
Book 8

Development in R/3 ABAP

5

ABAP is the intrinsic programming language of R/3. It is a fourth generation language comparable to the one of dBase. It has full object support since the R/3 4.6 release, however there had been limited object support in early releases as well.

U:\Book\Book_08.doc

Development in R/3 ABAP

10

ABAP is the intrinsic programming language of R/3. It is a fourth generation language comparable to the one of dBase. It has full object support since the R/3 4.6 release, however there had been limited object support in early releases as well.

What to Read in This Part

Development in R/3 ABAP	1
1 The SAP R/3 ABAP Objects Workbench	3
1.1 Getting Familiar With the R/3 SAPGUI	4
1.2 Hierarchy of Development Objects In R/3	7
1.3 SE80: Creating A New Development Class	13
1.4 SE38: Creating ABAPs.....	16
1.5 SE80: Creating A New Function Group	17
1.6 SE37: Creating A New Function Module	20
1.7 SWO1: Creating A New Object Class in R/3	23
1.8 SE10: Change Requests.....	24
2 First Steps Into ABAP IV	26
2.1 Hello World ABAP	27
2.2 Using Parameters.....	29
2.3 Select-Options	31
2.4 SQL-Selects	34
2.5 Internal Tables – Recordsets in ABAP	36
2.6 Function Modules	38
3 SAP R/3 Business Suite And R/3 Data Model Views	39
3.1 About Model Views.....	40
3.2 Important R/3 Data Objects.....	42
3.3 A Sample Business Scenario.....	43

3.4	RFC Functions, BAPIs or IDocs	44
3.5	SAP Sample Database: FLIGHT	46
4	Examples of ABAP Function Modules	47
4.1	FUNCTION Y_DEMO_FLIGHT_LOADDATA	48
4.2	FUNCTION Y_DEMO_IDOC_GENERATE_ORDERS	49
5	Batch Input Recording	52
5.1	Recording a Transaction With SHDB	53

15

Fehler! Es wurden keine Einträge für das Inhaltsverzeichnis gefunden.

Development in R/3 ABAP 3

1 The SAP R/3 ABAP Objects Workbench

The Interactive Development Environment (IDE) of R/3 is called the ABAP Workbench. This chapter shall give you a quick introduction to the concept. It is meant to give you an initial feeling for the workbench and to be a stepping stone to further dedicated ABAP training material or to self learning using the ABAP online help.

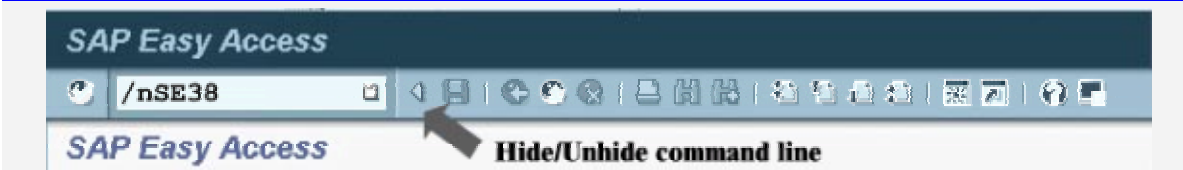
1.1 Getting Familiar With the R/3 SAPGUI

The SAPGUI is the SAP R/3 user interface. It is a specially designed browser client to access R/3 transactions. This chapter demonstrates the most important ways to navigate through R/3 and tells you how to call transactions (programs) and how to find interesting technical details about them.

Every SAP screen has a command window where transaction codes can be entered

After you have successfully logged into your R/3 system you will see the session screen. On the top left corner of your session screen there is a command window. Starting with SAPGUI release 4.6 this command line window can be made visible and invisible by clicking the little arrow next to it, while in earlier releases of the SAPGUI the command line windows has always been visible.

Figure 1: Sample R/3 command line



⇒

Every transaction has a transaction code assigned

R/3 is a transaction based system. The term transaction is to be understood in a very broad sense. Generally you can see a transaction as a synonym of a program run by R/3. The command window is used to enter shortcut codes to any transaction within R/3. If you enter `/nSE38` you will jump directly to the ABAP editor, `/nSE80` will lead you to the ABAP workbench – R/3's IDE –, `/nSE16` will display the database table explorer and `/nVA02` lets you edit a sales order.

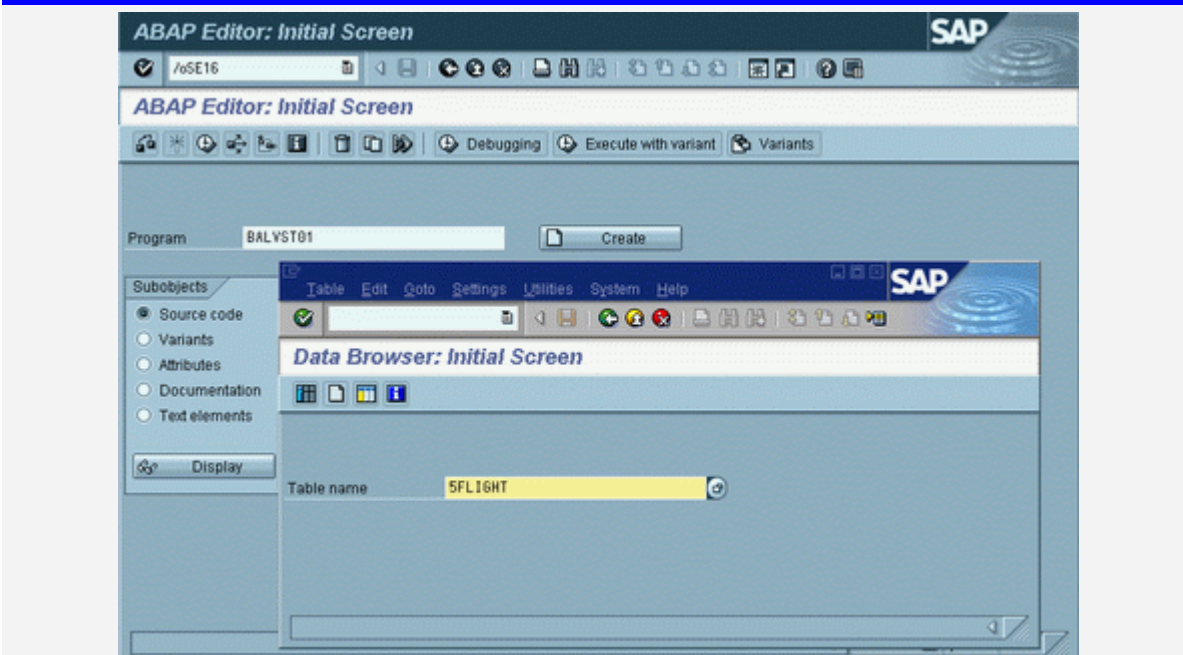
30

The prefix `/n` is used to call a transaction in same window, `/o` opens a parallel session

Entering `/nSE38` opens the transaction SE38 (ABAP editor) in the same window. If you want to call the transaction in a parallel window then you would have to replace the prefix `/n` with a `/o`, so the code would be `/oSE38`. The `/o` stands traditionally for the word Mode which was the old word for session used by R/2 and the IBM CICS operating system.

35

Figure 2: Opening a parallel modus (session) in R/3 using the `/o` command – a second window pops up



Development in R/3 ABAP 5

⇒

All menu selections are translated into transaction codes

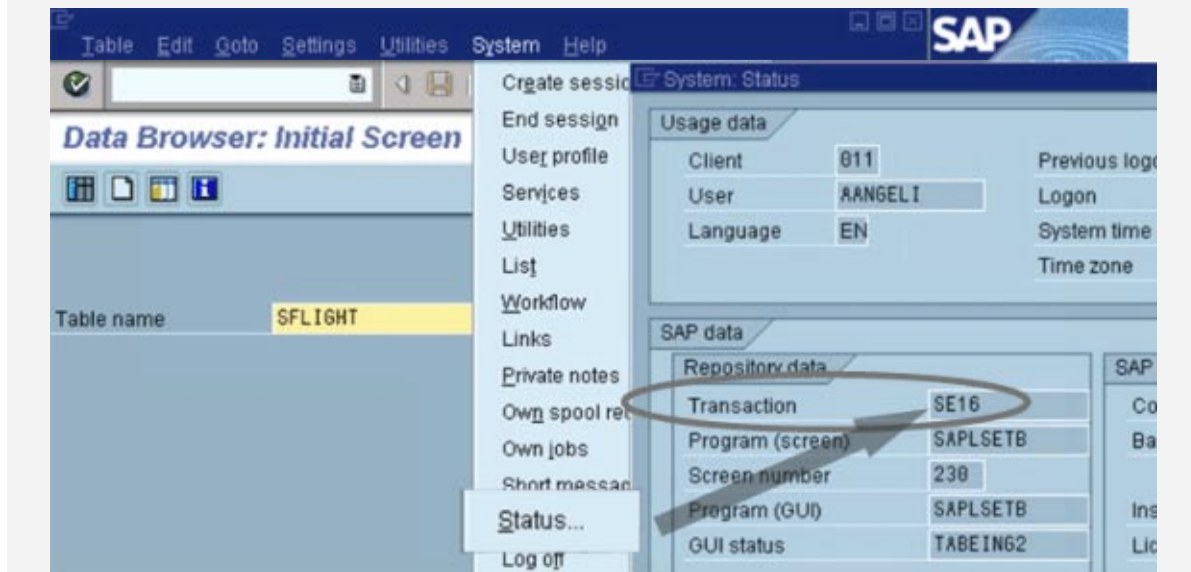
All this functionality is also accessible through the menu forest displayed but all transactions have a unique transaction code assigned which can be used as a shortcut via the command window. In fact choosing a function from the menu does nothing more than simulate the entry of a transaction code in the command line.

40

Session information can be displayed from the SYSTEM -> STATUS menu

There are several ways to find the transaction code. If you are already in the transaction, you can look at the menu option SYSTEM -> STATUS where among other information the transaction code is displayed.

Figure 3: Session information can be displayed from the SYSTEM -> STATUS menu

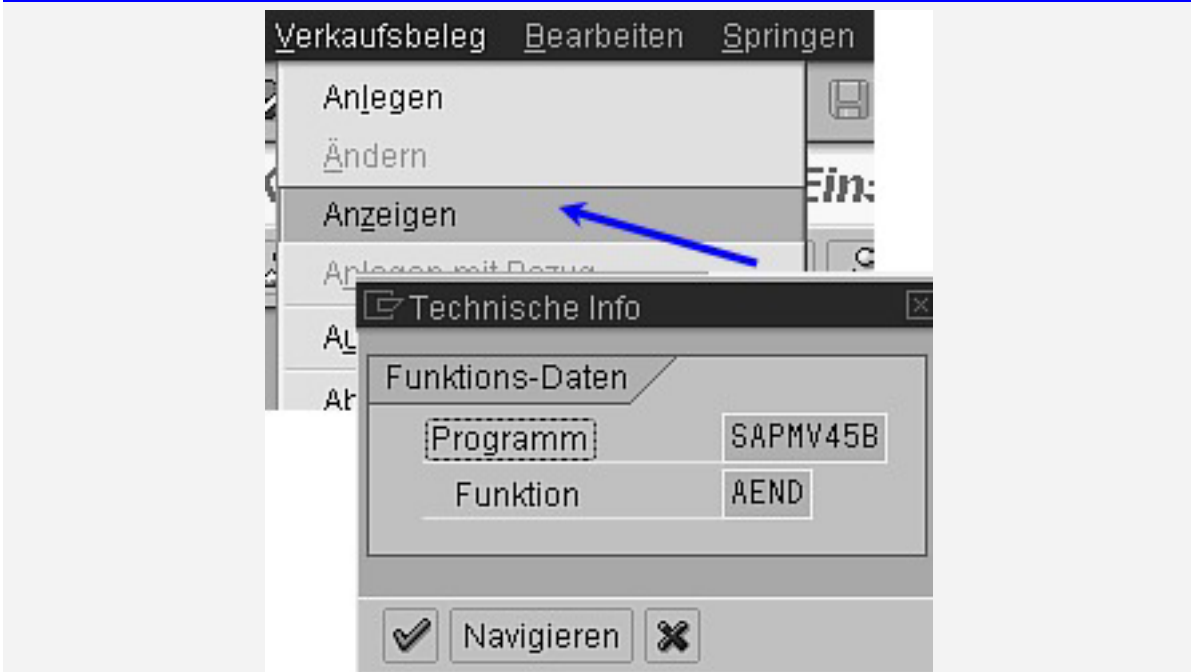


⇒

Action codes are shown by pressing F1 and mouse down while the mouse is over the menu entry

You can also see the transaction code directly by holding the mouse button down and pressing F1 while pointing to a menu entry. This will open a help window with the information about the menu entry. Please note that most codes associated with a menu entry are actually not transaction codes but simple action codes that are evaluated inside the active transaction.

Figure 4: Pressing F1 and mouse down simultaneously displays the actual command associated with the menu entry

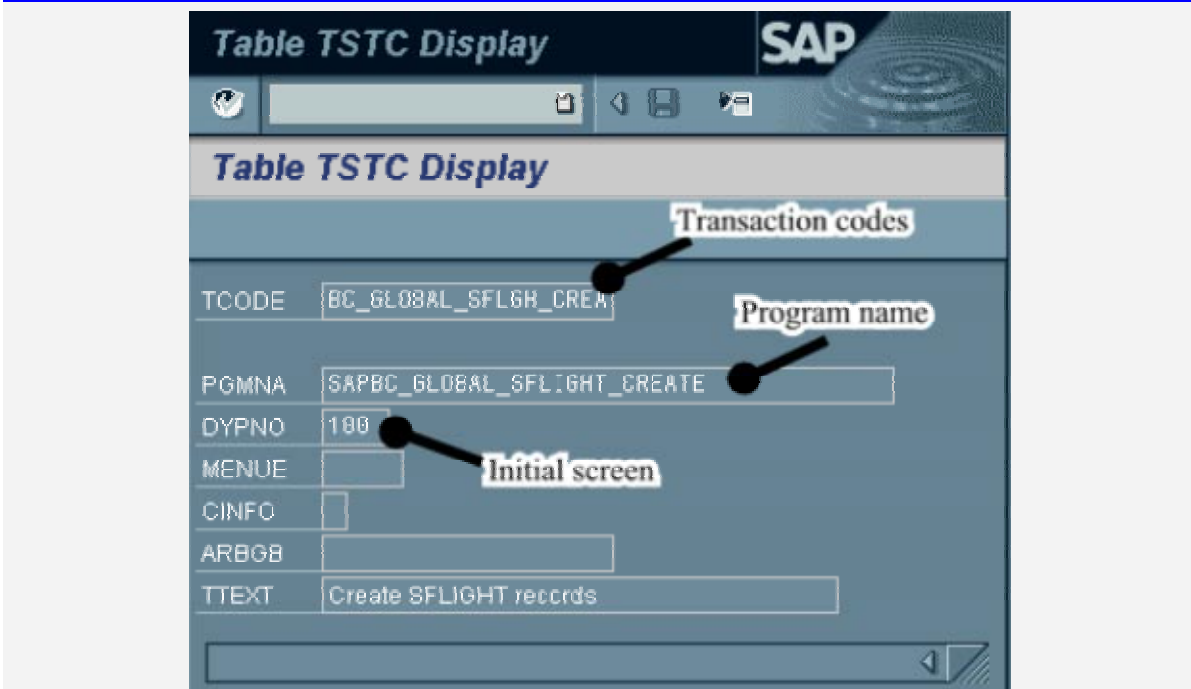


⇒

Transaction codes are linked to programs

Transactions are associated with programs in the database table TSTC. This table holds one entry per transaction code and the name of the program plus eventual calling instructions. You can view the content of TSTC like of any other flat database table with transaction SE16.

Figure 5: Transaction code as assigned in table TSTC

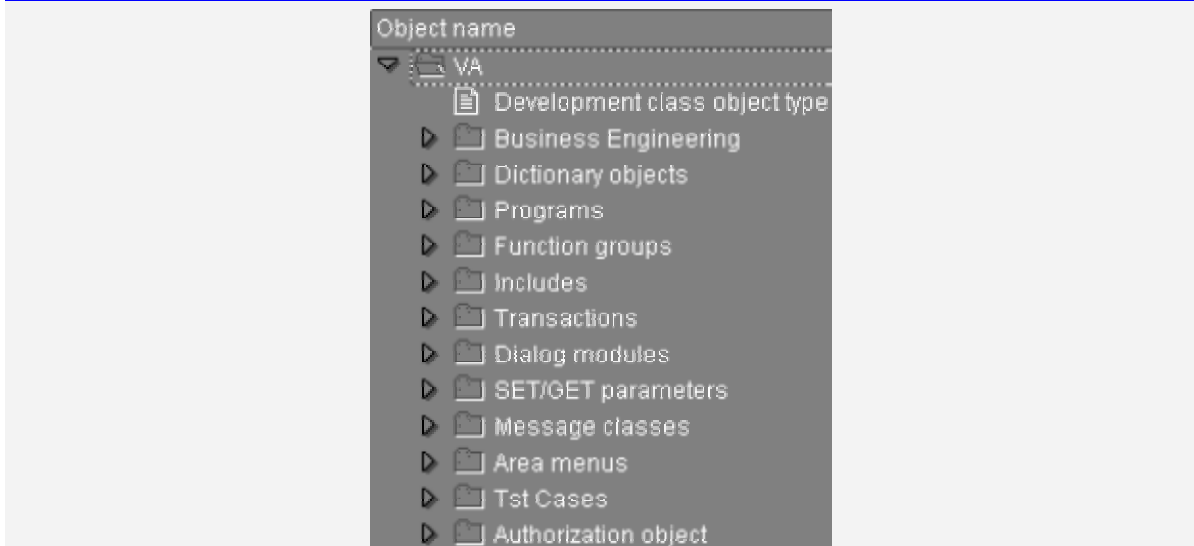


⇒

1.2 Hierarchy of Development Objects In R/3

Development objects in R/3 like programs, function pools, database tables and views are organised in an object hierarchy with the development class on top, which is the R/3 equivalent of a Visual Basic, C++ or Delphi project.

Figure 6: Hierarchy of development objects in R/3



⇒

Development Classes

Development classes are the ABAP project groups that are only used to group the development objects into categories. Development objects should be created by the development team.

On top of the hierarchy of development objects in R/3 are the development classes. They are simple predefined name tags which are assigned to any development object. Every object in development has one and only one development class assignment as the primary categorisation attribute.

Although it is a widespread habit that development classes are created by systems administrators, according to something they might call a plan, it is more prudent in my view to leave this task to the development team. These classes are meant to contain a collection of development objects, organised in such a way as to make finding them easy and logical. Because of this purely organisational character of a development class, it is wise not to put too many objects into the same class. Only objects which belong together should be collected in the same class. It is preferable to create a number of classes, perhaps with similar names, than to have one big overcrowded class.

Developer Keys

Developer key is required to modify objects

Before you can create or modify development objects in SAP you need to be registered with SAP as a developer for the system on which you are working. For that purpose your system administrator will request a *developer key* from SAP via SAPNET. Without such a key you will not be allowed to modify any development or data dictionary object in R/3. The purpose of the developer key is mainly as a registration tool, because the annual license fees of an R/3 installation are partly dependent on the number of active developers on the installation.

60

65

70

Programs

Every transaction has a transaction code assigned

R/3 programs are called reports or ABAPs and are the most simple form of development objects. If you enter /nSE38 you will jump directly to the ABAP editor.

Subroutine Program Pools

Subroutine libraries in R/3 are called Subroutine pools. Principally they are ordinary programs with the one exception, that they have no main program to execute. The routines in the pool are rather designed to be called by other program elements.

Module Pools

Module pools are a variant of the subroutine pool. In addition to subroutines, they also contain input forms (called *dynpro* or *screen*). Like subroutine pools, a module pool cannot be called directly. Rather, one of the stored dynpro screen is assigned a transaction code and is called by requesting the respective transaction code either from the command line or from a menu entry.

Function Groups

Functions are library sub routines

Functions in ABAP are principally the same thing as in any other programming language: callable subroutines which are stored in libraries and are accessible to any other program.

Function groups are libraries of functions

R/3 function groups are the libraries for R/3 functions. Every R/3 function is member of exactly one function group. In addition to functions every function group contains a global data segment which can hold declarations of variables, which can be shared between all the functions of the same function group. Principally you can move functions from one function group to another.

Data Dictionary Objects

Database objects like tables, views, data elements and domains are stored in the data dictionary

As a database driven application system, SAP R/3 depends very heavily on the quality of its data dictionary. Every database table and its characteristics are catalogued in the data dictionary. Transaction SE11 calls the editor for the database schemes.

Figure 7: Data dictionary objects seen by the workbench browser (SE80)



⇒

Data Dictionary Element	Description
Database table	Physical table in the R/3 database
View	SQL View

Development in R/3 ABAP 9

Data Type	A Domain with labels
Domain	Globally defined custom data type used to define data base tables
Search help	Views on database used to display data entry help in interactive applications
Lock object	Defines database record locks



Data dictionary is a collection of database tables

The data dictionary is itself a collection of R/3 database tables that can be accessed from an ABAP program like any other table. You can view the content of the data dictionary tables with transaction SE80.

Figure 8: Names of data dictionary tables

Development in R/3 ABAP 11

ServerVariable	Description	Example
ALL_HTTP	All HTTP headers sent by the client to the server in the current HTTP request	{HTTP_ACCEPT:*/ HTTP_ACCEPT_LANGUAGE:en-gb HTTP_CONNECTION:Keep-Alive HTTP_HOST:localhost HTTP_USER_AGENT:Mozilla/4.0 (compatible; MSIE 6.0; Windows 98) HTTP_COOKIE:ASPSESSIONIDFFFKTHD=NFNBMIBLBOCDIIGIHACDIPB }
ALL_RAW	Retrieves all headers in raw form. The difference between ALL_RAW and ALL_HTTP is that ALL_HTTP places an HTTP_ prefix before the header name and the header name is always capitalized. In ALL_RAW the header name and values appear as they are sent by the client.	{Accept: */* Accept-Language: en-gb Connection: Keep-Alive Host: localhost User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows 98) Cookie: ASPSESSIONIDFFFKTHD=NFNBMIBLBOCDIIGIHACDIPB Accept-Encoding: gzip, deflate Extension: Security/Remote-Password }
APPL_MD_PATH	Retrieves the virtual IIS path (Metabase Directory) for the Application for the ISAPI DLL.	{/LM/W3SVC/1/ROOT/test}
APPL_PHYSICAL_PATH	Retrieves the physical path corresponding to the virtual IIS path (Metabase Directory). IIS converts the APPL_MD_PATH to the physical (directory) path to return this value.	{U:\Tutorial\webproject\}
AUTH_PASSWORD	The value entered in the client's authentication dialog. This variable is available only if Basic authentication is used.	{}
AUTH_TYPE	The authentication method that the server uses to validate users when they attempt to access a protected script.	{}
AUTH_USER	Raw authenticated user name.	{}
CERT_COOKIE	Unique ID for client certificate, returned as a string. Can be used as a signature for the whole client certificate.	{}
CERT_FLAGS	bit0 is set to 1 if the client certificate is present. bit1 is set to 1 if the cCertification authority of the client certificate is invalid (it is not in the list of recognized CAs on the server).	{}
CERT_ISSUER	Issuer field of the client certificate (O=MS, OU=IAS, CN=user name, C=USA).	{}
CERT_KEYSIZE	Number of bits in Secure Sockets Layer connection key size. For example, 128.	{}
CERT_SECRETKEYSIZE	Number of bits in server certificate private key. For example, 1024.	{}
CERT_SERIALNUMBER	Serial number field of the client certificate.	{}
CERT_SERVER_ISSUER	Issuer field of the server certificate.	{}
CERT_SERVER_SUBJECT	Subject field of the server certificate.	{}
CERT_SUBJECT	Subject field of the client certificate.	{}
CONTENT_LENGTH	The length of the content as given by the client.	{0}
CONTENT_TYPE	The data type of the content. Used with queries that have attached information,	{}



R/3 tables ← Columns ←
Data elements ←
Domains

R/3 tables are defined by one or more columns. Every table column is a simple (non-structured) data type, which is defined as a data element. A data element is actually a variation of a domain. Every data element is defined by a domain plus a set of label texts that can be used in different contexts.

Example

105

E.g. you can have a domain called CHAR10 defining a field of ten characters. Then you can define data elements e.g. ORDERNUMBER based on CHAR10 to define a sales order number and a data element ACCOUNTNO to define an account number. Both data elements are physically compatible, so you can assign a variable A defined as CHAR10 to a variable B defined as ORDERNUMBER and also B to A. However, when displayed on a screen, the field is shown with different i.e. contextually appropriate label tags.

110

1.3 SE80: Creating A New Development Class

R/3 classifies programs, DDic objects and other development objects in development classes. They are a simple grouping attribute to get an easy overview of related objects. They serve the same purpose as folders do in other tree-like organised systems.

Development classes are like folders

Development classes are a simple grouping attribute which is attached to any repository object. They serve the same purpose as folders do in email systems or subdirectories in DOS, Windows or UNIX.

Tool for developers, not a tool for administrator

Development classes are a tool to assist a developer to organise and find his objects by packing them into groups. Development classes should be created by the developer who uses it. We want to discourage you from sharing development classes between developers, unless they really work very closely together or the objects are interdisciplinary. Development classes have not been designed for the system administrator to monitor and supervise work. There are other and better means for security administration.

120

Stored as TRDIR-DEVCLASS

Development classes are a column DEVCLASS in the central attributes table TADIR, which is the central registry for all repository objects. Check table TRDIR with SE16 to see how your development object is recorded there.

Create and maintain development classes in SE80

A development class can be created with SE80. If you type the name of a non existing development class you may be automatically asked whether you want to create it.

Creation Strip

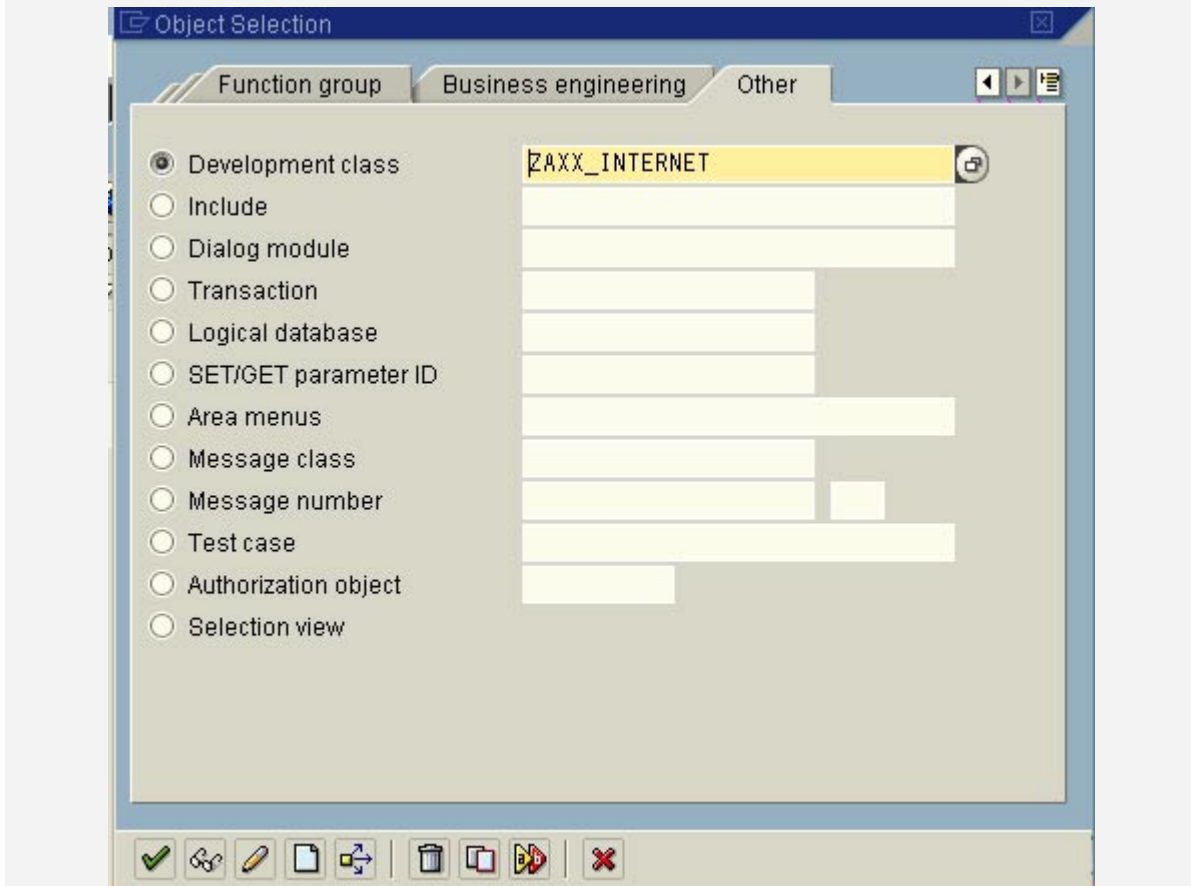
130

The following screen snap shot strip demonstrates an exemplary creation of a development class.

Figure 9: Choose the "New Object" creation dialog from the SE80 workbench



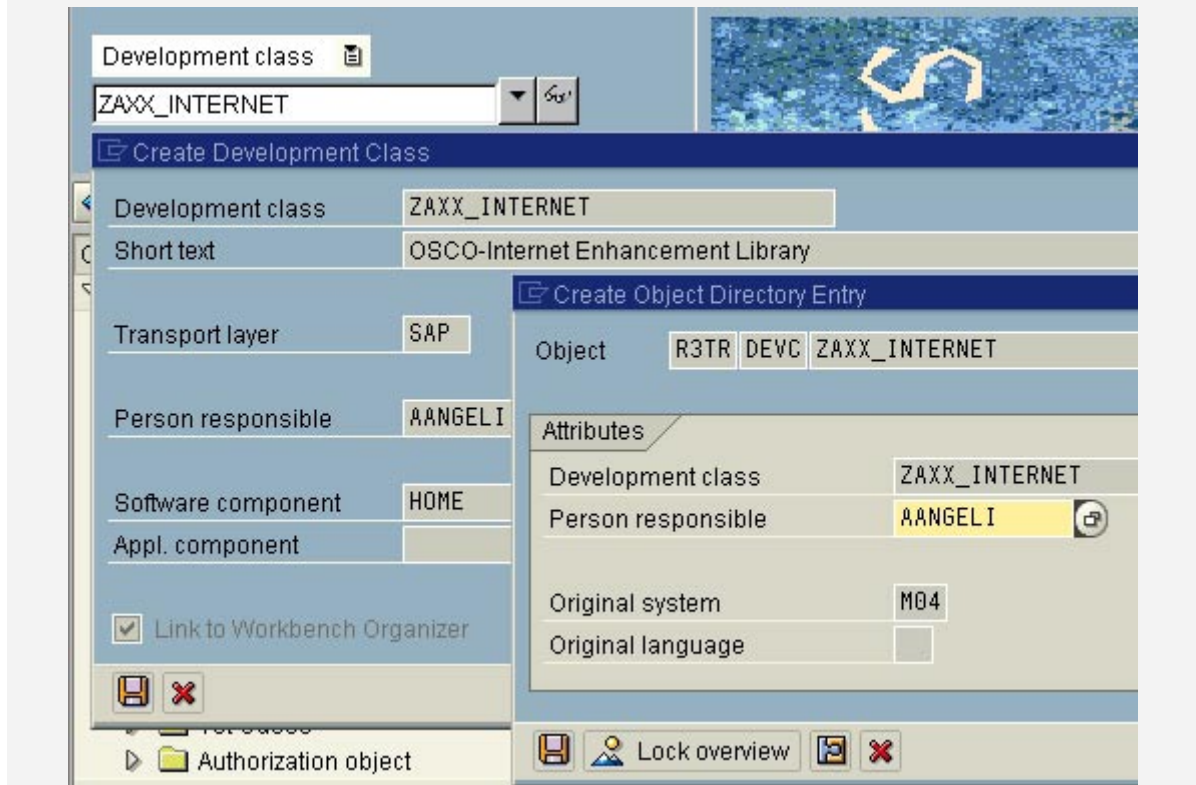
⇒

Figure 10: Enter the name of the new development class

Before saving a development you will be asked details about the object directory. The proposed values are usually correct. If not consult your SAP basis administrator for information about the requested details.

Development in R/3 ABAP 15

Figure 11: The object directory data is usually automatically proposed

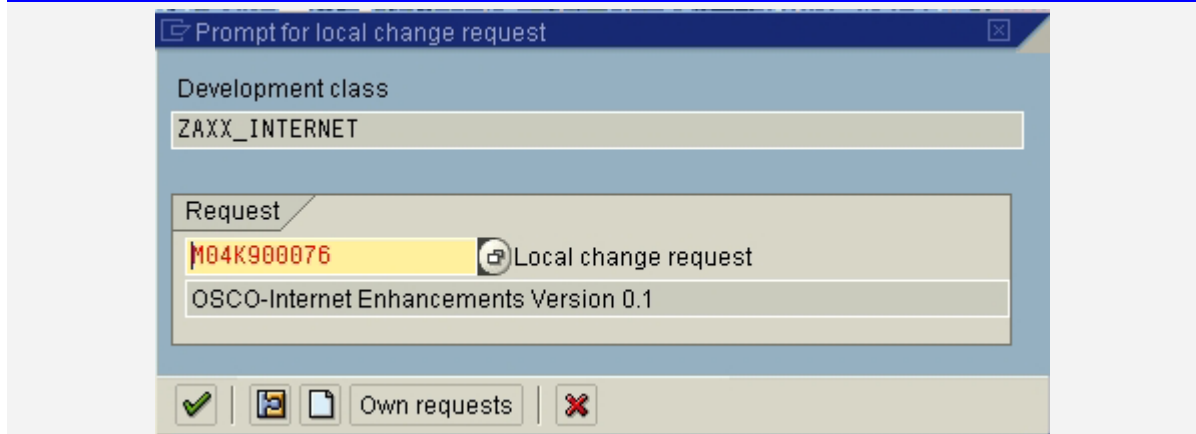


⇒

135

Then you may be asked to enter a transport request. These are lists of objects that have been modified in the system and are maintained forcibly by R/3.

Figure 12: Enter the name of a change request and create one if necessary



⇒

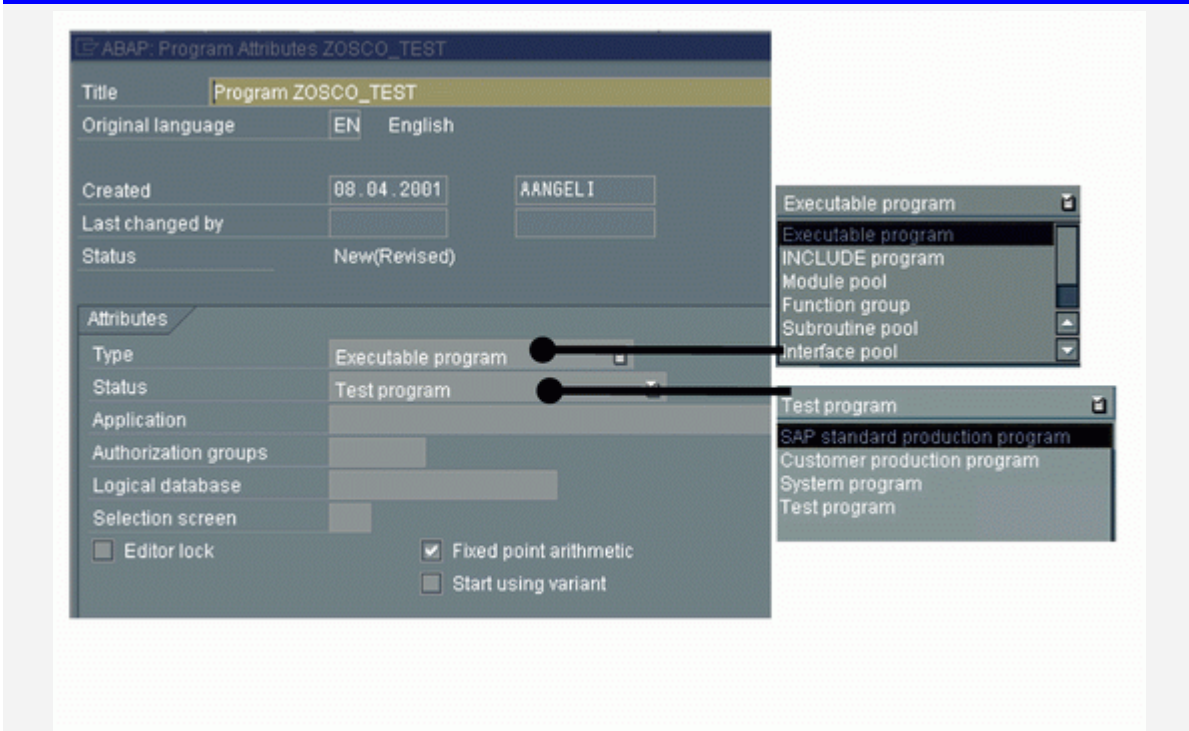
1.4 SE38: Creating ABAPs

ABAPs or reports are the programs in R/3. They are edited with the ABAP editor, transaction SE38.

When you create a new report you have to enter some program attributes. Generally you would choose to create an executable program. The status of the program can be set to an appropriate value. These attributes are used to classify the created programs to facilitate retrieval later.

140

Figure 13: Setting the program attributes



1.5 SE80: Creating A New Function Group

Function groups are libraries where ABAP functions are stored.

145

Before you can create a function in ABAP you have to create a library to store it in. This library is actually an ABAP code frame. The name of the ABAP frame is SAPL+the name you give to the library. E.g. if your library will be ZAXX_INTERNET then R/3 generates an ABAP with the name SAPLZAXX_INTERNET. This ABAP can be examined with SE38 like any other ABAP, however is protected from direct editing, thus forcing the developer to access the function code through the SE37 transaction.

Strip: Create a function group

150

This is a screen sequence to create the function group ZAXX_INTERNET with the transaction SE80.

Figure 14: Create a new function group with the new object menu entry

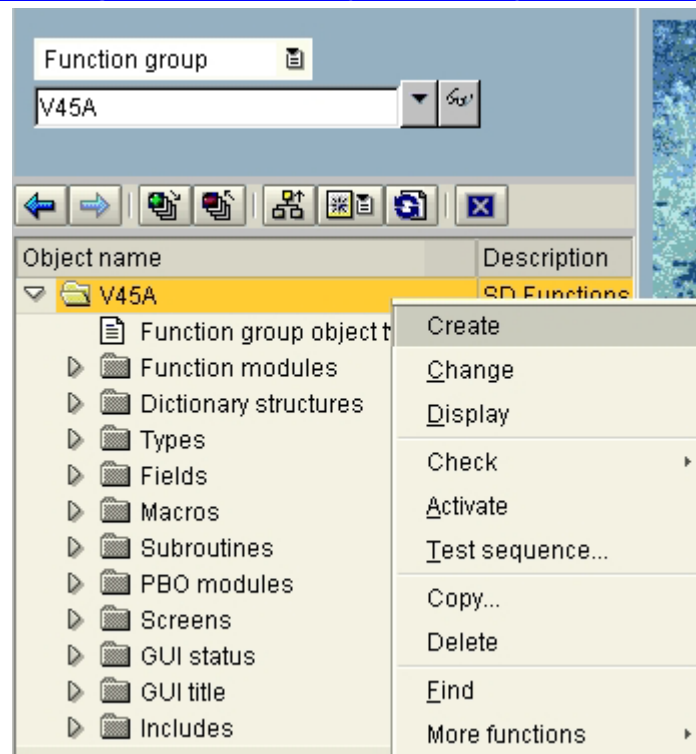
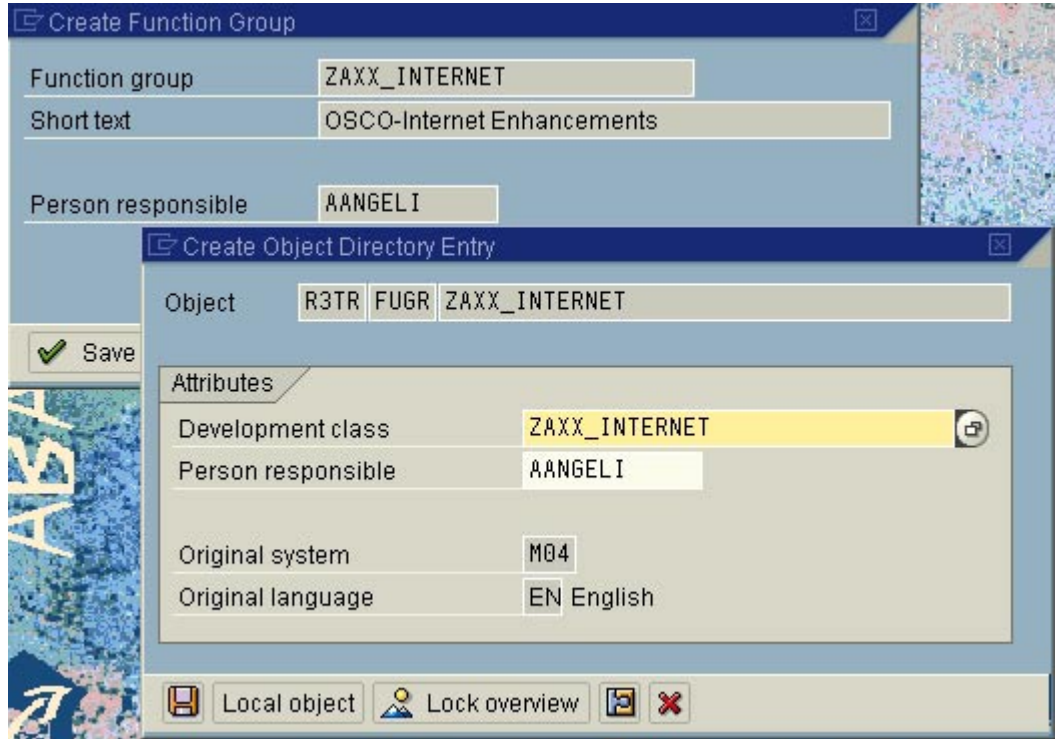
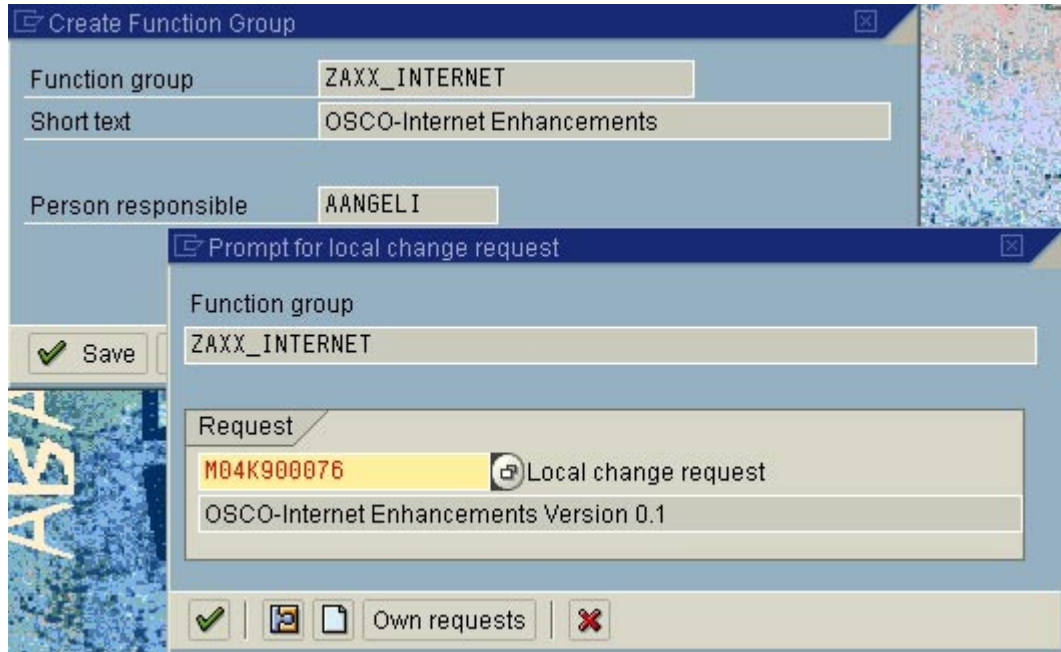


Figure 15: Assign a development class to the new object



→

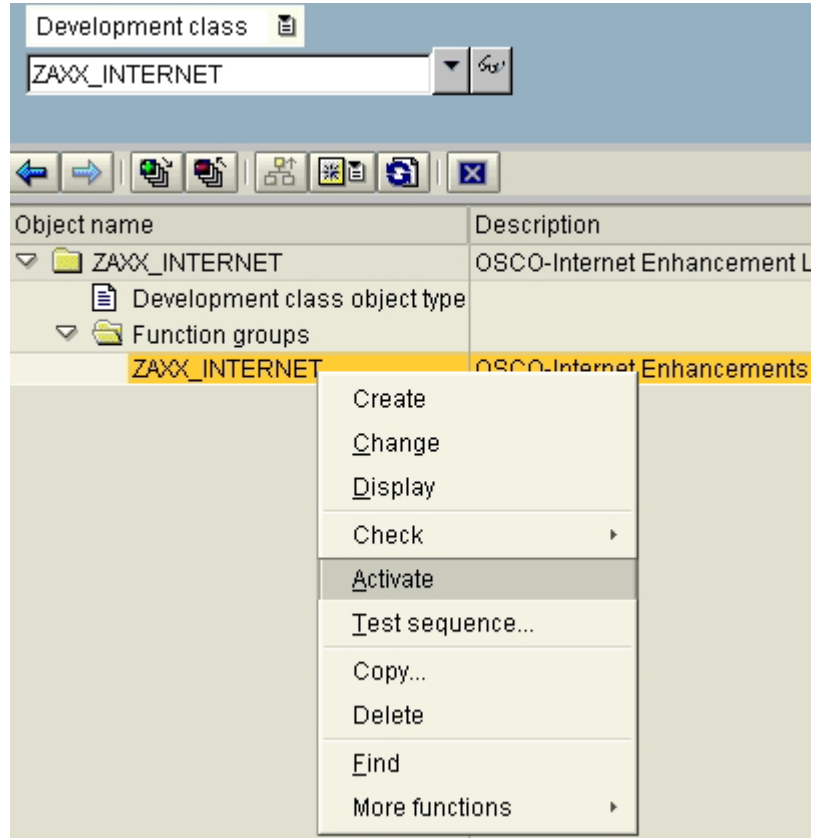
Figure 16: Every modified object needs to be assigned to a transport request



→

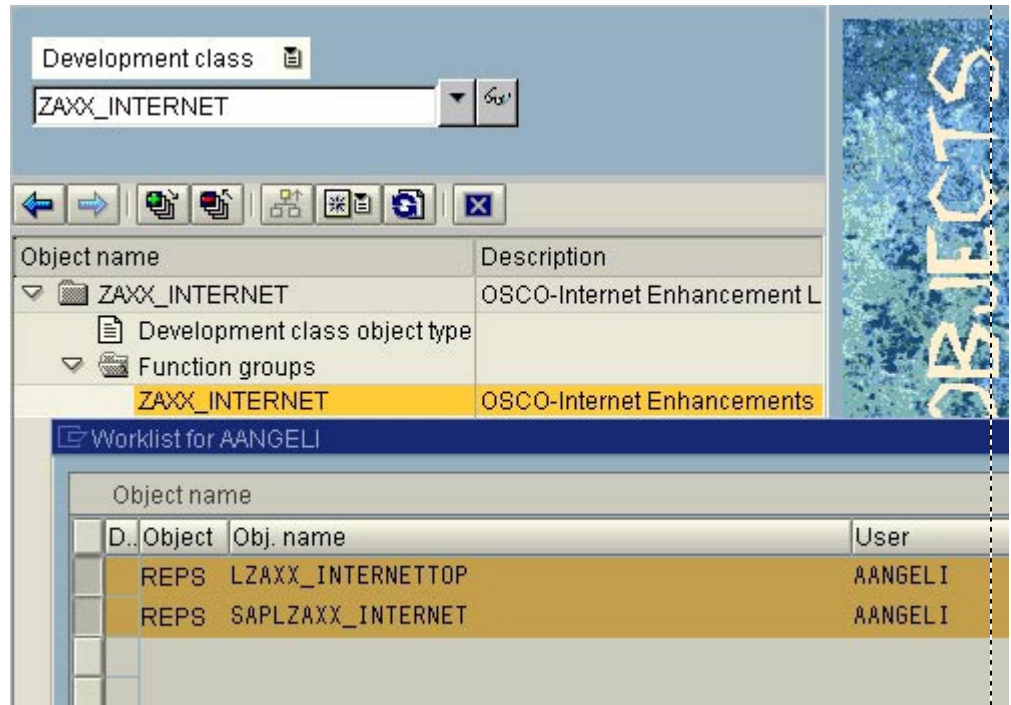
Development in R/3 ABAP 19

Figure 17: The new function group needs to be activated



⇒

Figure 18: For activation choose the relevant or all object from the selection panel



⇒

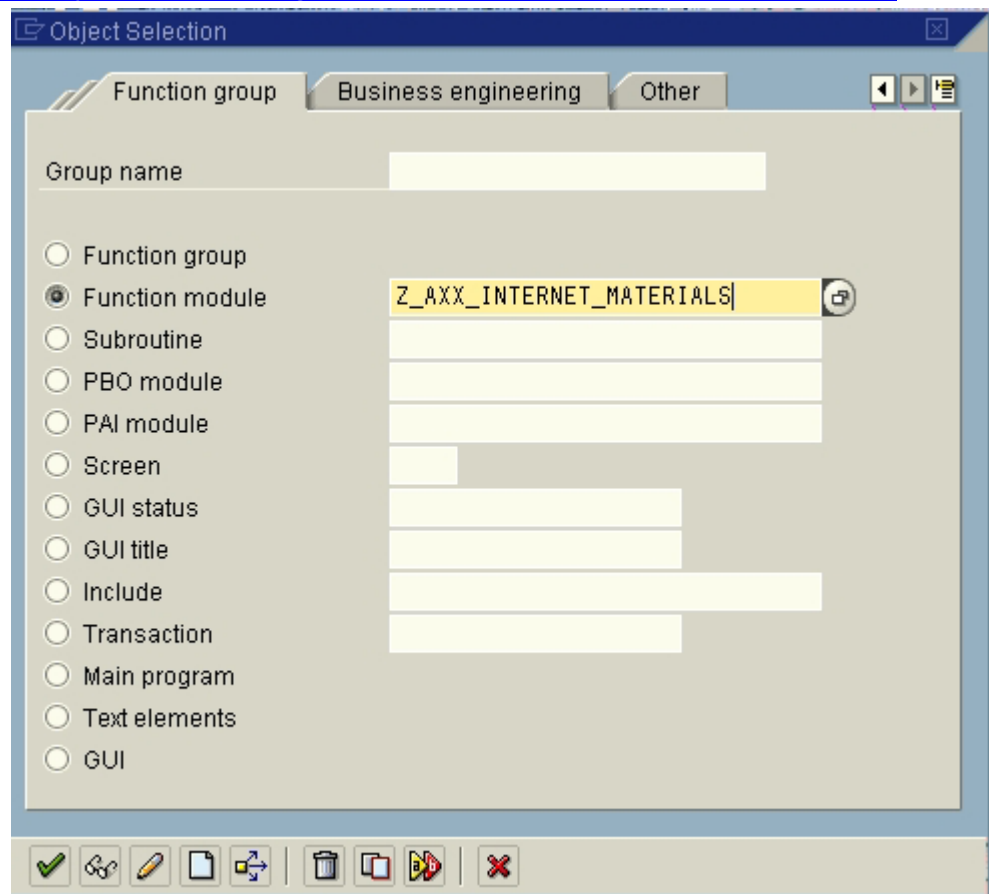
1.6 SE37: Creating A New Function Module

Function modules are public sub routines stored in function groups.

Strip: Creating a function module

The picture strip below shows an example of creating the framework for the function module Z_AXX_INTERNET_MATERIALS which we will store in the function group ZAXX_INTERNET.

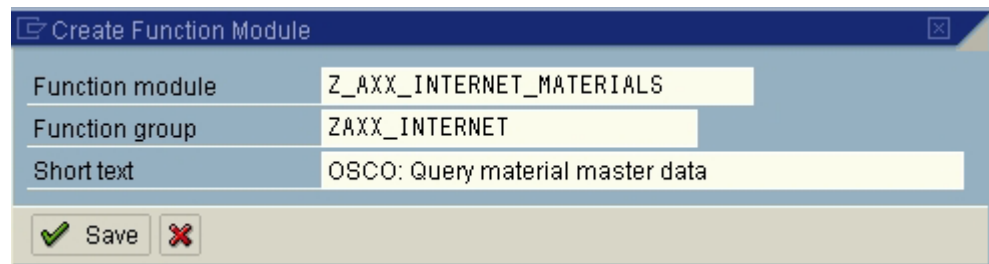
Figure 19: Choose the “New Object” creation dialog from the SE80 workbench



160

⇒

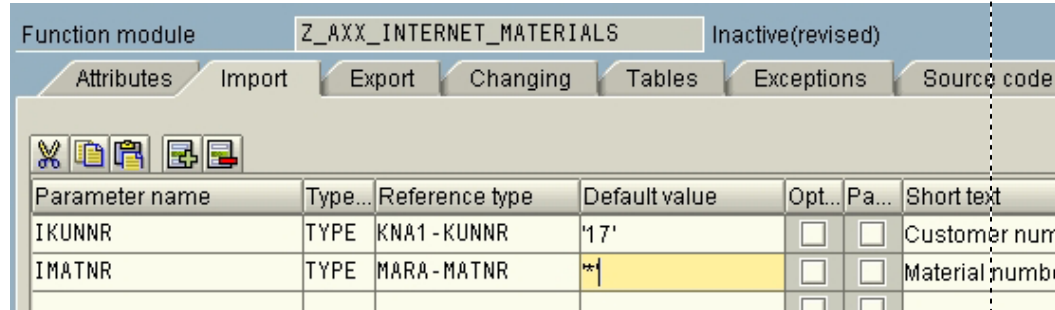
Figure 20: Assign the function to an existing function group



⇒

Development in R/3 ABAP 21

Figure 21: Enter import parameters for the function module



Parameter name	Type...	Reference type	Default value	Opt...	Pa...	Short text
IKUNNR	TYPE	KNA1-KUNNR	'17'	<input type="checkbox"/>	<input type="checkbox"/>	Customer num
IMATNR	TYPE	MARA-MATNR	**	<input type="checkbox"/>	<input type="checkbox"/>	Material numb

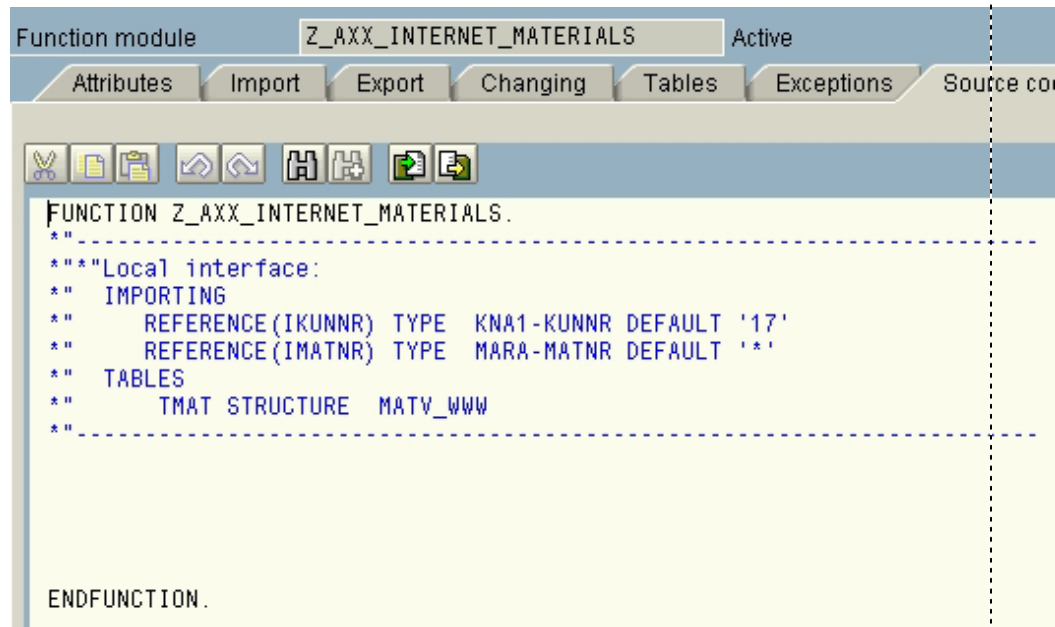
⇒

Entering the source code for the function

After defining the interface of the function, the ABAP engine will automatically generate a frame for the function. Between the FUNCTION and ENDFUNCTION statement you can enter your own program statements.

165

Figure 22: The interface has generated the frame for the function body



```
FUNCTION Z_AXX_INTERNET_MATERIALS.  
  *"  
  *""Local interface:  
  *  IMPORTING  
  *    REFERENCE(IKUNNR) TYPE  KNA1-KUNNR DEFAULT '17'  
  *    REFERENCE(IMATNR) TYPE  MARA-MATNR DEFAULT '*'  
  *  TABLES  
  *    TMAT STRUCTURE  MATV_WWW  
  *"  
  
  ENDFUNCTION.
```

Setting the RFC Attribute

If you want to call the function from a remote system, e.g. from Visual Basic or ASP the function must be RFC enabled. This is achieved by setting the RFC attribute in the general dialogue. However, an RFC function has some restrictions, mainly that all parameters must be strictly typed.

170

Figure 23: Setting the RFC attribute

The screenshot shows the SAP Function Builder interface for the function module `Z_AXX_INTERNET_MATERIALS`. The interface is divided into several sections:

- Classification:**
 - Function group: `ZAXX_INTERNET`
 - Application: `OSCO-Internet Enhancements`
 - Short text: `OSCO: Query material master data`
- Processing type:**
 - Normal function module
 - Remote-enabled module (highlighted with a red oval)
 - Update module
 - Start immed.
 - Immediate start, no restart
 - Start delayed
 - Coll.run
- General data:**
 - Person responsible: `AANGELI`
 - Last changed by: `AANGELI`
 - Changed on: `27.01.2001`
 - Development class: `ZAXX_INTERNET`
 - Program name: `SAPLZAXX_INTERNET`
 - Program name: `LZAXX_INTERNETU01`
 - Original language: `EN`
 - Not released:
 - Edit lock
 - Global



1.7 SWO1: Creating A New Object Class in R/3

In addition to its already rich features, there is full object support built in with ABAP objects. The object support exists since releases 3.1 and with release 4.6 the object support has ripened to its full extent.

175

The object definition examples that follow are based on release 4.6 and ABAP objects. There has been limited object support ever since release 3.1. The object support there has been sufficient for most applications and their most important application use, the definition of workflows and message pipes. What has changed in 4.6 is the way in which objects are defined and how they are stored in the repository. As far as features are concerned, ABAP objects added the polymorphism and remodelled the way inheritance is implemented, which was previously know as supertype.

180

There are basically two ways to define object classes with ABAP objects. You can define a class instream as part of standard ABAP code using the key words CLASS and ENDCLASS. Alternatively you may define them with the object builder, which will store the defined classes in the ABAP repository as library modules, similar to the way function modules are implemented.

Instream Creation of An Object Class

185

Define an Object Class Library With the Object Builder

1.8 SE10: Change Requests

During a project you will typically create or make changes to development objects and customising entries. Most of these will be transported to other SAP environments for integration or acceptance testing and finally to the productive system. SAP provides the change request facility to manage the orderly movement of these objects through the systems.

Other objects created, like test or utility programs, which were never intended to be transported can remain outside the change request facility. When you create an object you can choose to register this object in a change request or declare it to be a private object.

Change requests keep track of repository changes

Change requests are very powerful tools used to keep track of modified development objects or customising items in a development environment. These requests are used to synchronise the objects between the development and the other systems. A change request also allows you to group related objects which have changed to ensure that they will be transported to another system together.

Change requests will only record the name of the changed object

Change requests record the names of modified objects. When a developer decides to transport his modifications to the production system, the change request organiser evaluates its recordings and copies the modified items to an export file. This export file can then be imported to an another R/3 system.

Change requests do not record the nature of the modification

Change requests do not record the actual changes. They will not know what has changed in an ABAP or database table structure. It will only be aware of the fact that the object has changed (or rather, been touched) and requires synchronisation.

Therefore it makes sense that you are asked to register the object in a change request (existing or new) only the first time you modify a particular object. Once the object is recorded as changed, you may modify it under the same request until it is released for transport. The first time you modify the object after release you will again be asked to register it in a change request.

SE10 – transport organizer

Change requests are organised and handled with transaction SE10. There you can view and modify a change request, include additional objects and finally export the change request. You should not delete objects from a transport request, because the change request only records that the object has been touched. If it is touched, it is touched and this fact is recorded. If the object has not changed it is does no harm to transport it to a production system. **If you consequently use the transport management then an untouched object in R/3 will be definitely the same as in the production system. R/3 does not allow the same development object in different transport requests.**

STMS – Transport management system

Change requests should be transported with transaction STMS. The transport should be executed by the same person who did the changes in development. However a proper timing of when to transport your modifications in to production is essential. When you transport a program modification, while someone executes the very same program, it is very like that the person will loose the date that she/he is working on.

The responsibility of the person who does the transport is as follows:

- Immediately check the functioning of the transport objects in the target system
- If any errors are found react immediately, correct the error in the development system and urgently transport the repaired object

This can usually only be done by the person who did the modifications. It is generally out of the possibilities and duties of the basis administrator.

Development in R/3 ABAP 25

Duties of the basis administrator

235

- The duty of the SAP R/3 basis administrator is to

- Train the developer how to do change requests
- Inform the developers about risks and possible side effects of transports
- Coordinate change requests
- Approve the changes recorded in a request
- Supervise (but not execute!) the transport transfer activities

☺ Of course the basis administrator can do transports of his own developments.

240

Figure 24: Creation dialogue for a change request

The image shows two overlapping SAP dialog boxes. The top one is titled 'Prompt for local change request' and has a 'Development class' field containing 'ZAXX_INTERNET'. Below it is a 'Request' field with two empty input lines. The bottom dialog is titled 'Create Request' and contains a form with the following fields: 'Request' (empty), 'Short description' (OSCO-Internet Enhancements Version 0.1), 'Project' (empty with a lock icon), 'Owner' (AANGELI), 'Status' (New), and 'Last changed' (27.01.2001 21:08:40). There are also buttons for 'Own requests' and a red 'X' button.

⇒

Database tables for transport requests

All transport requests are stored in R/3 database tables starting like E070* and E071*. The principal tables are the following ones.

Figure 25: Tables used to store transport requests

R/3 Name	Description
E070	Parent entry of a transport request
E071	List of sub tasks of a transport
E071K	Keys of database table entries include in a request

⇒

2 First Steps Into ABAP IV

This section is a quick introduction to the main concepts of ABAP IV programming language. The idea of this primer is to give you a starting point in writing your own programs. It is far from a complete reference but rather expects that you have developer access to an R/3 system and can type in your own programs and play around with them. The main source for the language concepts shall be the ABAP online help, while the chapters presented here are a selection of the most important and relevant concepts of the language.

2.1 Hello World ABAP

As it is custom in IT circles we will start with a hello world example. Inspecting these simple lines of codes should give you a feeling how to enter a program, test it and how to use the help.

Listing 1: Hello world ABAP

```
REPORT ZSEXAM01.
*****
* Simple Hello World example                                     *
* Also shows usage of formatting options as color settings      *
* or symbol display.                                           *
*****

WRITE: / 'Hello World'.
WRITE: / 'Hello World' COLOR COL_TOTAL.      " changes color
WRITE: / 'Hello World' COLOR COL_NEGATIVE.   " changes color
WRITE: / 'Hello World' COLOR COL_POSITIVE.   " changes color

Output as produced by the hello world example
16.04.1999                               Hello World Example.          1
-----
Hello World
Hello World
Hello World
Hello World
```

245

This little ABAP report outputs four times the phrase “Hello World” on the screen in different colors.

Every statement is terminated by a dot.

```
WRITE: / 'Hello World'.
```

The separator between commands is the dot (.). In PASCAL this would be the semicolon (;) and in Visual Basic it is the end of the line. In other words: after a dot, the ABAP parser assumes a new statement.

250

The write statement outputs the string and parameter on the screen

```
WRITE: / 'Hello World' .
```

The write statement is the general output instruction in ABAP. It is followed by the constant or a variable to be displayed.

Colons can separate repeating parameter blocks

With the WRITE statement you learn an element of the standard ABAP syntax. Whenever you want to repeat a statement several times, you can write it once and have it followed by a colon (:), and then list the parameter blocks separated by a comma.

```
WRITE: 'Hello', 'World'.
```

Is the same as

```
WRITE 'Hello'.
WRITE 'World'.
```

While

```
WRITE 'Hello', 'World'. "Incorrect !!!!, no colon
```

is syntactically incorrect (mind the missing colon).

260

The / instructs to go to a new line

```
WRITE: / 'Hello World' .
```

The single slash means to begin printing the following output at the start of a new line.

Pressing F1 when the cursor is over a statement displays help

To learn more about the WRITE statement you may want to refer to the excellent ABAP online help. It is displayed in its context when you press F1 when the cursor is positioned over a valid ABAP statement. If you work with the ABAP command line editor, you can equally type “HELP write” in the command line. There you will also learn about the formatting options like the COLOR attribute.

Now call the ABAP editor with SE38 or from the menu and enter the program

Now call the ABAP editor using transaction SE38 and enter the hello world program. You may call the editor from somewhere in the menu but I will not tell you how, because using the menus is not good practice for a good developer.

2.2 Using Parameters

This sample demonstrates how easy parameter screens for an ABAP can be designed, what processing events exist and how arithmetic calculations are done.

Listing 2: ABAP that uses screen entry parameters and execution events

```
REPORT ZSEXAM02.
*****
PARAMETERS: YOURNAME(25) OBLIGATORY.
PARAMETERS: FAKTOR01 TYPE P DECIMALS 2 OBLIGATORY DEFAULT '123456.79'.
PARAMETERS: FAKTOR02 TYPE P DECIMALS 2 OBLIGATORY.
*****
DATA: ERGEBNIS TYPE P DECIMALS 4.
*****
INITIALIZATION.
FAKTOR02 = '0.09'.
FAKTOR02 = FAKTOR02 * 7.

*****
START-OF-SELECTION.
WRITE: / 'Hello'.
WRITE: YOURNAME.
SKIP 2.
ERGEBNIS = FAKTOR01 * FAKTOR02.
WRITE: / FAKTOR01, ' mal ', FAKTOR02, ' gibt ', ERGEBNIS.
SKIP.
WRITE: ' Oder etwas schoener MIT RUNDEN:'.
WRITE: / FAKTOR01, ' mal ', FAKTOR02, ' gibt '
, ERGEBNIS DECIMALS 2.

*****
END-OF-SELECTION.
```

ABAP uses events at certain checkpoints

The statements START-OF-SELECTION, END-OF-SELECTION and INITIALIZATION are referred to as EVENTS. Actually, these are subroutines (FORM) which are called by the system watchdog whenever an appropriate event occurs. Therefore START-OF-SELECTION will be called upon entering the program body, whereas INITIALIZATION is called once at the very beginning of execution before any other part of the ABAP is executed. These events exist in other languages as well as shown in the following table.

280

Figure 26: Correspondence of events in different languages

ABAP	Pascal	Visual Basic	Description
INITIALIZATION	Constructor	Sub Class_Start	At start of execution
START-OF-SELECTION	Body	Sub Main	Start of main program
END-OF-SELECTION	Destructor	Sub Class_Terminate	Called before programs ends normally

PARAMETER statement displays a data entry screen

The PARAMETER statement displays an entry field on the welcome screen where the user can enter data values before execution of the main program. The parameter screen will be displayed after executing the INITIALIZATION event but before processing the START-OF-SELECTION event. Remember to press F1 over the PARAMTER key word in the ABAP editor to see detailed and exhaustive help about the use of parameters and designing sophisticated input screen.

285

```
PARAMETERS: YOURNAME(25) OBLIGATORY.
PARAMETERS: FAKTOR01 TYPE P DECIMALS 2 OBLIGATORY DEFAULT
'123456.79'.
PARAMETERS: FAKTOR02 TYPE P DECIMALS 2 OBLIGATORY.
```

Variables are declared with the DATA command

```
DATA: ERGEBNIS TYPE P DECIMALS 4.
```

Before anything else happens, the code after **INITIALIZATION** is processed

```
INITIALIZATION.  
FAKTOR02 = '0.09'.  
FAKTOR02 = FAKTOR02 * 7.
```

After displaying the parameter screen and waiting for input, the **START-OF-SELECTION** block is executed

This is the main program or program body.

```
START-OF-SELECTION.  
WRITE: / 'Hello'.  
WRITE: YOURNAME.  
SKIP 2.  
ERGBNIS = FAKTOR01 * FAKTOR02.  
WRITE: / FAKTOR01, ' mal ', FAKTOR02, ' gibt ', ERGBNIS.  
SKIP. "blank line  
WRITE: ' Oder etwas schoener MIT RUNDEN:'.  
WRITE: / FAKTOR01, ' mal ', FAKTOR02, ' gibt '  
          , ERGBNIS DECIMALS 2.
```

END-OF-SELECTION is executed last

```
END-OF-SELECTION.
```

The **END-OF-SELECTION** part is the code to be executed immediately before the program ends. It can be regarded as the program **TERMINATOR**. There is no code to be executed at the **END-OF-SELECTION** event in our example. However, we put the empty statement there to enhance the readability of the program. It could have been left out, however.

2.3 Select-Options

Select-Options on the report program initial screen are a convenient way to enter simple or complex selection criteria. They correspond to internal range tables that are used to generate simple or complex WHERE statements for SQL selects.

Listing 3: ABAP that uses select options and does an SQL select

```

REPORT ZSEXAM03.
*****
* This sample demonstrates the SQL-like database selection.          *
* It also restricts the scan of the database to a range              *
* We are using the table T005T, which contains a language           *
* specific name of all countries known to SAP                        *
*-----*
* Giving the command SHOW T005T in the editor command line          *
* we may see all the fields of table T005T                          *
*-----*
*Fldname   Key Datenelem. Typ  Länge PrüfTab  Kurzbeschreibung
*MANDT     MANDT   CLNT    3 T000    Mandant
*SPRAS     SPRAS   LANG    1 T002    Sprachenschlüssel

*LAND1     LAND1   CHAR    3 T005    Länderschlüssel
*LANDX     LANDX   CHAR    15      Bezeichnung des Landes
*NATIO     NATIO   CHAR    15      Bezeichnung der Nation
*****
TABLES: T005T.                " A table with all country codes

* Interval-Restriction for language code "SPRachSchlüssel"
SELECT-OPTIONS: S_SPRAS FOR T005T-SPRAS.
* Interval-Restriction for language code "SPRachSchlüssel"
SELECT-OPTIONS: S_LAND FOR T005T-LANDX.

START-OF-SELECTION.          " Optional but good practice

* Retrieve Data from table
SELECT *
      UP TO 5 ROWS           " no more than 5 hits
      FROM T005T             " read table T005T
      WHERE SPRAS IN S_SPRAS " Restrict language
      AND LANDX IN S_LAND   " Restrict Country by name
      ORDER BY LAND1.       " Sort result
WRITE: / T005T-SPRAS.      " The / begins a new line (CR)
WRITE: SY-VLINE, T005T-LAND1. " sy-vline is a system variable
WRITE: SY-VLINE, T005T-LANDX. " that draws a vertical line
WRITE: SY-VLINE, T005T-NATIO.
ENDSELECT.
    
```

300

When you execute the above program, a screen appears and asks for a language code to be entered in order to limit the search for countries.

For language S_SPRAS = 'E' and country S_LAND = 'G*' the result would look similar to the following output. Note that we limit the retrieval to 5 matching result rows by saying UP TO 5 ROWS.

305

Listing 4: Sample output of the program above

S_SPRAS	S_LAND		
* E	DE	GERMANY	GERMAN
* E	GA	GABON	GABONESE
* E	GB	GREAT BRITAIN	BRITISH
* E	GD	GRENADA	GRENADIAN
* E	GE	GEORGIA	GEORGIAN

SELECT-Options create internal range tables

Select-Options is a pretty clever program statements. It declares an internal table (kind of ARRAY, more or less the same as an ADO recordset) with a special structure. This internal table has always four columns:

Figure 27: Columns of a range table

SIGN	it is either I or E, I means to include the found results, E excludes them
OPTION	takes a comparison command like EQ - equal, LE = less equal, LT = less than GT = Greater than, GE = greater equal BT = Between Low and High CP = compare with wildcards (e.g. CP US*)
LOW	Low value used to compare
HIGH	High value to compare, only for BT option

310 When you declared a select option like this

```
SELECT-OPTIONS: S_LAND FOR T005T-LANDX.
```

The ABAP run time processor would build the internal table according to the data entered by the user. If e.g. he entered "US" for USA, then the internal table would have the following entry:

```
I | EQ | US
```

315 If you entered a wildcarded value like D* for all countries starting with D, the entry would look like:

```
I | CP | D*
```

If you entered a range all between A and DZ, the entry would look like:

```
I | BT | A | DZ
```

320 The "FOR fieldname" attribute is used to determine the data type of the screen input field. This determines the size of the input field and can call conversion routines if necessary, e.g. from character to packed decimal or float and vice versa.

If you do not want the range table to be displayed on the screen and rather construct it yourself, you can use the RANGE statement instead to declare a table.

```
RANGES: S_LAND FOR T005T-LANDX
```

325 **Select-options and ranges allow implicit OR statements** One of the benefits of having the select-option ranges is obvious when building where clauses for SQL selects. A range can be specified with the IN operator as follows:

```
SELECT * FROM T005T WHERE LANDX IN S_LAND.
```

330 If the S_LAND select-options then contains multiple rows like:

```
Sign|Option|Low |High |
I | EQ | US | |
I | CP | D* | |
I | BT | A | BZ
```

The the ABAP processor would automatically construct an SQL statement like the following:

```
SELECT * FROM T005T
WHERE ( LANDX = 'US' )
OR ( LANDX LIKE 'D%' )
OR ( LANDX >= 'A' AND LANDX <= 'BZ' )
```

335 If the select-options table is empty, then it is ignored, thus

```
SELECT * FROM T005T WHERE LANDX IN S_LAND.
```

Development in R/3 ABAP 33

With an empty table S_LAND would retrieve all records from T005T, i.e. be the same as not having the WHERE clause at all:

```
SELECT * FROM T005T
```

2.4 SQL-Selects

In order to read a database table, ABAP supports a subset of the OPEN SQL standard,,,,.

TABLES The tables statement declares a buffer memory area to hold the data of actual record.

```
TABLES: T005, *T005X.
```

The buffer can be declared explicitly as well:

```
DATA: XT005 like T005.
```

345

The buffer can be declared as an internal table, too:

```
DATA: T_T005 like T005 OCCURS 0 WITH HEADER LINE..
```

SELECT The main statement is the SELECT ... ENDSELECT statement, which defines a loop over all matching records of a specified table or database view and executes the statements in between once for every row retrieved. If no INTO buffer is specified a buffer with the same name as the table is assumed.

350

```
SELECT * FROM T005.
...
ENDSELECT.
```

Is the same as

```
SELECT * FROM T005 INTO T005.
...
ENDSELECT.
```

Alternatives:

```
SELECT * FROM T005 INTO *T005.
...
ENDSELECT.
```

355

```
SELECT * FROM T005 INTO XT005.
...
ENDSELECT.
```

And the following reads the whole database table into the memory without generating a loop.

```
SELECT * FROM T005 INTO TABLE T_T005.
```

UP TO n Rows – limit the number of rows retrieved The very useful addition UP TO n ROWS lets you limit the number of rows retrieved and exits the select loop if the specified number of rounds is reached. The statement EXIT can be used to leave the loop pre-emptively.

```
SELECT * UP TO 5 ROWS FROM T005 INTO TABLE T_T005.
```

INSERT Insert the data stored in the specified buffer in the database table.

```
INSERT INTO T005.
```

Is the same as

```
INSERT INTO T005 FROM T005.
```

Alternatives:

Development in R/3 ABAP 35

370

```
INSERT INTO T005 FROM *T005.  
INSERT INTO T005 FROM TABLE T_T005. "insert all records from T_T005
```

UPDATE

Update is the same as insert, but requires that a record with the same key already exists in the database table.

```
UPDATE T005.
```

Is the same as

```
UPDATE T005 FROM T005.
```

375

Alternatives:

```
UPDATE T005 FROM *T005.  
UPDATE T005 FROM TABLE T_T005. "UPDATE all recs from T_T005
```

MODIFY

This is a combination of insert and update. ABAP first tries to insert the record. If it already exists, it updates it appropriately.

```
MODIFY T005.
```

380

Is the same as

```
MODIFY T005 FROM T005.
```

Alternatives:

```
MODIFY T005 FROM *T005.  
MODIFY T005 FROM TABLE T_T005. "MODIFY all recs from T_T005
```

385

So a modify is the same as:

```
INSERT INTO T005.  
IF SY-SUBRC <> 0.  
    UPDATE T005.  
ENDIF.
```

2.5 Internal Tables – Recordsets in ABAP

Internal tables are what ABAP knows instead of Arrays and they are pretty much like the Visual Basic ADO record sets.

Listing 5: ABAP that uses select options and does an SQL select

```
REPORT ZSEXAM04.
TABLES: T005T.           " A table with all country codes

* Now we declare an internal table. This is a data structure which
* is rarely found in other PL.
DATA: BEGIN OF T_T005T OCCURS 5.
      INCLUDE STRUCTURE T005T.   " use definition from data dictionary
DATA: END OF T_T005T.

START-OF-SELECTION.           " Optional but good practice
SELECT *
      UP TO 5 ROWS              " no more than 5 hits
      FROM T005T                " read table T005T
      INTO TABLE T_T005T      " deposit result into internal table
      WHERE SPRAS EQ 'E'       " Restrict language to English
      ORDER BY LAND1.          " Sort result

* the loop statement loops over all rows in the internal table.
LOOP AT T_T005T.
  WRITE: / SY-TABIX, 'of', SY-TFILL. " SY-TABIX is the number of the
                                     " currently read row

  WRITE:   SY-VLINE, T_T005T-LANDX.
ENDLOOP.

WRITE: / 'We can also read the internal table directly:'.
READ TABLE T_T005T INDEX 2.
IF SY-SUBRC EQ 0.
  WRITE: / 'The second row of the table is'.
  WRITE:   SY-VLINE, T_T005T-LANDX.
ELSE.
  WRITE: / 'The internal table has less than 2 rows'.
ENDIF.
```

⇒

When you execute above program, the SQL select retrieves the matching entries from the database table T005T and stores the retrieved result records into the ABAP internal table T_T005T (the name is arbitrarily chosen and has been declared as DATA: BEGIN OF T_T005T OCCURS 5 ...). The rows of this internal table can be processed by means of the LOOP AT statement.

Listing 6: Sample output of the program above

```
* E DE GERMANY GERMANY
* E GA GABON GABONESE
* E GB GREAT BRITAIN BRITISH
* E GD GRENADA GRENADIAN
* E GE GEORGIA GEORGIAN
```

⇒

Internal tables are
database tables like
dynamic arrays

395

ABAP internal tables are declared by adding the option OCCURS 0 to DATA declaration.

The occurs parameter has originally been an estimate of the number of rows you may expect in the internal table. The ABAP engine allocated exactly the amount of memory that is needed to accommodate the specified number of entries. This would not mean that this limited the number of entries in the table, but adding additional members meant to reallocate memory clusters during runtime, costing performance. Allocating too much memory would otherwise mean wasting precious internal memory. Today a value of 0 leaves the memory management to the ABAP engine which is generally appropriate unless you are pretty sure about the number of records to expect.

390

400

Development in R/3 ABAP 37

Use data dictionary structure to define a table with INCLUDE STRUCTURE

```
DATA: BEGIN OF T_T005T OCCURS 5.  
      INCLUDE STRUCTURE T005T.  
DATA: END OF T_T005T.
```

405

The include statement used between the DATA BEGIN and DATA END tells the ABAP interpreter to insert the field definitions according the specified data dictionary structure.

Store SQL results directly in an internal table

410

When we execute the SELECT we specify this time the command INTO TABLE and the name of our internal table. This instructs the OPEN SQL to put all matching table rows into the specified internal table.

```
SELECT *  
      UP TO 5 ROWS           " no more than 5 hits  
      FROM T005T            " read table T005T  
      INTO TABLE T_T005T  " deposit result into internal  
table  
      WHERE SPRAS IN S_SPRAS " Restrict language  
      AND LANDX IN S_LAND   " Restrict Country by name  
      ORDER BY LAND1.      " Sort result
```

Process internal tables with LOOP .. ENDLOOP

To get the content of the internal table row by row we can use the LOOP AT ... ENDLOOP statement block.

```
LOOP AT T_T005T.  
...  
ENDLOOP.
```

2.6 Function Modules

Function Modules are ABAP library functions. They are declared with a special interface generator, the transaction SE37. Functions are the only program interface between ABAP and non R/3 programs like Visual Basic.

Function modules correspond to components

Function modules are principally the same as functions in other 4GL languages. They are programmed code which are stored in special collections, called function groups. In a way, they correspond to the COM component in Visual Basic, where the functions are the methods of the components. Because, function modules are not object classes, they sometimes lack functionality you might expect from object programming. In most practical situations, however, you will not miss true object classes when programming in ABAP.

Functions are organized in an ABAP code pool SAPL...

Function modules are organized as function pools, which are a collection of functions. A function pool is a single ABAP which is generated automatically by the function pool editor (SE37) and starts with the string SAPL You cannot edit a function pool directly. All modifications have to done via the transaction SE37.

RFC functions are a special subset

For our purpose as a web developer, we are mostly interest in special kind of functions in R/3, those which are RFC enabled. These functions are ordinary functions, with the restriction that they have fully typed parameters and that they have the RFC flag ticked, to indicate to the ABAP processors that RFC calls to that specific function are permitted.

3 SAP R/3 Business Suite And R/3 Data Model Views

The R/3 business data view is the principle source of your enterprise business intelligence. This is a more general variant of the entity-relationship model used in relational database design. It tells you how real world data is mapped and stored in database tables and how these tables are related and interact with each other.

3.1 About Model Views

Depending on the point of view of the spectator an application can be seen from the business organisation, from the IT infrastructure or from the user who operates the computer.

Models are projection on a multidimensional world

Models are generally projections on real world objects. They are like a two dimensional picture of a three dimensional world. Different groups in an enterprise like the business executive, the strategic planner or the IT responsible put a completely different and often conflicting view on the enterprise data.

Resource view

The resource view is stressed by the IT infrastructure responsible. They ask where the data and application is to be stored, how much space and CPU time it will consume and if the whole application is going to be safe and easy to maintain.

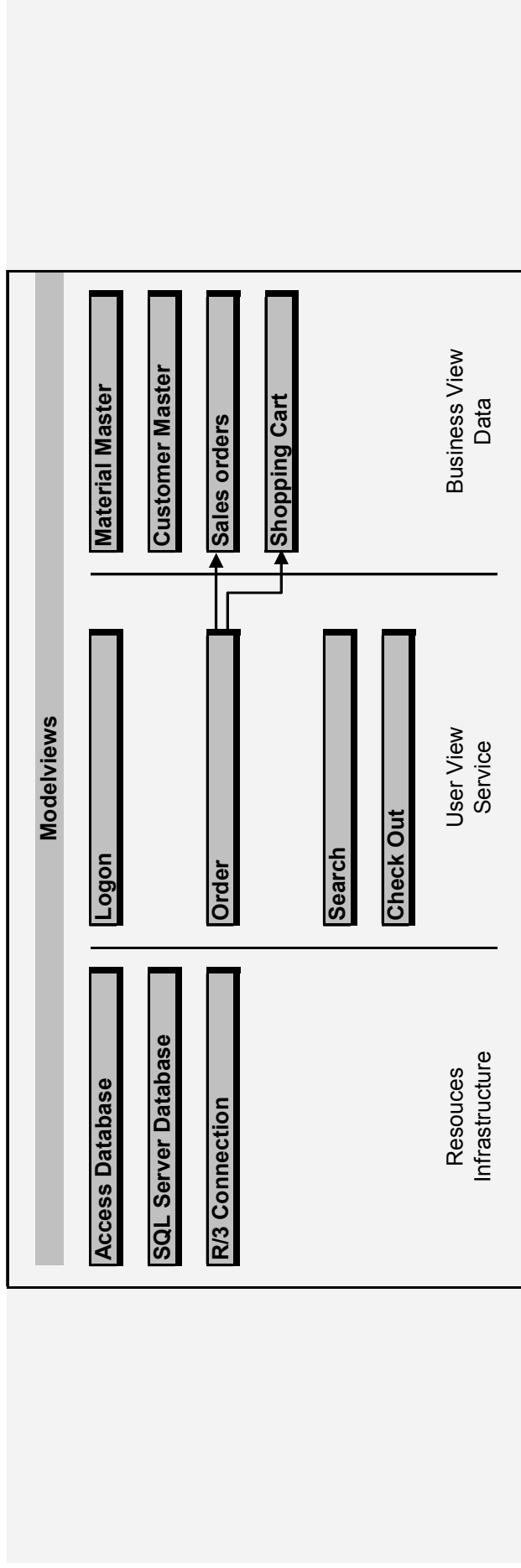
Business data view

The business data view is subject for the business strategists who engineer the organisation flow of the company's data. They tell you the abstracts how e.g. sales orders are recorded, how they trigger and influence shipments and what kind of data has to be recorded and kept over a period of time.

User view

The business user is the key person in an infrastructure. She/he will be the one who enters data into the computer and depending on the user's acceptance and care, the whole performance of your sales may be influenced. The user view is what the user sees: the screen forms, the shortcut keys to press and how the enterprise data is presented to him, e.g. is a material a product, a service a whole bill of material for him.

Figure 28: Various views of enterprise data



3.2 Important R/3 Data Objects

Knowing how R/3 models your data and maps them into database table may be essential for your forthcoming as ABAP developer. The illustrations in this chapters give you an overview of the most important business data.

A more complete overview can be queried with transaction SARA

Unfortunately does R/3 still not provide a sufficient entity model of its complete database. Finding the relations between tables means still to investigate ABAP codes, check possibly existing BAPI interfaces or simply asking an experienced consultant. Luckily there is the transaction SARA, which takes care of the archiving of business objects. This transactions gives a pretty complete entity model of the most important business objects in R/3, because to archive an object, the archive task needs to know if there are dependent table entries in other tables.

5

Figure 29: Important business objects in R/3

Fehler! Keine gültige Verknüpfung.

3.3 A Sample Business Scenario

To sharpen the imagination we want to model the scenario of a sales company with multiple stores in R/3. The stores shall be delivered from a central production plant and stores may sell products as well as other stores may claim a product to be transferred from a different store to its own location.

Models Models

10

3.4 RFC Functions, BAPIs or IDocs

When you want to write data back to R/3 you have make a basic decision which technology you use to send the data back to R/3.

RFC Functions

RFC Functions are ABAP coded functions that allow access via RFC

RFC enabled functions are standard ABAP coded function modules, that are flagged as RFC enabled. Enabling RFC is to be understood in the sense, that the developer allows external applications to call this function module. Allowing RFC calls to a function means, that the developer has to be careful how to code the routines. The most important restrictions for RFC functions are:

Statelessness: While a standard function can retain a global memory in the body of the function pool to which the function belongs, an RFC function creates its own memory space every time it is called and destroys the memory on leave.

Strong parameter typing: While general functions can have variant typing of its parameters, i.e. a parameter inherits the type of the passed parameter variable, an RFC function parameter must have a type.

BAPIs

BAPIs are publicly released RFC functions

BAPIs, the Business Application Programming Interface has been a hot topic in R/3 circles for long. To take the myth out of them: BAPIs are pre-coded RFC functions, which have been released by SAP for public use. Bluntly speaking, these are routines where SAP will have a bad conscience when it changes something incompatibly. Programmer would say it even simpler: BAPIs are RFC functions whose name starts with the letters BAPI.

IDocs

IDocs are ASCII files which are stored in an intermediate database

IDocs are data structures that allow data exchange between R/3 and other systems in ASCII format. The big advantage of IDocs is actually, that IDocs are usually stored in a database table. If there is anything wrong during the processing, the data is persistent in the database and can be reprocessed from that intermediate storage.

What to Use

I generally rule out BAPIs. The basic idea of having an SAP approved routine for data storage may be cunning, but the quality of the current BAPIs is not satisfactory. Either they are too inflexible or too general or their error handling facilities are very poor.

Instead of BAPIs you may write your own, dedicated function module that does a Call Transaction Using (CTU). Doing so you can write in a way that the CTU does process those screens only, that are likely to be safe with your data.

However, if you are uncertain about your data, then you may want to use IDocs instead. Their main difference is, that the IDocs will first be stored in the database tables EDIDC and EDID4, like a message is stored in a message queue. From there a message handler will process the IDocs whenever it thinks it is appropriate. Storing the data in the IDoc base is guaranteed, so the calling application can be certain that the data is taken care of.

Storing a Sales Order

The simple approach would be to write a function that directly updates the database. Knowing that the database tables involved in a sales order processing are VBAK, VBAP, you could code a function similar to the following:

```
UPDATE vbak SET vbeln='INT000001' kunnr='MCDONALD' .
```

```
UPDATE vbap SET vbeln='INT000001' matnr='HAMBURGER'  
kwmeng=10000.00 .
```

55

Apart from reinventing the wheel, this would be a very unstable solutions, because you never know how SAP might change the underlying database structures. In fact the example above would likely bring your installation down. There are many other tables involved in a single sales order update among others these are VBUK, VBUP, VBFA (document flow), KONV (prices), VBPA (partner).

60

3.5 SAP Sample Database: FLIGHT

65

Every SAP R/3 installation comes with a example database application „FLIGHT“, which is used to demonstrate the functionality of the ABAP programming language and is used during all SAP developer courses. It is SAP R/3's correspondent to Microsoft Windows Northwind database.

The database consist of a series of tables, namely the following most important ones.

SAIRPORT	Airports
SAPLANE	Plane
SBOOK	Flight booking
SBUSPART	Airline partner
SCARPLAN	Plane-airline assignment
SCARR	Airline
SCITAIRP	City-Airport assignment
SCOUNTER	Sales counter
SCPLANE	Cargo plane
SCURR	Exchange rates for Workbench trainin
SCURX	Currency for Workbench training data
SCUSTOM	Flight customers
SDESSERT	Inflight meal/Dessert
SFLIGHT	Flight
SFLIMEAL	Flight-Meal assignment
SGEOCITY	Geographical position of a city
SMACOURSE	Inflight meal/Main course
SMEAL	Inflight meal
SMEALT	Inflight meal/Description
SMENU	Menu
SPFLI	Flight schedule
SPPLANE	Passenger plane
SSTARTER	Inflight meal/Appetizer
STRAVELAG	Travel agency

⇒

Tables are in development class SAPBC_DATAMODEL

An uptodate listing of the tables and other DDIC objects belonging to the FLIGHT demo model can be found by viewing the objects of the development class SAPBC_DATAMODEL using the transaction SE80 (the workbench).

Data for the flight database can be generated using transaction BC_TOOLS_DATA

If you are lucky the FLIGHT database is already loaded with good data. If not, you can generate them using the transaction BC_TOOLS_DATA . Choose the desired number of data to be generated and wait until the data is there. Once the transaction has successfully finished, you can view the generated contents of datanbase table using transaction SE16.

4 Examples of ABAP Function Modules

In this chapter we want to introduce some sample ABAP function modules. Although the functions are certainly helpful in many circumstances, their main idea is to introduce to you some concepts of SAP R/3 ABAP programming and the use of such programs in interfaces.

4.1 FUNCTION Y_DEMO_FLIGHT_LOADDATA

Populate tables of the SAP training Flight Demo Model with data.

This is a demo function module, how to insert data into a table pool via RFC. The function expects a couple of table data as a parameter and inserts the data into the corresponding database table.

The table structure of the function module parameters are exactly the ones of the corresponding database tables. The insert is done with the ABAP MODIFY open SQL statement, which either inserts a record or updates it automatically if the record's key is already in the table.

85

Listing 7: ABAP function to insert data into the FLIGHT database tables

```
FUNCTION Y_DEMO_FLIGHT_LOADDATA.
*-----
*""Lokale Schnittstelle:
*   TABLES
*       TSCARR STRUCTURE  SCARR OPTIONAL
*       TSPFLI STRUCTURE  SPFLI OPTIONAL
*       TSFLIGHT STRUCTURE SFLIGHT OPTIONAL
*       TSBOOK STRUCTURE  SBOOK OPTIONAL
*-----
  MODIFY SCARR   FROM TABLE TSCARR.
  MODIFY SPFLI  FROM TABLE TSPFLI.
  MODIFY SFLIGHT FROM TABLE TSFLIGHT.
  MODIFY SBOOK  FROM TABLE TSBOOK.
ENDFUNCTION.
```

Listing 8: ' Visual Basic example to call the function Y_DEMO_FLIGHT_LOADDATA

```
Set funcControl = CreateObject("SAP.Functions")
Set TableFactoryCtrl = CreateObject("SAP.TableFactory.1")
Set myR3logonObj = New R3logonObj
funcControl.Connection = myR3logonObj.R3connection
Set fbY_DEMO_FLIGHT_LOADDATA = _
= funcControl.Add("Y_DEMO_FLIGHT_LOADDATA")
Set tTSBOOK = fbY_DEMO_FLIGHT_LOADDATA.Tables("TSBOOK")
Set tTSCARR = fbY_DEMO_FLIGHT_LOADDATA.Tables("TSCARR")
Set tTSFLIGHT = fbY_DEMO_FLIGHT_LOADDATA.Tables("TSFLIGHT")
Set tTSPFLI = fbY_DEMO_FLIGHT_LOADDATA.Tables("TSPFLI")
' -- add a row to SFLIGHT input table
tTSFLIGHT.AppendRow
linno = tTSFLIGHT.RowCount
tTSFLIGHT("MANDT", linno).Value = pMANDT ' = " "
tTSFLIGHT("CARRID", linno).Value = pCARRID ' = " "
tTSFLIGHT("CONNID", linno).Value = pCONNID ' = 0
tTSFLIGHT("FLDATE", linno).Value = pFLDATE ' = " "
tTSFLIGHT("PRICE", linno).Value = pPRICE ' = " "
tTSFLIGHT("CURRENCY", linno).Value = pCURRENCY ' = " "
' -- Call the Function via RFC
```

```
fbY_DEMO_FLIGHT_LOADDATA.Call
```

⇒

4.2 FUNCTION Y_DEMO_IDOC_GENERATE_ORDERS

This is a demo function module, with the purpose to create and submit an SAP R/3 IDoc to create a sales order document from the data passed to the function.

90

This is a very handy function for quickly creating sales order documents in R/3 from a web application, because the IDoc creation first stores all passed data safely in the IDoc base and then tries to trigger the much more complicated and failure sensitive creation of a sales document. The transaction will return the IDoc number as a reference number to the caller, which can be used to trace the action.

95

In case that the sales document could not be created, the IDoc can be viewed and edited if needed with the IDoc monitor (transactions WEDI or directly via WE02).

Using transaction BD87, the IDoc can be replayed visually (as a call transaction) and thus the operator can check the cause of error and fill in the missing or erroneous data if possible or appropriate.

Listing 9: ABAP Function that generates and submits an IDoc for sales orders

```

FUNCTION Y_DEMO_IDOC_GENERATE_ORDERS.
*-----
*"" Lokale Schnittstelle:
*   IMPORTING
*       VALUE(TORDERHEAD) LIKE VBAK STRUCTURE VBAK
*       VALUE(SENDER) LIKE EDIDC-SNDPRN DEFAULT 'DUMMY'
*       VALUE(IAUART) LIKE VBAK-AUART DEFAULT 'TA'
*   EXPORTING
*       VALUE(IDOCNUM) LIKE EDIDD-DOCNUM
*       VALUE(ERROR_BEFORE_CALL_APPLICATION)
*           LIKE EDI_HELP-ERROR_FLAG
*   TABLES
*       TITEMS STRUCTURE ORDPAR OPTIONAL
*-----

TABLES: NAST, VBAK.

TABLES: E1EDK01. " IDOC: BELEGKOPF DATENrALLGEMEIN
TABLES: E1EDK14. " IDOC: BELEGKOPF ORGANISATIONSDATE
TABLES: E1EDK03. " IDOC: BELEGKOPF DATUMSSEGMENT
TABLES: E1EDK04. " IDOC: Belegkopf Steuern
TABLES: E1EDK05. " IDOC: BELEGKOPF KONDITIONEN
TABLES: E1EDKA1. " IDOC: BELEGKOPF PARTNERINFORMATIO
TABLES: E1EDK02. " IDOC: BELEGKOPF REFERENZDATEN
TABLES: E1EDK17. " IDOC: BELEGKOPF LIEFERBEDINGUNGEN
TABLES: E1EDK18. " IDOC: BELEGKOPF ZAHLUNGSBEDINGUNG
TABLES: E1EDP19. " Object identification
TABLES: E1EDKT1. " IDOC: BELEGKOPF TEXTIDENTIFIKATIO
TABLES: E1EDP01. " IDOC: BELEGPOSITIONuDATEN ALLGEME
TABLES: E1CUCFG. " CU:CKONFIGURATIONSDATEN
TABLES: E1EDL37. " VERSANDELEMENTKOPFeuern
TABLES: E1EDS01. " IDOC: SUMMENSEGMENTuALLGEMEIN

DATA: INT_EDIDD LIKE EDI_DD OCCURS 0 WITH HEADER LINE.
DATA: INT_EDIDC LIKE EDI_DC OCCURS 0 WITH HEADER LINE.
DATA: OWN_LOGICAL_SYSTEM LIKE TBDLS-LOGSYS.
    
```

100

- * If you get back 9999999999 then the function call has been successful
- * but the IDoc has not been created.
- * If the RFC returns a 0, then the function has never been called
- * " Possible Causes : SYNTAX Error in function pool!

105

```
IDOCNUM = '9999999999'.
```

* Basic identification and Reference Segment

```
E1EDK01-ACTION = '000'.
E1EDK01-CURCY = TORDERHEAD-WAERK.
E1EDK01-WKURS = '1.0000'.
E1EDK01-BELNR = TORDERHEAD-VBELN.
INT_EDIDD-SEGNAM = 'E1EDK01'.
INT_EDIDD-SDATA = E1EDK01.
APPEND INT_EDIDD.
```

* - Specify the Order type to use for creation in R/3. Default: TA

```
E1EDK14-QUALF = '012'.
E1EDK14-ORGID = IAUART.
INT_EDIDD-SEGNAM = 'E1EDK14'.
INT_EDIDD-SDATA = E1EDK14.
APPEND INT_EDIDD.
```

```
INT_EDIDD-SEGNAM = 'E1EDKA1'.
E1EDKA1-PARVW = 'AG'.
E1EDKA1-PARTN = TORDERHEAD-KUNNR.
INT_EDIDD-SDATA = E1EDKA1.
APPEND INT_EDIDD.
E1EDKA1-PARVW = 'WE'.
E1EDKA1-PARTN = TORDERHEAD-KUNNR.
INT_EDIDD-SDATA = E1EDKA1.
APPEND INT_EDIDD.
```

* --Process Order Items now

```
LOOP AT TITEMS.
E1EDP01-ACTION = '000'.           "Quantity
E1EDP01-POSEX = TITEMS-VBELP.
E1EDP01-MENGE = TITEMS-MENGE.
E1EDP01-MENEE = TITEMS-GMEIN.
E1EDP01-MENGE = TITEMS-MENGE.
INT_EDIDD-SEGNAM = 'E1EDP01'.
INT_EDIDD-SDATA = E1EDP01.
APPEND INT_EDIDD.
```

```
E1EDP19-QUALF = '002'.           "Material Ident
E1EDP19-IDTNR = TITEMS-MATNR.
INT_EDIDD-SEGNAM = 'E1EDP19'.
INT_EDIDD-SDATA = E1EDP19.
APPEND INT_EDIDD.
ENDLOOP.
```

110

* Generate an IDoc Header

```
INT_EDIDC-MESTYP = 'ORDERS'.
INT_EDIDC-IDOCTYP = 'ORDERS03'.
INT_EDIDC-SNDPRT = 'LS'.
INT_EDIDC-SNDPRN = SENDER.
INT_EDIDC-SNDPOR = 'ANGELIAX'.
INT_EDIDC-RCVPRT = 'LS'.
```

```
CALL FUNCTION 'OWN_LOGICAL_SYSTEM_GET'
IMPORTING
  OWN_LOGICAL_SYSTEM = OWN_LOGICAL_SYSTEM
EXCEPTIONS
  OWN_LOGICAL_SYSTEM_NOT_DEFINED = 1
  OTHERS = 2.

INT_EDIDC-RCVPRN = OWN_LOGICAL_SYSTEM.
INT_EDIDC-DIRECT = '2'.
```

* - Submit the IDoc for processing.

```

CALL FUNCTION 'IDOC_INBOUND_SYNCHRONOUS'
  EXPORTING
    INT_EDIDC                = INT_EDIDC
  *   ONLINE                  = '0'
  IMPORTING
    DOCNUM                   = IDOCNUM
    ERROR_BEFORE_CALL_APPLICATION = ERROR_BEFORE_CALL_APPLICATION
  TABLES
    INT_EDIDD                = INT_EDIDD
  EXCEPTIONS
    IDOC_NOT_SAVED          = 1
    OTHERS                   = 2.

ENDFUNCTION.

```



Listing 10: Visual Basic example to call the function

```

Public eERROR_BEFORE_CALL_APPLICATION As SAPFunctionsOCX.Parameter
Public eIDOCNUM As SAPFunctionsOCX.Parameter
Public iIAUART As SAPFunctionsOCX.Parameter
Public iSENDER As SAPFunctionsOCX.Parameter
Public iTORDERHEAD As SAPFunctionsOCX.Structure
Public tITEMS As SAPTableFactoryCtrl.Table
Set funcControl = CreateObject("SAP.Functions")
Set TableFactoryCtrl = CreateObject("SAP.TableFactory.1")
Set myR3logonObj = New R3logonObj
funcControl.Connection = myR3logonObj.R3connection
Set fbY_DEMO_IDOC_GENERATE_ORDERS _
= funcControl.Add("Y_DEMO_IDOC_GENERATE_ORDERS")
Set eERROR_BEFORE_CALL_APPLICATION = fbY_DEMO_IDOC_GENERATE_ORDERS.Im
Set eIDOCNUM = fbY_DEMO_IDOC_GENERATE_ORDERS.Imports("IDOCNUM")
Set iIAUART = fbY_DEMO_IDOC_GENERATE_ORDERS.Exports("IAUART")
Set iSENDER = fbY_DEMO_IDOC_GENERATE_ORDERS.Exports("SENDER")
Set iTORDERHEAD =
fbY_DEMO_IDOC_GENERATE_ORDERS.Exports("TORDERHEAD")
Set tITEMS = fbY_DEMO_IDOC_GENERATE_ORDERS.Tables("ITEMS")
' -- add a row to SFLIGHT input table
iTORDERHEAD("VBELN").Value = 'EXT0000001'
iTORDERHEAD("AUART").Value = 'TA'
iTORDERHEAD("BSTNK").Value = 'Via Internet'
tITEMS.AppendRow
linno = tITEMS.RowCount
tITEMS(linno, "VBELP") = '10'
tITEMS(linno, "MATNR") = '100'
tITEMS(linno, "MENGE") = '13'
tITEMS(linno, "GMEIN") = 'PCE'
' -- Call the Function via RFC
fbY_DEMO_IDOC_GENERATE_ORDERS.Call

```



Listing 11: Visual Basic Class to access function

Fehler! Keine gültige Verknüpfung.

5 Batch Input Recording

The batch input (BTCL) recorder (SHDB) is a precious tool to develop inbound IDocs. It records any transaction like a macro recorder. From the recording an ABAP function module can be created. This lets you easily create data input programs, without coding new transactions.

5.1 Recording a Transaction With SHDB

The BTCI recorder lets you record the screen sequences and values entered during a transaction. It is one of the most precious tools in R/3 since release 3.1. It allows a fruitful cooperation between programmer and application consultant.

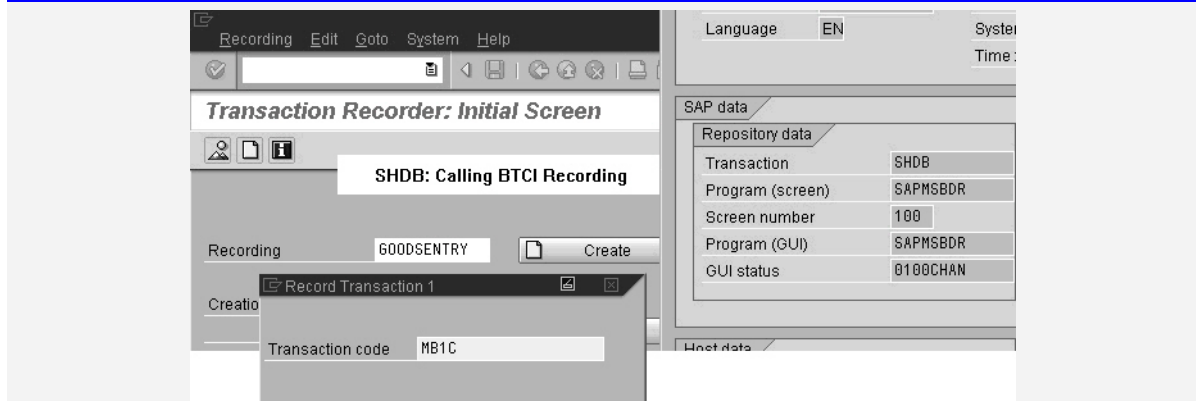
The section below will show you an example of, how the transaction SHDB works. With the recording you can easily create an ABAP, which is able to create BTCI files.

120

Record a session with transaction SHDB

You will be asked for a session name and the name of the transaction to record. Then you can enter the data into the transaction as usual.

Figure 30: Starting a new recording with SHDB



Now the transaction is played and all entries recorded

The following screens will show the usual transaction screens. All entries that you make are recorded together with the screen name and eventual cursor positions.

Figure 31: First screen of MB1C (goods entry)

The screenshot shows the SAP MB1C initial screen. At the top, there is a menu bar with 'Other goods receipts', 'Edit', 'Goto', 'Movement type', 'Environment', and 'System'. Below the menu is a toolbar with various icons. The main title is 'Enter Other Goods Receipts: Initial Screen'. Below the title are four buttons: 'New item', 'To reservation...', 'To order...', and 'WM parameters...'. The screen is divided into several sections:

- Document date:** 30.09.2001
- Posting date:** 30.09.2001
- Material slip:** WIRELESS20019203
- Doc.header text:** Wireless 9203
- Defaults for document items:**
 - Movement type:** 501
 - Plant:** S01
 - Storage location:** S02
 - Special stock:**
 - Reason for movement:**
 - Suggest zero lines:**
- GR/GI slip:**
 - Print
 - Individual slip
 - Indiv.slip w.inspect.text
 - Collective slip

Figure 32: Recorded Detail Screen for goods entry

The screenshot shows the SAP MB1C 'New Items' screen. At the top, there is a menu bar with 'Other goods receipts', 'Edit', 'Goto', 'Movement type', 'Environment', and 'System'. Below the menu is a toolbar with various icons. The main title is 'Enter Other Goods Receipts: New Items'. Below the title are four buttons: 'New item', 'To reservation...', 'To order...', and 'To purchase order...'. The screen is divided into several sections:

- Movement type:** 501 Receipt w/o PO
- G/L account:**
- Vendor:** **Recipient:**
- Items table:**

Item	Material	Quantity	UnE	SLoc	Batch	Re	Plnt
1	LSQ_ROH_11	12	PCE	S02			S01
2		<input type="text"/>		S02			S01
3				S02			S01

⇒

From the recorded session, you can generate an ABAP

After you finished the recording you have the possibility to generate ABAP coding from it. This will be a sequence of statements which can generate a batch input session, which is an exact replay of the recorded one.

Put the coding into a function module

130

To make the recorded code usable for other program, you should make a function module out of it. Starting with release 4.5A the recorded provides a feature to automatically generate such a function module. For earlier release we give the coding of a program which fulfils this task further down.

Test the function module and add eventual loops for detail processing.

135

The created function module should work without modification for testing at least. However, you probably will need to modify it, e.g. by adding a loop for processing multiple entries in a table control (scroll area).