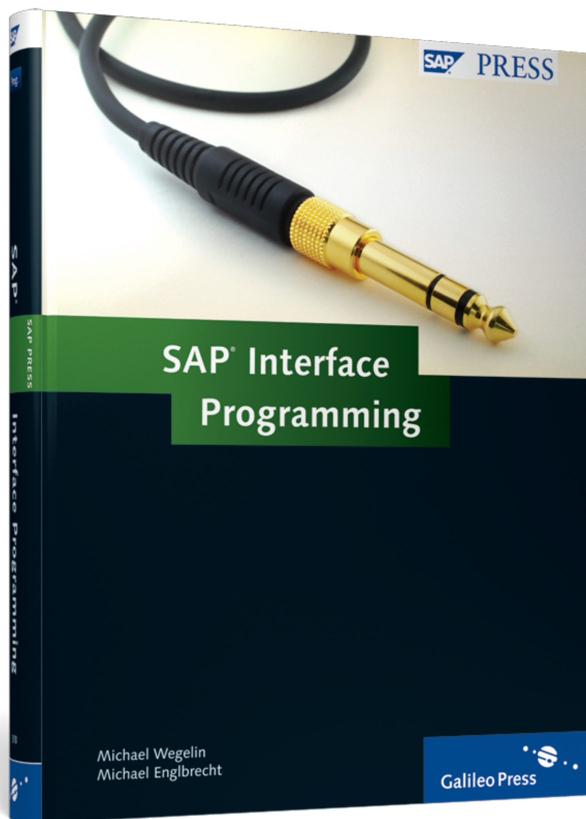
Michael Wegelin and Michael Englbrecht

# SAP® Interface Programming



Bonn • Boston

# Contents at a Glance

1 Basic Principles of SAP Interface Programming ......... 13

2 Remote Function Call with ABAP ............................... 95

3 Remote Function Call with C ..................................... 145

4 Remote Function Call with Java ................................. 217

5 IDocs and ALE ........................................................... 271

6 Service-Oriented Architecture Protocol ...................... 311

7 SAP NetWeaver Process Integration .......................... 347

A Bibliography ............................................................... 391

B The Authors ............................................................... 393

# Contents

Preface ........ 11

**1 Basic Principles of SAP Interface Programming ........ 13**

1.1 SAP NetWeaver Application Server ........ 13
1.1.1 SAP Solutions and SAP NetWeaver ........ 13
1.1.2 SAP NetWeaver Application Server ABAP ........ 16
1.1.3 SAP NetWeaver Application Server Java ........ 29
1.2 Security ........ 32
1.2.1 Security in Heterogeneous Landscapes ........ 32
1.2.2 User Management Engine ........ 36
1.2.3 KeyStores: Authentication, Signatures, Encryption ........ 38
1.2.4 Authentication and Authorization ........ 40
1.3 Programming SAP NetWeaver AS ABAP ........ 57
1.3.1 ABAP Dictionary ........ 58
1.3.2 Authentication and Authorization ........ 63
1.3.3 Number Ranges ........ 64
1.3.4 Function Modules ........ 64
1.3.5 Update Modules ........ 67
1.3.6 Application Functions and User Interfaces ........ 68
1.4 Overview of SAP Interface Technologies ........ 70
1.4.1 File Interface ........ 70
1.4.2 Remote Function Call ........ 72
1.4.3 BAPIs ........ 82
1.4.4 Application Link Enabling ........ 84
1.4.5 SOAP ........ 90
1.4.6 XI SOAP ........ 92

**2 Remote Function Call with ABAP ........ 95**

2.1 RFC Function Modules in ABAP ........ 95
2.1.1 Function Modules for Reading ........ 95
2.1.2 Call via sRFC ........ 98
2.1.3 Function Modules for Deleting and Changing ... 101

2.2 Transactional RFC ........ 106
2.3 Queued RFC ........ 111
2.3.1 qRFC with Outbound Queue ........ 112
2.3.2 qRFC with Outbound and Inbound Queue ........ 115
2.4 Business Objects and BAPIs ........ 118
2.4.1 Developing Business Objects ........ 118
2.4.2 Developing BAPIs ........ 119
2.4.3 "Helpvalues" Business Object ........ 141

**3 Remote Function Call with C ........ 145**

3.1 C RFC Library ........ 145
3.1.1 RFC Software Development Kit ........ 146
3.1.2 Connection Tests ........ 148
3.1.3 Compiling and Linking ........ 152
3.2 Simple RFC Clients and RFC Parameters ........ 153
3.2.1 Structure of an RFC Client Program ........ 154
3.2.2 Simple Parameters ........ 164
3.2.3 Structured Parameters ........ 167
3.2.4 Table Parameters ........ 171
3.3 More Complex RFC Clients ........ 174
3.3.1 Calling BAPIs ........ 174
3.3.2 Transactional RFC ........ 177
3.3.3 Queued RFC ........ 188
3.4 RFC Server ........ 189
3.4.1 Logon to Gateway ........ 192
3.4.2 Installing and Executing Functions ........ 194
3.4.3 Dispatching ........ 202
3.4.4 Transactional RFC ........ 205

**4 Remote Function Call with Java ........ 217**

4.1 SAP Java Connector ........ 217
4.1.1 Installation ........ 217
4.1.2 Architecture of SAP Java Connector ........ 218
4.1.3 Programming with SAP Java Connector ........ 220
4.1.4 Processing Tables and Structures ........ 229
4.1.5 Transactional RFC ........ 232

4.1.6 Queued RFC ..... 234
4.1.7 Metadata Processing ..... 236
4.2 Enterprise Connector ..... 237
4.2.1 Generating Proxy Classes ..... 238
4.2.2 Programming the Client ..... 243
4.3 RFC Server ..... 244
4.3.1 Server-Side and Client-Side Repository ..... 245
4.3.2 Programming a Simple JCo Server ..... 247
4.3.3 Registering a Function Handler ..... 249
4.3.4 Managing Transactions ..... 251
4.4 JCo RFC Provider Service ..... 252
4.5 SAP NetWeaver Portal Connector Framework ..... 256
4.5.1 Java Connector Architecture ..... 257
4.5.2 System Landscape of the Portal ..... 258
4.5.3 Introduction to Programming in the Portal ..... 261
4.5.4 Application Example of the Connector Framework ..... 264

**5 IDocs and ALE ..... 271**

5.1 IDocs ..... 272
5.1.1 Developing IDocs ..... 272
5.1.2 Creating IDocs ..... 275
5.1.3 Inbound Processing of IDocs ..... 279
5.2 ALE ..... 284
5.2.1 ALE Configuration ..... 284
5.2.2 Testing and Monitoring ..... 290
5.2.3 ALE Interface for BAPIs ..... 292
5.3 IDoc Programming with the C RFC Library ..... 296
5.3.1 IDoc Receiver ..... 297
5.3.2 IDoc Sender ..... 300
5.4 IDoc Programming with Java and JCo ..... 302
5.4.1 Preparation for the Use of IDoc Libraries ..... 302
5.4.2 Client Application for IDocs ..... 303
5.4.3 IDoc Server ..... 307
5.4.4 Configuration for the Dispatch of IDocs ..... 310

6 Service-Oriented Architecture Protocol ...................... 311

6.1 Web Services and Clients with SAP NetWeaver AS ABAP ........ 311
6.1.1 ABAP SOAP Web Service ........ 311
6.1.2 ABAP SOAP Web Client ........ 319
6.2 Web Services and Clients with SAP NetWeaver AS Java 323
6.2.1 Web Service Infrastructure in SAP NetWeaver AS Java ........ 323
6.2.2 Web Service Provider with J2EE ........ 326
6.2.3 Web Service Clients ........ 333
6.2.4 Adaptive Web Service with Web Dynpro ........ 338
6.3 SOAP Programming with Java ........ 338
6.3.1 Java API for XML Web Services ........ 338
6.3.2 Implementing a Web Service Client ........ 340
6.3.3 Implementing a Web Service Provider ........ 341
6.4 SOAP Programming with C# ........ 342

7 SAP NetWeaver Process Integration ........................ 347

7.1 SAP NetWeaver Exchange Infrastructure 3.0 ........ 347
7.1.1 System Landscape Directory ........ 349
7.1.2 Integration Repository ........ 352
7.1.3 ABAP XI Proxies ........ 356
7.1.4 Java XI Proxies ........ 360
7.1.5 Integration Directory ........ 366
7.2 SAP NetWeaver Process Integration 7.1 ........ 372
7.2.1 Service Interfaces ........ 372
7.2.2 ABAP SOAP Proxies ........ 374
7.2.3 Java SOAP Proxies ........ 383

Appendices ........................................ 391

A Bibliography ........ 391
B The Authors ........ 393

Index ........ 395

*This chapter summarizes the basic knowledge you will need for SAP interface programming: information about the SAP NetWeaver Application Server architecture, structure of ABAP programs, and supported application protocols.*

# 1 Basic Principles of SAP Interface Programming

This first chapter provides an overview of the SAP NetWeaver Application Server architecture (SAP NetWeaver AS) and focuses in particular on the components responsible for communicating with external systems. It gives you a brief introduction into programming with ABAP and Java and presents the ABAP Workbench and SAP NetWeaver Developer Studio (NWDS) tools. This is followed by an initial listing of the interfaces supported by SAP NetWeaver AS and also a description of the libraries and connectors for the C, Java, and C# programming languages required for interface programming.

## 1.1 SAP NetWeaver Application Server

In this section, we introduce you to the SAP NetWeaver AS architecture. We will concentrate here on components particularly important for interface programming. These include application layer components responsible for executing ABAP and Java programs.

### 1.1.1 SAP Solutions and SAP NetWeaver

SAP NetWeaver AS is the basic SAP NetWeaver module of the SAP infrastructure platform. SAP NetWeaver consists of a range of servers that provide different infrastructure services within an SAP system landscape:

- **SAP NetWeaver Application Server**
  Platform for other SAP NetWeaver servers and for business applications
- **SAP NetWeaver Business Warehouse**
  Data warehouse for integrating and analyzing data from an entire company
- **SAP NetWeaver Exchange Infrastructure**
  Integration platform for A2A and B2B scenarios
- **SAP NetWeaver Master Data Management**
  Platform for guaranteeing data consistency in application systems
- **SAP NetWeaver Mobile**
  Platform for mobile solutions
- **SAP NetWeaver Portal**
  Web portal that provides users with role-specific, company-wide access to relevant data and applications

Role of SAP NetWeaver AS

As you can see, SAP NetWeaver AS Server plays a key role within an SAP system landscape: It is the technology platform not only for infrastructural SAP NetWeaver servers but also for all SAP business applications such as SAP Customer Relationship Management (SAP CRM), SAP ERP Financials, SAP ERP Human Capital Management (SAP ERP HCM), SAP Product Lifecycle Management (SAP PLM), SAP Supplier Relationship Management (SAP SRM), and SAP Supply Chain Management (SAP SCM).

SAP NetWeaver AS is a portable application server that runs on many different hardware, operating system, and database platforms. It supports applications written in ABAP or Java. In principle, it therefore consists of two application servers that can either be installed separately (*single stack*) or together (*double stack*).

Client-server architecture

SAP NetWeaver AS forms the middle layer of a classic three-layer client-server architecture. This architecture comprises the data layer, application layer and presentation layer, as illustrated in Figure 1.1.

The *data layer* is composed of a relational database, for example, Oracle or Microsoft SQL Server. Business master data, transaction data, and system data are stored in database tables. The schemas for the ABAP and Java part are strictly separated from each other within the database. This

means that ABAP applications cannot directly access data in the Java part of the database, and vice versa.

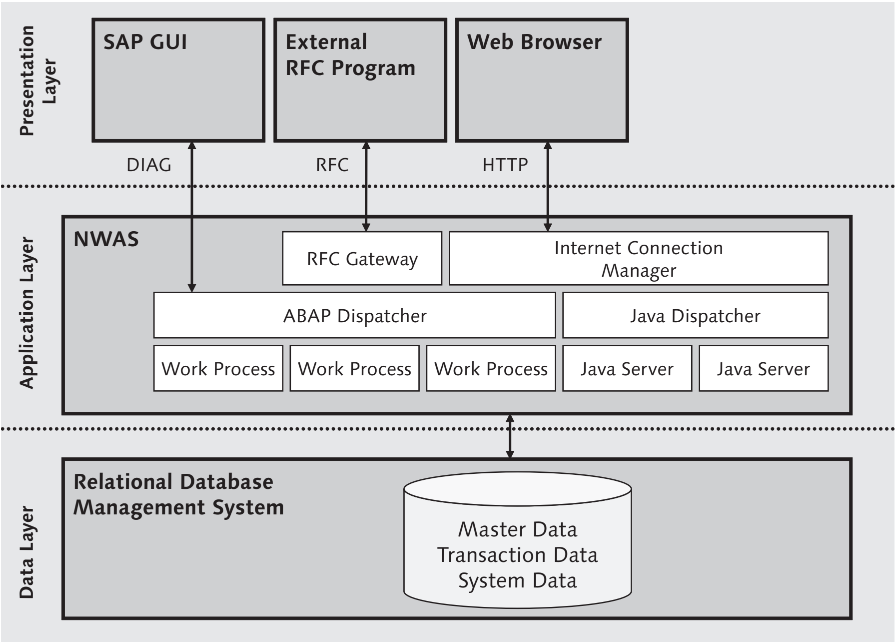



**Figure 1.1** Three-Layer Architecture of an SAP System

Application layer

The *application layer* is made up of processes that execute the application programs written either in ABAP or Java. Graphical user dialogs displayed by the presentation layer are also created in the application layer. SAP NetWeaver AS contains two technology stacks: the ABAP stack (light gray in Figure 1.1) and the Java stack (white in Figure 1.1). When installing SAP NetWeaver AS, you can either install only the ABAP stack (SAP NetWeaver AS ABAP) or the Java stack (SAP NetWeaver AS Java) or both stacks together (double stack system).

Presentation layer

The application layer results are displayed for the user on the *presentation layer*. This layer accepts the entries made by a user and forwards his commands to the application layer. The presentation layer is implemented using the *SAP Graphical User Interface* (SAP GUI) or an HTML browser. These programs are installed on the desktop PCs of users who want to work with the applications on SAP NetWeaver AS.

### 1.1.2 SAP NetWeaver Application Server ABAP

The majority of and most important SAP solutions are implemented in ABAP, so it is particularly important to know the architecture and runtime behavior of SAP NetWeaver AS ABAP. If you have never worked with an SAP system before, in the next section, you will learn how to log on to the system and navigate between applications, in advance of then learning about SAP NetWeaver AS ABAP in detail.

#### Logging On and Navigating with the SAP GUI

To work interactively with an ABAP system, start the *SAP GUI* on your local work station. This is a Windows application that can communicate with SAP NetWeaver AS ABAP through the *Dynamic Information and Action Gateway* (DIAG) communication protocol. As well as SAP GUI for Windows, there is also SAP GUI for Java and SAP GUI for HTML.

User session

When you start the SAP GUI, it connects to a SAP NetWeaver AS ABAP of the application layer. During a *user session*, which begins with the logon and ends with the logoff, you always work with exactly one SAP NetWeaver AS ABAP of an entire system. This server is specified for the SAP GUI at the start, or it is allocated by the SAP system message server through a load balancing mechanism.

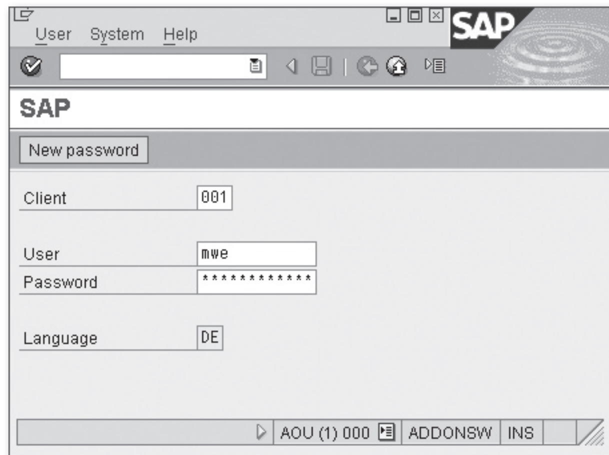



**Figure 1.2** SAP GUI Logon Screen

Logon

After you start the SAP GUI, a logon screen like the one shown in Figure 1.2 appears. You must specify four pieces of information to log on:

1. **Client**
   SAP systems are *client-enabled*, which means that within an SAP system, you can store types of data that, from a business perspective, are independent of each other. This is achieved by the fact that the first column of every business table contains the three-digit client number. The database interface for SAP NetWeaver AS ABAP adds the current logon client of the user to the `WHERE` clause of each database query. This means that a user only ever sees and processes data belonging to the client he has used to log on. In practice, this is generally used on quality assurance and test systems to separate different test datasets from one another.
2. **User**
   To enable a user to log on to the system, a client-based user master record must be created for him. The user is identified by a unique user name within the client. The user enters this user name when logging on.
3. **Password**
   The logon will only be successful if the user has entered the valid password for the user.
4. **Language**
   SAP systems are multilingual. Developers can translate all texts displayed on the GUI into different languages. The language the user specifies when logging on will determine the language in which these texts are output.

Screen layout

The SAP Easy Access screen opens after the user logs on. As you can see from Figure 1.3, the screen is divided into different areas:

- **Menu bar**
  The top line of the screen contains dropdown menus used to execute application functions. This menu bar includes different functions for each application. Only the last two menu options, System and Help, are the same for all applications.

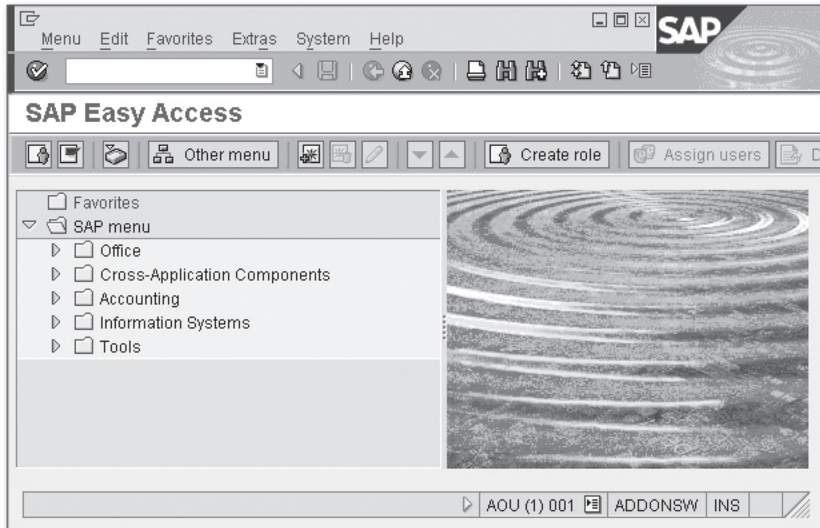



**Figure 1.3** SAP Easy Access: Default Initial Screen

- **Standard toolbar**
  The standard toolbar underneath the menu bar is the same for all applications and provides access to important system functions like Entry Check, Command Input, Save, Return, Exit, Cancel, Print, Search and Scroll in Lists.

  The first icon in the standard toolbar, a white checkmark on a green ball, triggers a check on the entries made by the user. Incorrect entries will cause the incorrect input fields to be displayed in red, and an error message will appear in the status bar at the bottom of the screen.

  Command field

  The *command field* for entering commands directly is not generally displayed after the first logon, but the user can display it by clicking the small gray triangle in the standard toolbar. Transactions (ABAP programs) can be started and finished by entering commands in the command field. Table 1.1 lists the most important commands.

- **Title bar**
  The title of the program just executed is displayed under the standard toolbar.

- **Application toolbar**
  The application toolbar is located under the title bar. A mouse click can trigger the most important application-specific functions here.
- **Screen**
  The majority of the screen occupies an area where a user can enter data, choose functions, or view lists. The SAP Easy Access application has a logo in the right-hand window area. The left window area contains a tree, the SAP Menu, from which business programs can be started.
- **Status bar**
  The status bar appears at the very bottom. System messages (errors, warnings, and information) are displayed here. Other information such as the system ID, logon client, user name, transaction code of the application just executed, or host name of the application server are also provided there. This additional information may also initially be concealed behind a small gray triangle.

| Command | Function |
|---|---|
| /n | Ends the running transaction. |
| /nxxxx | Ends the running transaction and starts Transaction xxxx. |
| /i | Deletes the current external mode. |
| /o | Lists all external modes. A new external mode can be started from the dialog box. |
| /oxxxx | Starts a new external mode with Transaction xxxx. |

**Table 1.1** Important Commands for Command Field in Standard Toolbar

**External mode**

During a user session, a user always communicates with the same SAP NetWeaver AS ABAP through the SAP GUI. The user can open up to six windows, or *external modes*, during a session. Different programs can run in these windows, between which the user can switch. SAP NetWeaver AS stores the entire user context within a session.

### SAP NetWeaver AS ABAP Instance

Within an SAP system, there is only one single database process on the data layer, but on the application layer, there may be several SAP NetWeaver AS ABAP instances that execute ABAP programs. Each

*instance* in turn consists of different processes that perform specifically limited tasks. An instance is uniquely identified by the specification of three parameters:

- **System ID**
  The system ID identifies an SAP system uniquely within a group of systems. The system ID is an identifier consisting of three characters (letters and digits), for example, SD1. The system ID is normally used as the name for an SAP system database.
- **Host name**
  All SAP instance processes run on the same host. However, there may be several SAP instances within a system that generally run on different hosts. The host name therefore has to be specified to identify an instance.
- **System number**
  Several SAP instances, however, can also run on a host. To distinguish different instances uniquely on the same host, a third specification is therefore needed, the two-digit system number. This system number can accept any value between 00 and 99. The system numbers of two SAP instances on the same host must be different, even if these SAP instances belong to different SAP systems.

An SAP instance name is therefore made up of three components: *<SID>*_*<HOST>*_*<SYSNR>*.

*<SID>* here is the three-character system ID, *<HOST>* is the host name of the application server, and *<SYSNR>* is the two-digit system number of the SAP instance.

Communication protocols

An SAP instance can communicate with external programs through different application protocols. These application protocols are transported through the *Transaction Control Protocol* (TCP). An SAP instance component opens a TCP port on its host for each application protocol. SAP Basis supports the following three protocols:

- **Dynamic Information and Action Gateway (DIAG)**
  DIAG is an application protocol used for communication between the SAP GUI and SAP instance. The SAP instance uses this protocol to send screens and lists to the SAP GUI and to accept user commands.

The *dispatcher* implements this protocol within the SAP instance. The dispatcher opens the TCP port with the number 32<*SYSNR*> on the host to communicate with DIAG. Because the port numbers must be unique, every instance on a host must have a different port number. In the services file of computers that want to communicate with an SAP instance through this port, the *sapdp*<*SYSNR*> service name is assigned to the port.

- **Remote Function Call (RFC)**
  RFC is an SAP application protocol that external programs can use to call function modules in the SAP instance. In this case, parameters can be transferred to the function module and received by it.

  RFC is implemented by the *gateway*. The gateway opens the TCP port with the number 33<*SYSNR*> on the host; the corresponding service name is sapgw<*SYSNR*>.

  When the SAP GUI is being installed on a computer, the service names and relevant port numbers are automatically entered into the services file. The services file is located under <*WINDIR*>*/system32/drivers/etc* on Windows systems. Listing 1.1 shows an extract from this services file.

```
# Copyright (c) 1993-1999 Microsoft Corp.
#
# This file contains port numbers for
# known services in accordance with IANA.
#
# Format:
#
# <service name>  <port number>/<protocol>  [Alias...]
echo                7/tcp
echo                7/udp
discard             9/tcp    sink null
#...
sapdp00          3200/tcp
sapdp01          3201/tcp
sapdp02          3202/tcp
#...
sapgw00          3300/tcp
sapgw01          3301/tcp
sapgw02          3302/tcp
#...
```

**Listing 1.1** Assigning Service Names to Port Numbers in Services File

- **Hypertext Transfer Protocol (HTTP)**
  Since Release 6.10, SAP Basis has provided an additional application protocol with HTTP. This protocol is implemented by the *Internet Connection Manager* (ICM). ICM accepts HTTP requests and forwards them to corresponding applications. The standard port for the HTTP service is 80, but an SAP system administrator can choose any other TCP port for this service.

Work processes

The dispatcher plays a key role within an SAP instance. It accepts requests and forwards them for execution to the next free work process.

**Optimum Use of Resources by Dispatching to Work Processes**

There are generally a great deal fewer work processes within an SAP instance than there are active user logons to this instance. This is because a considerable amount of time usually elapses between two requests from a user — from the perspective of the computer at any rate. During the time in which a user reads what is output on a screen to when he enters data, the work process can process a request from another user.

Owing to the fact that only some work processes can process requests from many users, resources can be optimally used on the application server. Significantly fewer processes are run, and there are also considerably fewer connections to the database, as would be the case if a separate work process were assigned to each user.

However, this also noticeably affects the transaction concept, as we will illustrate in this section under the heading, ABAP Transaction Concept.

Types of work processes

The types of work processes managed by an SAP instance dispatcher differ and are each identified by one of the letters D, B, S, U or E.

- **Dialog work processes (D)**
  These work processes execute ABAP programs in a user dialog and function modules called through RFC.
- **Background work processes (B)**
  These work processes execute ABAP programs that were scheduled for background processing. These ABAP programs do not conduct a user dialog, and the required entries are already specified at the time they are scheduled.
- **Spool work processes (S)**
  These work processes are responsible for printing lists.

- **Update work processes (U)**
  These work processes execute programs that write changes into business database tables. These programs are triggered from dialog programs, as we will discuss in more detail later.

- **Lock management work process (E)**
  These work processes are represented by the letter E from the term *enqueue* (which means to place a data item in a queue). There is exactly one instance with exactly one lock management work process within an SAP system. ABAP programs use this process to set and release data record locks. It is important that only one of the many dialog processes of the different servers can ever set a specific lock at a time. For this reason, there must only be one E process in the whole SAP system, and lock requests must be lined up in a single queue. We will look at this concept in more detail in this section under the heading, Lock Objects.

**Memory areas**

In addition to these work processes, an instance mainly has memory areas that all work processes can share:

- **Buffer area for database contents**
  Data that was read from a work process from the database can be stored in this buffer area and is then available for all work processes. This speeds up read access to this type of data.

- **Roll area for user contexts**
  The contexts of logged-on users are stored in this memory area. In particular, these include the statuses of programs that have just been executed for a user.

**Load distribution through the message server**

There is a specific instance in an SAP system that makes another central service available with the *message server*. The message server is used for communicating between dispatchers of all SAP instances in an SAP system. It is through the message server that work processes of an instance find the enqueue process responsible for lock management.

Another important message server task is distributing the load between SAP instances. The administrator can divide SAP instances of an SAP system into several logon groups. To distribute the load between SAP instances of a logon group, the SAP GUI can be configured to address the message server at the start so that it can be distributed to an SAP instance of a specific logon group. Owing to the periodic SAP instance messages

to the message server through their momentary loading, the message server can select a suitable SAP instance from the requested logon group and assign it to the SAP GUI for logging on.

The SAP GUI and dispatchers also communicate with the message server through TCP. The SAP system administrator assigns a port number to the message server. The service name chosen and entered into the services files is *sapms<SID>*.

The SAP GUI and other client programs consequently have two options to connect with an SAP instance: directly by specifying the application server host name and the system number, or indirectly by specifying the host name and service name of the message server. These two options are summarized in Table 1.2.

| Connection Type | Required Information |
|---|---|
| Without load distribution | ▶ Host name or IP address of application server.<br>▶ System number of ABAP instance.<br>▶ The host name and service name of an alternative gateway can also be specified for RFC connections if the gateway of the specified application server is not to be used. |
| With load distribution | ▶ Host name or IP address of message server of SAP system.<br>▶ Service name of message server.<br>▶ Logon group. |

**Table 1.2** Information a Client Must Provide to Connect to SAP NetWeaver AS ABAP

### Firewalls and SAP Routers

Two connection options

External clients generally have no direct access to an SAP instance through TCP/IP. Instead, they are separated from the SAP instance by firewalls and at least one (though normally two) SAP routers, as shown in Figure 1.4.

An *SAP router* bundles client requests onto several SAP instances. This means that only the route to the IP address and the SAP router port must be released in the firewall before the SAP router. When installing the SAP router, the administrator usually chooses 3299/TCP as the TCP port. This

is the highest possible port number for a dispatcher, so it is unlikely that this port is already being used by an SAP instance dispatcher.

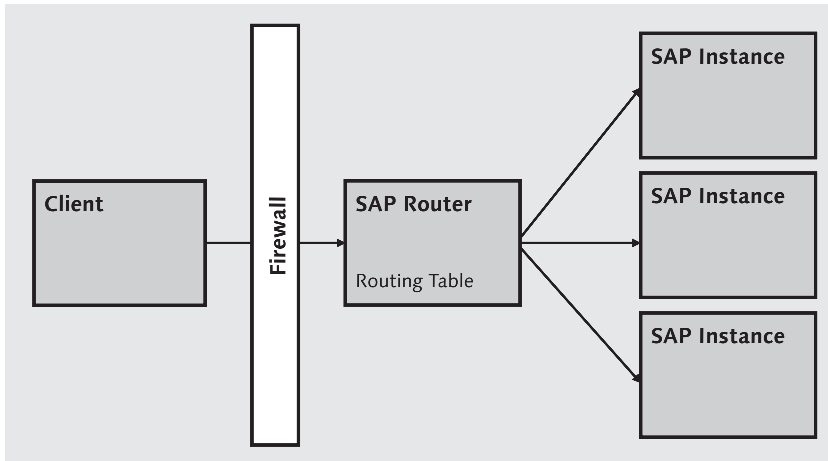



**Figure 1.4** Firewall and SAP Router Ensure Access to SAP Instances

The SAP router contains a table where the administrator releases or locks routes from clients (identified by their IP address) to SAP instances (identified by their IP address and system number).

If a client wants to connect to an SAP instance, in addition to specifying the application server host name and instance system number, it must also indicate the host name or IP address of the SAP router and its port number. This information is specified in the *SAP router string*, which prefixes the application server host name:

SAP router string

```
/H/<routerhost>/S/<routerport>/H/
```

`<routerhost>` here is the host name or IP address of the SAP router, and `<routerport>` is its port number.

### Dialog Processing

The diagram in Figure 1.5 shows the process flow of a *dialog transaction*. The user starts an SAP transaction on the SAP GUI by entering the transaction code in the command field on the standard toolbar.

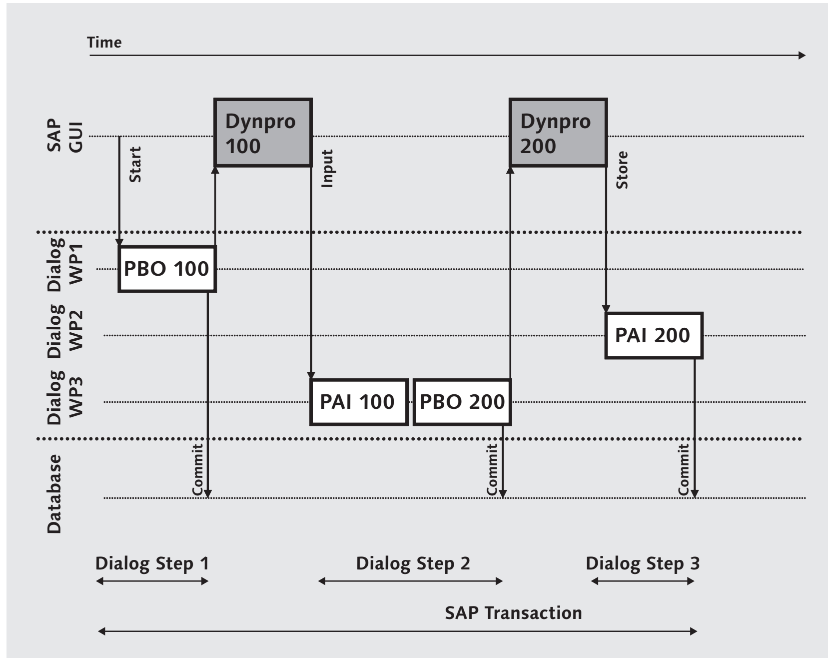



**Figure 1.5** Dialog Transaction Process Flow

In the example shown in this figure, two screens are displayed for the user in the course of the Transaction: Dynpro 100 and Dynpro 200. User entries, such as the key for the data record to be processed, are made in Dynpro 100. The data record is changed, and its backup triggered in Dynpro 200.

Dialog steps

From the perspective of the application server, this SAP transaction is divided into three *dialog steps* that can be executed by three different dialog work processes. The flow logic is defined with the screen when dialogs are processed in SAP systems. *Process After Input modules* (PAI modules) are processed after the user has made his entries on the SAP GUI. *Process Before Output modules* (PBO modules) are processed before the screen is displayed on the SAP GUI. Therefore, a dialog step consists of the PAI modules of one screen and PBO modules of the next screen.

**One Dialog Work Process for Many Users**

After a dialog work process has processed a dialog step, it is immediately available again for the next user. This is possible because dialog work processes work very quickly compared to the speed at which a user works.

As a result, many users can be dealt with on the application and database servers using a small number of dialog work processes and therefore a relatively low consumption of resources.

**Implicit database commit**

Because a work process carries out completely unrelated dialog steps consecutively, it must execute a commit on the database after every dialog step. This is triggered by SAP Basis without any further intervention from the ABAP developer. This means that database changes that were made in this dialog step are committed on the database after each dialog step without an explicit `COMMIT WORK` statement by the ABAP developer.

If the ABAP developer programs an explicit `ROLLBACK WORK`, this rollback will of course also be forwarded to the database. A rollback on the database is therefore only ever possible up to the beginning of the current dialog step. A rollback on the database across several dialog steps is not possible with database-related technology.

If a user transaction therefore stretches across several dialog steps, specific precautionary measures must be taken to ensure a rollback is possible if an error occurs or if the user cancels a transaction.

### ABAP Transaction Concept

A user transaction that may stretch across several dialog steps is also known as an *SAP transaction* or *SAP Logical Unit of Work* (SAP LUW) and therefore differs from the database transaction or *Database Logical Unit of Work* (DB LUW). As we explained in the previous section, specific precautionary measures must be taken for an SAP transaction to ensure that a rollback is possible if the user cancels a transaction or if an error occurs. This applies in particular to changing transactions that create, change, or delete master or transaction data in the database.

**Locks**

A changing SAP transaction generally begins with the user selecting from the initial screen the data record he wants to change. In the first dialog step, the dialog work process sets a lock on this data record. To do this, it communicates with the enqueue work process, exactly one of which

runs in every SAP system. If a lock has not yet been set on the requested data record, the enqueue work process enters the lock in its lock table. If a lock is already set, the dialog work process must issue a corresponding error message to the user and cannot continue with the intended change.

In the subsequent dialog steps, the dialog work process in the dialog with the user collects the changes to be made to the data record. These changes are not carried out immediately but are collected and flagged for changing.

Asynchronous updating

SAP Basis provides *update function modules* technology for this purpose. These are function modules where the developer encapsulates changing the `UPDATE`, `INSERT` and `DELETE` database statements and which have been flagged by the developer as update function modules. While the changes are being collected, the dialog program calls these function modules and transfers the data to be changed to them through their interface. The SAP system does not execute these function modules straightaway but stores the name of the function module and transferred data in the update table for executing later.

Commit and rollback

This means the user can easily cancel the SAP transaction at any time. In this case, or if an error occurs during the SAP transaction, the dialog program simply has to execute the `ROLLBACK WORK` statement. The SAP system then rejects all entries in the update table and deletes the set locks.

If the user exits the SAP transaction with the request to save his or her data, the dialog program executes the `COMMIT WORK` statement. In this case, the SAP system starts a new task asynchronously in an update work process.

The update work process executes the flagged function modules in the sequence in which they were entered in the update table and, in doing so, transfers the data that was stored in the update table to them. Only now do the function modules actually implement the database changes.

If everything runs smoothly, the update work process executes a commit on the database and deletes the set locks. If an error occurs anywhere in one of the function modules, the update work process executes a rollback on the database and informs the user. Rollback is possible across several function modules because the update work process is working within a single database LUW.

### 1.1.3 SAP NetWeaver Application Server Java

The Java stack represents the second part of the application platform of the SAP NetWeaver infrastructure. SAP NetWeaver 7.0 implements the J2EE Version 1.3 specification developed by Sun Microsystems. Besides purely implementing the specification, SAP provides enhancements that simplify integration into SAP-based landscapes. Many known topics from the ABAP world are also integrated into the Java stack to produce a homogeneous landscape on both sides. The following section is a brief overview and is not intended to be an in-depth introduction to the individual aspects of the J2EE specification. We will return to some topics in the Java programming examples in Chapter 4, Remote Function Call with Java; Chapter 5, IDocs and ALE; and Chapter 6, Service-Oriented Architecture Protocol.

**Introduction to J2EE**

*Java 2 Enterprise Edition* (J2EE) was defined as the standard for developing large enterprise applications. The objective was to simplify the development of applications that require a well-built infrastructure. Reusing application components was also a key aim.

This was achieved by separating technical and specialist aspects. This is referred to as *separation of concerns*, whereby certain vertical aspects of an application or application infrastructure do not have to be implemented again every time by the relevant development team. The infrastructure undertakes the implementation and provision of technical aspects such as security, transactional processing, or similar important application components that require a high degree of knowledge and smooth implementation. Furthermore, the application server is also responsible for managing the different phases in the life of an application component.

**Application components**

The application components are called *Enterprise Java Beans* (EJB). The specification in Version 1.3 differentiates between three different component types:

- Session beans (SB)
- Entity beans
- Message-driven beans (MDB)

*Session beans* are differentiated in terms of whether they are stateful (*Stateful Session Beans*, SFSB) or stateless (*Stateless Session Bean*, SLSB). Session beans are used for implementing business functions.

The second component type, the *entity bean*, was defined to map persistence. Entity beans are provided as *container-managed* (CM) and *bean-managed* (BM) entity beans. The objective of CM entity beans is that the container itself implements the SQL-relevant aspects, and as a result, the developer only has to define declarative specifications at development time. Session beans, like entity beans, are called synchronously.

Message-driven beans in integration scenarios

In many infrastructures, particularly in integration scenarios, asynchronous communication is essential. The specification describes *message-driven beans* (MDB) for this purpose. As the name suggests, MDBs are based on the exchange of messages. However, the message generator is not (as is the case with session beans) directly in contact with the message receiver; rather, messages are delivered through the medium of *Message-Oriented Middleware* (MOM). There has been a proliferation of proprietary MOM implementations over the years, so Sun responded to this development and defined a standard. This standard is called *Java Message Service* (JMS). The JMS specification consists of an *Application Programming Interface* (API) specification and a *Service Provider Interface* (SPI) description. Senders and receivers of messages use the API part of the specification. The SPI part of the specification in contrast is implemented by the particular manufacturer of the JMS providers. This abstraction from a specific product or protocol for exchanging messages makes message-driven beans particularly appealing for integration scenarios.

Design by contract

All three components define their own lifecycle, which is nevertheless completely abstract for the developer. The developer can rely on the fact that the application server will handle the processing properly. The developer only comes in contact with the individual phases of the lifecycle to the extent the interface agreement stipulates for this between the application server and component. The interface agreement defines *hook methods* in this case. These methods represent the contract between the component and server. This is referred to as *design by contract*. Contracts control the interaction between the application components on the one hand and the application server on the other.

Problems with the current standard

At first glance, the benefits of separation of concerns and definition of contracts appear to be obvious. But if we look more closely at the separation of concerns in particular, it turns out only to be a shift of responsibilities for the application server that has to carry out most of the tasks. This significantly increases the complexity for the application server.

That is also precisely what happened with J2EE. At worst, applications that rely very heavily on this division of labor experience performance problems when the number of users increases.

One specific example of this is the persistence layer. If entity beans are used incorrectly, this can very quickly cause the performance of the application to deteriorate dramatically. This resulted in J2EE being used as the platform but without the use of EJBs.

Sun Microsystems could not ignore this development, and in the course of changing the Java language standard, the company also changed or completely restructured the J2EE standard at the same time. Due to the serious change, the standard was also given a new name. J2EE therefore became *Java Enterprise Edition* (JEE).

SAP also implemented this standard and released it for the first time with *SAP NetWeaver 7.1 Composition Environment* (SAP NetWeaver CE). SAP NetWeaver CE is the forerunner of SAP NetWeaver 7.2 but not its basis. SAP NetWeaver CE is a standalone platform. The aim of this infrastructure is to make it easy to implement composite applications and at the same time also provide an infrastructure for service orientation. This infrastructure supports you, like the development tool, in creating and managing an SOA landscape.

From the perspective of the JEE standard, the previous strong dependency of components on the application server was reduced. This is achieved by using annotations in the source text. Annotations supersede complex and error-prone XML configuration files (*Deployment Descriptors*).

**Java stack layers**

But let's return to the SAP NetWeaver Java 7.0 architecture. The Java stack of an SAP system runs within the application layer (refer to Figure 1.1). The Java server itself is divided into three layers, as is also normally the case with other Java application servers.

The top layer contains the *web container*. This is used as a runtime environment for executing applications based on servlets and JSPs (*Java Server Pages*). Both are technologies described within the J2EE specification and are ultimately used for dynamically programming web pages. In addition to the standard, the SAP web container provides a Web Dynpro runtime environment for Java applications.

The middle layer provides the environment for executing EJB components. The *EJB container* defines the runtime environment for the components. It contains session beans and entity beans as well as message-driven beans.

The lowest layer of the Java server is the *data access layer*. SAP searched very thoroughly here for an analogy to the ABAP world. When you use ABAP, you can trust that your application is completely database-independent — and so too the source code that accesses the database. This is guaranteed by using Open SQL. But because J2EE is a standard, and there is also a very much greater degree of freedom in the Java world in terms of accessing a database, SAP provides not just one technology but many technologies for accessing databases.

## 1.2 Security

Heterogeneous, integrative landscapes must offer a higher level of security. This is an important aspect when programming interfaces in particular. When you give external applications access to your SAP systems, you must make sure that these external systems only see the data and, above all, only change the data they are allowed to see and change. You must ensure that only correctly identified applications get access to your network.

Facets of security

The highest level of security can only be given if it consists of different facets. These facets include authenticating and authorizing users and encrypting messages. The following section describes how these facets look in theory, in order for us then to discuss how to implement security concepts.

### 1.2.1 Security in Heterogeneous Landscapes

In heterogeneous landscapes with a high degree of integration, security plays a very important role. Due to the nature of integration, however, one universal solution cannot be arrived at for security. We also have to understand that there is never 100 % security; all we can ultimately do is make it more difficult for attackers to achieve their objective.

There are two basic types of security in a company. One is *infrastructural security*. This is security that can be achieved through firewalls or backup

concepts. It is less about the software and more about the hardware used to protect the data and the infrastructure.

The other is *application security*. This type of security is achieved by using programming tools to get the highest possible level of security. We want to concentrate on this security type in this section. Infrastructural and application security can of course also be combined. An example of this is access control in companies.

**Requirements for security-critical systems**

As with all systems, you can generally also describe certain security-relevant landscape requirements for integrative landscapes:

- **System security**
  Monitoring concepts.
- **Data security**
  Confidentiality, integrity, and non-repudiation of data.
- **Access security**
  User authentication and authorization.

This multilevel security shows that interaction from different components and concepts increases the level of security in the system.

### Data Confidentiality

Data security contains concepts for data confidentiality, in other words, protection against unauthorized access to information exchanged between systems. This is guaranteed using encryption concepts. There are different concepts involved here. The two best-known concepts are *public-key cryptography* and *private-key cryptography*. They are also referred to as asymmetric and symmetric procedures.

**Symmetric encryption**

*Symmetric procedures* work with only one cryptographic key. The key is used for encrypting and decrypting data. Symmetric procedures are very difficult to use directly, particularly for encrypting communication data, because the key must be exchanged before communication.

**Asymmetric encryption**

*Asymmetric procedures* use two corresponding keys for cryptographic operations: a public key and a private key. All operations performed using one key can be undone or verified by the other key. Security is achieved by the fact that information can only be traced back from one of the keys to the other key in a very time-consuming way.

Besides purely symmetric or asymmetric cryptography, there are combinations of both options. One of the best known is *Secure Socket Layer* (SSL).

### Data Integrity

Besides confidentiality, data integrity also plays an important role in integration scenarios. Data integrity basically means that data exchanged between two communication participants cannot be changed, or at least changes cannot be established, during transport.

Hash functions

**Hash Functions**

The *hash value* is used for data integrity. The hash value is the result of processing a *hash function*. Hash functions are used wherever abridged information, displayed as clearly as possible, is required. Hash functions are *one-way functions*. Information must not be able to be traced back from a hash value to the displayed abridged information.

This now also raises the question of how the receiver can identify whether the data was changed when being transported. To detect a change, the actual status must be compared with the planned status. The information therefore actually has to be transferred twice. Before sending the information, the sender creates a hash value from the information to be transferred. The sender subsequently sends the information itself and then its hash value. The receiver can calculate the hash value from the data received and then compare it with the information received.

### Non-Repudiation of Information

Finally, data exchanged between two communication participants must be non-repudiated. Non-repudiation of information can, as in real life, be guaranteed by a signature.

Digital signatures

*Digital signatures* are used in different ways in heterogeneous systems. Digital signatures are quite easy to create. The hash value is calculated for information that is to be digitally signed. The hash value together with the private key of the signer is used as an entry for an algorithm for the digital signature. Finally, the information together with the digital signature and hash value can then be transferred to the receiver.

In accordance with the basic principle of asymmetric cryptography, the receiver of a digital signature must verify this signature using the public

key of the sender. This is not a security problem because the public key can be exchanged between communication participants without any cause for concern. However, for public keys to be exchanged without any problems, private keys must be kept absolutely secure. After all, what happens if the private key of the sender is compromised? An attacker could have obtained the private key and can now sign data in the name of another sender. Key management as central as possible therefore has to be established.

Key management

The term *key management* covers creating key pairs, managing *lock lists*, certifying keys, and providing a directory service. The key management area is the central point of contact for all services associated with the key material. It is also responsible for all communication partners being able to trust the key material. The keys of communication participants that were compromised are listed in the lock list. In integrative scenarios in particular, it is important that an incoming quotation actually comes from the sender specified on the quotation letter. This security can only be achieved if a common area responsible for the key material and corresponding infrastructure having a high level of security is set up. This central area must identify public keys from communication partners as being trustworthy. This is done using certificates.

Certificates

*Digital certificates* are nothing other than proof that the public key belongs to a certain identity and only to this identity. Certificates can be issued for not only a natural person but also for computers or applications and therefore also represent an option for authentication. Through a certificate, the central management area uses a digital signature to confirm that a key belongs to an identity. The signature itself or its verification is problematic here. The receiver needs the public key of the signer to verify whether a signature is correct. As already mentioned, the receiver obtains this key using the certificate from the signer and must in turn check the signature of the signer of that certificate. This checking of public keys can be very complex and drawn out. For this reason, the communication participant or application decides the level of *trust* as of which the key can be classified as secure. Infrastructures that represent this type of trust and are responsible for managing certificates are called *public-key infrastructures* (PKI). We do not want to discuss PKIs in great detail here, but you should note that using them may involve a great deal of effort in a company. (For more information, refer to PKI: *Implementing & Managing E-Security* [McGraw-Hill Professional, 2003].)

### 1.2.2 User Management Engine

Now that we have discussed the basic terms and security concepts, we will turn our attention to the SAP NetWeaver AS and implementing the said concepts.

The *User Management Engine* (UME) plays the principal role in SAP NetWeaver AS Java. The UME is a central service of the application server that is used to manage users, group affiliations, authorizations, and roles. Users and groups can be stored in different containers here and managed in databases, directory services (e.g., LDAP [*Lightweight Directory Access Protocol*]), or the ABAP stack itself.

Managing security identities

The type of installation can also affect the UME and the way security identities are stored. When a double stack is installed, the UME is automatically configured in such a way that the installed ABAP stack is used for saving users. Access to the J2EE stack is set to *read-only* by default. This means user data can only be read using the UME. The standard user SAPJSF is used for communicating between the UME and ABAP user management. If only the Java stack is installed, the UME can be configured through the *Visual Administrator* (VA) in such a way that users are retrieved from another ABAP system.

Using configuration files, the administrator basically has the freedom to manage security identities. He can change the configuration either using the J2EE Config tool or SAP NetWeaver Portal. The configuration here includes setting the data source and also defining which identity attributes (in addition to the ones defined in a system based on the X.500 ISO standard) are managed.

Figure 1.6 illustrates the basic UME structure. The *UME Persistence Manager* is responsible for communication between the UME and different data sources. It does not do this directly but instead uses suitable adapters for the relevant backend. Adapters therefore provide a very simple, flexible option to connect back-end systems from different manufacturers to the UME.

Principal API

At the core of the UME is the *principal API*. This programming interface gives the developer program-based access to identity information from the different connected systems.

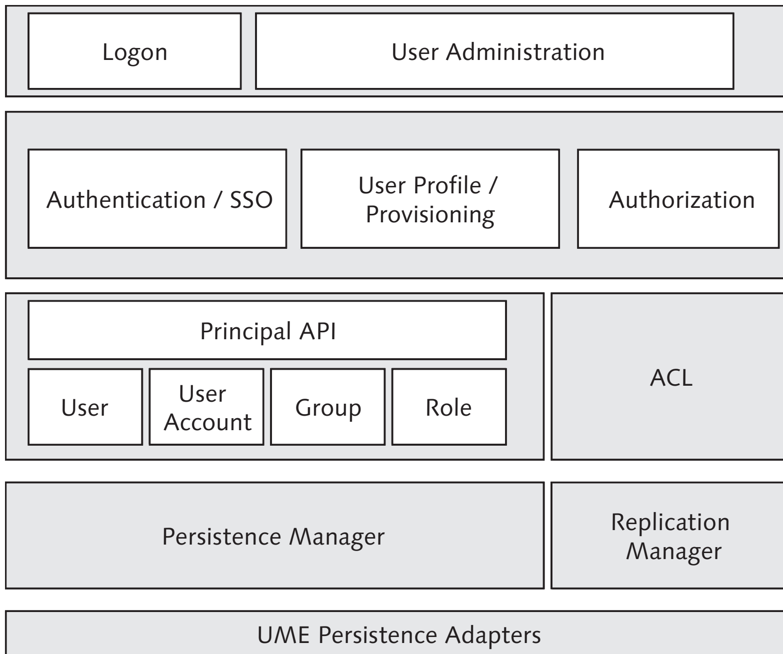



**Figure 1.6** User Management Engine

After you have logged on successfully to the Java stack, your identity is available as an `IUser` object. This interface contains all methods required for read access; however, write access to the identity through `IUser` is not possible. The `IUserMaint` interface (`Maint` stands for maintenance here) is available for write access. Separating the two access options was introduced for performance reasons. Write access would always result in a lock on the object, which would lead to the identity having to be locked in the backend every time a user was accessed. This restriction might be considered annoying, but write access is more of a rare case anyway. Write access on the *user datastore* is usually only used if users themselves can set up an account on the portal (*self-registration process*).

In addition to the principal API mentioned in the previous paragraph, the UME also provides interfaces for working with roles and groups. This means you can also easily access authorization structures for a user in your own applications and decide, based on these structures,

whether a certain user should get the full range of application functions or only a version adapted to his authorization structure. The UME fits perfectly into the security requirements of the J2EE specification from Sun Microsystems.

### 1.2.3 KeyStores: Authentication, Signatures, Encryption

In Section 1.2.1, Security in Heterogeneous Landscapes, we already discussed some of the security concepts, but we did not specifically illustrate how they are used. In this section, we will now describe how security is used and implemented.

**Providing key material**

As already mentioned, the main focus when using cryptographic functions should be on managing and saving key material securely. Java itself provides an option to store key material and certificates using *KeyStores*. KeyStores are a type of database for keys, and besides their own key material, they can also contain confidential certificates. Access to a KeyStore is password-protected.

KeyStore contents are created by *cryptography providers*. Cryptography providers offer flexibility in using different implementations of cryptographic functions. The basis for implementing these types of providers are the *Java Cryptography Architecture* (JCA) and its *Java Cryptography Extension* (JCE). Both specifications are described by Sun Microsystems and implemented by the relevant provider. When KeyStore files are accessed, all the providers that were used for creating the files must be known in the Java runtime context. A provider can either be registered at runtime by transferring a parameter or statically by using a configuration file. Each cryptography provider can define its own format for saving KeyStore contents. Two different types are provided in the Java standard for representing KeyStore file contents:

- *JKS* is the KeyStore type supported by the Sun provider.
- *PKCS12* defines a transfer syntax for PKCS#12 key material.

The PKCS12 standard describes an option as to how content can be processed independently of a provider.

**KeyStore tools**

There are two options available in the standard version for processing KeyStore files. You can use the `java.security.KeyStore` Java class for access from Java applications. For administrative purposes, the Java

installation provides the command line-based *keytool* that you can use to create and manage contents of a KeyStore file.

KeyStores and SAP KeyStores are naturally also used in the SAP Java stack. After the installation, a KeyStore file is created automatically the first time you start the Java stack. The file is called *verify.der* and contains the key material for the application server. This key material is used for creating the digital signature "under" the SAP logon ticket generated with each logon to the application server. The file is therefore the basis for managing trust between the SAP NetWeaver Java stack and other infrastructure components. If you now use the *keytool* on the *verify.der* file with the `-printcert` switch, you get the digital certificate information:

```
c:\keytool –printcert –file verify.der
```

**Displaying digital certificates**

The statement issued indicates the validity and issuer of the certificate. As mentioned, this information can be important for authentication and authorization, particularly when establishing trust. The public key of the certificate owner can also be used for encrypting and verifying digital signatures. It would be very awkward in this case if KeyStore files were stored unstructured on the server. SAP deals with this problem with a central service on the J2EE engine, the *J2EE KeyStore service*. The J2EE KeyStore service is used as a central service in the application server for saving and accessing all KeyStore files and their contents.

### J2EE KeyStore Service — Single Sign-On in J2EE Server

The J2EE KeyStore service is available in the *Visual Administrator* (VA) and *SAP NetWeaver Administrator*. Here we will show you how to manage KeyStore files with the Visual Administrator.

After you log on to the VA, navigate to the Key Storage service by selecting the Cluster • SID • Server • Services • Key Storage path. In this view, you have a very detailed overview of the *verify.der* file contents (Figure 1.7).

**KeyStore service in VA**

As already mentioned, this KeyStore file was created and registered in the KeyStore service the first time the J2EE engine was started. You can also use the KeyStore service to connect other systems to the J2EE server securely. To define other trust relationships, all you have to do is add the corresponding certificate using the Load button. Click Load, and expand the Files of Type dropdown menu to get an idea of the range of functions of the service.

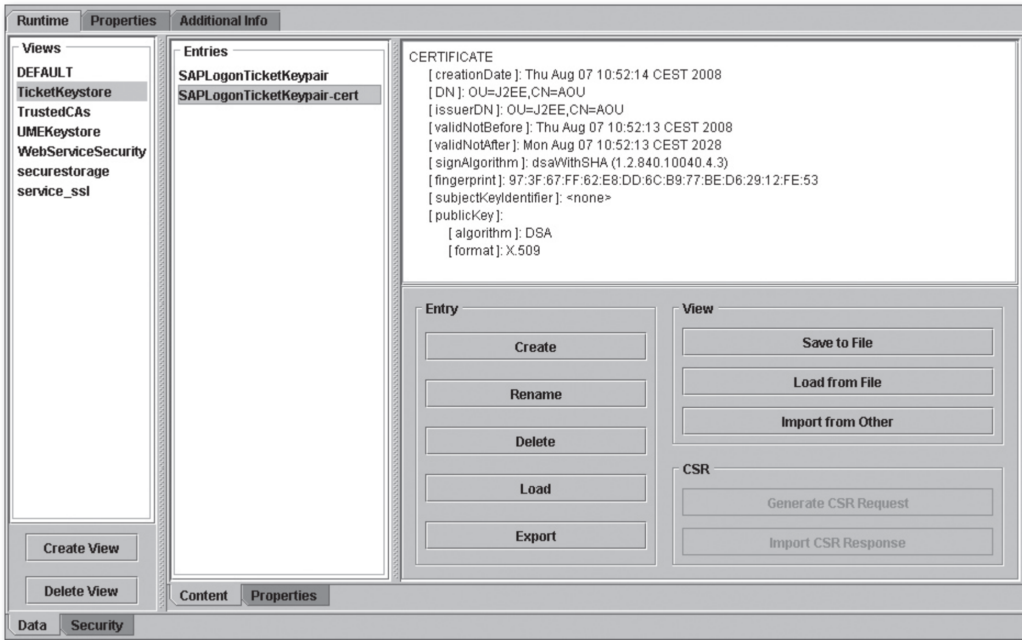



**Figure 1.7** KeyStore Service in Visual Administrator

Not only is it able to include certificates, it can also act as a key management tool for cryptographic keys. As you can see, the KeyStore service is a similar concept to the STRUSTSSO2 ABAP transaction. You use this transaction to manage keys and certificates on the ABAP stack.

#### Personal Security Environment

SAP uses *PSE files (Personal Security Environment* files) to store key information. Like KeyStore files, PSE files contain a list of trustworthy certificates in addition to their own key information. The *sapgenpse* command line-based tool is used for working simply with PSE files. Access to contents is password-protected. The *sapgenpse* tool is hugely similar to the one known from the Java Development Kit, the *keytool*, which you will learn about in the next section.

### 1.2.4 Authentication and Authorization

After having now discussed the basics of key management, in this next section we turn our attention to the aspects of authentication and autho-

rization we already addressed briefly in Section 1.2.1, Security in Heterogeneous Landscapes. We will discuss *Single Sign-On* (SSO) configuration between the SAP NetWeaver Portal and an ABAP system and configuring *Secure Network Communication* (SNC).

**Authentication**

Particularly, but not exclusively, in heterogeneous and shared integration scenarios, only those identities that have been explicitly granted access rights are allowed to have access to information. The accessing identity must be authorized for access. To verify the authorization of the accessing identity, the first step involves authenticating it. Authentication and downstream authorization are two mechanisms that should not be missing in any system and must occur for every access attempt.

**Multiple sign-on**

Authentication concepts range from querying a combination of the user name and password to checking biometric characteristics. Which concept is used for authentication will depend on the existing infrastructure or products. Most systems offer not just one type of authentication but rather let the user choose the option that can best be integrated into the infrastructure. However, one problem persists with this range of options: with a heterogeneous, complex landscape, there is a risk that the user will be required to authenticate himself several times a day (whenever he wants to access another system). If the user also has different user IDs and passwords for the different systems, it is extremely difficult, or only possible by making notes of the access data, to keep track of everything.

**Single sign-on to simplify the login procedure**

Using *single sign-on* can prevent you from having to log on repeatedly when switching systems. SSO solutions help, in that users only have to log on to one system, preferably the operating system when the system is started. If a user then switches systems in between, the central system delegates his identity. The user saves time by establishing a “relationship of trust” between the different systems.

**Disadvantages of SSO solutions**

Unfortunately, there are also disadvantages to SSO solutions. The first disadvantage related purely to the infrastructure is that user identities must be homogenized. User *mmayer* must have the same user ID on all systems, therefore, *mmayer* in this example. A second disadvantage is that this individual access permission is a security risk. If a potential attacker is in possession of the single access ID, all doors to the system are open to him. To sum up, it is therefore safe to say that a high level of security

can only be achieved by sensibly coupling different security functions. This also includes decisions about where and how SSO is used.

#### Single Sign-On with SAP NetWeaver Portal

We will take a closer look at an SSO scenario now. Let's assume a user wants to access an SAP NetWeaver AS ABAP from an SAP NetWeaver Portal. For this access, the user simply has to authenticate himself on the portal. Every other access from the portal does not require explicit user authentication. For this to be possible, a system first has to be declared as a *ticket issuer*. The ticket issuer is the infrastructure instance responsible for issuing SAP logon tickets. It is therefore also the root of a relationship of trust. In the scenario discussed here, the issuer is almost always the portal. The reason for this is that a *single-point-of-access strategy* is chosen with the portal integration. All applications are exclusively called through the portal for this type of strategy. An explicit call through the SAP GUI, for example, would not be welcome here. The ticket issuer signs all created logon tickets digitally and stores them as non-persistent cookies in the relevant browser session. The digital signature is verified against an external system for authentication. All systems addressed from the portal in this case need a relationship of trust to the ticket issuer. As already mentioned, a key pair consisting of a public and private key is required to create a digital signature. The key pair for the portal is automatically created when the portal server is started for the first time after the installation and then the private key for creating the digital signature is used "under" the SAP logon ticket.

**Transferring the SAP logon ticket**

If the user from the portal now calls a function module from the ABAP system, the SAP logon ticket is transferred to the ABAP system for each call. Unfortunately, it is not enough just to transfer the ticket — the ABAP system naturally also has to verify whether the user has system authorization, whether there is sufficient trust in the ticket, and lastly, whether the accessing system is authorized to access the client. This requires some configuration steps:

- Exporting the digital certificate of the portal server
- Configuring profile parameters to accept logon tickets
- Adding an ACL entry (*Access Control List*) to the ABAP stack for the portal server

- Importing the digital certificate of the portal server using Transaction STRUSTSSO2

We describe the configuration steps in detail in the next section.

#### Exporting the Certificate

The certificate is exported directly into SAP NetWeaver Portal. You must log on as a user with administrator rights for this purpose. Then navigate to the management view for the keys by selecting SYSTEM ADMINISTRATION • SYSTEM CONFIGURATION • KEYSTORE ADMINISTRATION. Choose the SAPLogonTicketKeypair-cert entry from the dropdown list here. Then click the Download verify.der File button, and save this on your hard drive. The file is saved as a ZIP file. Unpack the ZIP file.

#### Adding an ACL Entry to the ABAP Stack for the Portal Server

After you have exported the certificate, you can begin configuring the ABAP stack. To enable the portal server to be recognized as a system allowed to communicate with the ABAP stack, an ACL entry must be created for it. You make the ACL entry in the `TWPSSO2ACL` table. Start Transaction SM30 for the configuration, and enter the table name in the search field. Then add a new entry to the table with the following information:

ACL maintenance using Transaction SM30

- Portal system ID
- Client to be accessed
- Subject name
- Name of issuer of the certificates (SID of server containing the SAP logon tickets)

You can read this information from the KeyStore administration of your portal. To do this, select the ROLE SYSTEM ADMINISTRATION • SYSTEM CONFIGURATION • KEYSTORE ADMINISTRATION path. Then choose the SAPLogTicketKeypair-cert entry from the dropdown list.

#### Configuring Profile Parameters

To ensure the ABAP system can also accept and process SAP logon tickets, you must set two profile parameters. Start Transaction RZ10 to do this. If you execute the transaction for the first time, no profiles exist yet.

To change this, choose the Import Profiles of Active Servers option from the Utilities menu. The parameters are set in the instance profile of the server. Set the value of the `login/accept_sso2_ticket` parameter to 1. This parameter specifies that SAP logon tickets are accepted for authentication. Also set the value of the `login/create_sso2_ticket` parameter to 0. With this parameter, you signal to the server that it can also act as a ticket issuer.

After you have set both parameters, you must start the server again. You can then test the configuration in Transaction SSO2. To do this, you set the value in the RFC Destination field to `NONE`, and then press [F8]. The result should correspond to the screen shown in Figure 1.8.

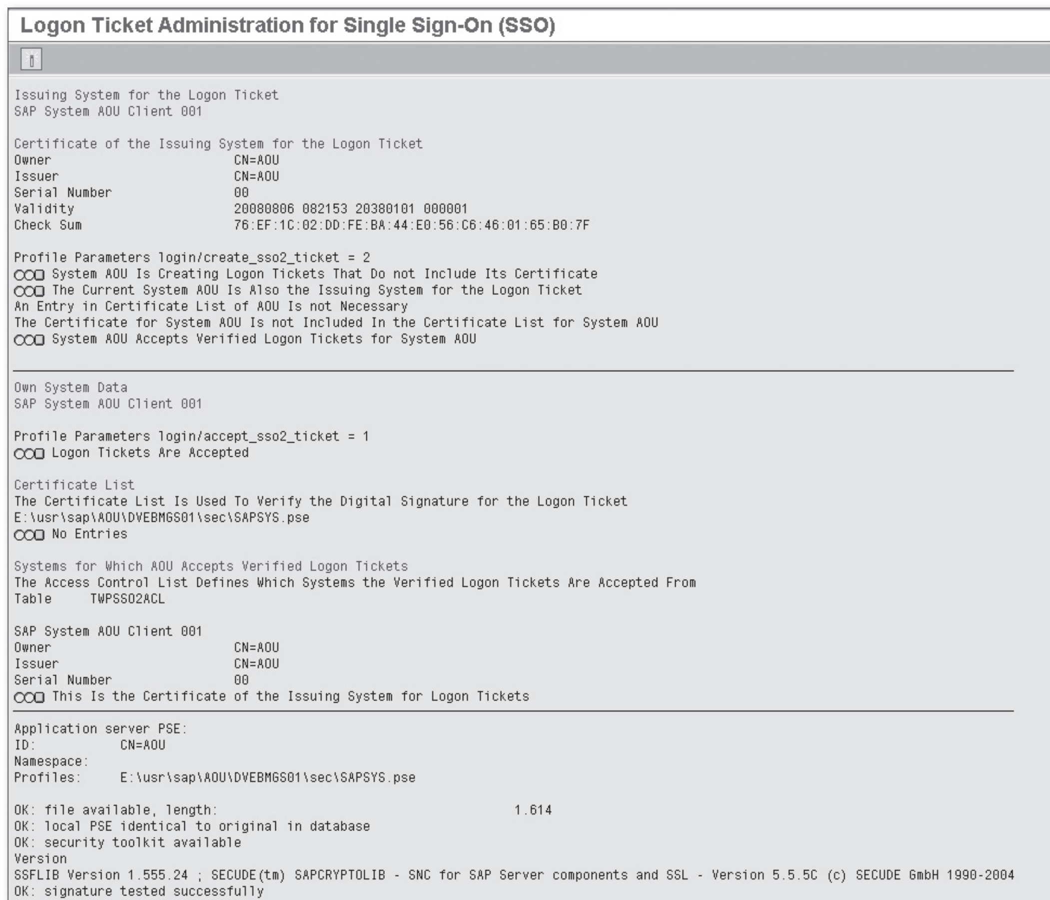



**Figure 1.8** Executing Transaction SSO2 with a "NONE" Destination

### Importing the Certificate in the SAP System

You conclude the configuration by importing the portal certificate into the ABAP system and defining a relationship of trust between the systems. To do this, start Transaction STRUSTSS02. When you open the *System PSE* folder, you will see the dialog box displayed in Figure 1.9. Click the Import certificate button to start the dialog box for uploading the exported portal certificate. Choose Binary as the file format here. When you confirm the dialog box, you are directed back to the initial screen.

**Figure 1.9** ABAP System KeyStore

The dialog box now shows an owner, issuer, and other information about the certificate. Click the Add to Certificate List button to import the portal certificate into the certificate list.

Homogenizing user IDs

Using the logon ticket requires user IDs to be homogenized in all systems. In some cases, this may cause a considerable increase in the time and effort required to homogenize IDs. Another option in the context of the portal is to use user mapping: This involves connecting users within the different systems with each other through the UME. Similar to homogenizing user IDs, user mapping can result in a great deal of time and effort being required for very large systems. The issue of how authentication can be controlled across several systems through mapping has to be clarified. A process as to how users in the company obtain access to a corresponding system must also be defined.

### Integrating External Systems Securely

After having discussed pure SAP integration, we will now look at the options for using systems from an external SAP landscape securely. The question that often arises when integrating these types of systems is how to perform the authentication. There are many different techniques ranging from pure configuration to programming authentication mechanisms.

Programming an authentication solution

Let's take a closer look at programming an authentication solution. We will assume that we are accessing an SAP Java stack application addressed on a J2EE application server from another manufacturer. This means that proprietary communication protocols are used throughout and direct communication is not possible. We will mainly focus on the question of how an SAP logon ticket can be validated. The actual application integration is irrelevant here.

We can use the SAP logon ticket very easily for authenticating a user in the infrastructure just described. After the user has logged on to the J2EE server, the ticket is created and can be used from now on for authentication. The challenge is simply to verify the ticket in an external application.

Verification libraries

SAP provides two different libraries for verifying logon tickets. The *SAP Crypto Toolkit* is an API based purely on Java for using cryptographic functions within the SAP landscape. This also includes working with

the SAP logon ticket. The SAPSSOEXT verification library provides similar functions to the SAP Crypto Toolkit, but you can use this API both from Java per JNI (*Java Native Interface*) and from C or C++. It is available for different platforms.

Which of the verification methods is used will ultimately depend on the technological platform. In mixed landscapes (with Java and ABAP and possibly other external programs), it is certainly wise to revert to the SAPSSOEXT library because then the same implementation of all components can be used. In pure Java landscapes, it is easier to use only the SAP Crypto Toolkit.

### Step 1: Setting Up the Class Path

To begin with, download the toolkit from the SAP Service Marketplace. You will find it under the DOWNLOAD • SAP CRYPTOGRAPHIC SOFTWARE path. After you have unpacked the CAR archive, you will find a file with the extension *.sda* there. The extension stands for *Software Deployment Archive*, and Java applications are deployed on the SAP J2EE server using these types of files. Unpack the file, and you will find a number of Java archives (JAR). The central Java archive is called *iaik_jce.jar*. IAIK here stands for *Institute for Applied Information Processing and Communication*. One of the many types of work performed by this institute of the Graz University of Technology in Austria includes implementing Java-based cryptographic functions collectively referred to as *cryptography providers*. SAP uses these cryptography providers in its Java-based solutions. As well as the cryptography provider, you still need three other JAR files, which you will find under the following paths:

- *<drive>:\usr\sap\<SID>\j2ee\cluster\server0\bin\ext\com.sap.security. api.sda\com.sap.security.api.jar*
- *<drive>:\usr\sap\<SID>\j2ee\cluster\server0\bin\ext\com.sap.security.api.sda\com.sap.security.core.sda\com.sap.security.core.jar*
- *<drive>:\usr\sap\<SID>\j2ee\cluster\server0\bin\system\com.sap. security.api.sda\logging.jar*

Add these JAR files to the class path of your development environment. You can then continue with establishing the relationship of trust.

### Step 2: Establishing a Relationship of Trust Using a KeyStore File

**Integrating certificates**

Like the ABAP infrastructure, the J2EE server certificate must be integrated into an external environment before the ticket can be verified. The most secure way to integrate certificates is to use KeyStores. As already mentioned, you can manage KeyStores using the `keytool` command-line tool. You can use the `-alias` parameter to find and address entries more easily in a KeyStore, and you can use `-storepass` to specify for the keytool call the password for protecting the KeyStore. The parameters in the following `keytool` call make it almost self-explanatory:

```
keytool -import -alias portal -file x:\verify.der
-keystore c:\demokeystore.pse -storepass mypassword
```

The certificate for the *verify.der* file is included in the *demokeystore.pse* KeyStore file. You can also see that not only the `sapgenpse` tool but also the `keytool` can process PSE files.

After you have imported the certificate into the KeyStore and as a result it is available for verifying logon tickets, you can begin implementing the ticket verification. This implementation basically consists of two steps: reading trustworthy certificates from the PSE file and verifying the SAP logon ticket.

### Step 3: Implementing Java Functions

**Reading trustworthy certificates**

Listing 1.2 is a method for importing a PSE file. The call parameters the `loadCertsFromPSE` method receives include the full file path for the PSE file and the password used for accessing the file securely. As return parameters, the method delivers a list of trustworthy certificates that were read from the PSE file.

```
public stat-
ic X509Certificate[] loadCertsFromPSE(String pse, String pwd) {
  java.security.cert.X509Certificate[] certificates = null;
  char passwd[] = pwd.toCharArray();
  InputStream stream = null;
  ArrayList certs = new ArrayList();
  try {
    stream = new FileInputStream(pse);
    java.security.KeyStore store =
     java.security.KeyStore.getInstance("JKS", "SUN");
    store.load(stream, passwd);
```

```
    Enumeration enu = store.aliases();
    while (enu.hasMoreElements()) {
      String alias = (String) enu.nextElement();
      if (store.isCertificateEntry(alias)) {
        certs.add(store.getCertificate(alias));
      }
    }
    stream.close();
    if (certs.size() < 1) {
      System.out.println(
        "PSE does not contain any certificates");
    }
    certificates =
      (java.security.cert.X509Certificate[]) certs.toArray(
       new java.security.cert.X509Certificate[0]);
    return certificates;
  } catch (Exception e) {
  }
}
```

**Listing 1.2** Importing and Exporting X509 Certificates

**Importing a PSE file**

Because the trustworthy certificates are stored within the PSE file, you must first import the PSE file. You do this by creating a `FileInputStream`. As already mentioned, PSE files are an SAP interpretation of the KeyStore concept, so it is little wonder that we can also use the `KeyStore` class for processing PSE files. You create the KeyStore instance using the `getInstance` factory method. The call parameter this KeyStore instance gets specifies which type of KeyStore representation is to be created. In our case, we will use the JKS standard implementation from Sun (*Java Key Store*). The last step for creating the KeyStore object involves calling the `load` method. It initializes the newly created KeyStore object with information from the imported file. By specifying an alias, you can then access the content and, as a result, the certificates.

**Validating the ticket**

You will remember that the logon ticket is stored as a cookie on the client. If you now call a function in an external Java-based system, the SAP logon ticket will be sent to the system as a string. The `getSSOTicket` method shown in Listing 1.3 receives the SAP logon ticket as a string and must be decoded for further processing. The `verify` method of the `Ticket` class performs the actual ticket verification. We create a class

instance for this and initialize it using the list of known certificates and the decoded ticket.

```
private Ticket getSSOTicket(String encodedTicket)
  throws Exception {
  try {
    String decodedTicket = URLDecoder.decode(
       encodedTicket, "UTF-8");
    decodedTicket = decodedTicket.replace(' ', '+');
    X509Certificate[] certificates =
      loadCertsFromPSE(keystorePath, "password");
    IAIK provider = new IAIK();
    Security.addProvider(provider);
    com.sap.security.core.ticket.imp.Ticket ticket =
      new com.sap.security.core.ticket.imp.Ticket();
    ticket.setCertificates(certificates);
    ticket.setTicket(decodedTicket);
    ticket.verify();
    return ticket;
  } catch (UnsupportedEncodingException uncodingEx) {
    // Process exception sensibly
  } catch (TicketVerifyException tvEx) {
    // Process exception sensibly
  }
  return null;
}
```

**Listing 1.3** Creating and Verifying the SAP Logon Ticket

### Prospects for Using the SAP Logon Ticket in Other J2EE Application Servers

JAAS

Creating and forwarding the ticket to the SAP system are recurring tasks, and using the SAP logon ticket in other J2EE servers is immensely important. The J2EE standard allows for this requirement in its specification by describing the authentication and authorization in a separate specification. The *Java Authentication and Authorization Service* (JAAS) describes how separate authentication and authorization concepts can be integrated into the J2EE server.

Pluggable authentication model

For this purpose, JAAS implements the *Pluggable Authentication Model* (PAM) usual for UNIX. Authentication and authorization are not hard coded in the application in this case but rather are provided through flexibly exchangeable modules. All JAAS modules have a standard interface.

This and the responsibilities of the modules are defined in the specification. You can therefore use JAAS to integrate your own concepts into the SAP NetWeaver AS and also implement authentication per SAP logon ticket into another J2EE server. You create authentication components using login modules. In increasingly greater service orientation, you can therefore use uniform security concepts across system boundaries so that communication between the portal and ABAP system and from there to a non-SAP J2EE server can be achieved. With Java-to-Java communication, for which the SAP logon ticket is used for authentication, the ticket verification could be implemented in login modules.

### Security on the Transport Layer

The last sections dealt almost exclusively with authentication and authorization. These are two important aspects, but protected data access is not much use if their exchange occurs in "plain text." An attacker can then "eavesdrop" very easily on the network to access confidential information.

**Transport security with SNC or SSL**

We will therefore now cover using security on the transport layer. This will encompass the authenticity of communication partners, data integrity, and data confidentiality. Two different security protocols are supported for securing communication between SAP NetWeaver components:

- *Secure Socket Layer* (SSL) for securing HTTP
- *Secure Network Communication* (SNC) for securing RFC and DIAG

SSL and SNC both work on the basis of certificates. In the following section, we will look in more detail at using SNC and explain how its infrastructure is set up.

### Introduction to Secure Network Communication

SNC is used for securing communication between two SAP NetWeaver communication partners. SNC actually only describes a layer in the SAP NetWeaver system architecture, which describes an interface for integrating external security products. To put it more specifically, the interface between the SAP system kernel layer and a library (implementation) of an external manufacturer is described via SNC. Library manufacturers must implement GSS-API V2 (*Generic Security Services Application Pro-*

*gramming Interface Version 2*) in this case. It is loaded dynamically during the SNC initialization phase. SAP provides its own implementation of this type of external library for using SNC: SAP Cryptolib. The SNC approach is very much similar to the previously mentioned PAM concept of the Java Authentication and Authorization Service (JAAS). As already mentioned, SNC addresses three areas of security:

Levels of security with SNC

- Authentication only
- Protection of integrity
- Protection of data through encryption

The SNC user can therefore achieve security at several levels. When authentication is used, the system merely secures the identity of the communication partner. However, to secure communication during transit between the communication partners, it is essential to use data integrity and data protection through encryption. This is configured using profile parameters in the instance profile of the application server.

As already mentioned, SNC works with certificates. The one communication partner must have the digital certificate of the other communication partner and extend him a certain amount of trust. The key information here must be in a form that is difficult to abuse. The *Personal Security Environment* (PSE) is used in saving key information. Besides a certificate list of those certificates that are trusted, PSE files also contain their own key information.

Strength of trust scenarios

The strength of trust is described within SNC using two scenarios:

- **All communication partners that communicate with the ABAP system via SNC use the same PSE.**
  The advantage of this scenario is that management in the ABAP stack is much easier. The time and effort required before secure communication can be achieved is also significantly lower. This is because a new PSE does not have to be created for every communication partner, and the certificate of the partner therefore does not have to be imported into the ABAP stack.

  However, the disadvantage of this scenario cannot be ignored: An attacker may get possession of the PSE file, and the time and effort required to close any security loops — depending on the number of communication partners — could turn out to be enormous. Also bear

in mind that localizing communication problems for a partner is really complicated because it is very difficult to find out which SNC connection is faulty.

- **Every communication partner gets an individual PSE.**
  The big advantage of scenario 1 becomes a big problem here: The initial administration effort is greater and spontaneous communication therefore impossible. Before communication can take place, both partners must export their digital certificate from the PSE and import the certificate of the other.

**Combinations**

Combinations of both scenarios are also possible by forming several groups of systems and by each system group using the same PSE.

### Installing SAP Cryptolib

Before you can start configuring the SNC, you need SAP Cryptolib for the SAP NetWeaver AS ABAP. Follow the DOWNLOAD • SAP CRYPTOGRAPHIC SOFTWARE path under *http://www.service.sap.com/download* to download the library. You will find the libraries for all SAP-supported platforms as CAR files under this address.

Because the examples in this book were developed on Windows 2003, we have decided to use the Windows platform. In the file, you will find three folders for the relevant instance of a Windows platform. Each of these folders contains the *sapcrypto.dll* file and a version of the *sapgenpse.exe* tool. In addition to a number of text files, you will also find a file called *ticket*. This file provides the license certificate for creating your own certificates.

**Installation path**

To install SAP Cryptolib, simply copy the files into the corresponding application server directories, *exe* and *sec*. The `DIR_INSTANCE` configuration parameter is important for the storage location of the files. It points to *<drive>\usr\sap\<SID>\<instance>* on Windows. The parameter is prefixed to all profile parameters to specify the installation directory of the server instance.

Copy the *sapcrypto.dll* and *sapgenpse.exe* files into the directory specified in the `DIR_EXECUTABLE` profile parameter. On Windows, the parameter normally points to *DIR_INSTANCE\exe*.

Then store the file with the ticket in the *DIR_INSTANCE\sec subdirectory.* If there is already a file called ticket in this directory, create a subdirectory and store the file there.

**SECUDIR environment variable**

To complete the installation, set the `SECUDIR` environment variable of the operating system in the *sec* subdirectory where the ticket is stored — in other words, in *DIR_INSTANCE\sec* or in the subdirectory you created. You must bear in mind here that the environment variable is set under the user, under which the application server is to be executed later. By setting variables, you let the runtime environment know where to find trustworthy tickets.

### Configuring the Application Server for Using SNC

Now that we have installed SAP Cryptolib, we will turn to configuring SNC. In the first part of the configuration, you must notify the application server of the required information about the process parameters for SNC. You do this using the instance profile. Start Transaction RZ10 for this. Then open the instance profile in Extended Maintenance mode. Press [Shift] + [F8] to switch to edit mode, and start adding the parameters. You can do this either by clicking the Parameter button or pressing the [F5] key. Table 1.3 shows the parameters to be set and their descriptions.

| Parameter Name | Value | Description |
|---|---|---|
| `snc/enable` | »1« | Activates the use of SNC |
| `snc/gssapi_lib` | $(DIR_EXECUTABLE)\ sapcrypto.dll | Refers to the GSS-API implementation |
| `snc/identity/as` | p:CN=AOU, OU=IT, O=ADO, C=DE | Unique application server identifier |
| `snc/data_ protection/max` | »3« | Maximum security support |
| `snc/data_ protection/min` | »1« | Minimum security support |

**Table 1.3** Overview of SNC Profile Parameters and Their Descriptions

| Parameter Name | Value | Description |
|---|---|---|
| snc/data_ protection/use | »3« | Security support to be used |
| snc/accept_ insecure_cpic | »1« | Access without SNC per CPIC connection allowed |
| snc/accept_ insecure_gui | »1« | 1 = access without SNC allowed per SAP GUI |
| snc/accept_ insecure_r3int_rfc | »1« | 1 = SAP system-internal access without SNC allowed |
| snc/accept_ insecure_rfc | »1« | 1 = access via RFC also without SNC allowed |
| sec/libsapsecu | $(DIR_EXECUTABLE)\ sapcrypto.dll | File name of SAP cryptographic API to be loaded when server started |
| ssf/ssfapi_lib | $(DIR_EXECUTABLE)\ sapcrypto.dll | File name of SAP cryptographic library |
| ssf/name | SAPSECULIB | Specifies environment variables of operating system |

**Table 1.3** Overview of SNC Profile Parameters and Their Descriptions (Cont.)

As you can see from the Description field in the table, the parameters are set in such a way that unsecure communication with the ABAP system is still possible. To complete the configuration, set the value of the `snc/enable` parameter to 1. You should then restart the server.

**Check the Installation and Settings Thoroughly**

Make sure you only set the value of the `snc/enable` parameter to 1 after you are sure the SAP Cryptolib installation is complete and all parameters in Table 1.3 have been set properly. If the library has been installed and configured incorrectly, the application server will no longer start.

### Creating the SNC PSE Container

After the server has uploaded successfully, you can begin completing the configuration in the server. Because SNC is based on public-key cryptography using digital certificates, each communication partner must produce a trustworthy certificate.

Start Transaction STRUSTSSO2 to configure a relationships of trust in the ABAP system. After you start the transaction, you will see an entry highlighted in red on the left of the screen in your system: SNC (SAP Cryptolib). Select the Create menu option from the context menu by clicking the entry. In the window that appears, specify the same value as the one in the `scn/identity/as` profile parameter. This will create a PSE file for your application server. After you have confirmed your entries, you will be requested to create a password for the SNC PSE entry just created. Click the Password button (Figure 1.10) to create a password that will be used to protect access to the PSE file.

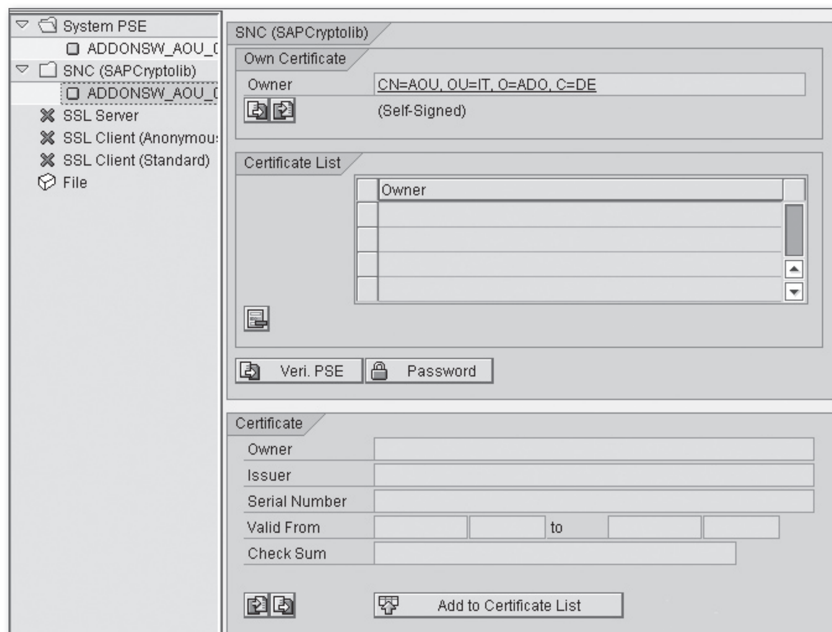



**Figure 1.10** Transaction STRUSTSSO2 with SNC PSE File

### Testing the Configuration

Unfortunately, there is no integrated way to test whether the SNC environment was installed and configured successfully. You would have to wait until the SNC environment was used from an RFC client. But if an error then occurs, the worst-case scenario is that you will not know what has caused the error: the configuration, the client itself or the client certificate.

However, SAP Notes 800240 and 912405 provide you with a program you can copy into your ABAP system using Transaction SE80. The program is called `ZSSF_TEST_PSE`.

**Test program on the SAP Service Marketplace**

## 1.3 Programming SAP NetWeaver AS ABAP

Next, we will develop a small application in ABAP that we will repeatedly use as an example in Chapter 2, Remote Function Call with ABAP, up to Chapter 7, SAP NetWeaver Process Integration. Our sample application will manage sales orders and vendor purchase orders for a book wholesaler. We will supply this application with inbound interfaces (for book store sales orders) and outbound interfaces (for vendor orders to publishing companies) in different interface technologies:

- *Remote Function Call* (RFC) in Section 2.1, RFC Function Modules in ABAP
- *Business Application Programming Interface* (BAPI) in Section 2.4, Business Objects and BAPIs
- *Application Link Enabling* (ALE) in Chapter 5, IDocs and ALE
- *Service-Oriented Architecture Protocol* (SOAP) in Chapter 6, Service-Oriented Architecture Protocol
- *XI Protocol* (SAP Exchange Infrastructure) in Chapter 7, SAP NetWeaver Process Integration

In Section 1.4, Overview of SAP Interface Technologies, we will take a closer look at the interface technologies listed here and explain them in the previously mentioned chapters using programmed examples. We will also program the corresponding outbound interfaces for book stores and inbound interfaces for publishing companies in C, Java, and C# programming languages. Figure 1.11 shows the systems involved.

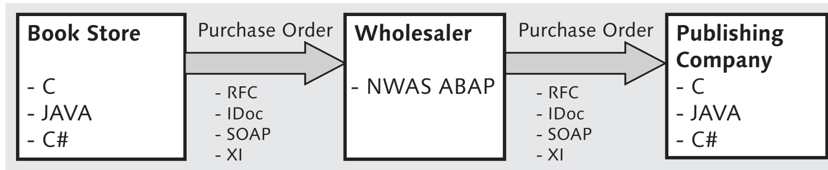



**Figure 1.11** Sample Application

### 1.3.1 ABAP Dictionary

The application consists of two database tables with purchase order header data and item data:

- `ZIFPORDER`
  This table contains purchase order information.
- `ZIFPORDERPOS`
  This table contains information about purchase order items.

Sales orders and vendor purchase orders will be stored in the same tables in this case and will differ from each other by the content of the `DOCTYPE` field. This field contains the value SO for *sales order* entries and PO for *purchase order* entries.

**Packages**

We will assign all repository objects to the `ZIFP` package we will create using ABAP Workbench SE80. To do this, start Transaction SE80, and choose the Package entry from the dropdown list. Specify "ZIFP" as the name of the package, and click the button with the glasses icon to display the objects assigned to this package. Because the package does not yet exist, you are offered the chance to create it. Also create a new transport request at the same time for the new repository objects to be created.

#### Database Tables

After you have created the package, you can define the two transparent tables either directly from the Object Navigator or using Transaction SE11.

The fields for ZIFPORDER, their data elements (business data types), lengths, and descriptions are listed in Table 1.4, while the details about ZIFPORDERPOS are listed in Table 1.5.

Fields of sample tables

| Field | Data Element | Data Type | Length | Description |
|---|---|---|---|---|
| MANDT | MANDT | CLNT | 3 | Client |
| ORDERID | ZIFPORDERID | NUMC | 10 | Purchase order number |
| TYPE | ZIFPORDERTPYE | CHAR | 2 | Purchase order type |
| REFID | ZIFPREFID | NUMC | 10 | Reference number |
| BUYER | ZIFPBUYER | CHAR | 50 | Buyer |
| SELLER | ZIFPSELLER | CHAR | 50 | Vendor |
| DATE | ZIFPORDERDATE | DATS | 8 | Purchase order date |

**Table 1.4** ZIFPORDER Table Columns

| Field | Data Element | Data Type | Length | Description |
|---|---|---|---|---|
| MANDT | MANDT | CLNT | 3 | Client |
| ORDERID | ZIFPORDERID | NUMC | 10 | Purchase order number |
| ORDERPOS | ZIFPPOS | NUMC | 5 | Purchase order item |
| MATID | ZIFPMATID | CHAR | 20 | Material number |
| MATTEXT | ZIFPMATTEXT | CHAR | 40 | Material text |
| ORDERCOUNT | ZIFPCOUNT | INT4 | – | Number |
| PRICE | ZIFPPRICE | CURR | 8.4 | Price |
| CURRENCY | ZIFPCURRENCY | CUKY | 5 | Currency |

**Table 1.5** ZIFPORDERPOS Table Columns

Figure 1.12 shows the definition of the ZIFPORDERPOS table in the *ABAP Dictionary* (DDIC).

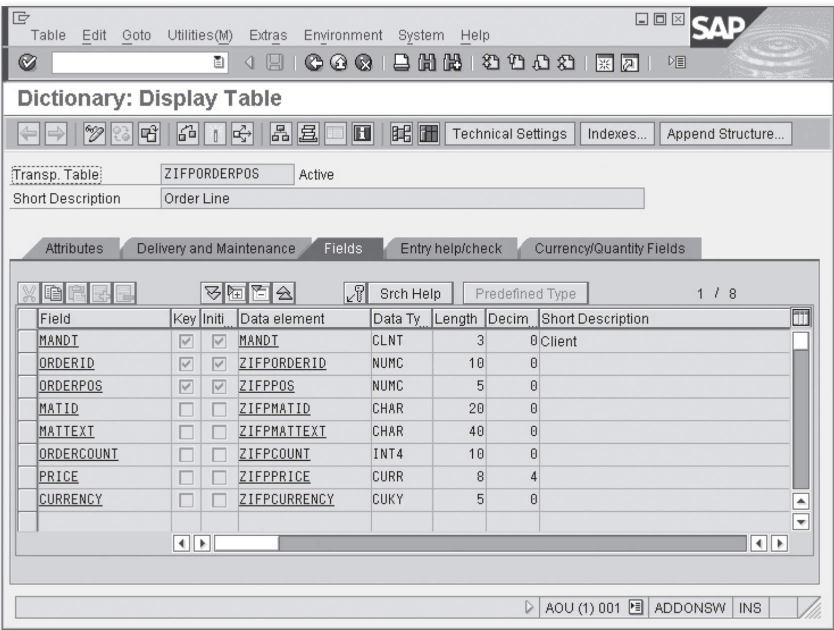



**Figure 1.12** ZIFPORDERPOS Table in ABAP Dictionary

**Foreign key**

The key fields of the ZIFPORDER table are MANDT and ORDERID. In contrast, the key fields of the ZIFPORDERPOS table are MANDT, ORDERID, and ORDERPOS. The MANDT field in both tables is a check field with the T000 check table that contains the clients. The ORDERID field of the ZIFPORDERPOS table is a check field with the ZIFPORDER check table. The foreign key in the ZIFPORDERPOS table, which uniquely defines an entry from the ZIFPORDER table, is made up of the ZIFPORDER-MANDT and ZIFORDER-ORDERID fields. The system can use these foreign key dependencies later to run automatic input checks on dynpros that use the check fields.

**Table maintenance**

When defining tables, you can select the relevant option from a dropdown list on the Delivery and Maintenance tab to specify whether table maintenance is allowed. Specify that this should be allowed, so that you will be able to maintain table contents using Transactions SE16 and SA16.

In the technical properties, which you can access by selecting the GOTO • TECHNICAL SETTINGS menu path, specify that both tables contain master and transaction data. Also set the size category in this dialog box to 0. Both tables contain transaction data, so they should not be buffered.

The corresponding database tables are created as soon as you activate the table definitions. You can now enter the first test data into the tables using Transaction SE16 or SA16.

### Search Help

Because we will need the `ZIFPORDER` table when discussing the `Helpvalues` business object in Section 2.4.3, Helpvalues Business Object, we will create two elementary search help options and one collective search help for it:

- **ZIFPORDERA**
  This elementary search help is used for searching for purchase orders by the name of the buyer.
- **ZIFPORDERB**
  This elementary search help is used for searching for purchase orders by the name of the seller.
- **ZIFPORDER**
  This collective search help includes the `ZIFPORDERA` and `ZIFPORDERB` elementary search help options.

**Collective search help**

You create search help using the ABAP Dictionary (Transaction SE11). Search help is used to make it easier to design input help (F4 help) for displaying valid input values.

**Search help for purchase orders**

The `ZIFPORDERA` search help shown in Figure 1.13 obtains its data from the `ZIFPORDER` database table. This is the *selection method* of the search help that includes the four `ORDERID`, `TYPE`, `BUYER`, and `SELLER` search help parameters. These four parameters are shown on the search help hit list (in the LPos column in Figure 1.13). The first three of these parameters, `ORDERID`, `TYPE`, and `BUYER`, are displayed on the search help selection screen (in the SPos column in Figure 1.13). The search help selection screen is a user dialog where the user can specify selection criteria to limit the number of hits. If you want the search help to list only sales orders, enter the "SO" value into the Type selection field in the selection screen.

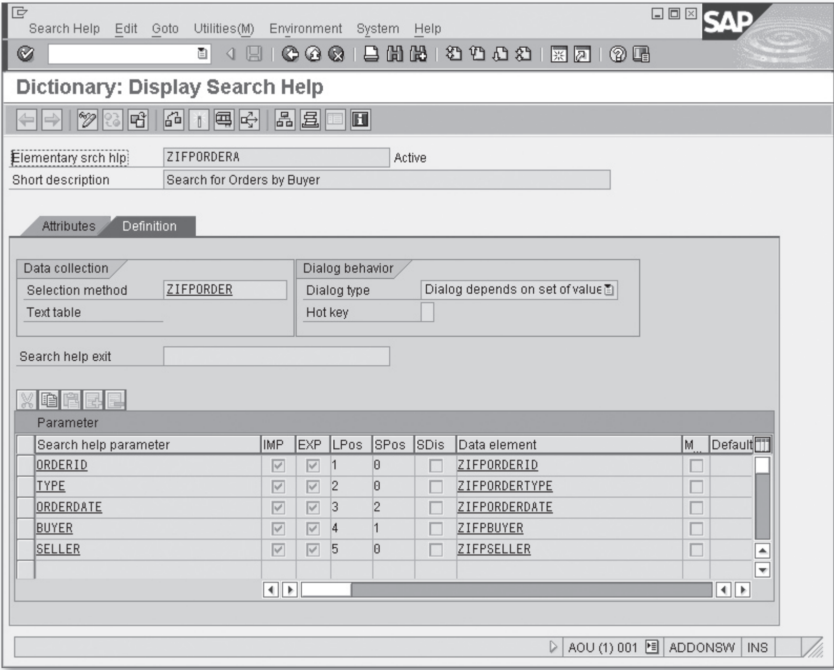



**Figure 1.13** ZIFPORDERA Elementary Search Help

**Attaching search help**

You can attach search help to check tables, table fields, or data elements, although it is most often attached to check tables. When you request input help (F4 input help) on a check field with a check table, the search help attached to it is used. In our example, we will attach the `ZIFPORDER` search help to the `ZIFPORDER` check table. When you then search for a valid purchase order number in a check field of the `ZIFPORDER` check table, the system uses this search help.

The two elementary search help options, `ZIFPORDERA` and `ZIFPORDERB`, are included in the `ZIFPORDER` collective search help. This collective search help is attached to the `ZIFPORDER` check table and also to the `ORDERID` field of this table.

### Lock Objects

When developing an ABAP application, you must make sure that many users can work with an SAP system simultaneously, so before you delete or change any data records, you must determine whether another user

is currently working with the data record. First of all, you set an exclusive lock on the corresponding data record. If the lock cannot be set successfully because the data record is already being processed by another user, we must cancel the processing. If the lock can be set, we are free to delete or change the data record.

**Developing lock objects**

To be able to set locks, you develop a *lock object* in Transaction SE11 and an `EZIFPORDER` lock object with the `ORDERID` and `ORDERPOS` fields to enable you to lock all the items for a purchase order. The lock object relates to the primary `ZIFPORDER` table and secondary `ZIFPORDERPOS` table. Specify both of these tables when creating the lock object in Transaction SE11.

**Lock modules**

The two function modules, `ENQUEUE_EZIFPORDER` and `DEQUEUE_EZIFPORDER`, are generated when you activate the lock object. You can use them to set and remove locks. These two function modules have the `ORDERID` and `ORDERPOS` import parameters that you can use to uniquely determine from the `ZIFPORDERPOS` table, the data record to be locked. All items for a purchase order may also be locked generically, whereby nothing is transferred for the `ORDERID` lock argument. The remaining `ENQUEUE_EZBOOK` import parameters control the lock mechanism details but are prepopulated with default values sufficient for our purposes.

### 1.3.2 Authentication and Authorization

When logging on via the SAP GUI, a user is authenticated by specifying a user name, password, and client. These three details must match the data assigned when the user master data record is created in Transaction SU01. However, the application itself must establish whether an authenticated user is also authorized to perform certain actions. To do this, it uses the `AUTHORITY-CHECK` statement to compare whether authorizations with specific values defined by the application are stored in the profiles for the user.

**Authorization objects**

We will create the `ZIFPORDER` *authorization object* with the `ACTVT` field (for "Activity") for this in the Object Navigator (SE80) or Transaction SU21. The values the `ACTVT` field can have are 01 (Add or Create), 02 (Change), 03 (Display), and 06 (Delete).

To give a user authorization to display purchase orders, the administrator must create *authorization* for this authorization object in Transaction SU03 and set the value of the `ACTVT` field for this authorization to 03. In

Transaction SU02, the administrator then assigns this authorization to a profile that he subsequently assigns in turn to the corresponding users in Transaction SU01.

**Authorization check**

Before an application displays purchases orders for the user, it must check the user's profiles to verify whether the user actually has authorization for the ZIFPORDER authorization object where the value of the ACTVT field is set to 03. If this is not the case, the application is not permitted to display the data to the user. Listing 1.4 shows the corresponding ABAP source code for this check.

```
* Check display authorization
AUTHORITY-CHECK OBJECT 'ZIFPORDER' ID 'ACTVT' FIELD '03'.
CHECK SY-SUBRC EQ 0.
* Only proceed here if SY-SUBRC is = 0 .
* Display data.
```

**Listing 1.4** Checking Authorization for Displaying Purchase Orders

### 1.3.3 Number Ranges

For order IDs to be created automatically for new purchase orders, you must create a *number range object* in Transaction SNRO (number range maintenance). The name you assign to this number range is ZIFPORDER. You use the ZIFPORDERDOM domain for the length of numbers of this number range object.

**Number range intervals**

After you create and save the number range object, you can create a corresponding *number range interval* for it. You do this on the initial screen of Transaction SNRO by clicking the Number Ranges button in the application toolbar. From there, you can create new intervals or change existing ones by clicking the Change Intervals button. An interval of this number range is displayed in Figure 1.14. You use the NUMBER_GET_NEXT function module to assign a number from this interval.

### 1.3.4 Function Modules

Function modules are modularization units of the ABAP programming language. They contain executable source code and can be called from ABAP programs using the CALL FUNCTION ABAP statement. They have a well-defined interface for transferring and receiving data. This interface defines the parameter types and transfer direction. Data can only

be transferred to and received by the function module through this interface.

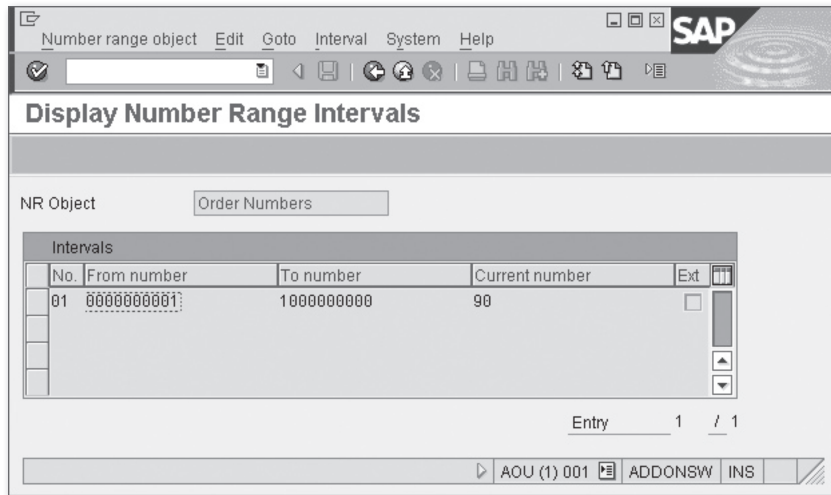



**Figure 1.14** Number Range Interval of ZIFPORDER Number Range Object for Purchase Order Numbers

This section deals with features common to all function modules. We will come across specific function modules in the following sections, in particular in Sections 1.3.5, Update Modules, and 1.4.2, Remote Function Call, and in Chapter 2, Remote Function Call with ABAP.

**Function Builder**

The ABAP developer uses the *Function Builder* (Transaction SE37) to create function modules.

In addition to the source text, the Function Builder also defines the interface and properties for a function module.

**Function groups**

Every function module is part of a *function group*. A function group is a program (F type) that can contain global data, dynpros, modules, and other components, like any other program. For example, the `DDIF_FIELDINFO_GET` function module is a member of the `SDIFRUNTIME` function group. This function group contains other function modules such as `DDIF_NAMETAB_GET`, `DD_INT_UPDATE_DDFTX`, and `DDIF_FIELDLABEL_GET` for accessing the ABAP Dictionary.

Every function module of a function group can access the global data of this function group. A new instance of the global data for the function group is created in this case for every calling program; the global data contents of a function group therefore differ for every calling program. When a program calls two function modules of a function group, the first function module can write data into the global data of the function group that the second function module can then read and use. This is how function module calls can become stateful.

Function module interface

With the calling program, the function module can only exchange data through its interface. There are five parts to the interface:

- **Import**
  Data is transferred to the function module through import parameters. The function module cannot change the actual parameters here. Import parameters can be simple or structured and also flagged as optional. A default value is also identified in this case when the parameters are being defined.
- **Export**
  After it has been executed, the function module receives data through export parameters. Export parameters can be simple or structured.
- **Changing**
  Changing parameters are a combination of import and export parameters: Data can be transferred to the function module through changing parameters and then changed and returned again by this function module.
- **Tables**
  Import, export, and changing parameters may only be simple or structured up to Release 6.40. You use table parameters to transfer internal tables (*arrays*). Table parameters are always transferred by reference. They are therefore used to transfer data to the function module and return values from the function module. To find out which of these two functions are performed by a table parameter, you will have to refer to the function module documentation.

Exception statuses

- **Exceptions**
  Possible exception statuses are also defined in the function module interface. If an error occurs, the function module triggers one of the exceptions defined by the `RAISE` keyword. The caller is responsible

for evaluating the function module return code. If an exception has occurred, the return code will be a value not equal to zero.

Every function module has this type of interface. A function module has other properties besides the interface. The next section covers update modules, while Chapter 2, Remote Function Call with ABAP, deals with RFC modules.

### 1.3.5 Update Modules

Based on the transaction concept explained in Section 1.1.2, SAP NetWeaver Application Server ABAP, application programs should not program database changes directly using `INSERT`, `UPDATE`, and `DELETE` statements because these database changes are committed on the database by an implicit database commit immediately after each dialog step. Instead, the application programs should call special function modules with an `IN UPDATE TASK` statement. These function modules are identified in the properties as *update modules*. They are not executed directly when called but only when the application program triggers the `COMMIT WORK` command. It is only then that these update modules are executed asynchronously in a separate V-type work process and that they store the data transferred to them at the time of the call to the database. This concept also means that an SAP LUW consisting of several dialog steps can be canceled by a `ROLLBACK WORK` without committing the changed data to the database.

**Mass updating**

To begin with, update modules for creating data records should store the data transferred to them in an internal table within the function group and then use the `ON COMMIT` statement to call a subroutine that will perform the actual update with *array insert technologies*. You can create several objects efficiently within a transaction using update modules programmed in this way. The subroutine that performs the actual update is only executed once and saves all data records in a single database access. This is particularly important if mass data is to be read.

Listing 1.5 shows the `z_ifp_v_create_order` update module. It first writes the transferred parameters for header and item data into the internal `it_orderheader_buffer` and `it_orderpos_buffer` tables that were defined in the global data area of the `ZIFP` function group. Then it calls the `buffersave` subroutine using the `ON COMMIT` statement. Because

of this statement, the subroutine is actually only executed once in the update process.

```
FUNCTION z_ifp_v_create_order.
*"--------------------------------------------------------------
*"*"Update function module:
*"*"Local interface:
*"  IMPORTING
*"     VALUE(IM_ORDERHEADER) TYPE  ZIFPORDER
*"  TABLES
*"      IM_ORDERPOS STRUCTURE  ZIFPORDERPOS
*"--------------------------------------------------------------
" Save the transferred data in the buffer table first
  APPEND im_orderheader TO it_orderheader_buffer.
  IF sy-subrc NE 0.
    MESSAGE a601(zifp) WITH im_orderheader-orderid.
  ENDIF.
  APPEND LINES OF im_orderpos TO it_orderpos_buffer.
  IF sy-subrc NE 0.
    MESSAGE a602(zifp) WITH im_orderheader-orderid.
  ENDIF.
  " Save entire table with commit
  PERFORM buffersave ON COMMIT.
ENDFUNCTION.
FORM buffersave.
  INSERT zifporder FROM TABLE it_orderheader_buffer.
  IF sy-subrc NE 0.
    MESSAGE a603(zifp).
  ENDIF.
  INSERT zifporderpos from table it_orderpos_buffer.
  IF sy-subrc NE 0.
    MESSAGE a604(zifp).
  ENDIF.
ENDFORM.                    "buffersave
```

**Listing 1.5** z_ifp_v_create_order Update Module

### 1.3.6 Application Functions and User Interfaces

So far, you have programmed the data access layer for the sample application with transparent tables, search help, lock objects, authorization objects, and update modules. Next we would need to develop the application functions and user interface for our application. But this is not necessary for our purposes, and the procedure is adequately described

in other books (e.g., *ABAP Objects* [SAP PRESS, 2007]), so we will only outline the rough procedure here.

Encapsulating functions

For an application to be supplied with alternative user interfaces or be used by external programs through interfaces, it is useful to encapsulate the application functions in function modules. We will see this in Chapter 2, Remote Function Call with ABAP. We will use the function modules we develop there as the basis for ALE, SOAP, and SAP NetWeaver XI interfaces in Chapter 5, IDocs and ALE; Chapter 6, Service-Oriented Architecture Protocol; and Chapter 7, SAP NetWeaver Process Integration. If you want an application to be interface-enabled, the interprocess communication technology used is therefore less important than the sensible breakdown of application functions in elementary modules that can be used by different users.

Graphical user interfaces

If the application functions are initially encapsulated in this way, it is also very easy to create graphical user interfaces using different techniques. There are at least four options to do this in ABAP:

- **Reports**
  Reports are type 1 ABAP programs that basically select data on the database and output it in the form of lists in the SAP GUI.
- **Module pools**
  Module pools are type M ABAP programs that consist of individual dynpros and are called through transaction codes.
- **Business Server Pages**
  Business Server Pages (BSPs) are HTML pages with embedded ABAP source code. These applications are not used through the SAP GUI but through a web browser (see *Web Programming in ABAP with the SAP Web Application Server* [SAP PRESS, 2005]).
- **ABAP Web Dynpro**
  ABAP Web Dynpro is a new framework for developing web applications. The interface here is not developed directly in HTML but rather described with graphical tools for the Web Dynpro runtime environment (see *Getting Started with Web Dynpro ABAP* [SAP PRESS, 2010]).

Developing user interfaces without ABAP

You can of course also develop user interfaces entirely without ABAP, for example, with Web Dynpro Java, Java Server Pages (JSPs), Java servlets, ASP.NET (*Active Server Pages*), and many other technologies.

## 1.4 Overview of SAP Interface Technologies

In this section, we will present the most important interface technologies used by ABAP programs:

- The *file interface* for reading and writing files (Section 1.4.1, File Interface)
- *Remote Function Call* (RFC) for calling function modules remotely (Section 1.4.2, Remote Function Call)
- *Business Application Programming Interfaces* (BAPIs), which provide an object-oriented view of application functions (Section 2.4, Business Objects and BAPIs)
- The ALE interface (*Application Link Enabling*) for exchanging messages asynchronously (Chapter 5, IDocs and ALE)
- The *Service-Oriented Architecture Protocol* (SOAP) based on the HTTP and XML Internet standards (Chapter 6, Service-Oriented Architecture Protocol)
- The *XI protocol* used by SAP NetWeaver XI/SAP NetWeaver PI (Chapter 7, SAP NetWeaver Process Integration)

In this section, we will explain the basic interdependencies and show you the most important transactions for managing interfaces.

### 1.4.1 File Interface

One method (which is also important in practice) to exchange data between two systems is to send it through a file system. The sending system writes data into a file that is then transported to a target system directory. This file is read by the target system and imported into the database. SAP systems already have completed programs for importing important types of data. However, these programs expect the file in a specific format. The file is read and the data written to one of three types in the database tables:

Import programs

- **Call transaction**
  The ABAP programming language knows the `CALL TRANSACTION` keyword that can be used to call an SAP transaction. The data to be entered is transferred to this transaction in the form of a *batch input table*. For every dynpro input field of the called transaction, the data

to be entered is available in the batch input table. The table also contains the function code to be used for every dynpro to complete an entry and navigate to the next dynpro.

A call transaction program reads the file with the data to be entered, creates a batch input table for every data record, and transfers this table to the SAP transaction using `CALL TRANSACTION` (which is normally executed online) to create a new data record. The called transaction is processed in the background, so the dynpros are not displayed. All input checks programmed in the SAP transaction naturally take place. This ensures that only valid and consistent data reaches the database through the call transaction technique.

- **Batch input**
  The first step with the batch input program runs like the call transaction program: The file with the data to be entered is read and a batch input table is created for every data record. However, this batch input table is not forwarded to the SAP transaction directly but is instead saved in a batch input session in the database. The batch input monitor (Transaction SM35) later reads this batch input session, and the batch input tables contained in it are transferred to the SAP transaction. We refer to this as "processing" the batch input session.

  **Batch input monitor**

  During this processing, a protocol is created that can be analyzed later using the batch input monitor. If an error occurs, you can correct the data and process the session again. This option of controlling and correcting errors is the main advantage of the batch input procedure compared to the call transaction technique.

- **Direct input**
  The two preceding techniques may be too slow for mass data transfers in particular. Direct input programs therefore exist for some data types, particularly financial accounting documents. These programs read the file with the data to be entered, check the data, and store it directly in the database using `UPDATE`. Existing dialog transactions are therefore not used here for performance reasons.

**CALL TRANSACTION in function modules**

These three techniques of program-driven data entry are sometimes encapsulated within function modules. The data to be entered is transferred through `TABLES` parameters of the function module in this case. The function module can then write the data to the database tables using

CALL TRANSACTION or direct input or set up a batch input session for processing later. Using CALL TRANSACTION in a custom-developed function module is particularly useful if a suitable BAPI is not available.

### 1.4.2 Remote Function Call

You can also call ABAP function modules from external programs using the *Remote Function Call* (RFC) application protocol. RFC-enabled function modules are specifically identified in their properties as being remote-enabled. The properties for the DDIF_FIELDINFO_GET function module are displayed in Figure 1.15. It is identified as a remote-enabled function module in the Processing Type section, which means it can be called by external programs.

**Figure 1.15** DDIF_FIELDINFO_GET RFC Function Module

Parameter typing

All parameters with a reference to the ABAP Dictionary must be typed for remote-enabled modules, and the transfer type for import and export parameters must always be passed by vlaue. This means you must select the PASS VALUE selection field for all parameters. Only since Release 6.40 are remote-enabled function modules also able to have changing parameters.

#### RFC Destinations

You call a function module from an ABAP program using the `CALL FUNCTION` command. If you want the function module to be executed in another system, you use the `DESTINATION <dest>` addition, whereby `<dest>` is a name maintained in Transaction SM59 (RFC Destinations). The technical information required by the calling system to reach the target system is maintained with the name in this transaction. We are particularly interested in two RFC connection types:

- Connection type 3: The target system is an SAP system.
- Connection type T: The target system is an external system.

ABAP target system

If the target system is an SAP system, the technical information you maintain when creating an RFC destination includes details about the application host or message server, the gateway to be used, and the details for logging on to the target system.

To create an RFC destination in Transaction SM59, click the Create button on the application toolbar. A dialog box then appears where you can enter the general destination attributes. These attributes include the name, a short description, and, in particular, the destination type.

To connect to an SAP system, choose Connection Type 3. As soon as you press [Enter], the appearance of the screen changes (Figure 1.16). You can use the Yes/No fields of the Load Balancing option to choose whether the message server of the target system is to define an application server when the connection is being established or whether you want to communicate with a specific application server. In the first case, you must specify the host name or IP address of the message server, the system ID, and logon group; in the second, you specify the host name or IP address of the application server and the system number of the SAP instance.

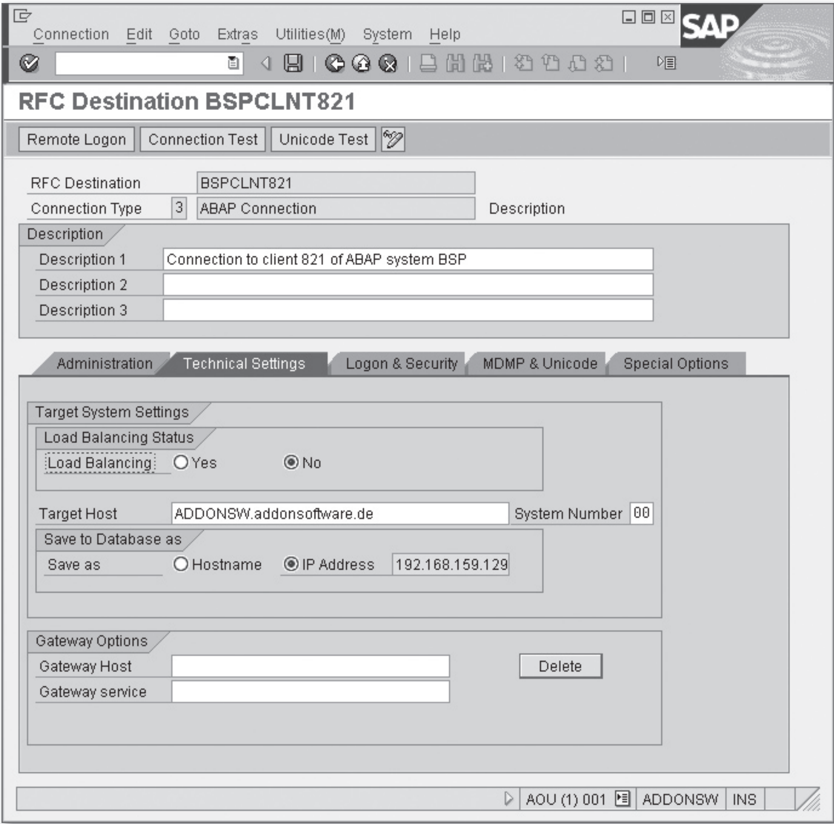



**Figure 1.16** Type 3 RFC Destination

Lastly, you still have to specify the client, user and password, and language you want to use to log on to the target system. You can use the Connection Test button on the application toolbar to check whether a connection can be made to the other system. If you click the Remote Login button, a new SAP GUI session is started with the other system.

**External target system**

However, if the target system is an external system, in Transaction SM59, you must specify how this external system can be reached. An RFC destination to an external system is Connection Type T. As soon as you have entered the name, connection type, and description of the destination, press the `Enter` key. The screen will then look like the one shown in Figure 1.17.

There are four ways an external system can be reached:

- **Starting the program on the application server**
  You select the Start on Application Server option here. In the Program input field, you specify the path to an RFC server program on the application server. The program must be on the application server where the ABAP program that triggers the `CALL FUNCTION` command is executed. When called, the specified program is started on the application server and executes the required function.
- **Starting the program on an explicit host**
  You select the Start on Explicit Host option here. In the Target Host and Program input fields, you specify the host name and path to an RFC server program on this host. When called, the specified program is started on the specified host and executes the required function.
- **Starting the program on the presentation server**
  Here you specify the path to an RFC server program on the presentation server. For this, you select the Start on Front-End Work Station option. In the Program input field, you specify the path to an RFC server program on the frontend work station. The program must be on the work station of the user who executes the ABAP program using the `CALL FUNCTION` command. When called, the specified program is started on the work station of that user and executes the required function.
- **Registration**
  Registering with the gateway
  Here you specify a Program ID under which the external program is registered for an SAP gateway. The external program must therefore have already been started beforehand, irrespective of executing `CALL FUNCTION`. Like the destination name, the program ID is case-sensitive, so uppercase and lowercase spelling is important. It is also important that you specify the gateway on which the external program has been registered. To do this, you specify the name of the gateway host and gateway service in the two Gateway Host and Gateway Service fields. The gateway service is always called *sapgw<SYSNR>*, where *<SYSNR>* is the two-digit system number of the SAP instance to which the gateway belongs.

  In reality, the registration is probably the most important way and therefore also shown in Figure 1.17. The external program is started

independently by a *Windows service* or a *UNIX daemon* and registered on a gateway under a program ID. You can use the external program functions from an ABAP program from this point onward.

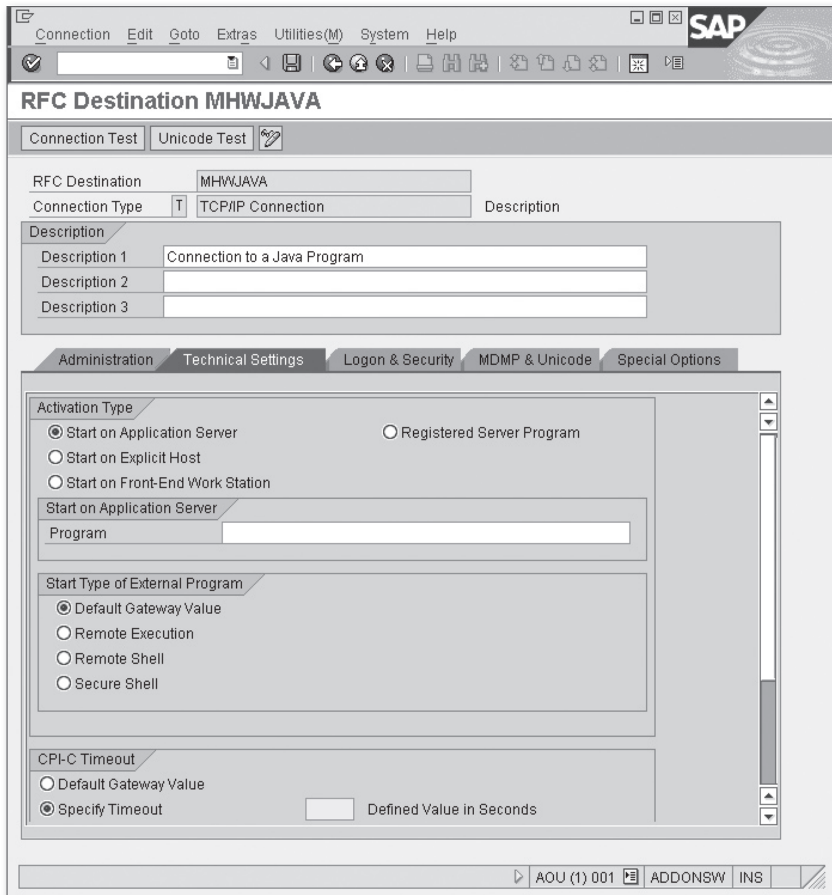



**Figure 1.17** Type T RFC Destination with Registration

**Specifying a destination when calling function modules**

As soon as you have maintained a destination, and the connection test has been successful, you can use `CALL FUNCTION ... DESTINATION ...` to call function modules from an ABAP program. Follow the `DESTINATION` keyword with the name of the RFC destination as you maintained it in Transaction SM59.

**Testing in the Function Builder**

You can also test calling a function module in an external system using Transaction SE37, the Function Builder, without having to write an ABAP program. Enter the function module name in the initial screen of the Function Builder, and click the Test/Execute button (F8) on the application toolbar. In the next dialog box that appears for entering the transfer parameters, you can also enter the destination name. Be careful: This name is case-sensitive, so uppercase and lowercase spelling is also important here.

### RFC Types

Over time, SAP has developed four types of RFCs:

- **Synchronous RFC (sRFC)**
  You use this normal RFC execution to read data in the called system.

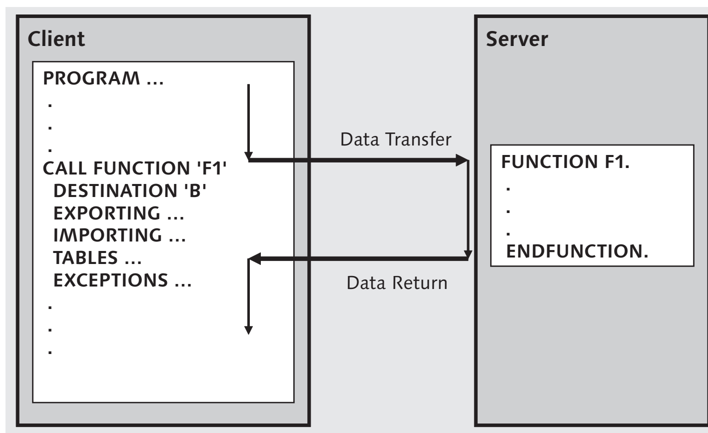



**Figure 1.18** Synchronous RFC Process Schema

**sRFC for reading data**

The synchronous RFC process schema is illustrated in Figure 1.18. The client calls the function module and waits until the function module has been processed. The client program then continues directly after the function module is called and can evaluate the exceptions and return parameters of the function module. An implicit database commit occurs in the client when the function module is called because the call begins a new dialog step and ends the old one. An implicit database commit also occurs in the target system after the function module is processed. The function module is therefore exe-

cuted within a single database LUW. Synchronous RFC is normally used to read data.

- **Asynchronous RFC (aRFC)**
  Asynchronous RFC involves the client program continuing directly after the function module is called, without waiting for the called function module to be processed. This means tasks can be processed in parallel.

aRFC for load balancing

The asynchronous RFC process schema is illustrated in Figure 1.19. The client program continues directly after the function module is called, without waiting for the called function module to be processed. Therefore, neither the return parameters nor specific exceptions for the function module can be received and evaluated. When the function module is called, however, a `FORM` subroutine can be registered, which the system will call after the function module has been processed and where the return parameters for the function module can be received.

An implicit database commit also occurs here in the client after the function module is called, and the function module is also processed in a separate database LUW in the target system. Asynchronous RFC is used to process tasks in parallel. It is only available for ABAP programs, not external programs.

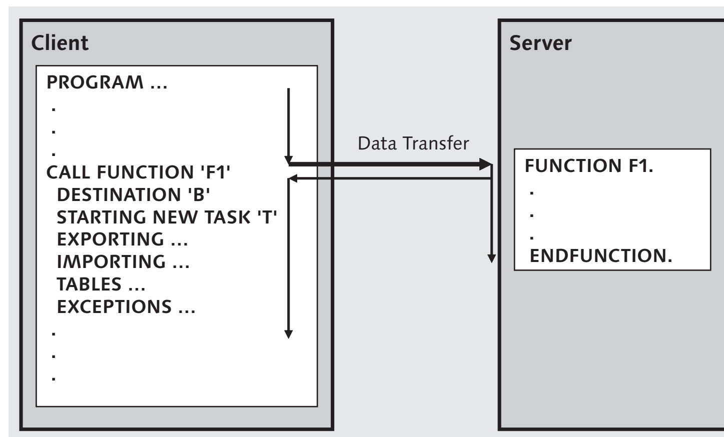



**Figure 1.19** Asynchronous RFC Process Schema

- **Transactional RFC (tRFC)**
  When the function module is called using import parameters with this type of RFC, a *transaction ID* (TID) is also specified. The function module is only executed specifically once on the server.

  tRFC for writing data

  The transactional RFC process schema is illustrated in Figure 1.20. When the TID is received, the called system checks whether data has already been processed once with this TID. It will only execute the called function module if this is not the case. This ensures that data with the same TID is only processed once.

  If a system or connection error occurs, the client can save the data together with the TID and send it once again at a later stage. Even if the data on the called system had already been processed, for example, because a connection error only occurred when confirmation was being returned, the client can send the data with this TID any number of times. This guarantees that the data is only processed once.

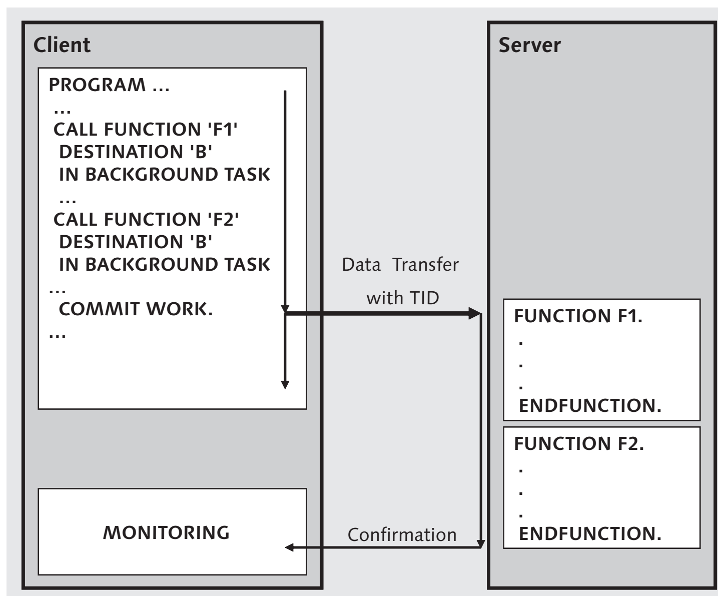



**Figure 1.20** Transactional RFC Process Schema

Like asynchronous RFC, the client does not get back the function module return parameters here either. An implicit database commit is

not executed when a function module is called on the client. The dialog step is only completed on the client when the explicit `COMMIT WORK` is executed. The called function modules are processed in a single database LUW on the server. tRFC is used to write or change data.

- **Queued RFC (qRFC)**
  Queued RFC ensures that two tRFC packages that have been triggered in succession by the client are also processed one after the other by the server. The sender writes the tRFC packages into a queue for this purpose. The packages in this queue are then sent by tRFC to the receiver in the same sequence in which they were written in the queue. Using tRFC ensures that a tRFC package is also only specifically processed once if an error occurs. A package from the queue is only sent to the receiver when the receiver has safely received the previous package. This is the only way of guaranteeing that the processing sequence will be preserved.

### Authorizations for RFC

Authorization checks occur within function modules, as they also do within every other ABAP program. This check verifies whether the user who executes the function module has *authorization* with the necessary field contents in one of the profiles assigned to his user master data record.

Authorization object

An authorization is an instance of an authorization object. The *authorization object* defines fields and possible values for these fields. For example, an authorization object field could be called `ACTVT` (for “activity”) and its possible field contents could be 01 for displaying, 02 for creating, and 03 for changing data. Another field could be called `BUKRS`, and its possible field contents might be company code numbers. So if the user has authorization where 01 is set as the value for the `ACTVT` field and 1000 is set for `BUKRS`, he is allowed to display data for company code 1000.

Within his program, the ABAP programmer uses the `AUTHORITY CHECK` ABAP statement to check whether the user has the necessary authorization to perform the required action. If so, the program will continue; otherwise, it will terminate with an error message.

Authorization object for function groups

As well as authorization checks that occur in the function module, the ABAP runtime system also checks the S_RFC authorization object. This authorization object contains three fields:

- **RFC_NAME**
  This field contains the name of a function group.
- **ACTVT**
  This field can only have value 16 (Execute).
- **RFC_TYPE**
  This field always contains the FUGR (function group) value.

The S_RFC authorization object can be used to control whether an RFC user is allowed to execute function modules of a particular function group. Unfortunately, a limitation on specific function modules of a function group is not currently possible.

RFC users should only be given the authorizations absolutely necessary for them to be able to execute the function module(s) required for the task, so under no circumstances should they be assigned the SAP_ALL authorization profile. To find out which authorizations are required for a specific interface scenario, you can set the value of the auth/authorization_trace profile parameter in the SAP instance profile file to "y." Selecting this setting means the system will write the performed authorization checks and their result into the USOBX table. This is described in more detail in SAP Note 0543164.

### Monitoring and Troubleshooting

An SAP system contains some tools for monitoring the execution of RFC calls and consequently informing you of errors. You can use Transaction ST22 to list and analyze *ABAP short dumps* that were triggered by RFC function modules. An ABAP short dump contains information on an error that has occurred and the part of the program where it happened. It also provides details on system variable contents, the call stack, and other useful information.

You can use the *Gateway Monitor* (Transaction SMGW) to display a gateway trace. To do this, you select the GOTO • TRACE • GATEWAY • DISPLAY FILE menu options.

RFC tracing

You can activate *RFC tracing* for a user in Transaction ST05. Until this setting is deactivated, an RFC trace will also be recorded for this user and can then be analyzed in Transaction ST05. RFC traces can also be recorded based on a destination. The TRACE flag is set for this when you maintain the destination in Transaction SM59. You can subsequently display this trace by selecting the RFC • DISPLAY TRACE menu options.

### 1.4.3 BAPIs

*Business objects* provide an object-oriented view of a major part of business data and transactions for an SAP system. You maintain object types of business objects in the *Business Object Repository* (BOR) using Transactions SWO1 and SWO. The components of such an object type are

- **Basic data**
  The technical data about the object type, such as the internal name, the release, or transport data.
- **Interfaces**
  The interfaces implemented by the object type.
- **Key fields**
  Fields whose contents uniquely identify an object type instance. They are often fields of database tables where the objects are stored persistently.
- **Attributes**
  The object type properties, either references to database fields or to other business objects.
- **Methods**
  Used to call transactions or function modules.
- **Events**
  Used in workflow definitions. Events are triggered through messages or status changes.

Business object methods

Some object type methods can be identified as Business Application Programming Interfaces (BAPIs). You always implement BAPIs as RFC function modules and can therefore use them like other RFC function modules. BAPIs also have other properties:

- **BAPIs have a stable interface.**
  SAP guarantees that a published BAPI interface is not changed incom-

patibly in a higher release. This means that optional parameters at most will be added in a higher release. But the BAPI normally remains untouched in higher releases. If there are function changes or enhancements, SAP provides a new BAPI with a new name. For example, the BAPI_SALESORDER_CREATEFROMDATA and BAPI_SALESORDER_CREATE_FROMDAT1 BAPIs were provided for creating sales orders. BAPIs that have become obsolete are kept for two more releases at least, so that old clients can also work with the new release.

- **BAPIs do not have a presentation layer.**
  BAPIs do not call dynpros using CALL SCREEN. They only communicate through the interface of their implementing function module. If BAPI results are to be displayed, the caller will be responsible for this. BAPIs can therefore also be used in programs where two hosts are to communicate between each other without user intervention.
- **BAPIs do not trigger exceptions.**
  BAPIs use the RETURN export parameter for success, warning, and error confirmations. Depending on the BAPI, this parameter is a BAPIRETURN, BAPIRETURN1, BAPIRET1, or BAPIRET2 type structure or an internal table with these line types. All these structures have the following two important fields in common:
  - TYPE
    This field can include the values S (for Success), E (for Error), W (for Warning), or I (for Information). This field may also contain a space instead of the value S.
  - MESSAGE
    This field contains a message text with information about an error.
- **Databases are updated in the update task.**
  BAPIs always perform database updates by calling an update module using CALL FUNCTION IN BACKGROUND TASK. A COMMIT WORK or ROLLBACK WORK is not programmed within the BAPI. The following two BAPIs are available to actually perform or reject the database update:
  - BAPI_TRANSACTION_COMMIT
    Performs all allocated update tasks. You can specify in an import parameter for this BAPI whether you want the database to be updated synchronously or asynchronously.

Transaction concept for BAPIs

- `BAPI_TRANSACTION_ROLLBACK`
  Rejects the allocated database updates.

**Standard BAPIs**

Most business objects provide the following standard BAPIs:

- **`GetList`**
  Search criteria can generally be transferred to this BAPI. A table containing all object keys corresponding to this search criteria is returned as the result.
- **`GetDetail`**
  This BAPI requires an object key as an import parameter. It returns the other attributes for the object.
- **`Change`**
  With this BAPI, by specifying an object key and the new attributes, you can change the previous object attributes.
- **`CreateFromData`**
  With this BAPI, you can specify the attributes of a new object to be created. The key of the new object to be created is returned.

**Transaction BAPI**

You can use Transaction BAPI to display a list of all business objects and their BAPIs. The display is either arranged by application area or alphabetically by business object. Transaction BAPI provides detailed documentation for every business object, its BAPIs, and their parameters. From this transaction, you can also call the tools used to test and manage BAPIs.

Transaction BAPI differentiates between instance-dependent and instance-independent BAPIs. To call *instance-dependent BAPIs*, you must specify a business object key. The standard `GetDetail` and `Change` BAPIs are examples of instance-dependent BAPIs. *Instance-independent BAPIs* in contrast do not need a key specified for them. They work independently of an object instance. Examples are the standard `GetList` and `CreateFromData` BAPIs.

### 1.4.4 Application Link Enabling

Application Link Enabling (ALE) provides services and tools within SAP systems to exchange messages between applications in different systems. The messages are normally exchanged by sending and receiving electronic documents known as *Intermediate Documents* (IDocs). The purpose

of sending these messages is to implement shared business processes. ALE enables customizing data, master data, and application data to be shared on different systems.

One example is the delivery of sales orders in SAP ERP. As soon as it has been established when certain goods will be delivered to certain customers, the SAP system sends a delivery order to an external warehouse. The goods are loaded onto pallets and trucks there. As soon as the truck moves off with the delivery, an order confirmation is sent back to the SAP system. Apart from delivery orders and order confirmations, customer and material master data must also be exchanged in this scenario.

**ALE architecture**

A diagram of the ALE architecture is illustrated in Figure 1.21. The sending application is responsible for setting up the IDoc, possibly determining the receivers and transferring the IDoc to the ALE layer.

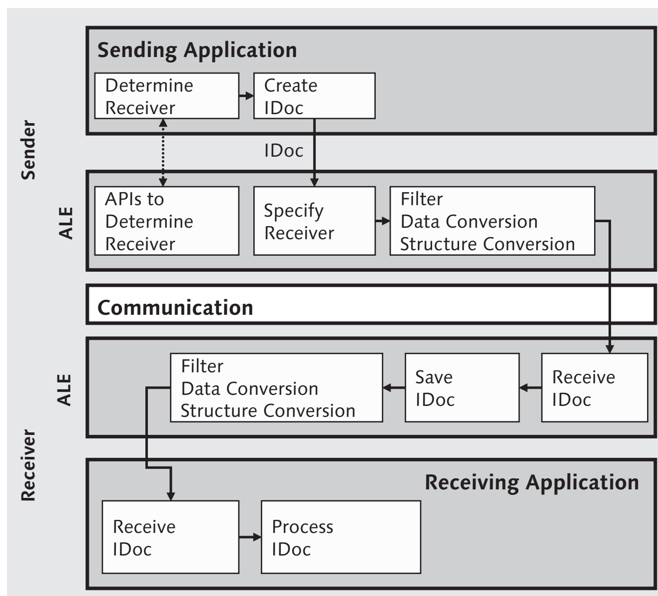



**Figure 1.21** ALE Architecture

After the IDoc is received, the ALE layer determines the receivers, if this has not already been done by the sending application. The IDoc is then processed further according to configurable rules: Individual fields or

segments can be filtered, data converted, or the IDoc structure adapted to the structure of older releases. An IDoc is created for every receiver in this case and transferred to the communication layer for transporting further.

The communication layer forwards the IDoc to the target system. The `IDOC_INBOUND_ASYNCHRONOUS` function module is normally called for this in the target system through tRFC. But you can also configure other transport mechanisms such as the transfer of data through the file system.

After it is received, the IDoc is stored in the target system and forwarded to the ALE layer. Like in the sender, in the ALE layer, the received IDoc can go through some more processing steps before it is forwarded to a function module of the receiving application.

Logical systems

Senders and receivers in the ALE context are referred to as *logical systems*. Logical systems are identified by names that are 10 characters long. You maintain logical system names in Transaction SALE as part of ALE Customizing. A short description of the logical system is also stored there in addition to a name. Every client in an SAP system gets a logical name. It generally consists of the system ID and client, for example, `UL0CLNT800` for the logical system, which corresponds to client 800 in the `UL0` system.

Messages

Messages are exchanged between logical systems with ALE. You maintain the names of possible messages together with a short description in Transaction WE81. For instance, the `MATMAS` message (material master) stands for an exchange of material master data, and the `DELVRY` message (delivery) describes an exchange of delivery orders.

### Intermediate Documents

Messages between two logical systems are sent in the form of intermediate documents (IDocs). These documents contain data to be exchanged.

IDoc segments

An IDoc is made up of data records also known as *segments*. Each segment begins with a header 63 bytes long, followed by the actual data up to 1,000 characters in length. The header in every segment has the same structure and the entire IDoc follows a hierarchical structure: Each segment can point to a parent segment located above the corresponding segment in the hierarchy (Table 1.6).

You define IDoc segments in Transaction WE31. You give the segments a name and short description there. But above all, you give the SDATA field structure a more detailed description.

| Field | Length | Description |
|---|---|---|
| SEGNAM | 30 | Type name of segment. This defines the SDATA data field structure. |
| MANDT | 3 | SAP client number. |
| DOCNUM | 16 | Unique IDoc number to which this segment belongs. |
| SEGNUM | 6 | Sequential segment number within the IDoc. |
| PSGNUM | 6 | Sequential number of parent segment. |
| HLEVEL | 2 | Hierarchy level of this segment. |
| SDATA | 1,000 | Actual application data. |

**Table 1.6** IDoc Segment Structure

An IDoc segment can be found in several *IDoc types*. You define IDoc types in Transaction WE30. In doing so, you determine which segments are found in the IDoc type, whether the segment in question is optional, how many segments of this type maximum can be found, and which are the child segments of the segment in this IDoc type.

Message types

An IDoc type is assigned to one or more *message types*. For example, you can use the ORDERS04 IDoc to transport ORDER (purchase order) or ORDRSP (order confirmation) messages. You assign messages to IDoc types in Transaction WE82. The SAP NetWeaver AS ABAP release and message name determine the IDoc type uniquely in this case. IDoc types used, for instance, are ORDERS01 as of Release 3.0, ORDERS02 as of Release 3.0D, ORDERS03 as of Release 4.0A, and ORDERS04 as of Release 4.5A.

### ALE Distribution Model

Which messages will be sent to which receivers by a sender is defined in a *distribution model* as part of ALE Customizing. The distribution model has a hierarchical structure with the sender system as its root. Under the sender system is the list of receiving systems, under that the messages, and under that the filters to be used on the messages.

ALE configuration

You maintain the settings in the ALE system in Transaction SALE. This transaction is a subset of the *SAP Implementation Guide*, which you can access in project management (Transaction SPRO) by selecting the SAP Reference Implementation Guide button. The steps required to set the ALE system are listed in this transaction. In the simplest case, they are the following steps:

1. Name the logical systems.
2. Assign a logical system to each SAP client.
3. Use Transaction SM59 to create tRFC destinations for the logical systems. To enable other definitions to be generated automatically in a later step, you should name the destination for a logical system exactly like the logical system itself.
4. Maintain the distribution model by defining which messages will be sent to specific receivers and the filters that will be used to do this. You can generate partner profiles from maintaining the distribution model. Partner profiles control the processing of inbound and outbound IDocs.

Partner profiles

5. Check the automatically generated partner profiles. Important distribution profile parameters are
   - **Outbound port**
     The outbound port defines how a partner can be reached. When the partner profiles are generated, the ports are created automatically if the logical systems and tRFC destinations created for them have the same name.
   - **Sending and processing mode**
     IDocs can either be sent and processed immediately or accumulated until a package containing a defined number of IDocs has been produced. The IDocs in this package are then sent and processed together.
   - **Package size**
     You can specify the size of the packages to be sent and processed.

Transaction SALE also provides access to many other functions for configuring IDoc processing and monitoring.

### BAPIs and ALE

Calling BAPIs asynchronously

Since SAP R/3 Release 4.0, BAPIs can also be called asynchronously through ALE. In fact, all new asynchronous interfaces (IDocs) since this

release are based on BAPIs. You can generate the necessary definitions for a given BAPI in Transaction BDBG:

- **Message type**
  A message type is generated for a BAPI.
- **IDoc type**
  Segments are generated for the BAPI export parameters, as is an IDoc type that can be used to transport these parameters. The IDoc type is assigned to the generated message type.
- **Outbound function module**
  This function module has the same import parameters as the BAPI and is called by the sending application instead of the BAPI. It creates an IDoc from the transfer parameters and transfers it to the ALE services for further processing.
- **Inbound function module**
  The ALE services use this function module to process the inbound IDoc. The function module extracts the data from the IDoc and calls the corresponding BAPI in the receiving system.

The generated message type can then be used like every other message type when an ALE distribution model is being created. If an application wants to call BAPIs in external systems, it proceeds as follows:

Using the ALE BAPI interface

1. **Querying filters**
   The `ALE_BAPI_GET_FILTEROBJECTS` function module is called to determine filter objects for a specific BAPI that were created when the distribution model was being maintained.
2. **Querying receivers**
   The `ALE_ASYNC_BAPI_GET_RECEIVER` function module is called to get a list of logical receiving systems from the distribution models. The calling application must transfer the table of filter objects for this purpose.
3. **Calling the BAPI asynchronously**
   The generated outbound function module for which (apart from the BAPI import parameters) the list of receiving systems is also specified is now called.
4. **COMMIT WORK**
   Finally, the application must ensure it forwards the generated IDoc to the ALE layer using the `COMMIT WORK` ABAP command.

This procedure means BAPI calls can be sent from one application to another as IDocs.

### 1.4.5 SOAP

Since Release 6.20, SOAP (*Service-Oriented Architecture Protocol)* services and clients can also be implemented using SAP NetWeaver AS ABAP. SOAP over HTTP involves the SOAP client sending a request in the form of an XML SOAP document to a SOAP server that replies with an XML SOAP document.

**Internet Connection Manager**

SAP NetWeaver AS ABAP calls a SOAP web service over HTTP POST. The call is first received by the *Internet Connection Manager* (ICM). You can use Transaction SMICM to view and change the ICM configuration. By selecting the GOTO • SERVICES menu options, you can see the TCP port on which the ICM receives HTTP requests. You can view the configured J2EE server ports (for a double-stack system) and a table of URL prefixes by choosing the GOTO • HTTP SERVER • DISPLAY DATA menu options. If an HTTP client requests a URL that begins with one of these prefixes, the ICM forwards the request to the *Internet Connection Framework* (ICF) on the ABAP stack; otherwise, it forwards it to the configured J2EE server.

**Internet Connection Framework**

You view the configured services in the ICF in Transaction SICF. Choose Service as the Hierarchy Type on the selection screen that appears when you start the transaction. A list of all HTTP services configured in the ICF is subsequently displayed. SOAP web services are implemented by SAP NetWeaver AS ABAP 7.0 through the services under the /sap/bc/srt node. You will see the IDoc (SOAP entry for IDoc), rfc (SOAP runtime for RFC), and xip (SOAP runtime for XI message interface) nodes directly under this node.

If you double-click the srt entry, you will see on the Handler List tab that the `CL_SOAP_HTTP_EXTENSION` ABAP class is entered as the handler for requests whose URL path begins with */sap/bc/srt/rfc*. The ICF analyzes the URL path of the request and forwards it to the corresponding handler class.

**Web service**

In Chapter 6, Service-Oriented Architecture Protocol, we will show you how to create a web service in the Object Navigator. After you create it,

you will have to release a web service for the SOAP runtime environment in Transaction WSCONFIG. You can use Transaction WSADMIN to change trace and log settings for web services, call the WSDL file (*Web Service Description Language*) for a web service, and test a web service. SAP provides the `wsnavigator` application on every SAP NetWeaver AS Java for this test. A URL on the web service WSDL can be transferred to this application. From the WSDL, it then creates a graphical user interface through which the web service can be tested.

To ensure you can test a web service from Transaction WSADMIN, select the Goto • Settings • Administration menu path to enter the URL for SAP NetWeaver AS Java in the form of *http://<javaserver>:<httpport>*. As soon as you have configured SAP NetWeaver AS Java, you can click the Web Service Homepage button (Ctrl + F8) to start a browser that will navigate to the `wsnavigator` application on the configured SAP NetWeaver AS Java. From there you can test the web service. Figure 1.22 shows the test screen for the `SRT_TESTS_FB_ADD_WS` web service.

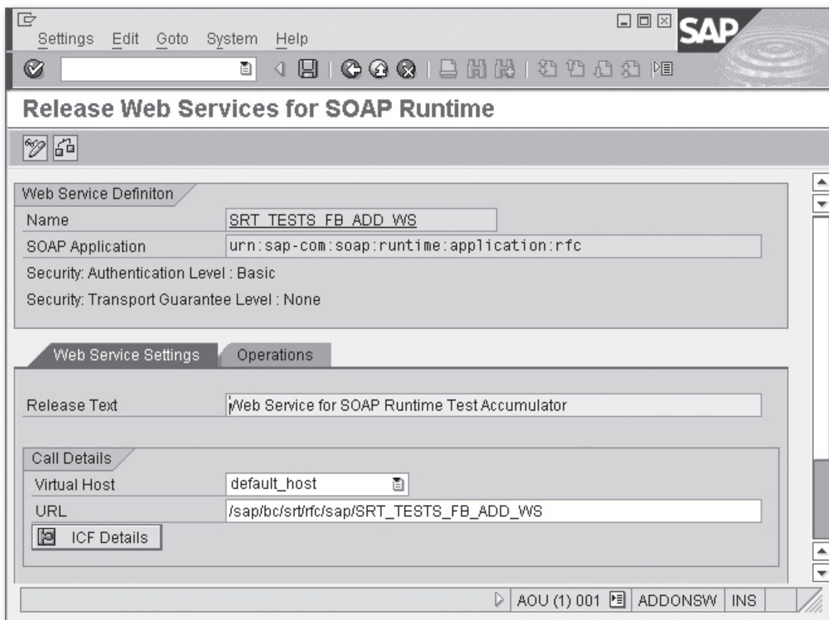



**Figure 1.22** Web Service Homepage of SRT_TESTS_FB_ADD_WS Web Service

### 1.4.6 XI SOAP

SAP NetWeaver PI and XI

With *SAP NetWeaver Process Integration* (SAP NetWeaver PI, formerly *SAP Exchange Infrastructure*, SAP NetWeaver XI), SAP provides a central platform for distributing application messages. The central *integration engine* within an SAP NetWeaver PI system communicates in this case with an SAP-specific version of SOAP over HTTP. This protocol is generally referred to as *XI SOAP*.

With this SOAP version, the HTTP body contains a MIME document (*Multipurpose Internet Mail Extensions*) consisting of a SOAP document and the actual message as an attachment. This means that — unlike standard SOAP — any number of binary files can be sent without the need to convert code. The SOAP header of an XI SOAP message contains SAP-specific information about the sender, message interface used, and required *quality of service*. The quality of service can take one of the following three values:

- **Best Effort (BE)**
  Communication occurs synchronously. If a communication error occurs, the client must repeat the call.
- **Exactly Once (EO)**
  Communication occurs asynchronously. If a communication error occurs, the message can be resent from the technical monitors. This value makes sure the message is only processed once.
- **Exactly Once In Order (EOIO)**
  Communication occurs like EO, but the processing sequence of several messages is also adhered to.

Transaction SPROXY

As we will show you in Chapter 7, SAP NetWeaver Process Integration, it is very easy on SAP NetWeaver AS ABAP to create client and server applications that use XI SOAP for communicating. First we will describe the outbound and inbound message interfaces in the *Integration Repository* of the SAP NetWeaver PI system. Then we will use Transaction SPROXY on SAP NetWeaver AS ABAP to create proxy classes that an application can use to send or receive messages with XI SOAP. Figure 1.23 shows Transaction SPROXY for the `Booking_Order_Confirmation_In` inbound message interface.

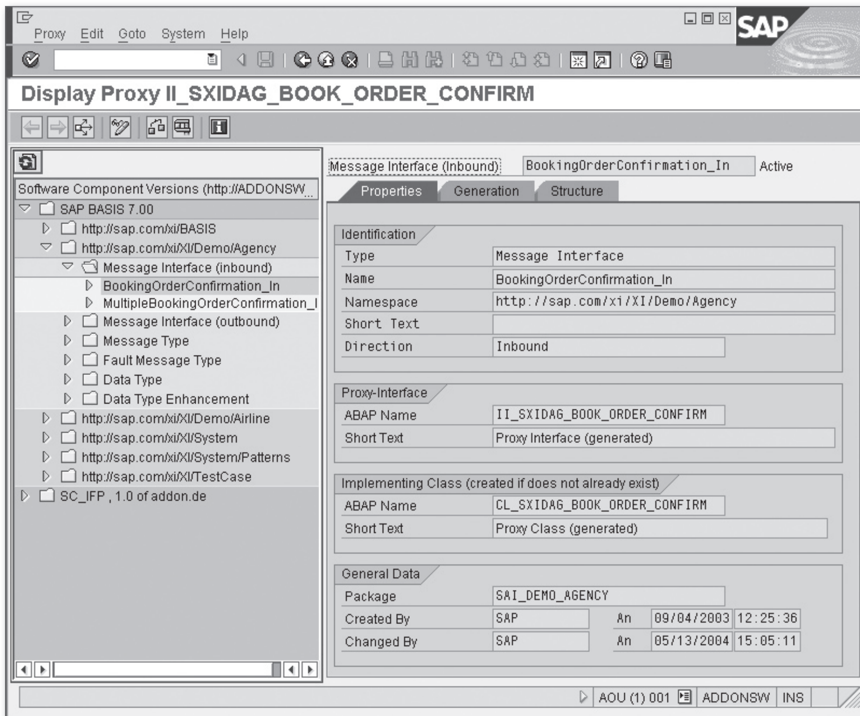



**Figure 1.23** Transaction SPROXY for an Inbound Message Interface

# Index

.NET, 342
.NET Connector (NCO), 342

## A

ABAP, 16
  *ABAP Dictionary, 58, 73, 119, 242*
  *ABAP Workbench, 13, 58*
  *Short dump, 81*
  *SOAP proxy, 374*
  *SOAP web client, 319*
  *SOAP web service, 311*
  *Stack, 15*
  *Transaction concept, 27*
  *Web Dynpro, 69*
  *XI proxy, 356*
Access
  *Security, 33*
ACL (Access Control List), 42
  *Entry, 42*
  *Maintenance, 43*
Active Server Pages .NET (ASP.NET), 69
Adapter object, 353
Adaptive web service, 338
ALE, 84, 271
  *Architecture, 85*
  *Configuration, 284*
  *Customizing, 86*
  *Distribution model, 87, 277, 286, 288, 293, 299, 310*
  *Inbound function module, 283, 293*
  *Inbound parameter, 289, 294*
  *Outbound function module, 292*
  *Partner profile, 88*
  *Process code, 283*
  *Receiver determination, 295*
  *Receiver port, 288*
  *Status monitor, 291*
Annotation, 31, 341
  *SOAPBinding, 341*
  *WebMethod, 341*
API (Application Programming Interface), 30
Application
  *wsnavigator, 91*
Application bar, 19
Application component, 29
Application layer, 13, 14, 16, 31
Application Link Enabling (ALE), 84
Application protocol, 20
Application security, 33
Application server, 14, 19, 29, 73, 149
  *Dedicated, 259*
Application server, 29
Architecture
  *Client-server, 14*
Array, 67
Array insert, 67, 122
ASP.NET, 69
Asymmetric procedures, 33
Authentication, 32, 40, 259
Authorization, 32, 41, 63, 80
  *Check, 97, 104, 141*
  *Object, 63, 80*

## B

Background
  *Work process, 22*
BAdI, 120
BAPI, 82, 118
  *ALE interface, 88, 292*
  *BAPI_TRANSACTION_COMMIT, 83, 134*
  *BAPI_TRANSACTION_ROLLBACK, 84*
  *Conventions, 119*
  *Error message, 130*
  *GetSearchhelp, 141*
  *Input help, 141*
  *Instance-dependent, 84, 129*
  *Instance-independent, 84, 136*
  *Interface, 82*

*Release, 127*
*RETURN parameter, 120, 122*
*Transaction model, 83, 110, 174*
*Update, 122*
Batch input, 71
Best Effort (BE), 92
Binding, 345, 378, 387
*Custom, 345*
Binding style
*Document, 316*
*RPC, 316*
BOR, 82
Browser, 15
BSP, 69
Buffer area, 23
Business Add-In (BAdI), 120
Business application, 14
Business Application Programming Interface (BAP), 82
Business object, 82, 118
*Helpvalues, 141*
Business Object Builder, 118
Business Server Pages (BSP), 69
Business system, 352, 366

## C

Cast, 267
CCI, 257
CE (SAP NetWeaver Composition Environment), 372
Certificate, 52, 56, 325
*Export, 43*
*Import, 45*
*X509, 325*
Changing parameter, 66, 158, 242
Check field, 60
Check table, 60
Class
*BindingProvider, 389*
*JCo.Client, 242*
*JCoConnectionData, 225*
*JCoContext, 223*
*JCoDestination, 226*
*JCoDestinationManager, 221*
*JCoDestinationMonitor, 224*
*JCoFieldIterator, 231*
*JCoFunction, 226*
*JCoRepository, 226*
*JCoServerFactory, 247, 248*
*JCoTable, 228*
*PortalRuntime, 265*
*RecordFactory, 266*
Client, 17
*Proxy, 356, 358, 363*
*Server architecture, 14*
Client, 17
Collective search help, 61
Command field, 18
Commit, 100
Common Client Interface (CCI), 257
Communication
*Asynchronous, 30*
*Façade, 238*
Compile, 152
Compiler, 152
Composite application, 31
Confidentiality, 33
Configuration scenario, 367
Connection, 265
*Direct, 223*
*Establish, 242*
*Management, 257*
*Parameter, 239*
*Pooled, 223*
Consumer, 380, 388
*Proxy, 380, 388*
C RFC library, 145, 220
Cryptography
*Private key, 33*
*Provider, 38, 47*
*Public key, 33*

## D

Data
*Security, 33*
Data access layer, 32
Database, 14

Database Logical Unit of Work
*(DB LUW), 27*
Data element, 58
Data layer, 14
Data type, 354, 372
DB LUW, 27
Deployable Java proxy, 337
Deployment descriptor, 31, 268, 327
Design by contract, 30
Destination, 98, 215, 255, 286, 287, 320
*,BACK';destination Ba, 100*
*Name, 338*
*,NONE';destination No, 100*
*SPACE;destination Sp, 100*
DIAG, 16, 20
Dialog
*Processing, 25*
*Step, 26*
*Transaction, 25*
*Work process, 22*
Digital
*Certificate, 35*
*Signature, 34*
Direct input, 71
Dispatcher, 21, 24
Dispatching, 202
Distribution model (ALE distribution model), 286
DLL, 219
Double stack, 14
Dynamic Information and Action Gateway (DIAG), 16
Dynamic Link Library (DLL), 219
Dynpro Screen, 26

## E

EAR, 362
*Project, 362, 383*
ECo, 238
EIS, 257
EJB, 29, 253, 326
*Container, 32*
*Project, 326, 360, 383*
Encryption, 32, 33
Endpoint API, 339
Enqueue, 23
*Work process, 28*
Enterprise application project, 327, 337
Enterprise Archive (EAR), 362
Enterprise Connector (ECo), 238
Enterprise Information System (EIS), 257
Enterprise Java Bean (EJB), 29
Enterprise Service, 376
*Browser, 384*
Enterprise Services Repository (ES Repository), 372
Entity bean, 29
Error handling, 242
ES Repository, 372, 384
Exactly Once (EO), 92
Exactly Once In Order (EOIO), 92
Exception, 66, 100, 159
*Communication_failure, 100*
*system_failure, 100*
Export parameter, 66, 158, 230, 242, 243, 266
Extensible Markup Language (XML), 306
Extensible Stylesheet Language for Transformations, 306

## F

Fault message type, 354
Fault type, 242
File
*app.config, 345*
*Interface, 70*
*saprfc.h, 155*
*saprfc.ini, 151, 193*
*Services, 21*
Firewall, 24
Flagged for changing, 28
Foreign key dependency, 60
Function
*ItAppLine, 171*
*ItCreate, 171*
*ItFill, 171*

*ItGetLine*, 171
*RfcAccept*, 189, 194
*RfcCallEx*, 160, 172
*RfcCallReceiveEx*, 160, 172
*RfcCancelRegisterServer*, 193
*RfcCheckRegisterServer*, 193
*RfcConfirmTransID*, 177
*RfcCreateTransID*, 177, 186
*RfcDispatch*, 190, 202
*RfcGetData*, 196, 197, 198
*RfcGetNameEx*, 204
*RfcIndirectCallEx*, 177, 186
*RfcInstallFunction*, 190, 194, 202
*RfcInstallTransactionControl*, 205, 207
*RfcInstallUnicodeStructure*, 174
*RfcLastErrorEx*, 164
*RfcListen*, 161, 203
*RFC_ON_CHECK_TID*, 205
*RFC_ON_COMMIT*, 205
*RFC_ON_CONFIRM_TID*, 205
*RFC_ON_ROLLBACK*, 205
*RfcOpenEx*, 155, 160
*RfcQueueInsert*, 188
*RfcRaise*, 196, 200
*RfcReceiveEx*, 160, 172
*RfcSendData*, 196, 197
*RfcWaitForRequest*, 202
*TID_check*, 208
*TID_commit*, 208
*TID_rollback*, 208
Function Builder, 65, 77, 119
Function group, 65
Function handler, 249
*Register*, 250
Function module, 28, 64
*ALE_INPUT_ALEAUD*, 279
*ALE_MODEL_INFO_GET*, 276
*BALW_BAPIRETURN_GET2*, 126
*BAPI_TRANSACTION_COMMIT*, 174
*IDOC_INBOUND_ASYNCHRONOUS*, 86, 296
*IN BACKGROUND TASK addition*, 106
*Interface*, 66
*IN UPDATE TASK statement*, 67
*MASTER_IDOC_DISTRIBUTE*, 279
*QIWK_REGISTER*, 117
*TRFC_QIN_RESTART*, 117, 189
*TRFC_SET_QIN_PROPERTIES*, 115
*Update*, 28


## G


Gateway, 21, 73, 189, 192, 253
*Host*, 75
*Monitor*, 81
*Service*, 75
Generic Security Services Application Programming Interface Ver, 52
GSS-API V2, 52


## H


Hash function, 34
Hash value, 34
Home interface, 366
Hook method, 30
Host name, 19, 20
HTTP (Hypertext Transfer Protocol), 22, 311


## I


IC, 325
ICF, 90
ICM, 22, 90
Identity, 37
IDoc, 84, 86, 271
*Basic Type*, 274
*Client*, 303
*Control record*, 277, 278, 281, 298, 300
*Creation*, 275
*Data record*, 279, 298, 300
*Number*, 282
*Post-processing*, 291
*Programming with C*, 296
*Programming with Java*, 302

*Receiver, 297*
*Segment, 86, 272, 303*
*Segment definition, 272*
*Segment type, 272*
*Sender, 300*
*Server, 307*
*Status, 282*
*XML processing, 307*
IDoc type, 87, 271, 273, 292
*ALEREQ01, 275*
Implementation Container (IC), 325
Import parameter, 66, 158, 226, 242, 243, 266
Inbound queue, 112, 116
Input check, 60
Instance, 19
*IDocRepository, 303*
*IInteraction, 266*
Integration
*Directory, 348, 366*
*Engine, 92, 356*
*Repository, 92, 348, 352*
*Server, 348*
Integration process, 353
Integration scenario, 353
Integrity, 33, 34
Interface, 13
*Agreement, 30*
*DestinationDataProvider, 222*
*Determination, 366, 369*
*IConnection, 265*
*IConnectorGatewayService, 265*
*IDocDocument, 303*
*IDocFactory, 303*
*IDocXMLProcessor, 307*
*IInteractionSpec, 266*
*IUser, 37*
*IUserMaint, 37*
*JCoCustomRepository, 245*
*JCoFunctionTemplate, 246*
*JCoIDocHandlerFactory, 308, 309*
*JCoIDocServer, 307*
*JCoListMetaData, 236, 246*
*JCoMetaData, 236*
*JCoRecord, 226*
*JCoServer, 247*
*JCoServerFunctionHandle, 249*
*JCoServerTIDHandler, 252, 308*
*Mapping, 354, 369*
*Object, 353*
*Pattern, 373*
*ServerDataProvider, 247*
*Technology, 70*
*Virtual <Pfeil>R<Normal> Virtual Interface (VI, 324*
Intermediate Document (IDoc), 271
Internet Connection Framework (ICF), 90
Internet Connection Manager (ICM), 22
ISO standard, 36
Iterator, 231

## J

J2EE, 29
*KeyStore service, 39*
JAAS, 50
Java
*Java 2 Enterprise Edition (J2EE), 29*
*Java API for XML Web Services (JAX-WS), 339*
*Java Architecture for XML Binding (JAXB), 339*
*Java Authentication and Authorization Service, 50*
*Java Connector Architecture, 256*
*Java Connector (JCo), 217*
*Java Cryptography Architecture, 38*
*Java Cryptography Extension (JCE), 38*
*Java Development Kit, 339*
*Java Enterprise Edition (JEE), 31*
*Java key store (JKS), 49*
*Java Message Service (JMS), 30*
*Java Naming and Directory Interface (JNDI), 253*
*Java Native Interface (JNI), 47*
*Java Server Faces (JSF), 337*
*Java Server Pages (JSP), 31*
*Library, 241*
*SOAP proxy, 383*

*Stack, 15, 29, 31*
*Web Dynpro, 69, 338*
*Web Start, 352*
*XI proxy, 360*
Java proxy
*Deployable, 337*
JAXB, 339
JAX-WS, 339
JCE, 38
JCo, 217, 302
*Architecture, 219*
*Factory class, 224*
*Installation, 217*
*RFC Provider Service, 253*
*Session management, 222*
JEE, 31
JKS, 49
JMS, 30
JNDI, 253, 362, 366
JNI, 47, 219
JSF, 337
JSP, 31, 69

## K

Key, 33
*Field, 60*
*Management, 35*
*Private, 33*
*Public, 33*
KeyStore, 38, 48
KeyStore content
*JKS, 38*
*PKCS12, 38*
KeyStore file
*verify.der, 39*

## L

LDAP, 36
Library
*SAPSSOEXT, 47*
Lifecycle, 30
Lightweight Directory Access Protocol (LDAP), 36
Linker, 152
Linking, 152
Load balancing, 16, 49, 73, 78, 149, 239
Load distribution, 23
Lock, 27, 104
*List, 35*
*Module, 63*
*Object, 63*
Lock management
*Work process, 23*
Login module, 51
Logon, 17
*Client, 19*
*Group, 23*
*Language, 17*
*Language (Logon language), 17*
*Ticket, 42, 49*
LUW, 27, 107, 114

## M

Management, 257
Mapping
*Object, 353*
*Program, 354*
Mass data, 67, 123
Mass updating, 67
Menu bar, 17
Message, 30, 100
*Class, 126*
*Interface, 92, 353, 354, 357*
*Mapping, 354*
*Number, 126*
*Server, 16, 23, 73, 149*
*Type, 87, 275, 292, 315, 354, 372*
Message-driven bean, 29
Message-Oriented Middleware (MOM), 30
Metadata, 236, 246, 303, 347
Method
*getFunctionTemplate, 236*
*JCoParameterList, 226*
Middleware

*Interface, 220*
*Server, 347*
MIME, 92
Mode, external, 19
Module pool, 69
MOM, 30
Multipurpose Internet Mail Extensions (MIME), 92

## N

Name collision, 319
NCO, 342
Non-repudiation, 33
Number range, 64

## O

Object
*JCoDestination, 221*
*JCoFunction, 249*
*JCoFunctionTemplate, 236*
*JCoIDocServerContext, 309*
*JCoServerContext, 249*
Object type, 119
One-way function, 34
Open SQL, 32
Operation, 373
Outbound queue, 112, 116, 373
Outside-in approach, 347

## P

Package, 58, 238
*Use access, 319*
Package Explorer, 262
PAI module, 26
PAM, 50
Parameter
*Export, 66, 158, 230, 242, 243, 266*
*IDOC_CONTROL_REC_40, 298*
*IDOC_DATA_REC_40, 298*
*Import, 66, 158, 226, 242, 243, 266*
*Scalar, 98*
*Structured, 98*
Partner profile, 286, 287, 288, 299
Pass by value, 98
Password, 17
Payload, 356
PBO module, 26
Performance, 31
Persistence, 30
Personal Security Environment (PSE), 40
PKI, 35
Pluggable Authentication Model (PAM), 50
Port
*Default, 320*
*Logical, 320, 336, 380*
*Number, 21*
Portal
*Application, 262*
*Component, 261, 262, 264*
*Project, 262*
Presentation layer, 14
Principal API, 36
Private-key cryptography, 33
Process After Input module (PAI module), 26
Process Before Output module (PBO module), 26
Process code, 289
Processing sequence, 80
Profile, 64
*Parameter, 43*
Program
*genh, 147, 167*
*rfcping, 146, 148*
*sapinfo, 146, 148*
*ZSSF_TEST_PSE, 57*
Program ID, 75, 250, 256
Protocol
*Handler, 325*
*Implementation, 325*
Provider, 376, 383
*Proxy, 385*
Proxy, 226, 303, 326
*Class, 238, 243, 266, 319, 336, 339, 342*

*Runtime, 363*
PSE, 40, 52
*Central, 52*
*File, 49*
*Individual, 53*
Public-key cryptography, 33
Public-key infrastructure (PKI), 35
Purchase order, 58

## Q

QIN Scheduler, 117, 189
*SMQR, 117*
QOUT Scheduler, 114
qRFC monitor
*For inbound queues, 116*
*For outbound queues, 114*
Quality of service, 92
Queue, 234
*Name, 188*

## R

Receiver
*Agreement, 367, 370*
*Determination, 366, 368*
Receiver channel, 366, 367
References project, 327
Registration, 193
Relationship of trust, 47, 48
Remote Function Call (RFC), 21
Remote interface, 366
Report, 69
Repository, 235, 253
*Server-side, 245*
Repudiation, 34
Resource adapter, 257
Resources
*Consumption, 27*
Responsibility, 30
RFC, 21, 72, 95
*Asynchronous, 78, 117*
*Client, 154, 220*
*Data types, 156, 227*
*Debugging, 150*
*Destination, 73*
*Error handling, 162*
*For changing, 101*
*For creating, 107*
*For deleting, 101*
*Function module, 95*
*Handle, 196*
*Parameter, 164, 167*
*Queued, 80, 111, 188, 234*
*Registered server, 75, 189, 250*
*Return code, 157*
*Server, 189, 244*
*Synchronous, 77, 95, 106, 225*
*Table parameter, 171*
*Tracing, 82, 150*
*Transactional, 79, 106, 109, 177, 205, 232*
*Type handle, 170*
*Unicode library, 157*
*With C, 145*
*With Java, 217*
RFC Software Development Kit, 146
Roll area, 23
Route, 24
Router string, 25

## S

Sales order, 58
Sample application, 57
SAP
*Gateway, 149*
*IDoc library, 302*
*Instance, 73*
*menu, 19*
*Router, 24, 149*
*Router string, 239*
*SAP CRM, 14*
*SAP Cryptolib, 53*
*SAP Crypto Toolkit, 46*
*SAP ERP, 14*
*SAP GUI, 15, 16, 20, 24*
*SAP Implementation Guide, 88*
*SAP Java Connector (JCo), 217*
*SAP Logical Unit of Work (SAP LUW), 27*

*SAP logon ticket, 39, 46*
*SAP NetWeaver Administrator, 39, 387*
*SAP NetWeaver AS, 13*
*SAP NetWeaver AS ABAP, 16, 57*
*SAP NetWeaver AS Java, 29*
*SAP NetWeaver BW, 14*
*SAP NetWeaver Composition Environment (CE), 31, 339*
*SAP NetWeaver Developer Studio (SAP NWDS), 261*
*SAP NetWeaver Exchange Infrastructure (SAP XI), 14*
*SAP NetWeaver Master Data Management, 14*
*SAP NetWeaver Mobile, 14*
*SAP NetWeaver PI, 92, 347, 372*
*SAP NetWeaver Portal, 14, 42, 258*
*SAP NetWeaver Portal Connector Framework, 256*
*SAP NetWeaver XI, 14, 92, 347*
*SAP NWDS, 13, 218, 238, 261, 323*
*SAP PLM, 14*
*SAP SCM, 14*
*SAP SRM, 14*
*System, 20*
Scheduler
*QIN, 117, 189*
*QOUT, 114*
Screen, 19, 26
Search help, 61, 62
Secure Network Communication (SNC), 51
Secure Socket Layer (SSL), 34
Security, 32
*Access, 33*
*Data, 33*
*Infrastructural, 32*
*Integration, 46*
*Management, 257*
*System, 33*
*Transport layer, 51*
SEI, 326
Selection, 137
Selection method, 61
Self-registration, 37
Separation of concerns, 29
Server proxy, 356, 360, 377
Service Endpoint Interface (SEI), 326
Service interface, 372, 373
Service-oriented architecture (SOA), 31
Service-Oriented Architecture Protocol (SOAP), 90
Service Provider Interface (SPI), 30
Services file, 21
Servlet, 31, 69
Session bean
*Stateful, 29*
*Stateless, 29, 254, 328*
Session management (JCo), 31
SID, 20, 149
Single point of access, 42
Single sign-on (SSO), 41
Single stack, 14
Size category, 61
SLD, 258, 348, 349, 372
SNC, 51
SNC PSE container, 55
SOA, 31
SOAP, 90, 311
*Action, 316*
*Body, 317*
*Document, 92*
*Envelope, 317*
*Header, 92, 317*
Software
*Component, 350, 352*
*Product, 349*
SPI, 30
Spool
*Work process, 22*
SSL, 34, 51
SSO, 41
Standalone Java proxies, 334
Standard
*BAPI, 84*
*Destination, 100*
*Toolbar, 18*
Stateful, 223
Stateless, 29, 254, 328, 373, 374
Statement
*AUTHORITY CHECK, 80*
*CALL FUNCTION, 64*
*CALL TRANSACTION, 70*

*COMMIT WORK, 27, 28, 67, 107*
*DELETE, 28*
*INSERT, 28*
*ROLLBACK WORK, 27, 28, 67, 106*
*UPDATE, 28*
Status bar, 19
Structure parameter, 230
Struts, 337
Symmetric procedure, 33
System
*External, 74*
*Logical, 86, 285, 286, 299, 310*
*Message, 19*
*Number, 20, 73, 149*
*PSE, 45*
*Security, 33*
*Technical, 351*
System ID (SID), 149
System Landscape, 258
*Editor, 260*
System Landscape Directory (SLD), 258

## T

Table
*ARFCSDATA, 106*
*ARFCSSTATE, 106*
*BAPIF4T, 141*
*Maintenance, 60*
*Parameter, 66, 159, 228, 230, 242*
*T100, 122, 129*
*TBDBE, 294*
*Transparent, 58*
*TWPSSO2ACL, 43*
TCP, 20
Ticket issuer, 42
TID, 79, 106, 177, 188, 208, 232, 305
*Management, 178, 206, 233, 251, 308*
Title bar, 18
Tool
*Keytool, 39*
*sapgenpse, 40, 53*
*wsgen, 339*
*wsimport, 339*
Transaction, 18, 19
*BD51, 283*
*BD64, 299*
*BD87, 291*
*BDBG, 292, 294*
*LPCONFIG, 320*
*Management, 257*
*RZ10, 43, 54*
*SA16, 60*
*SALE, 86, 88, 284, 299, 310*
*SE11, 58*
*SE16, 60*
*SE37, 65, 77*
*SE80, 58*
*SICF, 90*
*SM30, 43, 141*
*SM35, 71*
*SM58, 106*
*SM59, 73, 98, 193, 215, 250, 286, 287, 310, 320*
*SMGW, 81, 193*
*SMQ1, 114*
*SMQ2, 116, 234*
*SMQS, 114*
*SNRO, 64*
*SOAMANAGER, 378, 380*
*SPRO, 88*
*SPROXY, 92, 357, 358, 376*
*SSO2, 44*
*SSTRUSTSS02, 45*
*ST05, 82*
*ST22, 81*
*STRUSTSSO2, 40, 43, 56*
*SU01, 63*
*SU02, 64*
*SWO, 82*
*SWO1, 82, 118, 127*
*SXMB_IFR, 349, 367*
*SXMB_MONITOR, 371*
*WE20, 288, 310*
*WE30, 87, 273*
*WE31, 87, 272*
*WE42, 289*
*WE81, 86, 275*
*WE82, 275*
*WSADMIN, 91, 314, 316, 334*
*WSCONFIG, 91, 314*
Transaction code, 19

Transaction Control Protocol (TCP), 20
Transaction ID (TID), 79
Transaction model, 174
Transaction
  *SE80, 376*
Transport binding, 325
Transport object, 325
Trust, 35
Type handle, 170, 196

## U

UDDI, 319
UME, 36
UME Persistence Manager, 36
Universal Description and Discovery Interface Service, 319
UNIX daemon, 76
Update
  *Module, 67, 103*
  *Table, 28*
  *Work process, 23, 28*
User, 17
  *Context, 19*
  *Interface, 68*
  *Master record, 17*
  *Name, 17, 19*
  *SAPJSF, 36*
  *Session, 16, 19*
User datastore, 37
User Management Engine (UME), 36
User mapping, 46, 260

## V

VA, 39, 253
Variable
  *SECUDIR, 54*
Vendor purchase order, 58
Virtual Interface (VI), 324, 330, 331
Visual Administrator (VA), 39

## W

Web container, 31
Web Dynpro
  *ABAP, 69*
  *Java, 31, 69, 338*
Web module project, 337
Web service, 311, 378, 387
  *Adaptive, 334*
  *Binding, 324*
  *Endpoint, 312*
  *Home page, 91*
  *Interface, 313, 324*
  *Operation, 312*
  *Runtime, 325*
  *With J2EE, 326, 330*
  *With Java, 323*
  *Wizard, 312*
Web service client, 389
  *With C#, 342*
  *With Java, 326, 333, 340*
Web Service Definition (WSD), 324
Web Service Description Language (WSDL), 91
Web Service Navigator, 331, 378
Windows Service, 76
Work process, 22
WSD, 324, 331
WSDL, 91, 314, 320, 334, 343, 379, 381
  *Binding, 325*
  *Document style, 316*
  *RPC style, 316*

## X

X.500 (ISO standard), 36
XI SOAP, 92, 356
XML, 306
  *Message, 311*
  *Schema Definition Language (XSD), 315*
XSLT, 306, 354