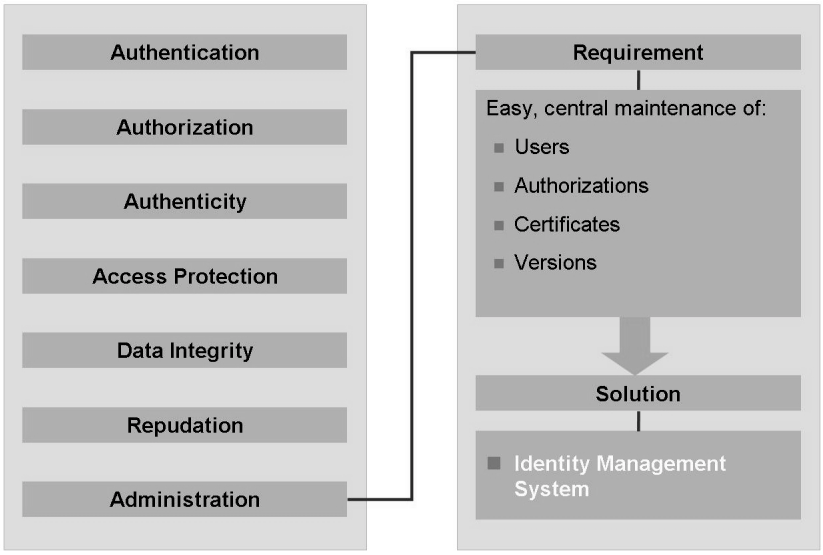


Figure 131: Security Requirement: Administration

WS Security

WS Security is a standard that can be used to secure SOAP messages. The SSL protocol is not used. Using WS Security, SOAP messages passed between the Web service provider and the Web service client are protected by means of digital XML signatures, XML encryption, time stamps, and security tokens.

At the time of writing, the standardization of WS Security was still in progress. For current information, please see SAP Note 716741.

Hint: WS Security can only be applied to SOAP messages. It is not supported by HTTP GET, HTTP Post, or SOAP with attachments. SAP NetWeaver AS 6.40 SP2 (ABAP) and later supports WS Security for Web services, but not for proxies.

Web Service Security doesn't just handle the security aspect of running Web services; it also handles publishing.

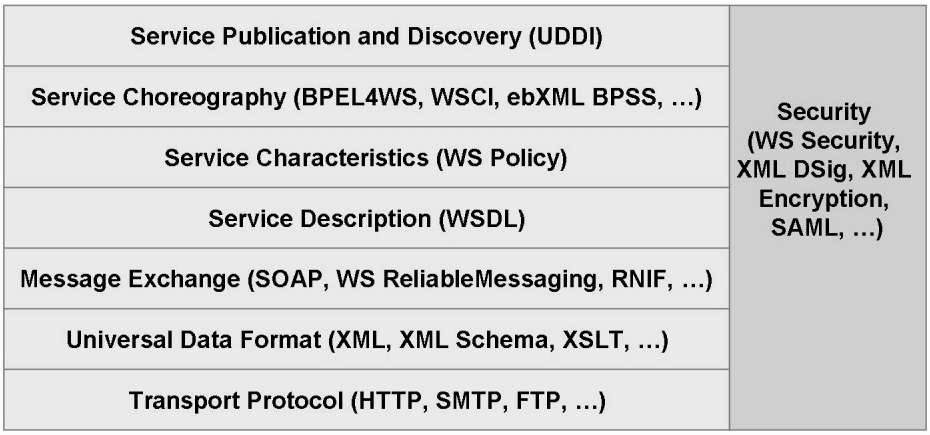


Figure 132: WS Security Environment

	Request	Solution
■ Confidentiality	■ Only authorized persons/systems should be allowed to access data	■ User/Password ■ Signature/Certificate ■ Encryption
■ Liability	■ How can I ensure non-repudiation?	■ Signature/Certificate
■ Authenticity/ Origin Check	■ Is the sender or receiver of the data the person they claim to be?	■ Signature/Certificate
■ Data Integrity	■ Prevent data from being manipulated or falsified	■ Signature ■ Encryption
■ Administration	■ Simple, central maintenance of: users, authorizations, certificates	■ Identity Management System (IMS)

Figure 133: Web Service Security Requirements

XML Signatures

Digital signatures are added to SOAP documents to ensure the integrity and authenticity of the message. If parts of the message are changed during transport, the signature becomes invalid and the message is rejected by the receiving page. Signatures can be added to the client request and the server

response. Signatures are always used in combination with a timestamp to prevent replays of the message (both the *SOAP:Envelope/SOAP:Body element* and the *SOAP:Envelope/SOAP:Header/wsse:Security/wsu:timestamp* are signed).

Security Tokens

Besides XML signatures, other credentials used to authenticate the Web service client may be included in the message. The SAP Web AS implementation of WS Security supports the Username security token and the X.509 security token. To prove possession of the X.509 certificates used in the X.509 security token, an XML signature using the corresponding private key is required.

XML Encryption

XML encryption is used to protect the transfer of message elements to certain recipients only. These recipients must have a valid key for decrypting the message. Encryption is used to protect elements that are sent as part of the SOAP message. XML encryption for SAP Web AS 6.40 is not currently supported.

WS ReliableMessaging

Reliable Messaging refers to the reliable delivery and correct sequencing of messages. These additional transaction-relevant Web service properties are also standardized.

Requirements	Recommended Technology
Data integrity, authentication, and immutability ■ Assurance that the data is authentic ■ Assurance that the sender is authentic	XML Signature
Confidentiality ■ Only forward messages to specific receivers	XML Encryption WS Security
Reliable Messaging ■ Reliable delivery of messages ■ Order of a message chain	WS ReliableMessaging, WS Reliability, ebXML Messaging

Figure 134: Web Service Requirements

Authorization

The execution authorization of Web service is regulated by standards such as Security Assertion Markup Language (SAML) and Extensible Access Control Markup Language (XACML). Started in 2001, SAML was developed by an OASIS consortium with the involvement of SAP as a standard XML language rule for exchanging security confirmations. It enables, for example, single sign-on, distributed transactions, and the linking of authorization services.

XACML is an XML schema that defines the structure of authorization agreements. This existing body of rules and regulations allows the management of access to resources.

Liberty Alliance and WS Identity are responsible for standardizing the establishment of identity and trust relationships.

The choreography of Web services is described using various standards such as:

- ebXML (electronic business XML)
- BPSS (Business Process Specification Schema)
- BPEL4WS (Business Process Execution Language for Web Services)
- WSCI (Web Service Choreography Interface)

There is currently no regulation in place on the agreement of message types and Web service protocols, or on the management of Web service life cycles. SAP is involved as an active partner with the United Nations in the development and implementation of these important standards.

Requirement	Recommended Technology
Authorization ■ Access control for the Web service	SAML, XACML
Identity and reliance	Liberty Alliance, WS Federation
Choreography ■ Description of message exchange	BPEL4WS, WSCI, BPML, ebXML BPSS
Requirements and capabilities ■ Description of Web service characteristics, orchestration, and transfer	
Negotiation and management ■ Agreement of message types and Web service protocols ■ Management of the Web service life cycle	?

Figure 135: The Missing Piece of the Puzzle

SAP provides various solutions to link the individual systems to a Web service infrastructure. Besides the ABAP SOAPAdapter, Web service solutions are incorporated into the SAP NetWeaver Java Stack and SAP NetWeaver XI Middleware. The following table shows the security-specific differences between the solutions.

	ABAP SOAP	NetWeaver SOAP
Point2Point (HTTPS)	✓	✓
Verification of the signature/SOAP user	using J2EE stack	✓
End2End encryption Payload	-	✓

Figure 136: Security Configuration of the SAP Web Service Infrastructure

Encryption

Although interoperability is not necessarily a security issue, it is still an important aspect of a service-oriented architecture (SOA), which cannot function without it. Most service specifications use a number of mechanisms for the same task. For example, the sender uses Triple DES, AES 128, AES 192, or AES256 algorithms for encryption; the recipient can only decrypt AES 128.

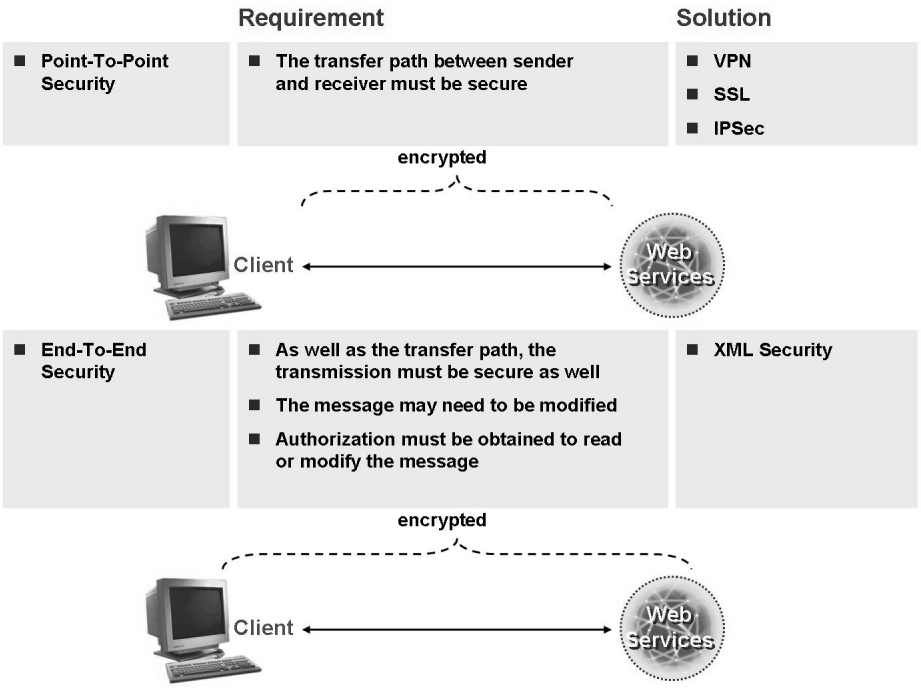


Figure 137: Network Topology Security Requirements

Point-to-point encryption

When transporting Web service data, it is expected that no one will be able to see or change the data. The data is encrypted and signed when it reaches the medium. Only the recipient can decrypt this data and guarantee the authenticity of the document by means of the signature. There are various Transport Layer Security (TLS) mechanisms available for this, such as Secure Socket Layer (SSL), Secure Shell (SSH), or IPsec.

Transport security: To activate SSL for a Web service, complete the following steps:

1. Activate SSL for *SAP Web AS* (see SAP online documentation: Using the Secure Sockets Layer Protocol with *SAP Web AS ABAP*).
2. Select *Integrity* in the Web service definition for the Web service.
3. In the *ICF* node (transaction SICF) for the Web service, check the *SSL* radio button under *Security Requirements*.

Authentication

Basic HTTP	In the *ICF* node, choose *Standard* under *Logon Procedure*
X.509 client certificate via SSL	In the ICF node, choose Oblig. via Client-Cert. (SSL) under Logon Procedure

End-to-end encryption

As a rule, Web services are not transmitted directly from an end point to the recipient, but rather through a number of intermediary instances. The message is therefore not protected against authorized access along the entire transport path. Encrypting the document itself and having it decrypted again by the recipient prevents the data from being seen by third parties. Encryption mechanisms in the Web service infrastructure therefore need to be used to ensure that documents no longer contain any plain text when being transferred to the transport medium.

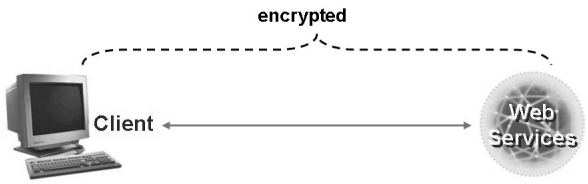


Figure 138: End-to-End Encryption

This type of encryption is partially covered by the ABAP Web Services Server. If the signature of the Web service document is required, the SAP Web AS J2EE Engine must perform this important step and set up the necessary steps for this in the system.

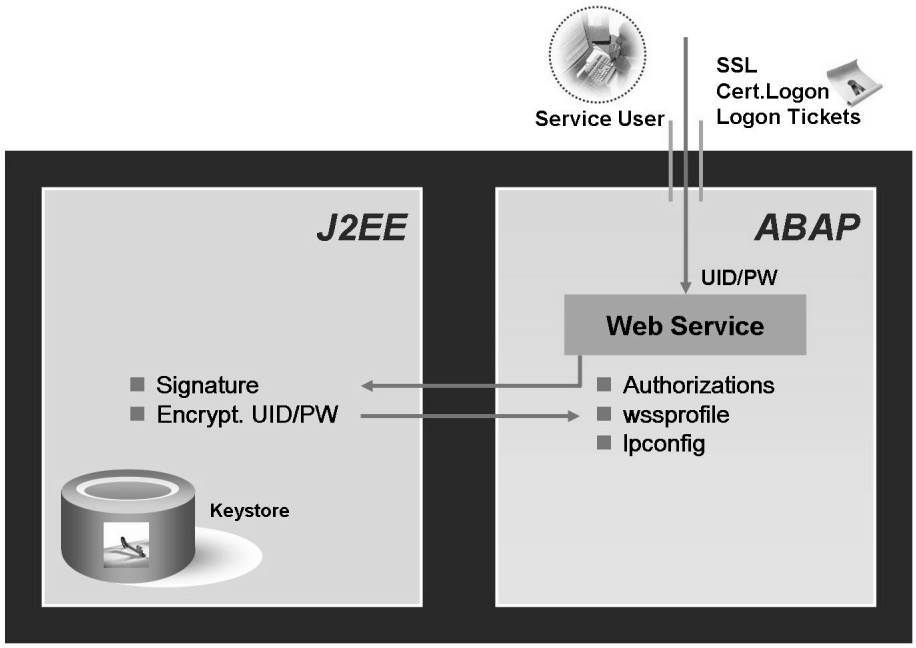


Figure 139: SAP Web AS ABAP and JAVA

Public key infrastructure (PKI)

PKI is a system that manages the trust relationships used for public key technology. The role of the PKI is to ensure that the public key certificate and certification authorities (CAs) can be validated and trusted. The collection of services and components involved in establishing and maintaining these trust relationships is known as the PKI.

Public key technology is a technology used to secure digital documents. Public key technology uses key pairs to provide its protection. Each participant receives an individual key pair consisting of a public key and a private key. These keys have the following characteristics: The keys are pairs; they belong together. You cannot obtain the private key from the public key. As the term indicates, the public key is available to the public. The owner of the key pair distributes the public key as required. The recipient of a signed document must, for example, know the signatory's public key in order to verify the digital signature. To send an encrypted document, the sender also needs to know the recipient's public key. The private key must not be disclosed. The owner of the keys uses the private key to generate his or her digital signature and to decrypt messages encrypted with his or her public key. The owner of the keys therefore needs to ensure that no unwanted parties can access the private key.

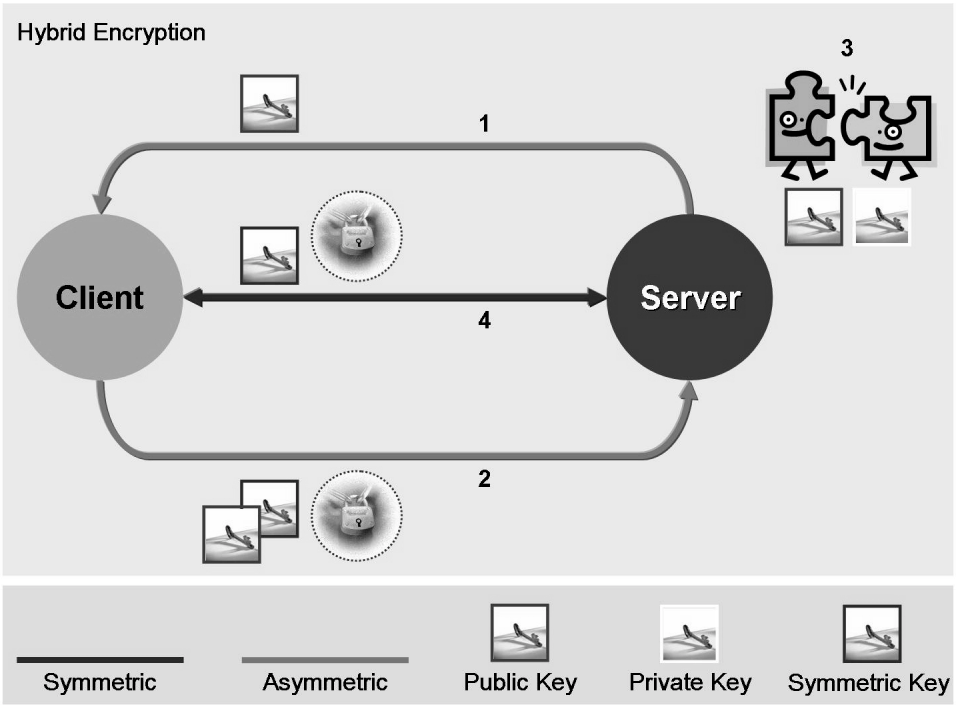


Figure 140: Hybrid Encryption

A **public key certificate** is a digital document that acts as the user's digital ID card. The public key certificate (also X.509 client certificate) is based on the X.509 format, an Internet standard developed by the International Telecommunication Union (ITU). Note: For more information on the ITU, refer to http://www.itu.int.

Public key certificates contain the public part of a user's public key information and are used for authentication purposes and for verifying digital signatures. A certification authority (CA) guarantees the identity of the certificate owner and approves the certificate and or certifies the user.

You use public key certificates to:

- Identify yourself to others using your own certificate
- Verify another user's digital signature using his or her certificate
- Encrypt a specific message for the certificate owner using a certificate

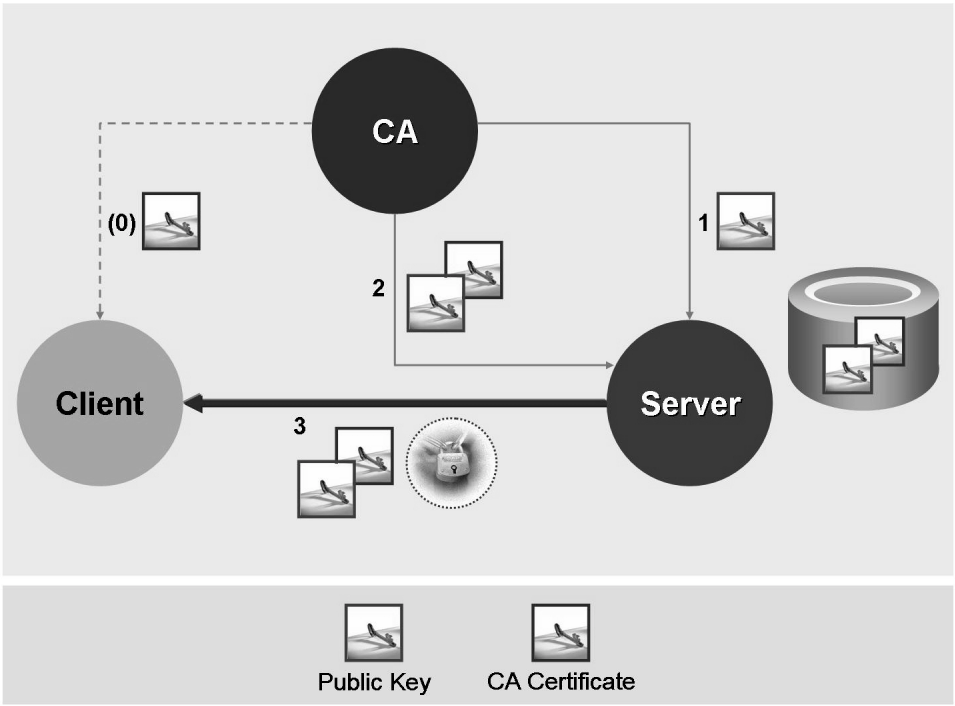


Figure 141: CA Trust Relationships

Digital Signatures

There are several reasons why you might want to verify a digital signature. For example:

- You have received a digitally signed document and you want to verify the identity of the sender.
- You want to verify the integrity of a signed document (when auditing archives, for example).

Before you can verify a digital signature, you need the following:

- The signed document that you want to verify
- The hash algorithm that the signatory used for his or her signature
- Access to the signatory's public key

Generally, you indicate that you want to *verify* a digital signature, and the system does the rest.

Hint: This step may also include part of a business workflow where the system requests that a digital signature be verified before proceeding.

The following figure shows what happens when you verify a digital signature:

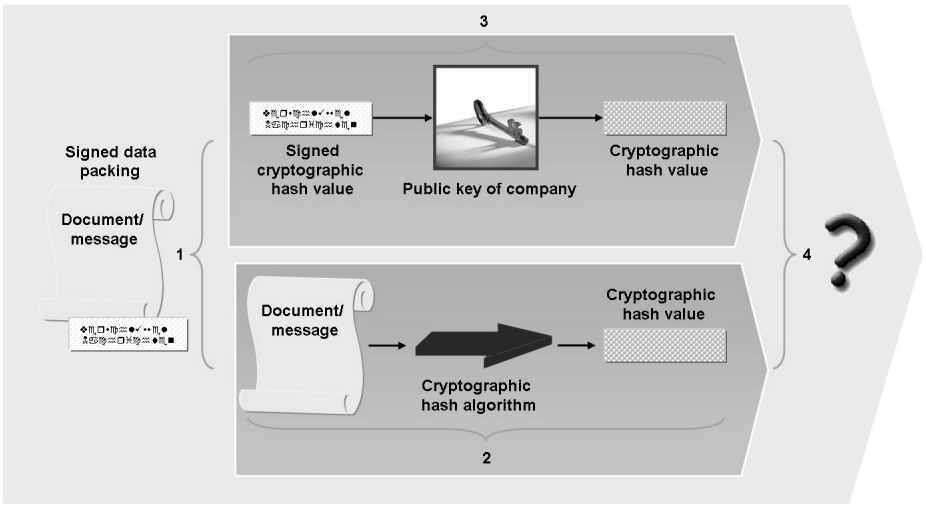


Figure 142: Verifying a Digital Signature

The following explains what happens at each step:

1. The digitally signed document is broken down into its components: the signed cryptographic hash value and the document itself.
2. The public key is applied to the signed cryptographic hash value. This allows you to obtain the cryptographic hash value of the original document.
3. The same hash algorithm that was used in the signing process is then applied to the document to be verified. The result is the cryptographic hash value of the signed document.
4. Both cryptographic hash values are compared.

Result

The result is either the acceptance or rejection of the digital signature, based on the following conclusions: If the cryptographic hash values are identical, then:

- The signer is who you think it is (that is, the signer is the owner of the private key that corresponds to the public key that you used to verify the signature).
- The document was not changed since it was signed

If the cryptographic hash values are not identical, then:

- The document may have been changed.
- The signer is not who you think it is (that is, the message was signed with a key other than the private key that corresponds to the public key that you used in the verification).

Digital Envelopes

Use a digital envelope to protect a digital document and prevent it from being visible to anyone other than the intended recipient. The following are possible reasons to use digital envelopes:

- Sending confidential data or documents across (possibly) insecure communication lines
- Storing confidential data or documents (for example, company-internal reports)

To create a digital envelope, you need access to the intended recipient's public key. How you obtain access to this public key depends on the public key infrastructure of your organization. You will also need the digital document that you want to protect. As an end user, you generally indicate that you want to *create an envelope* for a document and the system does the rest. The following figure shows what happens when you create a digital envelope.

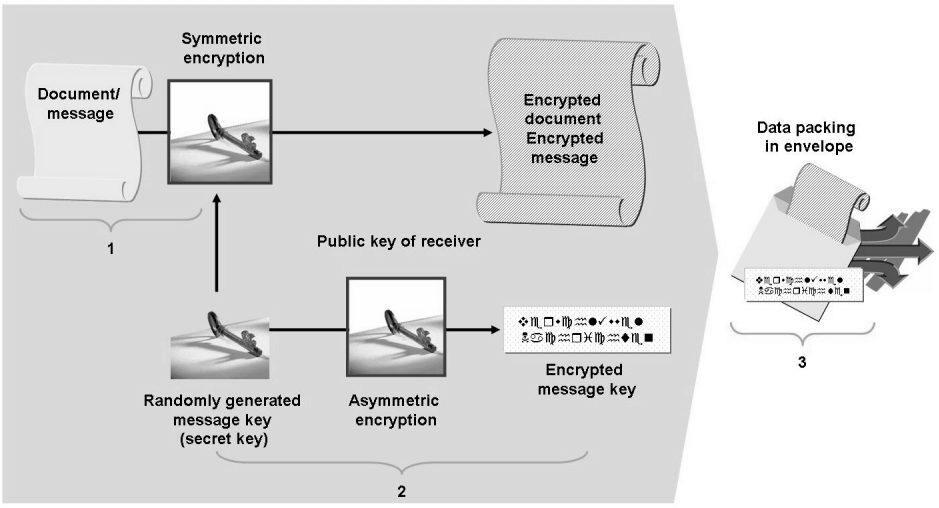


Figure 143: Creating a Digital Envelope

The following explains what happens at each step:

1. The message is encrypted using symmetric encryption. A new, randomly generated message key (secret key) is typically used for the encryption.

   Symmetric encryption means that the same (secret) key is used for both encryption and decryption. Anyone wanting to decrypt the message will need access to this key.

2. To transfer the secret key between the parties, the secret key is encrypted using the recipient's public key.
3. The encrypted document and the encrypted message key are packed together into the same data package for saving or for sending on to the intended recipient.

You receive a secured digital document that only the owner of the corresponding private key can view.

Document security

A WS Security header is added to provide authentication data such as a *wsse:Username* element or XML signature for the request and response message.

To activate document security, create security profiles based on templates. You assign these security profiles to Web service operations. To do this:

- Call transaction WSSPROFILE.
- Enter the name of the profile and choose *Create*.
- From the dialog box with the value list, choose a template for the security profile.

Hint: Profiles with a digital signature can only be used if the J2EE engine on your application server has been started.

- Enter the required data and save the profile.
- Assign the security profiles to the Web service operations.

Template:	Action:
CHECK_USERNAME CHECK_USERNAME_TIMESTAMP	**Action:** The Internet Communication Framework (ICF) of the ABAP stack requires a user to log on. If logon is to take place using a WS Security Username token, a user switch takes place during the SOAP runtime. Therefore, in this case a service user must be stored in transaction SICF. Choose an access profile based on the CHECK_USERNAME or CHECK_USERNAME_ TIMESTAMP template. In the ICF node, assign a valid service user in *Anonymous Logon Data*. This service user must not be a member of a security role. It is used by the SOAP runtime until the security protocol verifies the response using the wsse:Username security token.
CHECK_SIGNATURE	The XML signature contained in the SOAP document is verified. The verification itself takes place in the Java stack (for information about configuration, see Configuring Signature Processing). The configuration requires you to specify the J2EE keystore view that contains the Trusted Certificates.
SET_SIGNATURE	An XML signature is added to the SOAP document. The signature is created in the Java stack (see Configuring Signature Processing). The configuration requires you to specify the J2EE keystore view and the J2EE keystore alias for the private key.
SET_Username	A Username token is added to the SOAP document. You must specify a user name and password in the configuration.
SET_USERNAME_TIMESTAMP	A Username token and a time stamp are added to the SOAP document. You must specify the user name, password, and validity period in the configuration.

Figure 144: WS Security Profile Options

Using WS Security

To configure a Web service with WS Security, complete the following two steps:

- A profile with runtime configuration settings (such as X.509 certificate data) is required for each WS Security template used.
- After the WS Security profiles are created, you need to assign these profiles to the operations. You could assign a profile to several operations, for example, if the same certificate needs to be used for an XML signature, or if different profiles based on the same template need to be used for operations with different XML signatures.

WS Security profiles

The following WS Security templates are available for inbound/outbound messages.

Security Template:	Action:	Configuration Parameters:
CHECK_SIGNATURE	Checks the signature using SOAP:Envelope/SOAP:Body and SOAP:Envelope/SOAP:Header/wsse:Security/wsu:Timestamp, and checks the validity of the time stamp.	J2EE keystore view with the certificates of the Trusted Certificate authorities. A user-mapping between the X.509 certificate and the user is used for authentication.
CHECK_USERNAME	Authenticates the sender using the SOAP: Envelope/SOAP:Header/wsse:Security/wsse:Username element, which contains a time stamp, a user name, and a password.	None
CHECK_USERNAME_TIMESTAMP	Checks the validity of the SOAP:Envelope/SOAP:Header/wsse: Security/wsu:Timestamp and authenticates the sender by using a SOAP:Envelope/SOAP:Header/wsse: Security/wsse:Username element in the message, which contains a time stamp, a user name, and a password.	Maximum age of time stamp

Figure 145: Outbound Message (Client Request, Server Response)

Security Template:	Action:	Configuration Parameters:
SET_SIGNATURE	Adds a wsu:Timestamp to the message and signs the SOAP:Envelope/SOAP:Header/wsse:Security/wsu:Timestamp and SOAP:Envelope/SOAP:Body elements.	J2EE keystore view, J2EE keystore alias for the signature key
SET_USERNAME	Adds a SOAP:Envelope/SOAP:Header/wsse:Security/wsse:Username element to the message. This contains a time stamp, a user name, and a password. The password is encrypted, provided that the Cryptographic Toolkit has been installed.	User name, password
SET_USERNAME_TIMESTAMP	Adds a SOAP:Envelope/SOAP:Header/wsse:Security/wsu:Timestamp and a SOAP:Envelope/SOAP:Header/wsse:Security/wsse:Username element to the message. This contains a user name, and a password.	User name, password

Figure 146: Inbound Messages (Client Response, Server Request)

Using the Secure Sockets Layer protocol with SAP Web AS ABAP

Use

You can use the Secure Sockets Layer (SSL) protocol to make HTTP connections to and from the SAP Web Application Server more secure. When SSL is used, the data being transferred between the two parties (client and server) is encrypted and

the two partners can be authenticated. For example, if a user needs to transfer his or her account information, you can use SSL to authenticate the user and encrypt the information during transfer.

Hint: Users wanting to access a service that is protected with SSL will need to use the prefix HTTPS in the URL instead of HTTP.

Prerequisites

The server must have a public key pair and certificate. The SSL protocol must use public key technology. The server must therefore possess a public key pair and a corresponding public key certificate. It requires a key pair and certificate to identify itself as the server component, and a separate key pair and certificate if it needs to identify itself as a client component. These key pairs and certificates are stored in the server's own Personal Security Environments (PSEs) in the SSL server PSE and the SSL client PSE, respectively. You must be authorized to access the SAP Cryptographic Library.

Important: The distribution of the SAP Cryptographic Library is subject to and controlled by German export regulations and is not available to all customers. In addition, the library may be subject to local regulations in your own country that may further restrict the import, use, and (re-)export of cryptographic software If you have any questions on this issue, contact your local SAP subsidiary.

Features

With SSL, the SAP Web Application Server can provide the following functionality:

- **Server-side authentication** With server-side authentication, the server identifies itself to the client when the connection is established. This reduces the risk of using "fake" servers to obtain information from clients.
- **Client-side authentication** With client-side authentication, the client identifies itself when the connection is established. You can use SSL client-side authentication, for example, to authenticate users instead of using user IDs and passwords.
- **Mutual authentication** In this case, both the server and the client are authenticated.
- **Data encryption** In addition to authenticating the communication partners, the data being transferred between the client and server is encrypted, which provides for integrity and privacy protection. An eavesdropper cannot access or manipulate the data.

Use the following functions to maintain the server's SSL information:

- Profile parameter maintenance (transaction RZ10)
- Trust Manager (transaction STRUST)
- RFC destination maintenance (transaction SM59)
- ICM Monitor (transaction SMICM)
- SAPGENPSE configuration tool (for configuring a stand-alone SAP Web Dispatcher)

SAP Web Dispatcher

Each application server instance has its own server certificate. The client therefore needs to be able to externally address exactly this instance. The use of **load balancing** eliminates these restrictions. Load balancing resolves the SSL encryption before the application server instance.

Purpose

The SAP Web Dispatcher stands between the Internet and your SAP system. It is the point of entry for HTTP(S) requests in your system, which consists of one or more SAP Web Application Servers. As a software Web switch, it can refuse or accept connections and then process the request distribution for equal server balancing (load balancing).

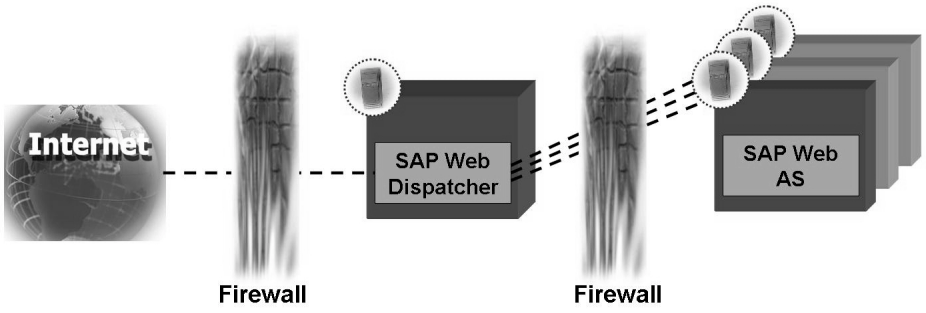


Figure 147: SAP Web Dispatcher

You can use the SAP Web Dispatcher in pure ABAP systems as well as in combined ABAP/J2EE systems and in pure J2EE systems.

Introduction notes

The SAP Web Dispatcher is recommended if you are using a SAP system with multiple SAP Web Application Servers for Web applications. The SAP Web Dispatcher is a separate program that you can run on a machine that is directly connected to the Internet.

Features

The SAP Web Dispatcher performs the following tasks:

- Chooses an appropriate application server (persistence in stateful applications, load balancing, ABAP, or J2EE server)
- URL filtering: You can define URLs that should be rejected
- Forwards, schedules, and (re)-encrypts SSL according to the configuration

Restrictions

The SAP Web Dispatcher is only relevant for a Web environment. In a classic SAP system, load balancing is carried out by the message server. SAP Web Dispatcher only forwards incoming HTTP(S) requests to the SAP Web Application Server. The response is then sent back to the client. Outgoing requests (to another SAP Web Application Server, for example) are not sent via the SAP Web Dispatcher; instead, they are sent via the proxy server for the corresponding intranet.

The main point to note for this security concept is that data communication via the Internet must be secured and the internal systems must be protected against external access. Authenticity and the security of data traffic within the organization are not discussed here.

Internal Use SAP Partner Only

Glossary

enterprise service

An enterprise service does not focus on detailed functions, but rather on a complete, industry-specific process that may consist of many small, individual steps. The enterprise service comprises all of the individual actions and therefore provides context-based business process logic. Examples include canceling a purchase order, processing a purchase order, and so on. The contextual nature of the service is important because the individual functions in an order cancellation service in the automobile industry will differ from those in a similar service in the media sector, for example.

Enterprise Services Architecture

Enterprise Services Architecture (ESA) is a new architectural model for SAP applications. Broadly speaking, ESA is not a product. Rather, it is an SAP concept for converting a service-based solution architecture. Enterprise Services Architecture will be used as a basis for all SAP solutions in the future. SAP solutions provide fully integrated processes that are based on Web services and that connect existing applications. In Enterprise Services Architecture (ESA), you have role-based user interfaces and the option of incorporating business processes that exist as enterprise services and whose data may be stored in diverse databases into your company's business process environment in accordance with general standards and with a minimum of effort. Using cross-system process definitions and workflow-based process control, you can reduce the amount of work that has to be completed by your “human integrators”.

HTML

Hypertext Markup Language

UDDI

Universal Description, Discovery, and Integration

W3C

World Wide Web Consortium

Web service

Web services are small, modular applications that can be made available on the Internet and are usually accessed as detailed functions in applications. There are agreed standards for describing and calling Web services. (Web Service Description Language (WSDL), Universal Description, Discovery, and Integration (UDDI), and Simple Object Access Protocol (SOAP)).

Internal Use SAP Partner Only

WS Security
WS Security is a standard that can be used to secure SOAP messages.

WSDL
Web Service Description Language

XML
eXtensible Markup Language

XSL
eXtensible Stylesheet Language

Internal Use SAP Partner Only

Index

Internal Use SAP Partner Only

Internal Use SAP Partner Only

Feedback

SAP AG has made every effort in the preparation of this course to ensure the accuracy and completeness of the materials. If you have any corrections or suggestions for improvement, please record them in the appropriate place in the course evaluation.

Internal Use SAP Partner Only