

Thomas Stutenbäumer

# Practical Guide to ABAP®

## Part 1: Conceptual Design, Development, Debugging

- ▶ Getting the most out of ABAP
- ▶ Approaches to debugging from beginner to advanced
- ▶ Guide for understanding and using the SAP Data Dictionary
- ▶ Expert ABAP development techniques

**Thomas Stutenbäumer**

# **Practical Guide to ABAP**

**Part 1: Conceptual Design, Development,  
Debugging**



**ISBN:** 978-3-96012394-1 (ePUB)

**Editor:** Lisa Jackson

**Copyeditor:** Christine Parizo

**Cover Design:** Philip Esch, Martin Munzel

**Cover Photo:** Fotolia: #65986573 © bluebay2014

**Interior Design:** Johann-Christian Hanke

All rights reserved

1st Edition 2016, Gleichen

© 2016 Espresso Tutorials GmbH

**URL:** [www.espresso-tutorials.com](http://www.espresso-tutorials.com)

This work is subject to copyright in its entirety. All rights reserved, especially the rights of translation, recital, reproduction, and duplication. Espresso Tutorials GmbH, Zum Gelenberg 11, 37130 Gleichen, Germany  
Regardless of the care taken in producing texts and illustrations, the publisher, the authors, and editors accept no legal liability whatsoever for possible mistakes and their consequences.

**Feedback:**

We greatly appreciate any kind of feedback you have concerning this book.  
Please mail us at [info@espresso-tutorials.com](mailto:info@espresso-tutorials.com).

## **Thank you for purchasing this book from Espresso Tutorials!**

Like a cup of espresso coffee, Espresso Tutorials SAP books are concise and effective. We know that your time is valuable and we deliver information in a succinct and straightforward manner. It only takes our readers a short amount of time to consume SAP concepts. Our books are well recognized in the industry for leveraging tutorial-style instruction and videos to show you step by step how to successfully work with SAP.

Check out our YouTube channel to watch our videos at <https://www.youtube.com/user/EspressoTutorials>.

If you are interested in SAP Finance and Controlling, join us at <http://www.fico-forum.com/forum2/> to get your SAP questions answered and contribute to discussions.

### **Related titles from Espresso Tutorials:**

- ▶ Sydnie McConnell & Martin Munzel: First Steps in SAP® (2nd, extended edition)  
<http://5045.espresso-tutorials.com>
- ▶ Antje Kunz: SAP® Legacy System Migration Workbench (LSMW)  
<http://5051.espresso-tutorials.com>
- ▶ Darren Hague: Universal Worklist with SAP NetWeaver® Portal  
<http://5076.espresso-tutorials.com>
- ▶ Michal Krawczyk: SAP® SOA Integration  
<http://5077.espresso-tutorials.com>
- ▶ Dominique Alfermann, Stefan Hartmann, Benedikt Engel: SAP® HANA Advanced Modeling  
<http://4110.espresso-tutorials.com>
- ▶ Kathi Kones: SAP List Viewer (ALV) – A Practical Guide for ABAP Developers

<http://5112.espresso-tutorials.com>

- ▶ Jelena Perfiljeva: What on Earth is an SAP IDoc?

<http://5130.espresso-tutorials.com>

- ▶ Thomas Stutenbäumer: Practical Guide to ABAP. Part 2: Performance, Enhancements, Transports

<http://5138.espresso-tutorials.com>



*All you can read:*

## The SAP eBook Library

<http://free.espresso-tutorials.com>



- ▶ Annual online subscription
- ▶ SAP information at your fingertips
- ▶ Free 30-day trial

# Table of Contents

[Cover](#)

[Title](#)

[Copyright / Imprint](#)

[Preface](#)

[1 Requirements and solution design](#)

[1.1 A request for practice](#)

[1.2 Considerations regarding implementation requests](#)

[1.3 Rough concept of the solution](#)

[1.4 Detailed concept of the solution](#)

[2 SAP Data Dictionary](#)

[2.1 SE11 – Entry to ABAP dictionary](#)

[2.2 Table properties](#)

[2.3 Table indexes](#)

[2.4 Foreign keys](#)

[2.5 Currency and quantity fields](#)

[2.6 Append structures](#)

[2.7 Database table ZGENERALCONTRACT](#)

[2.8 Enhancement categories of tables and structures](#)

[2.9 Views instead of select with join option](#)

[2.10 Conversion modules of domains](#)

[2.11 Table maintenance generator](#)

[2.12 Search help](#)

[2.13 Lock objects and lock modules](#)

[2.14 Database utility](#)

[2.15 Changing table contents](#)

[2.16 Who has changed the table content?](#)

[3 Proper debugging](#)

[3.1 Classical and new ABAP debugger](#)

- [3.2 Start, stop, and change debugger](#)
- [3.3 Breakpoint by command – First choice for troubleshooting](#)
- [3.4 Watchpoints](#)
- [3.5 Reverse action in debugger](#)
- [3.6 Program progress in the debugger – The ABAP stack](#)
- [3.7 Local and global variables](#)
- [3.8 Change table contents in running programs](#)
- [3.9 Special debugging](#)

## 4 ABAP development

- [4.1 Documenting programs](#)
- [4.2 Structure and readability of programs](#)
- [4.3 Error handling in programs](#)
- [4.4 Extended check of programs](#)
- [4.5 ABAP version management](#)
- [4.6 Compare source codes of different programs](#)
- [4.7 Log application usage](#)
- [4.8 Expensive work – Treatment of time slices](#)
- [4.9 Data types TIMESTAMP and TIMESTAMPL](#)
- [4.10 Field symbols and ASSIGN command](#)
- [4.11 Perfect output – ALV grid control](#)
- [4.12 Download and upload data](#)
- [4.13 ABAP code lines in tables](#)
- [4.14 Send e-mail from SAP](#)
- [4.15 “Dirty assign”](#)
- [4.16 Wait for update](#)
- [4.17 Write change documents](#)
- [4.18 Dynamic SELECT statement](#)
- [4.19 Number ranges and transaction SNRO](#)
- [4.20 Call transaction method and transaction SM35](#)

## 5 What has been implemented at this time

[A Author](#)

[B Sample programs](#)

[C Useful SAP transactions](#)

D Useful SAP database tables

E BDC\_OKCODE

F Disclaimer

# Preface

There are a lot of publications for the SAP computer language ABAP. Why another one? The existing literature with regard to ABAP often deals extensively with ABAP development basics or is so specialized that only a fraction of the content is relevant for practice. On one hand, the reader is overloaded by the amount of information; on the other, he doesn't receive enough information.

This book goes another way. Based on many years of development experience, I've collected methods and procedures used every day for the implementation of customer requirements. The results build upon and go beyond the basic ABAP knowledge that are necessary prerequisites for understanding this book.

Topics included here are those a professional developer applies daily and gets to know well. Most developers use only a fraction of what ABAP offers – the essence of ABAP – which they need for their daily work. That's what makes them professional developers. That is the essence of the content of this book.

The current ABAP developer is not a pure program coder. Technical and detailed knowledge of the ABAP programming language alone does not help with the demands placed on the developer. The developer must learn to put himself in the position of the users and capture, with empathy, what the users need and expect.

This means that an ABAP developer must deal with related topics, such as the various possibilities and customization options of SAP applications. The developer also needs knowledge of SAP basics, such as how to download files from SAP directories to workstations or how to import transport developments in other SAP systems.

The knowledge gathered in this book is based on my personal experience in 10 years of ABAP development for utility companies. The utility industry uses SAP core modules for financial accounting, materials management, and maintenance. They also work with an SAP industry-specific solution for the utilities industry (SAP IS-U). The liberalization of the German energy market over the past years has led to huge IT applications and requires many adaptations.

Each of the methods and procedures presented in this book can be transferred to other SAP applications and other lines of business. Under this premise, to take into account the many facets of the tasks of an ABAP developer, topics in this book go beyond the ABAP computer language.

The first volume of this practical guide begins with customer requirements and explains how to build a realization concept. I'll describe the most important aspects of the SAP Data Dictionary in detail and move on to the SAP debugger, which serves as a tool to develop programs and localizes errors in SAP programs. In the last chapter, I'll explain frequently used methods in the ABAP programming language.

The [first chapter](#) shows how a concrete requirement leads to solution attempts. Based on this implementation, I will go on to explain a number of procedures. Before writing the first line of a program, the developer must have a complete understanding of user requirements and develop a solution strategy. Without a strategy, different procedures can lead to user disappointment and developer displeasure because the developed programs have to be repeatedly modified or extended.

The [second chapter](#) belongs to the SAP Data Dictionary and its functions. The data model is the basis of any program. Based on customer requirements, the developer has to decide where the application data has to be stored: Are standard SAP tables suitable, or are Z-tables better as an enlargement of the SAP data model? Are database tables or customizing tables more helpful? Must table contents be changed by the user in the

production system? Each requirement leads to another realization of the data model.

The [third chapter](#) describes how to use the SAP debugger. This is the tool of choice for program development and for analyzing program errors. SAP distinguishes between the classical and new debuggers. The new debugger offers more possibilities for program and error analysis. In addition to a clear representation of program code, the local and global variables appear at the same time. The debugger enables forward and backward jumping in the program. Variables can be manipulated and functionalities can be tested during the runtime of the program under changed conditions.

The [fourth chapter](#) is dedicated to enlarged functions and developer tricks in ABAP. In particular, the need and the possibilities of program documentation and error handling are explained. I'll show program tricks, for example ALV grid lists, in the list and dialog processing, upload and download for data, the treatment of time slices, the call-transaction procedure, sending e-mail from SAP, the "dirty assign" method, and dynamic SELECT statements.

The [fifth and final chapter](#) is a summary detailing which requirements in [Chapter 1](#) can be solved with the explained methods and procedures and which ones require further knowledge.

The second volume of this practical guide introduces more elements of ABAP programming: the influence of the developer on the performance, modifications, and extensions to the SAP standard; customizing in SAP applications; data transportation; ways of troubleshooting; and using the SAP support portal when troubleshooting errors in standard SAP programs.

I'm convinced that this practical guide to SAP ABAP gives you the tools necessary to implement most of your daily requirements. The knowledge of these methods and procedures can lead to professional success. It is your personal skills and your ability to understand the user's needs to implement efficient programs using the explained tools. I raise no claim to

completeness. If you feel important sections or instructions are missing from, or are erroneous in, this book, please contact me. Your information can be included in a future edition of the book.

---

In the text, we use boxes to highlight important information. Each box is also equipped with a pictogram.

### Hint



Hints give practical tips for dealing with the current topic.

### Example



Examples serve to illustrate a topic better.

### Warning



Warnings point to possible sources of error or stumbling blocks in connection with a topic.

All screenshots printed in this book are subject to SAP SE copyright. For simplicity, we have waived printing the copyright separately under each screenshot.

# 1 Requirements and solution design

**Behind every new program is a customer requirement. This chapter uses a fictional customer request to show how a concept can lead to an implementation in SAP. This example is applied as far as possible in all explanations of methods and procedures contained in this guide.**

“Too much action, not enough thinking.” This quote from a sales employee shows a still-existing phenomenon: Functionalities that are promised to customers are not always implemented in the desired form. Or the application and the development time are underestimated. Or the developer receives insufficient descriptions of what it is that needs to be implemented. Therefore, IT service providers move from one programming disaster to the next, and customers’ frustrations grow.

What can be done to avoid these types of situations?

- ▶ Have the developer join in on customer discussions
- ▶ Only estimate the effort of experienced developers
- ▶ Agree on appointments between users and developers, where the required functionalities are discussed in detail
- ▶ Verbally summarize the requirements submitted in writing
- ▶ Do not paralyze the developers with bloated project requirements
- ▶ Arrange test cases and scopes, preselect test employees, and set a test duration
- ▶ Make sufficient hardware resources available in a timely manner
- ▶ Train the users shortly before rollout on the new application(s)

- ▶ Document the programs in a user-friendly manner, or enable another ABAP developer to maintain the work

This list is not comprehensive. For each IT service provider or IT department, one can advise the other. It is not my intention to provide organizational consulting; each good employee knows the weak points in his company best. Rather, I want to point out the pure knowledge of programming and the program techniques for successful program development. I will illustrate programming methods and procedures with practical examples. A sample request serves as the basis for further explanation of procedures and methods in the following chapters. The example shows the dimensions of a simple request and the resulting necessary work of an ABAP developer.

## 1.1 A request for practice

### Marking customers with high turnover



Utility companies that have high customer turnover should be marked for follow-up customer service. Utilities control the meter reading and thus indirectly the billing of their services through meter reading units. The aim of this action is to summarize customers with a high turnover in a new meter reading unit.

The following are requirements for our scenario.

- ▶ The contract accounts of high turnover customers should get the new *account class* “First Customer”
- ▶ A *business partner* with a contract account of account class “First Customer” should be identified in the business partner management field with the applicable customer service representative.
- ▶ In the SAP Customer Interaction Center (SAP CIC), the account class “First Customer” should be marked with a special entry in the business partner contacts so the user can easily identify the selected business partner.
- ▶ The *contract* of a customer should be assigned the label “general contract data” of the “First Customer” account. With general contract designations, limits, payment arrangements, and contact information can be maintained. A *general contract* contains the attributes: General Contract Number, General Contract Title, and General Contract Keeper. The data to maintain the general contracts should be uploaded to the SAP system from a local workstation.

- ▶ A conversion for the *meter reading unit* of a “First Customer” should be carried out on a predetermined date.
- ▶ By changing the meter reading unit, the *meter reading portion* (also called meter reading region) has to be changed. Moreover, to change the meter reading portion, the billing plan has to be changed. A program should allow the deadline-related conversion of the billing plan from “First Customer”.

## 1.2 Considerations regarding implementation requests

Regardless of how requests arise, creating a “First Customer” in different standard SAP programs is necessary.

1. Creating the contract account assumes customizing a new contract class.
2. Creating a “First Customer” in the business partner maintenance field requires additional fields in the display of business partners.
3. The assignment of general contracts to contracts requires additional input and output fields in SAP Contract Management.

Also, a new database table and a new search help are required.

- ▶ The input of general contracts requires creating a new database table to store the data.
- ▶ For correct assigning of a general contract to a contract, the user needs a search help.

For a deadline-related conversion of “First Customer,” you need a customer-specific program.

- ▶ The assignment of the new account class “First Customer” to selected contract accounts requires the development of a program to convert the account class for the selected business partner.
- ▶ For storing general contract data in a customer-specific table in an SAP system, uploads from an external file to an SAP system are needed.
- ▶ The meter reading unit is a field in the *time slice* of installations. A program has to be developed to create a new time slice for an installation with a new meter reading unit for a specific deadline.
- ▶ To introduce a new meter reading unit and a new meter reading portion, new billing plans have to be created to consider the new attributes. A program has to be developed that checks the portion for each billing

plan in a new billing period. If the portion in the billing plan is correct, no further actions are necessary. Otherwise, the old billing plan has to be deleted and a new billing plan created. If financial compensations on the old billing plan are carried out, the old billing plan cannot be deleted by using the new program. Before deleting such a billing plan, the compensation has to be manually withdrawn. After the conversion to a new billing plan, the compensation has to be posted manually to the new billing plan.

- ▶ Creating “First Customer” in SAP CIC through a special contact entry in the business partner overview requires writing a program that creates a special contact entry for the selected business partner.

The developer has to decide which way he can best implement the requirements in an SAP system.

### 1.3 Rough concept of the solution

The *data model* is the basic format for each program. An ABAP developer has to know where data can be found, whether standard SAP tables can be enhanced, and whether a new customer-specific table has to be created.

The example in Figure 1.1 is of an easy data model.

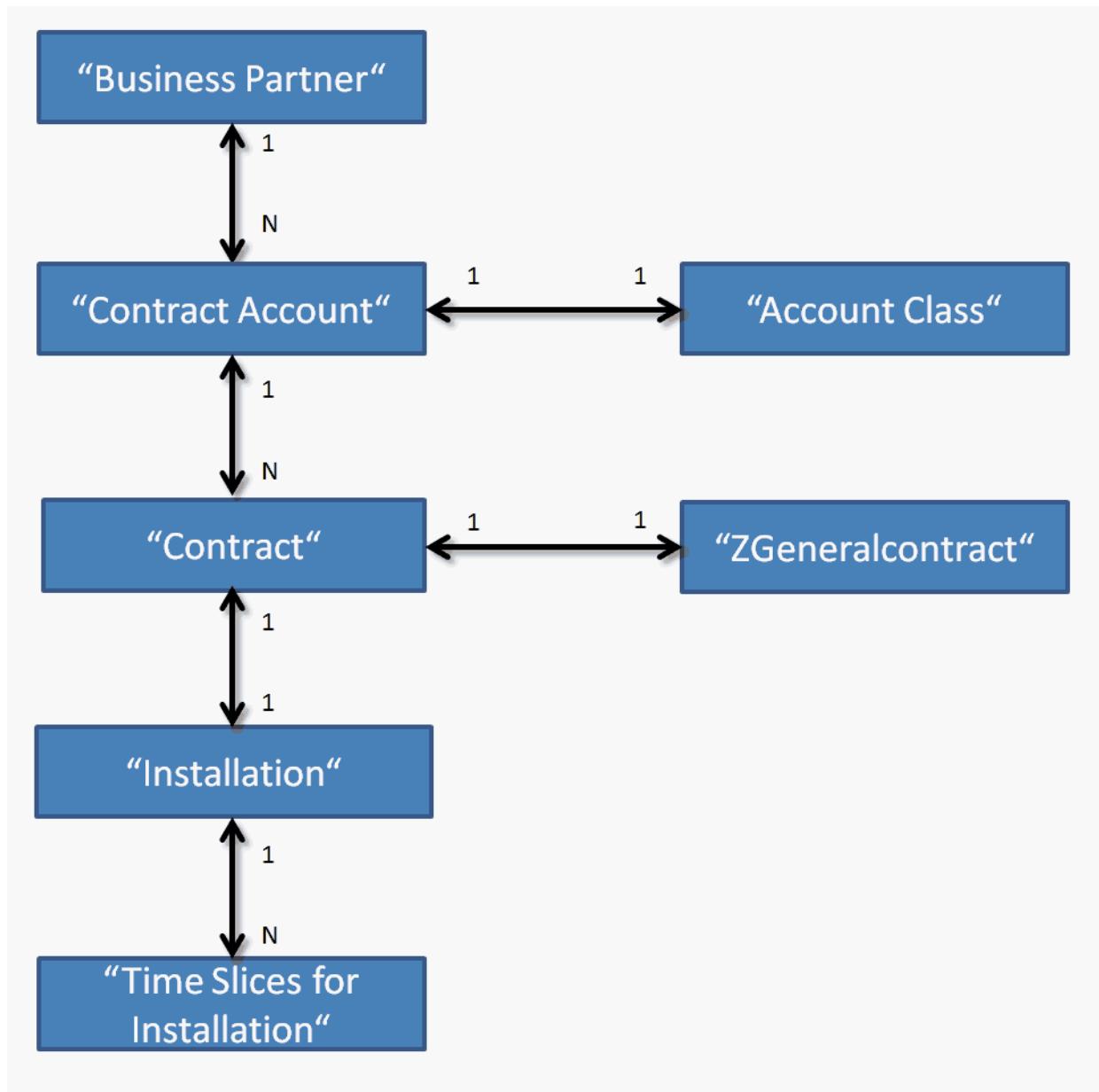


Figure 1.1: Data model for the requirement

One *business partner* can own several contract accounts. One *contract account* always has exactly one *account class*. One contract account can be assigned any number of contracts. One *contract* always has exactly one *general contract*. Each contract is assigned to exactly one installation. One *installation* can own any number of *installation time slices*. For general contract maintenance, there has to be a customer-specific database table. The assignment from a general contract to a contract is possible by adding the **GENERAL\_CONTRACT\_NUMBER** field in the database table for contract EVER. This field is created in the customer-include CI\_EVER (refer to field enhancements for contracts in [Section 2.6](#)).

The following activities are necessary to store data.

- ▶ Create table ZGENERALCONTRACT to store general contract data.
- ▶ Enhance fields in a contract for a connection between contracts and a general contract, including several fields for the contract and contract information.
- ▶ Customize a *check table* for the new account class. The check table guarantees that the user can only select account classes stored in this check table.

The requirements need some extensions of standard SAP programs (see [Section 1.4.1](#)) to:

- ▶ Display the account class in the contract account
- ▶ Display the attributes of “First Customer” in the business partner data
- ▶ Display general contracts in the contract management data
- ▶ Display specific business partner contacts in SAP CIC

Programs need to be developed for deadline conversions (see [Section 1.4.2](#)) to:

- ▶ Upload general contract account data

- ▶ Convert selected contract accounts to the new account class
- ▶ Convert the meter reading unit with a new time slice for selected installations
- ▶ Create specific business partner contacts for partners with the new account class
- ▶ Adjust selected billing plans

## **1.4 Detailed concept of the solution**

### **1.4.1 Adaption of standard SAP programs**

The adaption of a standard SAP program is a comprehensive chapter in *Practical Guide to SAP ABAP: Part 2*. The following is only a summary of the requirements.

#### **Display the contract class “First Customer” in a contract account**

The display is reached through customizing the new account class in the check table for the field **ACCOUNT CLASS** of a contract account. Customizing is addressed in the *Practical Guide to SAP ABAP: Part 2*.

#### **Display of “First Customer” in the business partner management view**

For a modification-free enhancement of a business partner, use **TRANSACTION BUPT**. You will find the solution for this problem formulation in the *Practical Guide to SAP ABAP: Part 2*, in the chapter titled, “Modifications and enhancements to SAP standards.”

#### **Display of general contracts in contract management**

Modification-free enhancements for fields in contract management are realized with *user exits* and *screen exits*. The solution is also found in the *Practical Guide to SAP ABAP: Part 2*, in the chapter, “Modifications and enhancements to SAP standards.”

### **1.4.2 Conversion programs**

Conversion programs must have an analyze function, which shows the user which objects to change in which form and which objects not to convert. Note that objects should not be converted more than once. A conversion program has to be “restart-able.” A record conversion program should only be run after a complete analysis of the current objects, or records, is

performed. The development of these programs is explained in the chapter about programming ALV grid outputs (see [Section 4.11](#)).

Let's focus on the required functionality of the programs to be developed.

## Conversion to a new account class

These programs have to be able to identify account classes that aren't customized.

## Time slices for meter reading units of installations

SAP software stores data object history in specific tables. The records are saved with a valid time frame, meaning they use a from-date and a to-date. The single records are named through a *time slice*.

To maintain the history of an installation concerning the meter reading unit, the time slices of an installation cannot have gaps in time nor overlap in time. So, how do you perform the conversion of meter reading units?

Initially, the solution is very simple: The installation gets a new time slice with the deadline. The existing time slice calculates the to-date with the deadline date minus one day. Other items to consider include:

- ▶ Is there an active contract for the installation?
- ▶ Does the installation already have the new meter reading unit for the conversion appointment?
- ▶ Does the installation have a to-date prior to the conversion date minus one day?

In our example, the user tests the meter reading unit far in the future and then chooses an earlier conversion date. This scenario leads to a row of case differentiations, which require different solutions. In our example, the user requirement – although it shows only one special case – overruns the

estimated costs. (Therefore, be careful when editing time slices and estimate the costs.)

For working with difficult time slices, refer to [Section 4.9](#).

## Business partner contact

To create business partner contacts in your program, you can use standard SAP function modules. Business partner contacts use a number range to enable continuous numbering. The development of this program is explained in [Section 4.19](#).

## Billing plans

The program must perform the basic checks for correct utility billing plans. Consider the following questions:

- ▶ Does the installation have a time slice with the new meter reading unit?
- ▶ Is the new meter reading unit assigned to a portion?
- ▶ Are there scheduled master data readings for the portion?
- ▶ Are there scheduled master data readings for the meter reading unit?
- ▶ Are there scheduled master data readings for the billing plan?
- ▶ Are there compensations for billing plans to convert?

Deleting and creating billing plans is possible with standard SAP transactions, which are well developed in SAP. For example, the transaction for deleting billing plans does not work if compensatory payments have occurred. Therefore, it makes sense to use these transactions in the program for the conversion of billing plans. In ABAP programs, the command **CALL TRANSACTION** is used. Using this command is possible only after a series of preparatory commands. The procedure is explained in [Section 4.20](#).

To create the realization concept, the performance of the functions to be developed needs to be known by all those involved.

In the first step, the developer needs to determine the type of data he needs for the functions. So, let's begin with the basics of data storage in SAP systems.

## 2 SAP Data Dictionary

**The basis of any ABAP program is the data processed by the program. Before writing the first line of code, the developer must know where the data to be processed can be found and how the data is stored in the SAP system.**

An SAP system has tens of thousands of database tables. They contain data or are used as check tables for entries in the database tables. A central tool, the SAP Data Dictionary, contains the overview of this wealth of information.

Information about tables and data structures and their relationships to each other are managed in the SAP Data Dictionary. It serves to create tables and define fields, as well as to assign data types.

Experience dealing with this Data Dictionary is a fundamental prerequisite for an ABAP developer. You can call it with **TRANSACTION SE11** (see Figure 2.1).

## 2.1 SE11 – Entry to ABAP dictionary

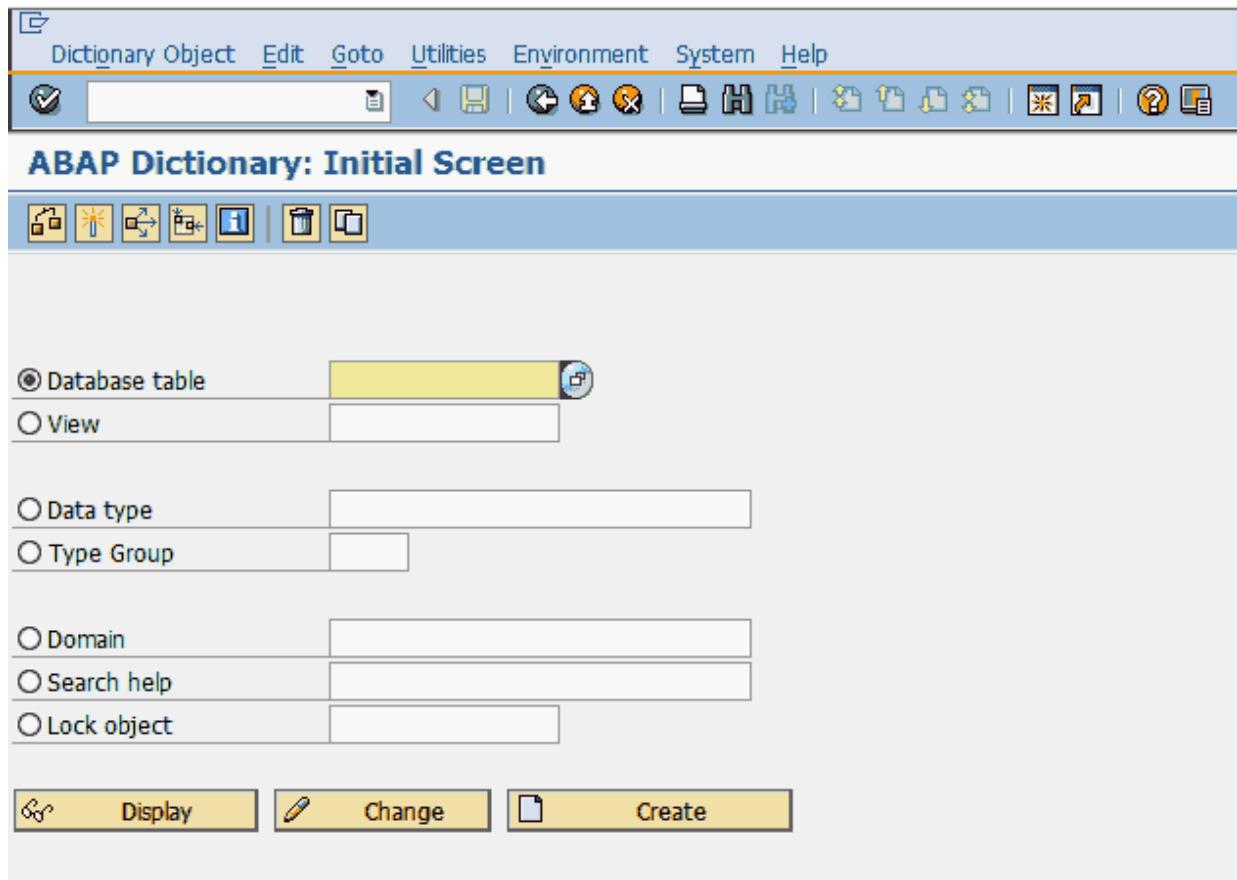


Figure 2.1: Transaction SE11 – the SAP ABAP Data Dictionary

The SAP ABAP Data Dictionary provides data to create, modify, test, activate, delete, and copy:

- ▶ Tables
- ▶ Views
- ▶ Data types (data elements, structures, and table types)
- ▶ Type groups
- ▶ Domains
- ▶ Search help
- ▶ Lock objects

In addition, it offers a set of tools for setting the properties of the dictionary objects, their extensions, their uses, and their environmental analyses.

## Environment analysis



By clicking the icon, you come to the environmental analysis of a dictionary object. The environment analysis shows the relation of a dictionary object to other dictionary objects (see Figure 2.2).

The screenshot shows the SAP Dictionary Environment Analysis interface. The top menu bar includes Objects, Edit, Goto, Settings, Utilities, Environment, System, and Help. The toolbar below has various icons for navigation and selection. The main title is "Environment Analysis". Below the title are four tables:

Data element	Short text	Package
<input type="checkbox"/> BU_BP_VALID_FROM	Validity Start BUT000 BP Data	S_BUPA_GENERAL
<input type="checkbox"/> BU_BP_VALID_TO	Validity End BUT000 BP Data	S_BUPA_GENERAL
<input type="checkbox"/> BU_PARTNER	Business Partner Number	S_BUPA_GENERAL
<input type="checkbox"/> BU_XPCPT	Business Purpose Completed Flag	S_BUPA_GENERAL
<input type="checkbox"/> MANDT	Client	SZS

Search Help Name	Short Description	Package
<input type="checkbox"/> BUPA	Business Partner	S_BUPA_GENERAL

Structure name	Short Description	Package
<input type="checkbox"/> BUS000_DAT	BP: General Data I (Data Fields - External)	S_BUPA_GENERAL
<input type="checkbox"/> BUS000_INT	BP: General Data I (Data Fields - Internal)	S_BUPA_GENERAL

Table Name	Short text	Package
<input type="checkbox"/> T000	Clients	STRM

Figure 2.2: Environment analysis of tables

## Where-used list

By clicking the **WHERE-USED LIST** icon, you can analyze which SAP objects (programs, classes, etc.) a dictionary object uses. Figure 2.3



shows an example of the where-used list of the **TABLE** BUT000 (business partner).

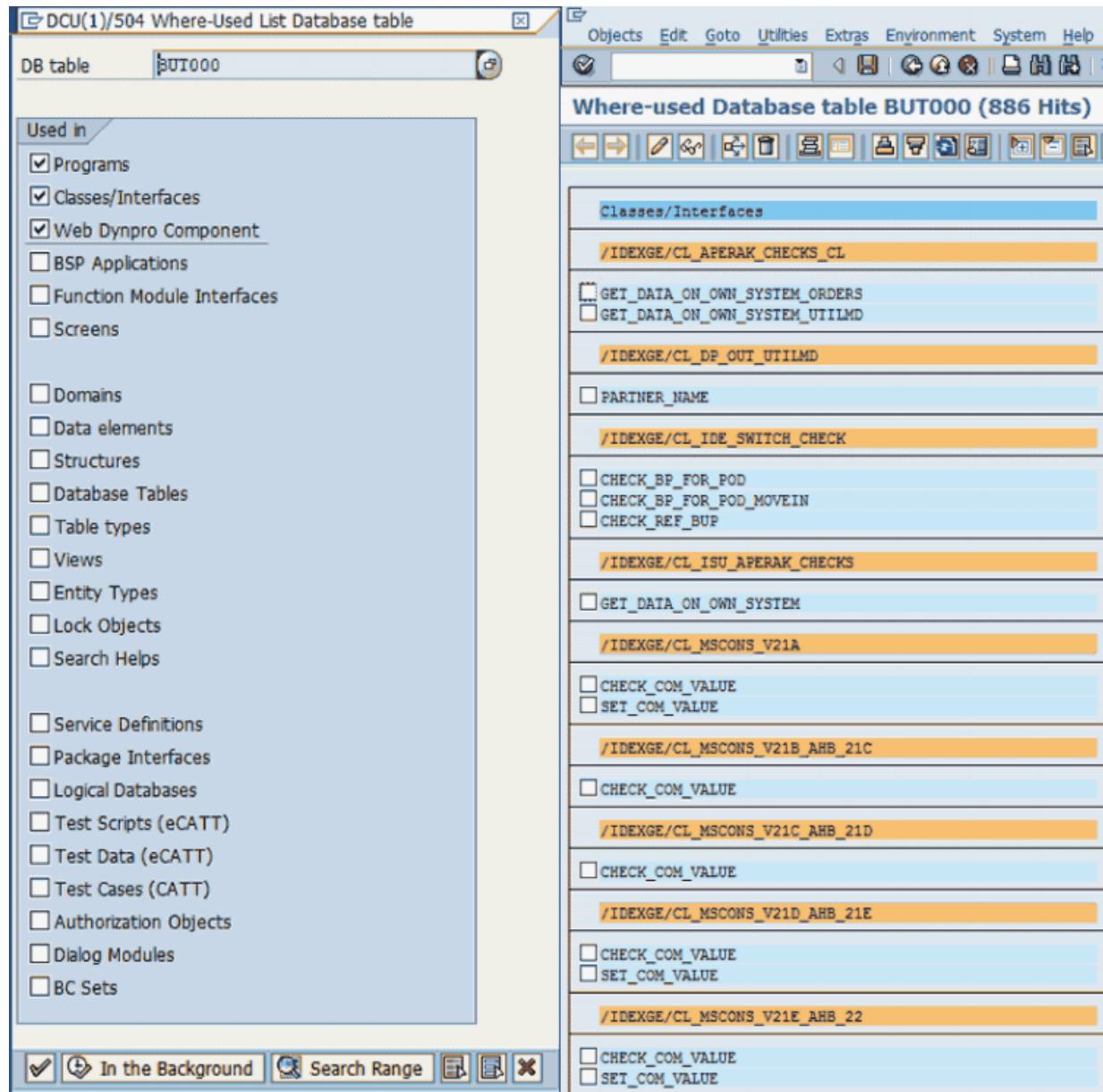


Figure 2.3: Where-used list of database tables

## 2.2 Table properties

If a table, such as the one in Figure 2.4, owns the first key field **MANDT** or **CLIENT** with the data element **MANDT**, the table is client-specific. This means that the table entries are valid, visible, and changeable only in the named SAP client.

The screenshot shows the SAP Dictionary: Display Table interface for table BUT000. The table has 10 fields:

Field	Key	Initi...	Data element	Data Type	Length	Decim...	Short Description
CLIENT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	MANDT	CLNT	3	0	Client
PARTNER	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	BU_PARTNER	CHAR	10	0	Business Partner Number
.INCLUDE	<input type="checkbox"/>	<input type="checkbox"/>	BUS000_DAT	STRU	0	0	BP: General Data I (Data Fields - External)
.INCLUDE	<input type="checkbox"/>	<input type="checkbox"/>	BUS000A	STRU	0	0	CBP: General data (independent of partner cat.)
.INCLUDE	<input type="checkbox"/>	<input type="checkbox"/>	BUS000AINI	STRU	0	0	CBP: General Data (not dep. on Part. Cat.), Initial Screen
TYPE	<input type="checkbox"/>	<input type="checkbox"/>	BU_TYPE	CHAR	1	0	Business partner category
BPKIND	<input type="checkbox"/>	<input type="checkbox"/>	BU_BPKIND	CHAR	4	0	Business Partner Type
BU_GROUP	<input type="checkbox"/>	<input type="checkbox"/>	BU_GROUP	CHAR	4	0	Business Partner Grouping

Figure 2.4: Client-specific database table BUT000

Tables without these key fields are client-independent. That means the table entries are valid, visible, and changeable in every SAP client of the SAP system (see Figure 2.5).

The screenshot shows the SAP Dictionary: Display Table interface for the table TADIR. The table contains 22 fields, with the first 7 shown in the visible area:

Field	Key	Initi...	Data element	Data Type	Length	Decim...	Short Description
PGMID	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	PGMID	CHAR	4	0	Program ID in Requests and Tasks
OBJECT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	TROBJTYPE	CHAR	4	0	Object Type
OBJ_NAME	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	SOBJ_NAME	CHAR	40	0	Object Name in Object Directory
KORRNUM	<input type="checkbox"/>	<input type="checkbox"/>	TRKORR_OLD	CHAR	10	0	Request/task up to and including Release 3.0
SRCSYSTEM	<input type="checkbox"/>	<input type="checkbox"/>	SRCSYSTEM	CHAR	10	0	Original System of Object
AUTHOR	<input type="checkbox"/>	<input type="checkbox"/>	RESPONSIBL	CHAR	12	0	Person Responsible for a Repository Object
SRCDEP	<input type="checkbox"/>	<input type="checkbox"/>	REPAIR	CHAR	1	0	Repair Flag for Repository Object
DEVCLASS	<input type="checkbox"/>	<input type="checkbox"/>	DEVCLASS	CHAR	30	0	Package

Figure 2.5: Client-independent table TADIR

Figure 2.6 shows the **DELIVERY CLASS** and **MAINTENANCE** properties of a table. These show the options available to display and change data entries.

The screenshot shows the SAP Dictionary: Display Table interface for the table BUT000. The 'Delivery and Maintenance' tab is selected. Two specific properties are highlighted with red boxes:

- Delivery Class:** A dropdown menu containing the value 'A Application table (master and transaction data)'.
- Data Browser/Table View Maint.:** A dropdown menu containing the value 'Display/Maintenance Allowed with Restrictions'.

Figure 2.6: Delivery and maintenance properties of a table

In most cases, the following delivery classes are used:

- ▶ A = Application table
- ▶ C = Customizing table "Table:Customizing table"

More delivery classes and their meanings can be found by hitting the “F1” help key on the **DELIVERY CLASS** field in **TRANSACTION SE11**.

*Application tables* contain master and transaction data. *Customizing tables* are, as a rule, check tables. They limit the entries in the application tables to the entries contained in the check table to prevent incorrect entries by the user.

Entries in customizing tables are made possible by table maintenance views. These entries are maintained with the help of **TRANSACTION SM30** (entry to maintenance view). These entries are carried out in test systems and are finalized with the help of customizing transports. All SAP system settings, which are done with the help of **TRANSACTION SPRO** (customizing: execute project), use table maintenance views.

In the field **DATA BROWSER/TABLE VIEW MAINTENANCE**, you can choose one of the following:

- ▶ Display/Maintenance Allowed
- ▶ Display/Maintenance Allowed with Restrictions
- ▶ Display/Maintenance Not Allowed

These settings affect the maintenance of table entries. With the setting “Display/Maintenance Not Allowed,” table entries cannot be displayed, changed, or inserted.

“Display/Maintenance Allowed with Restrictions” displays data sets with **TRANSACTION SE16**, **SE16N**, or **SM30**, but does not allow data sets to be changed or inserted. The option “Display/Maintenance Allowed” permits records to be changed and inserted. The technical settings of a table are used to describe the properties of the table for the underlying database system (see Figure 2.7).

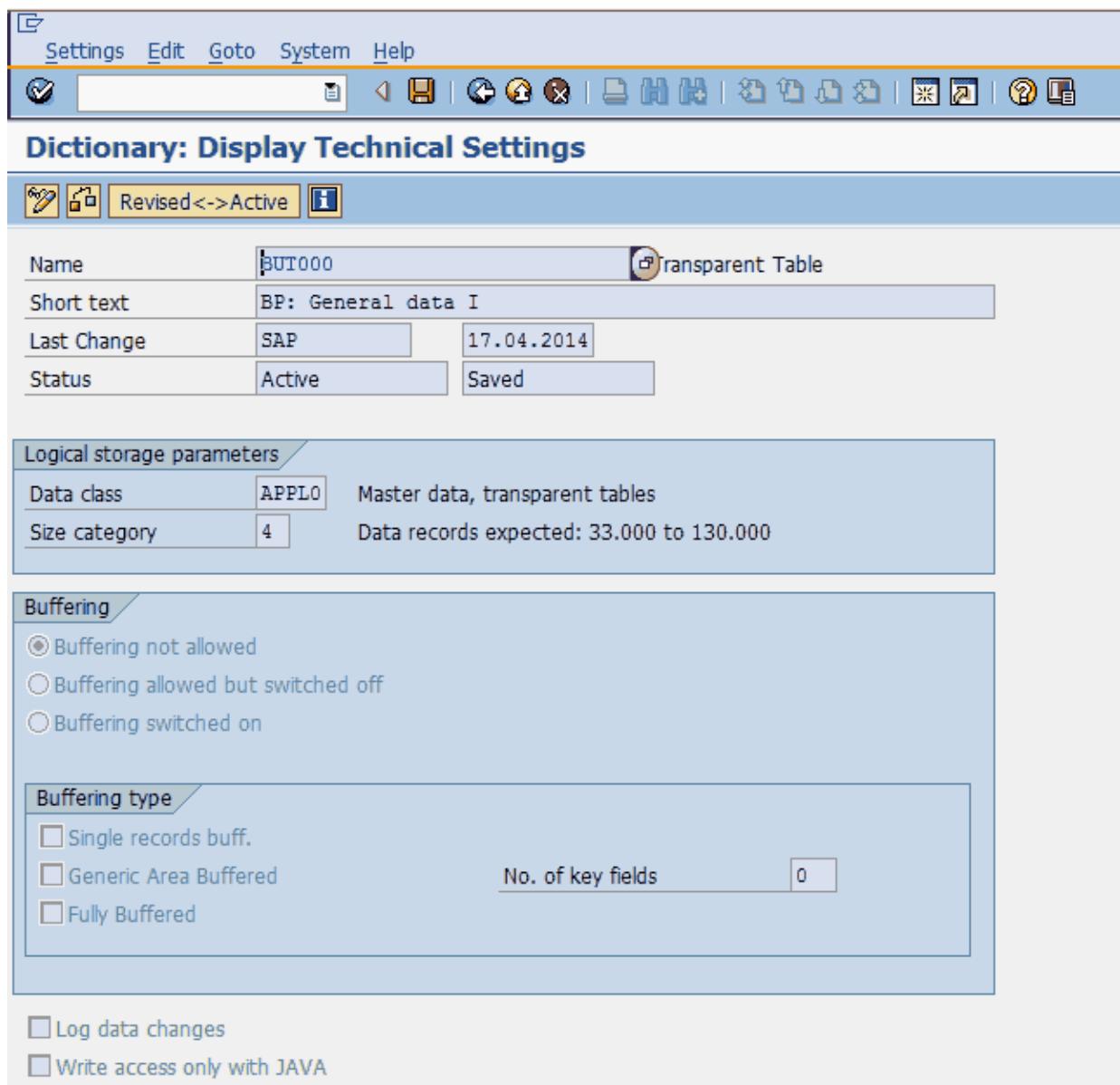


Figure 2.7: Technical settings of a table

**DATA CLASS** defines where records are stored in the physical memory (this is only valid for Oracle and Informix database systems). The **SIZE CATEGORY** describes the expected record amount of the table. This specification reserves an initial data location when creating the table in the database. The place is expanded dynamically according to the current size of the table. Buffering describes the property of databases that reserves certain parts of the database table for high performance in the main memory. Buffering table contents is of interest when check tables have only a few records.

Other table properties, such as indexes and check tables, are explained in later chapters.

Now let's consider the request shown in the first chapter to create a customer-specific Z-table for general contracts. This table contains master data. The records should be changed or inserted into the production system, so the table must have the delivery class "A" and the property "Display/Maintenance Allowed." The class is **APPL0**. The size category needs to be defined and is dependent on the amount of records expected. A buffering of the general contract data is not necessary. In this chapter on table view maintenance, you will learn how to create a transaction for the maintenance of table contents.

## 2.3 Table indexes

*Table indexes* are created to find necessary records in data tables with many data sets. Create a first index automatically by defining primary keys.

For example, look at Figure 2.8 for the contract account (FKKVKP). This table has the primary keys MANDT (client), VKONT (contract account number), and GPART (business partner number).

Field	Key	Initi...	Data element	Data Type	Length	Decim...	Short Description
MANDT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	MANDT	CLNT	3	0	Client
VKONT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	VKONT_KK	CHAR	12	0	Contract Account Number
GPART	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	GPART_KK	CHAR	10	0	Business Partner Number
VKBEZ	<input type="checkbox"/>	<input type="checkbox"/>	VKBEZ_KK	CHAR	35	0	Contract account name
.INCLUDE	<input type="checkbox"/>	<input type="checkbox"/>	FKKVKP_I	STRU	0	0	Data include for table FKKVKP
ERDAT	<input type="checkbox"/>	<input type="checkbox"/>	ERDAT	DATS	8	0	Date on Which Record Was Created
ERNAM	<input type="checkbox"/>	<input type="checkbox"/>	ERNAM	CHAR	12	0	Name of Person who Created the Object
AEDATP	<input type="checkbox"/>	<input type="checkbox"/>	AEDAT	DATS	8	0	Changed On
AENAMP	<input type="checkbox"/>	<input type="checkbox"/>	AENAM	CHAR	12	0	Name of Person Who Changed Object

Figure 2.8: Table FKKVKP for maintenance of contract accounts

A SELECT statement with the fields “Contract Account Number” and “Business Partner Number” in the WHERE condition would find the record quickly because of the implicit table index of primary keys.

The example described in [Chapter 1](#) requires storing the new value “First Customer” in some records of the field **ACCOUNT CLASS**. If programs use the property **ACCOUNT CLASS** to find contract accounts, the necessary records are found quickly with the help of an index on the field **ACCOUNT CLASS**.

Check which indexes are defined for the table FKKVKP by clicking the **Indexes...** button in **TRANSACTION SE11**.

Ind	Ex...	Short text	Status	Unique	Author	Date	DB index name	DBS1	IE	DBS2	DBS3	DBS4	DBS5
1		Index for Partner Number	Active		SAP	06.08.2014	FKKVKP~1						
2		Alternative payer	Active		STUTETH	24.11.2014	FKKVKP~2		E				
3		Alternative Payee	Active		SAP	06.08.2014	FKKVKP~3		E				
4		Alternative Invoice Recipient	Active		SAP	06.08.2014	FKKVKP~4		E				
5		Alternative Dunning Recipient	Active		SAP	06.08.2014	FKKVKP~5		E				
6		Alternative Contract Account for Coll...	Active		SAP	06.08.2014	FKKVKP~6		E				
7		Payment Contract Account	Active		SAP	06.08.2014	FKKVKP~7		E				
8		GUID for Business Agreement	Active		SAP	06.08.2014	FKKVKP~8		E				
9		Clerk Responsible	Active		SAP	06.08.2014		O	E				

Figure 2.9: Indexes of table FKKVK

Figure 2.9 shows that there are some already defined indexes for table FKKVKP. Customer-specific indexes, which are added to standard SAP indexes, own an **INDEX DETECTION** (first column) with “Z” as the first character.

You can analyze an index by double clicking on the corresponding index. Figure 2.10 shows which fields of table FKKVKP are defined in the index.

For example, the index with **DETECTION** “2” (a standard SAP index) has been defined on the **FIELD** Alternative Payer (**FIELD** ABWRE). The display of index “2” also shows that the index is active and defined on all database systems.

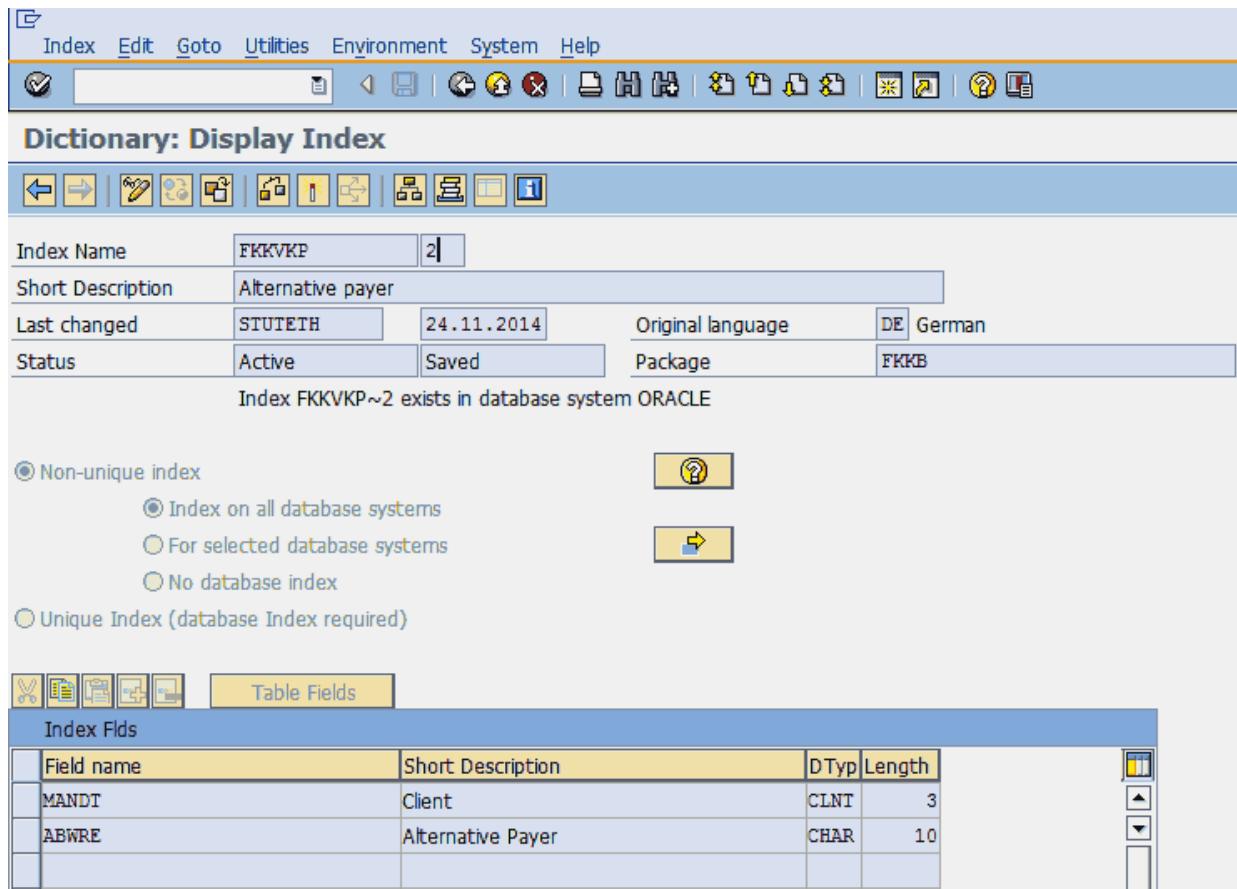


Figure 2.10: An index of table FKKVKP

A SELECT statement with the WHERE condition on the **FIELD** ABWRE finds a searched record from within a large amount of data in a short time through this index.

All indexes (also standard SAP indexes) can be included or excluded for determined databases by clicking the icon (see Figure 2.11).

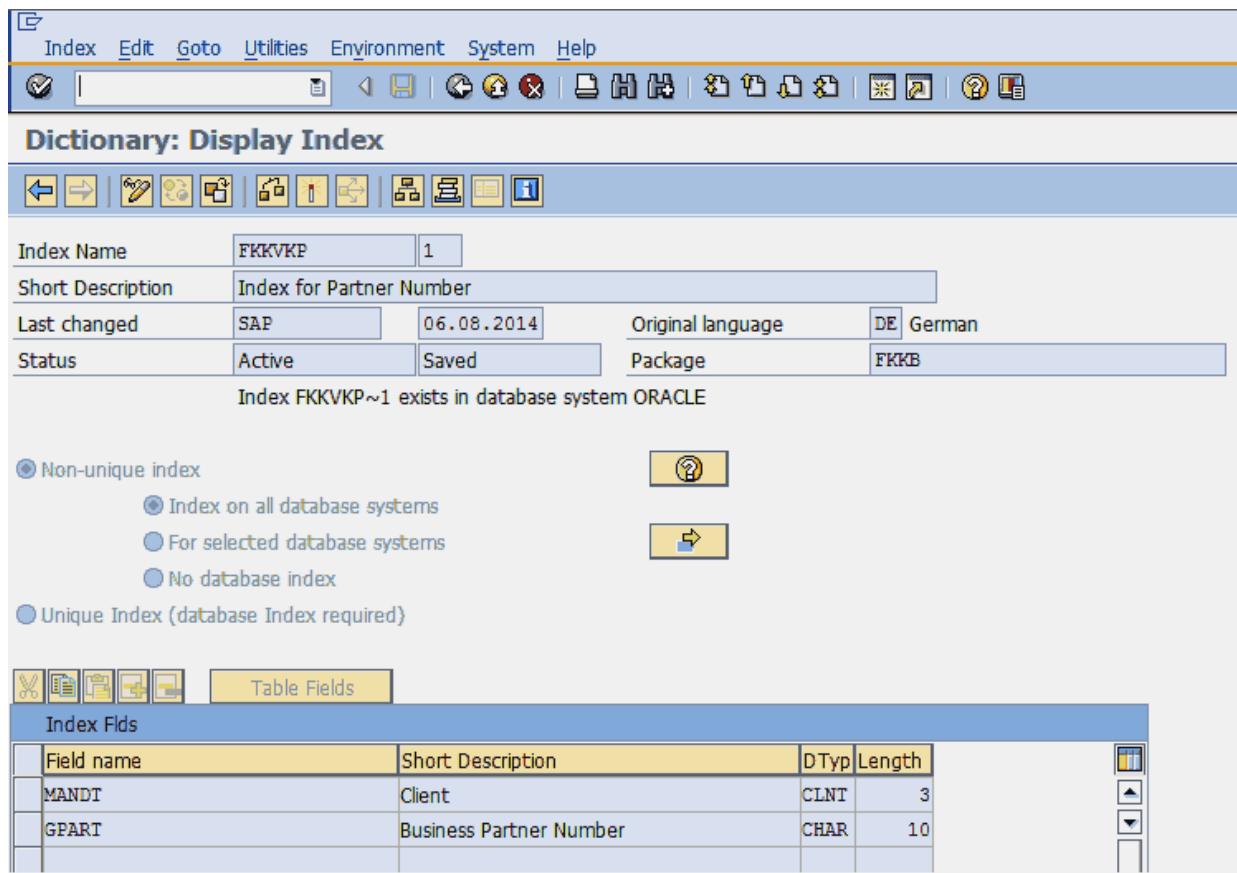


Figure 2.11: Changing standard SAP table index for partner number

The index overview column **DB-ST** appears if an index is active on all databases (see Figure 2.12).

If the field is empty, the index is active.

The DB-status “O” means that the index is not active and the entry “D” means the index is defined and active depending on the database.

There are no entries for databases in the last four columns (DBS), so the entries in the column **INCLUSION/EXCLUSION** are valid. If there is an entry with the character “I” (Inclusion), there is an active index in the database named in one of the last four columns.

If there is an entry with the character “E” (Exclusion), then there is no active index in the database named in one of the four last columns.

DB index name	DBSt	Inclusion/Exclusion	DBS	DBS	DBS	DBS
FKKVKP~1						
FKKVKP~2		E				
FKKVKP~3		E				
FKKVKP~4		E				
FKKVKP~5		E				
FKKVKP~6		E				
FKKVKP~7		E				
FKKVKP~8		E				
	O	E				

Figure 2.12: Include and exclude list of a table index

The developer has only a small influence in formulating the SELECT statement and which index is used by the database during the execution. The decision on which index is used is decided by the database system. It is probable that the database will use the index where fields are used in the WHERE condition, but not saved.

Therefore, when the index for the “Alternative Payer” is used to perform a **SELECT** statement, the **SELECT** command needs to have the following form:

```
SELECT SINGLE *
  FROM fkkvkp
  INTO ls_fkkvkp
 WHERE abwra = gv_abwra.
```

In standard SAP programs, there is no index for the field **ACCOUNT CLASS**. You can create an index by clicking the  icon in the display of table indexes. The developer must weigh whether such an index is required. (Note that, when using an index, the selection of contract accounts with a determined account class will be large, and every new index increases the database administration expense and a new index can be counterproductive to the performance.) SAP recommends a maximum amount of five indexes for a table. This maximum is supported by the following arguments.

- ▶ Changes in fields with an index have also been performed in the index.
- ▶ The data volume increases with every new index.

- ▶ The database internal optimizer receives too many choices and the probability to choose the wrong index increases.

## 2.4 Foreign keys

To avoid incorrect user input, you can define *foreign keys* for determined fields. These foreign keys (also called *check tables*) describe the relation between two tables. The function of foreign keys is shown with the help of the table for contract accounts in the database table FKKVKP.

A foreign key is defined for the field **BUSINESS PARTNER NUMBER** (GPART), which describes the relationship between the table for the business partner (BUT000) and the table for contract accounts (FKKVKP). See Figure 2.13.

Only a record for a contract account can be inserted into table FKKVKP if the field **BUSINESS PARTNER NUMBER** contains a number that exists in **TABLE BUT000** (Business Partner).

DCU(1)/504 Display Foreign Key FKKVKP-GPART

Short text					
Check table	BUT000				
Foreign Key Fields					
Check table	ChkTablFld	For.key table	Foreign Key Field	Generic	Constant
BUT000	CLIENT	FKKVKP	MANDT	<input type="checkbox"/>	
BUT000	PARTNER	FKKVKP	GPART	<input type="checkbox"/>	

Screen check

<input type="checkbox"/> Check required	Error message	MsgNo	AArea
---	---------------	-------	-------

Semantic attributes

Foreign key field type	<input checked="" type="radio"/> Not Specified <input type="radio"/> Non-key-fields/candidates <input type="radio"/> Key fields/candidates <input type="radio"/> Key fields of a text table
Cardinality	1 : CN

Figure 2.13: Foreign key of a database table

The **FOREIGN KEY FIELD TYPE** often will not be specified. In this case, choose the **CARDINALITY** for the foreign key “1:CN.” This means there can be any number of contract accounts for a business partner.

## 2.5 Currency and quantity fields

Fields that contain currency values have the ABAP-specific data type CURR (for currency). This currency amount must have a relationship to a field with the ABAP-specific data type CUKY. The data type CUKY contains the currency key (e.g., EUR for EURO currency or USD for U.S. dollar).

This similarly applies to quantities that have the ABAP data type QUAN (quantity). Quantities must have a relationship to a field with the data type UNIT.

Normally, another field for the currency key is added to a table with currency amounts. The same applies to tables with a quantity; they have another field with the data type UNIT. In the last tabulator of the table display in **TRANSACTION SE11** (see Figure 2.14), the field for currency amount with **DATA TYPE CURR** must be assigned a field with **DATA TYPE CUKY**.

The screenshot shows the SAP Dictionary: Display Table interface. The top menu bar includes Table, Edit, Goto, Utilities, Extras, Environment, System, and Help. Below the menu is a toolbar with various icons. The title bar reads "Dictionary: Display Table". The sub-title bar shows "Transp. Table FKKVKP Active" and "Short Description Contract Account Partner-Specific". The tabs at the bottom are Attributes, Delivery and Maintenance, Fields, Entry help/check, and Currency/Quantity Fields. The current view is the Fields tab. A search bar "Search Help" is located above the table. The table itself has columns: Field, Data element, Data Ty., Reference table, Ref. field, and Short Description. There are four rows of data:

Field	Data element	Data Ty.	Reference table	Ref. field	Short Description
GUID	BUAG_GUID	RAW			Business Agreement GUID
DDLAM	DDLAM_KK	CURR	FFKKVKP	DDLCU	Direct Debit Limit
DDLCU	DDLCU_KK	CUKY			Currency of Direct Debit Limit
DDLNM	DDLNM_KK	NUMC			Number of Months for Calculation of Direct Debit Limit

Figure 2.14: Currency field and currency key

The data type CUKY is used by hitting the “F4” help key to check table TCURC for the currency key and table TCURT for the currency name.

The quantity unit with data type UNIT is used by hitting the “F4” help key to check table T006 for standardized units.

## 2.6 Append structures

*Append structures* provide more fields for database tables. Many standard SAP append structures exist for standard SAP tables. For example, for contract table EVER the append structure is EVERA. This database structure contains general fields for contracts in SAP IS-U.

In a table display, the fields of append structures are in blue and original table fields are in black (see Figure 2.15).

The screenshot shows the SAP Dictionary: Display Table interface. The top menu bar includes Table, Edit, Goto, Utilities, Extras, Environment, System, and Help. Below the menu is a toolbar with various icons. The main title is "Dictionary: Display Table". A sub-header indicates "Transp. Table EVER Active" and "Short Description IS-U Cont.". Below this, there are tabs: Attributes, Delivery and Maintenance, Fields (which is selected), Entry help/check, and Currency/Quantity Fields. The Fields table has columns: Field, Key, Init..., Data element, Data Type, Length, Decim..., and Short Description. The table rows are:

Field	Key	Init...	Data element	Data Type	Length	Decim...	Short Description
MANDT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	MANDT	CLNT	3	0	Client
VERTRAG	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	VERTRAG	CHAR	10	0	Contract
.INCLUDE	<input type="checkbox"/>	<input checked="" type="checkbox"/>	EVERA	STRU	0	0	IS-U Contract: General Fields
BUKRS	<input type="checkbox"/>	<input type="checkbox"/>	BUKRS	CHAR	4	0	Company Code
SPARTE	<input type="checkbox"/>	<input type="checkbox"/>	SPARTE	CHAR	2	0	Division
EIGENVERBR	<input type="checkbox"/>	<input type="checkbox"/>	EIGENVERBR	CHAR	1	0	Plant or company consumption

Figure 2.15: Append structure of a database table

For ABAP development in particular the *CI-includes* (customizing-includes) are of interest. These structures are used for additional customer-specific table fields without using an SAP modification (see the chapter “Modification and enhancements to SAP standards” in the *Practical Guide to SAP ABAP: Part 2*).

In our example, we need additional fields for the contract. We add these fields to the customizing-include CI\_EVER (see Figure 2.16):

- ▶ GENERAL\_CONTRACT\_NUMBER is the general contract account number.

- ▶ GENERAL\_CONTRACT\_TITLE is the general contract title.
- ▶ GENERAL\_CONTRACT\_KEEPER is the name of the general contract keeper.
- ▶ ZZ\_CONTACT\_CONTRACT is the customer contact for contracts .
- ▶ ZZ\_COMM\_CONTRACT is the customer communication data for contracts.
- ▶ ZZ\_CONTACT\_BILLING is the customer contact for billing.
- ▶ ZZ\_COMM\_BILLING is the customer communication data for billing.
- ▶ ZZ\_CONTACT\_METER\_READ is the customer contact for meter reading.
- ▶ ZZ\_COMM\_METER\_READ is the customer communication data for meter reading.

The screenshot shows the SAP Dictionary: Display Structure interface. The structure is named CI\_EVER, which is active. The short description is "Customer Include zum Vertrag". The table displays various components and their types, including several custom fields starting with ZZ\_.

Component	Typing Method	Component Type	Data Type	Length	Decim...	Short Description
GENERAL_CONTRACT_NUMBER	Types	ZZ_GENERAL_CONTRACT_NUMBER	CHAR	10	0	First Customer - General Contract Number
GENERAL_CONTRACT_TITLE	Types	ZZ_GENERAL_CONTRACT_TITLE	CHAR	20	0	First Customer - General Contract Title
GENERAL_CONTRACT_KEEPER	Types	ZZ_GENERAL_CONTRACT_KEEPER	CHAR	12	0	First Customer - General Contract Keeper
ZZ_CONTACT_CONTRACT	Types	ZZ_CONTACT_CONTRACT	CHAR	40	0	First Customer - Contact for contract
ZZ_COMM_CONTRACT	Types	ZZ_COMM_CONTRACT	CHAR	40	0	First Customer - Communication for Contract
ZZ_CONTACT_BILLING	Types	ZZ_CONTACT_BILLING	CHAR	40	0	First Customer - Contact for Billing
ZZ_COMM_BILLING	Types	ZZ_COMM_BILLING	CHAR	40	0	First Customer - Communication for Billing
ZZ_CONTACT_METER_READ	Types	ZZ_CONTACT_METER_READ	CHAR	40	0	First Customer - Contact für Meter Reading
ZZ_COMM_METER_READ	Types	ZZ_COMM_METER_READ	CHAR	40	0	First Customer - Communication for Meter Reading

Figure 2.16: Customer-specific fields in customizing-include of a standard SAP table

In ABAP programs, you can treat the customer-include fields the same way as table fields.

To activate additional customized fields, include CI\_EVER so the table EVER is generated. Moreover, depending on the structure of the table EVER (table EVER can be included in more than 400 structures), the activation of

additional fields is treated by the transport management system, when the transport is imported to production systems. Depending on the amount of records in the table EVER as well as the amount of dependent structures, this procedure can take a while.

### Plan transports of CI-includes



Changing CI-includes in production systems can cause program crashes (dumps) due to table modifications and dependent structures. Hence, the import of such a transport is only to be carried out to a production system in a non-production time.

## 2.7 Database table ZGENERALCONTRACT

We can now create the customer-specific table ZGENERALCONTRACT.

The final remarks in [Section 2.1](#) result in the customer-specific **TABLE** ZGENERALCONTRACT with the properties shown in Figure 2.17.

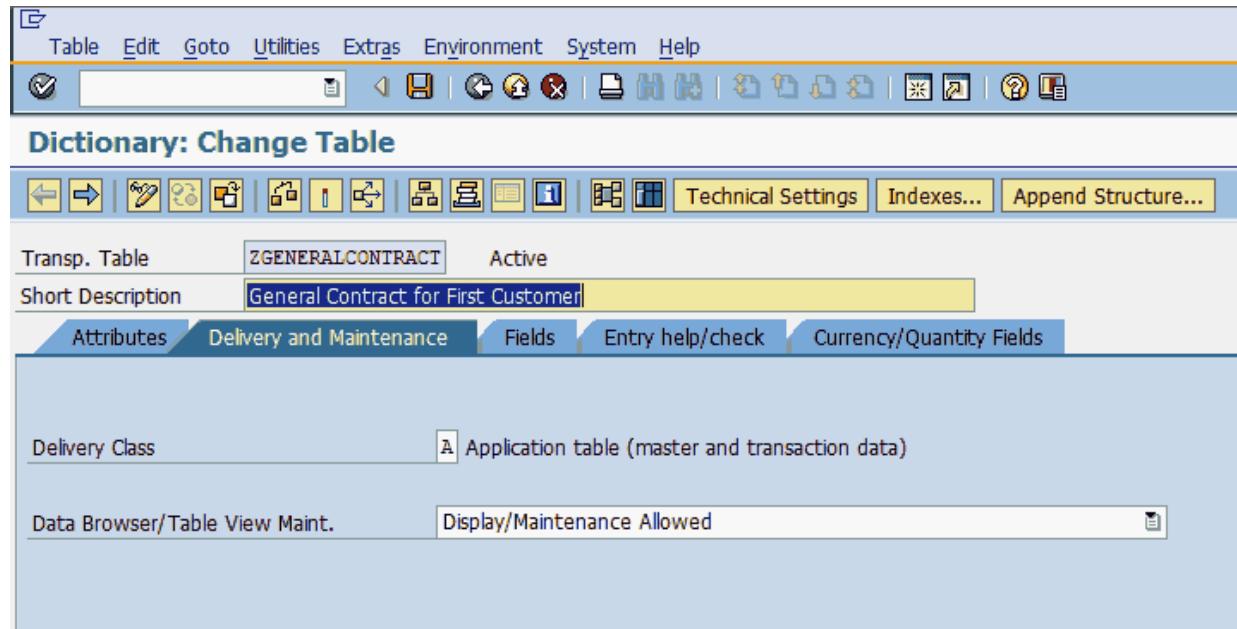


Figure 2.17: Delivery class and table view maintenance of a Z-table

To maintain data in the production system, you have to choose the delivery class “Application Table” and the table view “Display/Maintenance Allowed.”

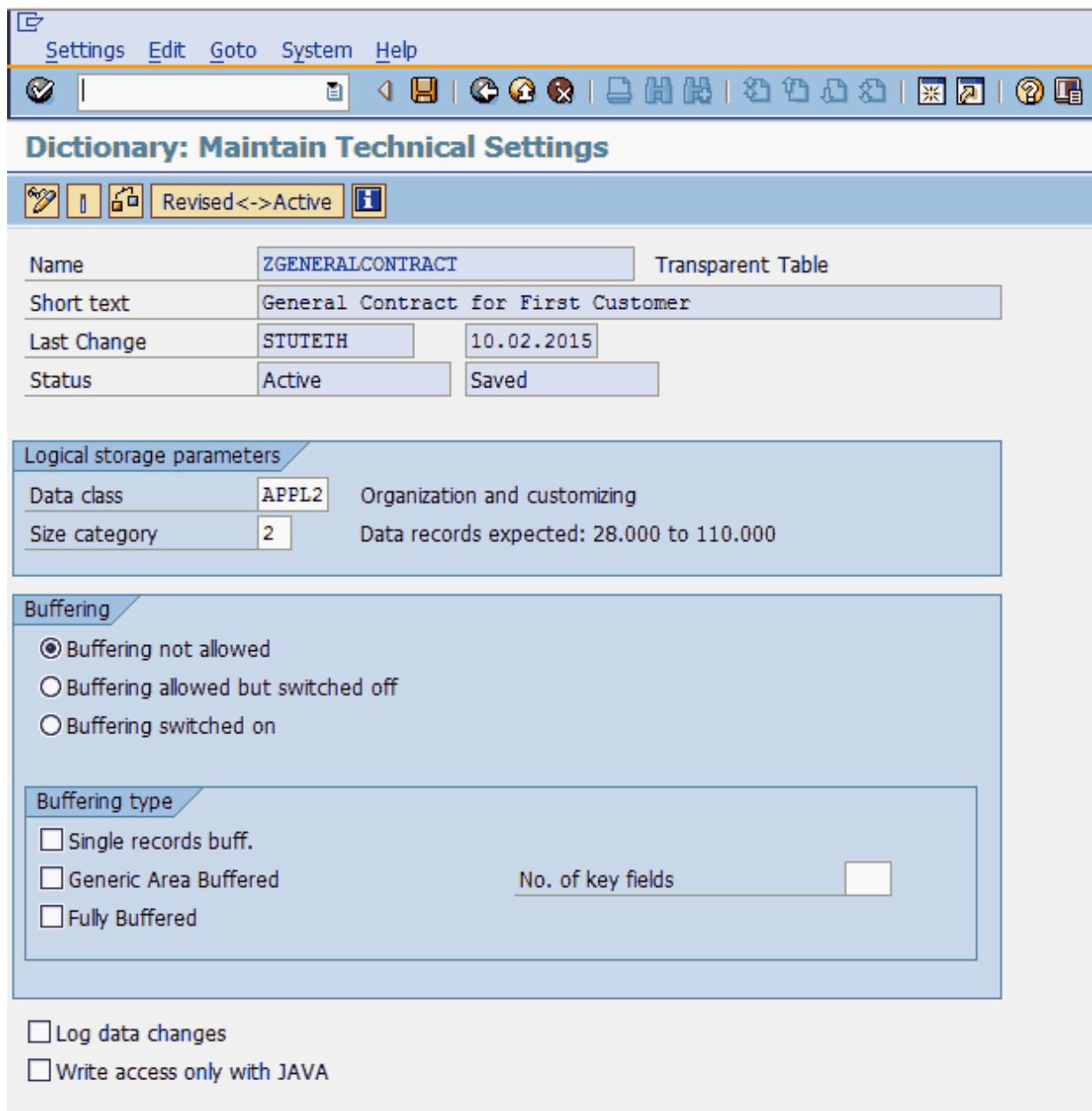


Figure 2.18: Technical settings of a Z-table

Figure 2.18 shows an amount between 28,000 and 110,000 general contracts are expected.

Figure 2.19 shows the fields of the table.

The screenshot shows the SAP Dictionary: Display Table interface. The top menu bar includes Table, Edit, Goto, Utilities, Extras, Environment, System, and Help. The toolbar contains various icons for table management. The title bar says "Dictionary: Display Table". Below the title bar, there are tabs for "Attributes", "Delivery and Maintenance", "Fields" (which is selected), "Entry help/check", and "Currency/Quantity Fields". A sub-toolbar below the tabs includes icons for table operations and buttons for "Technical Settings", "Indexes...", and "Append Structure...". The main area displays a table of fields for the table ZGENERALCONTRACT. The table has columns for Field, Key, Init..., Data element, Data ..., Length, De..., and Short Description. The data rows are as follows:

Field	Key	Init...	Data element	Data ...	Length	De...	Short Description
MANDT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	MANDT	CLNT	3	0	Client
GENERAL_CONTRACT_NUMBER	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	ZZ_GENERAL_CONTRACT_NUMBER	CHAR	10	0	First Customer - General Contract Number
GENERAL_CONTRACT_TITLE	<input type="checkbox"/>	<input checked="" type="checkbox"/>	ZZ_GENERAL_CONTRACT_TITLE	CHAR	20	0	First Customer - General Contract Title
GENERAL_CONTRACT_KEEPER	<input type="checkbox"/>	<input checked="" type="checkbox"/>	ZZ_GENERAL_CONTRACT_KEEPER	CHAR	12	0	First Customer - General Contract Keeper
ERDAT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	ERDAT	DATS	8	0	Date on Which Record Was Created
ERNAM	<input type="checkbox"/>	<input checked="" type="checkbox"/>	ERNAM	CHAR	12	0	Name of Person who Created the Object
AEDAT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	AEDAT	DATS	8	0	Changed On
AENAM	<input type="checkbox"/>	<input checked="" type="checkbox"/>	AENAM	CHAR	12	0	Name of Person Who Changed Object

Figure 2.19: Fields of table ZGENERALCONTRACT

A general contract keeper can only be a user who has an account for the relevant SAP system. Thus, we define “USR01” as a check table for the FIELD “General Contract Keeper” (see Figure 2.20).

DCU(3)/504 Create Foreign Key ZGENERALCONTRACT-GENERAL\_CONTRACT\_KEEPER

Short text					
Check table	USR01		<input type="button" value=""/>	Generate proposal	
Foreign Key Fields					
Check table	ChkTabId	For.key table	Foreign Key Field	Generic	Constant
USR01	MANDT	ZGENERALC...	MANDT	<input type="checkbox"/>	
USR01	BNAME	ZGENERALC...	GENERAL_CONTRACT_KEEPER	<input type="checkbox"/>	
<input type="button"/> <input type="button"/> <input type="button"/> <input type="button"/> <input type="button"/> <input type="button"/>					

Screen check

<input checked="" type="checkbox"/> Check required	Error message	MsgNo	AArea
--	---------------	-------	-------

Semantic attributes

Foreign key field type	<input checked="" type="radio"/> Not Specified <input type="radio"/> Non-key-fields/candidates <input type="radio"/> Key fields/candidates <input type="radio"/> Key fields of a text table
Cardinality	<input type="text"/> : <input type="text"/> <input type="button" value="1"/>

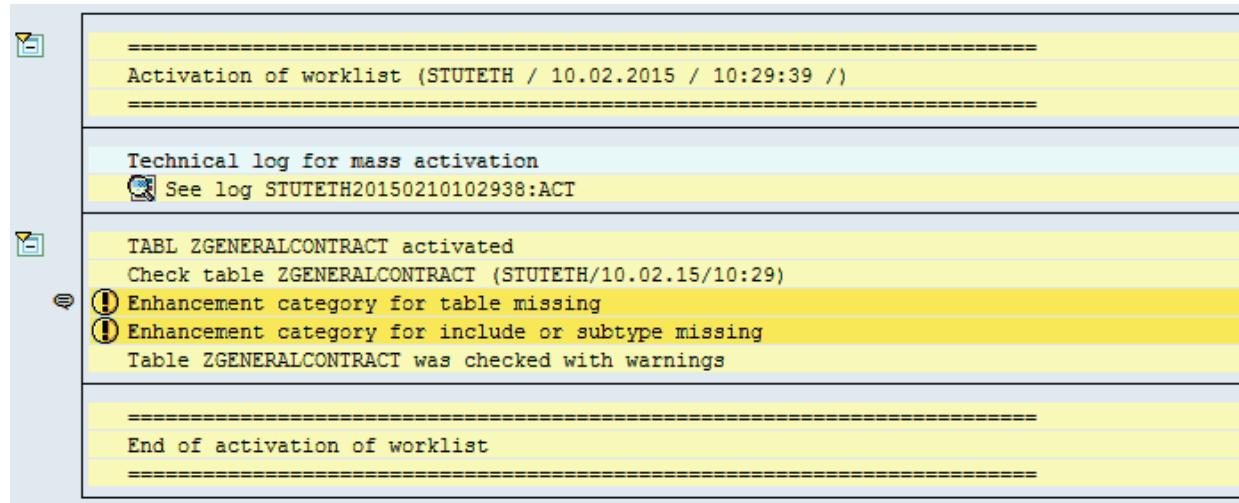
Copy

Figure 2.20: Check table for the field “General Contract Keeper” of table ZGENERALCONTRACT

When inputting general contracts, an SAP user’s name must be used for the general contract keeper field.

## 2.8 Enhancement categories of tables and structures

A warning message appears when a new database table is activated if the *enhancement category* of the table has not been specified. See Figure 2.21.



The screenshot shows a system log window with the following text:

```
=====
Activation of worklist (STUTETH / 10.02.2015 / 10:29:39 /)
=====

Technical log for mass activation
See log STUTETH20150210102938:ACT

TABL ZGENERALCONTRACT activated
Check table ZGENERALCONTRACT (STUTETH/10.02.15/10:29)
  ⓘ Enhancement category for table missing
  ⓘ Enhancement category for include or subtype missing
    Table ZGENERALCONTRACT was checked with warnings

=====
End of activation of worklist
=====
```

Figure 2.21: Warning that enhancement category has not been specified

You can extend tables and structures by using the customer-includes or other append structures. Such enhancements refer to the current table and all dependent tables or structures that use the current table as an include structure or a referenced structure. When enhancing a table or structure, runtime errors can occur with type checks for deep structures or in programs that use field values with offsets and field lengths.

Through the pull-down menu **EXTRAS • ENHANCEMENT CATEGORY** in **TRANSACTION SE11**, you have different choices for setting the **ENHANCEMENT CATEGORY** (see Figure 2.22).

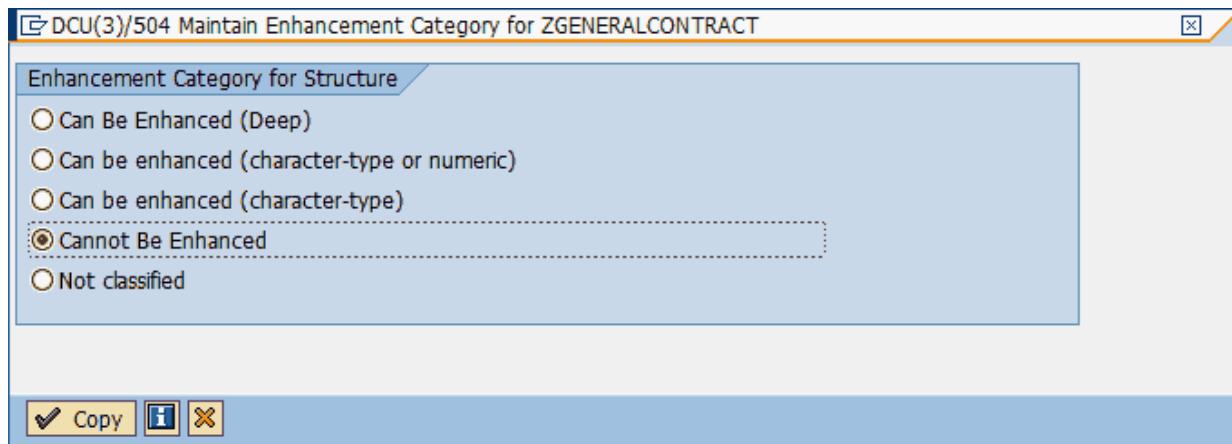


Figure 2.22: Enhancement categories for tables or structures

These enhancement categories have the following meanings:

- ▶ “Cannot be enhanced” = the table/structure cannot be enhanced in any way.
- ▶ “Can be enhanced (character-type)” = all components of enhancements have to be represented by a character (C, N, D, or T). Source table/structure and related enhancements with CI-includes or append structures are defeated by this restriction.
- ▶ “Can be enhanced (character-type or numeric)” = the table/structure and related enhancements should not use deep data types, such as tables, references, strings.
- ▶ “Can be enhanced (Deep)” = the table/structure and related enhancements can be any data type.
- ▶ “Not classified” (default) = this category can be chosen for a transitional state, but not for creating structures.

It is important to have the categorization relate to structures that use the current table or structure as included. In a row of included tables or structures, the following enhancement category is more restrictive than the enhancement category of the previous table/structure:

Table/structure A -> Include table/structure B > Include table/structure C

The following hierarchy is valid for the enhancement categories:

Cannot be enhanced < Can be enhanced (character-type) < Can be enhanced (character-type or numeric) < Can be enhanced (Deep)

The enhancement category “Not classified” is the default setting for a new customer-specific table. SAP recommends choosing “Can be enhanced (Deep)” for customer-specific tables/structures, so that included tables/structures can have any enhancement category.

In our example, no further structures or tables are included in the customer-specific table ZGENERALCONTRACT. In addition, this table will not be included with other tables. Therefore, it is practical to use the enhancement category “Cannot be enhanced.”

A new activation of the table ZGENERALCONTRACT won’t show any warnings.

## 2.9 Views instead of select with join option

ABAP programs often need data from several database tables. The tables stand with each other when related, such as when the requirement is that all contract accounts and contracts are selected for a given business partner.

These requests are realized with SELECT statements with JOIN options (see Figure 2.23).

```
SELECT a~partner a~type
      b~vkont b~vkbez b~loevm
      c~vertrag c~bukrs c~sparte
      c~einzdat c~auszdat c~billfinit c~loevm
FROM  ( ( but000 AS a
      JOIN fkkvkp AS b ON a~partner = b~gpart )
      JOIN ever   AS c ON b~vkont   = c~vkonto )
INTO TABLE gt_input
WHERE a~partner IN so_gpart
AND   b~vkont   IN so_vkont
AND   c~vertrag IN so_vertr
AND   b~loevm <> 'X'
AND   c~loevm <> 'X'.
```

Figure 2.23: SELECT with JOIN option over the three tables BUT000, FKKVKP, and EVER

These SELECT statements are difficult to develop and their correctness can only be checked through a program runtime. The danger of such JOIN options consists in the fact that with inefficient programming too many unneeded records are selected from the database. The result is that the runtime of programs is extended needlessly. In addition, the SELECT statements with JOIN option cannot be reused in other programs.

For efficiency, use database views instead of the SELECT statements. You can create database views with the help of **TRANSACTION SE11**. When creating database views, you can define the relevant database tables, their relations, selection conditions, and the view fields.

The SELECT statement shown in Figure 2.23 can be replaced by the **DATABASE VIEW Z\_GP\_VKONT\_VERTR** (see Figure 2.24).

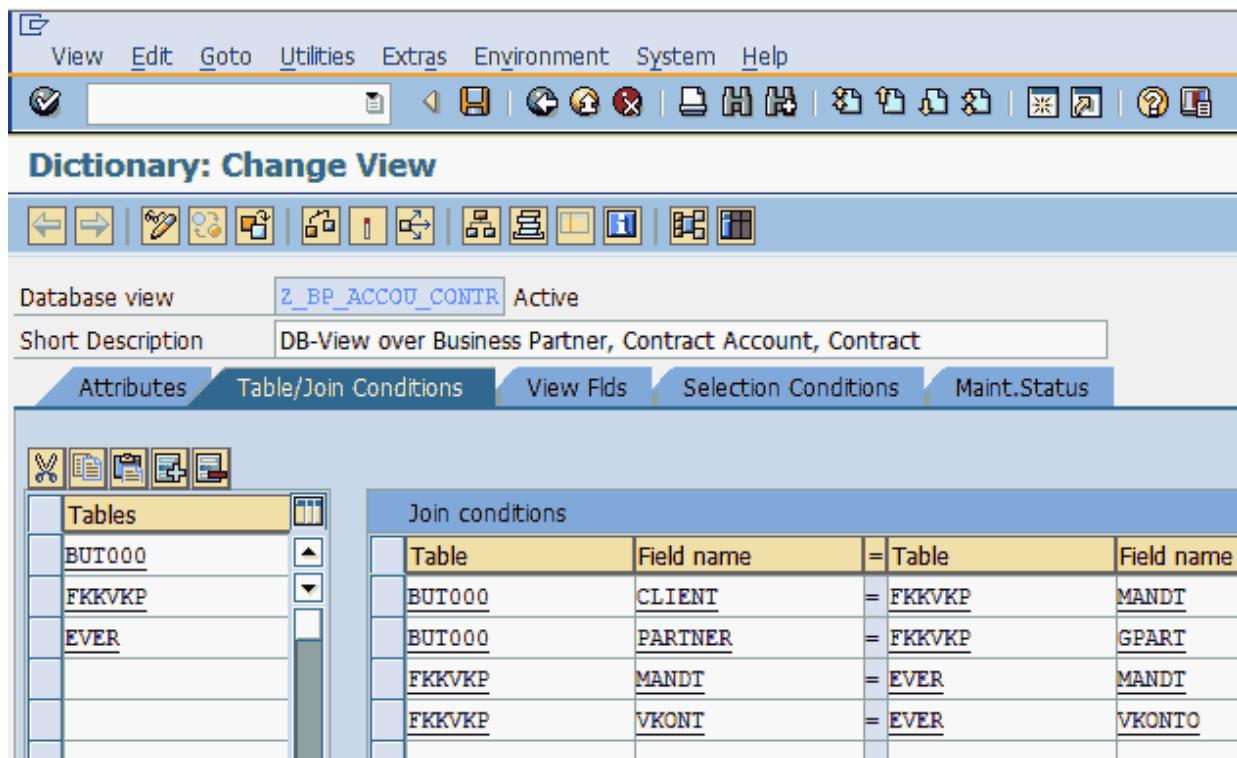


Figure 2.24: Tables and JOIN conditions of a database view

The screenshot shows the SAP Dictionary: Change View interface, similar to Figure 2.24, but focusing on "View fields".

**View fields:**

View field	Table	Field	Key	Data elem.	Mod	DTyp	Length	Short description
CLIENT	BUT000	CLIENT	<input checked="" type="checkbox"/>	MANDT	<input type="checkbox"/>	CLNT	3	Client
PARTNER	BUT000	PARTNER	<input checked="" type="checkbox"/>	BU_PARTNER	<input type="checkbox"/>	CHAR	10	Business Partner Number
TYPE	BUT000	TYPE	<input checked="" type="checkbox"/>	BU_TYPE	<input type="checkbox"/>	CHAR	1	Business partner category
VKONT	FKKVP	VKONT	<input checked="" type="checkbox"/>	VKONT_KK	<input type="checkbox"/>	CHAR	12	Contract Account Number
VKBEZ	FKKVP	VKBEZ	<input checked="" type="checkbox"/>	VKBEZ_KK	<input type="checkbox"/>	CHAR	35	Contract account name
LOEVM_FKKVP	FKKVP	LOEVM	<input checked="" type="checkbox"/>	LOEVM_KK	<input type="checkbox"/>	CHAR	1	Mark Contract Account for Deletion
KTOKL	FKKVP	KTOKL	<input checked="" type="checkbox"/>	KTOKLASSE	<input type="checkbox"/>	CHAR	4	Account class
VERTRAG	EVER	VERTRAG	<input checked="" type="checkbox"/>	VERTRAG	<input type="checkbox"/>	CHAR	10	Contract
BUKRS	EVER	BUKRS	<input type="checkbox"/>	BUKRS	<input type="checkbox"/>	CHAR	4	Company Code
SPARTE	EVER	SPARTE	<input type="checkbox"/>	SPARTE	<input type="checkbox"/>	CHAR	2	Division
EINZDAT	EVER	EINZDAT	<input type="checkbox"/>	EINZDAT	<input type="checkbox"/>	DATS	8	Move-In Date
AUSZDAT	EVER	AUSZDAT	<input type="checkbox"/>	AUSZDAT	<input type="checkbox"/>	DATS	8	Move-Out Date
BILLINIT	EVER	BILLINIT	<input type="checkbox"/>	BILLINIT	<input type="checkbox"/>	CHAR	1	Contract terminated for billing reasons
LOEVM_EVER	EVER	LOEVM	<input type="checkbox"/>	LOEVM	<input type="checkbox"/>	CHAR	1	Deletion Indicator

Figure 2.25: View fields of a database view

Figure 2.25 shows the selected fields of the involved tables.

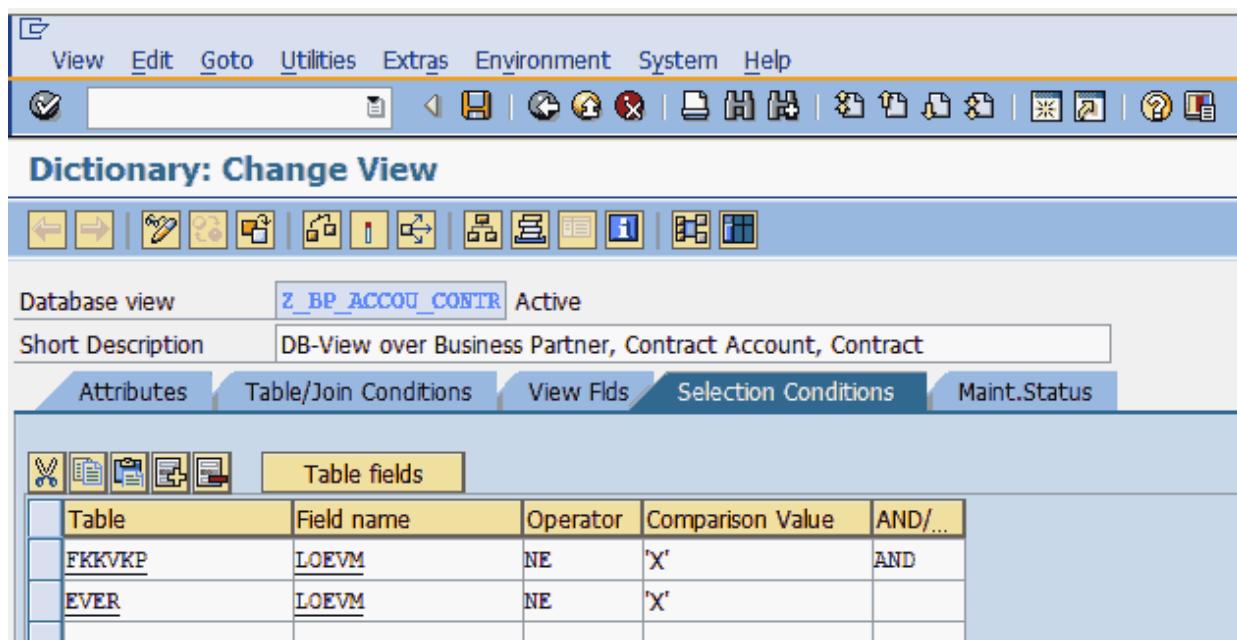


Figure 2.26: Selection conditions of a database view

The selection condition of the view allows restrictions for data selection (see Figure 2.26). In this example, contract accounts and contracts with deletion mark ('X') are selected.

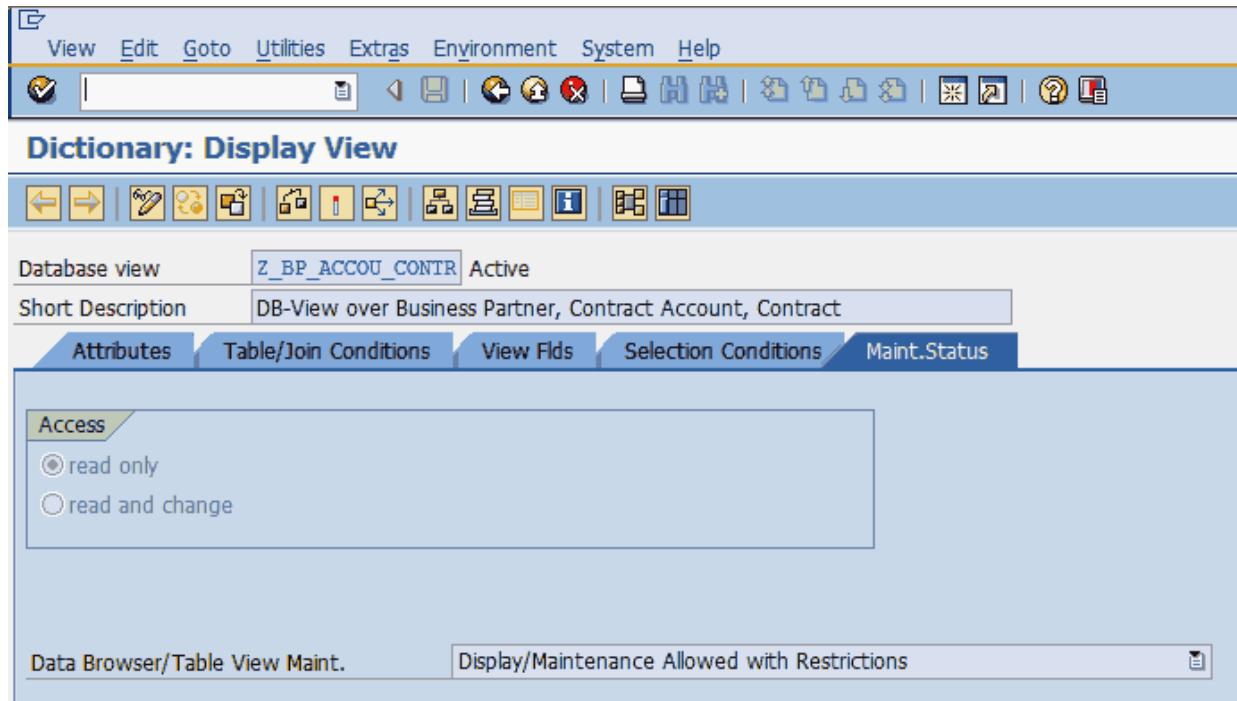


Figure 2.27: Properties of a database view

The properties of a view are determined during creation (see Figure 2.27). Directly after input of a new view name, a popup screen prompting the user to choose the view type appears (see Figure 2.28).

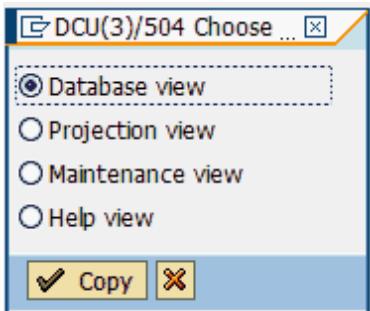


Figure 2.28: Popup screen to choose the view type

The **Database view** allows the selection of data, but not the changing or insertion of new records. If you want to change datasets or insert new records, you have to choose a maintenance view.

The **Projections view** is used to hide single fields of a table during the selection to reduce data amounts for function module interfaces.

The **Maintenance view** allows the collection of several tables to an economic object. The contained data sets can be created or changed in several tables at the same time. Maintenance view properties have the same properties as database tables.

The **Help view** is used for data selection of records from different tables in search help. Help view properties have the same properties as database tables.

If the display, change, delete, and insert features in a view are allowed, the view can be used to modify records (see Figure 2.29).

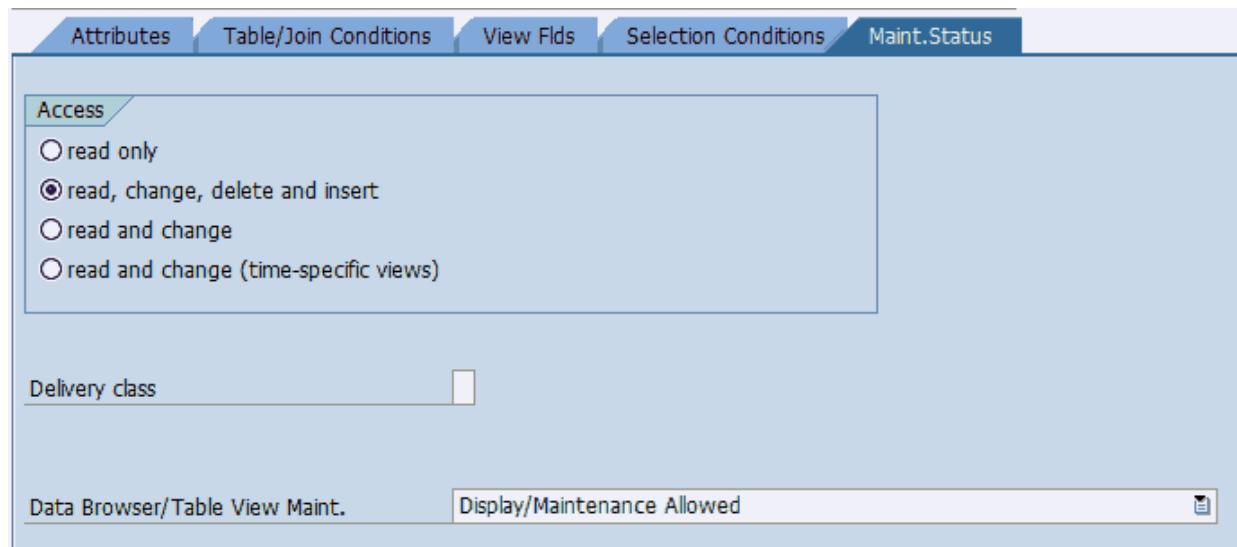


Figure 2.29: Definition of the properties of a maintenance view

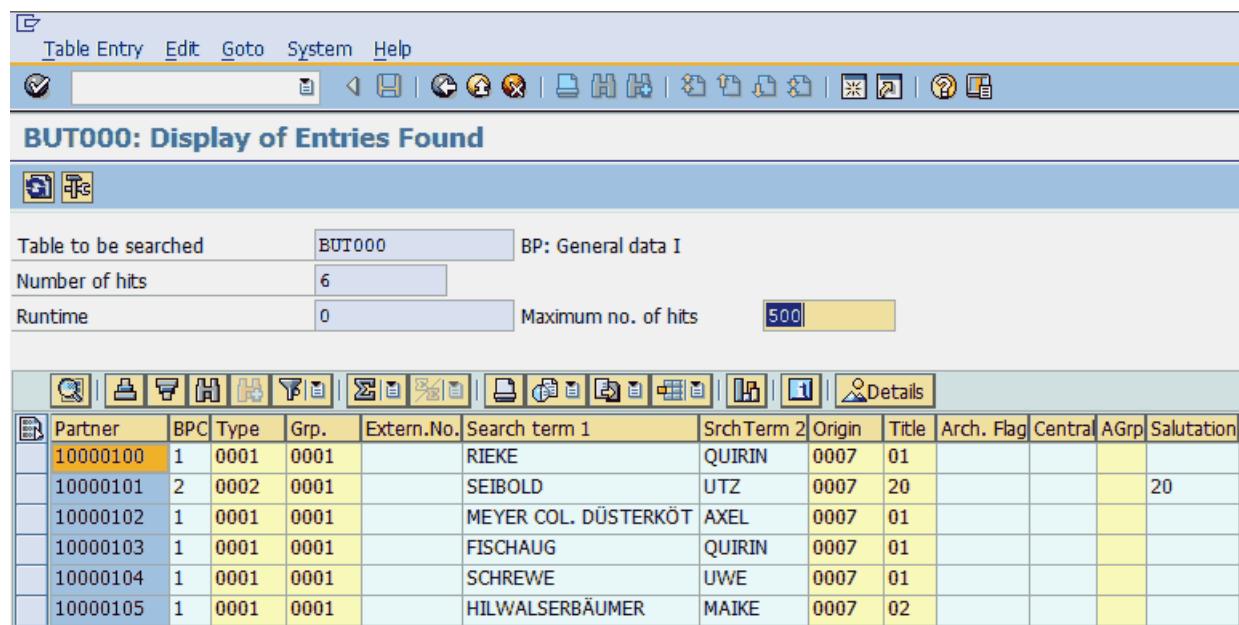
Defining views do not load the database. Views can be reused and tested directly after their activation. The selected records of a view can be checked without further program coding.

Views allow only an *inner join*. That means views only select records in the involved tables. In the examples above, the view doesn't select a business partner without a contract account or without a contract.

## 2.10 Conversion modules of domains

With the selection of data in ABAP programs, *domains* are the type declarations of data elements.

Use **TRANSACTION** SE16N to select **TABLE** BUT000, e.g., the **BUSINESS PARTNER** with the number “10000100,” to get the record shown in Figure 2.30.



The screenshot shows the SAP SE16N transaction interface. The title bar reads "BUT000: Display of Entries Found". The toolbar includes standard SAP icons for search, edit, and system functions. Below the toolbar, there are input fields for "Table to be searched" (BUT000), "Number of hits" (6), "Runtime" (0), and "Maximum no. of hits" (500). The main area displays a table of records with the following columns: Partner, BPC, Type, Grp., Extern.No., Search term 1, SrchTerm 2, Origin, Title, Arch. Flag, Central, AGrp, and Salutation. The first record in the table is highlighted with yellow background and orange text for the Partner field (10000100).

Partner	BPC	Type	Grp.	Extern.No.	Search term 1	SrchTerm 2	Origin	Title	Arch. Flag	Central	AGrp	Salutation
10000100	1	0001	0001		RIEKE	QUIRIN	0007	01				
10000101	2	0002	0001		SEIBOLD	UTZ	0007	20				20
10000102	1	0001	0001		MEYER COL. DÜSTERKÖT	AXEL	0007	01				
10000103	1	0001	0001		FISCHAUG	QUIRIN	0007	01				
10000104	1	0001	0001		SCHREWE	UWE	0007	01				
10000105	1	0001	0001		HILWALSERBÄUMER	MAIKE	0007	02				

Figure 2.30: Display of records from table BUT000 (Business Partner) with transaction SE16N

You cannot get the same business partner with a SELECT statement in an ABAP program by using the same business partner number.

The reason is that the business partner number field is required to have 10 digits; it is stored in the database with 10 digits. In other words, the eight-digit business partner number “10000100” is stored in the database with the 10-digit number “0010000100.” A double click on the record, as shown in Figure 2.30, confirms this statement (see Figure 2.31). The left side shows the business partner number in **TRANSACTION** SE16N, and the right side shows the number stored in the database.

Fld Name	Val.	Technical Field Name	Value Unconverted
Client	504	CLIENT	504
Business Partner	10000100	PARTNER	0010000100
BP category	1	TYPE	1
BP Type	0001	BPKIND	0001
Grouping	0001	BU_GROUP	0001
External BP Number		BPEXT	
Search term 1	RIEKE	BU_SORT1	RIEKE
Search term 2	QUIRIN	BU_SORT2	QUIRIN
Data Origin	0007	SOURCE	0007
Title	01	TITLE	01
Archiving Flag		XDELE	
Central Block		XBLCK	

Figure 2.31: Display and stored value of a business partner number

When selecting a record with **TRANSACTION** SE16N, the business partner number is transformed from an internal format (with leading zeros) to an external format (without leading zeros) by an SAP selection program.

The SAP selection program uses conversion routines, also known as function modules. To select a record from a database in an ABAP program, you have to use the same conversion routines. This conversion function module is in the domain of the data element of the business partner number.

The business partner number in **TABLE** BUT000 has the data element PARTNER. This data element is typed with the **DOMAIN** BU\_PARTNER (see Figure 2.32).

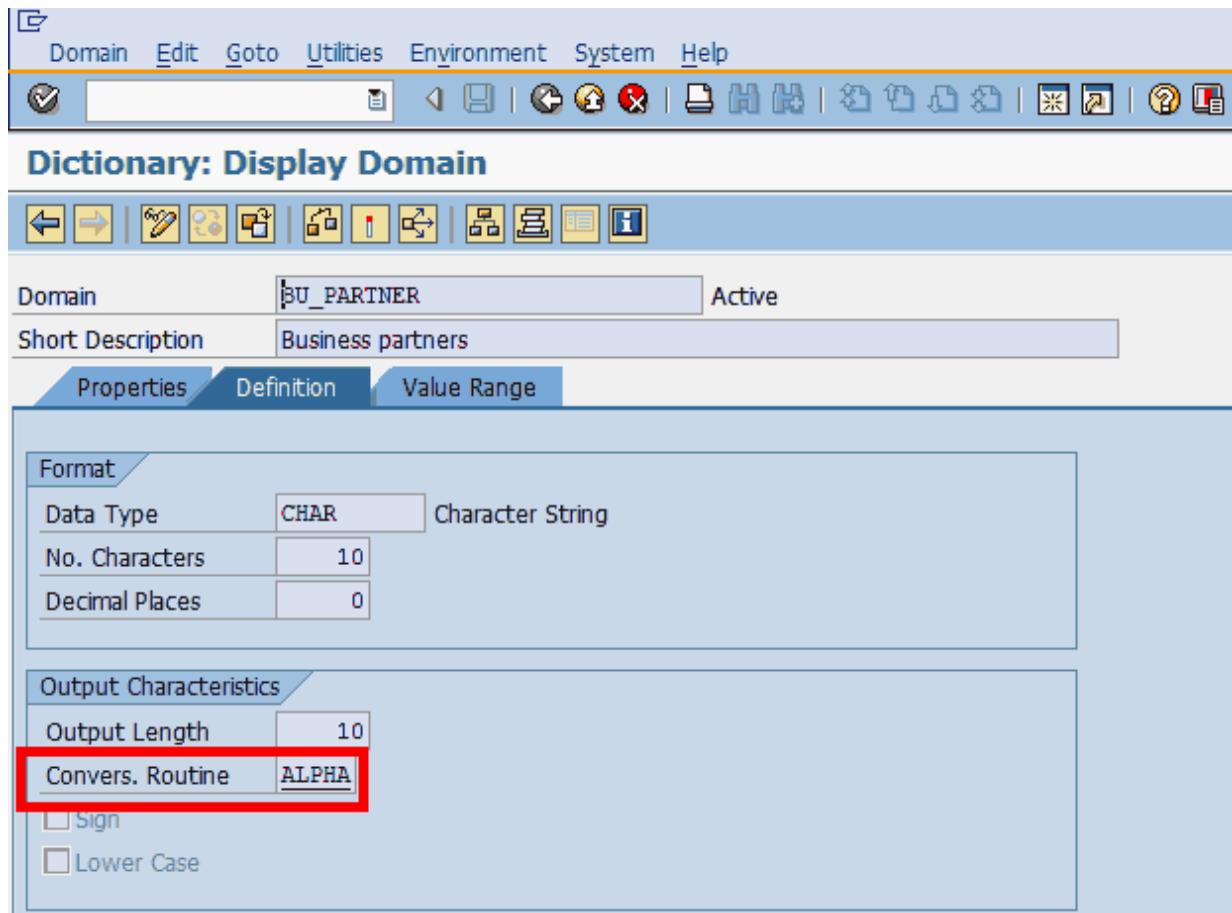


Figure 2.32: The domain BU\_PARTNER

**OUTPUT CHARACTERISTICS** is in the **CONVERS.-ROUTINE** field with the term “ALPHA.” A double click on this term shows the used function modules for conversion (see Figure 2.33).

DCU(3)/504 Repository Info System: Function Modules Find (4 Hits)	
Function group	Function group short text
Function Module Name	Short text for function module
ALFA	ALPHA conversion
<input type="checkbox"/> CONVERSION_EXIT_ALPHA_INPUT	Conversion exit ALPHA, external->internal
<input type="checkbox"/> CONVERSION_EXIT_ALPHA_OUTPUT	Conversion exit ALPHA, internal->external
SDHI	
<input type="checkbox"/> CONVERSION_EXIT_ALPHA_RANGE_I	
<input type="checkbox"/> CONVERSION_EXIT_ALPHA_RANGE_O	

Figure 2.33: Conversion routines for business partner number table maintenance generator

## 2.11 Table maintenance generator

The table maintenance generator allows the user to display, change, delete, or insert data sets.

### 2.11.1 Generate table maintenance views

Create table maintenance views with **TRANSACTION SE11**. Before you can create a table maintenance view, you have to create and activate a function group, e.g., ZGENERALCONT. This function group is used for development objects in the creation process of the table maintenance view.

In our example, we use the table ZGENERALCONTRACT for the management of general contracts. It is a client-specific application table where data maintenance is allowed. Use the pull-down menu of **TRANSACTION SE11 UTILITIES • TABLE MAINTENANCE GENERATOR** to display the menu for creating table maintenance views. In our example, we use the **AUTHORIZATION GROUP &NC&** (without authorization group) and the **FUNCTION GROUP ZCU\_GENERALCONT** created earlier (see Figure 2.34).

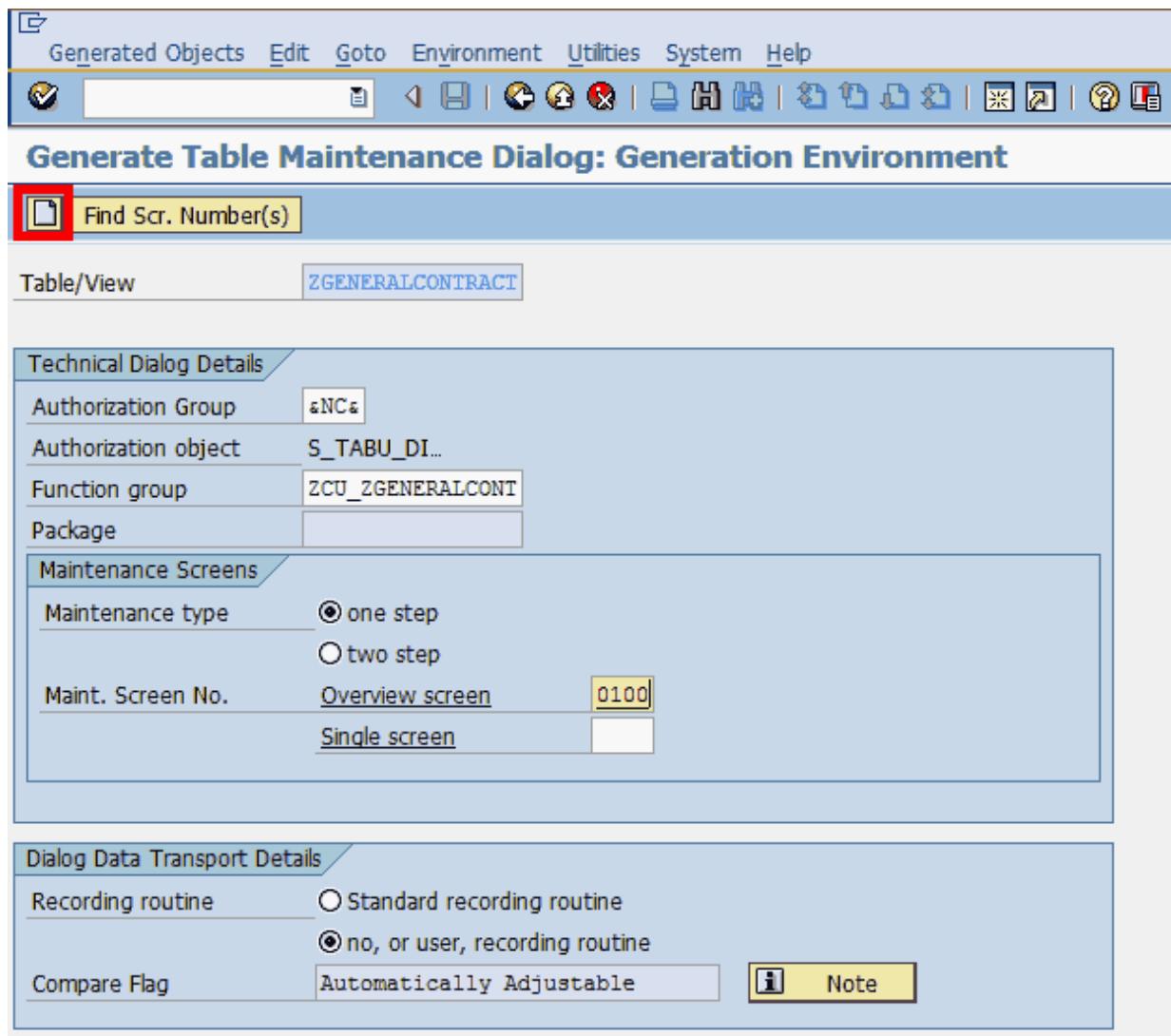


Figure 2.34: Create a table maintenance view

Use the maintenance screen number “0100” to create the table maintenance view by clicking the  icon.

After creating the view, the message “Order successfully finished” appears in the status strip of the SAP menu. The table maintenance view with the screen (screen 0100) has been created successfully.

Table content now can be maintained by using **TRANSACTION SM30** (see Figure 2.35).

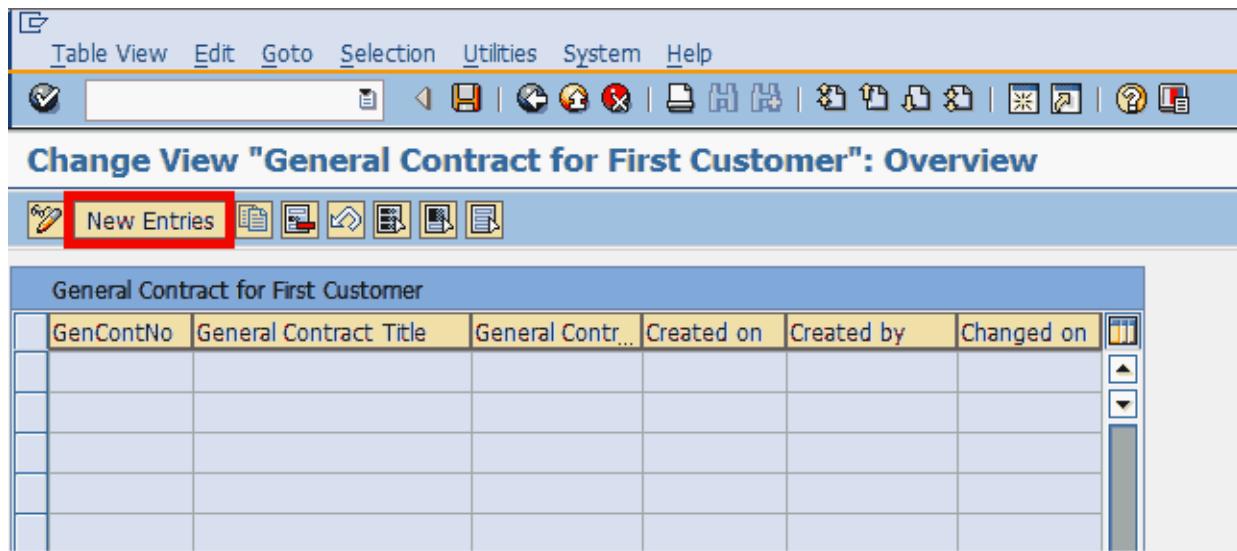


Figure 2.35: Table maintenance view in transaction SM30

With a mouse click on the **New Entries** button, you can insert new records to the table ZGENERALCONTRACT. Existing records can be modified or deleted.

Because this table is a customizing table, a request for transport is created when changing any table content.

### 2.11.2 Enhancement of table maintenance transactions

In addition, each record documents who has created the record and when, or who has changed it and when. The creation date and creator, as well as the change date and change by, shall be automatically added to each record when creating or changing data. User modification of this data must be prevented.

Implementing this request requires an extension of the source code in the maintenance view. Changes are attached to the maintenance view. To develop this enhancement, you have to modify the **MAINTENANCE SCREEN** (see Figure 2.36) and create a function module for the event 01 before saving data in the database.

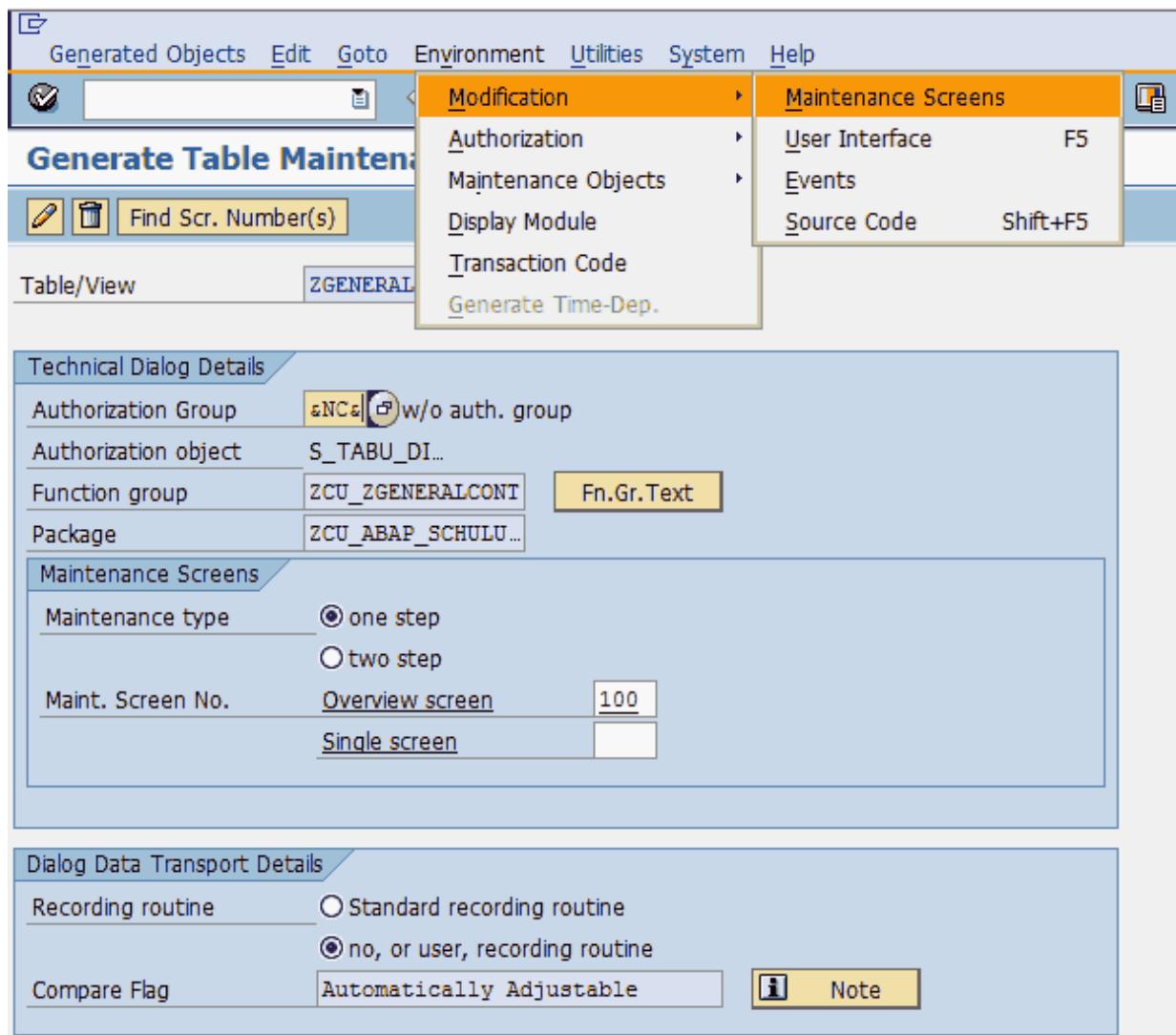


Figure 2.36: Modification of a table maintenance view

The menu path **ENVIRONMENT • MODIFICATION • MAINTENANCE SCREENS** takes you to the first step: the modification of the maintenance view. In the maintenance view, the fields **CREATED BY**, **CREATED ON**, **CHANGED BY**, and **CHANGED ON** may be indicated but not ready for input.

When modifying the maintenance screen, the generated maintenance screen “100” is displayed. For changing the input and output capability of the screen, switch on the tab element list (see Figure 2.37); by clicking the icon, you enter change mode.

The screenshot shows the SAP Screen Painter interface for changing screen SAPLZCU\_ZGENERALCONT. The main window displays a table of screen elements with various attributes. A red box highlights the 'Input' column for specific elements in the 'ELEMENTS' row.

H	M	Name	Type	Line	Co	De	Vis	He	Sc	Format	Input	Output	Out	Dict.fi	Dict	Property list
+	TCTRL_ZGENERALCONTRACT	Table	1	1	83	83	59									
-	VIM_FRAME_FIELD	I/O	1	0	60	60	1			CHAR						
-	*ZGENERALCONTRACT-GENERAL_CONTRACT_NUMBER	Text	1	1	40	10	1									1
-	*ZGENERALCONTRACT-GENERAL_CONTRACT_TITLE	Text	1	2	40	20	1									V
-	*ZGENERALCONTRACT-GENERAL_CONTRACT_KEEPER	Text	1	3	40	12	1									V
-	*ZGENERALCONTRACT-ERDAT	Text	1	4	40	10	1									V
-	*ZGENERALCONTRACT-ERNAM	Text	1	5	40	12	1									V
-	*ZGENERALCONTRACT-AEDAT	Text	1	6	40	10	1									V
-	*ZGENERALCONTRACT-AENAM	Text	1	7	40	12	1									V
-	VIM_MARKED	Check	1	0	1	1	1			CHAR	<input checked="" type="checkbox"/>					
-	ZGENERALCONTRACT-GENERAL_CONTRACT_NUMBER	I/O	1	1	10	10	1			CHAR	<input checked="" type="checkbox"/>	X				
-	ZGENERALCONTRACT-GENERAL_CONTRACT_TITLE	I/O	1	2	20	20	1			CHAR	<input checked="" type="checkbox"/>	X				
-	ZGENERALCONTRACT-GENERAL_CONTRACT_KEEPER	I/O	1	3	12	12	1			CHAR	<input checked="" type="checkbox"/>	X				
-	ZGENERALCONTRACT-ERDAT	I/O	1	4	10	10	1			DATS	<input checked="" type="checkbox"/>	X				
-	ZGENERALCONTRACT-ERNAM	I/O	1	5	12	12	1			CHAR	<input checked="" type="checkbox"/>	X				
-	ZGENERALCONTRACT-AEDAT	I/O	1	6	10	10	1			DATS	<input checked="" type="checkbox"/>	X				
-	ZGENERALCONTRACT-AENAM	I/O	1	7	12	12	1			CHAR	<input checked="" type="checkbox"/>	X				
	VIM_POSI_PUSH	Push	61	19	20	20	1			CHAR	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	X
	VIM_POSITION_INFO	I/O	61	40	30	30	1			CHAR	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	X
	OK_CODE	OK	0	0	20	20	1			OK						X

Figure 2.37: Screen of the table maintenance view in Screen Painter

Remove the check mark beside the ELEMENTS ZGENERALCONTRACT-ERDAT, -ERNAM, -AEDAT, and -AENAM in the column INPUT (see Figure 2.38). Activate this change by clicking on the icon (see Figure 2.38).

H...	M	Name	Type...	Line	Co...	De...	Vis...	He...	Sc...	Format	Input	Output	Out...	Dict.f...	Dict...	Property list
-	-	VIM_MARKED	Check	1	0	1	1	1	CHAR	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	X	
-	-	ZGENERALCONTRACT-GENERAL_CONTRACT_NUMBER	I/O	1	1	10	10	1	CHAR	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	X	
-	-	ZGENERALCONTRACT-GENERAL_CONTRACT_TITLE	I/O	1	2	20	20	1	CHAR	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	X	
-	-	ZGENERALCONTRACT-GENERAL_CONTRACT_KEEPER	I/O	1	3	12	12	1	CHAR	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	X	
-	-	ZGENERALCONTRACT-ERDAT	I/O	1	4	10	10	1	DATS	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	X	
-	-	ZGENERALCONTRACT-ERNAM	I/O	1	5	12	12	1	CHAR	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	X	
-	-	ZGENERALCONTRACT-AEDAT	I/O	1	6	10	10	1	DATS	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	X	
-	-	ZGENERALCONTRACT-AENAM	I/O	1	7	12	12	1	CHAR	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	X	
		VIM_POSI_PUSH	Push	61	19	20	20	1	CHAR	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		
		VIM_POSITION_INFO	I/O	61	40	30	30	1	CHAR	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>		
		OK_CODE	OK	0	0	20	20	1	OK					<input type="checkbox"/>		

Figure 2.38: Screen modifications of the table maintenance view

Table maintenance columns that have changed terms from the Data Dictionary:

- ▶ ERDAT – Created on
- ▶ ERNAM – Created by
- ▶ AEDAT – Changed on
- ▶ AENAM – Changed by

By checking the screen modifications with the help of **TRANSACTION SM30**, you see that these fields, by insertion or changing of records, aren't white. Therefore, you can't input data in these fields (see Figure 2.39).

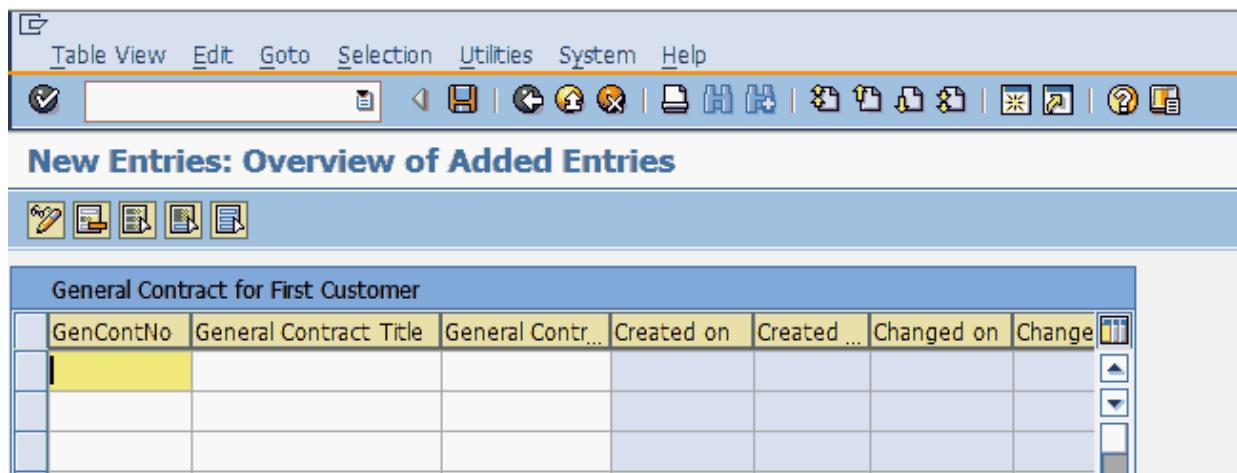


Figure 2.39: Modified screen of the table maintenance view

The fields ERDAT, ERDAT by creation, AEDAT, and AENAM by changing a record have to be filled automatically. Therefore, you have to create a function module for the event 01 of the table maintenance view.

To create this new function module, use **TRANSACTION SE11** in change mode for the table ZGENERALCONTRACT. Then use the menu **UTILITIES • TABLE MAINTENANCE GENERATOR.**

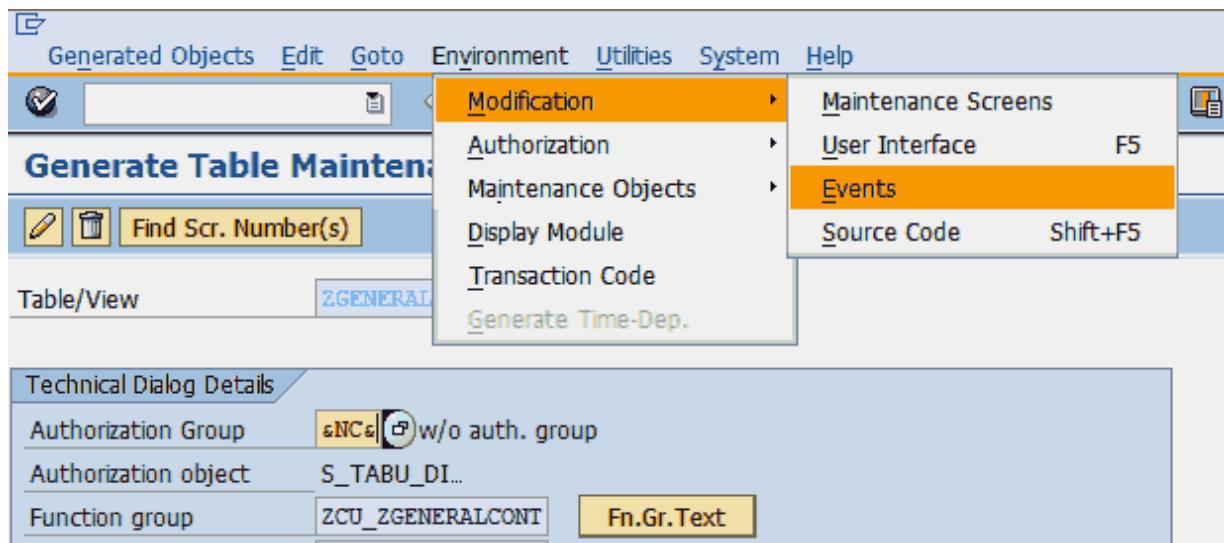


Figure 2.40: Event modification of table maintenance view

With menu **MODIFICATION • EVENTS**, you can activate new events (see Figure 2.40). Confirm the following popup screen (see Figure 2.41) by clicking the icon.

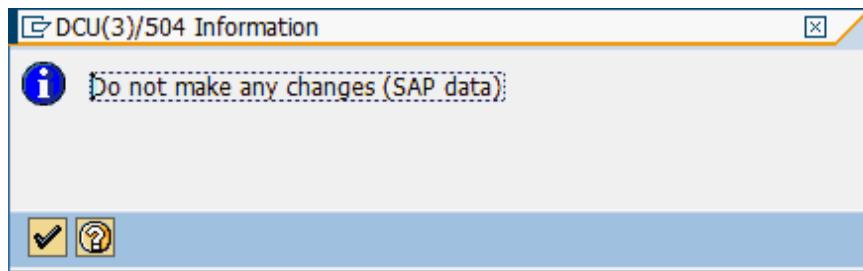


Figure 2.41: Confirmation of popup screen

By clicking the **New Entries** button on the next screen, you can create a form routine for the event 01 (before saving the data in the database). Hitting the “F4” help key on a field in the first column, event, shows a row of possible events you can create and form routines you can activate for the table maintenance view (see Figure 2.42).

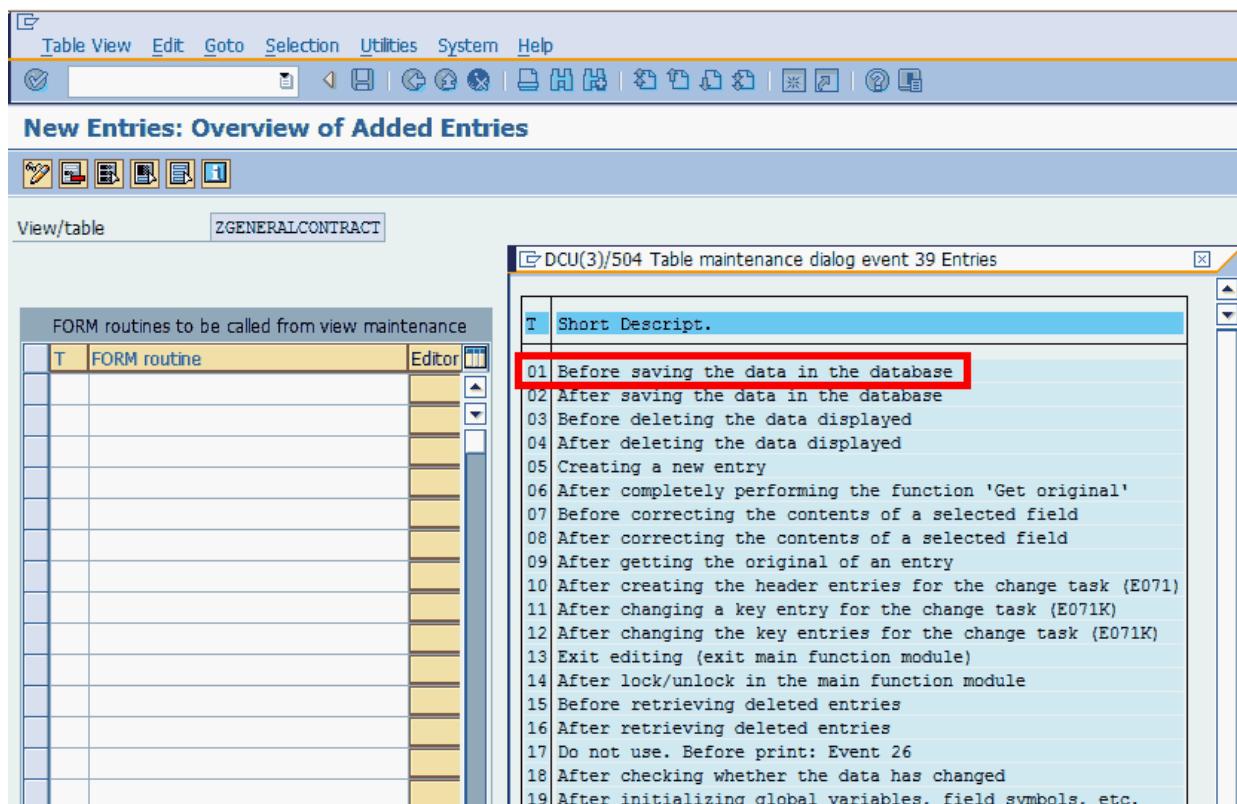


Figure 2.42: Events for table maintenance views

Save the form routine ZFIRST\_CUSTOMER\_DATE. With a mouse click in the last column, you will reach the **EDITOR** (see Figure 2.43) to develop the source code of the form routine.

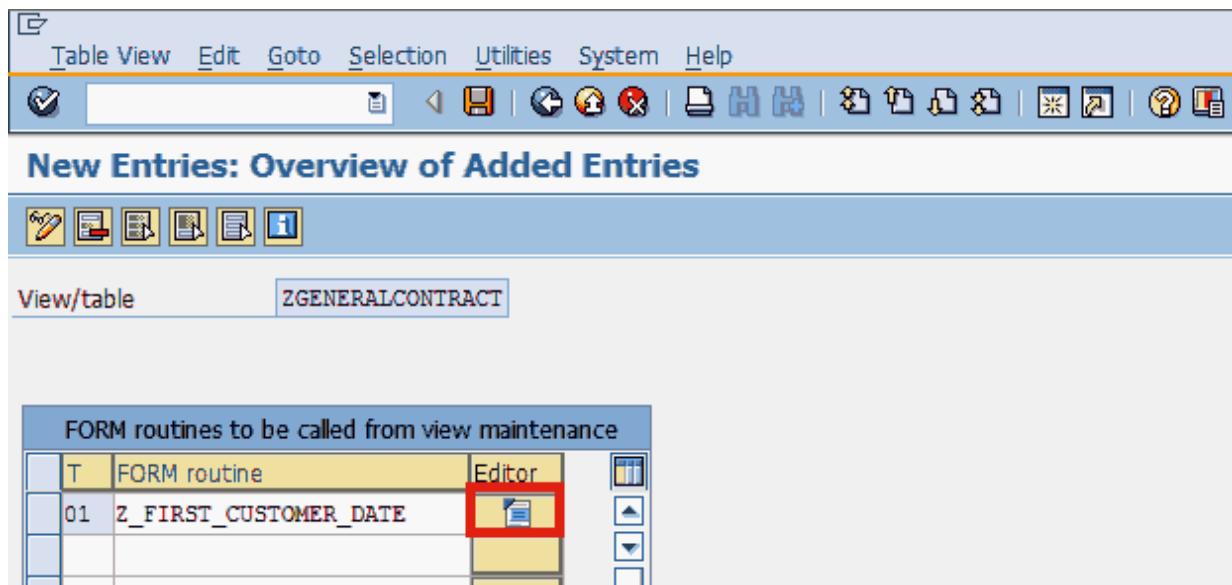


Figure 2.43: Form routine for an event of the table maintenance view

For the source code of the form routine, choose a “new include” of function group ZCU\_ZGENERALCONT (see Figure 2.44). This function group also contains the generated objects of the table maintenance view.

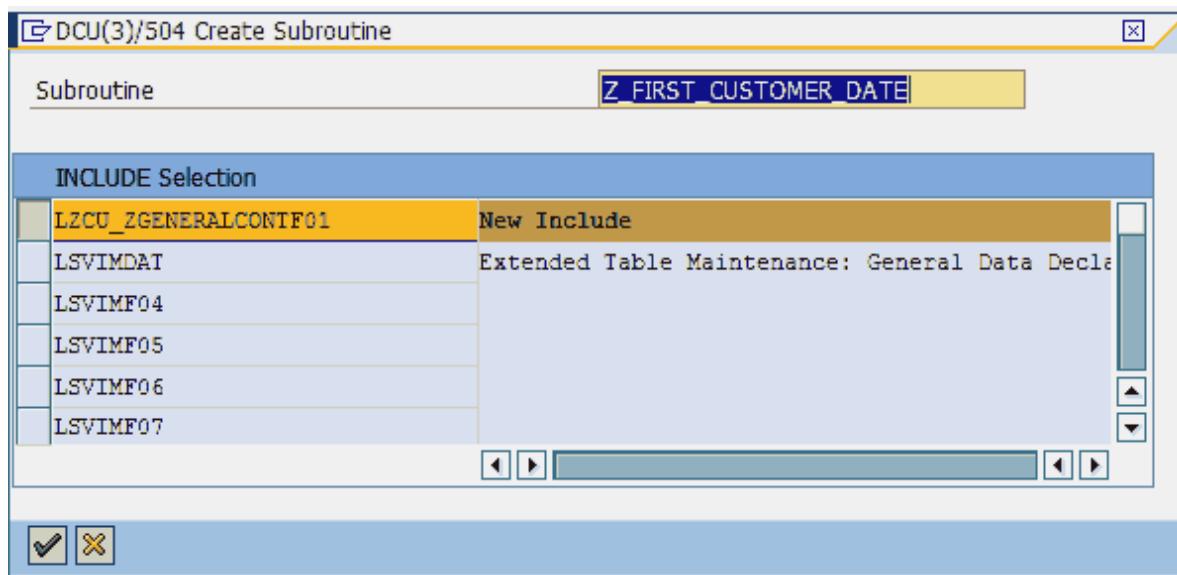


Figure 2.44: Function group of the table maintenance view

With a *breakpoint* in this subroutine, you can analyze the necessary source code while using **TRANSACTION** SM30 with the table ZGENERALCONTRACT to save or change records.

```

FORM z_first_customer_date.

DATA: lv_index      LIKE sy-tabix,
      lv_action(1) TYPE c.

LOOP AT total.
  lv_action = total+85(1).

  IF lv_action = 'N'. "New Entry
    READ TABLE extract WITH KEY <vim_xtotal_key>.
    IF sy-subrc EQ 0.
      lv_index = sy-tabix.
    ELSE.
      CLEAR lv_index.
    ENDIF.
    total+45(8)  = sy-datum. "Date on Which Record Was Created
    total+53(12) = sy-uname. "Name of Person who Created the Object
*   Change output-table "total"
    MODIFY total.
    CHECK lv_index > 0.
    extract = total.
    MODIFY extract INDEX lv_index.
  ENDIF.

  IF lv_action = 'U'. "Change Entry
    READ TABLE extract WITH KEY <vim_xtotal_key>.
    IF sy-subrc EQ 0.
      lv_index = sy-tabix.
    ELSE.
      CLEAR lv_index.
    ENDIF.
    total+65(8)  = sy-datum. "Change on
    total+73(12) = sy-uname. "Name of Person Who Changed Object
*   Change output-table "total"
    MODIFY total.
    CHECK lv_index > 0.
    extract = total.
    MODIFY extract INDEX lv_index.
  ENDIF.

ENDLOOP.

ENDFORM.                               "Z FIRST CUSTOMER DATE.

```

Figure 2.45: Coding for event 01 of the table maintenance view

It is difficult to find the number of digits where the contents of the fields ERNAM, AEDAT, ERDAT, and AENAM in the string TOTAL can be found (see Figure 2.45). Checking the functionality of the maintenance view shows that the subroutine creates the contents of the fields “Created on” (here

10.02.2015) and “Created by” (here STUTETH) when you insert a new record (see Figure 2.46).

GenContNo	General Contract Title	General Contract Keeper	Created on	Created by	Changed on	Changed by
7000000000	GENERAL CONTRACT 1	HEINEDA	10.02.2015	STUTETH		
7000000001	GENERAL CONTRACT 2	TEGETMA	10.02.2015	STUTETH		

Figure 2.46: Automatically fill the content in the fields “Created on” and “Created by”

Also, the contents of the fields “Changed on” and “Changed by” are filled by changing a record (see Figure 2.47).

GenContNo	General Contract Title	General Contract Keeper	Created on	Created by	Changed on	Changed by
7000000000	GENERAL CONTRACT 3	HEINEDA	10.02.2015	STUTETH	10.02.2015	STUTETH
7000000001	GENERAL CONTRACT 4	TEGETMA	10.02.2015	STUTETH	10.02.2015	STUTETH

Figure 2.47: Automatically fill the content in the fields “Changed on” and “Changed by”

You can find all developments for the table maintenance view of the table ZGENERALCONTRACT in the function group ZCU\_ZGENERALCONT, which has been used to create the table maintenance view. You can also make the necessary changes through the ABAP development environment (**TRANSACTION SE80**).

### 2.11.3 Create table maintenance transactions

In almost all production systems, very few users have permission to use **TRANSACTION SM30**. Therefore, a separate transaction must be applied for the maintenance of the data, the *table maintenance transaction*. The staff

that manages permissions can then assign permissions for executing the transaction as needed to responsible users.

To create this table maintenance transaction, use **TRANSACTION** SE93 (maintain transaction). Create a *parameter transaction* for the **TRANSACTION** Z\_GENERAL\_CONTRACT (see Figure 2.48).

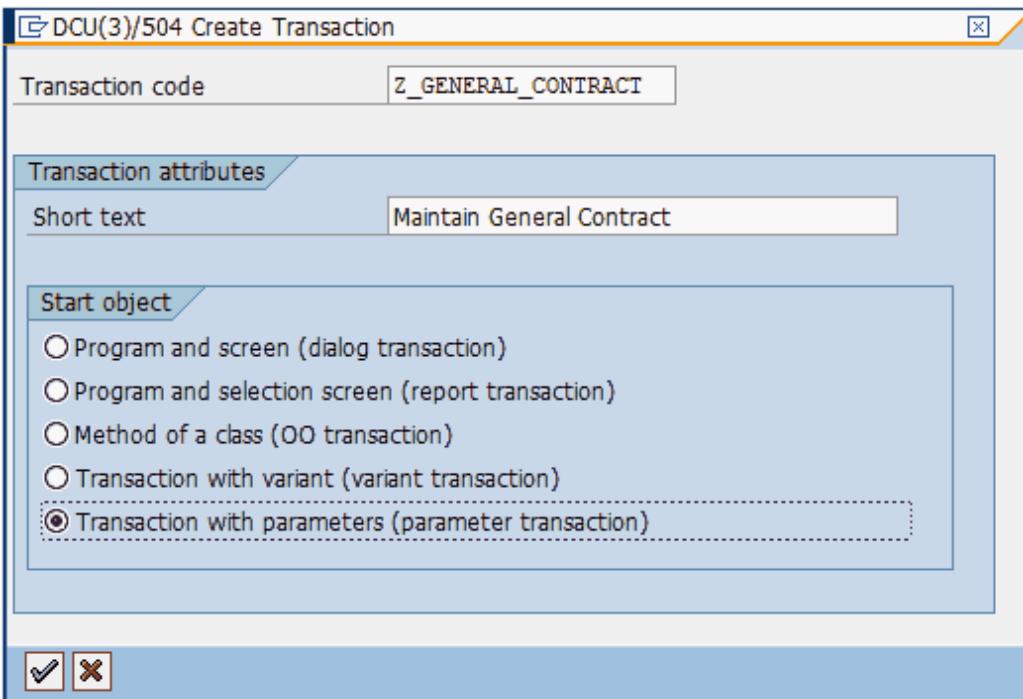


Figure 2.48: Creating a parameter transaction

To change the content of table ZGENERALCONTRACT, use new parameter **TRANSACTION** SM30 to fill in the name of the table that has to be maintained automatically.

To input the table name in **TRANSACTION** SM30, you need the name of this field in screen of **TRANSACTION** SM30. By using the parameter **TRANSACTION** SM30, the name of the table ZGENERALCONTRACT is populated automatically in this screen field (see Figure 2.49 and Figure 2.50). You can find this field name on the first screen of **TRANSACTION** SM30. Use “F1” help on the field **TABLE/VIEW** and then click the **Technical Information** button (see Figure 2.49).

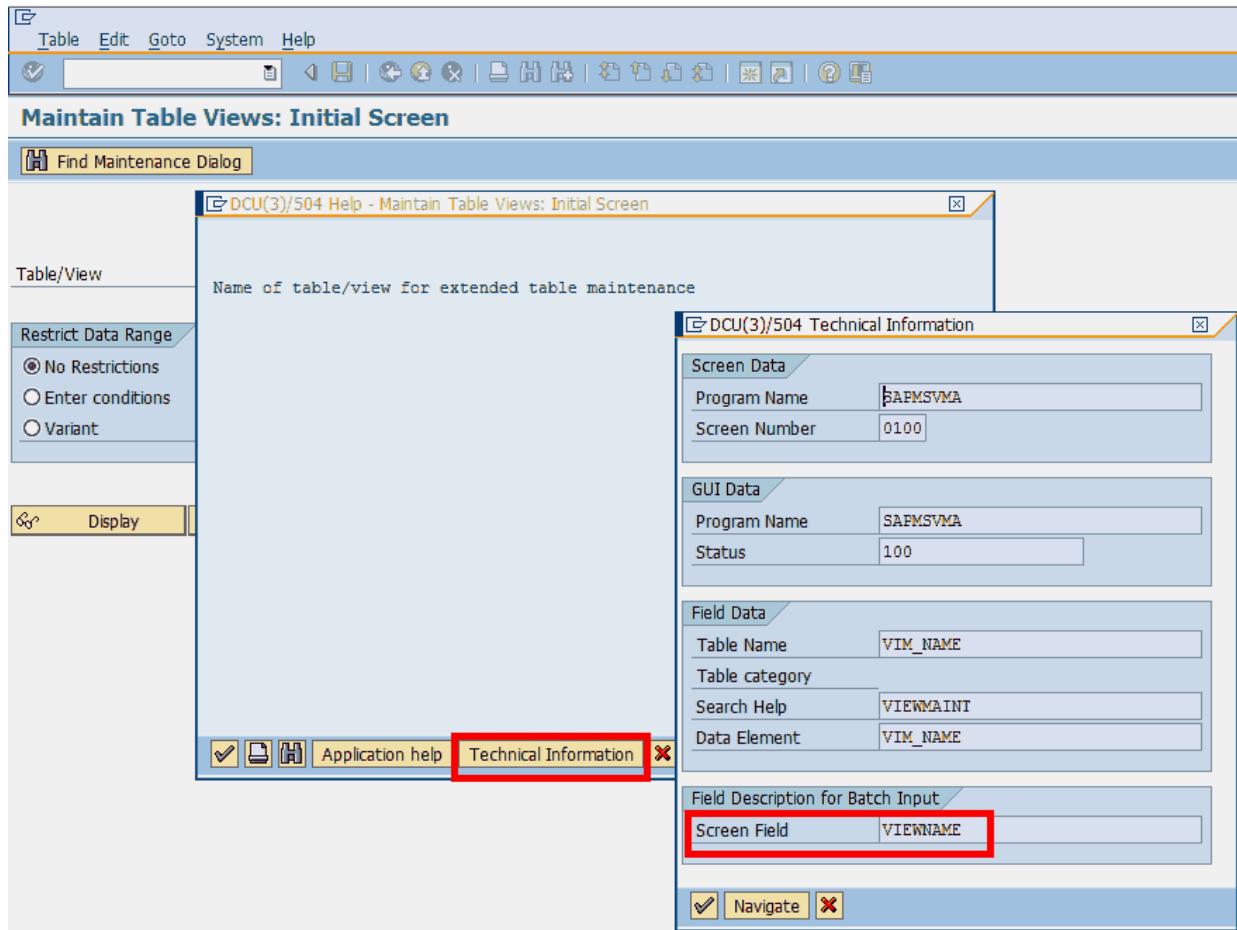


Figure 2.49: Technical information of a screen field

Alternatively, you can find the name “Update” for the button on the screen 0100 in the module pool SAPMSVMA.

Figure 2.50 shows all the settings required to create a **TRANSACTION** named Z\_GENERAL\_CONTRACT. A user with the appropriate permissions can call these transactions in the production system.

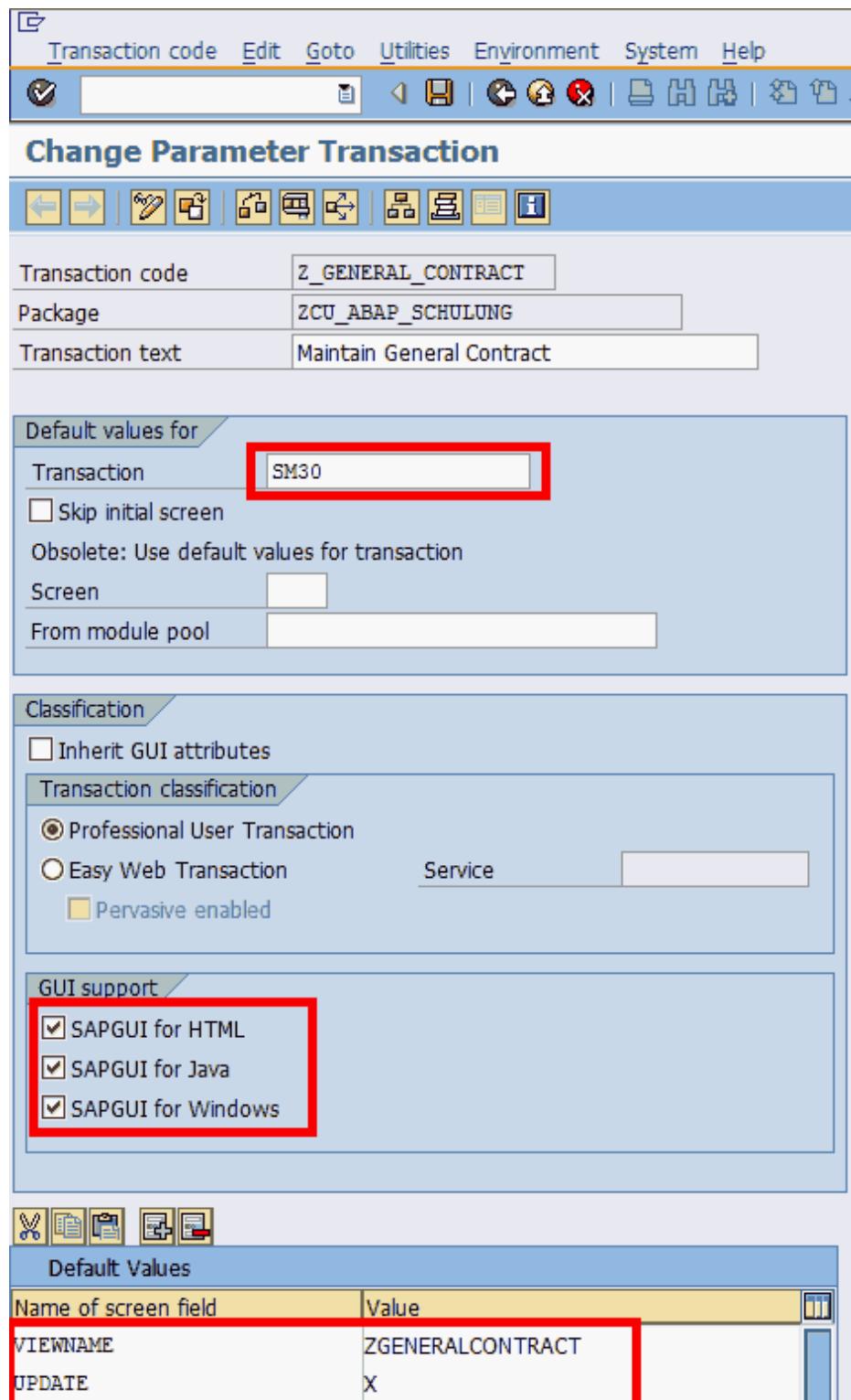


Figure 2.50: Creating a table maintenance transaction

## 2.12 Search help

In many SAP screens, you can use the “F4” help key to input values. With this function, the user can display all possible values for a field. In addition, “F4” help provides extensive search capabilities.

By calling “F4” help, the program runs through the checks shown in the *search help hierarchy* (see Figure 2.51).

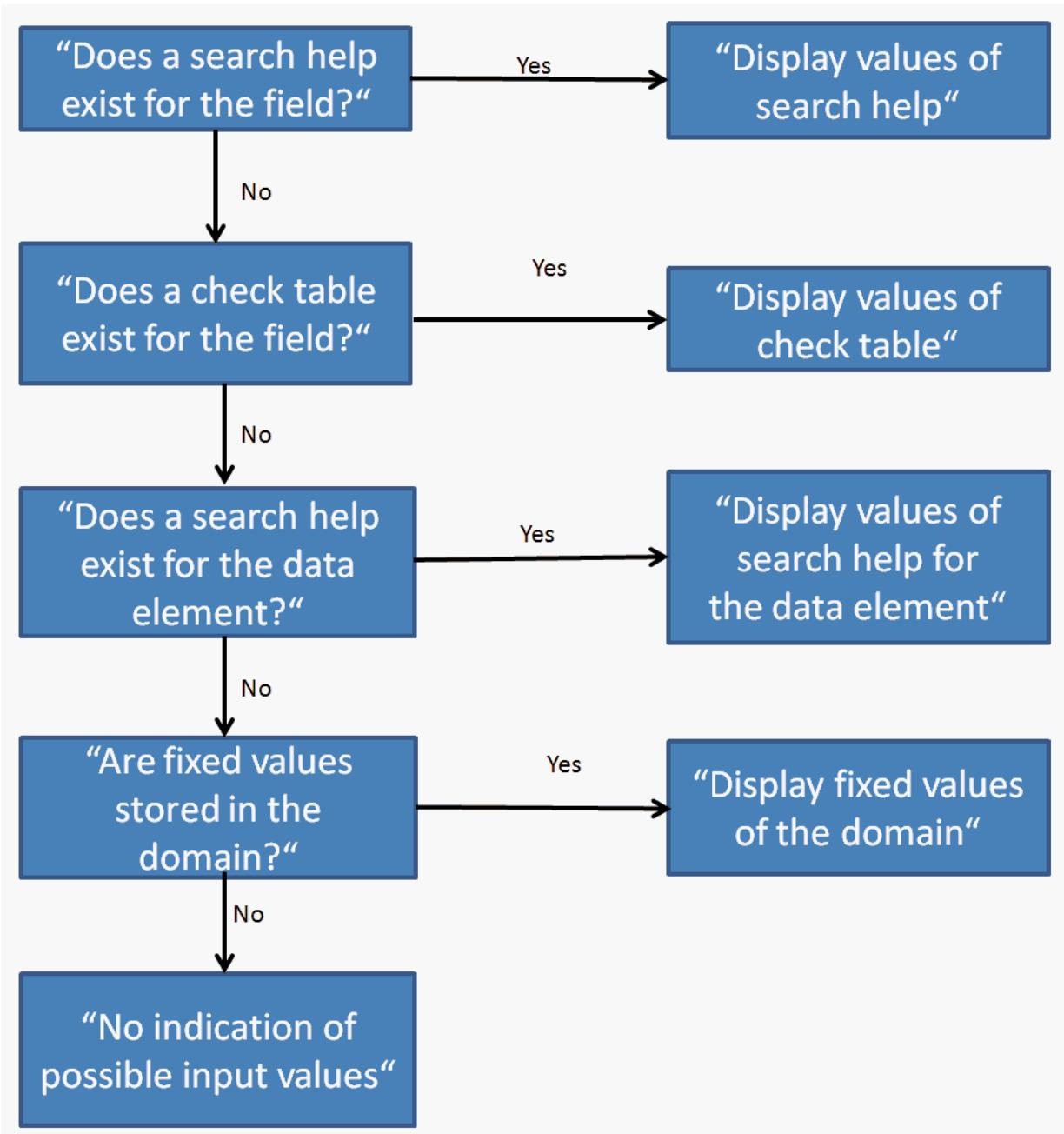


Figure 2.51: Hierarchy of input check for screen fields

I have already described the properties of check tables, foreign keys, and defaults in domains. Search help is the most powerful tool for users to find specific records. For example, we use the search help for business partner. The user can display **BUSINESS PARTNER** with **TRANSACTION BUG3**.

By hitting the “**F4**” help key in the business partner field, the user gets a complex search help (see Figure 2.52).

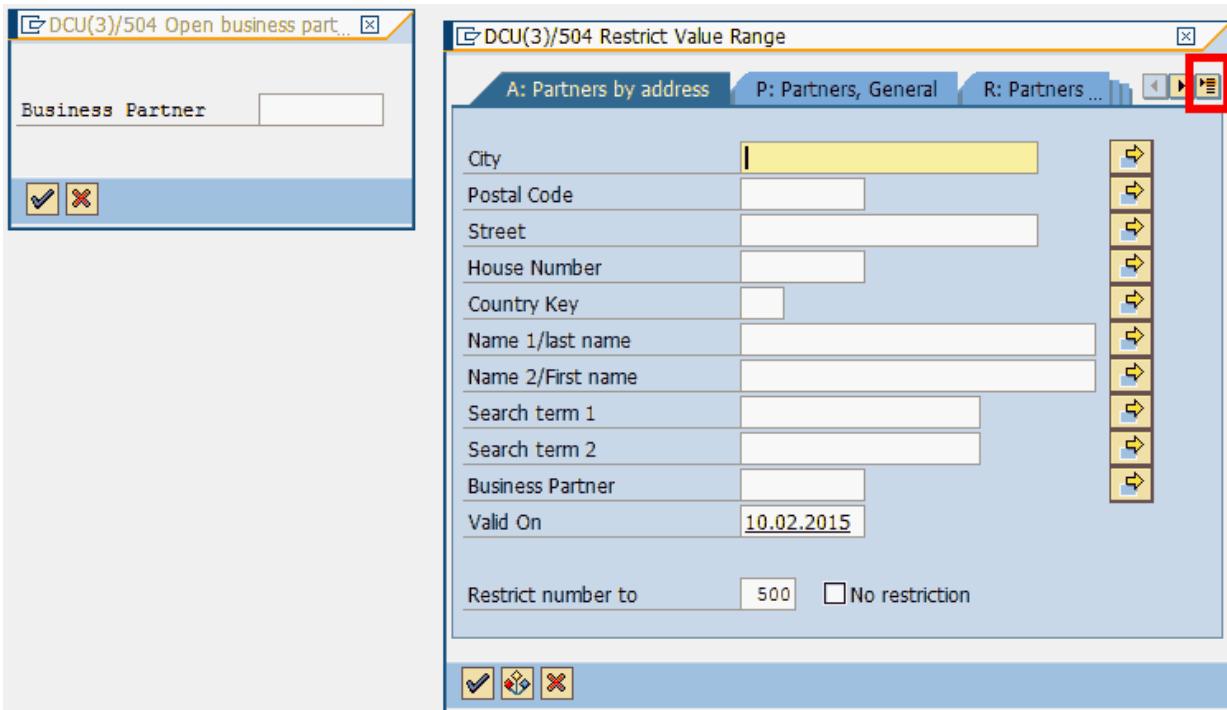


Figure 2.52: Complex search help for business partner

Click on the icon in the right upper corner to show the search helps that can be used in different tabs. With these complex search helps, the user can select business partner with different criteria (see Figure 2.53).

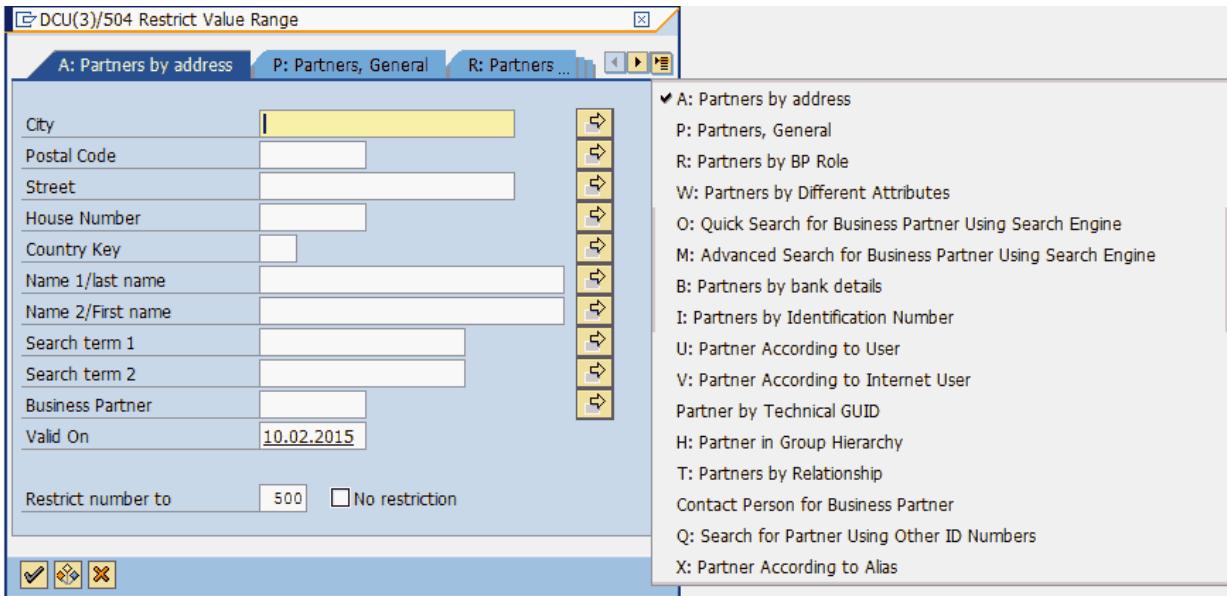


Figure 2.53: Complex search help for business partner

Such complex search helps are named by SAP as *collective search helps*. They consist of different *elementary search helps*.

In the example, you can see a standard SAP search help (Figure 2.53).

To learn which search help is used in our example, you have to use the search help hierarchy (search help for a field ( check table ( search help for a data element ( fixed values in the domain)).

Hit the “F1” key on the **BUSINESS PARTNER** field and click on the **Technical Information** button to find out which screen help is used in the current program (see Figure 2.54).

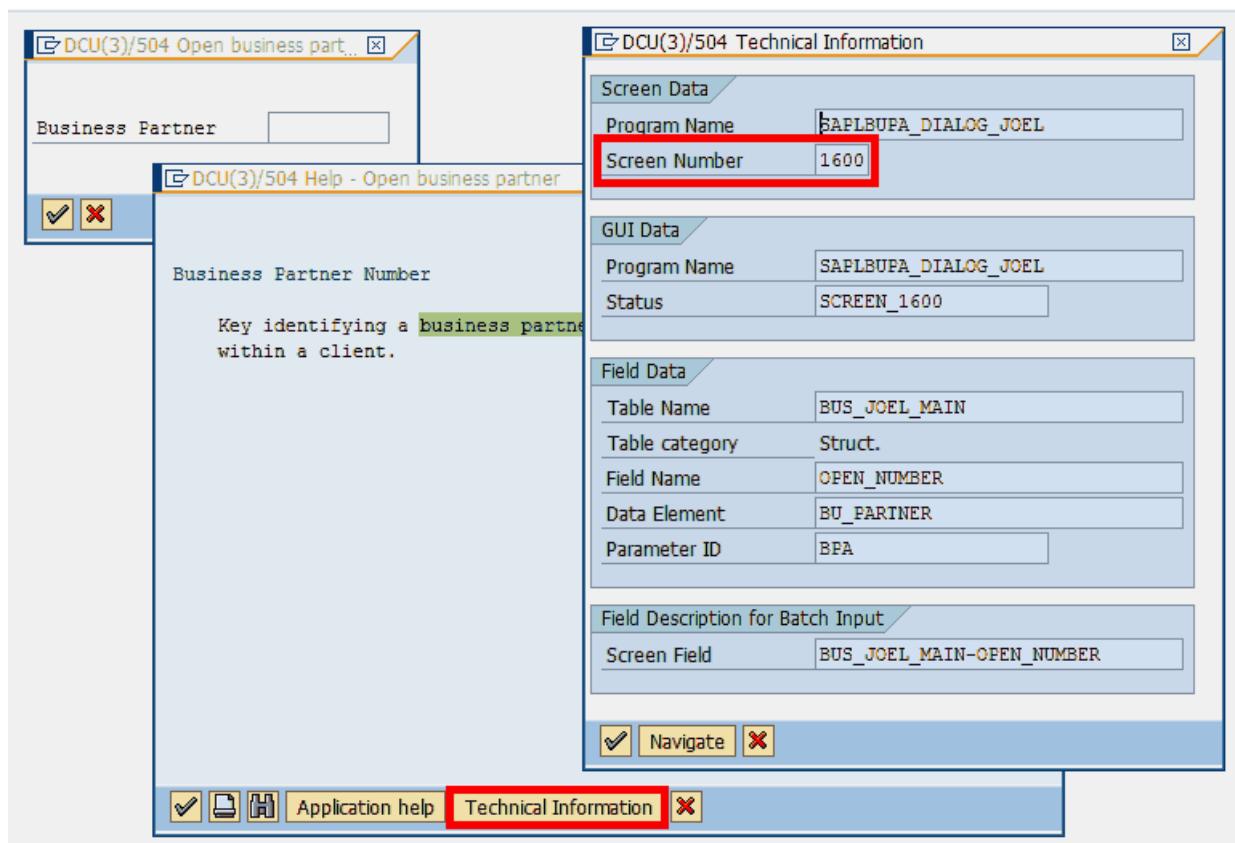


Figure 2.54: The search for the search help

There are two places where the search help can be found.

1. The search help is linked to the screen field OPEN\_NUMBER in screen 1600 of the program SAPLBUPA\_DIALOG\_JOEL.

2. The search help is linked to a field of the check table for the domain of the **DATA ELEMENT** BU\_PARTNER.

To analyze the information, double click on **SCREEN NUMBER** 1600, as shown in Figure 2.54. This shows you the developer's perspective of the screen (see Figure 2.55).

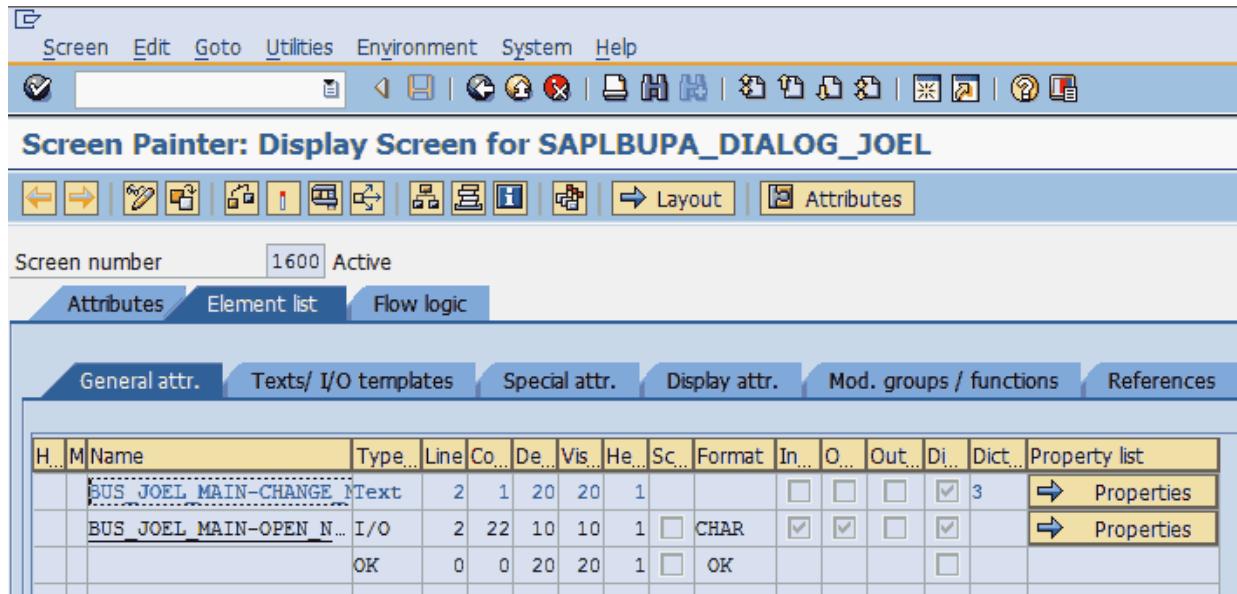


Figure 2.55: The developer's perspective

Click on the button to see the Screen Painter screen (see Figure 2.56).

Double click on the input field for **BUSINESS PARTNER** to see its properties.

Figure 2.56 shows that the **SEARCH HELP** field in attributes is empty, meaning the “F4” search help is not linked to the screen field.

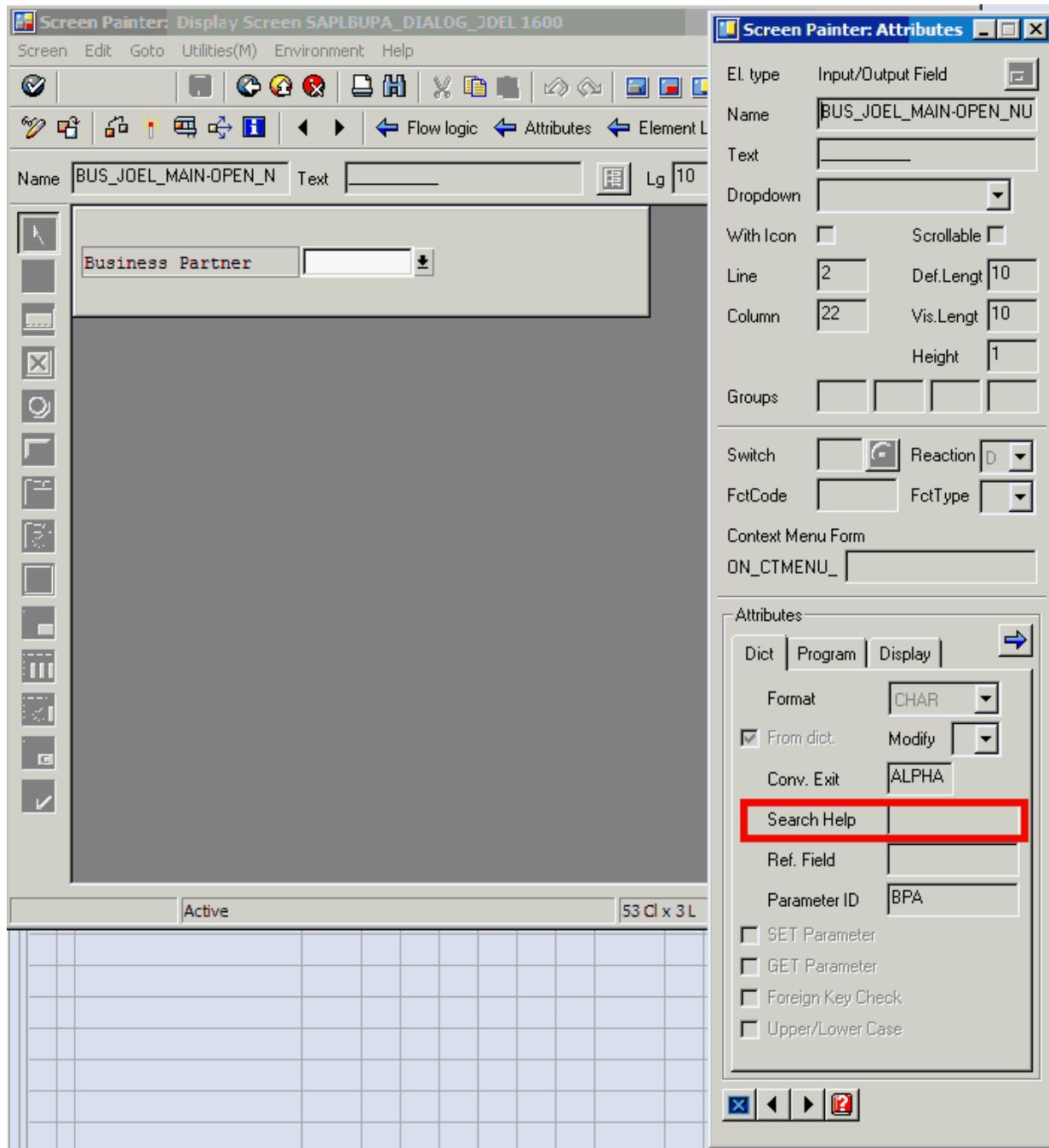


Figure 2.56: Layout editor

Now analyze the second possibility: The search help is linked to a field of a table. Close the layout editor screen and the developer's perspective screen. Double click on the data element BU\_PARTNER and then click on the technical information button (see Figure 2.54). The data element is shown in **TRANSACTION SE11** (see Figure 2.57).

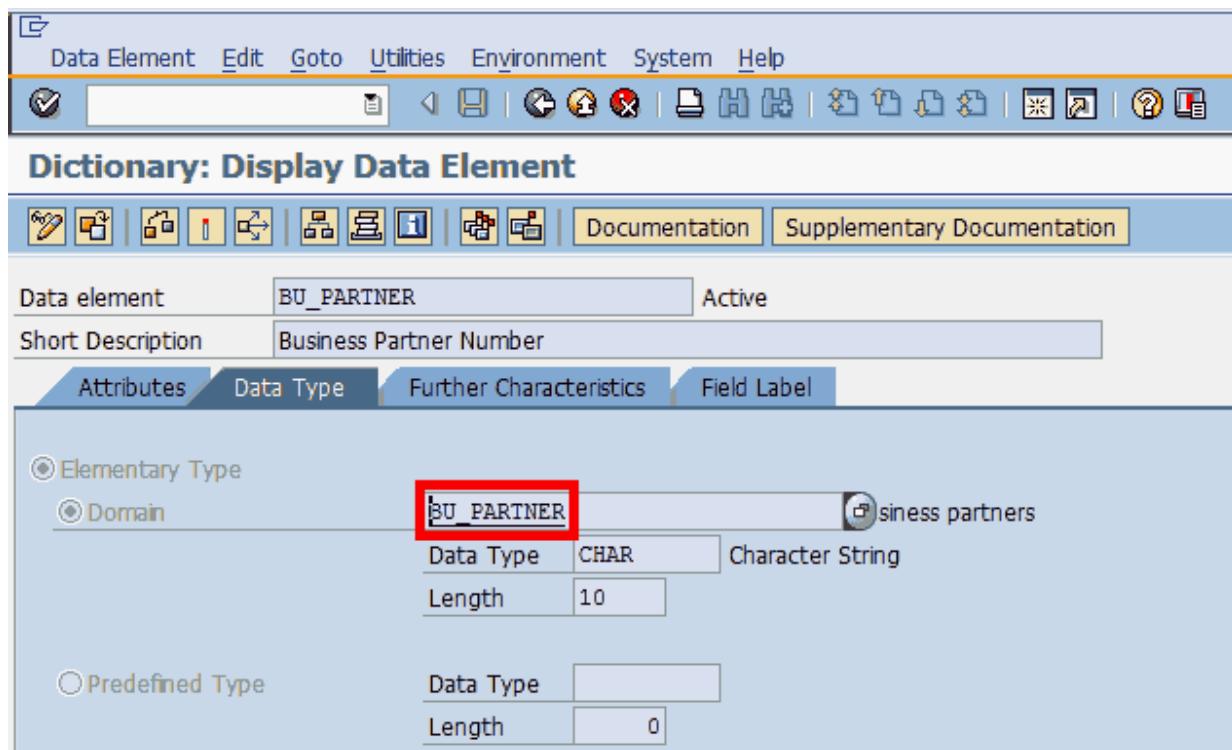


Figure 2.57: Data element and its domain

The data element BU\_PARTNER owns the **DOMAIN** of the same name. Double click on the domain name BU\_PARTNER. This domain is shown in transaction SE11 (see Figure 2.58).

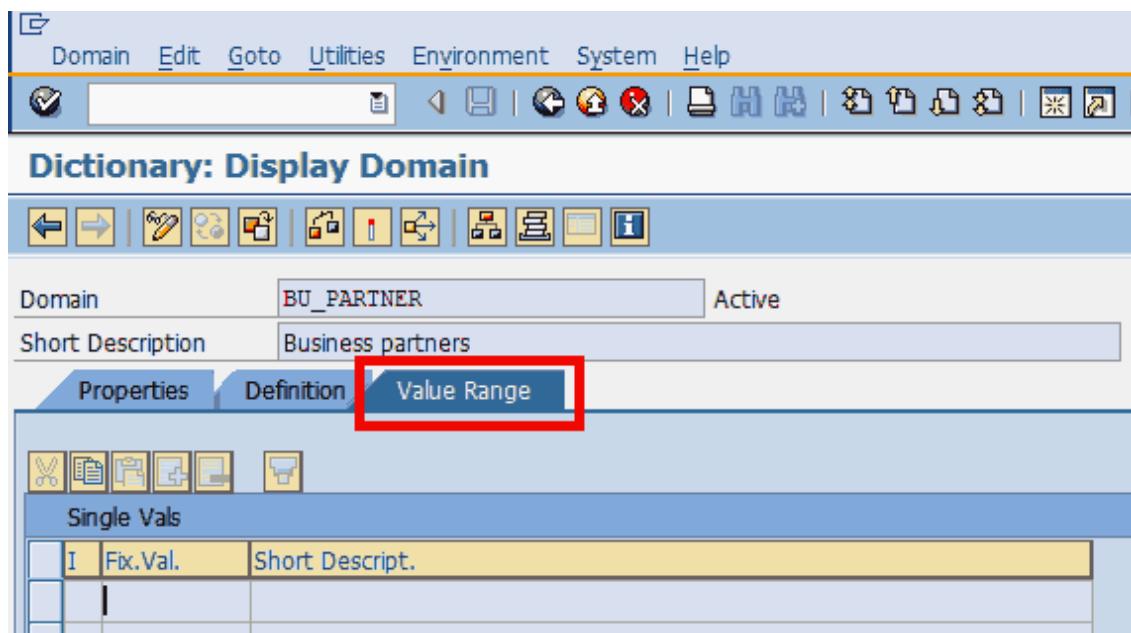


Figure 2.58: The domain BU\_PARTNER

The database **TABLE BUT000** is shown in the field **VALUE TABLE** (see Figure 2.59) of the tab **VALUE RANGE**.

A screenshot of the SAP Dictionary interface. At the top, there is a table titled "Intervals" with columns: Lower limit, UpperLimit, and Short Descript.. Below this is a search bar with "Value Table" and "BUT000". A red box highlights the "Value Table" and "BUT000" fields.

Figure 2.59: Value table BUT000 for the domain BU\_PARTNER

With another double click on value **TABLE BUT000** (Figure 2.59), you will see the table in the display mode of transaction SE11 (Figure 2.60).

A screenshot of the SAP Dictionary interface, specifically the "Fields" tab for domain BUT000. The "Entry help/check" tab is highlighted with a red box. The table below shows fields like CLIENT, PARTNER, and .INCLUDE with their respective data elements and check tables. The "SRCH HELP" column for the PARTNER field shows "BUPA", which is also highlighted with a red box.

Field	Data element	Data Ty...	Foreign ...	Check table	Origin of the input help	Srch Help	De... Domain
CLIENT	MANDT	CLNT	<input checked="" type="checkbox"/>	T000	Input help implemented with c...	H T000	<input type="checkbox"/> MANDT
PARTNER	BU_PARTNER	CHAR	<input type="checkbox"/>		Explicit search help attachme...	BUPA	<input type="checkbox"/> BU_PARTNER
.INCLUDE	BUS000_DAT	...	<input type="checkbox"/>				<input type="checkbox"/>

Figure 2.60: Search help for domain BUT000

Go to tab **ENTRY HELP/CHECK**. In the **FIELD** column, find “Partner” with **DATA ELEMENT BU\_PARTNER**. You can see BUPA in the **SRCH HELP** column. This search help can be displayed with **TRANSACTION SE11** (see Figure 2.61).

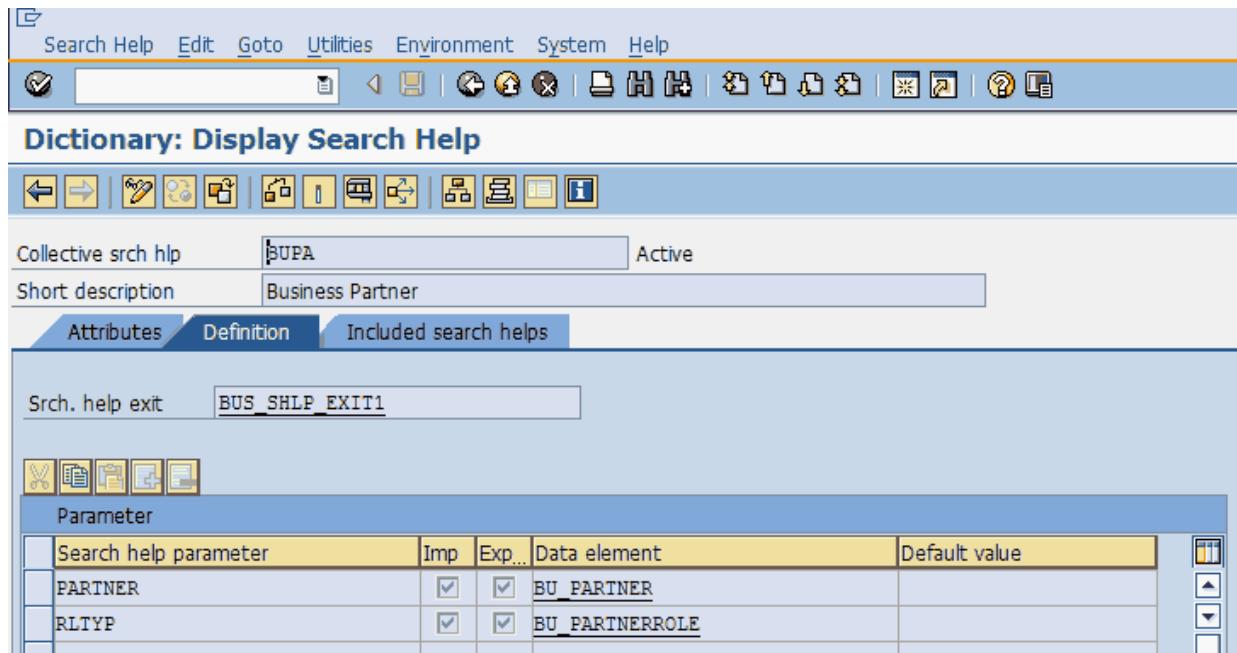


Figure 2.61: Search help for business partner in SE11

The BUPA search help is a collective search help. As shown on tab **INCLUDED SEARCH HELPS**, the elementary search helps are displayed in black, and the collective search helps are displayed in blue (see Figure 2.62).

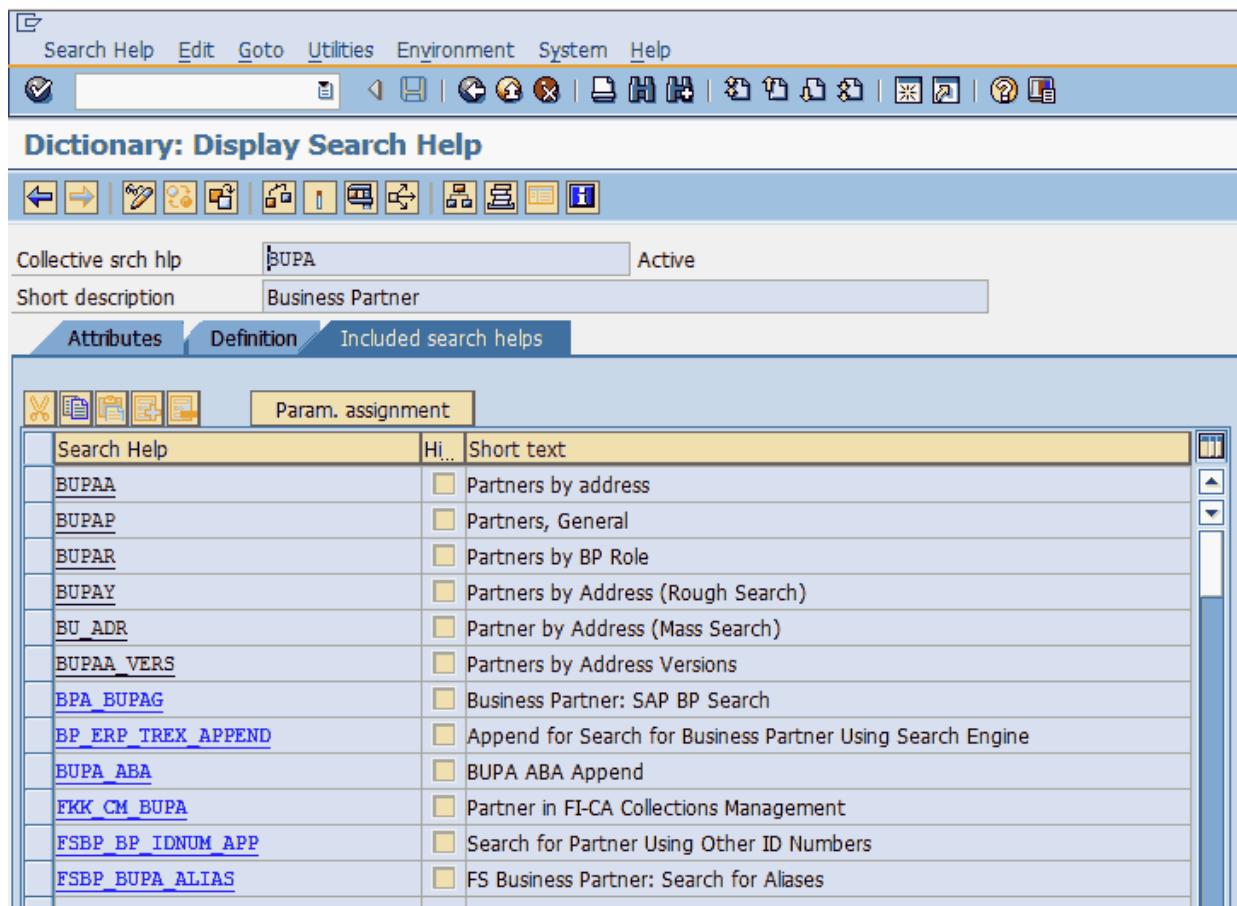


Figure 2.62: Collective search help BUPA and included elementary search helps

### 2.12.1 Modification-free changes of standard SAP search helps

Not all search helps delivered by SAP are of interest to all companies. For example, the search help for business partner “technical GUID” is used by SAP for worldwide identification of business partners. Consider that the standard SAP search help can be changed without carrying out an SAP modification by using a SSCR key. For every modification in a standard SAP system, you have to register an SSCR key so SAP can see which modifications are made.

In the following example, the standard SAP search help for business partners should only display the elementary search:

- ▶ BUPAA – Partners by address
- ▶ BUPAP – Partners, general

- ▶ BUPAR – Partner by BP role
- ▶ BUPAY – Partner by address (rough search)
- ▶ BU\_ADR – Partner by address (mass search)
- ▶ BUPAA\_VERS – Partner by address versions

Display the **COLLECTIVE SEARCH HELP** for BUPA with **TRANSACTION SE11** (see Figure 2.63).

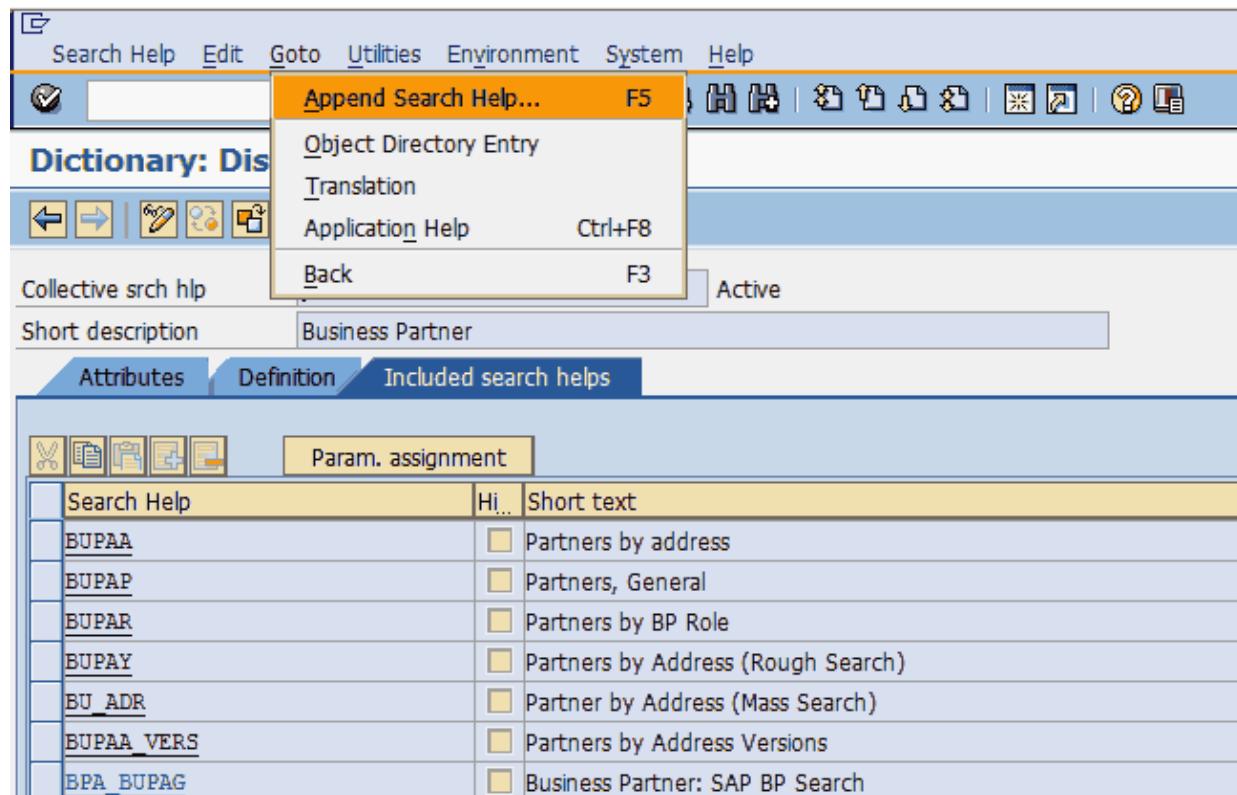


Figure 2.63: Find append search help of a search help

Use the pull-down menu **GOTO • APPEND SEARCH HELP** and find the tab **INCLUDED SEARCH HELPS**. Using this menu path, you can view the appended search helps of the BUPA search help (see Figure 2.64).

The screenshot shows a SAP application window titled "DCU(3)/504 Append for BUPA". The interface includes a toolbar with various icons for search, filter, and data manipulation. Below the toolbar is a table with three columns: "Object Name", "Status", and "Short text". The table lists several entries:

Object Name	Status	Short text
BPA_BUPAG	Active	Business Partner: SAP BP Search
BP_ERP_TREX_APPEND	Active	Append for Search for Business Partner Using...
BUPA_ABA	Active	BUPA ABA Append
CMAC_ASH_BUPA	New	Append SearchHelp for Business Partner
FKK_CM_BUPA	Active	Partner in FI-CA Collections Management
FSBP_BP_IDNUM_APP	Active	Search for Partner Using Other ID Numbers
FSBP_BUPA_ALIAS	Active	FS Business Partner: Search for Aliases
ISM_MJGMO	New	Search Helps for IS-M Roles

At the bottom of the screen are two buttons: a checked checkbox and an unchecked X button.

Figure 2.64: Append search help of a search help

Now include a new search help named ZBUPA as an append search help to the standard SAP search help BUPA.

Click the **CREATE** icon. Enter the name of the new append search help ZBUPA, as shown in Figure 2.65, and confirm by clicking the icon.

The screenshot shows a SAP application window titled "DCU(3)/504 Create Append Search Help for BUPA". It has a single input field labeled "Append Name" containing the value "ZBUPA". At the bottom are two buttons: a checked checkbox and an unchecked X button.

Figure 2.65: Create a customer-specific append search help

Now you have to take all of the elementary search helps for the standard SAP search help BUPA to the new append search help called ZBUPA. Now you have to include all elementary search helps of this append search help to hide them afterwards from the standard SAP search help (see Figure 2.66).

It is best practice to display the standard search help in a second SAP screen mode. Display all elementary search helps of the append search help BUPAA\_VERS with a double click in your first screen mode. In the case of the append search help BUPAA\_VERS, there is only the one elementary search BP\_BUPAG. Instead of the append search help BUPAA\_VERS,

include the phrase elementary search help BP\_BUPAG to the new search help ZBUPA.

This procedure is applied to all append search helps from a BUPA standard search help.

Figure 2.66 shows all included search helps of the new **COLLECTIVE SEARCH HELP** ZBUPA.

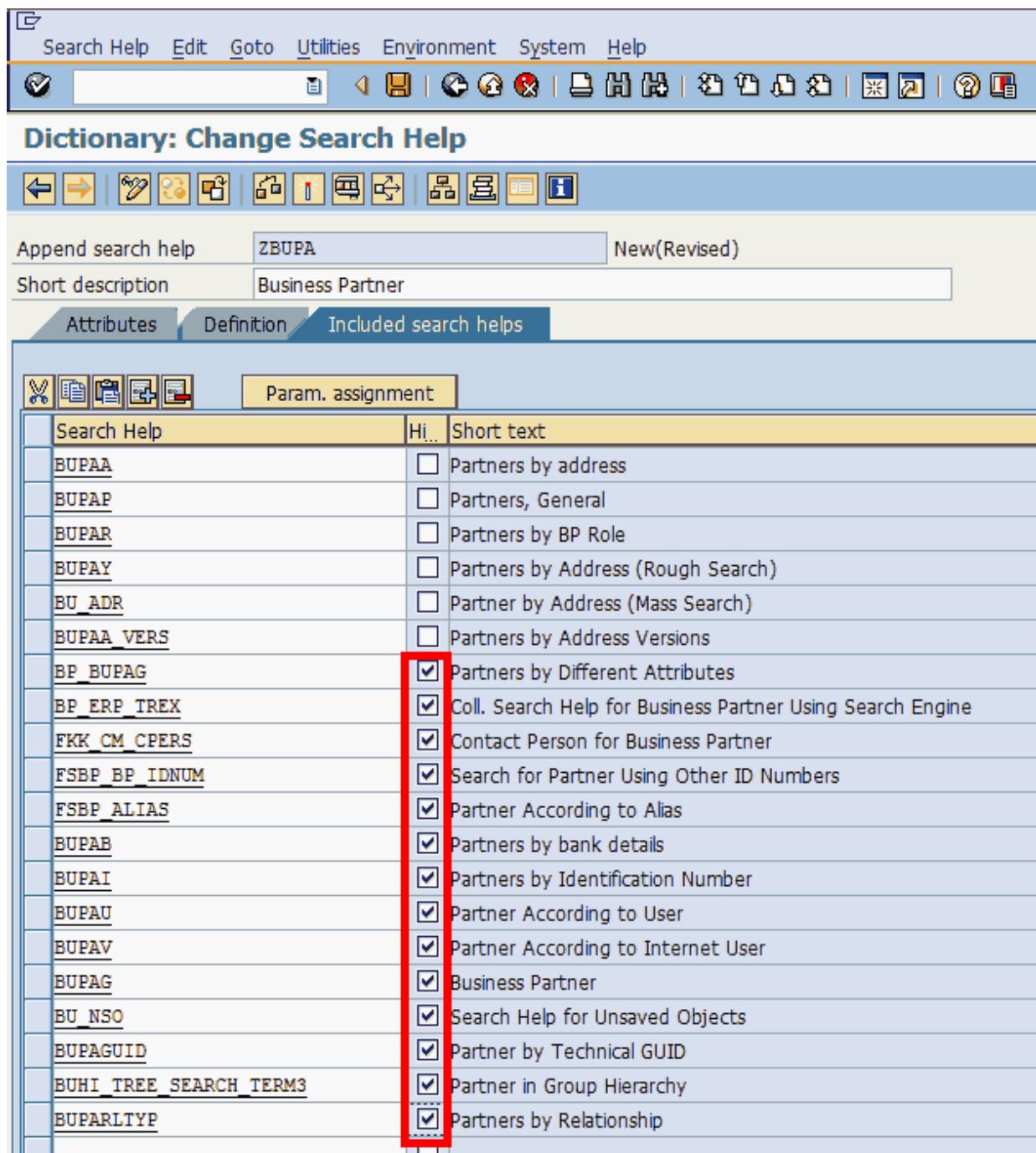


Figure 2.66: Hide elementary standard SAP search helps

Click the associated box in the **HIDDEN** column for all search helps that should no longer be displayed to the user.

Finally, activate and save the new append search help. The display of the standard SAP search help BUPA now shows the included **SEARCH HELP**

ZBUPA. See Figure 2.67.

The screenshot shows the SAP Fiori interface for managing search helps. At the top, there's a header with 'Collective srch hlp' and a search bar containing 'BUPA'. To the right of the search bar is the status 'Active'. Below the header, there are three tabs: 'Attributes', 'Definition', and 'Included search helps'. The 'Included search helps' tab is selected. In the top right corner of this tab, there's a button labeled 'Param. assignment'. Below this, there's a toolbar with icons for delete, edit, copy, move, and add. A table lists various search helps, each with a small icon and a brief description. The row for 'ZBUPA' is highlighted with a red box. The table has columns for 'Search Help' and 'Hi...'. The 'Search Help' column contains names like 'BUPAA', 'BUPAP', etc., and the 'Hi...' column contains descriptions like 'Partners by address', 'Partners, General', etc. The 'ZBUPA' row is specifically described as 'Business Partner'.

Search Help	Hi...
BUPAA	Partners by address
BUPAP	Partners, General
BUPAR	Partners by BP Role
BUPAY	Partners by Address (Rough Search)
BU_ADR	Partner by Address (Mass Search)
BUPAA_VERS	Partners by Address Versions
BPA_BUPAG	Business Partner: SAP BP Search
BP_ERP_TREX_APPEND	Append for Search for Business Partner Using Search Engine
BUPA_ABA	BUPA ABA Append
FKK_CM_BUPA	Partner in FI-CA Collections Management
FSBP_BP_IDNUM_APP	Search for Partner Using Other ID Numbers
FSBP_BUPA_ALIAS	FS Business Partner: Search for Aliases
ZBUPA	Business Partner

Figure 2.67: Customer-specific append search help in a standard SAP collective search help

Hiding search helps in ZBUPA means that the search helps are not shown in BUPA for the business partner (see Figure 2.68).

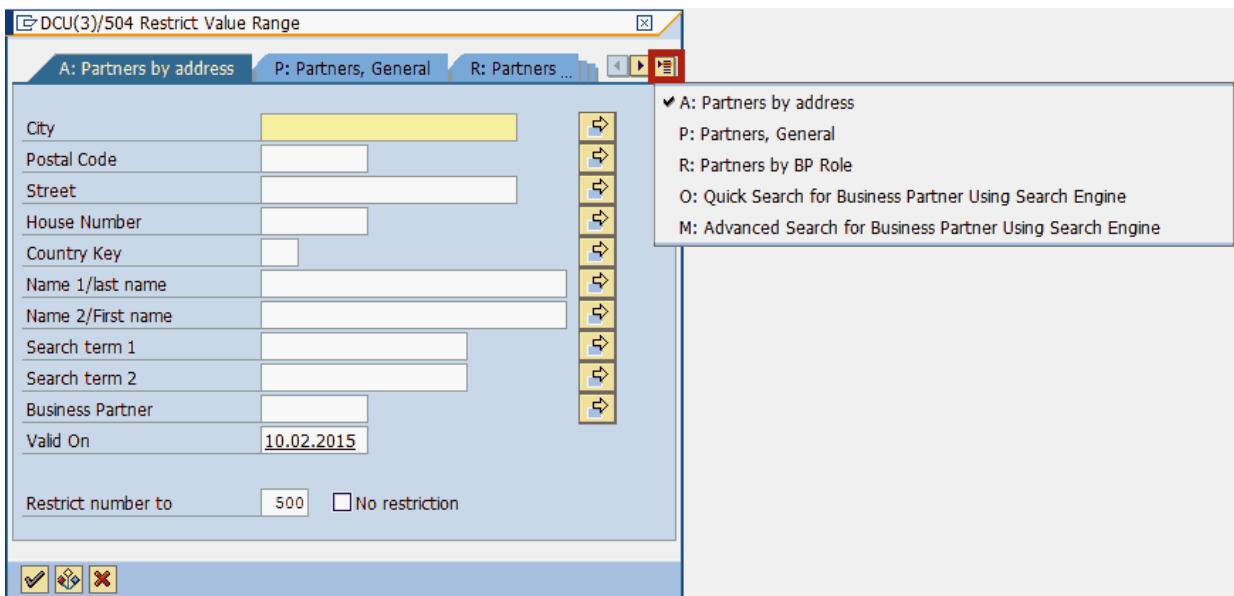


Figure 2.68: Modification-free change of a complex standard SAP search help

For comparison, check out the unchanged standard SAP search help (see Figure 2.69).

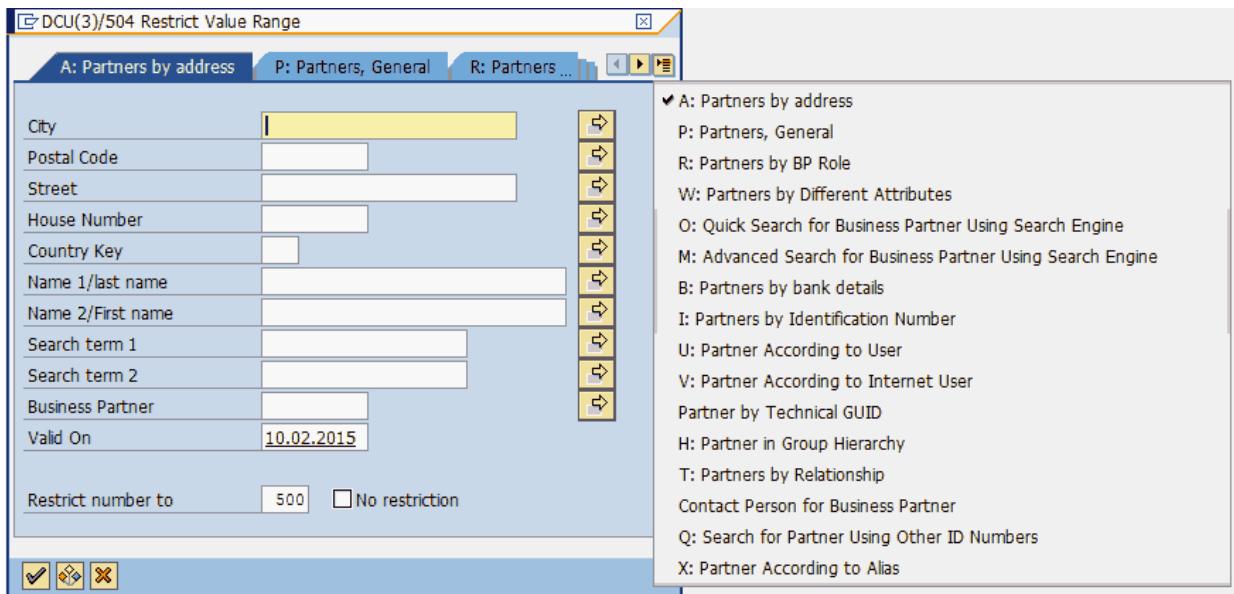


Figure 2.69: Complex standard SAP search help for business partner

## 2.12.2 Search help exits to limit the number of hits

Search helps are needed to give the user a specific number of possible input values. These hit amounts can be enhanced or limited with the help of *search*

*help exits*. Search help exits are function modules and can be used only with elementary search helps.

To create customer-specific search help exits, use F4IF\_SHLP\_EXIT\_EXAMPLE. Copy the search help by creating a customer-specific search help exit and use the necessary parameters in the customer-specific function module. As an example, for the assumed changing and table parameters, see Figure 2.70 and Figure 2.71.

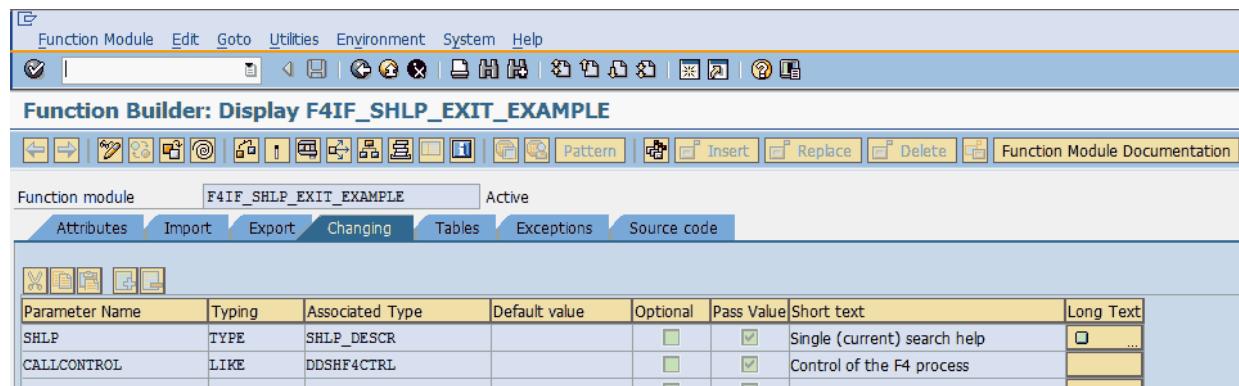


Figure 2.70: Changing the parameters of search help exits

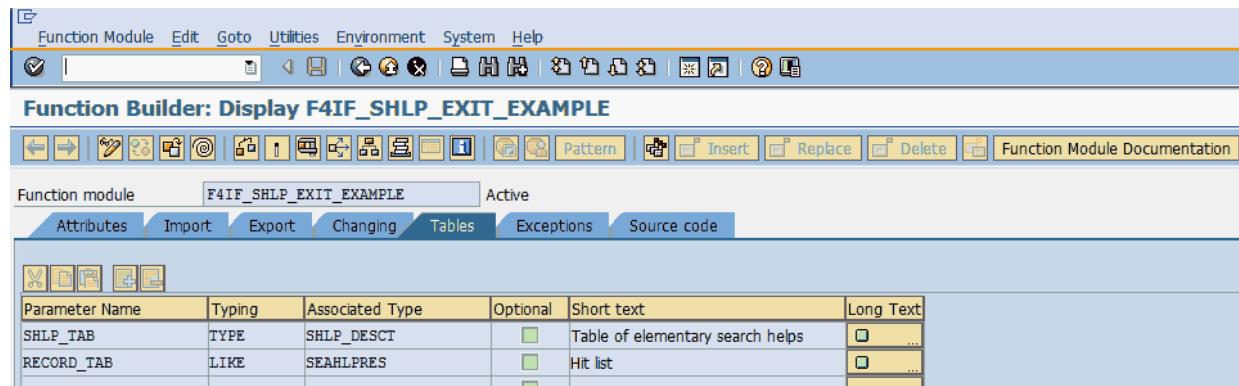


Figure 2.71: Table parameter of search help exits

The documentation of the standard SAP search help exit explains ([Function Module Documentation](#) button) that different events (*call control steps*) are processed by clicking the “F4” help key (see Figure 2.72).

The screenshot shows the SAP Function Builder interface with the title "Function Builder: Display F4IF\_SHLP\_EXIT\_EXAMPLE". The function module name "F4IF\_SHLP\_EXIT\_EXAMPLE" is selected in the top bar. The code editor displays the following ABAP code:

```
1 FUNCTION F4IF_SHLP_EXIT_EXAMPLE.
2   *"--*
3   **"Lokale Schnittstelle:
4   **" TABLES
5   **"     SHLP_TAB TYPE SHLP_DESCT
6   **"     RECORD_TAB STRUCTURE SEAHLPRES
7   **"     CHANGING
8   **"       VALUE(SHLP) TYPE SHLP_DESCR
9   **"       VALUE(CALLCONTROL) LIKE DDSHF4CTRL STRUCTURE DDSHF4CTRL
10  *"--*
11
12  * EXIT immediately, if you do not want to handle this step
13  IF CALLCONTROL-STEP <> 'SELONE' AND
14    CALLCONTROL-STEP <> 'SELECT' AND
15    " AND SO ON
16    CALLCONTROL-STEP <> 'DISP'.
17      EXIT.
18    ENDIF.
19
20  *"--*
21  * STEP SELONE (Select one of the elementary searchhelps)
22  *"--*
23  * This step is only called for collective searchhelps. It may be used
24  * to reduce the amount of elementary searchhelps given in SHLP_TAB.
25  * The compound searchhelp is given in SHLP.
26  * If you do not change CALLCONTROL-STEP, the next step is the
27  * dialog, to select one of the elementary searchhelps.
28  * If you want to skip this dialog, you have to return the selected
29  * elementary searchhelp in SHLP and to change CALLCONTROL-STEP to
30  * either to 'PRESEL' or to 'SELECT'.
31  IF CALLCONTROL-STEP = 'SELONE'.
32    * PERFORM SELONE .....
33    EXIT.
34  ENDIF.
```

Figure 2.72: Example coding in function module F4IF\_SHLP\_EXIT\_EXAMPLE

Customer-specific search help exits should include “SHLP\_EXIT” in its name. You can easily find these exits by hitting the “F4” help key in **TRANSACTION SE37** for maintaining function modules.

## Creating a search help exit



As a practical example, look at an IT service provider with different customers. All customers are working in the same SAP client and have different company codes. The collective search help BUPA for a business partner identifies all business partners stored in an SAP client without noticing the corresponding company code. The users in this SAP client are staff members of different customers. The assignment of a business partner to a company code can be found in the contract account. The database table FKKVKP has the fields “OPBUK” (company code group) or “STDBK” (standard company code).

Use the custom search help exit in this case, so that the “F4” help shows the business partner company code-dependent for the responsible users. You have to develop the search help exit so that business partner selection is user dependent. A business partner without a contract account is visible to all users.

For example, we used the modified standard SAP search help BUPA. To explain the method, it is enough to reduce the search help exit to one elementary search help. Use the elementary search help BUPAA “Partner by Address” and copy it to the elementary custom search help ZBUPAA. After this, include ZBUPAA in the custom search help ZBUPA and hide all other search helps in ZBUPA.

The search help ZBUPA now only shows the search “Partner by Address” (see Figure 2.73).

DCU(4)/504 Restrict Value Range

A: Partners by address

City		
Postal Code		
Street		
House Number		
Country Key		
Name 1/last name		
Name 2/First name		
Search term 1		
Search term 2		
Business Partner		
Valid On	10.02.2015	
Restrict number to	500	<input type="checkbox"/> No restriction

Figure 2.73: Modified search help for business partners

Selected values from the elementary search help ZBUPAA are given to the collective search help ZBUPA when you define the **PARAMETER ASSIGNMENT** between this search help on the level of the collective search help (see Figure 2.74).

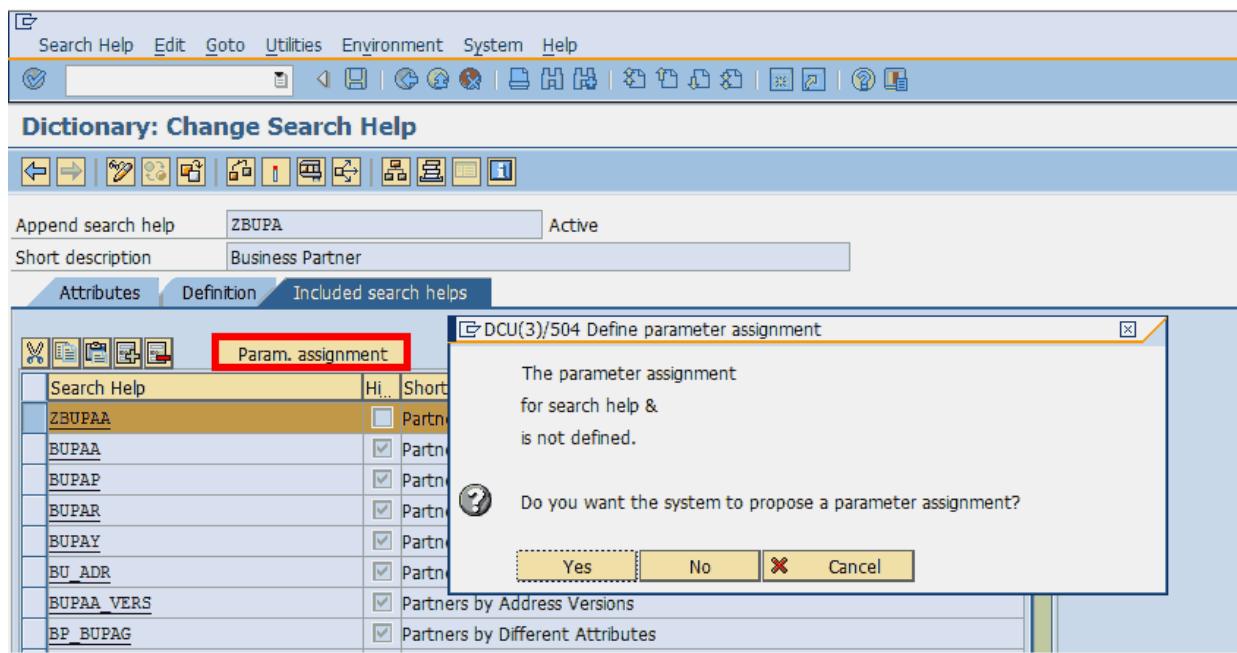


Figure 2.74: Parameter assignment of a collective search help and an elementary search help

By defining the parameter assignment, the SAP system makes a proposal for the assignment based on the defined fields (see Figure 2.75).

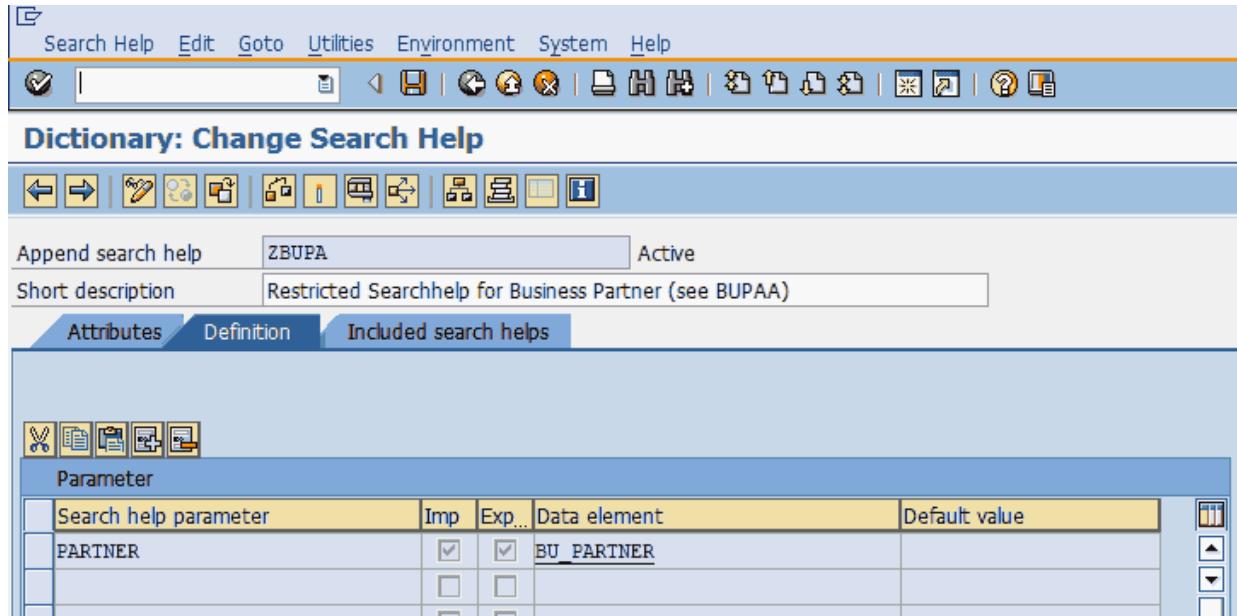


Figure 2.75: Proposal for defining the parameter assignment

If necessary, delete or insert parameter assignments. Accept your choice by clicking the **Copy** button and then activate the search help ZBUPA.

The standard search help exit for the search help BUPAA is the function module BUP\_SHLP\_EXIT\_SEVERALS. This belongs to the function group BUD\_SHLP. The search help exit uses subroutines with different selection methods, which also are needed for the new custom search help exit in a physically changed form. To do this, copy the whole **FUNCTION GROUP** BUD\_SHLP with the help of **TRANSACTION** SE80 to the new **FUNCTION GROUP** ZCU\_BUD\_SHLP (see Figure 2.76).

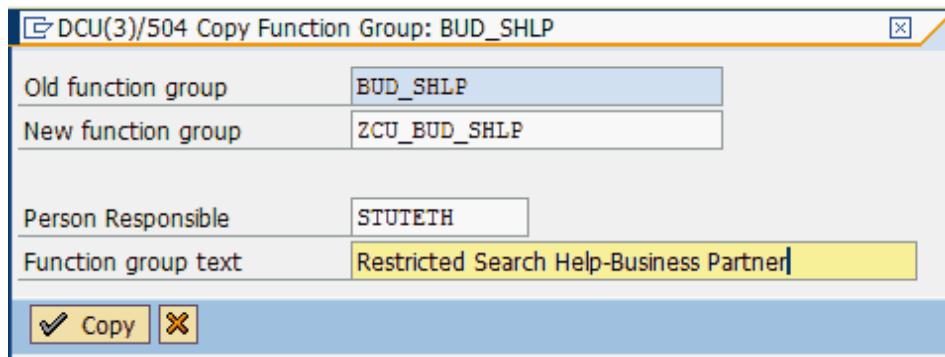


Figure 2.76: Copy of a function group

In Figure 2.77, all function modules of the function group are copied to the customer name, formatted as a Z-name.

Old function module name	New function module name
BUS_SHLP_EXIT1	ZCU_BUS_SHLP_EXIT1
BUP_SHLP_EXIT_SEVERALS	ZCU_BUP_SHLP_EXIT_SEVERALS
BUP_SHLP_EXIT_BPARTNER_ROLE	ZCU_BUP_SHLP_EXIT_BPARTNER_ROLE
BUP_SHLP_EXIT_BPARTNER_FUZZY	ZCU_BUP_SHLP_EXIT_BPARTNER_FUZZY
F4IF_SHLP_EXIT_ROLES	ZCU_F4IF_SHLP_EXIT_ROLES
BUS_RELTYF_F4_EXIT	ZCU_BUS_RELTYF_F4_EXIT
BUB_BUPR_RELKIND_F4_EXIT	ZCU_BUB_BUPR_RELKIND_F4_EXIT
BUP_SHLP_EXIT_TBZ9A	ZCU_BUP_SHLP_EXIT_TBZ9A

Copy   
    
    
    
 

Figure 2.77: Copy all function modules of a function group

Now you can make necessary enhancements to the new function modules.

Change the search help exit of **ELEMENTARY SEARCH HELP Z\_BUPAA** from the standard **BUP\_SHLP\_EXIT\_SEVERALS** to the custom **SEARCH HELP EXIT Z\_BUP\_SHLP\_EXIT\_SEVERALS** (see Figure 2.78).

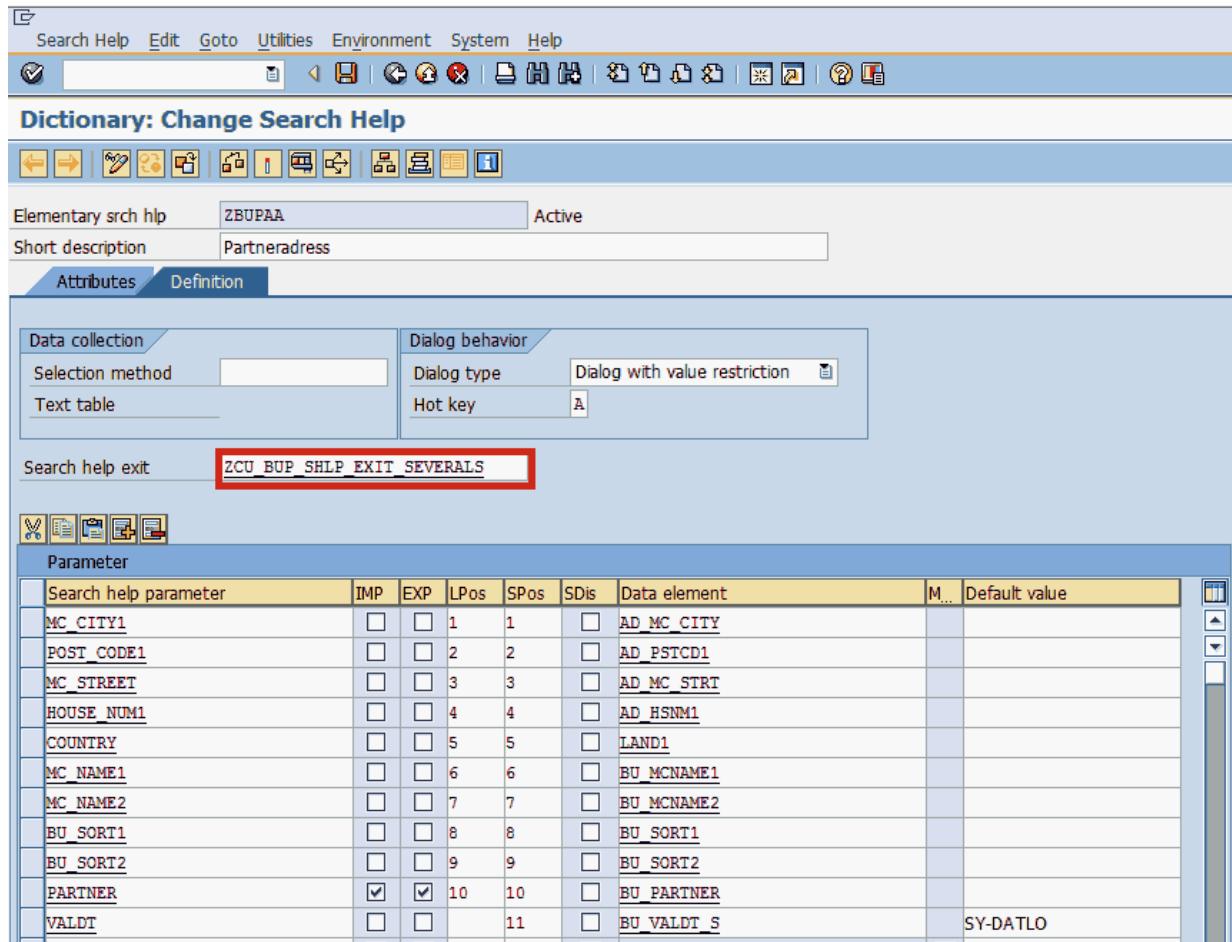


Figure 2.78: Usage of the custom search help exit

Throughout the new function group ZCU\_BUD\_SHLP in the “old” search help, BUPAA is questioned, and the new search help ZBUPAA is complemented (e.g., in IF or CHECK statements). Now global searches for terms in the function group ZCU\_BUD\_SHLP help locate these places (see Figure 2.79).

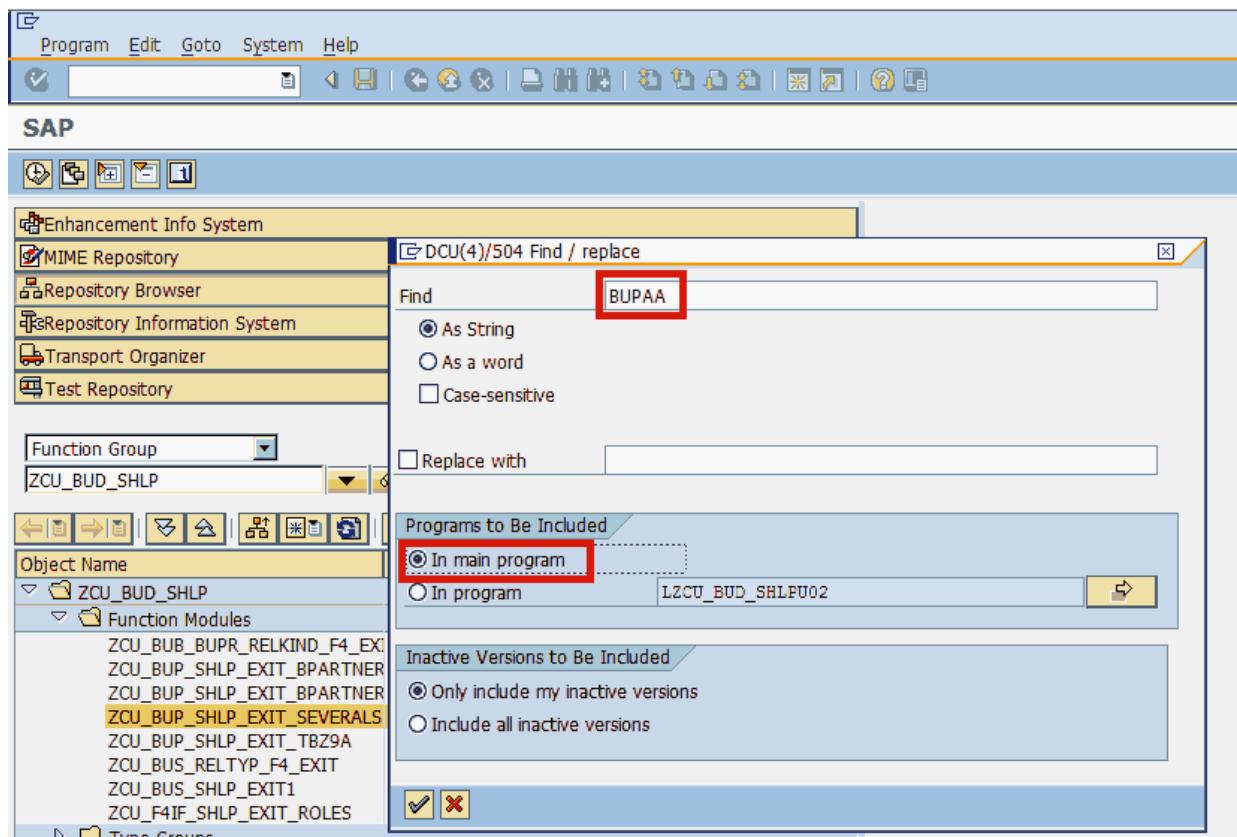


Figure 2.79: Modification of a copied standard SAP search help exit

A check of the hit list has to show that at all places where the query of the search help BUPAA is used, the new elementary search help is considered. With this change, the selection of business partner with the new search help will function perfectly (see Figure 2.80).

Program/Enhancement	Found locs/short description
ZCU_BUD_SHLP	<pre> 30      WHEN 'BUPAA' OR 'ZBUPAA' OR 'BUPAA_VERS' OR 'BUADR'. 643     CHECK shlp-shlpname = 'BUPAA'           OR shlp-shlpname = 'ZBUPAA'           OR shlp-shlpname = 'BUPAA_VERS'. 741     IF shlp-shlpname = 'BUPAA' OR shlp-shlpname = 'ZBUPAA'. 762         SELECT * FROM m_bupaa INTO CORRESPONDING FIELDS OF TABLE lt_result           WHERE partner IN lr_partner           AND post_code1 IN lr_pcode1           AND mc_street IN lr_street </pre>

Figure 2.80: Result of the global search for the term “BUPAA”

The new search help exit Z\_BUP\_SHLP\_EXIT\_SEVERALS can be adapted to the request, i.e. the company code-dependent selection of business partner can be realized.

You always need to know which company code a user is assigned to. This can be done by the authorization check or through a custom-specific table in which the assignment between user and company code is stored.

The enhancement for the company code-dependent selection of the business partner is in the search help exit Z\_BUP\_SHLP\_EXIT\_SEVERALS of the elementary search help ZBUPAA.

This code is inserted directly after the selection for output the hit list (i.e. in the CALLCONTROL-STEP = ‘SELECT’). See Figure 2.81.

```

*   Insert T. Stutenbäumer, ConUti GmbH, 10.02.2015
*   Check the table ZCU_USER_BUKRS if the user is
*   authorized to display and change the business partner
*   What is taken for the assignment of a business partner
*   to a company code, whether the business partner has a
*   contract account with the standard company code.
*   Has a business partner no contract, all users are
*   permitted to display and change the business partner
*   If the user does not have permission to view the
*   business partner from the company code to display or change,
*   so this business partner be deleted from the hit list

LOOP AT record_tab.
  lv_partner = record_tab-string+103(10).
  SELECT SINGLE stdbk
    FROM fkkvkp
    INTO lv_stdbk
    WHERE gpart = lv_partner.
  IF sy-subrc <> 0.
    CONTINUE.
  ENDIF.
  IF NOT lv_stdbk IS INITIAL.
    SELECT SINGLE *
      FROM zcu_user_bukrs
      INTO ls_zcu_user_bukrs
      WHERE username = sy-uname
        AND bukrs = lv_stdbk.
    IF sy-subrc <> 0.
      *          User don't have the permission
      DELETE record_tab.
    ELSE.
      *          User have the permission
      CONTINUE.
    ENDIF.
  ENDIF.
ENDLOOP.
*   End of Insert T. Stutenbäumer, ConUti GmbH, 10.02.2015

```

Figure 2.81: Coding for the company code-dependent display of business partners

You cannot view both tables BUT000 and FKKVKP here because a view only selects data with an *inner join*. That means the view only selects data that exists in both tables. Business partner without a contract account will not be shown.

Be careful: The selection of a business partner is limited through the maximum number of hits. In some cases, it is possible that after filtering the

company code-dependent business partner, no record is left. In such cases, a new business partner selection has to be made.

### 2.12.3 Search help for transfer of values to the screen

You can use search help to give values selected by hitting the “F4” help key, within defined fields on a screen.

Refer back to [Chapter 1](#) for details on how general contract data stored in a custom table is selected and assigned to a contract.

In [Section 2.6](#) we inserted some customer fields to include CI\_EVER (see Figure 2.16).

This data is shown in a customer-specific sub-screen for the contract. Using the “F4” help key enables you to select a general contract from the custom table ZGENERALCONTRACT and assign it to a contract.

The customer-specific screen enhancements are explained in the *Practical Guide to SAP ABAP: Part 2* in the chapter named “Changes and enhancements to SAP standards.” For now, assume that these developments have been made.

When using the search help for general contract data, the developer has to answer the question: Which fields should be transferred from the search help to the customer-specific sub-screen of the contract?

#### Take over from data of the search help to the screen



If you only want to move the contract number to the screen, you can assign the search help ZGENERALCONTRACT to the screen field. For search help, which is assigned to a screen field, the value of the first field can be moved to the screen field.

If more than one field value needs to move to screen fields, e.g., the values of the contract title, assigning the search help to a screen field doesn't work.

In the example, the selected general contract number, the general contract title, and the general contract keeper shall move the search help to the corresponding fields on the screen. Therefore, the search help has to be assigned to the field GENERAL\_CONTACT\_NUMBER of the structure CI\_EVER.

In the first step, we have to define the search help with the help of **TRANSACTION SE11** and the properties as shown in Figure 2.82.

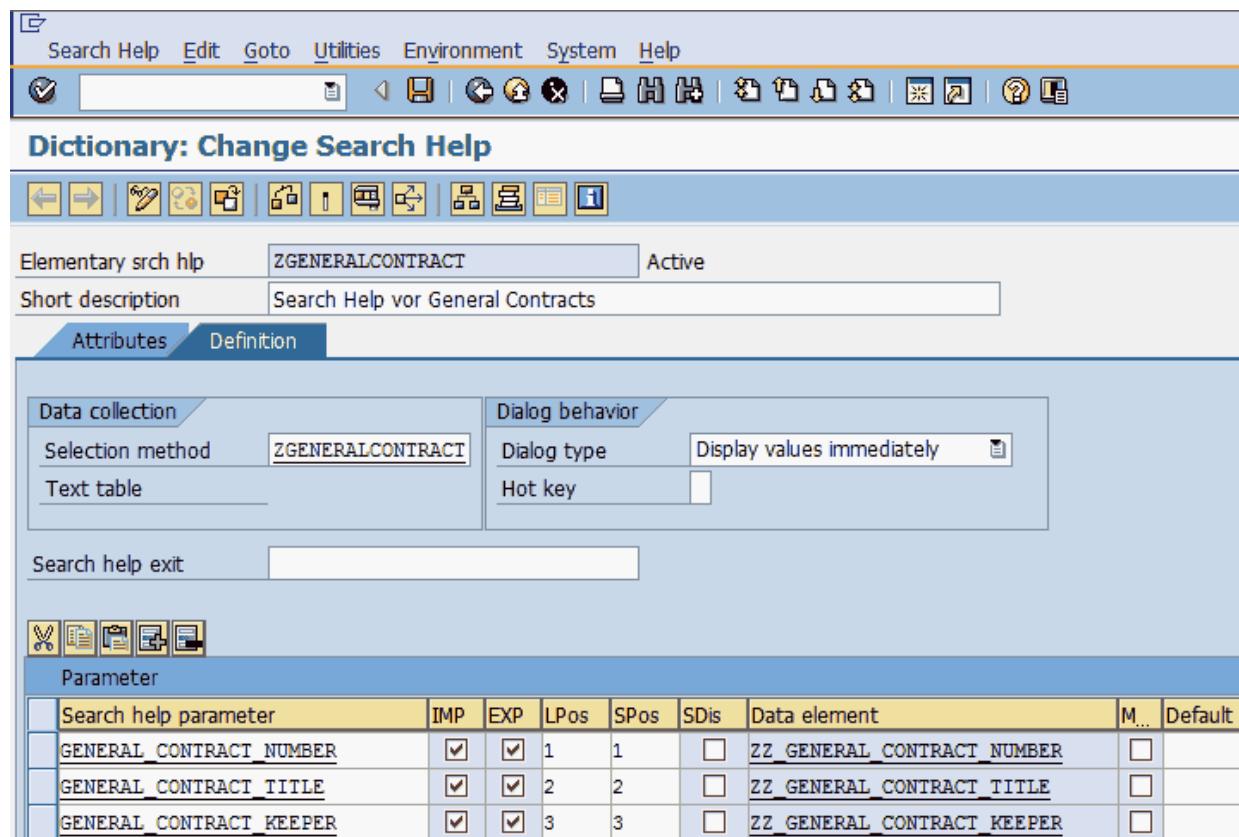


Figure 2.82: Search help for ZGENERALCONTRACT

The assignment of the search help to the field **GENERAL\_CONTRACT\_NUMBER** in **STRUCTURE CI\_EVER** is defined in SE11

tab **ENTRY HELP/CHECK** in the **SEARCH HELP** column. In the row with **GENERAL\_CONTRACT\_NUMBER**, the search help ZGENERALCONTRACT has to be inserted. Confirm the input by clicking the  icon (see Figure 2.83).

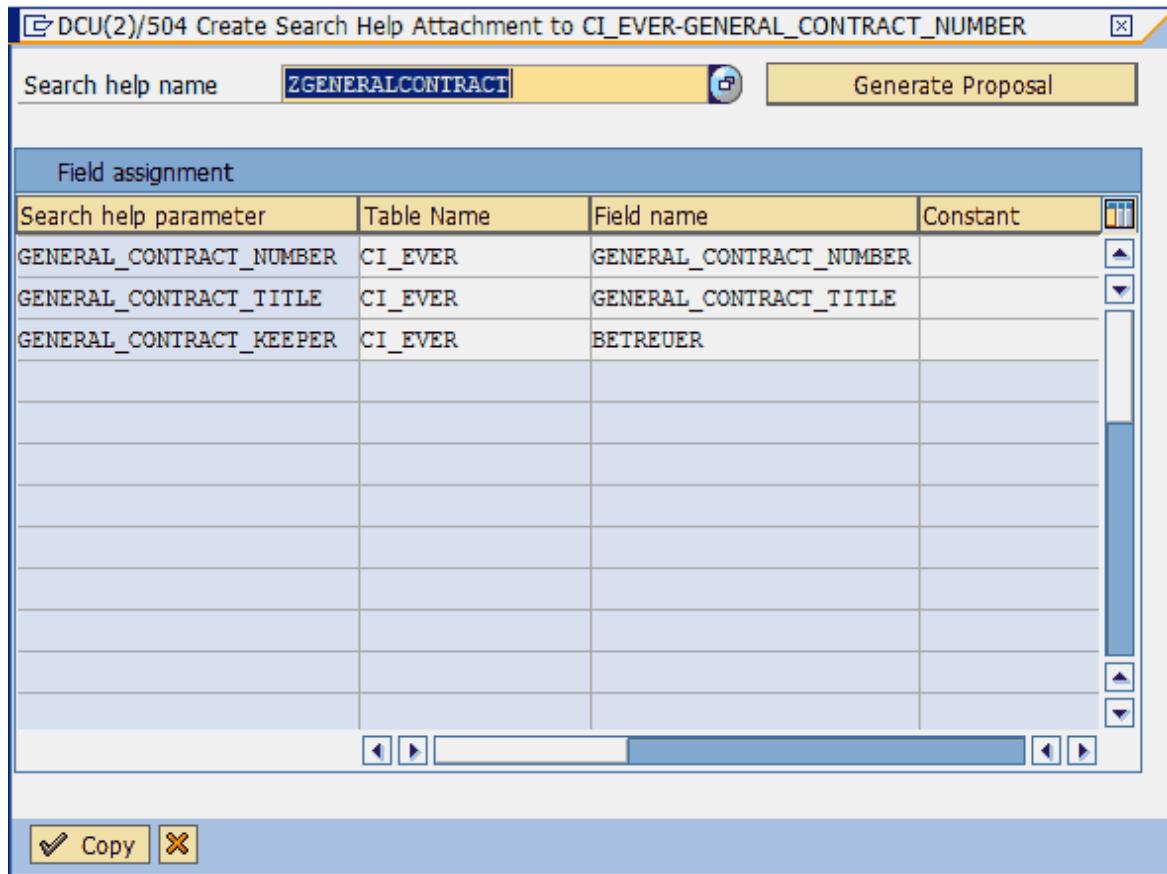


Figure 2.83: Assignment of the search help to a data element of a database field

The SAP system suggests an account for the corresponding fields and assigns the structure fields and fields of the table ZGENERALCONTRACT.

This assignment is in order, so confirm the suggestion by clicking the  **Copy** button. Activate the structure CI\_EVER. Don't forget, the activation of this structure can take a while depending on the number of dependent structures.

Now hit the “F4” help key while on the customer sub-screen to see the relevant general information for a contract (see Figure 2.84).

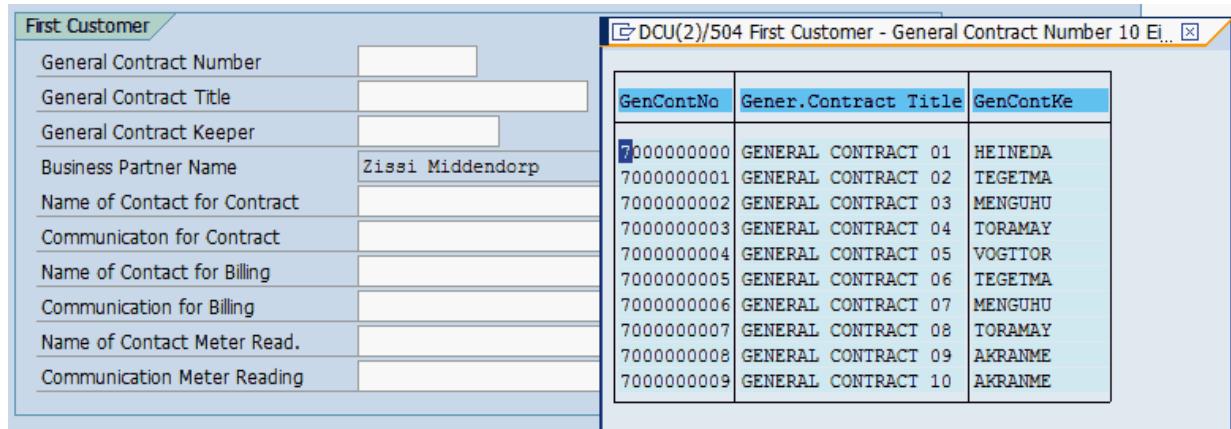


Figure 2.84: Search help for ZGENERALCONTRACT on the customer sub-screen to maintain contracts

Figure 2.85 shows that the values of the fields **GENERAL\_CONTRACT\_NUMBER**, **GENERAL\_CONTRACT\_TITLE**, and **GENERAL\_CONTRACT\_KEEPER** of the selected records are moved to the fields of the customer sub-screen.

Figure 2.85: Takeover of values from search help to fields in a sub-screen

Additional programming is necessary to store the selected values in the database. These changes are explained in the *Practical Guide to SAP ABAP: Part 2*.

## 2.13 Lock objects and lock modules

With **TRANSACTION** SE11 you can create, change, and delete *lock objects*. Lock objects are used to prevent simultaneous processing of a single data set by two users.

A locked object can be displayed by a subsequent user but not be changed or deleted. If the subsequent user tries to make changes, he will receive a message saying that the record is locked by another user.

Table 2.1 shows different available locks of records in SAP.

Read Lock	<i>S</i> (Shared)	Several users can display locked records at the same time (via transactions). Requests for other read locks are accepted, even if the requests are from other users.
Write Lock	<i>E</i> (Exclusive)	A write lock of another user on an object that has a read lock is rejected; also, any other enhanced write locks (X) are rejected.
Enhanced Write Lock	<i>X</i> (eXclusive non-cumulative)	Write locks can be requested several times from the same transaction and are released successively. An enhanced write lock can only be requested once. Any other request for a lock is rejected.
Optimistic Lock	<i>O</i> (Optimistic)	Optimistic locks behave like read locks but can be converted into write locks.

Table 2.1: Types of record locks

Locked records are stored in a *lock table*. This table is not a database table but a table in the main memory of the SAP central instance. You can display locked objects with **TRANSACTION** SM12 (for example see Figure 2.86).

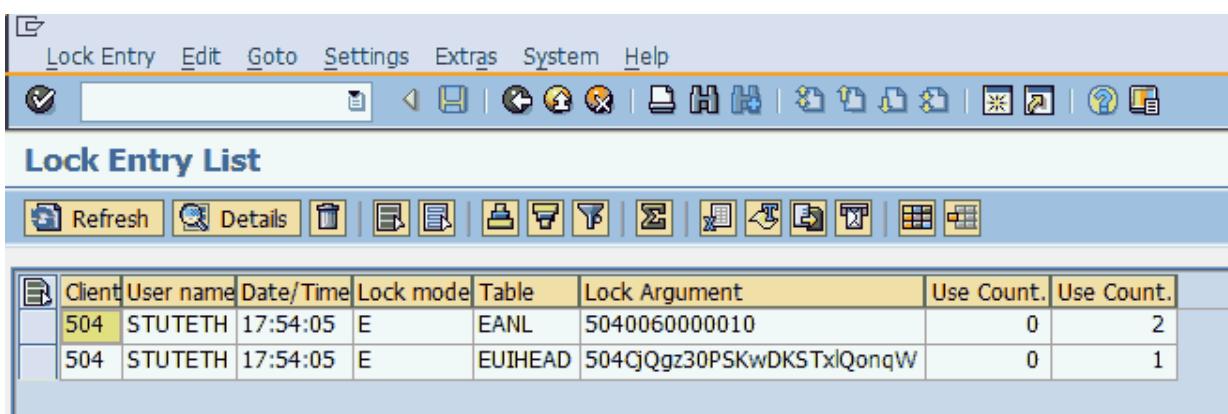


Figure 2.86: Display of locked records with transaction SM12

To delete records from this table manually, mark the record in the first column and use the pull-down **LOCK ENTRY • DELETE**.

#### Check first, then delete



An unwary delete of lock entries may cause data inconsistencies.

This is the program procedure to follow when using locking objects:

1. Select the record to be changed
2. Request a write lock with an *ENQUEUE module*
3. Change the selected record
4. Remove the lock with a *DEQUEUE module*

#### Delete unneeded locks of a program



The developer has to pay attention when requested locks are removed and when the lock is no longer used. Unlocking generally is performed if a change to the database failed. If locks are not deleted, they can cause a memory overflow. The result is that the SAP system “stands,” that means no user can work with the system.

The size of the lock table is a system parameter that is set by the SAP Basis Team.

To change access to a database table, the program must lock the database object. To determine which lock object is to be used, use the “F4” help key in **TRANSACTION** SE11.

Alternatively, edit one corresponding object by using a transaction. For example, use **TRANSACTION** SM12 in another SAP mode to display the locked object.

Let's take the example mentioned in [Chapter 1](#) to change the meter reading unit of an installation time slice. In this case, we are looking for the lock object of an SAP IS-U installation. Using **TRANSACTION** ES31 (change installation), the corresponding installation is locked for write access by another user (see Figure 2.87).

The screenshot shows the SAP interface for Transaction SM12, titled "DCU(2)/504 Lock Entry Details". The window is divided into three main sections: "User Name and Lock Argument", "Lock Owner ID", and "Technical Attributes".

**User Name and Lock Argument:**

Client	504
User name	STUTETH
Table name	EANL
Lock argument	5040060000010

**Lock Owner ID:**

Lock Owner	20150210175405230000010500R3DCUCI.....
Host name	R3DCUCI.....
SAP System Number	0
Work Process	005
Date	10.02.2015 17:54:05 230000

**Technical Attributes:**

Backup flag	
Transaction Code	ES32
Lock Object Name	E_EANL
Cumulative Counter	0   2

At the bottom of the window are three buttons: a checked checkbox, a refresh button, and a close button.

Figure 2.87: Display of lock records in transaction SM12

View the **TECHNICAL ATTRIBUTES** of **TRANSACTION SM12** for this locked record. The locked object (also called **ENQUEUE-OBJECT**) is named with “E\_EANL.”

You can display this locked object in the Data Dictionary (SE11) (see Figure 2.88).

### Don't change standard SAP lock objects



Do not change standard SAP lock objects. If you do, the lock in standard SAP transactions will no longer work.

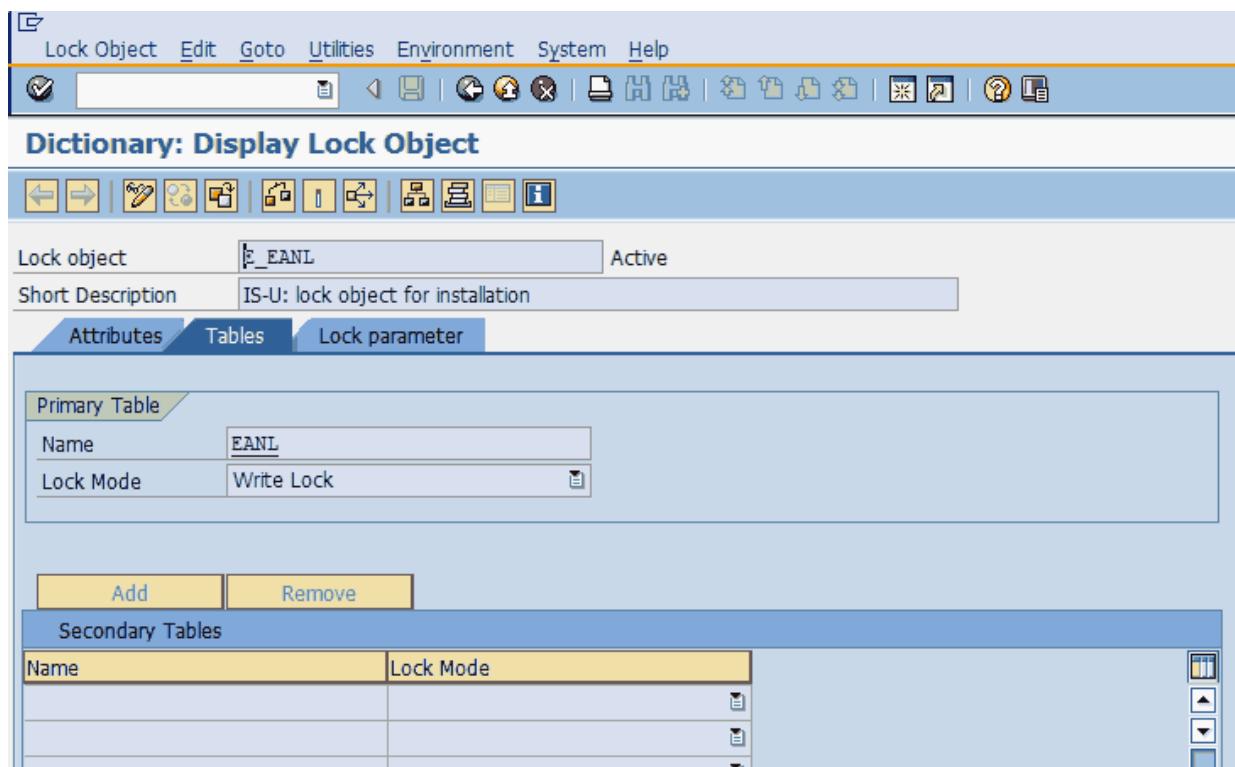


Figure 2.88: The lock object in the Data Dictionary

To request locks for installation, you have to use the **LOCK PARAMETER** shown in Figure 2.89.

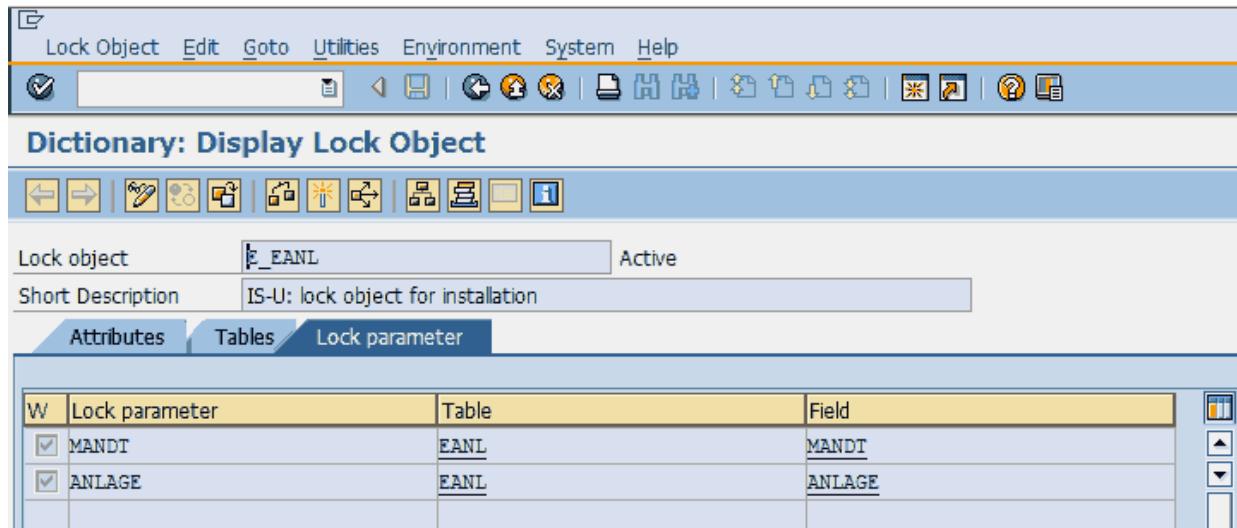


Figure 2.89: Lock parameters of a lock object

With the pull-down menu **Go to • LOCK MODULES**, you get the necessary information for using the correct lock modules.

For SAP IS-U installations, the lock module DEQUEUE\_E\_EANL and the unlock module ENQUEUE\_E\_EANL are displayed.

The example of the lock object E\_EANL for the SAP IS-U installations shows the naming convention for this function module.

- ▶ DEQUEUE\_<Name of the lock object> for locking database objects
- ▶ ENQUEUE\_<Name of the lock object> for unlocking database objects

During development, you can use the *pattern generator* to code the calling of lock modules (see Figure 2.90).

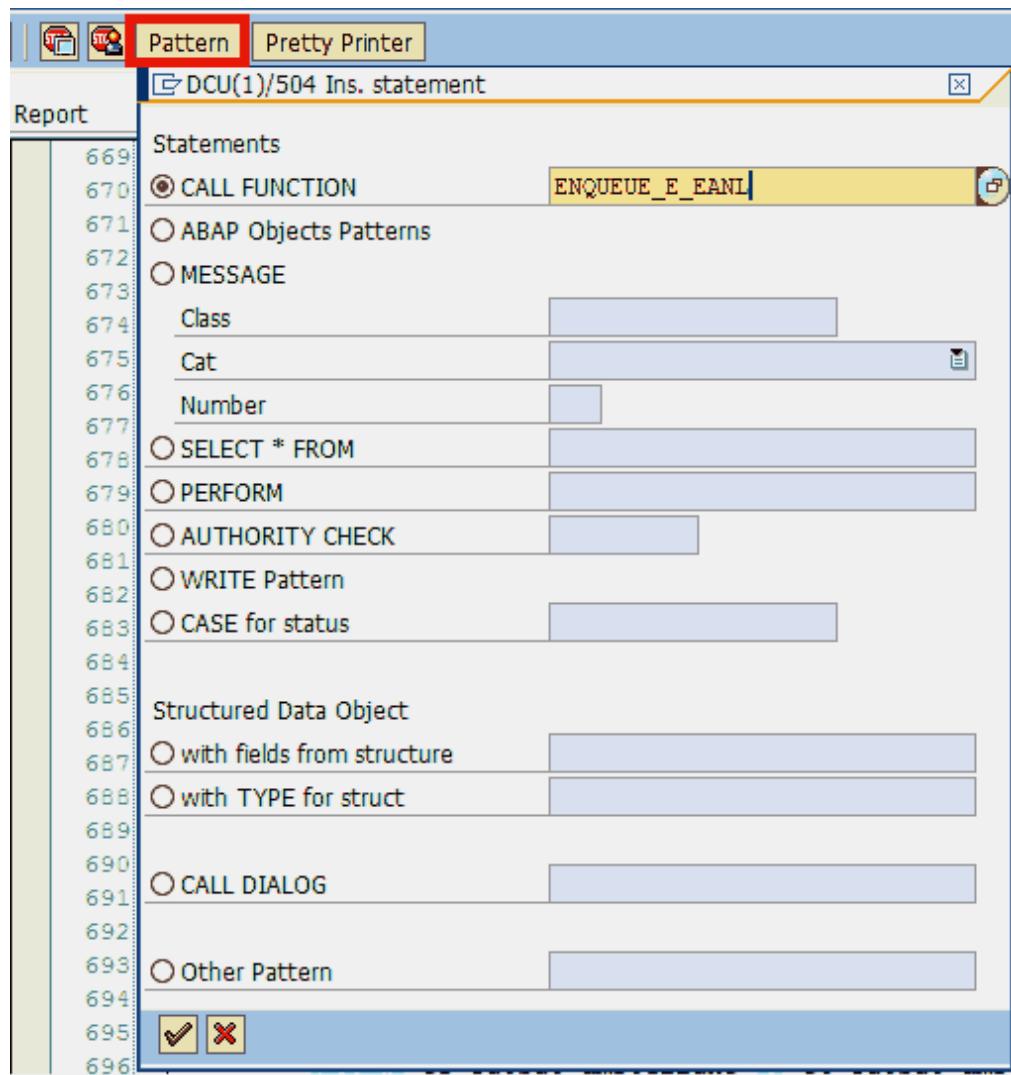


Figure 2.90: Pattern generator in ABAP editor for calling lock modules

The pattern generator provides source code, which has to be completed by the developer. See the coding for locking an object in our example program ZCU\_FIRST\_CUSTOMER\_ABLEINH shown in Figure 2.91.

```

*      Locking the Installation
CALL FUNCTION 'ENQUEUE_E_EANL'
  EXPORTING
    mode_eanl      = 'E'
    mandt         = sy-mandt
    anlage        = gs_output-anlage
    x_anlage      = ''
    _scope         = '2'
    _wait          = ''
    _collect       = ''
  EXCEPTIONS
    foreign_lock   = 1
    system_failure = 2
    OTHERS          = 3.

  IF sy-subrc = 1.
    gs_output_umstellung-anmerkung =
    'Error: Installation actually using from another user'.
    APPEND gs_output_umstellung TO gt_output_umstellung.
    CONTINUE.
  ELSEIF sy-subrc > 1.
    gs_output_umstellung-anmerkung =
    'Error: Installation Installation could not be locked for Update'.
    APPEND gs_output_umstellung TO gt_output_umstellung.
    CONTINUE.
  ENDIF.

```

Figure 2.91: Source code for calling the lock module

If locking an object is not possible, the program must output a message to the user.

After the final installation record update, the unlock module has to be called. Similar to coding the calling of the lock module, the developer can use the pattern generator for coding the call of the unlock module (see Figure 2.92). No error handling message is needed to unlock an object.

```

Unlock the Installation
CALL FUNCTION 'DEQUEUE_E_EANL'
  EXPORTING
    mode_eanl = 'E'
    mandt     = sy-mandt
    anlage    = gs_output-anlage.

```

Figure 2.92: Source code for calling the de-queue function module

## 2.14 Database utility

Use the *database utility* if you add new fields, delete fields, or change the length of fields in an existing database table. Its purpose is to adapt the table records in a modified structure.

### Example



The domain ZZ\_GENERAL\_CONTRACT\_KEEPER for the responsible person of a general contract in table EVER (contract) was changed from a character field with the length of 20 to a character field with the length of 12.

By activating these changes in our example, you get an error message. The activation protocol is shown in Figure 2.93.

A screenshot of the SAP Activation Protocol interface. The title bar shows 'STUTETH20150212144951'. The main area displays a list of messages:

- TABL EUE\_UI\_NODE was adjusted
- ⚠ TABL EVER is inconsistent in active version  
Check table EVER (STUTETH/12.02.15/14:51)
- ⓘ Enhancement category 3 possible, but include or subtly. not yet classified
- ⓘ Index EVER-ZAN completely contains the fields of index ANL
- ⓘ Index EVER-ZTR completely contains the fields of index TVR
- EVER-ABSZYK: Table EVERD in search help attachment differs from table of search field EVER-TXJCD: Table EVERD in search help attachment differs from table of search field
- Field GENERAL\_CONTRACT\_KEEPER: Length change  
 ALTER TABLE is not possible
- ⚠ Structure change at field level (convert table EVER)  
Check on table EVER resulted in errors

Figure 2.93: Error message: Conversion of table EVER is necessary

There are three warnings (yellow exclamation mark), one error message (first red triangle exclamation mark), and one necessary action to repair the error message (second red triangle exclamation mark).

Convert the table EVER, which currently has the table status as “partly active.” The conversion of the table is necessary to adjust the existing

records to the changed database structure. This action is made using the database utility.

To do this, you call the table EVER with **TRANSACTION SE11** in display mode. Then choose the menu path **UTILITIES • DATABASE-OBJECT • DATABASE UTILITY**. You see the screen as displayed in Figure 2.94.

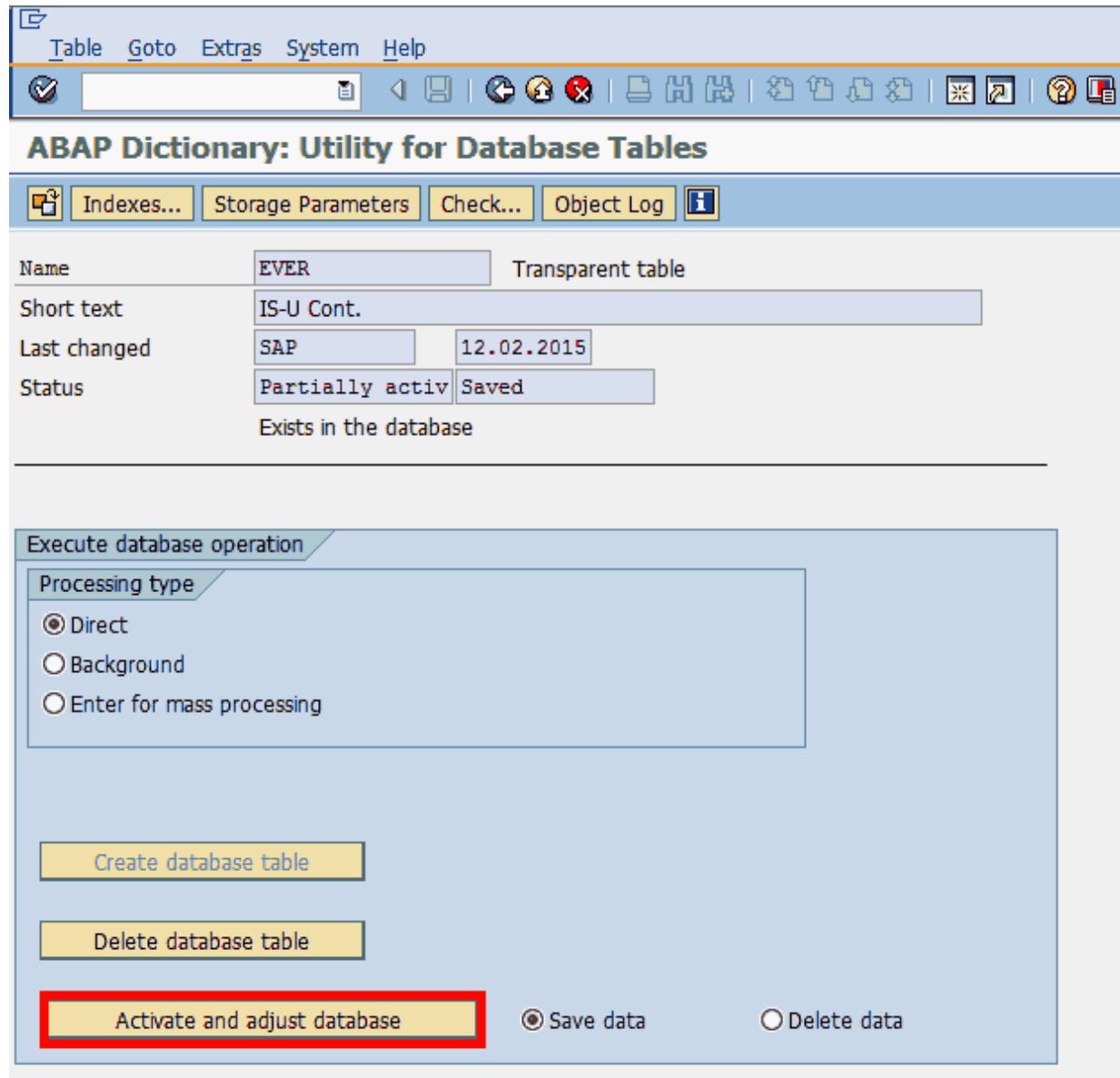


Figure 2.94: Activate and adjust database changes

For adjusting the table EVER, click on the **Activate and adjust database** button with the variant **SAVE DATA**. The database table is updated.

For example, Entries in the field **GENERAL\_CONTRACT\_KEEPER** have been shortened from 20 characters to 12 characters. The stored name of the responsible person for the contract “Mr. Customerfriendly” now has the name “Mr. Customer.”

Adjusting the table can take some time since each data record and all dependent tables and structures that use the table must be analyzed and modified.

Once complete, you see the message, “Request for EVER executed successfully.” Table EVER is now active. There is no need to re-activate the table.

If the field length change of **GENERAL\_CONTRACT\_KEEPER** will be activated, the adjustment of the table in the production system is made by the transport management system. No re-adjustment of the database using the database utility is necessary.

#### Be careful transporting customer-includes



You have to pay attention not to transport customer-includes during production times. Since a number of dependent structures have to be adapted, the transport can cause hard program crashes (dumps).

## 2.15 Changing table contents

In practice, there are always cases where “hard” changes of a record in the database are required, such as if a program crashed before changes had been saved and stored in the database. Such changes to the database in production systems have to be documented for later review by chartered accountants. The documentation must include the date, the time, the person, the affected table, the record, and the reason for the change. Normally, no user is authorized to perform database changes in a production system directly in the database (for reasons of traceability). For this reason, an *emergency permission* is assigned to the developer making the change and the time of the change. Using this emergency permission has to be documented.

Modifying table contents is a constant task in development and test systems. There are two options:

1. **TRANSACTION** SE16N
2. **TRANSACTION** SE11 with the help of the ABAP debugger

### 2.15.1 Changing table contents with transaction SE16N

Take the previous example about changing the meter reading unit of an installation. The meter reading unit is stored in time slices of table EANLH. Use **TRANSACTION** SE16N to display the table contents. In the selection screen of SE16N for table ENALH, select the installation number of the meter reading unit you have to change.

Before starting the selection, add the command `&sap_edit` in the command line and press the “Enter” key (see Figure 2.95).

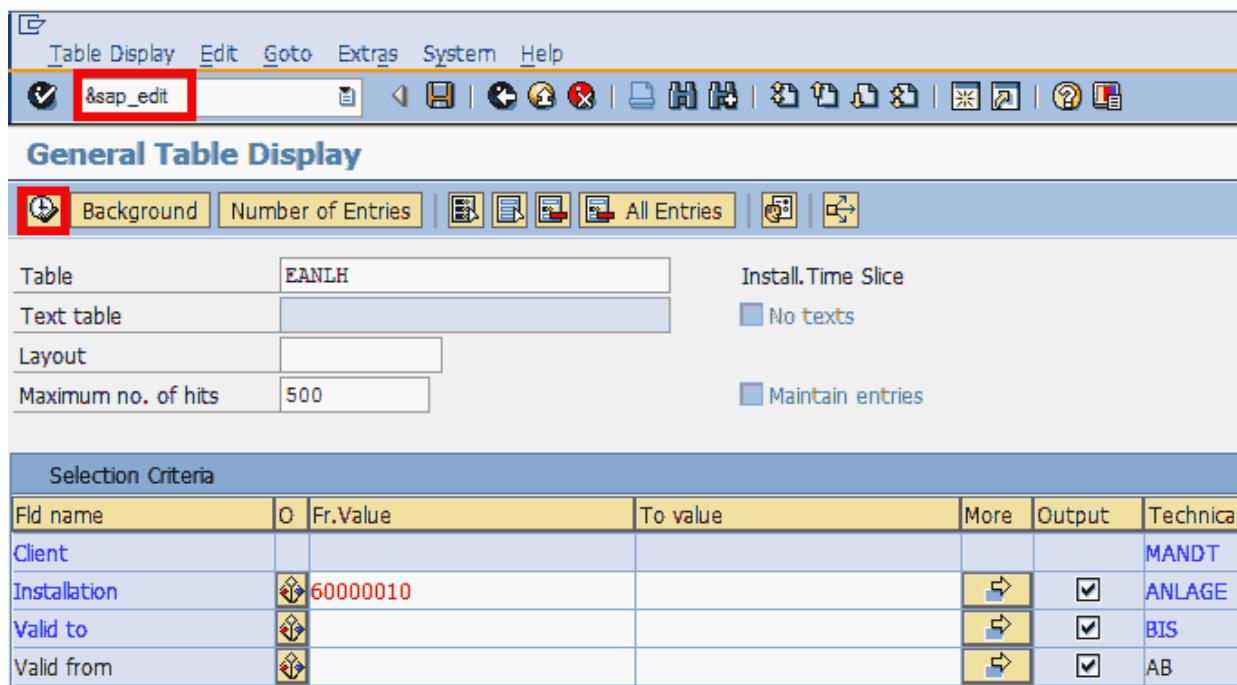


Figure 2.95: Changing table contents with transaction SE16N

The message “SAP editing function is activated” appears in the status line. Then select the record by clicking the icon. Now you can change the meter reading unit of the applicable time slice (see Figure 2.96).

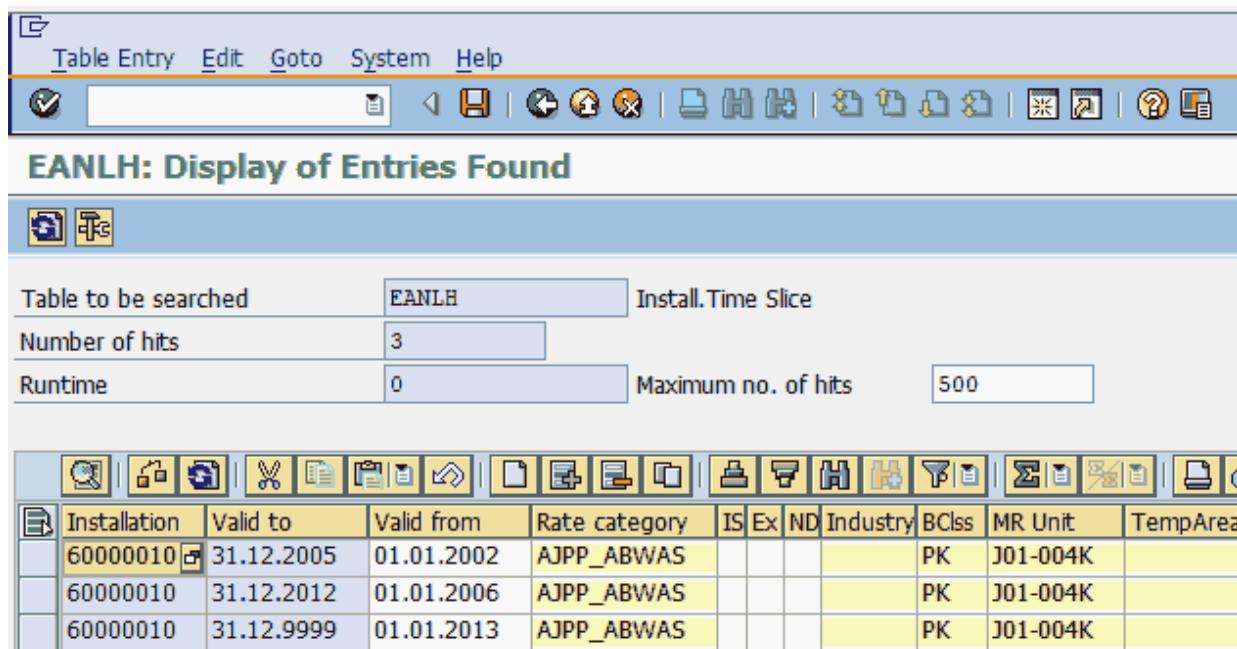


Figure 2.96: Store changes of records

Save the change by clicking the  icon.

The save is confirmed by a popup message, as shown in Figure 2.97.

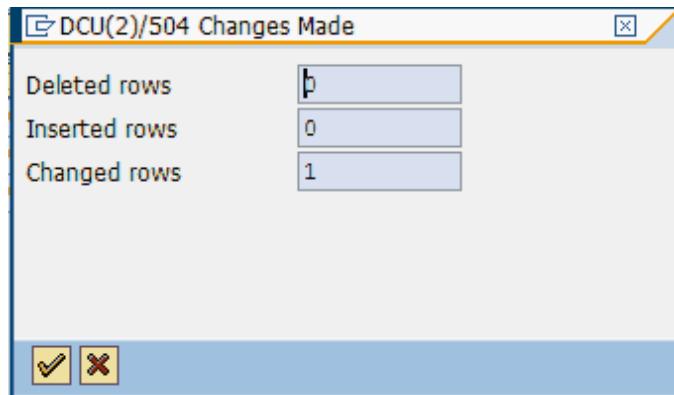


Figure 2.97: Message after saving changes to table contents

Inputs are tested against a test table. If an item has not been defined in the table, an error is reported by **TRANSACTION SE16N** (see Figure 2.98).

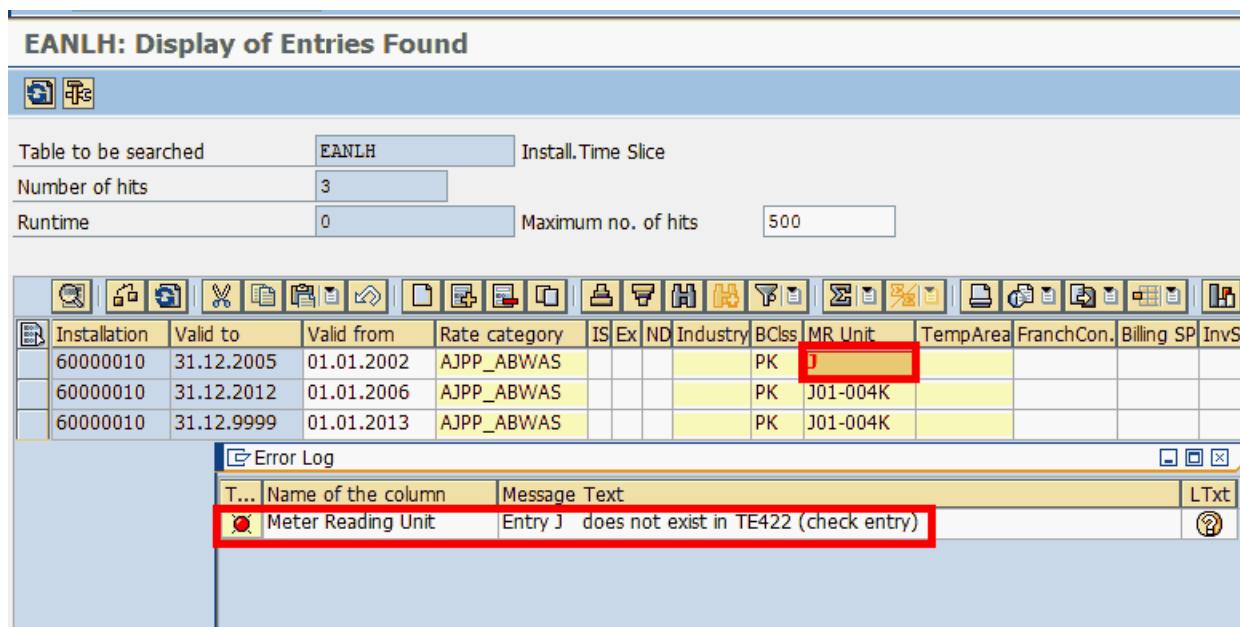


Figure 2.98: Error message from changing table content

Only after correcting the field content (e.g., by hitting the “F4” help key) can the changed record be stored in the database.

**TRANSACTION SE16N** shows the following icons (among others):



By clicking the icon, you can insert new records. By clicking the icon, you can insert a new line. By clicking the icon, you can delete a line or an existing record. And by clicking the icon, you can copy marked lines.

If you want to insert a couple of records to a database table, you can do this with the help of a spreadsheet (e.g., Microsoft Excel). Use an Excel sheet with exactly the same fields as the database table. Enter the new records in the corresponding fields of the Excel sheet. Then click the icon in SE16N and put the cursor in the first empty field. Copy the records of the Excel sheet to the database table by copy and paste. With this procedure, you can insert hundreds of records to the database table. This procedure is often used when filling a database table with data for the first time. After copy and paste, click the icon in **TRANSACTION** SE16N to store the new records.

### Change content of uneditable tables



You can use the following procedure if a table is not editable through the account properties. After entering the command `&sap_edit` by using **TRANSACTION** SE16N, start the debugger with the command `/h` in the command line before selecting any records. Set the value of the variable `GD-EDIT = X` with the help of the debugger. You can now change records in this table.

Changes made with the help of **TRANSACTION** SE16N are documented in the database tables `SE16N_CD_KEY` and `SE16N_CD_DATA`.

The database table `SE16N_CD_KEY` contains the name of the edited table (**TAB**), the changed by name (**UNAME**), the date (**SDATE**), and the time (**STIME**) of the change. The **ID**, or the key field of this table, is data type

TIMESTAMP. The use of SE16N is logged on the first line (see Figure 2.99).

The screenshot shows the SAP SE16N\_CD\_KEY table display. The title bar reads "SE16N\_CD\_KEY: Display of Entries Found". The table has columns: ID, Table, User, Start Date, Time, and Change was made for all clients. One record is displayed: ID 20.150.212.140.803,8280000, Table EANLH, User STUTETH, Start Date 12.02.2015, Time 15:08:03, and Change was made for all clients.

ID	Table	User	Start Date	Time	Change was made for all clients
20.150.212.140.803,8280000	EANLH	STUTETH	12.02.2015	15:08:03	-

Figure 2.99: A record of table SE16N\_CD\_KEY

With the field **ID**, you can see and analyze the changed records of table SE16N\_CD\_DATA (see Figure 2.100).

The screenshot shows the SAP SE16N\_CD\_DATA table display. The title bar reads "SE16N\_CD\_DATA: Display of Entries Found". The table has columns: ID, Seq. No., Change Type (D: Dele...), Table, Length, and Data Part. One record is displayed: ID 20.150.212.140.803,8280000, Seq. No. 1, Change Type M, Table EANLH, Length 3.000, and Data Part 50400600000102005123120020101AJPP\_ABWAS PK J01-004K.

ID	Seq. No.	Change Type (D: Dele...)	Table	Length	Data Part
20.150.212.140.803,8280000	1	M	EANLH	3.000	50400600000102005123120020101AJPP_ABWAS PK J01-004K

Figure 2.100: A record in table SE16N\_CD\_DATA

Protocol tables are used by chartered accountants. Deleting records in these tables is only possible with customer-specific programs.

## 2.15.2 Change data sets with the ABAP debugger

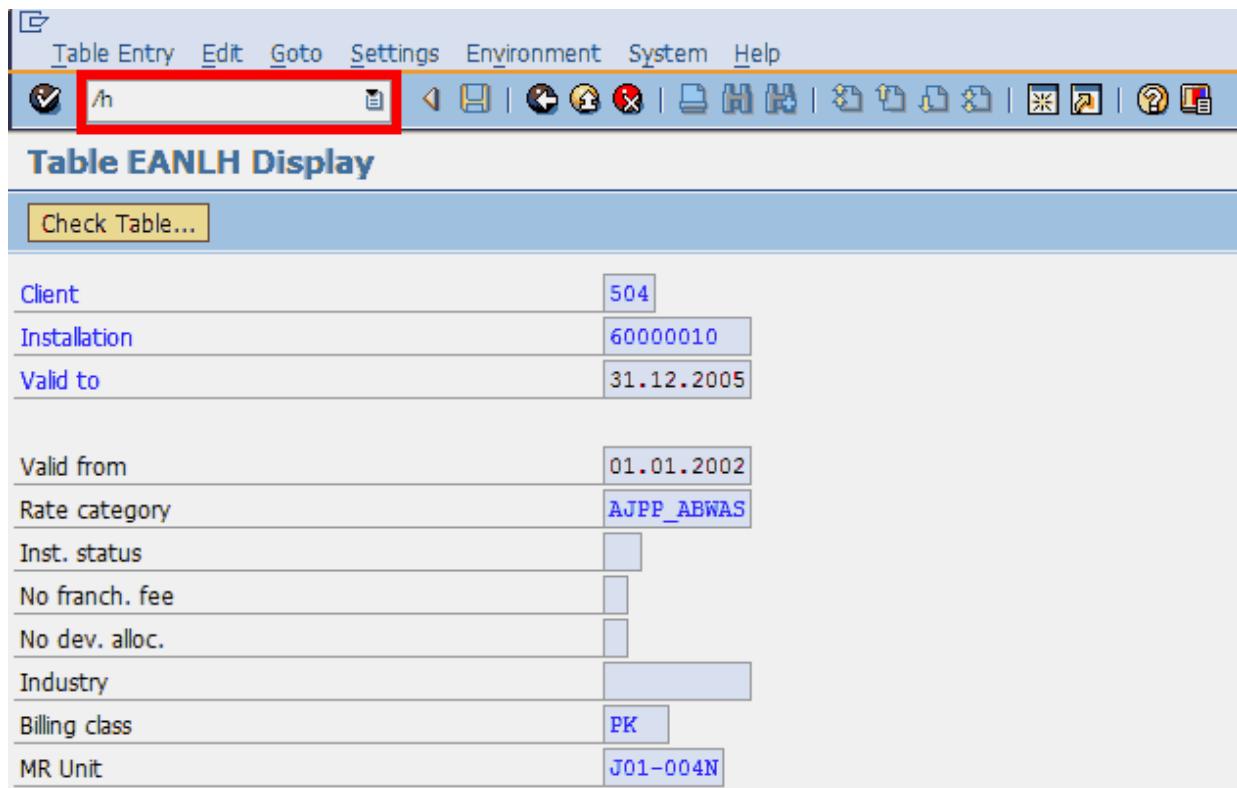
Another method to change records in a database is with the ABAP debugger ([Chapter 3](#) provides detailed information about ABAP debugger).

First, you have to display the record you want to change in the display mode of **TRANSACTION SE11** (see Figure 2.101).

	Cl.	Installat.	Valid to	Valid fr.	Rate cat.	IS	Ex	ND	Industry	BCls	MR Unit
	504	0060000010	31.12.2005	01.01.2002	AJPP_ABWAS					PK	J01-004N
	504	0060000010	31.12.2012	01.01.2006	AJPP_ABWAS					PK	J01-004K
	504	0060000010	31.12.9999	01.01.2013	AJPP_ABWAS					PK	J01-004K

Figure 2.101: Display records with transaction SE11

With a double click on a record, the detailed information of the record is displayed. In this screen (see Figure 2.102), start the debugger with the command /h in the command line and confirm this command by hitting the “Enter” key.

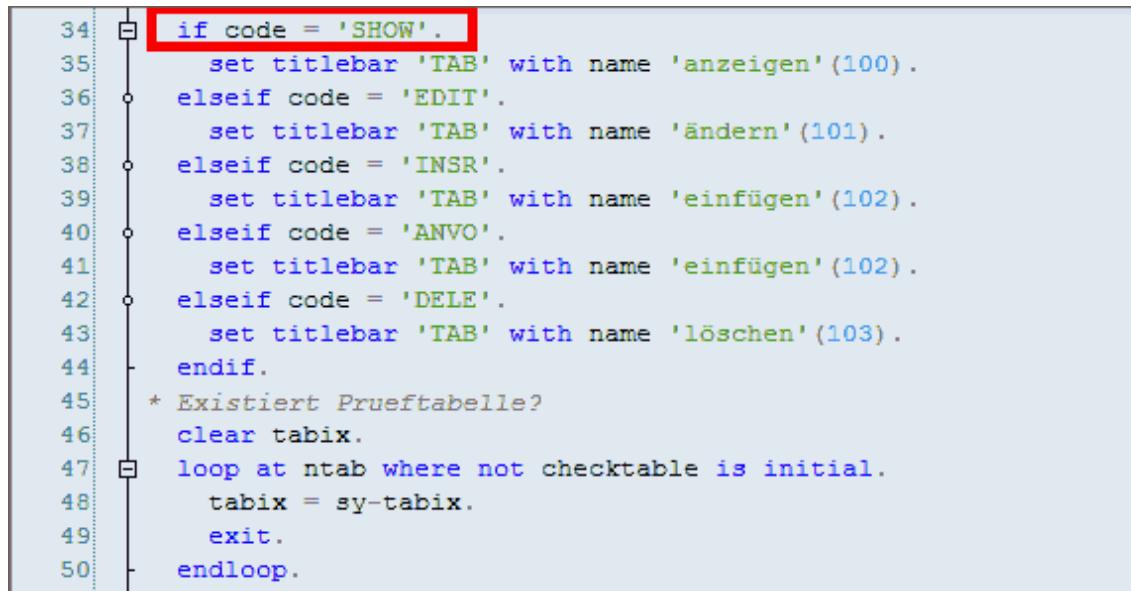


The screenshot shows the SAP SE11 transaction interface. The title bar includes 'Table Entry', 'Edit', 'Goto', 'Settings', 'Environment', 'System', and 'Help'. A toolbar below has various icons. The main area is titled 'Table EANLH Display' with a 'Check Table...' button. Below is a table with several rows:

Client	504
Installation	60000010
Valid to	31.12.2005
Valid from	01.01.2002
Rate category	AJPP_ABWAS
Inst. status	
No franch. fee	
No dev. alloc.	
Industry	
Billing class	PK
MR Unit	J01-004N

Figure 2.102: Display detailed information of a record with transaction SE11

Repeatedly hit “Enter” until the debugger starts. The result is shown in Figure 2.103.



```

34      if code = 'SHOW'.
35          set titlebar 'TAB' with name 'anzeigen'(100).
36      elseif code = 'EDIT'.
37          set titlebar 'TAB' with name 'ändern'(101).
38      elseif code = 'INSR'.
39          set titlebar 'TAB' with name 'einfügen'(102).
40      elseif code = 'ANVO'.
41          set titlebar 'TAB' with name 'einfügen'(102).
42      elseif code = 'DELETE'.
43          set titlebar 'TAB' with name 'löschen'(103).
44      endif.
45      * Existiert Prueftabelle?
46      clear tabix.
47      loop at ntab where not checktable is initial.
48          tabix = sy-tabix.
49          exit.
50      endloop.

```

Figure 2.103: Debugger after detailed display of a record with transaction SE11

With a double click on the **VARIABLE CODE**, you will be able to edit the value (see Figure 2.104).

St...	Variable	Va...	Val.	Ch...	Hexadecimal Value
	CODE		SHOW		3484F57

Figure 2.104: Editing the variable CODE in the debugger

With a mouse click on the icon, you can change the value of the variable CODE. The following values can be entered instead of the value SHOW (display):

- ▶ DELE to delete the displayed record
- ▶ ANVO to copy the displayed record
- ▶ EDIT to change the field value of a record
- ▶ INSR to insert a new record

You have to confirm the change of the value by hitting the “Enter” key. The result is shown in Figure 2.105.

St...	Variable	Va...	Val.	Ch...	Hexadecimal Value
	CODE		EDIT		45444954

Figure 2.105: Changed value of the variable CODE in the debugger

Finalize the debugger process by clicking on the icon. See Figure 2.106.

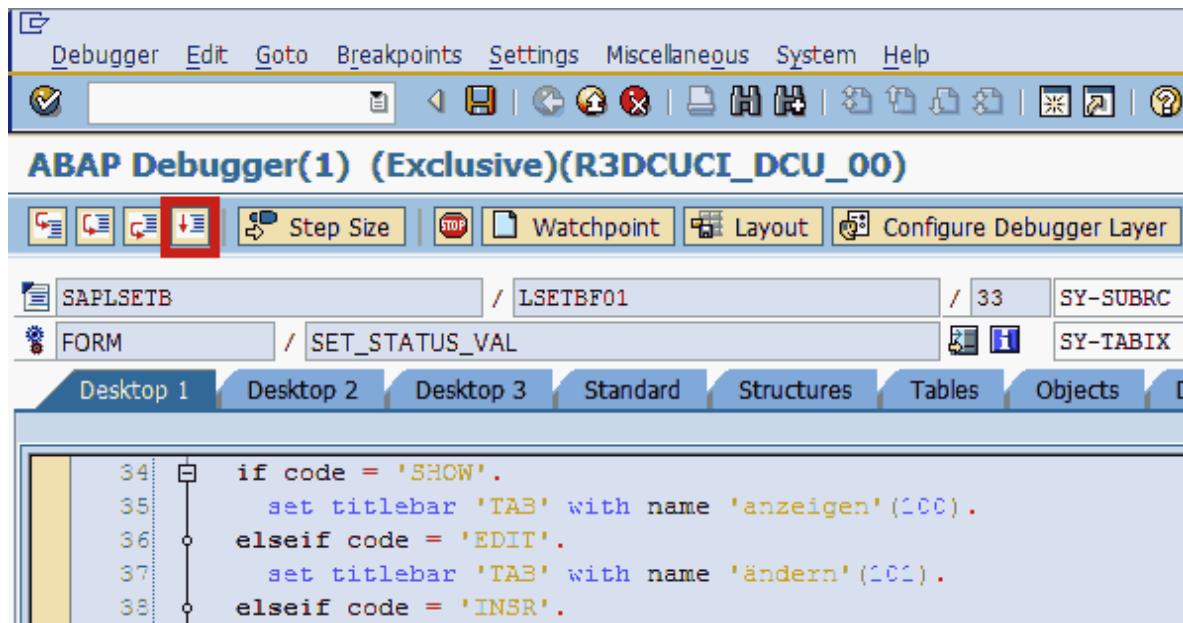


Figure 2.106: Finalizing the debugger

Now you can edit the record (see Figure 2.107).

Similar to changing the record with **TRANSACTION SE16N**, you have to save this change by clicking the icon. If there is a check table defined to a changed field, the input will be verified by that defined check table.

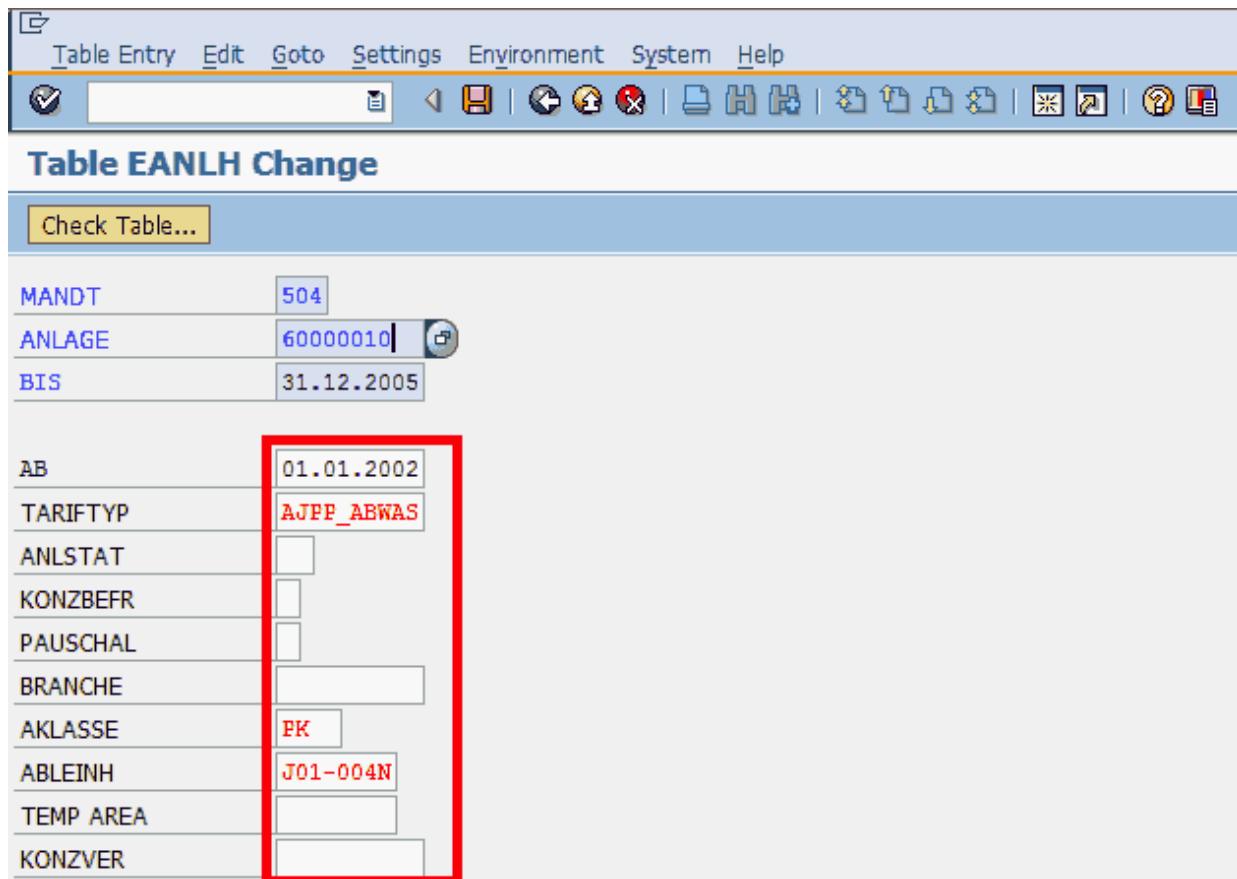


Figure 2.107: White fields can be changed

Clicking on the icon to save the change. The SAP program sends the success message “Database record successfully created” to the status line.

## 2.16 Who has changed the table content?

Often, when content in a table has been changed, it isn't easy to determine why the change was made. This is annoying, especially with customized settings, because such changes can manipulate basic functions of the SAP system. One reason for changes can include different employees of an IT service provider working on the same environment without knowing each other. Normally, for customized entries, the transport management system moves the entries into other systems. In these instances, you can find the answer with the help of the *transport organizer tool*.

To get the answer, you have to use **TRANSACTION SE09** or **SE10** (transport organizer). You call the transport organizer tool by clicking on the **Tool** icon (see Figure 2.108).

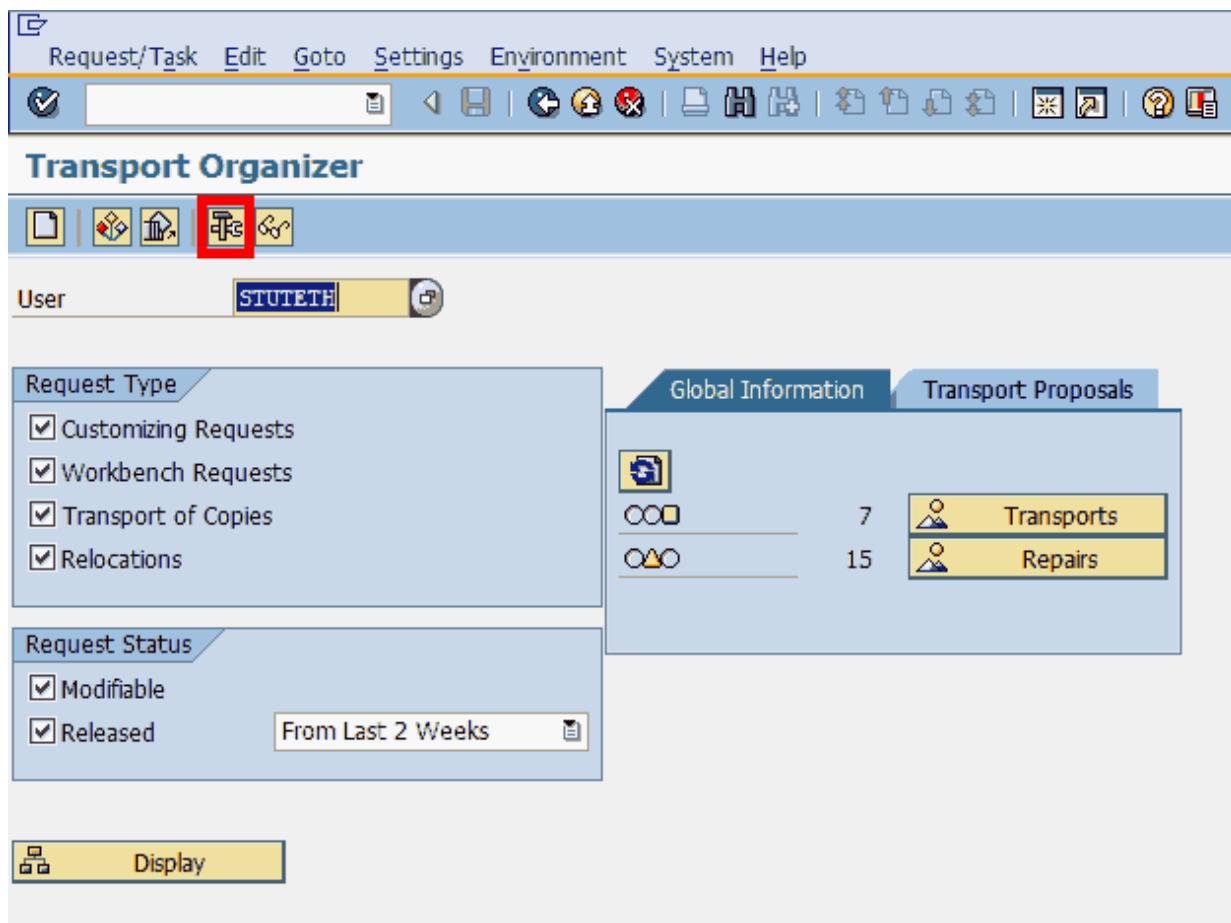


Figure 2.108: Transport organizer tool

To search for objects within requests or tasks in the transport organizer screen, click on the line with the  icon and confirm this choice by clicking the  icon (see Figure 2.109).

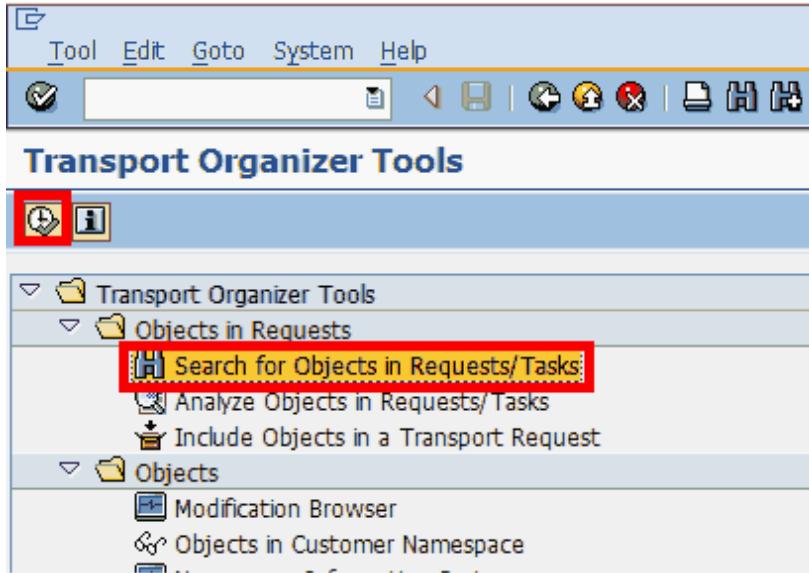


Figure 2.109: Search for an object in requests/tasks

In the next screen, you can search for requests and tasks that have transported table contents. Use the **OBJECT R3TR TABU**. In the **SELECTION** field, type the name of the customizing table. In this example, we are looking for requests/tasks including records of the table Kontoklasse (see Figure 2.110). Start the selection by clicking the **EXECUTE**  icon.

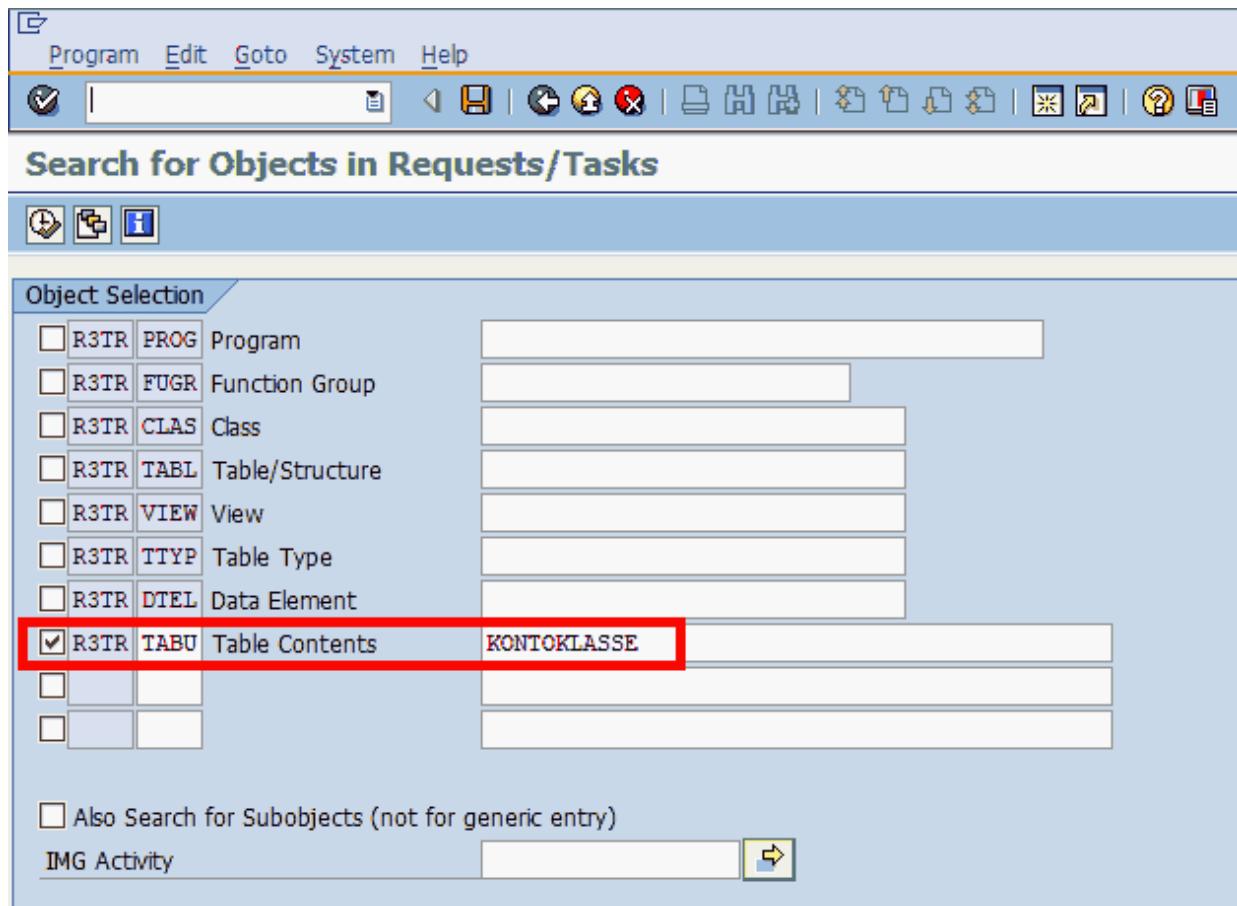


Figure 2.110: Search for requests/tasks with table contents

The result is a list of requests and tasks that include table contents of the selected table Kontoklasse. See Figure 2.111.

T01K911746	Stammdaten allgemein	STUTETH	20.06.2001	Customizing	Released
T01K913315	Änderung Kontoklassen	STUTETH	20.06.2001	Customizing	Released
T01K914837	Grundcustomizing Sparten und Nummerkreise	STUTETH	20.06.2001	Customizing	Released
T01K920418	Kontoklassen	STUTETH	20.11.2001	Customizing	Released
T01K920723	Partnerarten/Kontoklassen	STUTETH	20.11.2001	Customizing	Released
T01K921611	Ableseinheiten Ahlen	STUTETH	04.01.2002	Customizing	Released
T01K925705	Customizing-Aufträge Vogt, Veltiy, Emamjomeh, Bauer	STUTETH	12.02.2002	Customizing	Released
T01K956229	T508: Spannungsebene in Anlage	STUTETH	24.05.2004	Customizing	Released
T01K996207	isu Auftr.3020 071005 Kontenklasse NN_E	STUTETH	13.02.2008	Customizing	Released
T01K9A0ORH	INVOIC / REMADV SAMMLER Produktivsetzung gesamt	STUTETH	26.02.2008	Transport of copies	Released
T01KT00416	Mandantenexport Mandantenabh. Objekte	STUTETH	29.09.2001	Client Transport	Released
T01KT00526	Mandantenexport Mandantenabh. Objekte	STUTETH	05.11.2001	Client Transport	Released

Figure 2.111: Display of results by searching changes to table contents

To get detailed information about the objects of a transport, double click on the **TRANSPORT** icon

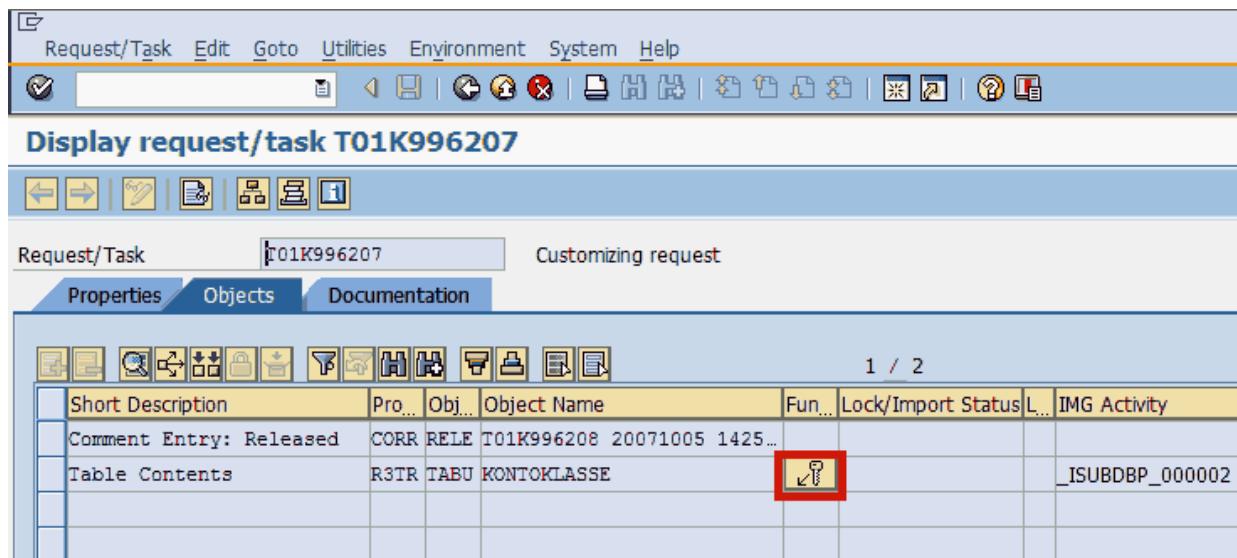


Figure 2.112: Objects of a request

By clicking on the icon, you will see which table contents are included in the transport (see Figure 2.113).

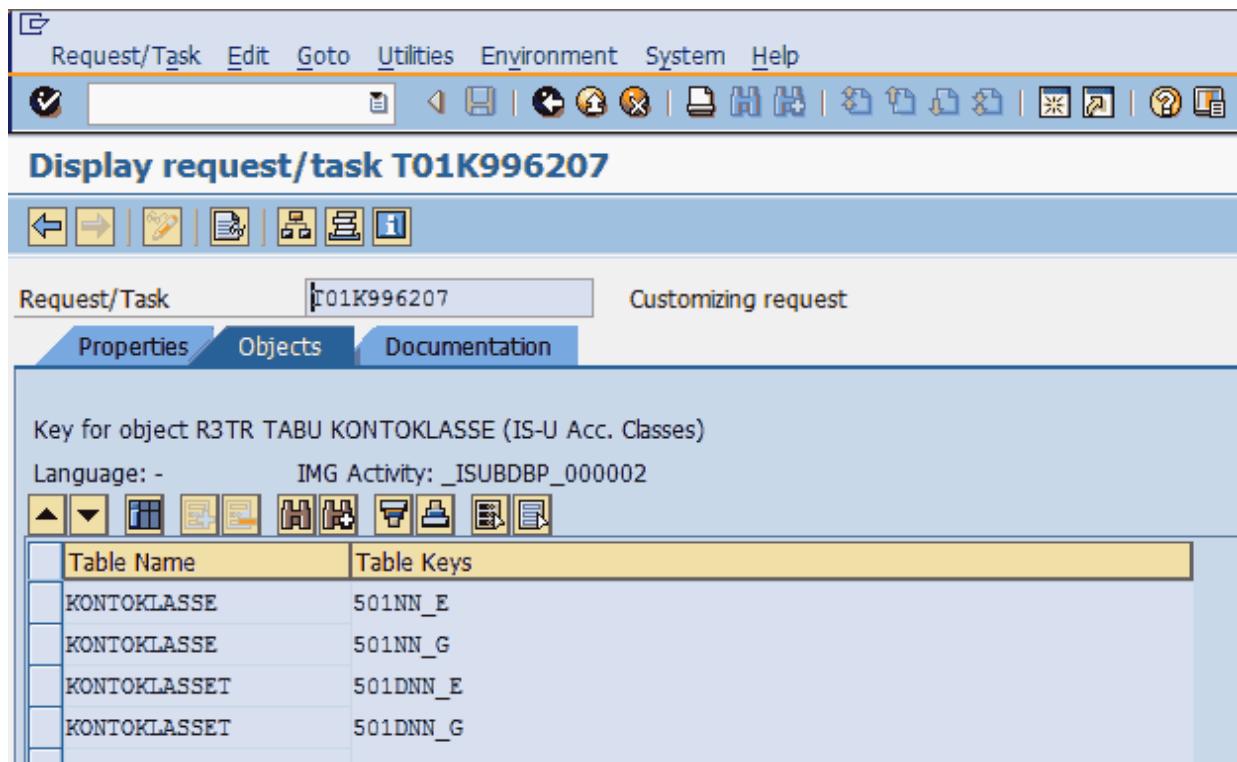


Figure 2.113: Table contents of a transport

In this case, the transport included two records from table KONTOKLASSE (account class) and two from KONTOKLASSET (text for the account class).

To search changes of other table types, e.g., application tables, the query is more complicated. Here are the different possibilities:

- ▶ Changes made with **TRANSACTION** SE16N can be analyzed with the help of the tables SE16N\_CD\_KEY and SE16N\_CD\_DATA
- ▶ If the logging of changed table content has been switched on in the system, you can analyze database access by using the table DBTABLOG. (If the logging has not been switched on in the system, you are not able to analyze the database log with this method.)
- ▶ Certain changes to SAP objects are stored as change documents (see [Section 4.19](#)); change documents are stored in the tables CDHDR (header information of a change document) and CDPOS (detailed information of a change document)

# 3 Proper debugging

SAP provided a new debugger for developers some time ago. The debugger has become a powerful tool through innovation. However, the developer uses only a fraction of the essentials of this tool. This chapter describes the methods required for the ABAP developer who deals with the debugger on a daily basis.

First, I'll show the different structures of both debuggers. Then, I'll explain how to start and end a debugger session and how to move between both debuggers.

To explain how to use the debugger, we'll use an example to search a user exit, which is used when changing a contract with **TRANSACTION** ES21 (change contract).

### 3.1 Classical and new ABAP debugger

Start the classical debugger by inputting a contract number. Next, type /h on the command line and then click the icon to select the contract with the given contract number.

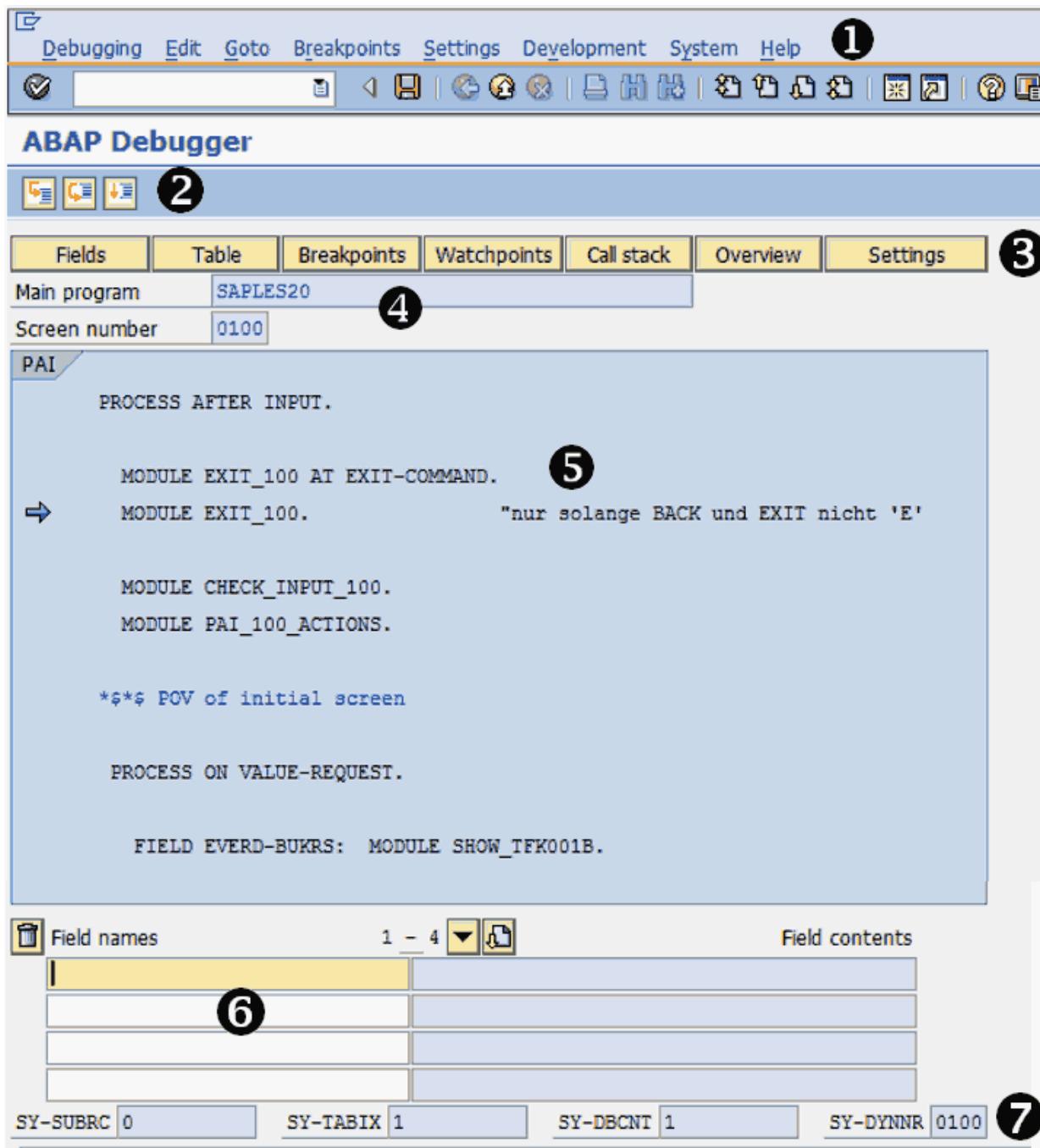


Figure 3.1: The classical debugger

The classical debugger has the following screen sections (see Figure 3.1):

- ① Pull-down menu
- ② Icon strip
- ③ Button strip
- ④ Description of the displayed program
- ⑤ Source code with current position of the program (blue arrow)
- ⑥ Variable section where field values are displayed
- ⑦ Line for system variables (e.g., SY-SUBRC)

You can change to the new debugger, display system sections (e.g., variables in ABAP memory), start the system debugger, or create breakpoints and watchpoints with a pull-down menu.

The options within the **icon strip**  allow you to choose the type of forward movement in the program run. From left to right, the displayed icons have the following meanings:



**SINGLE STEP**: the next command will be performed.



**EXECUTE**: if the cursor stays in front of a processing block (e.g., form routine, function module), the processing block will be executed.



**RETURN**: the current processing block will be processed.



**CONTINUE**: the program will be processed until the next breakpoint; if there are no further breakpoints, the whole program will be processed and the debugger will be closed.

The **button strip** allows the display of variables and their values, tables and their values, breakpoints, watchpoints, calls (until the program calls the processing blocks), and overviews (all processing blocks of the current

program). The display of variables changes depending on the selected display section.

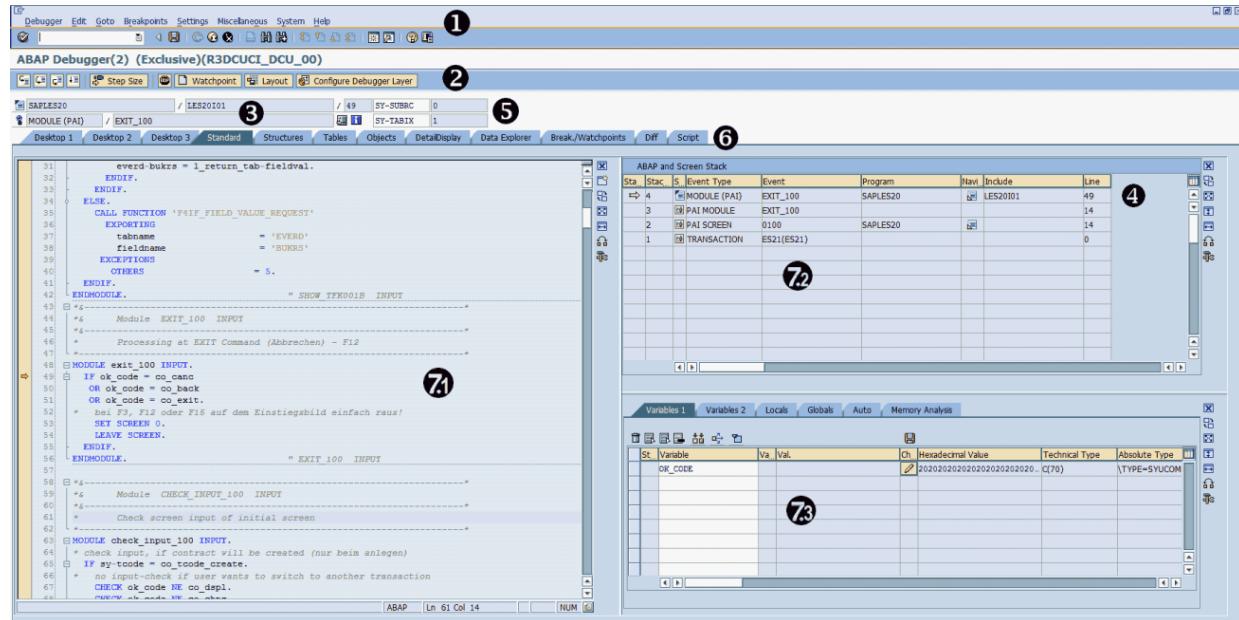


Figure 3.2: The new debugger

In contrast, the new debugger shows a significantly different layout with the following screen components (see Figure 3.2):

- ①** Pull-down menu
- ②** Button strip
- ③** Name of the displayed program
- ④** Display of the current processing block
- ⑤** System variables (default: SY-SUBRC and SY-TABIX)
- ⑥** Selection of 12 different tabs, each of them allows another view on the current status of the processed program
- ⑦** Standard view with three-screen sections

Left hand: the source code **7.1**

On the top right: the ABAP stack **7.2**

On the right below: different tabs for the display of program variables **7.3**

The new debugger provides a cleaner image of programs and the related statuses compared to the classic. Because I prefer the new debugger, I'll

briefly explain the main and often-needed tools of this new debugger. I will also show how the debugger is called in special situations.

## 3.2 Start, stop, and change debugger

You can start the new debugger via two routes:

- ▶ You call a program and start the debugger with the command /h in the command line.
- ▶ You can set a breakpoint at the place in the program where the debugger should stop the program runs.

By default, the new debugger should start. If it doesn't, make corrections in **TRANSACTION SE80**.

In **TRANSACTION SE80**, use the pull-down menu **UTILITIES • SETTINGS**. Click on the tab **ABAP EDITOR** and **DEBUGGING**, and then choose the “New Debugger” (see Figure 3.3).

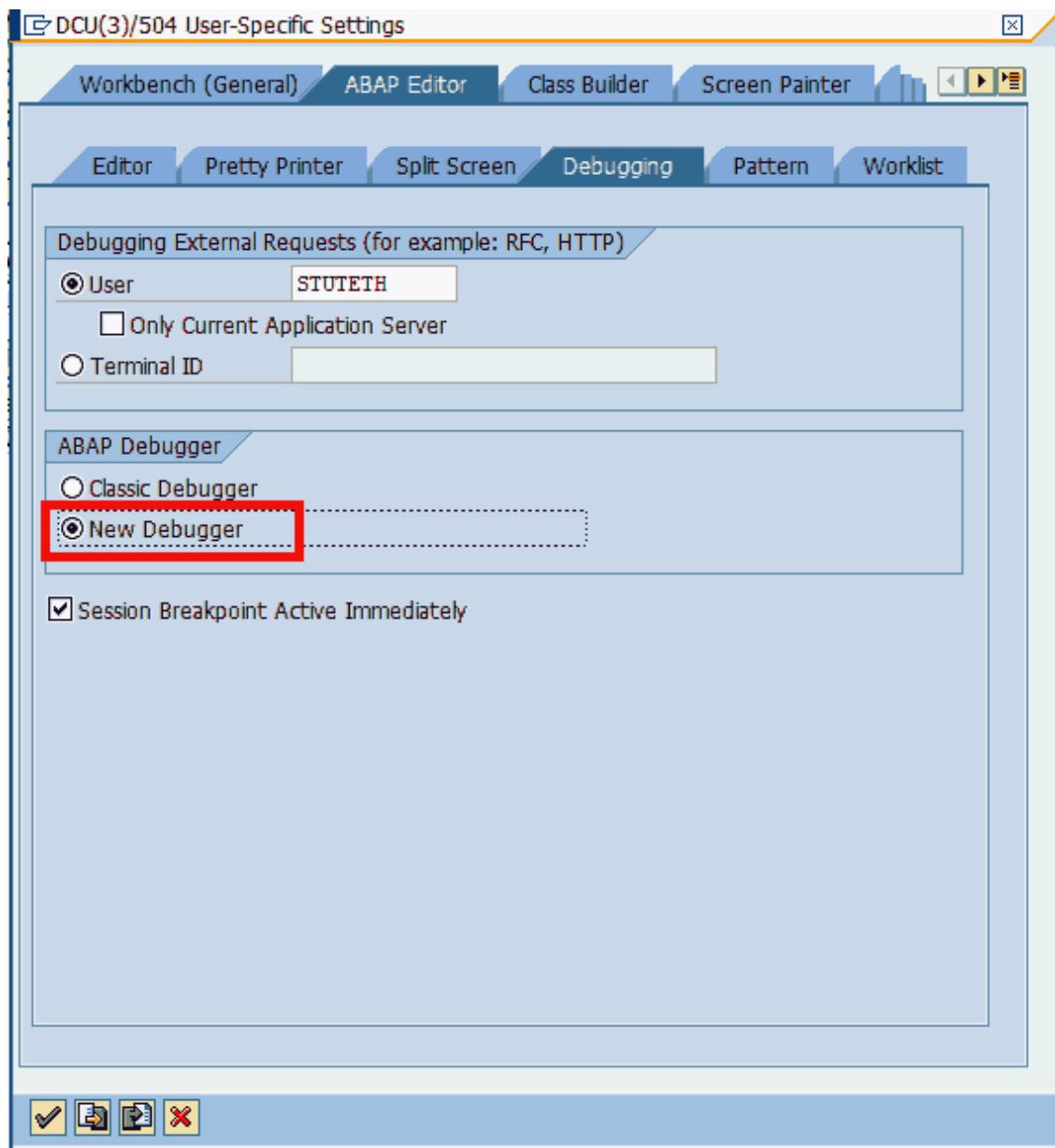


Figure 3.3: User-specific settings for the debugger in transaction SE80

The classic debugger is displayed when the user has already opened five screen modes (reaching the default-setting maximum of six screens). For the use of the new debugger, another free mode is necessary. In this case, the message “No further external mode for the new debugger available” appears in the status line.

If you want to use the new ABAP debugger in these instances, you have to close another screen mode and call the new debugger, as shown in Figure

### 3.4.

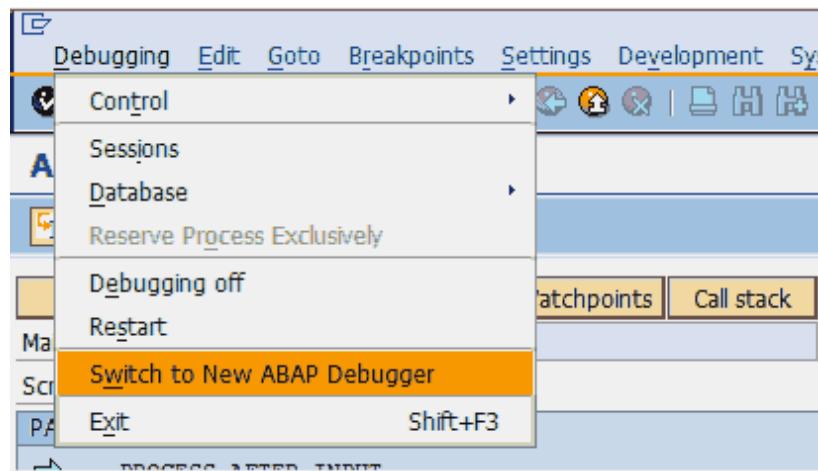


Figure 3.4: Change between the classic and the new debugger

There are two ways to exit the debugger:

1. The program is terminated by clicking on either of these two icons: . However, a simple termination can lead to data inconsistencies.
2. End the program properly. You can do this by using the pull-down menu **BREAKPOINTS** with function **DELETE ALL BPs** or **DEACTIVATE ALL BP**. If you require the same breakpoints in another program test, the deactivate option makes sense. Otherwise, you should delete all breakpoints (BPs). By clicking the **CONTINUE** icon or hitting the “F8” key, the program runs to completion.

#### Be careful while finishing the debugger



The developer needs to know when the debugger has stopped the program run. Terminating the program run at a breakpoint can cause data inconsistencies.

### 3.3 Breakpoint by command – First choice for troubleshooting

Unfortunately, in many cases, SAP error messages are not helpful, i.e., the user receives an error message that doesn't describe the causal error. For error analysis, the first troubleshooting method is the debugger.

The following example is constructed, but in practice is quite easy to find. Start the program for the adjustment of meter reading units (see Figure 3.5).

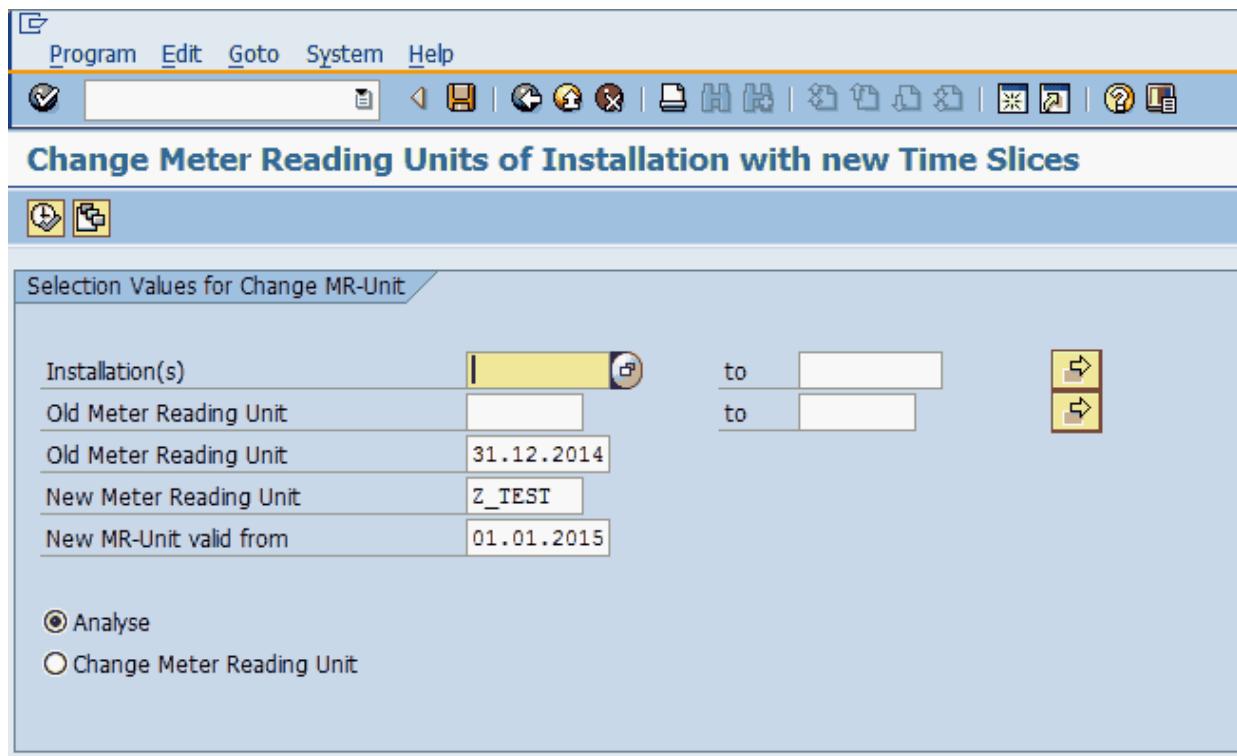


Figure 3.5: Starting the program for the adjustment of meter reading units

After the program starts, the user sees the message “Insufficient Customizing” in the status line – a message without any meaning. The user doesn't recognize the error which has caused the message, and he receives no information on how to repair the error.

To analyze the error, start the program with the same selections. Double click the error message in the status line to see detailed information about the error message (see Figure 3.6).

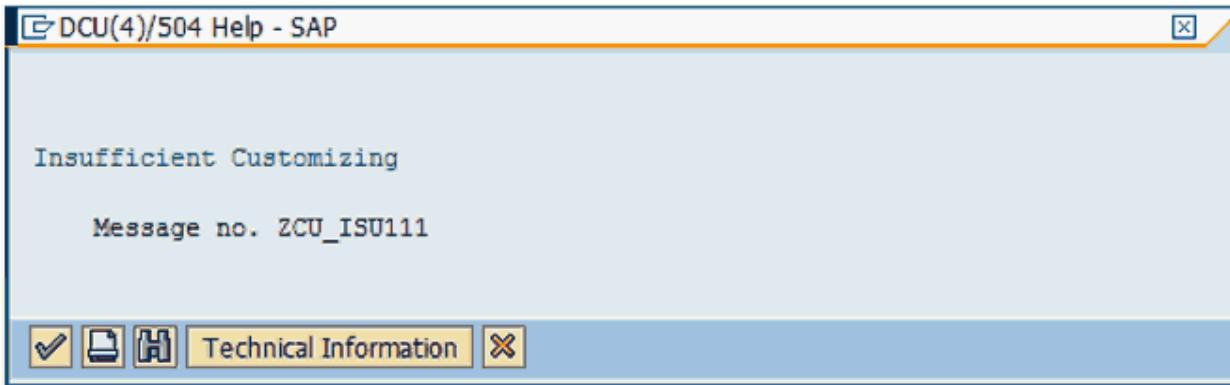


Figure 3.6: Display of message ID and message number of an error message

The error message that is shown has the message-id “ZCU\_ISU” and the message number “111.”

Start the program again with the same selections but activate the debugger with the command /h before starting the program run (see Figure 3.7).

```

21: SELECT-OPTIONS: so_ablal FOR eanlh-ableinh. "Old MR-Unit
22: PARAMETERS: pa_datb TYPE dats          OBLIGATORY DEFAULT '20141231'. "Old MR-Unit v
23: PARAMETERS: pa_abline LIKE te422-termschl OBLIGATORY DEFAULT 'Z_TEST'. "New MR-Unit
24: PARAMETERS: pa_data TYPE dats          OBLIGATORY DEFAULT '20150101'. "New MR-Unit v
25: SELECTION-SCREEN SKIP.
26: PARAMETERS: pa_anal RADIOBUTTON GROUP aus1 DEFAULT 'X'.
27: PARAMETERS: pa_umst RADIOBUTTON GROUP aus1 .
28: SELECTION-SCREEN SKIP.
29: SELECTION-SCREEN END OF BLOCK sel.
30: *-----*
31:
32: *-----*
33: AT SELECTION-SCREEN.
34:
35: * Check: selected to-date against selected from-date
36: | gv_differenz = pa_data - pa_datb.
37:
38: * Program termination if time gab or superposition time
39: | IF gv_differenz > 1.
40: |   MESSAGE e012(zcu_isu).
41: | ELSEIF gv_differenz < 1.
42: |   MESSAGE e013(zcu_isu).
43: | ENDIF.
44:
45:
46:
47:
48: *-----*
49: | * START-OF-SELECTION
50: | *
51: |
52: START-OF-SELECTION.
53:
54: * Check: selected new meter reading unit customized in table TE422
55: SELECT SINGLE *
56:   FROM te422
57:   INTO gs_te422
58:   WHERE termschl = pa_abline.

```

Figure 3.7: The debugger stops the program run after starting the program

In many instances, the program goes through a series of processing blocks until the point is reached where the program calls the error message. That's why it is useful to set a breakpoint at a message statement.

Use the pull-down menu **BREAKPOINTS • BREAKPOINT AT • BREAKPOINT AT MESSAGE** and choose the function “Breakpoint by Message.”

In the new screen, enter the previously noted details of the error message (see Figure 3.8) and confirm the input by **Breakpoint set** “Breakpoint set.”

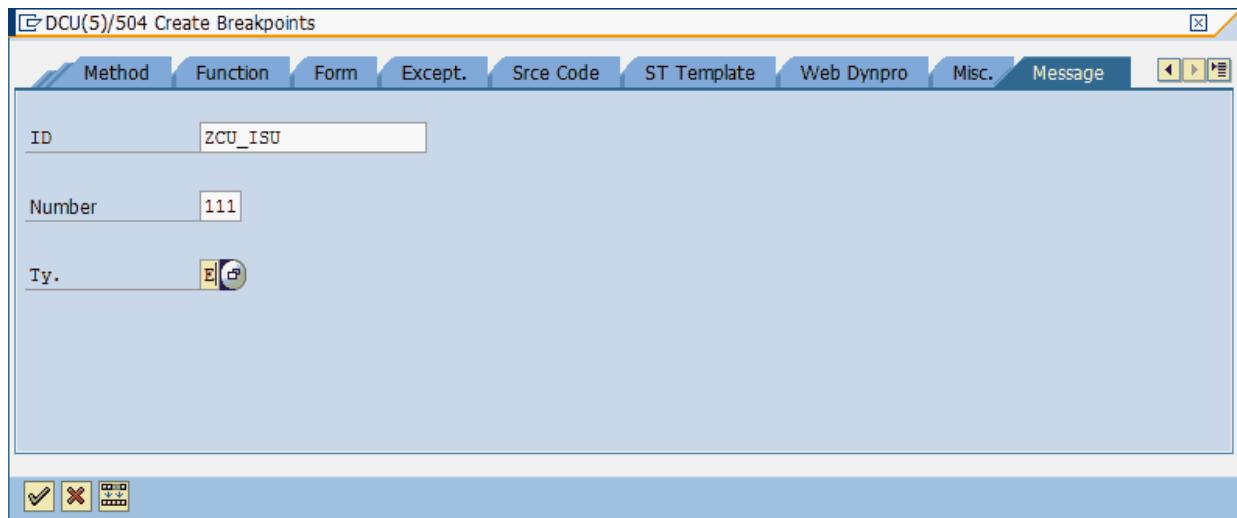


Figure 3.8: Details of the error message

Click the **CONTINUE** icon or hit the “**F8**” key to continue the program run.

The program stops at the point where the message will be printed by the program with the MESSAGE command. Look at the coding and you can see the meter reading unit needing installation adjustment isn’t available in customizing table TE422 (see Figure 3.9).

```

49  *-----*
50  | * START-OF-SELECTION
51  |
52  START-OF-SELECTION.
53
54  * Check: selected new meter reading unit customized in table TE422
55  SELECT SINGLE *
56    FROM te422
57    INTO gs_te422
58    WHERE termschl = pa_ablne.
59
60  IF sv-subrc <> 0.
61  |   MESSAGE e111(zcu_isu).
62  ENDIF.
63
64

```

Figure 3.9: Breakpoint stops the program run at the requested program coding line

When using a printed error message such as, “The new meter reading unit is missed in Table TE422,” troubleshooting is not necessary (see notes in [Section 4.1](#)).

The method “Breakpoint at command” can be used at various times when determining:

- ▶ Which database tables are used in a standard SAP transaction – here you can use a breakpoint at the **SELECT** statement
- ▶ Whether a user exit is available in a standard SAP transaction or not – here you can use a breakpoint at the command “Call Customer-Function”
- ▶ Which database tables are modified in a standard SAP transaction – here you can use a breakpoint at the command INSERT or MODIFY

## 3.4 Watchpoints

A *watchpoint* is a breakpoint attached to a condition. Watchpoints are used when, for example, you have to analyze the program behavior before a certain record in a loop within an internal table with many records.

### How to use a watchpoint



By using the example program ZCU\_FIRST\_CUSTOMER\_MR\_UNIT, 100 installations with the numbers 0060000001 to 0060000100 should be analyzed. During the program run, an error occurs in a LOOP statement for the installation 0060000090.

A breakpoint is entered at the command “LOOP AT gt\_output INTO gs\_output” before running the program. The program stops at this command with the first installation 0060000001. To make sure the program does not stop at each of the following 89 loops, choose **BREAKPOINTS • CREATE WATCHPOINT** and create a watchpoint on the **VARIABLE GS\_OUTPUT-ANLAGE** with the **CONDITION ‘0060000090’** (see Figure 3.10).

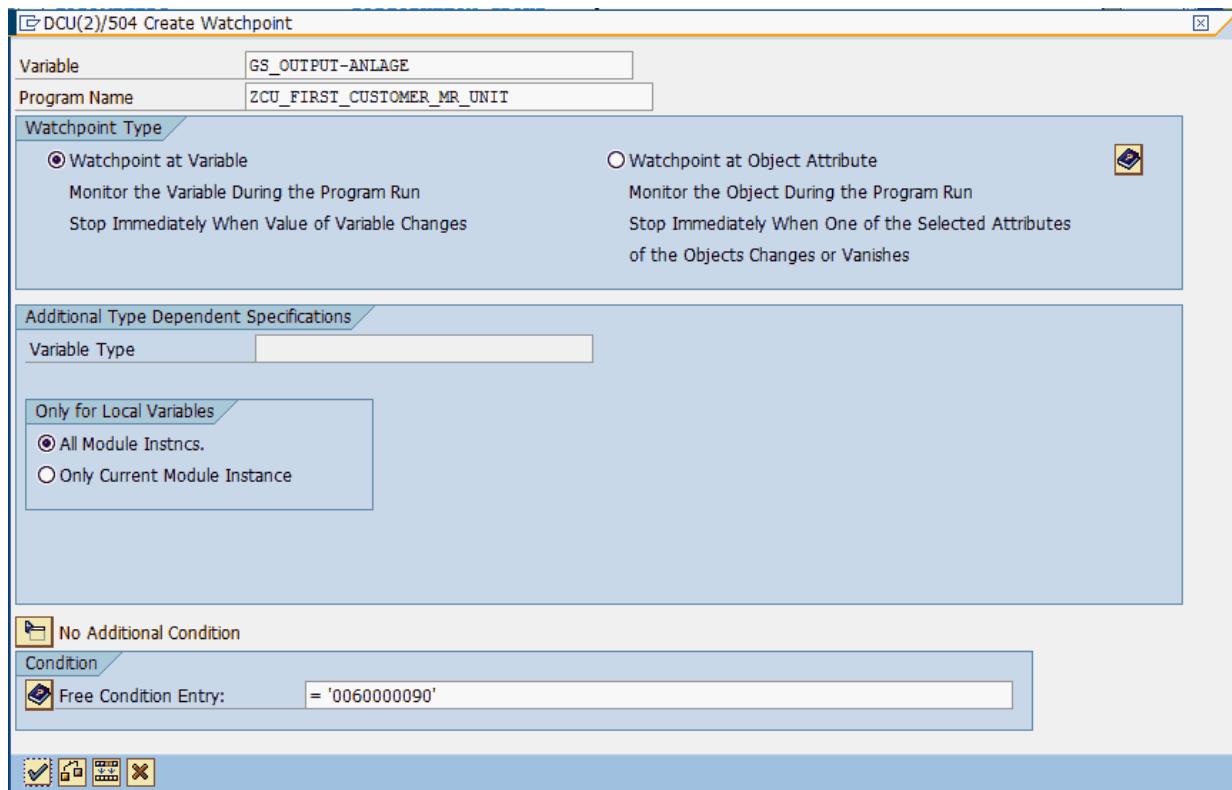


Figure 3.10: Create a watchpoint

The message “Watchpoint has been created” appears in the status line. The program stops at that point, when the variable “gs\_output-anlage” has reached the value “0060000090” (see Figure 3.11). Now you can analyze the loop flow for this installation.

```

151: * Use only time slices of installation,
152: * which Datefrom are bigger equal the datefrom of new meter reading unit
153: LOOP AT gt_output INTO gs_output.
154:   IF gs_output-bis < pa_data.
155:     DELETE gt_output_help WHERE anlage = gs_output-anlage
156:                           AND      bis      = gs_output-bis.
157:   ENDIF.
158: ENDLOOP.

```

Variables 1	Variables 2	Locals	Globals	Auto	Memory Analysis
St...	Variable	Va...	Val.	Ch...	Hexadecimal Value
	GS_OUTPUT-ANLAGE		0060000090		30303630303030303930

Figure 3.11: Stop a watchpoint

If you don't enter a value in the condition entry (see Figure 3.10), the program stops each time if the value has been changed.

In contrast to the breakpoint, the watchpoint is deleted immediately after the end of the program. A breakpoint is cleared after two hours.

### 3.5 Reverse action in debugger

To analyze customer requests, sometimes it is necessary to test a *conditional branch* under different conditions. In such cases, going back to the branching place is useful. Then, after that *reverse jump*, you can modify the condition terms (e.g., the **SY-SUBRC**) and test another program course.

You can use the *reverse jump* in the debugger.

Figure 3.12 shows an example. The program has been stopped at line 383. The program branch shows that the program only runs through if the first 10 characters of the variable “gs\_output-note” has the value “Switch”.

The screenshot shows the SAP Debugger interface with the title bar "R3DCUCI\_DCU\_00". The menu bar includes "Debugger", "Edit", "Goto", "Breakpoints", "Settings", "Miscellaneous", "System", and "Help". The toolbar contains icons for single step, execute, return, continue, and breakpoints. A status bar at the bottom displays the message "several time slices". The code editor shows the following lines:

```
370 *      Check if the change in a settled period takes place
371 IF gs_output-note(6) = 'Switch'.
372 REFRESH lt_ever.
373
374      SELECT *
375      FROM ever
376      INTO TABLE lt_ever
377      WHERE anlage = gs_output-anlage
378      AND loevm    <> 'X'
379      AND auszdat  >= pa_data.
380
381
382
383 IF sy-subrc = 0.
384
385 *      Take all contracts of contract account into account
```

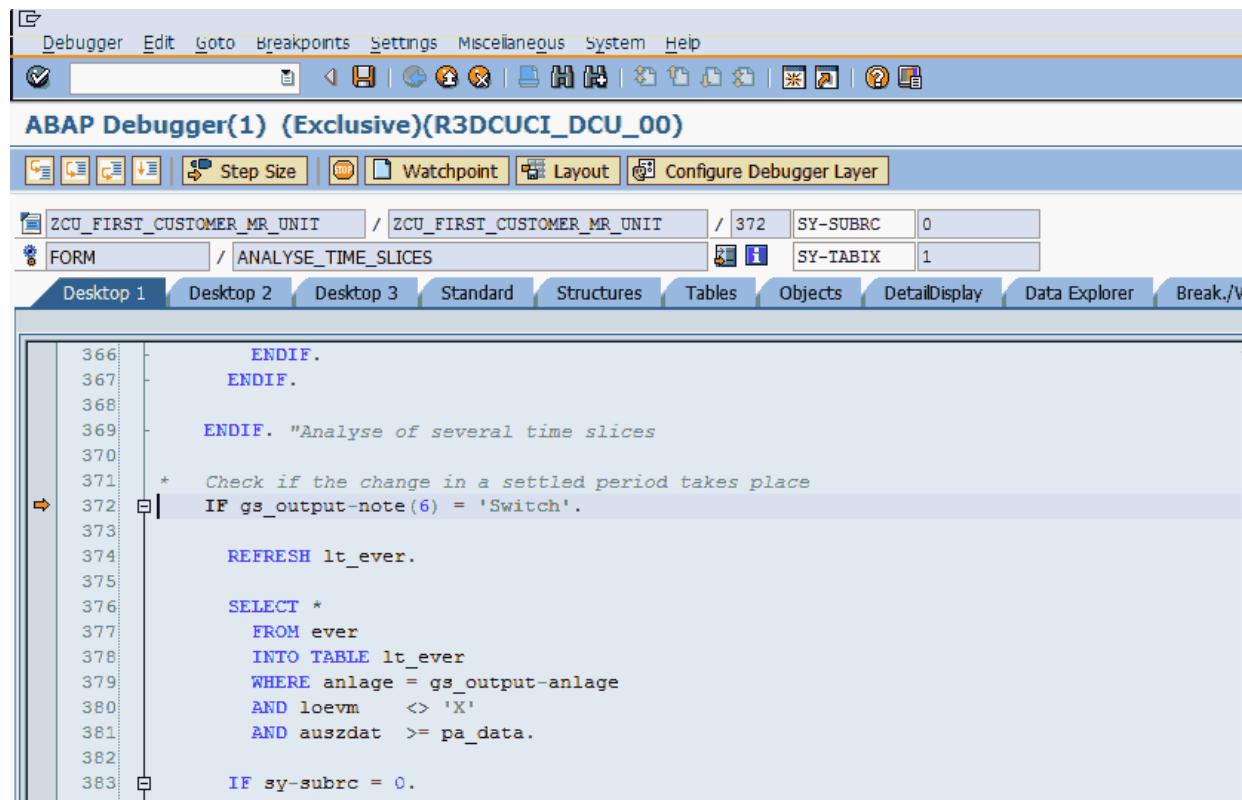
Line 372 and line 383 are highlighted with red boxes. Line 372 contains the condition `IF gs_output-note(6) = 'Switch'.`. Line 383 contains the condition `IF sy-subrc = 0.`.

Figure 3.12: Debugger before the reverse jump

Now you want to analyze the program behavior if the variable does not have the value “Switch.”

Place the cursor on the line with the program branch and use the pull-down menu **DEBUGGER • GO TO STATEMENT**. The program run jumps from line 383 back to line 372.

In Figure 3.13, the yellow arrow in front of line 372 shows the current state of the program run. You can change the value of the variable “gs\_output-note” to “Switch” by using the debugger. After this manipulation, the program continues under the IF statement.



The screenshot shows the ABAP Debugger interface. The title bar reads "ABAP Debugger(1) (Exclusive)(R3DCUCI\_DCU\_00)". The toolbar includes icons for Debugger, Edit, Goto, Breakpoints, Settings, Miscellaneous, System, and Help. Below the toolbar is a menu bar with "Step Size", "Watchpoint", "Layout", and "Configure Debugger Layer". The status bar at the bottom shows the current line number (372), SY-SUBRC (0), and SY-TABIX (1). The main code editor window displays ABAP code. A yellow arrow points to line 372, indicating the current execution point. The code is as follows:

```
366      ENDIF.  
367      ENDIF.  
368  
369      ENDIF. "Analyse of several time slices  
370  
371      * Check if the change in a settled period takes place  
372      IF gs_output-note(6) = 'Switch'.  
373  
374          REFRESH lt_ever.  
375  
376          SELECT *  
377              FROM ever  
378              INTO TABLE lt_ever  
379              WHERE anlage = gs_output-anlage  
380              AND loevm    <> 'X'  
381              AND auszdat  >= pa_data.  
382  
383      IF sy-subrc = 0.
```

Figure 3.13: Debugger after the reverse jump

## Reverse jump in the debugger



Reverse jumps are only possible in a processing block. If the debugger shows the error message “Jump to this command not possible” in the status line, you have to review the processing blocks to find the processing block to which you want to jump back. Then you can make a reverse jump to

an earlier command for calling a processing block (e.g., a form routine or the call of a function module).

### 3.6 Program progress in the debugger – The ABAP stack

The new debugger shows all processing blocks of the program flow in an extra sub-screen.

Use **TRANSACTION** ES21 to change a contract. Before starting the transaction, create a breakpoint at the command “Call Customer-Function.” The new debugger shows the *ABAP stack* at the top right of the debugger screen (see Figure 3.2).

ABAP and Screen Stack							
Sta	Stac...	S...	Event Type	Event	Program	Navi...	Include
8			FORM	OPEN_CUSTOMER_SUBSCREEN	SAPLES20	LES20F30	225
7			FUNCTION	ISU_O_CONTRACT_OPEN	SAPLES20	LES20U06	267
6			FUNCTION	ISU_S_CONTRACT_CHANGE	SAPLES20	LES20U09	35
5			FORM	ENTRY	SAPLES20	LES20F00	296
4			MODULE (PAI)	PAI_100_ACTIONS	SAPLES20	LES20I01	115
3			PAI MODULE	PAI_100_ACTIONS			17
2			PAI SCREEN	0100	SAPLES20		17
1			TRANSACTION	ES21(ES21)			0

Figure 3.14: ABAP and screen stacks

Figure 3.14 shows the ABAP stack section with all screens and processing blocks that have been used in the current program line in reverse order.

The line with the yellow arrow in front shows the current state of the program flow. With a mouse click on another processing block, the debugger makes a reverse jump to the program line, where the next processing block has been called.

It is not possible to make a revers jump to a screen stack on a processing block of the screen flow logic. These processing blocks are marked with the  icon in the third column. However, you can make a reverse jump to a process after input (PAI) module – which is a form of subroutine. The reachable processing blocks are marked with the  icon in the third column.

This allows you to change export parameters of a calling function module. Such parameters can't be changed in the following processing block because there are no changing parameters.

So it is possible to change the variable “X\_UPD\_ONLINE = X” to the value “space”. In this case, no changes will be saved in the database because the variable is no longer selected.

Changing variables can cause inconsistencies in the course of the program or crash (dump) the program.

In the ABAP stack, the programs are called with their names and the names of the accompanying includes. If the program names start with the term “SAPL,” then they are related to function groups.

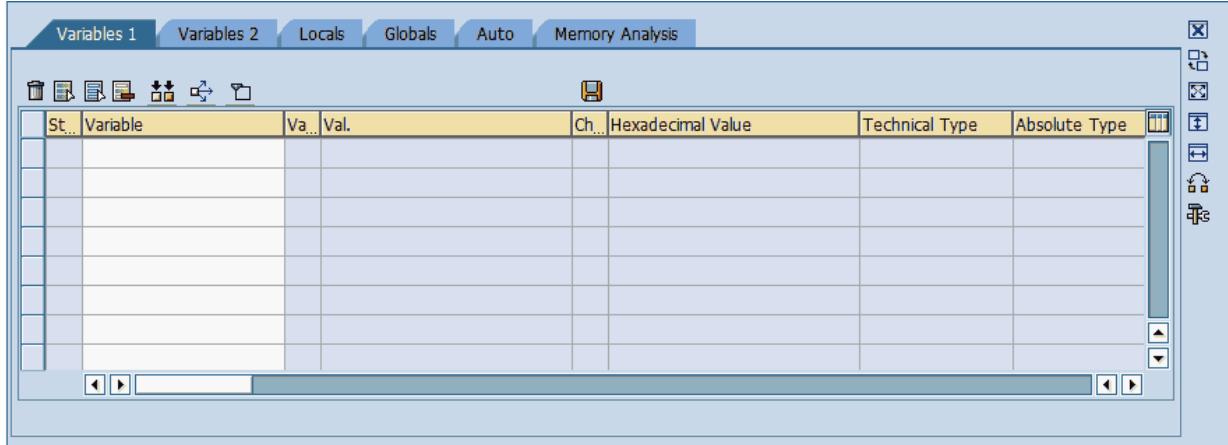
You can look at the coding of such function groups by taking the name after the term “SAPL” and using it in **TRANSACTION** SE80 for the object type “Function Group.”

In the example shown in Figure 3.14, the program is named “SAPLES20.” In **TRANSACTION** SE80, you can use the name “ES20” for calling the corresponding function group.

With the help of the ABAP stack, you can analyze all preceding processing blocks and you can change values of variables. By clicking the  icon in the seventh column **NAVI(GATION)**, you can navigate to the ABAP editor with the coding of the processing group and make changes.

### 3.7 Local and global variables

Figure 3.15 shows the variable area of the right lower area in the standard screen of the new debugger.



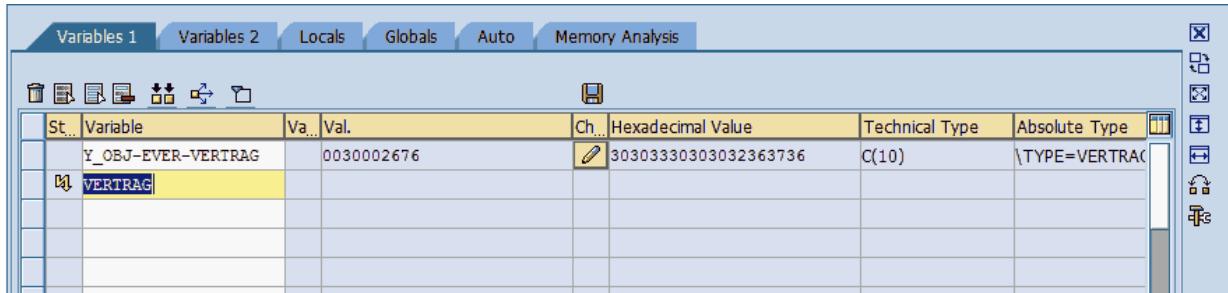
The screenshot shows the variable area of the debugger. At the top, there are tabs: Variables 1, Variables 2, Locals, Globals, Auto, and Memory Analysis. The Locals tab is selected. Below the tabs is a toolbar with icons for delete, copy, paste, and other functions. A large table follows, with columns labeled: St..., Variable, Va..., Val., Ch..., Hexadecimal Value, Technical Type, and Absolute Type. The table is currently empty, with no data rows visible.

St...	Variable	Va...	Val.	Ch...	Hexadecimal Value	Technical Type	Absolute Type

Figure 3.15: Local and global variables in the debugger

Selecting the tab **DESKTOP 3** of the debugger, the complete right screen area displays the variables of the program.

In tabs **VARIABLES 1** and **VARIABLES 2**, free eligible values and their values of the current processing block are shown (see Figure 3.16).



The screenshot shows the variable area of the debugger. The Locals tab is selected. The table now contains two data rows:

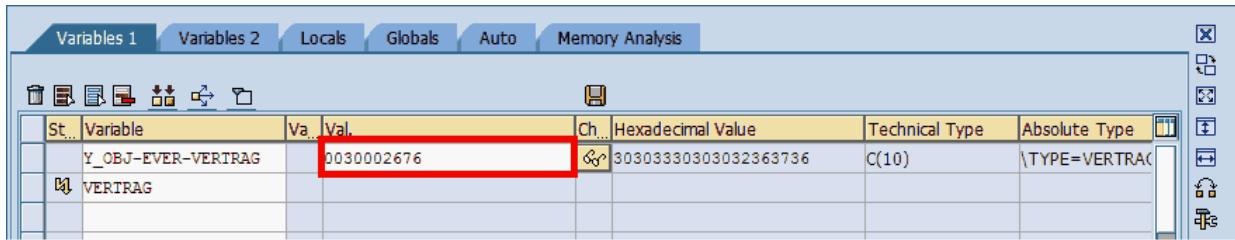
St...	Variable	Va...	Val.	Ch...	Hexadecimal Value	Technical Type	Absolute Type
	Y_OBJ-EVER-VERTRAG	0030002676			30303330303032363736	C(10)	\TYPE=VERTRAG
	VERTRAG						

Figure 3.16: Display of variable values in the debugger

In this example, the **VARIABLE** “XY\_OBJ-EVER-VERTRAG” with the current **VALUE** is displayed. The  icon in front of the variable VERTRAG means that this variable is not available in the current processing block.

The **PENCIL**  icon is displayed in the fifth column if the value of the variable is editable in the current processing block. To change the value,

click on the **PENCIL**  icon beside the value you want to change (see Figure 3.17).

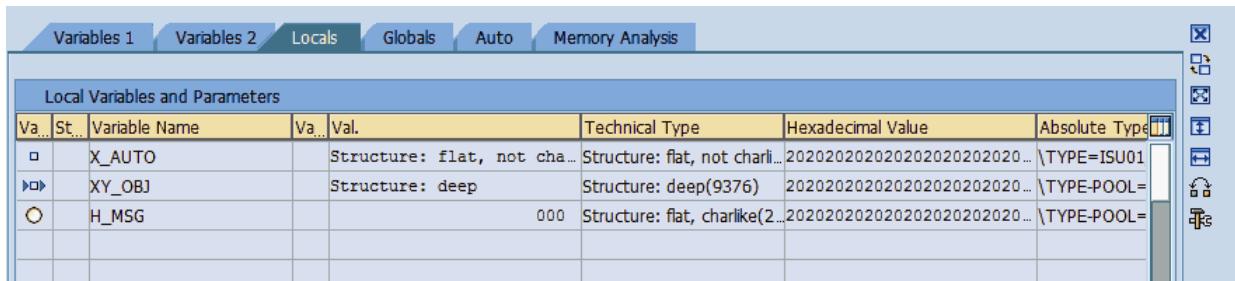


St...	Variable	Va...	Val.	Ch...	Hexadecimal Value	Technical Type	Absolute Type
□	Y_OBJ-EVER-VERTRAG		0030002676	✎	30303330303032363736	C(10)	\TYPE=VERTRAG
□	VERTRAG						

Figure 3.17: Change of variable values in the debugger

At this point, you can enter another contract number. To save the new contract number, confirm the change by hitting the “Enter” key.

The third tab of the variable area shows the local **VARIABLE NAME** listing of the current processing block (see Figure 3.18).



Local Variables and Parameters							
Va...	St...	Variable Name	Va...	Val.	Technical Type	Hexadecimal Value	Absolute Type
□	X_AUTO			Structure: flat, not cha...	Structure: flat, not charli...	202020202020202020202020...	\TYPE=ISU01
▶▷	XY_OBJ			Structure: deep	Structure: deep(9376)	202020202020202020202020...	\TYPE=POOL=
○	H_MSG			000	Structure: flat, charlike(2...	202020202020202020202020...	\TYPE=POOL=

Figure 3.18: Types of variables in the debugger

In the first column, the different types of variables are shown with different icons:

-  X\_AUTO is a changeable *using variable*
-  XY\_OBJ is a changeable *import variable*
-  H\_MSG is a *local variable*

The value of a structure variable can be changed with a double click on a variable name. After a double click, the complete structure of the variable is shown (see Figure 3.19 for an example of the XY\_OBJ structure).

Debugger Edit Goto Breakpoints Settings Miscellaneous System Help

**ABAP Debugger(1) (Exclusive)(R3DCUCI\_DCU\_00)**

Step Size Watchpoint Layout Configure Debugger Layer

SAPLES20 / LES20F30 / 225 SY-SUBRC 0  
 FORM / OPEN\_CUSTOMER\_SUBSCREEN SY-TABIX 1

Desktop 1 Desktop 2 Desktop 3 Standard Structures Tables Objects DetailDisplay Data Explorer Break./Watchpoints

Structures Fld.list

Struct. XY\_OBJ

Struct. Type Structure: deep(9376)

Exp.	Component	Val...	Val.	Cha...	Technical Type	Hexadecimal Value	Absolute Type	Rea...
	PUBLIC				Structure: flat, charlike	Structure: flat, ch...	20...	\TYPE=ISU00_OB...
	EVER		5040030002676001102 01			35303430303330303236373630303131...	\TYPE=EVER	
	MANDT		504		C(3)	353034		\TYPE=MANDT
	VERTRAG		0030002676		C(10)	30303330303032363736		\TYPE=VERTRAG
	BUKRS		0011		C(4)	30303131		\TYPE=BUKRS
	SPARTE		02		C(2)	3032		\TYPE=SPARTE
	EIGENVERB				C(1)	20		\TYPE=EIGENVER...
	KOFIZ	01			C(2)	3031		\TYPE=E_KOFIZ
	PORTION				C(8)	2020202020202020		\TYPE=PORTION...
	ABSLANFO				C(1)	20		\TYPE=ABSLANFO
	ABSZYK				C(2)	2020		\TYPE=ABSZYK
	ABSMNANP				C(1)	20		\TYPE=ABSMNANP
	GEMFAKT	1			C(1)	31		\TYPE=E_GEMFAK..
	MANABR				C(1)	20		\TYPE=MANABR
	ABRSPERR				C(2)	2020		\TYPE=ABRSPERR
	ABRFREIG				C(2)	2020		\TYPE=ABRFREIG
	BSTATUS				C(2)	2020		\TYPE=BEARKZ
	KOSTL				C(10)	2020202020202020		\TYPE=KOSTL
	VBEZ		VERTRAG 30002676		C(35)	56455254524147203330303236373620...	\TYPE=E_VBEZ	
	VBEGINN		00000000		D(8)	30303030303030		\TYPE=E_VBEGINN
	EINZDAT...		20020101		D(8)	3230303230313031		\TYPE=EINZDAT...
	VENDE		00000000		D(8)	30303030303030		\TYPE=E_VENDE

Figure 3.19: Change values of structure fields in the debugger

Click the **PENCIL** icon beside the variable value to change that value. Confirm the change by hitting the “Enter” key.

### Uneditable variable values



Variable values of a processing block that cannot be edited do not have the **PENCIL** icon in the column behind the value.

The fourth tab of the variable area (**Globals**) shows the global variables of the current processing block (see Figure 3.20). This variable is visible and changeable in all processing blocks of the same program.

The screenshot shows a debugger interface with a tab bar at the top: Variables 1, Variables 2, Locals, **Globals**, Auto, Memory Analysis. The Globals tab is selected. Below the tabs is a table titled "Global Variables". The columns are: Va..., St..., Variable Name, Va..., Val., Technical Type, Hexadecimal Value, and Absolute Type. The table contains the following data:

Va...	St...	Variable Name	Va...	Val.	Technical Type	Hexadecimal Value	Absolute Type
		BUT000		Structure: flat, not cha...	Structure: flat, not charli...	202020202020202020202020...	\TYPE=BUTO
		EVERD		0030002676	... Structure: flat, charlike(1 ...	2020203030333030323637...	\TYPE=EVERD
		EVERU			... Structure: flat, charlike(6 ...	202020202020202020202020...	\TYPE=EVERU
		EVERA			... Structure: flat, charlike(4 ...	202020202020202020202020...	\TYPE=EVERA
		T000		504Liefermandant Conuti	Structure: flat, charlike(1 ...	3530344C69656665726D616E...	\TYPE=T000
		T001		5040011Conuti Stadtwerke	Structure: flat, charlike(2 ...	35303430303131436F6E7574...	\TYPE=T001
		T001W			Structure: flat, not cha...	202020202020202020202020...	\TYPE=T001W
		TE015T		Structure: flat, charlike	Structure: flat, charlike(3 ...	202020202020202020202020...	\TYPE=TE015T

Figure 3.20: Global variables in the debugger

Global program variables are of particular interest for the “dirty assign” method shown in [Section 4.15](#).

### 3.8 Change table contents in running programs

The new ABAP debugger provides extensive tools to modify table contents while running a program. Use the already-known program example ZCU\_FIRST\_CUSTOMER\_ABLEINH (Figure 2.91) for conversion of the meter reading unit. To explain the manipulation of table content, select 100 installations and use a breakpoint at the LOOP statement. Double click on the table “GT\_OUTPUT” to see the table variable area on the bottom right of the screen (shown in Figure 3.21).

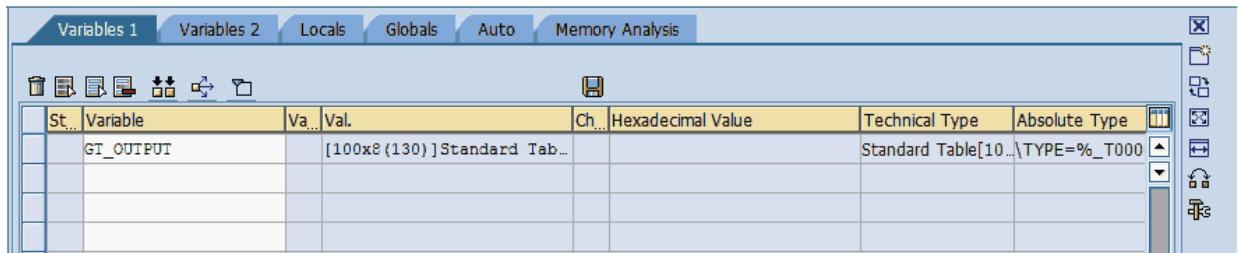


Figure 3.21: Display of tables in the new ABAP debugger

Double click on the table “GT\_OUTPUT” to open the table view in the debugger, as shown in Figure 3.22.

Row	ANLAGE [C(10)]	BIS [D(8)]	AB [D(8)]	AKLASSE [C(4)]	ABLEINH [C(8)]	SEVERAL_TS [C...]	INFINITELY_TS [...]	NOTE [C(90)]
1	0060000000	99991231	20130101	FK	J01-004K			
2	0060000001	99991231	20130101	FK	J01-004K			
3	0060000002	99991231	20091101	FK	J01-004K			
4	0060000003	99991231	20060101	FK	J01-004K			
5	0060000004	99991231	20091101	FK	J01-004K			
6	0060000005	99991231	20070101	FK	J01-004K			
7	0060000006	99991231	20100101	FK	J01-004K			
8	0060000007	99991231	20100101	FK	J01-004K			
9	0060000008	99991231	20100101	FK	J01-004K			
10	0060000009	99991231	20091101	FK	J01-004K			
11	0060000010	99991231	20130101	FK	J01-004K			

Figure 3.22: Table view in the debugger

On the right in the figure, you can see the **TOOL** icon. You can edit table contents, such as:

- ▶ Change single lines

- ▶ Delete marked lines
- ▶ Append lines
- ▶ Insert lines
- ▶ Delete line areas or the whole table

By clicking the **TOOL**  icon, you get the view shown in Figure 3.23.

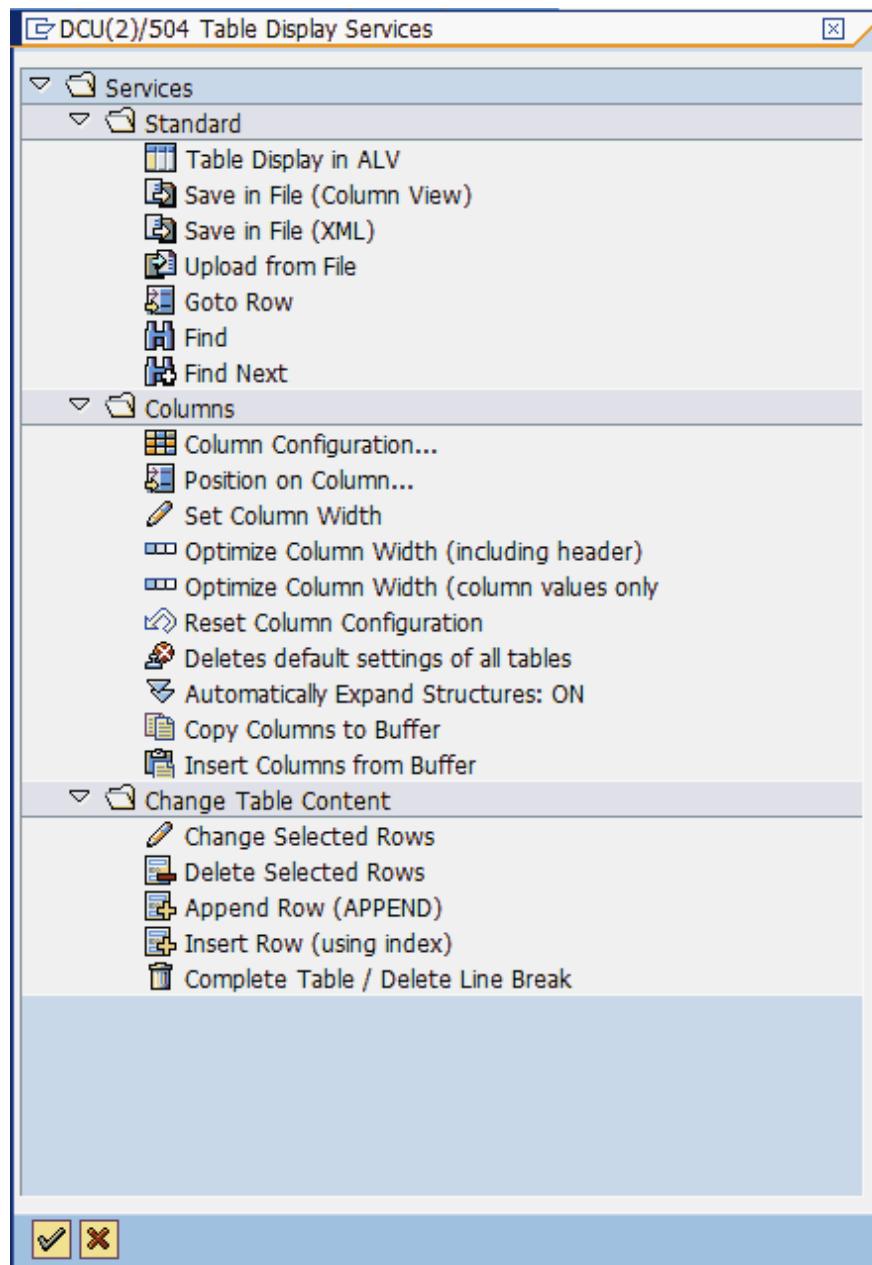


Figure 3.23: Tools to edit table content in the debugger

## 3.9 Special debugging

Now that you know the essential tools required in the ABAP debugger, let's get into certain special debugging techniques.

### 3.9.1 System debugger

The system debugger allows you to analyze certain SAP system programs that you can't debug with the normal debugger. These can be such programs as Screen Painter or the Data Dictionary.

In the next example, I use Screen Painter for the customer-specific screen 0200 of the function group XES20. After starting the Screen Painter to display screen, you can analyze the program course with the pull-down menu **SCREEN • TEST** (see Figure 3.24).

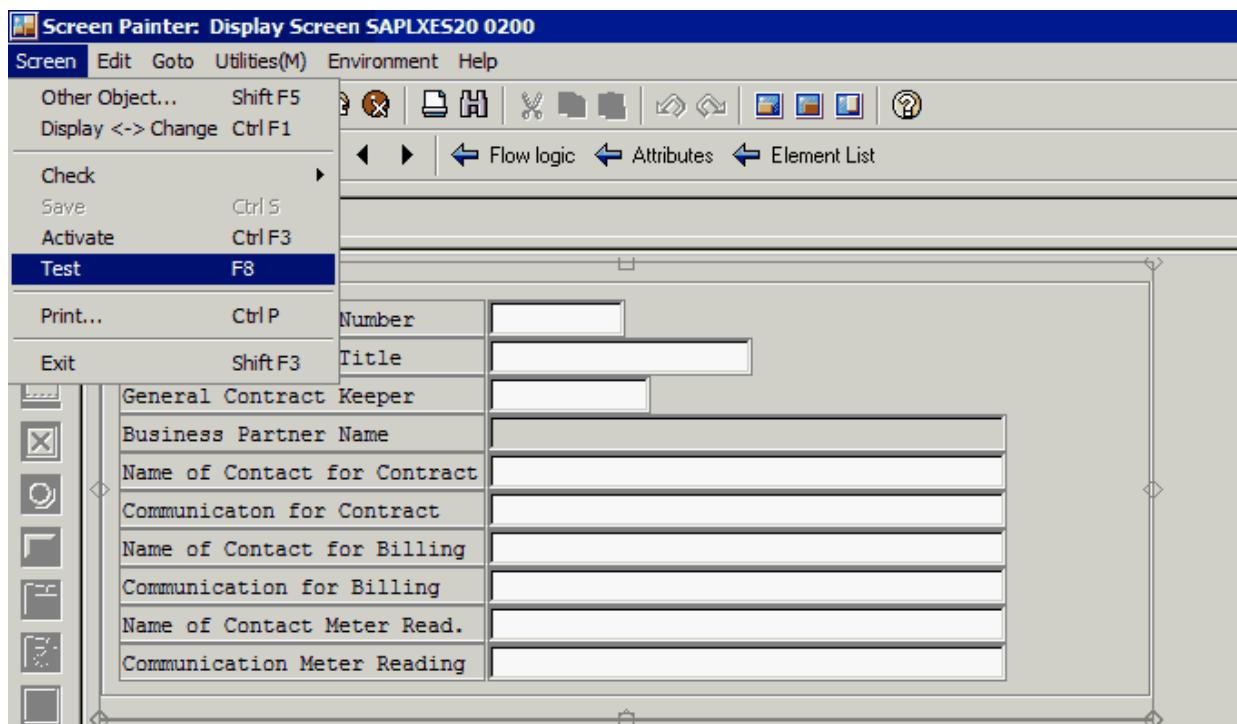


Figure 3.24: Screen test in the Screen Painter

Start the debugger with the command /h and hit the “Enter” key twice. Figure 3.25 shows the debugger with the ABAP stack of the screen logic.

ABAP and Screen Stack								
Sta...	Stac...	S...	Event Type	Event	Program	Navi...	Include	Line
⇒ 27			PAI SCREEN	0401	SAPLSCRO			20
26			FUNCTION	RS_DYNPRO_TEST	SAPLSCRO		LSCR0U01	366
25			FORM	SIMULATE	SAPLWBSCREEN		LWBSCREENF50	1.769
24			FORM	TALK_TO_GRAF_SCREEN	SAPLWBSCREEN		LWBSCREENF70	481
23			FORM	ENTER_GRAF_SCREEN	SAPLWBSCREEN		LWBSCREENF70	17
22			MODULE (PBO)	D2110_PBO	SAPLWBSCREEN		LWBSCREEN000	229
21			PBO MODULE	D2110_PBO				2
20			PBO SCREEN	2110	SAPLWBSCREEN			2
19			FUNCTION	WB_SCRP_PROCESS	SAPLWBSCREEN		LWBSCREENU02	182
18			METHOD	PROCESS_DYNPRO	CL_WB_SCREEN_PAINTER..		CL_WB_SCREEN_PAINTER..4	
17			METHOD	IF_WB_PROGRAM~PROC..	CL_WB_SCREEN_PAINTER..		CL_WB_SCREEN_PAINTER..281	
16			METHOD	DO_THE_NAVIGATION	CL_WB_NAVIGATOR==...		CL_WB_NAVIGATOR==...	149

Figure 3.25: ABAP stack of the screen logic

This ABAP stack shows that the program uses some processing blocks of the system program “SAPLWBSCREEN”. A reverse jump to these processing blocks with the classic debugger isn’t possible. With a double click on the program name SAPLWBSCREEN with the routine “SIMULATE” (highlighted in Figure 3.25) you receive the warning message “Navigation in stack is only possible with screen debugging” in the status line.

Now start the system debugger as shown in Figure 3.26 (Pull-down menu **SETTINGS • SYSTEM DEBUGGING ON/OFF**).

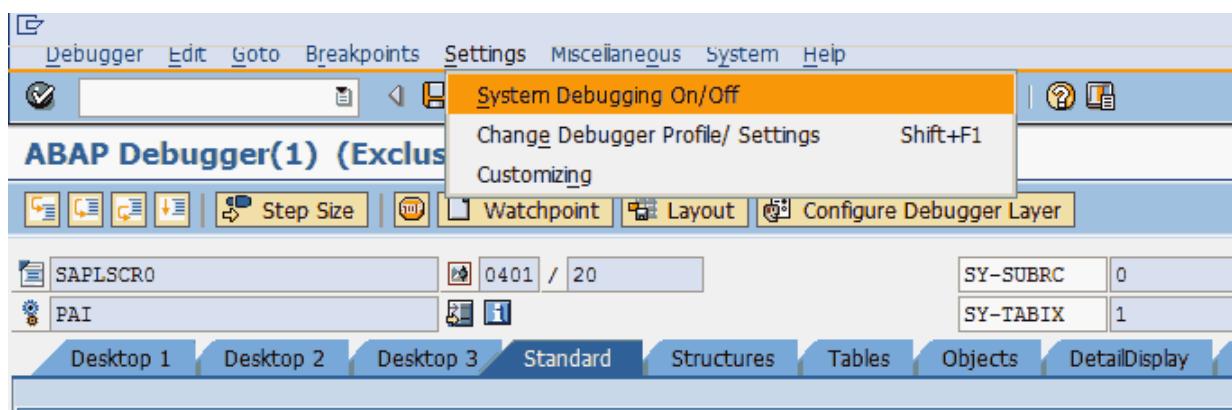


Figure 3.26: Starting the system debugger

The following message appears in the status line “System Debugging activated.” End the current debugger session by clicking the icon or hitting the “F8” key.

The screen test screen appears again. Start the debugger with the command /h and hit the “Enter” key twice to see the changed ABAP stack (see Figure 3.27).

ABAP and Screen Stack								
Sta...	Stac...	S...	Event Type	Event	Program	Navi...	Include	Line
⇒ 29			MODULE (PAI)	%_CTL_INPUT1	SAPMSSYD		SAPMSSYD	252
28			PAI MODULE	%_CTL_INPUT1				0
27			PAI SCREEN	0401	SAPLSCR0		CONTROL INPUT	0
26			FUNCTION	RS_DYNPRO_TEST	SAPLSCR0		LSCR0U01	366
25			FORM	SIMULATE	SAPLWBSCREEN		LWBSCREENF50	1.769
24			FORM	TALK_TO_GRAF_SCREEN	SAPLWBSCREEN		LWBSCREENF70	481
23			FORM	ENTER_GRAF_SCREEN	SAPLWBSCREEN		LWBSCREENF70	17
22			MODULE (PBO)	D2110_PBO	SAPLWBSCREEN		LWBSCREEN000	229
21			PBO MODULE	D2110_PBO				2
20			PBO SCREEN	2110	SAPLWBSCREEN			2
19			FUNCTION	WB_SCRP_PROCESS	SAPLWBSCREEN		LWBSCREENU02	182
18			METHOD	PROCESS_DYNPRO	CL_WB_SCREEN_PAINTER		CL_WB_SCREEN_PAINTER.4	

Figure 3.27: System programs in the system debugger

Now the debugger shows system programs such as the event %\_CTL\_INPUT1. A reverse jump to the Event SIMULATE of the program SAPLWBSCREEN is now possible.

The system debugger can be switched off using the pull-down menu **SETTINGS • SYSTEM DEBUGGING ON/OFF**. The status line then shows the message “System debugging deactivated.”

### Direct start of the system debugger



You can start the system debugger directly with the command `/hs` in the command line.

### 3.9.2 Debugging of popup screens

The use of the command line is not possible if popup screens appear. Therefore, you have to use a little trick to start the debugger directly after the

popup screen, but some preparatory work is necessary. Use the SAP logon pad (see Figure 3.28).

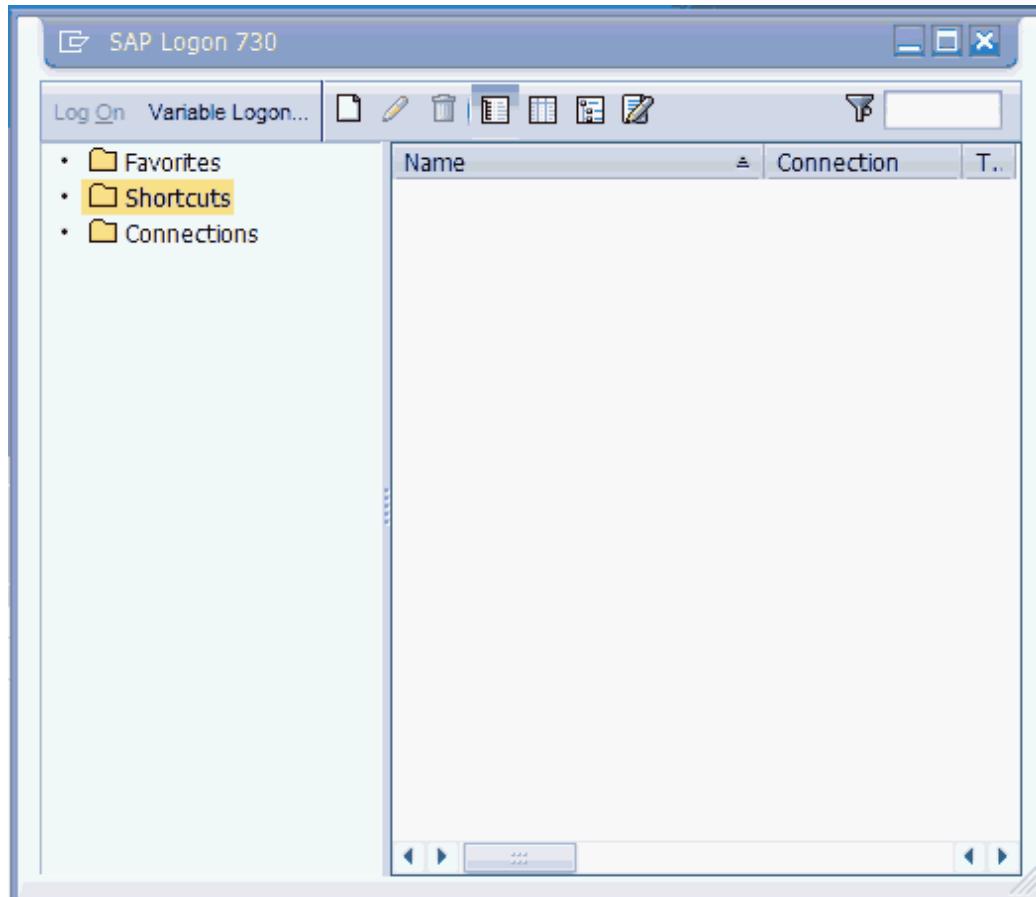


Figure 3.28: The SAP logon pad

Select the menu option **SHORTCUTS** in the left screen area. Create a new SAP shortcut using the icon. Enter details in the screen as shown in Figure 3.29.

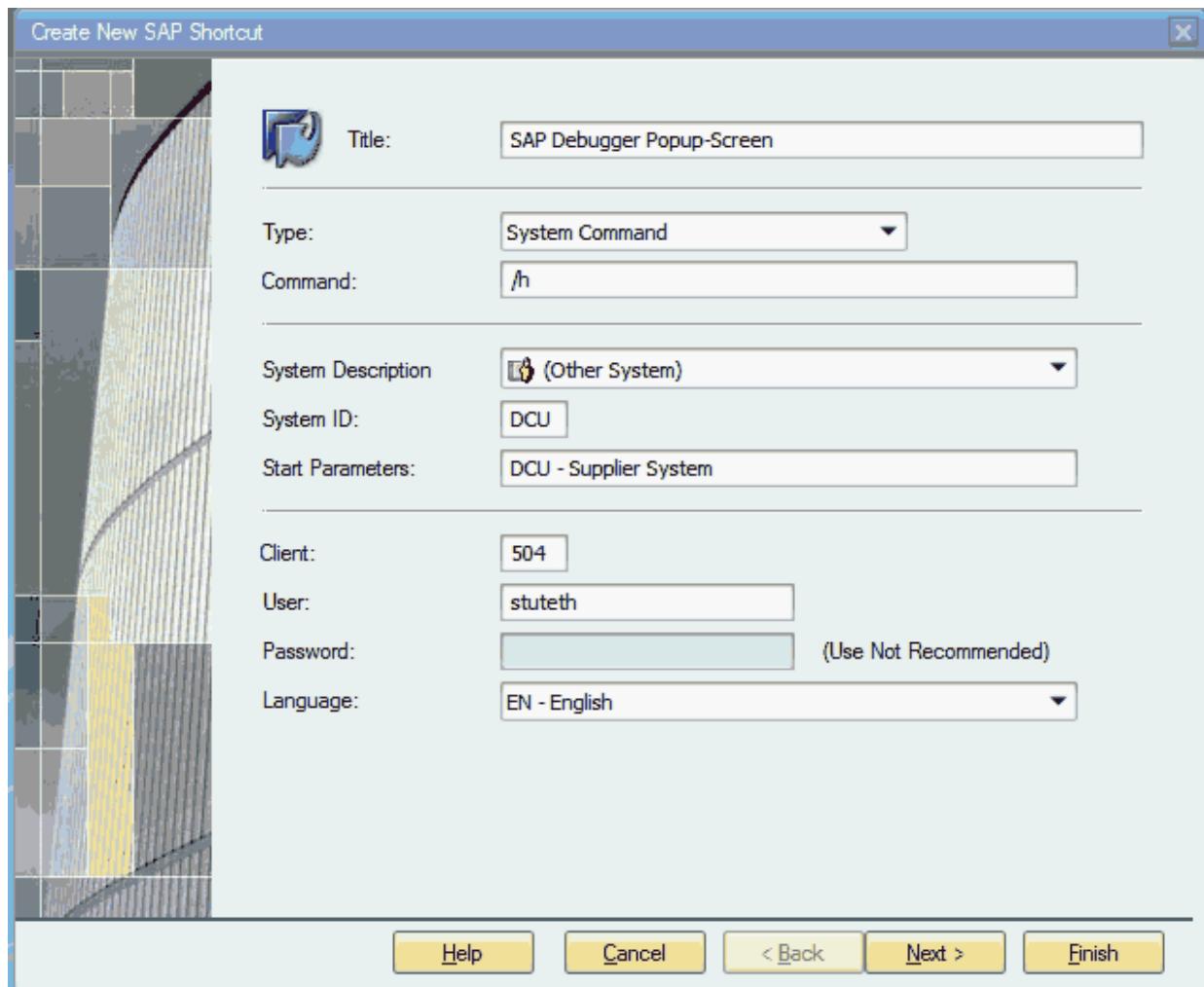


Figure 3.29: System command /h in SAP logon pad

Select the shortcut type “System Command” from the dropdown menu in the **TITLE** field and type the debugging command /h in the **COMMAND** field. Enter or select appropriate entries for the fields **SYSTEM DESCRIPTION**, **SYSTEM ID**, **START PARAMETERS**, **CLIENT**, **USER**, and **LANGUAGE**. Save these entries by clicking the **FINISH** button.

Now use the suitable transaction with the popup menu in the SAP system – in my example, I used **TRANSACTION** BUG3 to create **BUSINESS PARTNER**. A popup screen for choosing a business partner appears (see Figure 3.30).

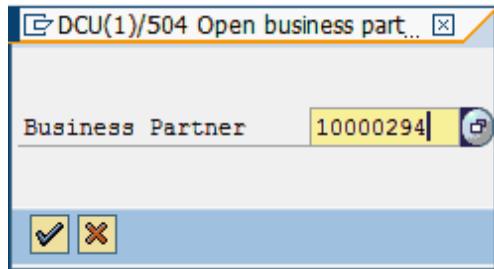


Figure 3.30: Choose a business partner in a popup screen

After inputting a business partner number, use the tab **SHORTCUTS** of the SAP logon pad. With a double click on the shortcut “SAP Debugger Popup Screen,” which has to be created before the debugger is started (Figure 3.28 and Figure 3.29, the current popup screen in the SAP system opens the business partner.

The status line for choosing the business partner shows the message “Debugger has been started.” Confirm your choice of the business partner by clicking the icon to start the debugger.

This method is suited for debugging popup screens and other sub screens.

### 3.9.3 Debugging background processing

You can also analyze the course of background programs with the debugger. You only need to write a few program lines. If you have to analyze standard SAP programs, the code can be implemented modification-free using *enhancement technology* (see the *Practical Guide to SAP ABAP: Part 2*).

For the next example, use program ZCU\_ABAP\_BACKGROUND\_DEBUGGING to display time slices of selected installations. This program will start in the background to demonstrate a method of background debugging.

First, create an endless loop at a place in the program that will be run through in every case. The endless loop may look like the code as shown in Figure 3.31.

```

* Start of Endlessloop for Background Debugging

gv_exit = space.

DO.
  IF gv_exit = 'X'.
    EXIT.
  ENDIF.
ENDDO.

* End of Endlessloop for Background Debugging

```

Figure 3.31: Endless loop for background debugging

To start the background processing, select 100 installations and execute the program as shown in Figure 3.32.

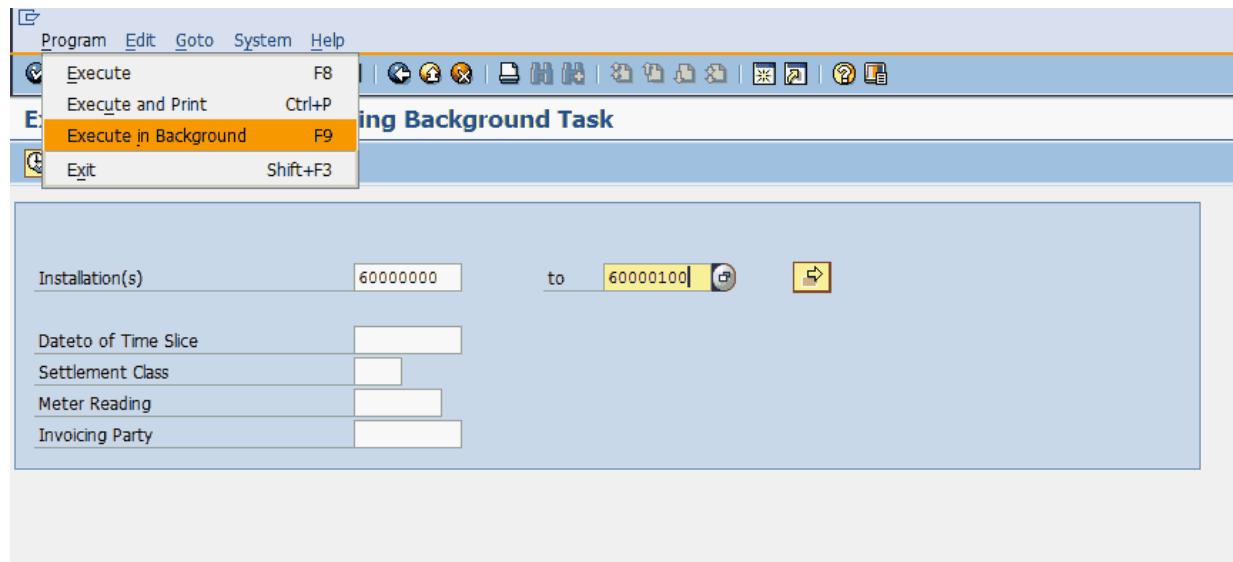


Figure 3.32: Execute a program in the background

Before the program starts in the background, a popup screen appears for inputting print parameters. Choose the printer “LOCL” and confirm the input by clicking the  icon. A new popup screen appears that enables you to input the start time of the program run. As shown in Figure 3.33, choose the start time **IMMEDIATE** and confirm the input by clicking the **SAVE**  icon.

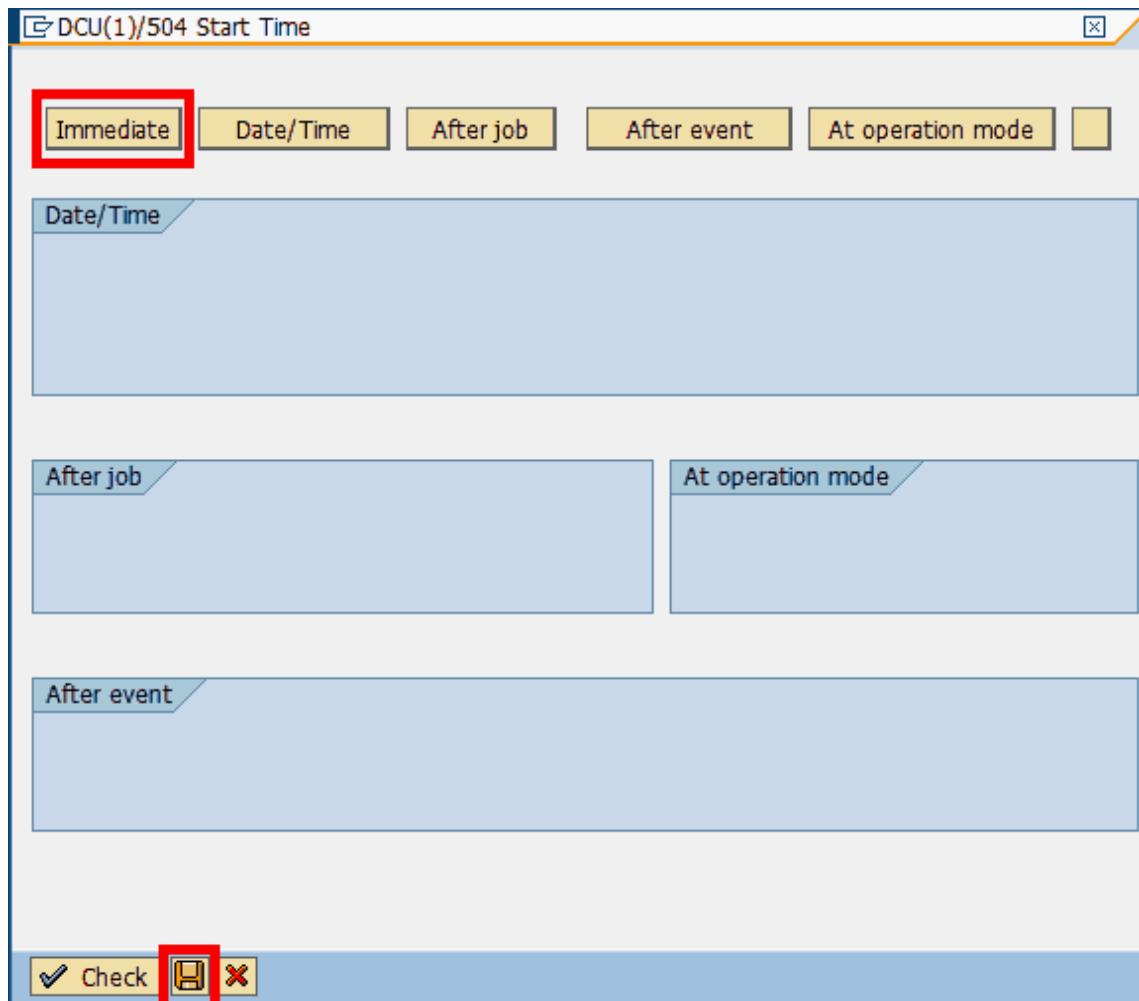


Figure 3.33: Start time for the background program run

The program now starts in the background.

For checking the program, use the pull-down menu **SYSTEM • SERVICES • JOBS • JOB OVERVIEW**.

The job overview shows the following screen (see Figure 3.34) if the job has been started successfully.

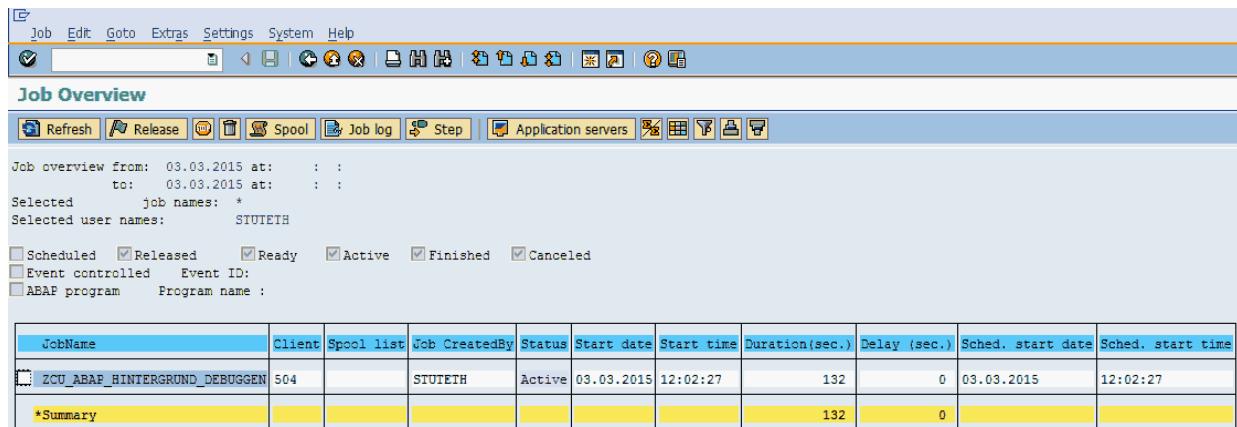


Figure 3.34: Active background job in endless loop

For the next step, use **TRANSACTION SM50** (process overview). The process overview shows, in line 40 of this example, that the batch run of the background job has been started (see Figure 3.35).

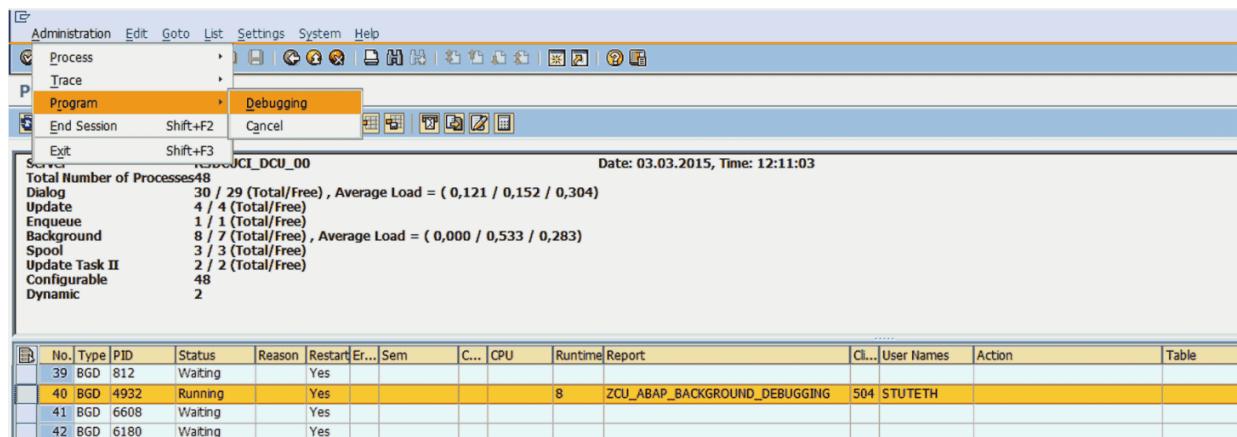


Figure 3.35: Process overview with transaction SM50

Mark the corresponding line in the first column and use the pull-down menu **ADMINISTRATION • PROGRAM • DEBUGGING** to start the debugger for background programs. A popup screen appears asking whether the program needs to be debugged or not. Confirm this question by clicking the  icon. The ABAP debugger starts and shows a command in the endless loop (see arrow in front of line number 65 in Figure 3.36).

The screenshot shows the ABAP Debugger interface. The title bar reads: '(/hs)ABAP Debugger(1) (Exclusive) - ATTACHED -(R3DCUCI\_DCU\_00)'. The menu bar includes: Debugger, Edit, Goto, Breakpoints, Settings, Miscellaneous, System, Help. The toolbar has various icons for file operations and debugger functions. The status bar shows: ZCU\_ABAP\_HINTERGRUND\_DEBUGGEN / ZCU\_ABAP\_HINTERGRUND\_DEBUGGEN / 65 SY-SUBRC 0 EVENT / START-OF-SELECTION. The code editor displays the following ABAP code:

```

60
61   * Endlessloop for debugging of programs,
62   * who run in background task
63   gv_exit = space.
64
65   DO.
66   IF gv_exit = 'X'.
67     EXIT.
68   ENDIF.
69   ENDDO.
70

```

Figure 3.36: Debugging background jobs with endless loops

To start the analysis of the program, move to the code line “IF gv\_exit = X”. With a double click on the variable “GV\_EXIT,” the variables appear in the debugger sub-screen for display variables (see Figure 3.37).

The screenshot shows the 'Variables 1' tab of the debugger. The table lists variables with their current values and hexadecimal representations. The 'GV\_EXIT' variable is highlighted.

St...	Variable	Va...	Val.	Ch...	Hexadecima...
	GV_EXIT		X		58

Figure 3.37: End the endless loop with the debugger

Change the value of the variable to “X” by clicking the icon in the line of the variable “GV\_EXIT.”

Going back to the program run code of the debugger, continue reviewing the program by using single steps. The program course comes to the exit command in the DO loop and continues behind the ENDDO command. Now you can analyze every program step by using the debugger.

### **3.9.4 System command “=dbug”**

Some standard SAP programs start in the background directly after parameters are selected. You can only debug such a program by using the enhancement technique.

For standard SAP program exits, there are some tricky, seldom-documented ways to debug background processes. Some programs in the contract current account, e.g., for the mass simulation FI-CA, can cause a reaction from the command “=DBUG” if the command is entered before starting the program. After using this command, the program doesn’t proceed as normal in the background, but online with debug-mode.

For example, use **TRANSACTION** EAMS01 to process a large number of invoices for energy consumption contract accounts (see Figure 3.38 ).

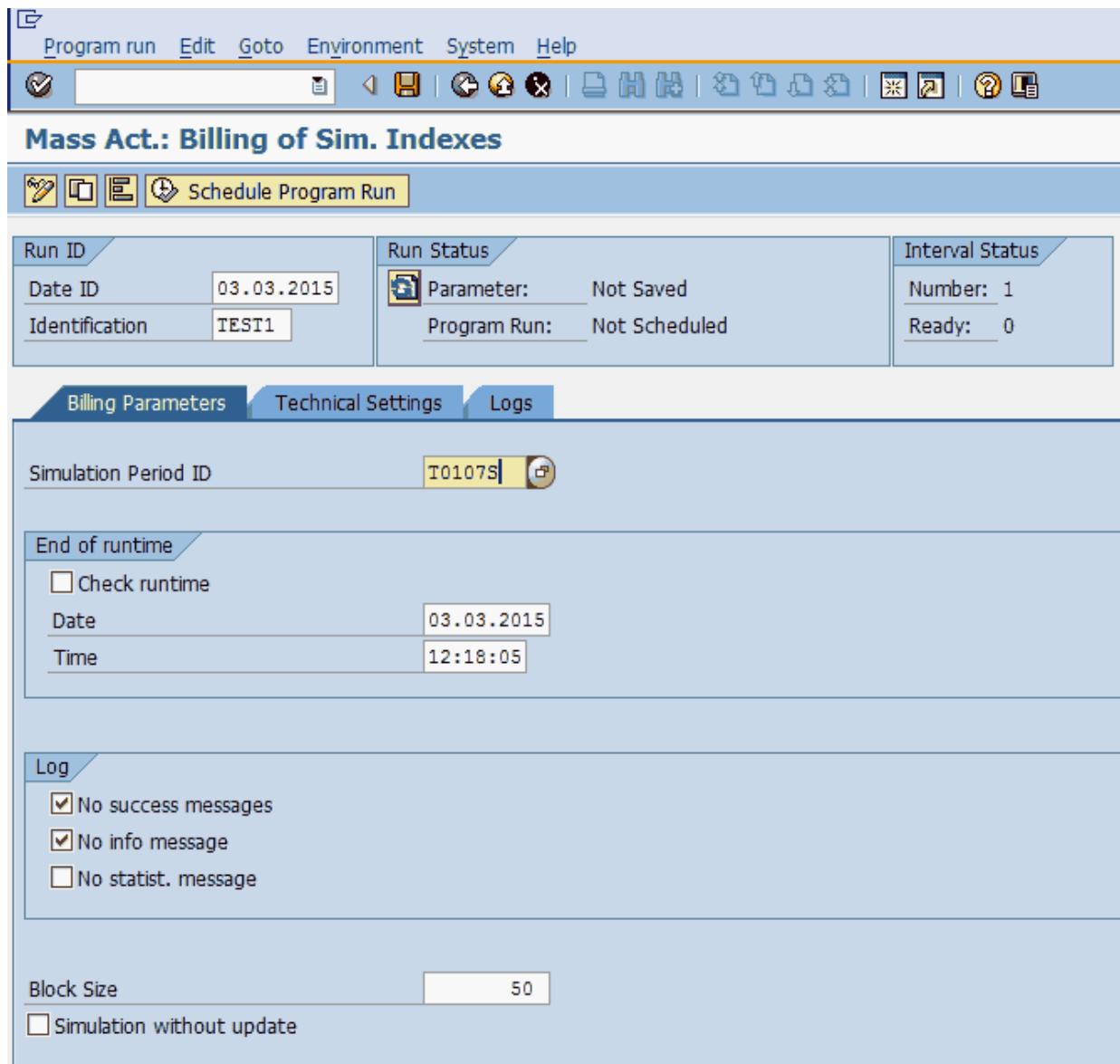


Figure 3.38: Start an account simulation with transaction EAMS01

After entering the selection parameters, save the parameters for the program run by clicking the icon. Then enter the command =dbug in the command line.

The message “Activity run started in debugging mode” appears in a popup screen.

The debugger starts when you start the program run by clicking the button (see Figure 3.39).

The screenshot shows the SAP ABAP Debugger interface. The title bar reads "ABAP Debugger(1) (Exclusive)(R3DCUCI\_DCU\_00)". The menu bar includes Debugger, Edit, Goto, Breakpoints, Settings, Miscellaneous, System, and Help. The toolbar has icons for various debugger functions. The main area displays the ABAP code for the program "RFKK\_MASS\_ACT\_SINGLE\_JOB". The code is as follows:

```

39  ****
40  START-OF-SELECTION.
41
42  * stop for debugging if desired.
43  IF p_xdebg NE space.
44    SET EXTENDED CHECK OFF.
45    BREAK-POINT.
46    SET EXTENDED CHECK ON.
47  ENDIF.
48
49  * for SAP Central Process Scheduling by Redwood
50  IF one_itvl = 'X'.
51
52  IF p_jobcnt = 0 AND sy-batch NE space.
53    CALL FUNCTION 'GET_JOB_RUNTIME_INFO'
54      IMPORTING
55        jobcount      = h_jobcount
56        jobname       = h_jobname
57      EXCEPTIONS
58        no_runtime_info = 1
59        OTHERS          = 2.
60    p_jobcnt = h_jobcount.
61    p_jobnam = h_jobname.
62  ENDIF.
63
64  ENDIF.
65

```

Figure 3.39: Debugger in a standard SAP program that normally starts in the background

The program course can be debugged and can proceed as normal.

### 3.9.5 Debugging methods of BOR objects

SAP provides business objects and methods through the *business object repository (BOR)*. Initially, the BOR was developed for the SAP workflow technology. Today, the BOR is used for saving business object types and their *business application programming interfaces (BAPIs*) (business application programming interfaces), as a functional interface for the archive (*archive link*), and for the control of communication and other generic object services.

A BOR object consists of descriptive data and the methods to work with this data (*capsulation*). The access to the data of an object and any changes are only possible with defined BOR object methods, also called *BAPIs*. The BOR objects can be seen as an early form of object orientation.

For example, a business object exists with the name “ISUPARTNER” for “business partner.” With the help of **TRANSACTION SW01**, BOR objects are created, displayed, and changed (see Figure 3.40).

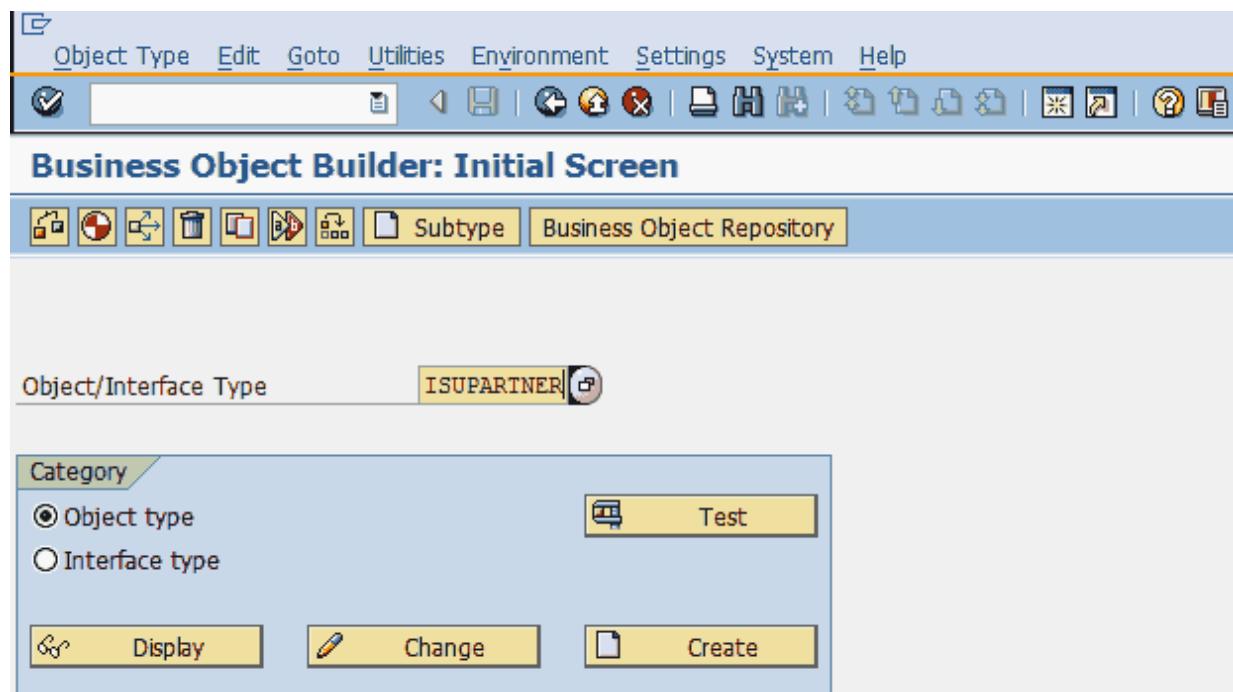


Figure 3.40: BOR objects with transaction SW01

By clicking the **Display** button, the components of a BOR object are displayed (see Figure 3.41).

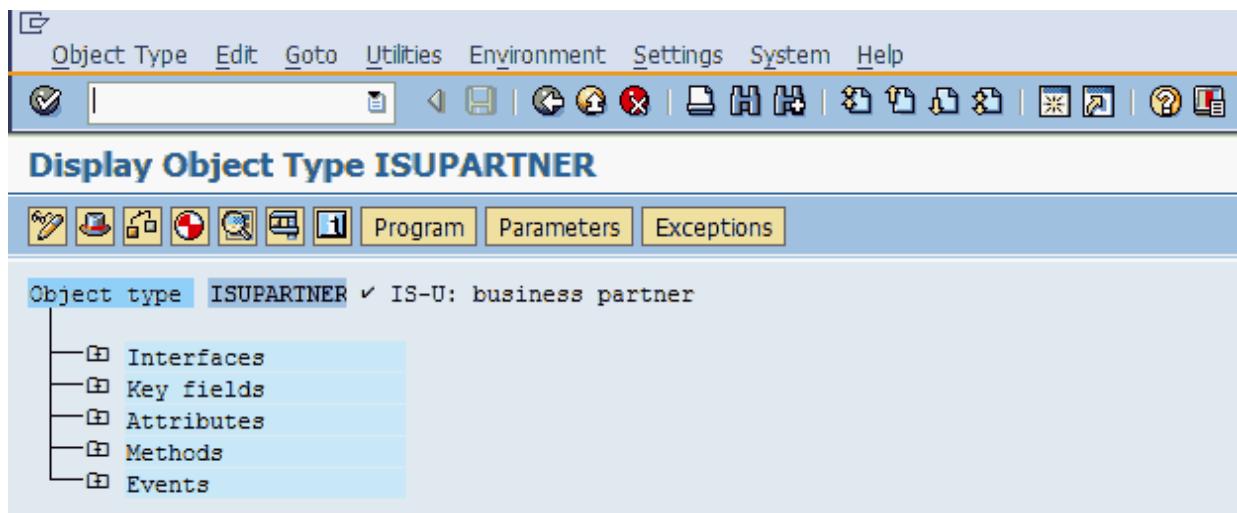


Figure 3.41: Overview of a BOR object

With a mouse click on the plus sign in front of **METHODS**, you can display all methods for BOR object “business partner” (see Figure 3.42).

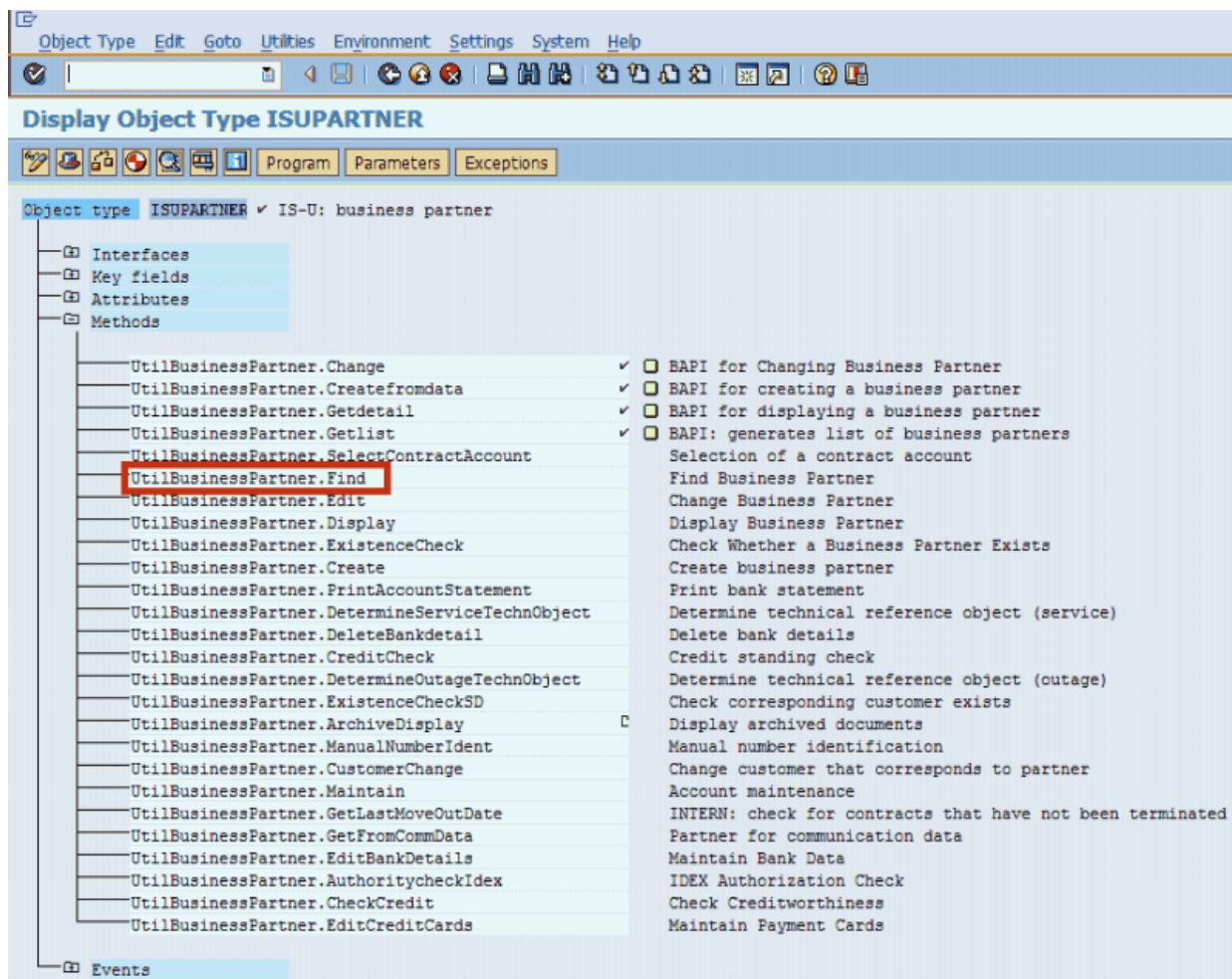


Figure 3.42: Methods of a BOR object

ABAP developers are often confronted with testing such methods. For example, take the method “UtilBusinessPartner.Find,” which searches for a selected business partner. Put the cursor on this method and click the **TEST** icon to display a screen for testing the BOR method (see Figure 3.43).

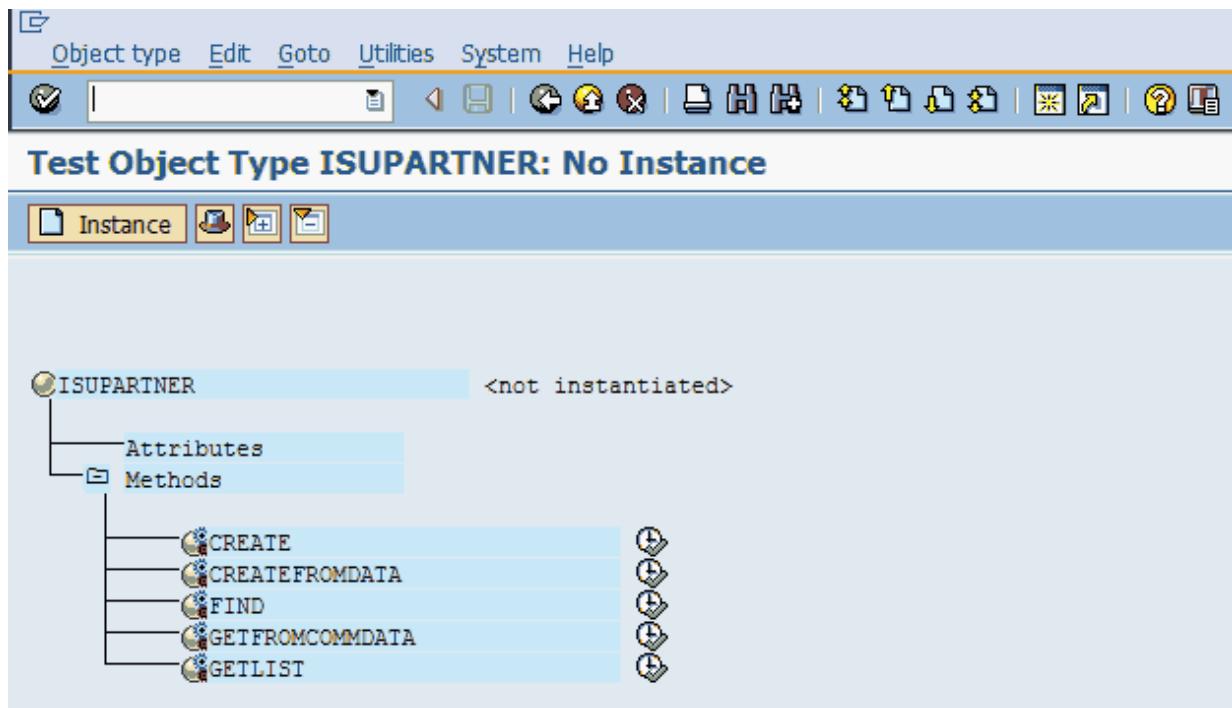


Figure 3.43: Testing methods of a BOR object

To test the method “Find,” you first need an object instance. By clicking the **Instance** button, a BOR object is created.

As shown in Figure 3.44, you have to input a **PARTNER NUMBER**.

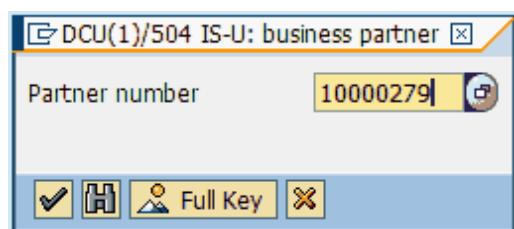


Figure 3.44: Create a BOR object business partner

The choice of a business partner number is confirmed by clicking the  icon. In the next screen, different methods for testing are displayed (see Figure 3.45 )

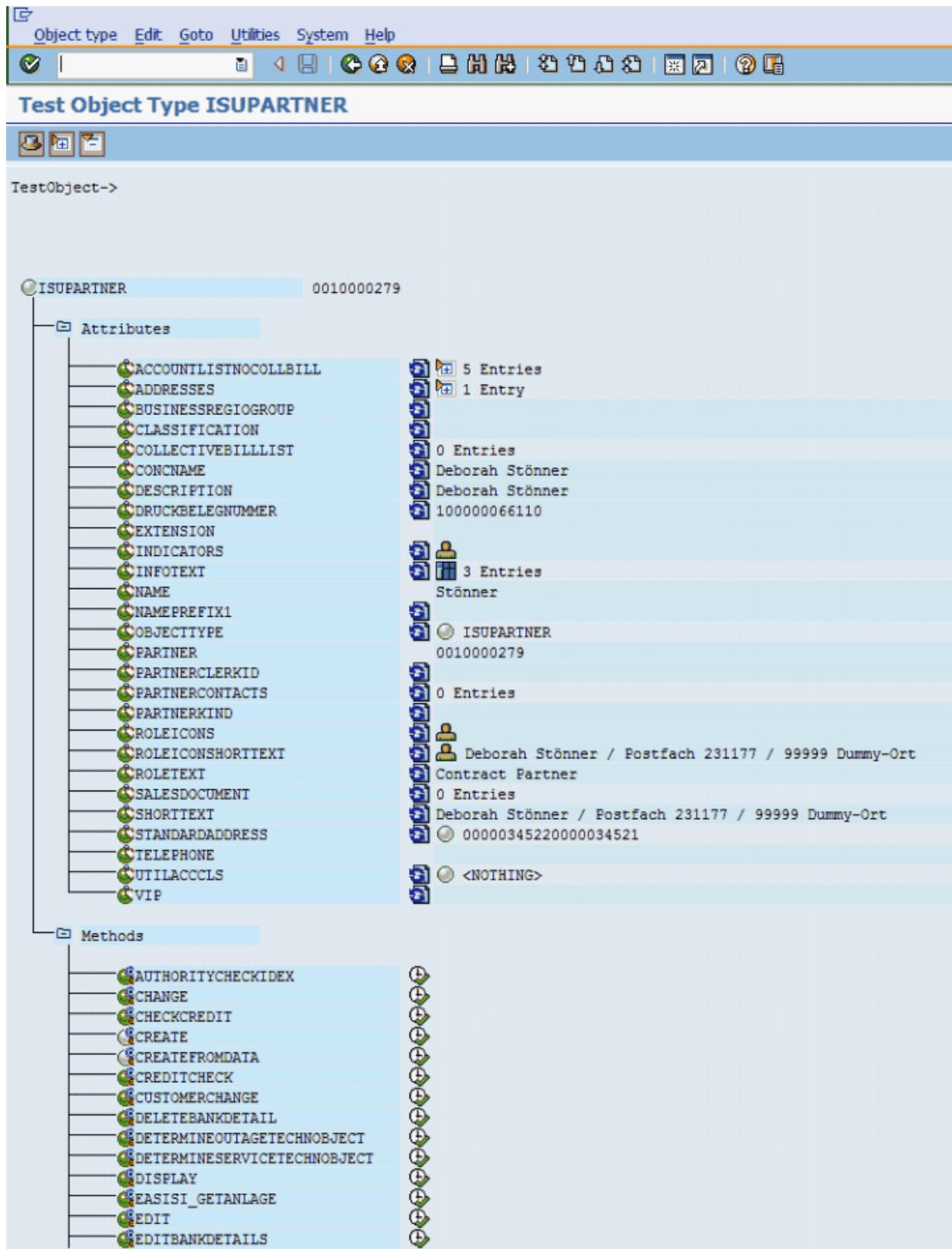


Figure 3.45: Testable BOR methods after creating a BOR object instance

The method “EDIT” can be tested by clicking the  icon beside this method. You see the screen as displayed in Figure 3.46. Now you can analyze and test the method.

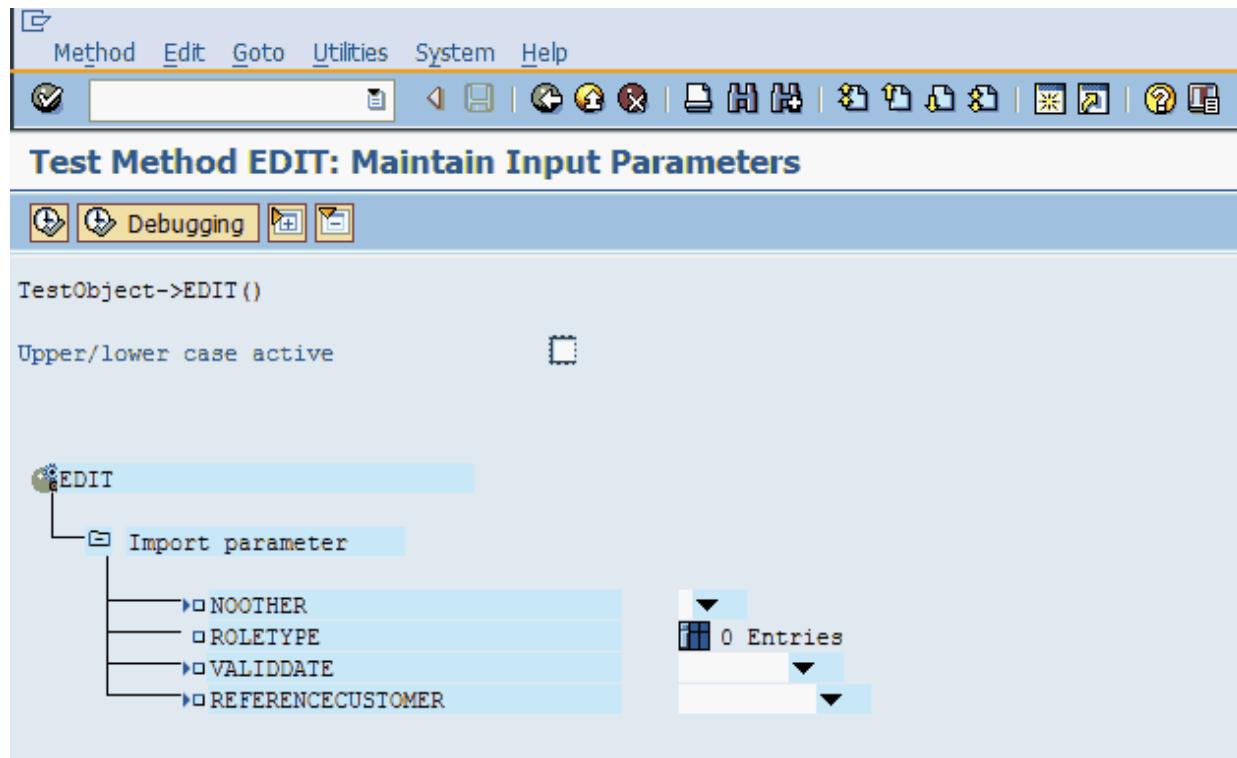


Figure 3.46: Test of a BOR method

By clicking the  button, the above selected business partner is displayed in change mode.

To use this method in debugging mode, click the  **Debugging** button. The program steps of the method “Edit” can be analyzed with the ABAP debugger (see Figure 3.47).

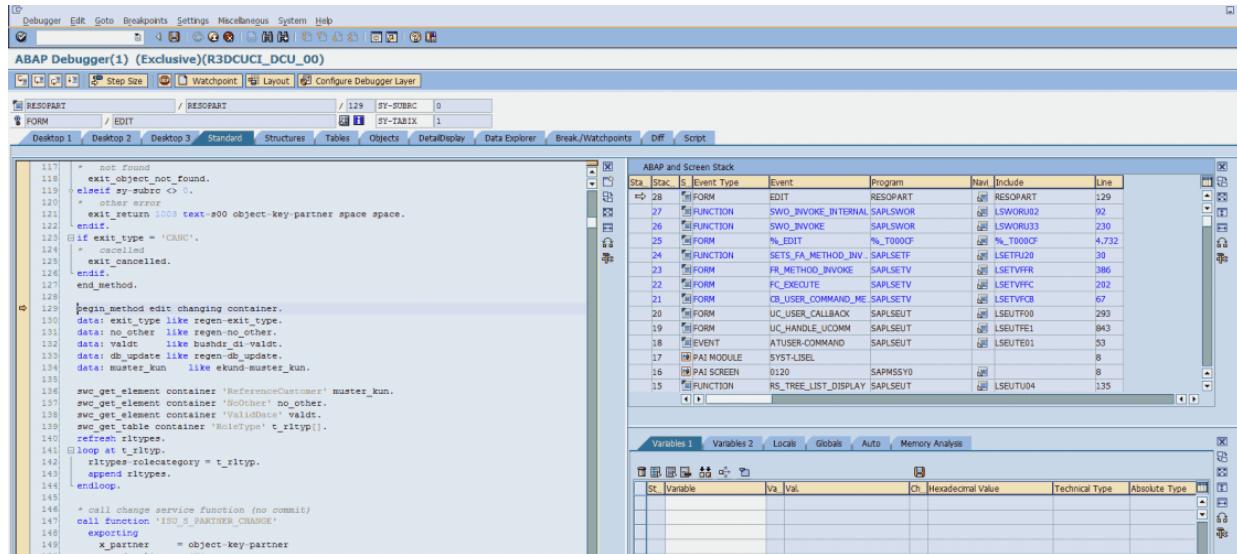


Figure 3.47: Start of the debugger for testing a BOR method

### 3.9.6 Enhancements in the debugger

Be careful if you have to debug standard SAP programs with enhancements. As an example, use a test enhancement at the end of the function module ISU\_O\_INSTLN\_OPEN. This function module displays an installation with **TRANSACTION** ES32. Before displaying the installation, set a breakpoint in the function module ISU\_O\_INSTLN\_OPEN. Then call **TRANSACTION** ES32 with an arbitrary installation. The debugger stops the program run, as expected, at the breakpoint (see Figure 3.48).

The screenshot shows the ABAP Debugger interface with the title "ABAP Debugger(1) (Exclusive)(R3DCUCI\_DCU\_00)". The code editor displays the following ABAP code:

```

172     y_wmode = y_obj->contr-wmode.
173
174     * return table of timeslices
175     IF yt_timeslice IS REQUESTED.
176         CLEAR: yt_timeslice, yt_timeslice[].
177         LOOP AT y_obj->eanlh INTO weanh.
178             MOVE-CORRESPONDING weanh TO yt_timeslice.
179             APPEND yt_timeslice.
180         ENDLOOP.
181     ENDIF.
182     * refresh of master data statistic
183     CALL FUNCTION 'ISU_MASTER_DATA_STATS_REFRESH'.
184     *    EXCEPTIONS
185     *        OTHERS    = 1.
186
187
188 ENDFUNCTION.

```

Line 188 is highlighted with a red box around the entire line, and there is a small icon resembling a snail shell in front of the line number.

Figure 3.48: Display of an enhancement in the debugger

Look at the first column. At the end of the function module – in front of line 188 – you see an icon that looks like a little snail .

This icon means there is an enhancement implemented before the end of the function module. If the current program course (yellow arrow in the first column) is exactly at this place, the yellow arrow covers the icon of the snail (see Figure 3.49).

The screenshot shows the ABAP Debugger interface with the title "ABAP Debugger(1) (Exclusive)(R3DCUCI\_DCU\_00)". The code editor displays the same ABAP code as Figure 3.48, but with a yellow arrow pointing to the line number 188. The yellow arrow covers the small snail icon in front of the line number.

Figure 3.49: ABAP debugger in front of an enhancement

Enter enhancement-only mode by clicking on the  icon for a single step.

As shown in Figure 3.49, look at the first column of the debugger to find the enhancement locations. This is especially true, if the snail symbol is hidden through the sign of the current program step (symbol ).

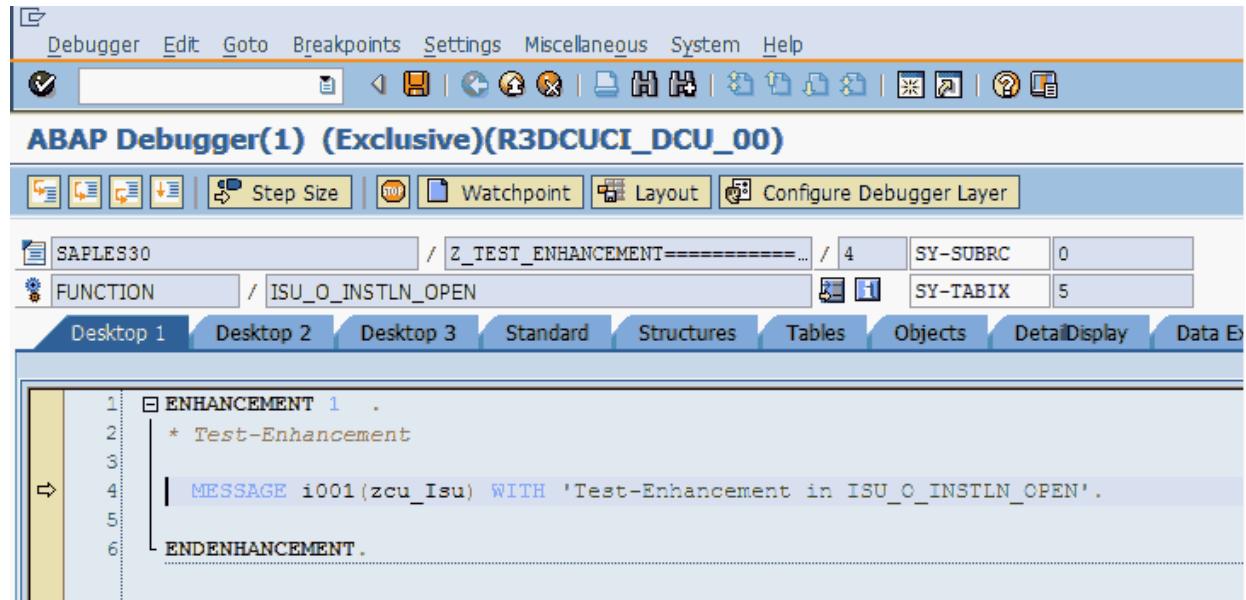


Figure 3.50: ABAP debugger in the enhancement

# **4 ABAP development**

**In this chapter, I want to explain special and often-needed procedures and methods of ABAP development. Knowledge of the ABAP programming language and the programming logic in report and dialog processing is required.**

## 4.1 Documenting programs

Customers of IT service providers sometimes pay a considerable sum for program development. In return, the developer has the task of documenting the program in a way that allows inexperienced users to understand which inputs a program requires, what function(s) the program has, and what results a program will produce.

In addition, the developer has the obligation to develop the program in such a way that it is readable and useful for other developers. To explain, let's take the example of the developed program ZCU\_FIRST\_CUSTOMER\_KONTOKLASSE.

To document a program for a user, it is important to include:

- ▶ Program title
- ▶ Meaningful structure of the selection screen
- ▶ Comprehensive description of input parameters in the selection screen
- ▶ Online program documentation
- ▶ Helpful information in error messages

The **program title** is created in the ABAP editor with the help of the pull-down menu **Go to • ATTRIBUTES** (see Figure 4.1).

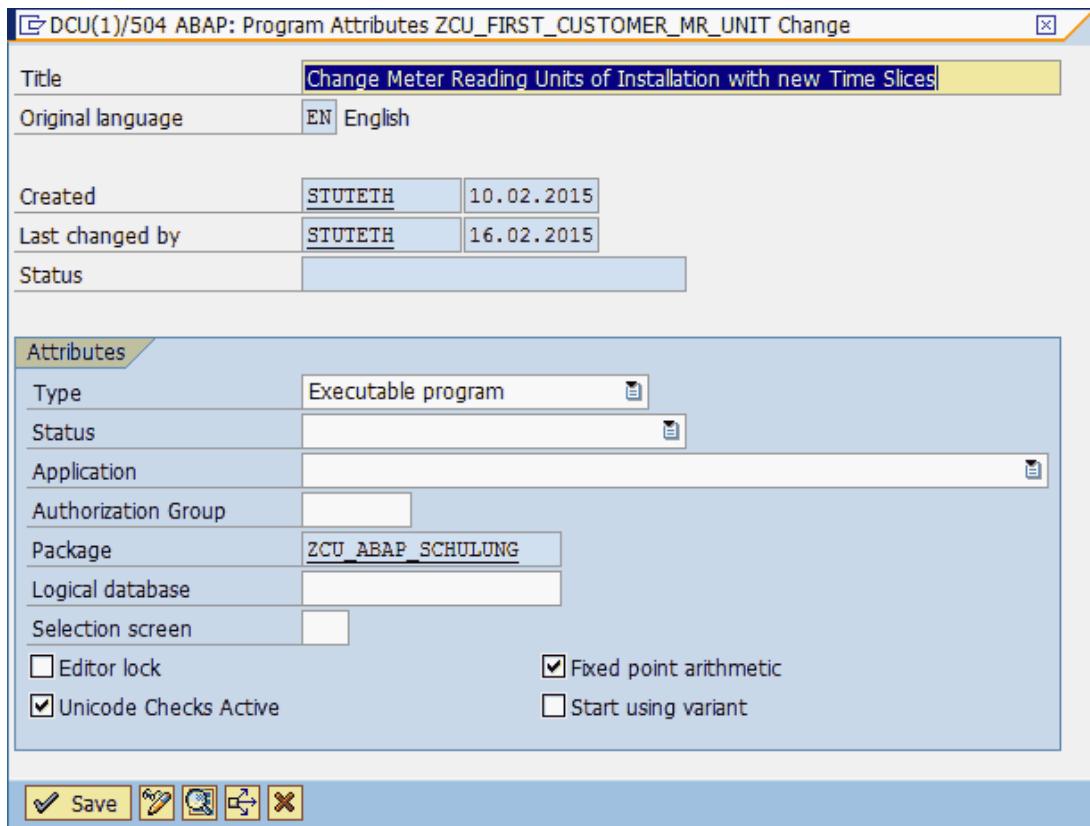


Figure 4.1: Creating a program title in the ABAP editor

Figure 4.2 shows the program title in the selection screen.

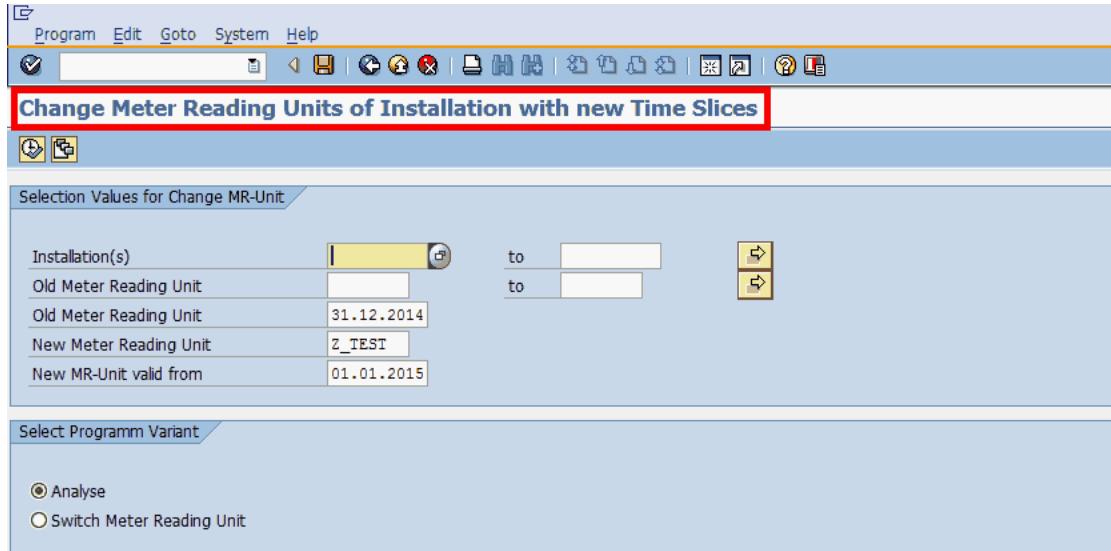


Figure 4.2: Program title in the selection screen

An appropriate selection screen structure varies and has multiple input areas with headings. Sample coding for a selection screen is displayed in Figure 4.3.

```

*-----#
* SELECTION-SCREEN
*-----#
SELECTION-SCREEN BEGIN OF BLOCK se1 WITH FRAME TITLE text-se1.
  SELECTION-SCREEN SKIP.
    SELECT-OPTIONS: so_anlag FOR eanh-anlage OBLIGATORY.
    SELECT-OPTIONS: so_ablal FOR eanh-ableinh.                                     "Old MR-Unit
    PARAMETERS:   pa_datb  TYPE dats OBLIGATORY DEFAULT '20141231'. "Old MR-Unit valid to
    PARAMETERS:   pa_ablne LIKE te422-termschl OBLIGATORY DEFAULT 'Z_TEST'. "New MR-Unit
    PARAMETERS:   pa_data  TYPE dats OBLIGATORY DEFAULT '20150101'. "New MR-Unit valid from
  SELECTION-SCREEN END OF BLOCK se1.
SELECTION-SCREEN BEGIN OF BLOCK se2 WITH FRAME TITLE text-se2.
  SELECTION-SCREEN SKIP.
    PARAMETERS:   pa_anal  RADIOBUTTON GROUP aus1 DEFAULT 'X'.
    PARAMETERS:   pa_swit  RADIOBUTTON GROUP aus1 .
  SELECTION-SCREEN END OF BLOCK se2.
*-----#

```

Figure 4.3: Coding for the selection screen

To input the description of **input parameter**, use the menu **GOTO • TEXT ELEMENTS • SELECTION TEXTS** in the ABAP editor.

Name	Text	Dictionary r...
PA_ABLNE	New Meter Reading Unit	<input type="checkbox"/>
PA_ANAL	Analyse	<input checked="" type="checkbox"/>
PA_DATA	New MR-Unit valid from	<input type="checkbox"/>
PA_DATB	Old Meter Reading Unit	<input type="checkbox"/>
PA_SWIT	Switch Meter Reading Unit	<input checked="" type="checkbox"/>
SO_ABLAL	Old Meter Reading Unit	<input type="checkbox"/>
SO_ANLAG	Installation(s)	<input type="checkbox"/>

Figure 4.4: Input the description for selection parameters

The title of the input areas in the selection screen are created by using the pull-down menu **Go to • TEXT ELEMENTS • TEXT SYMBOLS** (see Figure 4.5).

Sym	Text	Lngth	Max.
SE1	Selection Values for Change MR-Unit	35	60
SE2	Select Programm Variant	23	40

Figure 4.5: Input of text symbols

The **online program documentation** is created within the change mode of the ABAP editor with the menu **Go to • DOCUMENTATION**. By using this menu, another

editor screen opens that allows you to input the online documentation. The following is the minimum amount of information to include:

- ▶ Program task: short description of the program functions.
- ▶ Input parameters: short description of all selection parameters and noting which parameters are optional.
- ▶ Output parameters: all output values of the program.
- ▶ Remarks: tips for using the program, such as conditions for the program run or technical or organizational preliminary work.
- ▶ Information about the development: author of the program and date of the development.

Often, IT service providers create a detailed description for online documentation input.

The user can access the online program documentation by clicking the  icon on the selection screen (see Figure 4.6).

#### Enable program documentation



The developer must make sure that the last edited version of the online documentation has been activated by clicking the  icon; otherwise the documentation is visible only in the ABAP editor and is inaccessible to the user because the  icon doesn't exist on the selection screen.

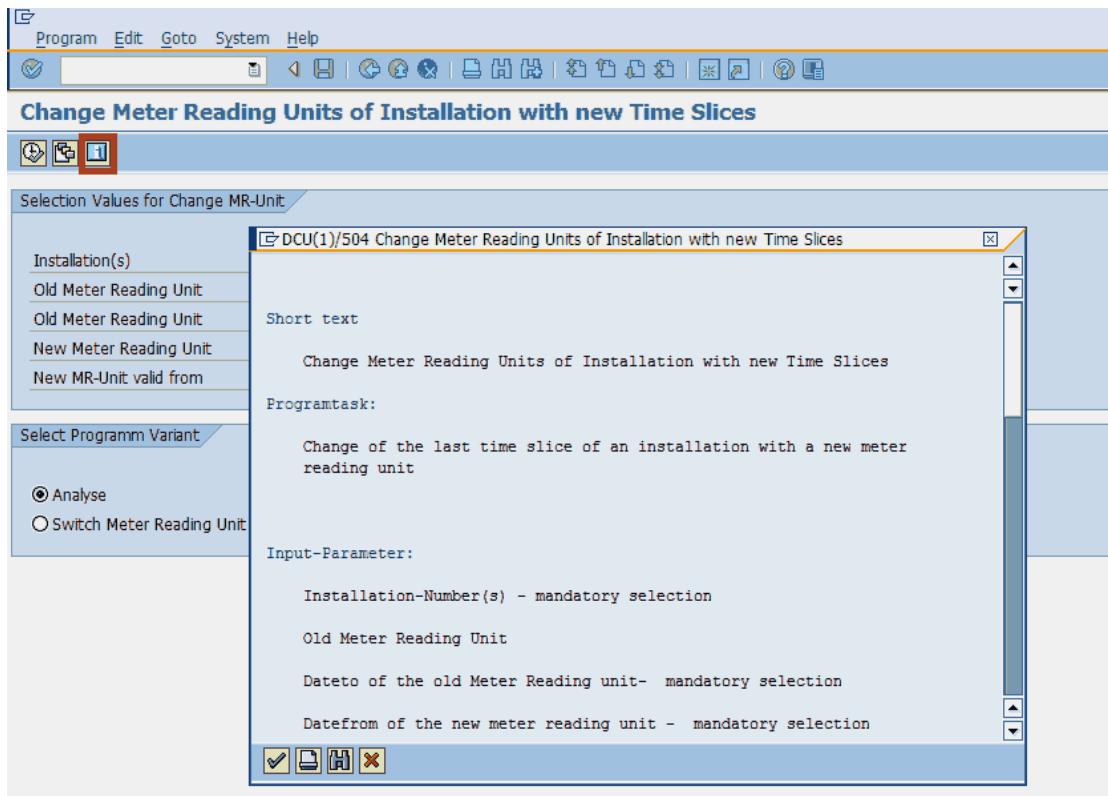


Figure 4.6: Online documentation in the selection screen

For subsequent maintenance, include the following in the program documentation:

- ▶ A functional and process description in the header of the program, as well as information about the developer and the date of the development, the change history with the reason(s) for change(s), the person making the change(s), and the date of change(s).
- ▶ Readable structure of the program: Classification of individual functions in subroutines and function modules.
- ▶ Use of the *pretty printers* in the ABAP editor with defined settings, e.g., keywords (like ABAP commands) in capital letters and other words and terms (like variable names) in lowercase letters.
- ▶ Inline documentation: comment lines in the coding about the function which has to be processed in the next lines, where a jump to a subroutine is necessary, the functions of form routines, the input and output parameters in the header of routines, and so on.

## 4.2 Structure and readability of programs

Customer programs generally are used for a long time, so it is probable that these programs will have to be enhanced or changed. The changes or enhancements could be necessary after a short time or after some months or years.

Often, the program developer can't remember the details after a long period. Therefore, it is important to design a program in a structured and readable form.

Customer programs can be structured in a simple form. The main program controls a row of subroutines, function modules, or methods, which contain the real programming logic.

To illustrate this, use the example program from [Chapter 1](#) for the adjustment of meter reading units of installations. The program has to execute the following steps:

1. Check selection parameters
2. Select installation time slices
3. Analyze installation time slices
4. Output the analysis results in the analyze program run
5. Adjust installation time slices in the update program run on the base of analysis
6. Output the updated program run

You see this clearly structured way of proceeding in the main program as shown in Figure 4.7.

### Expressive names in programs



In ABAP, you can use a maximum of 30 letters (including spaces) for the naming of form routines, function modules, and methods. Use this property to give names to the processing blocks. Cryptology for the name of subroutines, function modules, and methods is senseless as another developer will not be able to understand (decipher) the coding.

```

* Selection of installations
PERFORM selection_installation_timesli.

* Find out installations, which have to switch
PERFORM analyse_time_slices.

IF pa_anal = 'X'.

* Output of analyse processing
PERFORM output_analyse.

ELSEIF pa_swit = 'X'.

* Switch meter reading unit
PERFORM switch_meter_reading_unit.

* Output of switch processing
PERFORM output_switch.

ENDIF.

```

Figure 4.7: Understandable program structure

As shown in the coding for the selection screen in Figure 4.8, the ABAP commands are written in capital letters and all other words in lowercase letters.

```

*-----*
* SELECTION-SCREEN
*-----*
SELECTION-SCREEN BEGIN OF BLOCK sel1 WITH FRAME TITLE text-se1.
  SELECTION-SCREEN SKIP.
  SELECT-OPTIONS: so_anlag FOR eanlh-anlage OBLIGATORY.
  SELECT-OPTIONS: so_abal FOR eanlh-ableinh.          "Old MR-Unit
  PARAMETERS:    pa_datb TYPE dats      OBLIGATORY DEFAULT '20141231'. "Old MR-Unit valid to
  PARAMETERS:    pa_ablnr LIKE te422-termschl OBLIGATORY DEFAULT 'Z_TEST'.   "New MR-Unit
  PARAMETERS:    pa_data TYPE dats      OBLIGATORY DEFAULT '20150101'. "New MR-Unit valid from
SELECTION-SCREEN END OF BLOCK sel1.
SELECTION-SCREEN BEGIN OF BLOCK se2 WITH FRAME TITLE text-se2.
  SELECTION-SCREEN SKIP.
  PARAMETERS:    pa_anal  RADIOBUTTON GROUP ausl DEFAULT 'X'.
  PARAMETERS:    pa_swit  RADIOBUTTON GROUP ausl .
SELECTION-SCREEN END OF BLOCK se2.
*-----*

```

Figure 4.8: Coding for the selection screen

All commands are indented so you get a clear general view of the program.

When moving source code and using capital and lowercase letters, you can use the pretty printer.

The developer has to set the features of the pretty printer once in an SAP system by using the pull-down menu **UTILITIES • SETTINGS**.

You will find the user-specific settings for the pretty printer under the tab **ABAP EDITOR** in the tab **Pretty Printer** (see Figure 4.9).

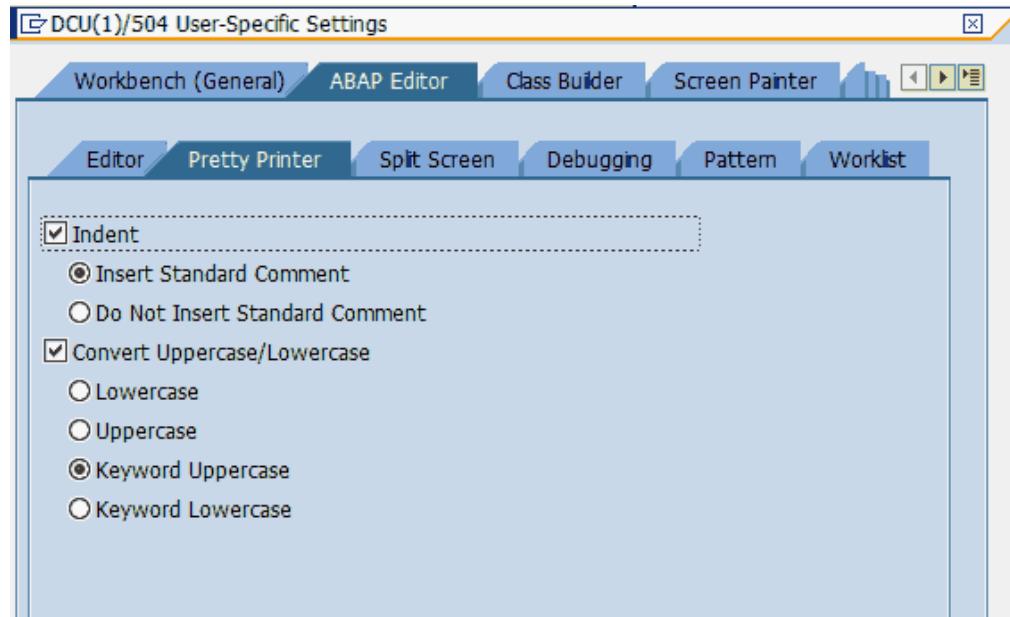


Figure 4.9: Settings for the pretty printer

After a program code change, the main program can look like the one shown in Figure 4.10.

```
* Selection of installations
    PERFORM SELECTION_installation_timeSLI.

* Find out installations, which have to switch
  Perform analyse_time_slices.

  IF pa_anal = 'X'.

*   Output of analyse processing
    PERFORM OUTPUT_Analyse.

ELSEIF pa_swit = 'X'.

*   Switch meter reading unit
    PERFORM switch_METER_READing_unit.

*   Output of switch processing
    PERFORM output_SWITCH.

ENDIF.
```

Figure 4.10: Source code without using the pretty printer

With the settings of the pretty printer shown above, you only have to click the **Pretty Printer** button once to get the coding of the main program as shown in Figure 4.7.

### Activate the program after using the pretty printer



Remember: Activate the program after using the pretty printer by clicking on the icon.

## 4.3 Error handling in programs

Besides good program documentation, error handling in customer development is a central element for user friendliness. As a developer, take the time to build comprehensible error handling into applications for the user. (Remember to take account of the time required so you can include it in the cost estimate for developing.)

Start by building the error handling message directly after developing each necessary function. There are different ways to announce errors to the user. The different kinds of errors include:

- ▶ Input error
- ▶ Missing customizing
- ▶ Faulty selections
- ▶ Feedback of error messages from function modules
- ▶ Feedback of exceptions from methods

Before successful error handling begins, consider all possible errors and develop each corresponding reaction.

To announce program run errors to the user, you can create:

- ▶ A directly reactive error message
- ▶ An output of the error after the program run

The *MESSAGE* command is the simplest way to handle error handling messages. As an example, use the *SELECT* command, as shown in Listing 4.1.

```
SELECT *
  FROM eanlh
 INTO TABLE gt_output
 WHERE anlage    IN so_anlag
   AND bis       = pa_bis
   AND akklasse  = pa_aklas
   AND ableinh   = pa_ablei
   AND billing_party = pa_billp.
```

ENDIF.

IF sy-subrc <> 0.

```
MESSAGE e000(zcu_isu).
ENDIF.
```

*Listing 4.1: Using the MESSAGE command*

The message class ZCU\_ISU contains the message number 000 after the text: “No Records found for the given Selection Criteria”.

The MESSAGE command has the following form:

```
MESSAGE <error type><error number>
(<Message Class>)
WITH <Message_Variable1>
      <Message_Variable2>
      <Message_Variable3>
      <Message_Variable4>
RAISING <Exception>.
```

*Listing 4.2: The MESSAGE command*

The option *RAISING* is needed in function modules and methods, which own exceptions as calling parameters. The corresponding calling program reacts depending on the released exception.

For the output of an error message, each of the four message lines can contain a maximum of 50 letters. If you need more than 200 letters to describe the error and the expected user action, use the long form of an error message.

When you double click on a message number in the ABAP editor, the message class with the error messages is shown in a new screen (see Figure 4.11).

Message class **ZCU\_ISU** Actv.

Attributes Messages

Message	Message Short Text	Self-Explanatory
100	No Data found by this selection criteria.	<input checked="" type="checkbox"/>
101		<input checked="" type="checkbox"/>
102		<input checked="" type="checkbox"/>
103		<input checked="" type="checkbox"/>
104		<input checked="" type="checkbox"/>
105	The action was canceled by the user.	<input checked="" type="checkbox"/>
106	The selected new meter reading unit is not in Table TE422.	<input checked="" type="checkbox"/>
107	No time slices for installation were found by this selection criteria.	<input checked="" type="checkbox"/>
108	Error by build up the <b>s</b> .	<input checked="" type="checkbox"/>
109	The new account class is not in table KONTOKLASSE. Termination.	<input checked="" type="checkbox"/>
110	No contract accounts were found by this selection criteria.	<input checked="" type="checkbox"/>
111	<b>Insufficient Customizing</b>	<input type="checkbox"/>
112	Selected Datefrom and Dateto generate a time gap! Termination !	<input checked="" type="checkbox"/>
113	Selected Datefrom and Dateto generate a time superposition! Termination !	<input checked="" type="checkbox"/>
114		<input checked="" type="checkbox"/>

Figure 4.11: Long text for the MESSAGE command

In the last column of each error message, the box in the **SELF-EXPLANATORY** column is selected. For error **MESSAGE** with number 111, the mark has been removed (see Figure 4.11), enabling use of the long form for the error message.

Click the **Long Text** button in change mode. Use the editor to add the long error text. You can describe a detailed error message and enable detailed information for the expected user action (see Figure 4.12).

**Change Message: ZCU\_ISU111 Language DE**

**&CAUSE&**  
The selected meter reading unit is not in table TE422

**&WHAT\_TO\_DO&**  
Before the change can be made to the selected meter reading, the meter reading have to be customized in table TE422.  
Use transaction SPRO -> SAP Reference IMG -> SAP Utilities -> Basic Functions  
-> Portioning and Scheduling -> Define Meter Reading Units

Figure 4.12: Create the long form of an error message

The input text is saved by clicking the  icon and activated by clicking the  icon. Exit the editor by clicking the **BACK**  icon.

If the message will be printed on the event AT-SELECTION SCREEN, the following error message appears in the status line “Insufficient Customizing.” A mouse click on the error message shows the long version of the message.

The *message type* decides the next course of the program. In ABAP, the following message types are common.

- ▶ A = Abort message; popup with the error message. After closing the message, the program is terminated.
- ▶ E = Error message; output of the error message to the status line
- ▶ I = Information message; popup screen with the error message. After closing the screen, the program continues.
- ▶ S = Status message; green message output in the status line. After confirming the message, the program continues after hitting the “Enter” key.
- ▶ W = Warning message; yellow message output in the status line. After confirming the message, the program continues after hitting the “Enter” key.
- ▶ X = Exit message, this kind of message leads to a program break (dump) and therefore shouldn’t be used.

Every message type reacts to different development objects and different types of a program run (batch or dialog) in a different way. The MESSAGE command differentiates the following development objects

- ▶ List processing
- ▶ Dialog processing
- ▶ Control processing
- ▶ Batch input
- ▶ Background processing
- ▶ Booking
- ▶ Conversion modules
- ▶ Procedures

- ▶ Remote function call (RFC) processing
- ▶ HTTP server

Online documentation exists for the different reactions in the different development objects, which you can reach by hitting the “F1” help key for the MESSAGE command.

If a message will be printed out, the system parameters of the structure SYST has the following values:

- ▶ SY-MSGID = <message class> with data type CHAR20
- ▶ SY-MSGTY = <message type> with data type CHAR 1
- ▶ SY-MSGNO = <message number> with data type NUM 3
- ▶ SY\_MSGV1 = <message\_variable1> with data type CHAR50
- ▶ SY\_MSGV2 = <message\_variable2> with data type CHAR50
- ▶ SY\_MSGV3 = <message\_variable3> with data type CHAR50
- ▶ SY\_MSGV4 = <Message\_variable4> with data type CHAR 50

If a message is printed in a function module by using the RAISING option, you can read out this value of the system structure SYST to build a suitable error handling response.

#### Database table T100



The database table T100 owns a central role. It contains all messages of all SAP message classes.

There are several T100\* tables that display messages and control the display. The most important tables are T100S and T100C (see Figure 4.13).

Table Name	Short text
T100	Messages
T100A	Message IDs for T100
T100C	Message Control by User
T1000	Assignment of message to object
T100S	Configurable system messages
T100SA	Application Areas for Configurable Message
T100T	Table T100A text
T100U	Last person to change messages
T100V	Assignment of messages to tables/views
T100VV	T100A and T100
T100W	Assign Messages to Workflow
T100X	Error Messages: Supplements

Figure 4.13: T100\* tables for message maintenance

Table T100S contains system messages and describes whether a message is switchable for all users or only for selected users.

With the help of table T100C, you can turn the single messages switch off for all users or for selected users and you can change the message type.

An error message can be switched from type “E” (error) to a message of type “W” (warning) to avoid aborting the program. The user can confirm the warning and the program continues.

Depending of the application area (table T100SA), you can change the settings of messages and their controls by using the maintenance view V\_T100C in **TRANSACTION** SM30. These changes also affect the behavior of standard SAP programs.

There are many options in ABAP to build suitable error handling. The methods explained above are sufficient for practice and to give users the necessary help in error cases.

## 4.4 Extended check of programs

The *extended program check "Program:Extended Check"* in the ABAP editor is another useful tool for all programs.

The extended program check uncovers the most inconsistencies even if they are syntactically correct. Not all checks are practical every time, but most of them can recognize program errors, which should be removed before going live. The extended program check doesn't guarantee that a program runs properly because the check of the correct type of parameters for the call function is missing. If the type of parameters between the calling program and the parameters of the function module differ, the program crashes (dumps).

For example, take the program from [Chapter 1](#) for creating customer contacts ZCU\_FIRST\_CUSTOMER\_CONTACT.

With a right mouse click on the program name, you can start the extended program check within **TRANSACTION** SE80 (see Figure 4.14).

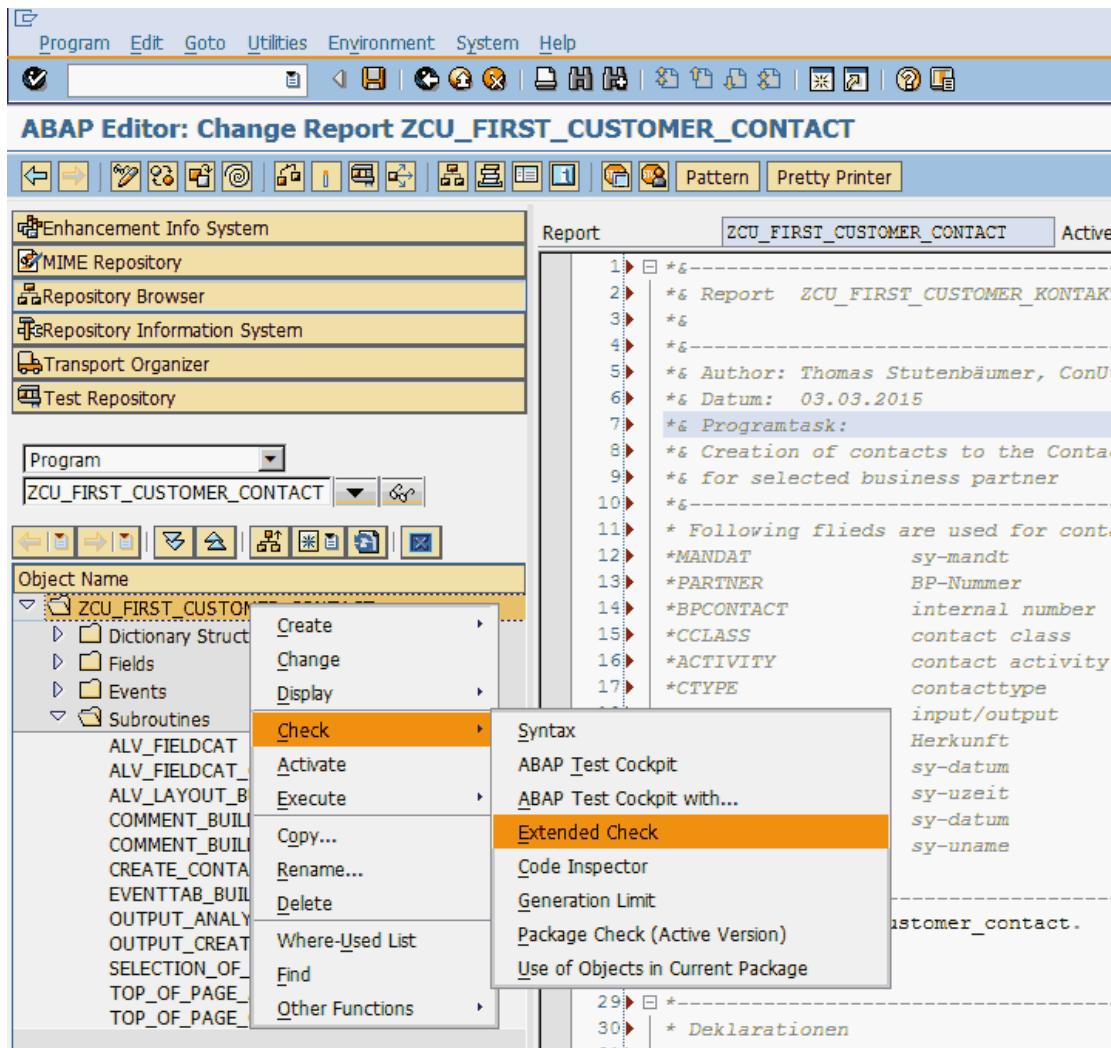


Figure 4.14: Call the extended program check

Figure 4.15 shows the different options for program checks.

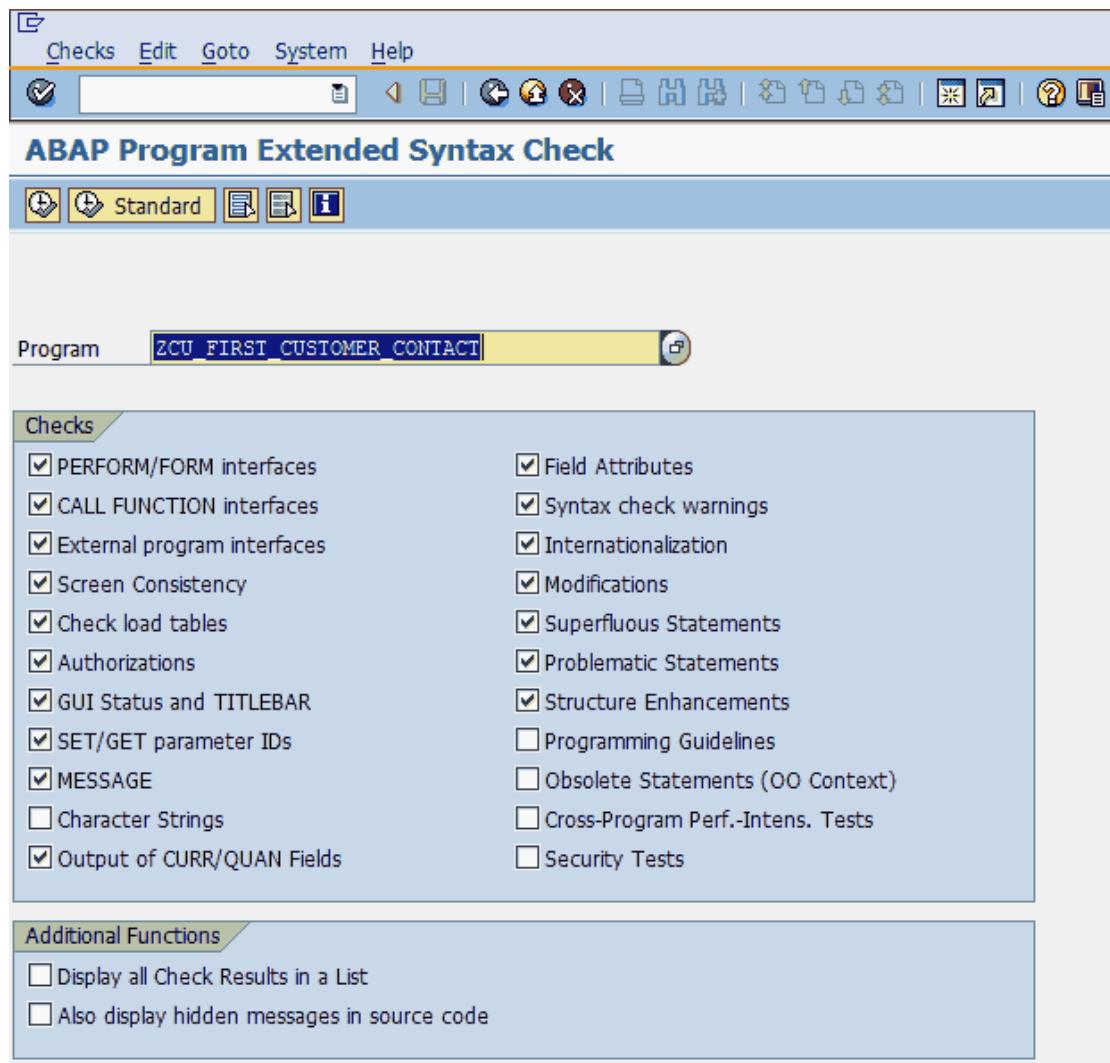


Figure 4.15: Different options for the extended program check

To demonstrate the extended program check, I've built some errors into the source code of the program. After that, I start the extended check. The program check shows one error, seven warnings, two hidden warnings, and no further messages (see Figure 4.16).

The screenshot shows the SAP SLIN Overview interface. The menu bar includes Checks, Edit, Goto, System, and Help. The toolbar contains various icons for navigation and system functions. The main title is "SLIN Overview". Below it are three buttons: "Display Results" (highlighted in yellow), "Display All Results", and "Display Single Test". The central part of the screen is a table with the following data:

Check for Program ZCU_FIRST_CUSTOMER_CONTACT	Error	Warnings	Messages
Test Environment	0	0	0
PERFORM/FORM Interfaces	0	0	0
CALL FUNCTION Interfaces	1	1	0
External Program Interfaces	0	0	0
Dynpro Consistency	0	0	0
Authorizations	0	0	0
GUI Status and TITLEBAR	0	0	0
SET/GET Parameter IDs	0	0	0
MESSAGE	0	0	0
Output of CURR/QUAN Fields	0	0	0
Field Attributes	0	3	0
Superfluous Statements	0	0	0
Syntax Check Warnings	0	2	0
Modifications	0	0	0
Check on Load Sizes	0	0	0
Internationalization	0	0	0
Problematic Statements	0	0	0
Structure Enhancements	0	0	0
Hidden Errors and Warnings	0	2	0

Figure 4.16: Output of the extended program check

With a double click on the number of analyzed errors, the system shows further information for the errors or warnings. In the following example, I've double clicked on the analyzed error message (see Figure 4.17).

The screenshot shows the SAP SLIN Overview interface. The menu bar includes Checks, Edit, Goto, System, and Help. The toolbar contains various icons for navigation and system functions. The main title is "SLIN Overview". Below it are two buttons: "Display" (highlighted in yellow) and "Change". The central part of the screen displays the following message:

```

Messages for CALL FUNCTION Interfaces(Error)

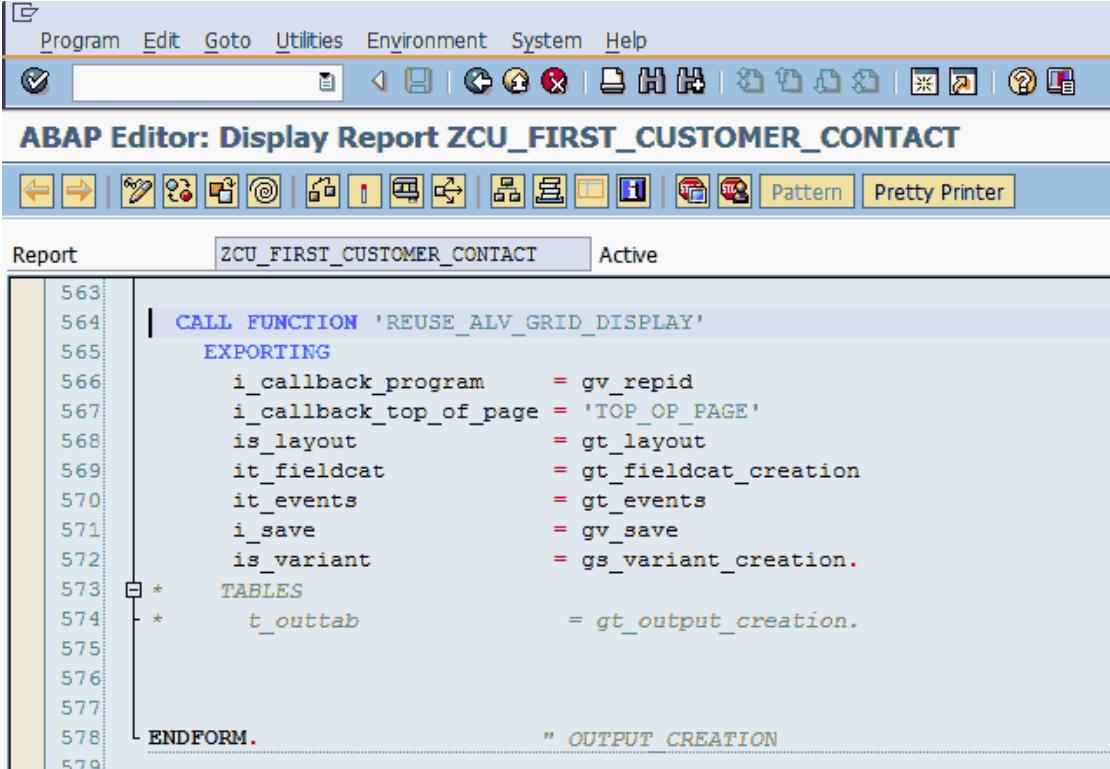
Program: ZCU_FIRST_CUSTOMER_CONTACT Row: 564 [Prio 2]
The "TABLES" parameter "T_OUTTAB" from the function module "REUSE_ALV_GRID_DISPLAY" is mandatory.
Declare an appropriate "TABLES" actual parameter.
Internal Message Code: MESSAGE G-B
Can be hidden using pragma ##FM_PAR_MIS (or pseudo comment "#EC FB_PAR_MIS")

```

Figure 4.17: Detailed message information in the extended program check

With another double click on the first line of the error message, the system jumps to the faulty place in the source code.

Figure 4.18 shows the table parameter in the call of the function module “REUSE\_ALV\_GRID\_DISPLAY” is missing.



The screenshot shows the ABAP Editor interface with the title "ABAP Editor: Display Report ZCU\_FIRST\_CUSTOMER\_CONTACT". The menu bar includes Program, Edit, Goto, Utilities, Environment, System, and Help. The toolbar has various icons for file operations like Open, Save, Print, and Copy. The main area displays ABAP code:

```
Report ZCU_FIRST_CUSTOMER_CONTACT Active
563 | CALL FUNCTION 'REUSE_ALV_GRID_DISPLAY'
564 |   EXPORTING
565 |     i_callback_program      = gv_repid
566 |     i_callback_top_of_page  = 'TOP_OF_PAGE'
567 |     is_layout                = gt_layout
568 |     it_fieldcat              = gt_fieldcat_creation
569 |     it_events                = gt_events
570 |     i_save                   = gv_save
571 |     is_variant               = gs_variant_creation.
572 |
573 |   * TABLES
574 |   *   t_outtab                = gt_output_creation.
575 |
576 |
577 |
578 | ENDFORM.                      " OUTPUT CREATION
579 |
```

Figure 4.18: Direct jump out of the extended program check to the source code

Now you can use the  icon to switch to change mode. Remove the comment sign (\*) in front of the line TABLES and the parameter T\_OUTTAB (lines 573 and 574). Activate the program by clicking on the  icon. An extended program check shows no more errors.

An error message results for the SY-SUBRC of the function module BCONTACT\_CREATE when no error handling exists (see Figure 4.19). This means that an error that occurs in this function module will not be handled since the program doesn't react to it.

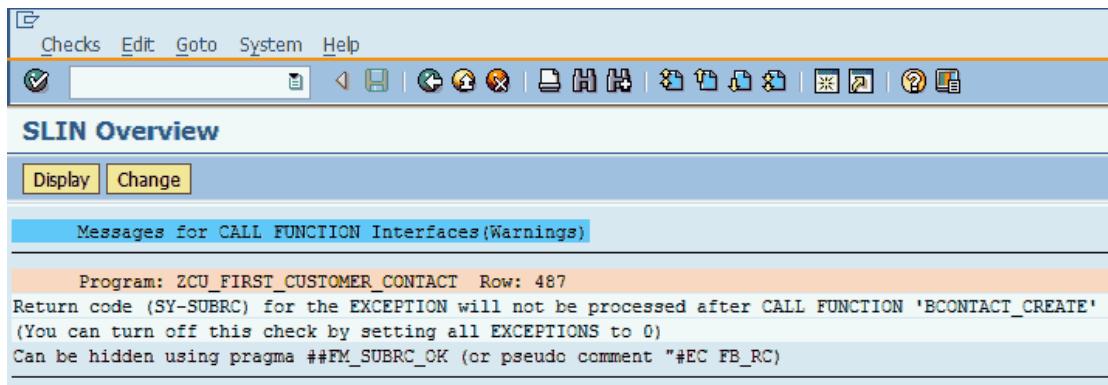


Figure 4.19: No error handling for SY-SUBRC of the function module

Even warnings are interesting. Warnings can help you find declared unused variables (see Figure 4.20).

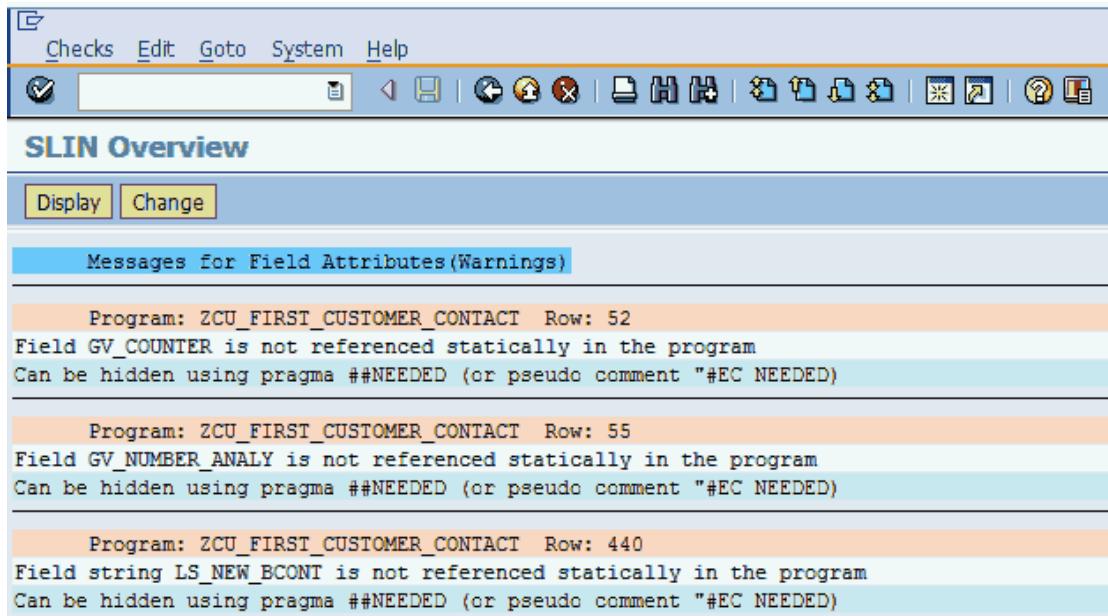


Figure 4.20: Messages for declared but not used variables

With a double click on the first line of the warning, the system jumps to the incorrect source code.

Certain warnings of the syntax check, as shown in Figure 4.21, can be hidden by using an ABAP command.

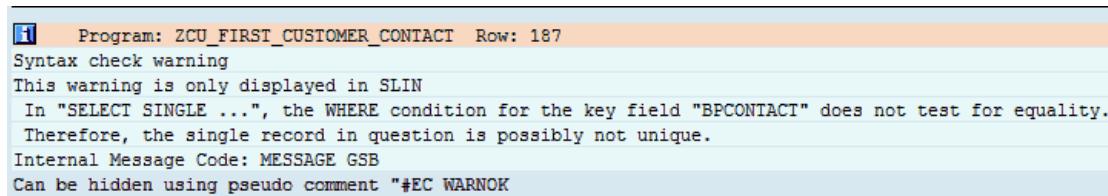


Figure 4.21: Warnings for SELECT SINGLE statements without using the key fields

With a double click on the first line of the message, the system shows the SELECT statement of the program code as shown in Figure 4.22.

```
* Selection of contacts for contact class First Customer
LOOP AT gt_output ASSIGNING <gs_output>.

  SELECT SINGLE *
    FROM bcont
    INTO CORRESPONDING FIELDS OF <gs_output>
    WHERE partner = <gs_output>-partner
    AND cclass = co_cclass "First Customer"
    AND activity = co_activity "activity"
    AND ctype = co_ctype. "Internal Info

    IF sy-subrc <> 0.
      <gs_output>-notes = 'Customer has no contact with the contact class First Customer'.
    ELSE.
      <gs_output>-notes = 'Customer has contact with the contact class First Customer'.
    ENDIF.

  ENDLOOP.
```

"#EC WARNOX

Figure 4.22: Fade out of a warning

Use the help of the SELECT statement to check whether a contact class exists for a selected business partner. You can hide this warning with the sign “#EC WARNOX” after the statement “SELECT SINGLE \*”.

The reduction of messages shows that the developer has recognized and analyzed the warning of the extended program check.

## 4.5 ABAP version management

*ABAP version management* is a useful and often-needed tool. It allows the comparison of different versions of development objects as well as Data Dictionary objects, such as programs, function modules, structures, and tables.

A new version of a development or dictionary object always is created automatically if a transport contains this object. By changing the object again, a new transport is created. This enables you to create two versions of an object.

For example, look at the program ZCU\_FIRST\_CUSTOMER\_CONTACT from [Chapter 1](#). In **TRANSACTION SE80**, you can find the version management screen by using the pull-down menu **UTILITIES • VERSIONS • VERSION MANAGEMENT**.

The version management screen shows every transport that includes the development object (see Figure 4.23).

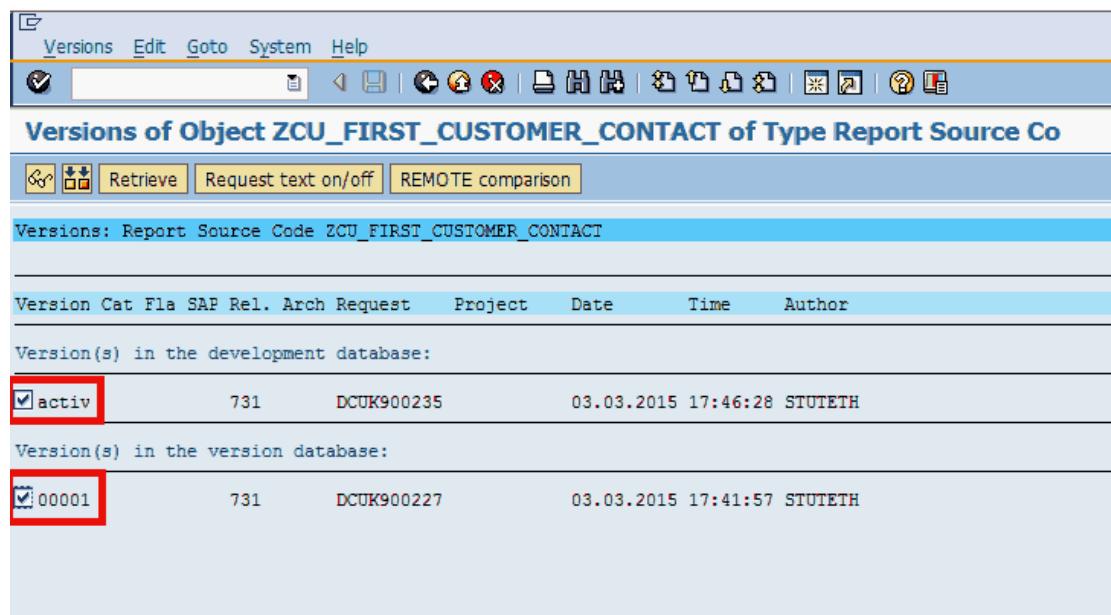


Figure 4.23: Display versions and their transports

There are two comparable versions of this example program. If you mark the check boxes in front of both transports and click the icon, you can compare the versions. All changes between the versions of the program are marked as shown in Figure 4.24. First, the display shows the comparison in one column. For a clearer overview, click on the button (see Figure 4.24).

The screenshot shows a 'Compare Programs: All' dialog. It displays two columns of ABAP code. The left column is labeled 'Version 1' and the right column is labeled 'Version 2'. The code compares declarations, type pools, and data sections between the two versions. Red highlights and crossed-out lines indicate differences between the two versions.

```
42 FIELD-SYMBOLS <gs_output> TYPE zcu_firstcustomer_contact.
43
44
45
46 DATA: gt_output          TYPE TABLE OF zcu_firstcustomer_contact,
47         gs_output          TYPE zcu_firstcustomer_contact,
48         gt_output_creation TYPE TABLE OF zcu_firstcustomer_contact,
49         gs_bcontc          TYPE bcontc,
50         gs_bconta          TYPE bconta,
51         gs_bcont            TYPE bcont,
52         gv_number_analyse TYPE i,
53         gv_number_creation TYPE i.
54
55
56 * Declarations for ALV-Grid-Controls
57 TYPE-POOLS: slis.
58
59 ... 12 unchanged lines omitted
60 DATA: BEGIN OF gs_variant_creation.
61 INCLUDE STRUCTURE disvariant.
62 DATA: END OF gs_variant_creation.
63
64 FIELD-SYMBOLS <gs_output> TYPE zcu_firstcustomer_contact.
65
66
67 DATA: gt_output          TYPE TABLE OF zcu_firstcustomer_contact,
68         gs_output          TYPE zcu_firstcustomer_contact,
69         gt_output_creation TYPE TABLE OF zcu_firstcustomer_contact,
70         gs_bcontc          TYPE bcontc,
71         gs_bconta          TYPE bconta,
72         gs_bcont            TYPE bcont,
73         gv_number_analyse TYPE i,
74         gv_number_creation TYPE i.
75
76 *-----*
77 * SELECTION-SCREEN
78 ... 89 unchanged lines omitted
79 FORM selection_of_contacts .
80
81
82 * Selection of Business Partner
83 SELECT *
```

Figure 4.24: Parallel comparison of two versions

Alternatively, you can click the **Settings** button to start the *split-screen editor* (see Figure 4.25). The red crosses in front of the source code show which lines are different between the versions.

The screenshot shows the SAP ABAP Split Screen Editor in Version Management Comparison Mode. It displays two columns of ABAP code. Red crosses are placed in front of lines that differ between the two versions. The left column is labeled 'Version 1' and the right column is labeled 'Version 2'.

```
28
29
30
31
32 TABLES: but000.
33
34 TYPE-POOLS: bpc01.
35
36 CONSTANTS: co_ciclass      TYPE ct_ciclass VALUE '0020', "First Customer
37             co_activity       TYPE ct_activit VALUE '0001', "First Customer
38             co_ctype          TYPE ct_ctype   VALUE '008',  "Internal Info
39             co_f_coming       TYPE ct_coming  VALUE '3',    "Internal
40             co_origin          TYPE ct_origin   VALUE '2'.  "Background processing
41
42
43 * Declarations for ALV-Grid-Controls
44 TYPE-POOLS: slis.
45
46 DATA: gt_fieldcat          TYPE alis_t_fieldcat_slv,
47         gt_fieldcat_creation TYPE alis_t_fieldcat_slv,
48         gt_events           TYPE alis_t_event,
49         gt_layout            TYPE alis_layout_slv,
```

```
28
29
30 * Declarations
31
32 TABLES: but000.
33
34 TYPE-POOLS: bpc01.
35
36 CONSTANTS: co_ciclass      TYPE ct_ciclass VALUE '0020', "First Customer
37             co_activity       TYPE ct_activit VALUE '0001', "First Customer
38             co_ctype          TYPE ct_ctype   VALUE '008',  "Internal Info
39             co_f_coming       TYPE ct_coming  VALUE '3',    "Internal
40             co_origin          TYPE ct_origin   VALUE '2'.  "Background processing
41
42
43 * Declarations for ALV-Grid-Controls
44 TYPE-POOLS: slis.
45
46 DATA: gt_fieldcat          TYPE alis_t_fieldcat_slv,
47         gt_fieldcat_creation TYPE alis_t_fieldcat_slv,
48         gt_events           TYPE alis_t_event,
49         gt_layout            TYPE alis_layout_slv,
```

Figure 4.25: Split-screen editor for the comparison of two versions

In the first version management screen (see Figure 4.23), you can click the **Retrieve** button to overwrite the current version number with an older version number. Mark the check field in the first column of the older version and then click the **Retrieve** button. The version management tool shows the message shown in Figure 4.26.

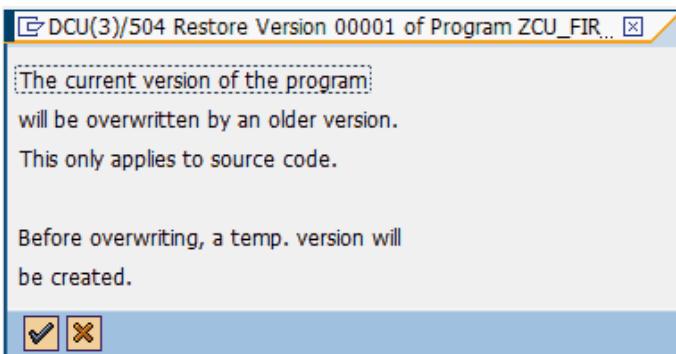


Figure 4.26: Info message by overwriting an active version with an older version

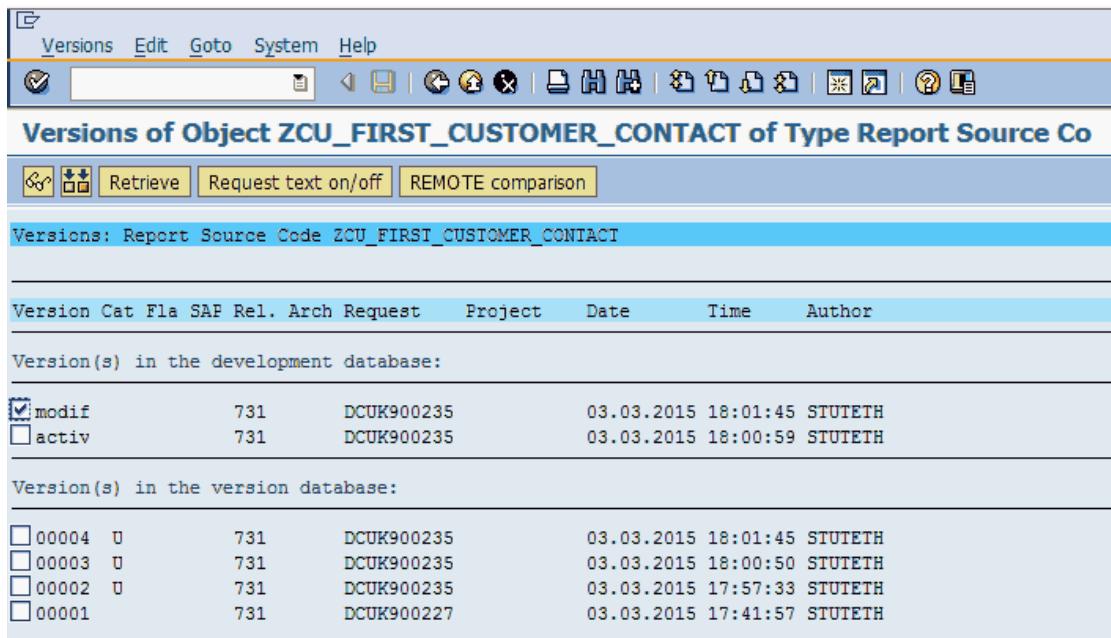
By clicking the **✓** icon, the source code of the active version is overwritten by the selected older version. The result is shown in Figure 4.27.

Versions of Object ZCU_FIRST_CUSTOMER_CONTACT of Type Report Source Co									
	Version	Cat	Fla	SAP Rel.	Arch Request	Project	Date	Time	Author
<b>Version(s) in the development database:</b>									
<hr/>									
<input type="checkbox"/>	modif		731		DCUK900235		03.03.2015	17:57:33	STUTETH
<input type="checkbox"/>	activ		731		DCUK900235		03.03.2015	17:46:28	STUTETH
<b>Version(s) in the version database:</b>									
<hr/>									
<input type="checkbox"/>	00002	U	731		DCUK900235		03.03.2015	17:57:33	STUTETH
<input checked="" type="checkbox"/>	00001		731		DCUK900227		03.03.2015	17:41:57	STUTETH

Figure 4.27: Display of a reactivated older program version

The newest version has the sign “modif” (for modification). At this time, the source version is always active. Now you have two possibilities:

1. The active version should be kept in the source version. In this case, you have to mark the active version in the first column. Click the  icon to display the source version. When you activate this version, the temporary modified version disappears.
2. The older version should be reactivated. In this case, you have to mark the modified version in the first column (see Figure 4.27). Display the reactivated older version by clicking on the  icon. If you activate this version by clicking the  icon, the source version is overwritten. Initially, an inactive program version is activated in the version management tool. However, as shown in Figure 4.28, the version management tool creates a new version with the number “0004” for the overwritten version. You can return to the overwritten version (0004) at any time.



The screenshot shows the SAP Version Management tool interface. The title bar reads "Versions of Object ZCU\_FIRST\_CUSTOMER\_CONTACT of Type Report Source Co". Below the title bar is a toolbar with various icons. The main area displays a table of program versions. The table has columns: Version, Cat, Fla, SAP Rel., Arch, Request, Project, Date, Time, and Author. There are two sections: "Version(s) in the development database:" and "Version(s) in the version database:". The "development database" section shows two rows: one for a modified version (checked) and one for an active version. The "version database" section shows four rows, each with a different version number (00001 to 00004) and their corresponding details.

Version	Cat	Fla	SAP Rel.	Arch	Request	Project	Date	Time	Author
modif		731				DCUK900235	03.03.2015	18:01:45	STUTETH
activ		731				DCUK900235	03.03.2015	18:00:59	STUTETH
00004	U	731				DCUK900235	03.03.2015	18:01:45	STUTETH
00003	U	731				DCUK900235	03.03.2015	18:00:50	STUTETH
00002	U	731				DCUK900235	03.03.2015	17:57:33	STUTETH
00001		731				DCUK900227	03.03.2015	17:41:57	STUTETH

Figure 4.28: Program versions in version management tool

The *remote comparison* is another important version management function. With this function, you can compare versions from the development system with current active versions on a test or production system.

You can start the remote comparison by clicking the **REMOTE comparison** button in the version management tool. In the popup screen, input the system ID or the **RFC-DESTINATION** of the **TARGET SYSTEM** (see Figure 4.29). Confirm your selection by clicking the  icon.

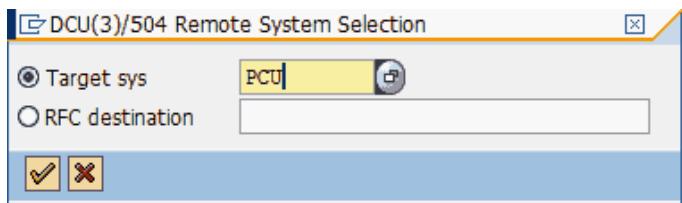


Figure 4.29: Input the system ID of the target system for a remote comparison

Finally, you have to click the **REMOTE comparison** button in the following screen. The selected version of the source system will be compared to the current version on the target system.

With this method, you can find out which version exists on the target system (e.g., on the production system). Moreover, version management also shows the transport number and the date and time at which the version has been activated on the target system.

## 4.6 Compare source codes of different programs

It's a different process to compare codes of different programs. To explain this function I've copied the program ZCU\_FIRST\_CUSTOMER\_CONTACT to the program ZCU\_FIRST\_CUSTOMER\_CONTACT\_NEW. Then I've modified the copied new program in a few places. For the program comparison, use the ABAP editor of **TRANSACTION SE80**. You can start the program comparison with the pull-down menu **UTILITIES • MORE UTILITIES • SPLIT-SCREEN EDITOR**. Figure 4.30 shows the first screen of the program comparison.

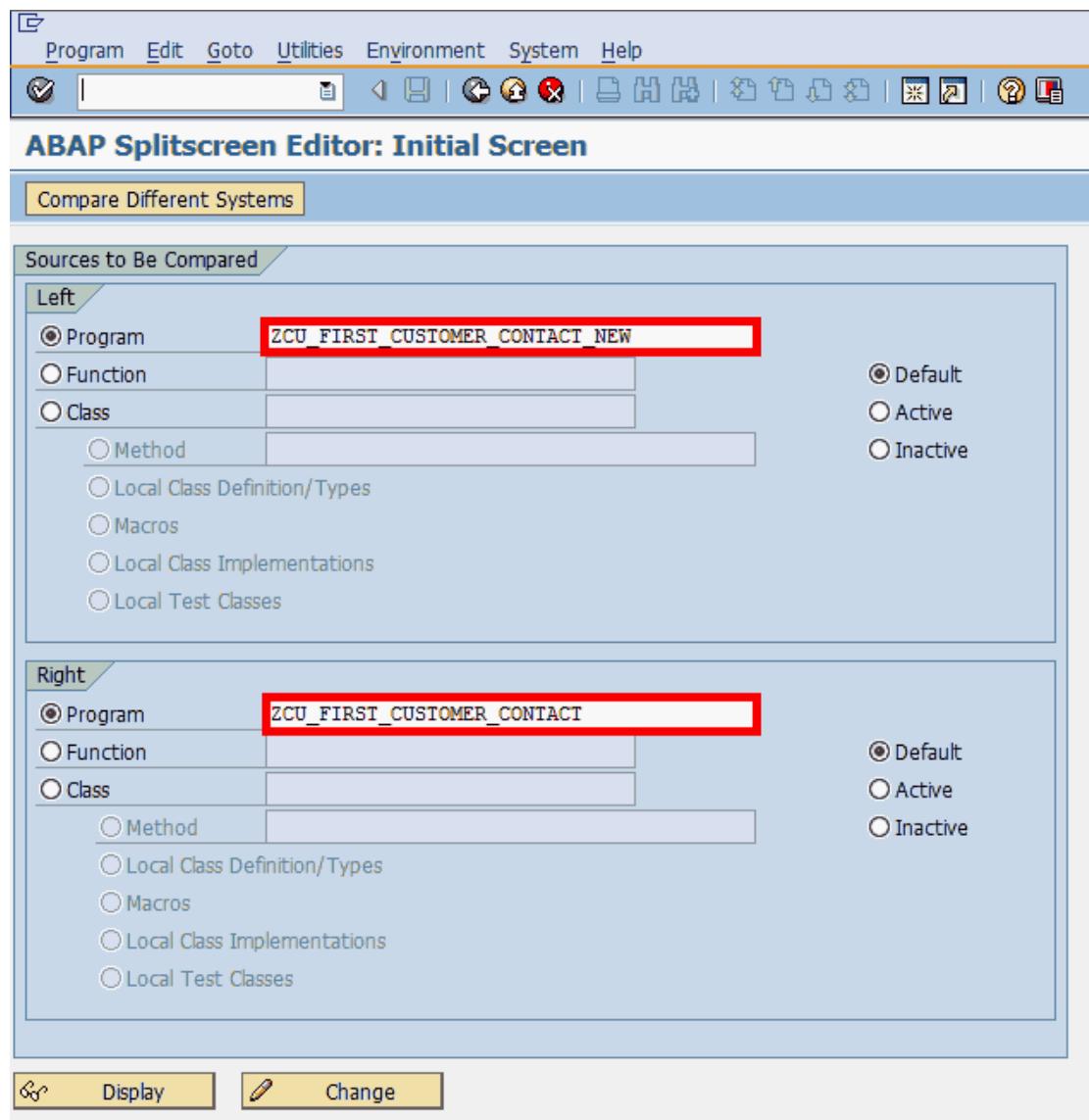


Figure 4.30: Input of different programs for the source code comparison

If you start with the source program in the ABAP editor, the system automatically prints the name of this program in the lower part of the screen.

The screenshot shows the ABAP Split-screen Editor interface. At the top, there's a menu bar with 'Program', 'Edit', 'Goto', 'Utilities', 'Environment', 'System', and 'Help'. Below the menu is a toolbar with various icons. The main area is divided into two panes. Both panes show the same ABAP code for 'ZCU\_FIRST\_CUSTOMER\_CONTACT\_NEW' and 'ZCU\_FIRST\_CUSTOMER\_CONTACT'. The code includes declarations, table definitions, type pools, and constants. The right pane also shows the 'CONSTANTS' section with the value '0020' assigned to 'co\_cclass'. The status bar at the bottom indicates the file is 'Active'.

```

Report  ZCU_FIRST_CUSTOMER_CONTACT_NEW Active          Report  ZCU_FIRST_CUSTOMER_CONTACT Active
1  * "*
2  *# Report ZCU_FIRST_CUSTOMER_KONTAKT
3  *
4  *
5  *# Author: Thomas Stüttenbäumer, ConUti GmbH
6  *# Datum: 03.03.2015
7  *# Programtask:
8  *# Creation of contacts to the Contact class "First Customer"
9  *# for selected business partner
10 *
11 * Following fields are used for contact creation
12 *MANDAT      sy-mandt
13 *PARTNER     BP-Number
14 *PROFACT    internal number
15 *CLASS       contact class BCNTC Fest = 0020 = First Customer
16 *ACTIVITY   contact activity BCNTA Fest = 0001 = First Customer
17 *CTYPE      contacttype BCNTT Fest = 008 = Interne Info
18 *F_COMING   input/output fixed value = 3 = internal
19 *ORIGIN     Herkunft fixed value = 2 = background processing
20 *CTDATE     sy-datum
21 *CTTIME     sy-useit
22 *ERDAT      sy-datum
23 *ERNAME    sy-uname
24 *ANMERKUNG
25 *
26 REPORT zcu_first_customer_contact_new.
27
28
29  * Deklarationen
30  *
31  *
32 TABLES: b0t000.
33
34 TYPE-POOLS: kpc01.
35
36 CONSTANTS: co_cclass      TYPE ct_cclass VALUE '0020', "First Customer

```

Figure 4.31: Source code comparison of different programs with the split-screen editor

In the split-screen editor, you can switch between display and change modes for both programs. In change mode, you can change both programs at the same time.

The split-screen editor also allows the comparison of programs over different connected SAP systems as well as comparing functions modules and methods of classes.

## 4.7 Log application usage

After a while, thousands of customer-developed programs can be found in SAP systems. Many of these programs are used only once and not ever required again. However, more often than not, the owner of the SAP system can't say which programs are no longer used.

It can be helpful to know exactly which program has been run in a defined period, especially to find errors stored in the database.

This and a host of other reasons support the need for documenting customer development. You can use a developed include program which logs each use of a program by writing a dataset to a customer-specific database table. "Program:Recording Using"

The include has to be called in every customer-specific program. (Note that the additional installation of such an inclusion in customer-developed programs generates a certain expenditure, but the expenditure pays off quickly for the IT provider.)

A protocol table like that explained above can contain the fields shown in Figure 4.32.

The screenshot shows the SAP Dictionary: Display Table interface. The title bar reads "Dictionary: Display Table". The toolbar includes icons for Table, Edit, Goto, Utilities, Extras, Environment, System, Help, and various navigation and search functions. The menu bar has "Table" selected. The main area displays a table titled "ZCU\_PROGRAM" with the status "Active". The table has 11 columns: Field, Key, Initia..., Data element, Data Type, Length, Decim..., and Short Description. The table contains 11 rows of data:

Field	Key	Initia...	Data element	Data Type	Length	Decim...	Short Description
MANDT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	MANDT	CLNT	3	0	Client
PGMID	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	PGMID	CHAR	4	0	Program ID in Requests and Tasks
OBJECT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	TROBJTYPE	CHAR	4	0	Object Type
OBJ_NAME	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	SOBJ_NAME	CHAR	40	0	Object Name in Object Directory
TCODE	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	TCODE	CHAR	20	0	Transaction Code
SYBATCH	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	SYBATCH	CHAR	1	0	Program is running in the background
TIME_STAMP	<input type="checkbox"/>	<input checked="" type="checkbox"/>	ZCU_LAST_TIME_S...	DEC	21	7	Timestamp for last use of a program
ADAT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	ZCU_LAST_DATE	DATS	8	0	Date of last program use
UZEIT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	ZCU_LAST_TIME	TIMS	6	0	Time of last program use
USERNAME	<input type="checkbox"/>	<input checked="" type="checkbox"/>	ZCU_LAST_USER	CHAR	12	0	Name of last user
NUMBER_USE	<input type="checkbox"/>	<input checked="" type="checkbox"/>	ZCU_NUMBER_CALLS	INT4	10	0	Number of Calls

Figure 4.32: Fields of a protocol table to record program use

This table has the following attributes and technical settings:

- ▶ Delivery class “A” for application table
- ▶ “Display/Maintenance Allowed with Restrictions,” meaning only data sets can be displayed in **TRANSACTION** SE16N or SE11
- ▶ In the technical settings, this table should have the data class “A” (for transaction data / transparent table), the size category “1” (depending on the amount of customer developments, such as size categories) and the buffering of the table isn’t allowed.
- ▶ The enhancement table setting category is **CANNOT BE ENHANCED**.

The necessary include is finished quickly (see Listing 4.3). Once a program has been used, a dataset for the program exists in the table ZCU\_PROGRAM. In this case, the username, the date, the time, the timestamp, and the number of uses have to be updated. If a program is called for the first time, a new dataset has to be inserted.

```
*&-----*
*& Include      ZCU_APPLICATION_USAGE
*&-----*
* Tasks of this include:
* This include writes information about application
* usage to table ZCU_PROGRAM
* This include has to implemented in every customer
* program, therefore use command
* INCLUDE ZCU_APPLICATION_USAGE
* The include may only be used in programs
* Using it in modules can lead to dumps due to the
* command Commit Work
*-----*
* Author: Thomas Stutenbäumer, Conuti GmbH
* Date: 03.03.2015
*-----*
* Log table
```

TABLES: zcu\_program.

DATA: lv\_time\_stamp TYPE timestamppl.

GET TIME STAMP FIELD lv\_time\_stamp.

```
zcu_program-mandt      = sy-mandt.  
zcu_program-pgmid      = 'R3TR'.  
zcu_program-object      = 'PROG'.  
zcu_program-obj_name    = sy-repid.  
zcu_program-tcode       = sy-tcode.  
zcu_program-sybatch     = sy-batch.  
zcu_program-time_stamp  = lv_time_stamp.  
zcu_program-adat       = sy-datum.  
zcu_program-uzeit       = sy-uzeit.  
zcu_program-username    = sy-uname.  
zcu_program-number_use   = 1.
```

\* Read log table

```
SELECT SINGLE *  
FROM zcu_program  
WHERE pgmid  = zcu_program-pgmid  " R3TR  
AND object  = zcu_program-object  " PROG  
AND obj_name = zcu_program-obj_name " Programname  
AND tcode   = zcu_program-tcode   " Transaction  
AND sybatch = zcu_program-sybatch. " Dia. or Batch (=X)
```

\* The program has been used before

IF sy-subrc = 0.

\* Number\_use increments with one and  
\* monitor the final use

```
IF zcu_program-number_use > 999999998.  
  zcu_program-number_use = 1.  
ELSE.  
  zcu_program-number_use = zcu_program-number_use + 1.  
ENDIF.
```

zcu\_program-adat = sy-datum. “ Date

```
zcu_program-uzeit = sy-uzeit. " Time  
zcu_program-username = sy-uname. " User  
zcu_program-sybatch = sy-batch. " Dia or Btch (=X)  
zcu_program-time_stamp = lv_time_stamp. "timestamp
```

UPDATE zcu\_program.

- \* Commit work in order to avoid dead lock situations
  - \* by using it from different programs at the same time
- COMMIT WORK.

ELSE.

- \* This program is used at the first time

INSERT zcu\_programm FROM zcu\_program.

- \* Commit work in order to avoid dead lock situations
  - \* by using it from different programs at the same time
- COMMIT WORK.

ENDIF.

\*-----\*

*Listing 4.3: Include for the protocol of program use*

The use of this include is obligatory for all new customer developments and needs to be defined.

## 4.8 Expensive work – Treatment of time slices

In practice, the developer often has to check and change validity periods of data sets (time slices). The treatment of such time slices often results in misunderstandings between the user and developer. (Note that even small changes in the request can affect considerable additional expenditure for the treatment of time slices.)

### Change of time slices



In [Chapter 1](#), the request was made to change the meter reading unit of an installation. The meter reading units are stored in time slices of the installation in the database table EANLH.

For the change of installation time slices with a free selectable from-date and a to-date equal to 12-31-9999 (sign for an infinite validity period), you have a limited amount of cases to take into account (see Figure 4.33).

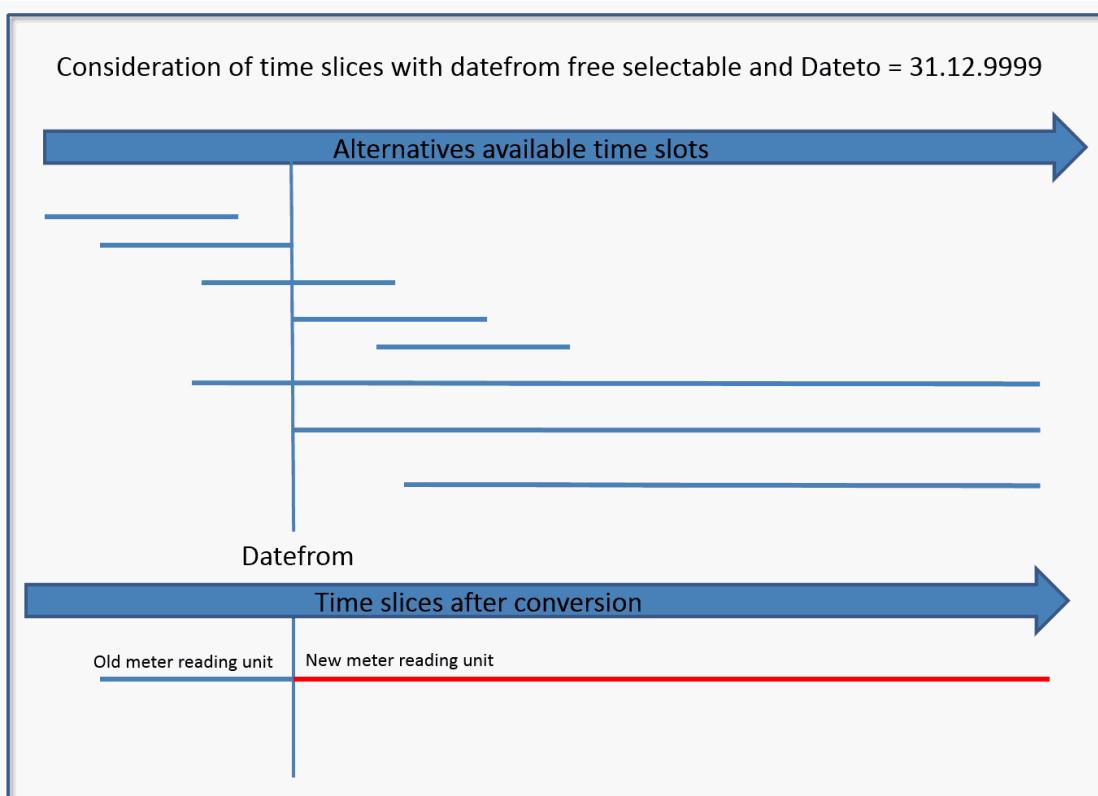


Figure 4.33: Time slices with a free selectable from-date

As shown in Figure 4.34 the amount of case differentiations increase exponentially if the to-date is also free selectable.

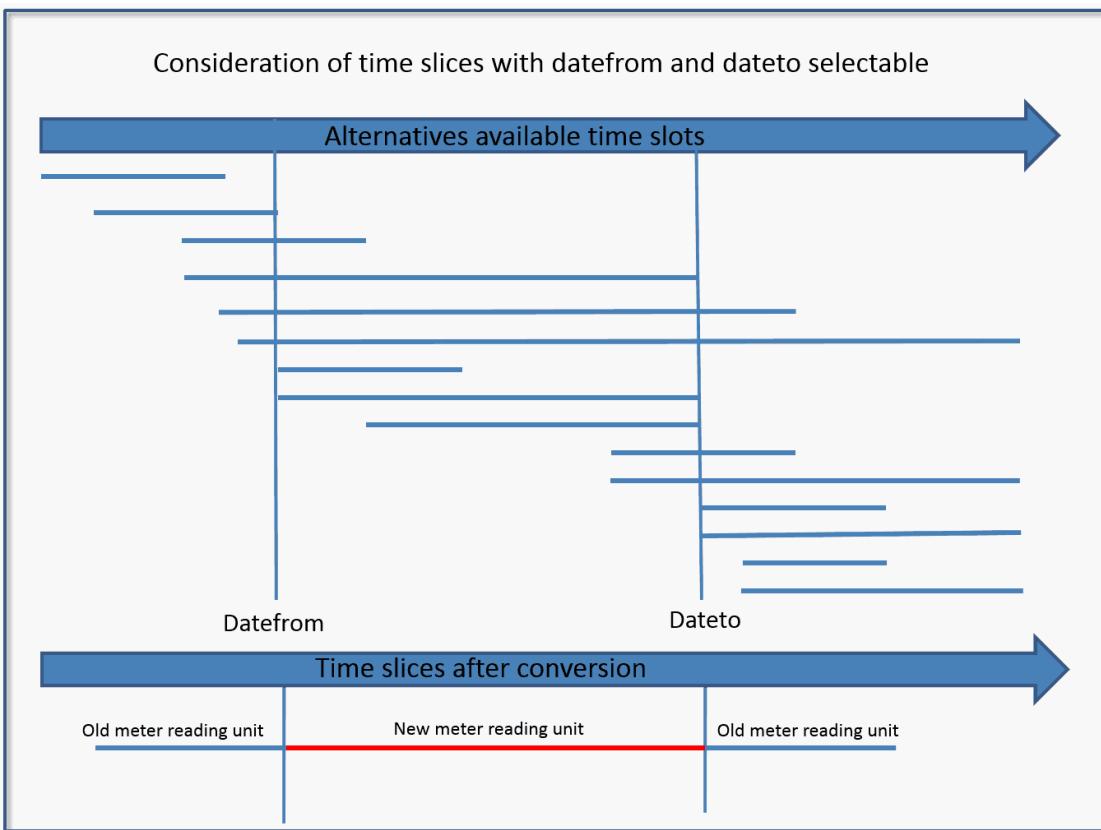


Figure 4.34: Time slices with a free selectable from- and to-date

The number of case differentiations in the program code increases from 8 cases to 15 cases if the from-date and to-date are free selectable. (This means the expenditure doubles for the program development. So it is important to grasp the demands exactly with regard to the time slices before you make an expenditure estimate.)

The example program ZCU\_FIRST\_CUSTOMER\_MR\_UNIT in [Section 1.4.2](#) contains the coding to change time slices with a free selectable from-date and a fixed to-date, which is equal to 12-31-9999. (You can download the source code from <http://abap.espresso-tutorials.com> (see [Appendix B](#))).

## 4.9 Data types **TIMESTAMP** and **TIMESTAMPL**

The ABAP data types **TIMESTAMP** and **TIMESTAMPL** seldom are documented. In standard SAP coding, these data types are often used if sensitive data has to be stored (e.g., for the validity period of banking details) or to record data set changes (e.g., with the help of **TRANSACTION** SE16N and the protocol in the database table SE16N\_CD\_DATA).

The data type **TIMESTAMP** or **TIMESTAMPL** defines an alphanumeric field where time stamps are stored. The time stamp is a special character string consisting of a date and time value (up to microseconds). The *data type* **TIMESTAMP** describes the short form of the time stamp. **TIMESTAMP** owns the domain TZNTSTMP and is a decimal field of the length 15 characters with spaces. The data type **TIMESTAMPL** describes the long form of the time stamp. It owns the domain TZNTSTMPL and has a length of 21 characters with spaces plus 7 post-comma places.

The statement “GET TIME STAMP FIELD gv\_time\_stamp” builds a UTC timestamp out of the system date and time and assigns the value to the variable gv\_time\_stamp.

The variable gv\_time\_stamp can have the data type **TIMESTAMP** or **TIMESTAMPL**. The data type of the time stamp variable is assigned in the short form or the long form.

Assignment of the time stamp to a variable with the data type **TIMESTAMPL**:  
20141105102808.2820000, translates to YYYYMMDDHHMMSS.xyz

Assignment of the time stamp to a variable with the data type **TIMESTAMP**:  
20141105102808, translates to YYYYMMDDHHMMSS

The shorthand symbols are:

- ▶ **YYYY** = Year
- ▶ **MM** = Month
- ▶ **DD** = Day
- ▶ **HH** = Hour
- ▶ **MM** = Minute
- ▶ **SS** = Seconds

- ▶ xyz = Post comma of the seconds

The displays of these data types in **TRANSACTION SE16N** look a little bit different from the date stored in the database.

Figure 4.35 shows an example of the validity period of a bank detail in the database table BUT0BK with the short form of the time stamp.

MANDT	PARTNER	BKVID	BANKS	BANKL	BANKN	BKONT	BKREF	KOINH	BK_VALID_TO	BK_VALID_FROM
504	0010000102	0001	DE	41450075	30746175			Meyer Col. Düsterkötter, Axel	99.991.231.235.959	10.101.000.000
504	0010000106	0001	DE	36060591	443150298			Thüel,Wisia	99.991.231.235.959	10.101.000.000
504	0010000107	0001	DE	41460116	672717747			Gotte,August	99.991.231.235.959	10.101.000.000
504	0010000109	0001	DE	41450075	177486235			Poos,Zinoba	99.991.231.235.959	10.101.000.000
504	0010000110	0001	DE	41450075	556080887			Tubesing,Sebastian	99.991.231.235.959	10.101.000.000
504	0010000112	0001	DE	41662465	586197487			Glarmann,Undine	99.991.231.235.959	10.101.000.000
504	0010000114	0001	DE	41670029	137780149			Venckhuß,Quax	99.991.231.235.959	10.101.000.000
504	0010000116	0001	DE	41450075	250560886			Wuthrich,Zilla	99.991.231.235.959	10.101.000.000

Figure 4.35: Short form of the time stamp in the database table BUT0BK (bank details)

The fields **BK\_VALID\_FROM** and **BK\_VALID\_TO** have the domain TZNTSTMP, which means the short form of the time stamp. The domain doesn't show any conversion routines.

In a SELECT statement, you can use a variable with the data type TIMESTAMP or TIMESTAMPL. If the field has the short form of the time stamp, the variable in the SELECT statement can have the data type of the long form.

The coding in Listing 4.4 checks whether a business partner has a valid bank detail:

```

DATA: gt_but0bk      TYPE TABLE OF but0bk,
      gv_time_stamp  TYPE timestamp.

SELECTION-SCREEN BEGIN OF BLOCK se1
  WITH FRAME TITLE text-se1.
SELECTION-SCREEN SKIP.
PARAMETERS: pa_gpart TYPE bu_partner.
SELECTION-SCREEN SKIP.
SELECTION-SCREEN END OF BLOCK se1.

```

GET TIME STAMP FIELD gv\_time\_stamp.

```

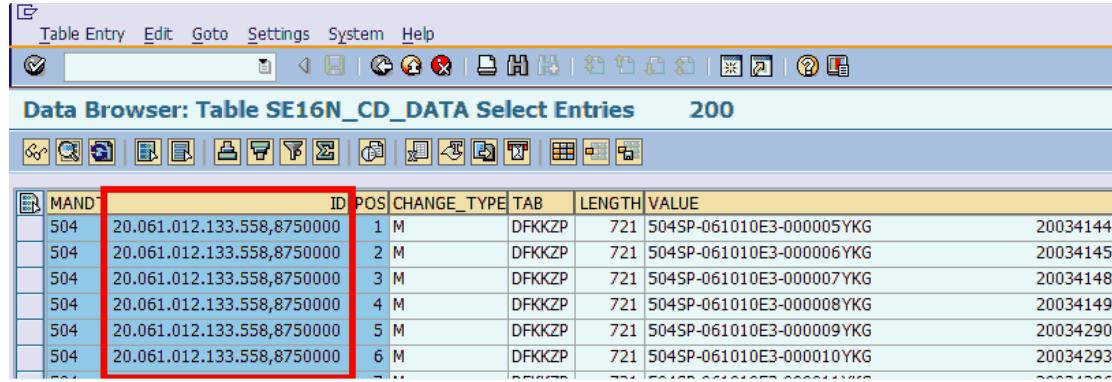
SELECT *
  FROM but0bk
 INTO TABLE gt_but0bk
 WHERE partner    = pa_gpart

```

```
AND bk_valid_from <= gv_time_stamp  
AND bk_valid_to >= gv_time_stamp.
```

*Listing 4.4: Validity check of bank details*

Figure 4.36 shows data sets with a field that uses the long form of the time stamp.



The screenshot shows the SAP Data Browser interface with the title "Data Browser: Table SE16N\_CD\_DATA Select Entries 200". The table has columns: MANDT, ID, POS, CHANGE\_TYPE, TAB, LENGTH, and VALUE. The first six rows of the table are highlighted with a red border. The "ID" column contains timestamps in the format "20.061.012.133.558,8750000". The "TAB" column contains "DFKKZP". The "VALUE" column contains numerical values: 20034144, 20034145, 20034148, 20034149, 20034290, and 20034293.

MANDT	ID	POS	CHANGE_TYPE	TAB	LENGTH	VALUE
504	20.061.012.133.558,8750000	1	M	DFKKZP	721	504SP-061010E3-000005YKG
504	20.061.012.133.558,8750000	2	M	DFKKZP	721	504SP-061010E3-000006YKG
504	20.061.012.133.558,8750000	3	M	DFKKZP	721	504SP-061010E3-000007YKG
504	20.061.012.133.558,8750000	4	M	DFKKZP	721	504SP-061010E3-000008YKG
504	20.061.012.133.558,8750000	5	M	DFKKZP	721	504SP-061010E3-000009YKG
504	20.061.012.133.558,8750000	6	M	DFKKZP	721	504SP-061010E3-000010YKG

*Figure 4.36: Long form of a time stamp in the database table SE16N\_CD\_DATA*

A conversion of the UTC time stamp to other time zones can be made with the help of the following statement.

```
CONVERT TIME STAMP gv_time_stamp
```

```
    TIME ZONE gv_timezone
```

*Listing 4.5: CONVERT TIME STAMP statement*

(For further information, hit the “F1” help key).

## 4.10 Field symbols and ASSIGN command

*Field symbols* are useful data objects. With field symbols, the developer has direct access to the memory content of a variable term. Field symbols have to be declared in angle brackets, differentiating them from other source code variables.

Field symbols should be typed (specified) as much as possible.

### Possible types of field symbols

This list shows possible types for field symbols.

#### Typification

##### Check for the assigned variable

No Type Information

All types of data objects can be assigned.

The field symbols get all attributes from the assigned data object.

TYPE ANY

For example, “No Type Information”

TYPE C, N, P, or X

Only data objects with the type C, N, P, or X can be assigned. The field symbol gets the field length and the decimal places (only for type P) from the assigned data object.

TYPE TABLE

If you assign a variable, the assigned variable must have the type of an internal standard table. This is the short form of TYPE STANDARD TABLE (see below).

TYPE ANY TABLE

With an assign, the system confirms the variable is an internal table. The attributes of the assigned table will be completely assumed to the field symbol (line type, table type, key fields).

TYPE INDEX TABLE

The system confirms the assigned data object is an internal index table (e.g., a standard table or a sorted table, but not hashed table). The attributes of the assigned table will be assumed completely to the field symbol (line type, table type, key fields).

TYPE STANDARD TABLE

The system confirms the assigned data object is an internal standard table.

The attributes of the assigned table will be assumed completely to the field symbol (line type, table type, key fields).

TYPE SORTED TABLE

The system confirms the assigned data object is an internal sorted table.

The attributes of the assigned table will be assumed completely to the field symbol (line type, table type, key fields).

TYPE HASHED TABLE

The system confirms the assigned data object is an internal hashed table.

The attributes of the assigned table will be assumed completely to the field symbol (line type, table type, key fields).

There are two kinds of declarations for the field symbols:

- ▶ FIELD-SYMBOLS: <field\_symbol>
- ▶ FIELD-SYMBOLS: <field\_symbol> TYPE (type)

A type assignment isn't necessary in every case. Field symbols can be used in dynamic programming (see [Section 4.18](#)).

You can use field symbols in all development objects with exception in the declaration part of classes and interfaces.

The ASSIGN command assigns the value of a variable to the field symbol in the memory. In this example, the value of the variable “gv\_anzahl” with type “integer” is assigned to the field symbol:

```
ASSIGN gv_anzahl TO <field_symbol>.
```

You have to check the SY-SUBRC directly after the ASSIGN command. If the assign is not successful and you use the field symbol, the program crashes (dumps).

### Dump by access to a non-assigned field symbol



Access to a non-assigned field symbol can cause a hard program interrupt (dump).

If the field symbol gets another value in the program, the value of the assigned variable also gets the new value. That means, with the command <field\_symbol> = 5, the variable “gv\_anzahl” gets the value 5.

For example, look at the program ZCU\_ABAP\_ASSIGN\_COMMAND. The ASSIGN statement is used in a LOOP command (see Figure 4.37).

```

*&-----*
*&      Form  ANALYSE_DATA
*&-----*
*      Check if business partner has a valid contract
*-----*
*  -->  gt_output  data to be analysed
* <--  gt_output  analysed data.
*-----*
FORM analyse_data .

LOOP AT gt_output ASSIGNING <gt_output>.
  IF <gt_output>-loevm_fkkvkp      = 'X'.
    <gt_output>-ergebnis          = 'contract account has been deleted'.
    CONTINUE.
  ENDIF.
  IF <gt_output>-loevm_ever       = 'X'.
    <gt_output>-vertrag_ungueltig = 'X'.
    <gt_output>-ergebnis          = 'contract has been deleted'.
    CONTINUE.
  ENDIF.
  IF <gt_output>-billfinit        = 'X'.
    <gt_output>-vertrag_ungueltig = 'X'.
    <gt_output>-ergebnis          = 'contract has been settled'.
    CONTINUE.
  ENDIF.
  IF <gt_output>-auszdat         < sy-datum.
    <gt_output>-vertrag_ungueltig = 'X'.
    <gt_output>-ergebnis          = 'moveout date before current date'.
    CONTINUE.
  ENDIF.
ENDLOOP.

ENDFORM.          " ANALYSE DATA

```

Figure 4.37: LOOP AT gt\_output ASSIGNING <gt\_output>

By using ASSIGNING <gt\_output> in the LOOP command, the MODIFY command for updating the table content in front of the ENDLOOP command is no longer necessary.

```

MODIFY gt_output FROM gs_output
  INDEX sy-tabix.

```

With ASSIGNING, the updated values of the current table line are stored automatically in the relevant table line of GT\_OUTPUT. Therefore, an update of the value in the field symbols also directly updates the value of the relevant line in the table.

Such an ASSIGN command only runs if the field symbol has a type in its declaration. Use a field symbol without a type assignment causes an error by the syntax check.

Field symbols often can be used in dynamic programming as structures in LOOP statements and – as shown in [Section 4.15](#) – by using the dirty assign method.

## 4.11 Perfect output – ALV grid control

SAP has efficient programming tools available for the developer through the *SAP control framework*. The control framework outputs the program features heading detail (name of program, user, etc.) with the help of ABAP commands (*custom controls*). The program logic of these controls runs on the application server and the frontend system is used as a container. According to the SAP GUI version, the custom controls are Active-X controls or JavaBeans.

The controls are black boxes for the developer. The developer can use these boxes to build the finest output shapes to the screen.

The control framework contains following functions:

- ▶ TREE control – build menus in the shape of tree structures
- ▶ EDIT control – make a text editor available in the application
- ▶ PICTURE control – show pictures on the screen
- ▶ CALENDAR control – input help for dates or date intervals
- ▶ TOOLBAR control – enhance the button line in an input screen
- ▶ DRAG&DROP control – shift objects on the screen
- ▶ ALV GRID control – build lists in a form similar to Microsoft Excel with a row of useful functions for list processing (*List-Viewer*)

In this section, I'll explain *ALV grid controls*. These controls are used to create the output for often-requested reports or screens.

In the development package “SLIS,” SAP makes many example programs available for ALV grid controls. These controls enable an extensive list of tools. However, in practice, only a few functions are needed to create ALV grid lists.

### 4.11.1 ALV grid lists in report programming

For developing lists, you can build helpful list tools for the user with a few standard function modules.

Using ALV grid controls



To explain the ALV grid controls, I use the program to convert the account class as described in [Chapter 1](#). I've developed the example program ZCU\_FIRST\_CUSTOMER\_ACCOUNT\_CL for this purpose. You can use this program in two ways: To analyze the contract account or to change the contract class of contract accounts. To explain how to build an ALV grid list, it is enough to look at the analyze part of the program.

The program has the following structures in the analyze section:

- ▶ Declare needed variables
- ▶ Build the selection screen
- ▶ Check input data
- ▶ Analyze selected data sets
- ▶ Output the analyze results

To display the analyze results in an ALV grid list, I always use a template (see example program ZCU\_ABAP\_ALV\_GRID\_TEMPLATE). This template can be used in any program containing an output list. You can download this template from <http://abap.espresso-tutorials.com> (see [Appendix B](#)).

You have to declare different variables for the ALV grid list in the declaration section (see Figure 4.38).

```
* Declarations for ALV-Grid-Controls
TYPE-POOLS: slis.

DATA: gt_fieldcat          TYPE slis_t_fieldcat_alv,
      gt_events            TYPE slis_t_event,
      gt_layout             TYPE slis_layout_alv,
      gt_list_top_of_page   TYPE slis_t_listheader,
      gv_repid              LIKE sy-repid,
      gv_save               TYPE c.

□ DATA: BEGIN OF gs_variant.
| INCLUDE STRUCTURE disvariant.
DATA: END OF gs_variant.
```

Figure 4.38: Declarations for list processing with ALV grid controls

The Type-Pool “SLIS” contains data types for the type assignment of variables for the list viewer. This declaration is necessary if you want to expand the functions of the list output. In this simple example, the declaration of “SLIS” isn’t necessary because the variables of the following fields are declared:

- ▶ GT\_FIELDCAT – internal table, which contains information of the fields of the output list
- ▶ GT\_EVENTS – internal table, which contains information for events on which the ALV grid control reacts
- ▶ GT\_LAYOUT – internal table with information to describe the design of the output list
- ▶ GT\_LIST\_TOP\_OF\_PAGE – internal table containing information about the output list header
- ▶ GV\_REPID – variable containing the current program name
- ▶ GV\_SAVE – variable defining in which way the user can change the layout of the output list

Results are shown for the data selections, the checks of the selected data sets, and the analysis of these data sets. The form for the output of the ALV grid list is a template, which has to be adjusted in a few places (see Figure 4.39).

```

*-----*
* Using ALV-Grid-Controls with modulus
*-----*
* Numbers of output dataset
DESCRIBE TABLE gt_output LINES gv_anzahl.

* fill fieldcatalog
PERFORM alv_fieldcat USING gt_fieldcat.

* control output layout
PERFORM alv_layout_build USING gt_layout.

* control event
PERFORM eventtab_build USING gt_events[].

gv_repid           = sy-repid.
gs_variant-report  = sy-repid.
gs_variant-username = sy-uname.

* define storage options for layout
gv_save = 'A'.

* Assign variants for online and background processing
gs_variant-report  = sy-repid.
gs_variant-handle   = '0001'.
gs_variant-variant  = '/STANDARD'.

CALL FUNCTION 'REUSE_ALV_GRID_DISPLAY'
  EXPORTING
    i_callback_program      = gv_repid
    i_callback_top_of_page  = 'TOP_OF_PAGE'
    is_layout                = gt_layout
    it_fieldcat              = gt_fieldcat
    it_events                = gt_events
    i_save                   = gv_save
    is_variant               = gs_variant
  TABLES
    t_outtab                = gt_output.

```

Figure 4.39: Source code for the output of an ALV grid list

To create the ALV grid list, follow these steps:

- ▶ Build the field catalog: define which fields (or columns) are displayed on the ALV grid, which data types have this field, and so on
- ▶ Build the layout of the list to be used
- ▶ Build the event control, such as for the output of the list header
- ▶ Define whether the user can change the layout of the list

- ▶ Build the ALV grid list by using the standard function module REUSE\_ALV\_GRID\_DISPLAY

To build the field catalog, use the source code shown in Figure 4.40.

```
*&-----*
*&      Form  alv_fieldcat
*&-----*
*      Fieldcatalog for ALV-Grid-Control
*-----*
*      -->GT_FIELDCAT  fieldcatalog
*-----*
FORM alv_fieldcat USING      gt_fieldcat TYPE slis_t_fieldcat_alv.

* fill fieldcatalog for ALV-List
CALL FUNCTION 'REUSE_ALV_FIELDCATALOG_MERGE'
  EXPORTING
    i_program_name      = 'ZCU_ABAP_ASSIGN_COMMAND'
    i_structure_name    = 'ZCU_ABAP_ASSIGN'
    i_client_never_display = 'X'
  CHANGING
    ct_fieldcat        = gt_fieldcat
  EXCEPTIONS
    inconsistent_interface = 1
    program_error       = 2
    OTHERS              = 3.

  IF sy-subrc <> 0.
    sy-msgid = 'ZCU_ISU'.
    sy-msgty = 'E'.
    sy-msgno = '308'.
    sy-msgv1 = 'Error in creation of'.
    sy-msgv2 = 'fieldcatalog for output'.
    MESSAGE ID sy-msgid TYPE sy-msgty NUMBER sy-msgno
          WITH sy-msgv1 sy-msgv2 sy-msgv3 sy-msgv4.
  ENDIF.

ENDFORM.                               " alv_fieldcat
```

Figure 4.40: Source code to build the field catalog

The standard function module REUSE\_ALV\_FIELDCATALOG\_MERGE along with a structure defines elements in the Data Dictionary and the structure of the table column. It is also possible to use internal program structures to define the column of the output list. Nevertheless, for performance reasons, SAP advises against it. So, construct the output structure by using **TRANSACTION SE11**.

To build the layout of the output list, use the form routine shown in Figure 4.41.

```

*&-----*
*&      Form  alv_layout_build
*&-----*
*      Format the layout of ALV-Grid-Controls
*-----*
*      -->GS_LAYOUT  layout format
*-----*
FORM alv_layout_build USING    gs_layout TYPE slis_layout_alv.

gs_layout-zebra              = 'X'.
gs_layout-cell_merge          = ' '.
gs_layout-colwidth_optimize   = ' '.
gs_layout-box_fieldname       = space.
gs_layout-no_vline            = 'X'.
gs_layout-no_colhead          = ' '.
CLEAR gs_layout-lights_fieldname.

ENDFORM.                      " alv_layout_build

```

Figure 4.41: Source code for defining the layout of an ALV grid list

The data layout includes formatting parameters for the list output. You can see more formatting parameters in the Data Dictionary structure “SLIS\_LAYOUT\_ALV.”

Along with providing the ALV grid list, you have to register events on which the ALV grid control reacts. This is necessary, for example, to print a list header, and requires you to register the event “TOP-OF-PAGE” (see Figure 4.42).

```

*&-----*
*&      Form  eventtab_build
*&-----*
*      registration of events
*-----*
*      -->GT_EVENTS[]  table of events
*-----*
FORM eventtab_build USING    gt_events TYPE slis_t_event.

DATA: ls_event TYPE slis_alv_event.

CALL FUNCTION 'REUSE_ALV_EVENTS_GET'
  EXPORTING
    i_list_type = 0
  IMPORTING
    et_events   = gt_events.

READ TABLE gt_events WITH KEY name =  slis_ev_top_of_page
  INTO ls_event.

IF sy-subrc = 0.
  MOVE 'TOP_OF_PAGE' TO ls_event-form.
  APPEND ls_event    TO gt_events.
ENDIF.

ENDFORM.                      " eventtab_build

```

Figure 4.42: Registration of the event for outputting a list header

There are many events to which the ALV grid control can react, such as:

- ▶ PF\_STATUS\_SET = Enhances the PF-Status
- ▶ USER\_COMMAND = Reacts to own function codes of the PF\_STATUS\_SET
- ▶ TOP\_OF\_PAGE = Reacts to a top of page
- ▶ END\_OF\_PAGE = Reacts to an end of page
- ▶ TOP\_OF\_LIST = Reacts to the begin of a list output
- ▶ END\_OF\_LIST = Reacts to the end of a list output

Finally, define which layout view uses the output and whether or not a user can create custom layouts. Use the variables I\_SAVE and IS\_VARIANT.

The following definitions are possible:

- ▶ I\_SAVE = space, IS\_VARIANT = space  
Only the current layout can be changed
- ▶ I\_SAVE = space, IS\_VARIANT = 'Layout\_Name'  
A delivered layout can be used
- ▶ I\_SAVE <> space, IS\_VARIANT = space  
A delivered layout can be used
- ▶ I\_SAVE = X, IS\_VARIANT = 'Layout\_Name'  
Only delivered layouts can be stored (valid for all users)
- ▶ I\_SAVE = U, IS\_VARIANT = 'Layout\_Name'  
The user can store user-defined layouts
- ▶ I\_SAVE = A, IS\_VARIANT = 'Layout\_Name'  
Delivered and user-defined layouts can be stored

To print a list header, you have to implement two form routines (see Figure 4.43).

```

*&-----*
*&      Form   TOP_OF_PAGE
*&-----*
*      header of output
*-----*
FORM top_of_page.                                     "#EC CALLED

  PERFORM comment_build  USING gt_list_top_of_page[].

  CALL FUNCTION 'REUSE_ALV_COMMENTARY_WRITE'
    EXPORTING
      it_list_commentary = gt_list_top_of_page.

ENDFORM.                                              " TOP_OF_PAGE

```

*Figure 4.43: Form routine for the event TOP\_OF\_PAGE of the ALV grid list*

The form routine “TOP-OF-PAGE” will not be directly called by the program, but implicitly used by the construction of the list, if the event “TOP-OF-PAGE” is raised (see Figure 4.44).

```

FORM comment_build USING gt_top_of_page TYPE slis_t listheader.

  DATA: ls_line      TYPE slis_listheader,
        lv_date(10) TYPE c,
        lv_time(8)  TYPE c,
        lv_sdt(2)   TYPE c,
        lv_min(2)   TYPE c,
        lv_sec(2)   TYPE c,
        lv_kopf(45) TYPE c.

    REFRESH gt_top_of_page[].

* Header
lv_kopf = 'Validity of Contracts'.

CLEAR ls_line.
ls_line-typ = 'H'.
ls_line-key = ''.
ls_line-info = lv_kopf. "CHAR 60
APPEND ls_line TO gt_top_of_page.

* Headerinfo: Typ S
CLEAR ls_line.
ls_line-typ = 'S'.

ls_line-key = 'Name:'.
ls_line-info = sy-uname.
APPEND ls_line TO gt_top_of_page.

WRITE sy-datum TO lv_date.
ls_line-key = 'Date:'.
ls_line-info = lv_date.
APPEND ls_line TO gt_top_of_page.

WRITE sy-datum TO lv_date.
lv_sdt = sy-uzeit+0(2).
lv_min = sy-uzeit+2(2).
lv_sec = sy-uzeit+4(2).
CONCATENATE lv_sdt ':' lv_min ':' lv_sec INTO lv_time.

ls_line-key = 'Time:'.
ls_line-info = lv_time.
APPEND ls_line TO gt_top_of_page.

ls_line-key = 'Analys. Contracts:'.
ls_line-info = gv_anzahl.
APPEND ls_line TO gt_top_of_page.

ENDFORM.          " comment build

```

Figure 4.44: Design the header information of the ALV grid list

The form routine “COMMENT\_BUILD” is called by the routine “TOP-OF-PAGE”.

After these development steps, the analysis of some contract accounts shows the following design (see Figure 4.45).

The screenshot shows a SAP application window with the title 'Demoprogram to Explain Assign-Command'. The main area displays a table titled 'Validity of Contracts' with 16 rows of data. The columns are: Lfd-Nr., BPartner, Contract Acct, Loevm VK, Contract, MvOut date, Vt. abger., Loevm Vert, Vertr. abg, Vertr. ung, and Ergebnis. The 'Ergebnis' column contains the text 'contract has been settled' for all rows. The data is as follows:

Lfd-Nr.	BPartner	Contract Acct	Loevm VK	Contract	MvOut date	Vt. abger.	Loevm Vert	Vertr. abg	Vertr. ung	Ergebnis
1	10000289	20000068		30000120	01.01.2004	X			X	contract has been settled
2	10000289	20000068		30000121	01.01.2004	X			X	contract has been settled
3	10001558	20000069		30000123	01.01.2002	X			X	contract has been settled
4	10001558	20000069		30000122	01.01.2002	X			X	contract has been settled
5	10001559	20000070		30000124	31.12.9999					
6	10001559	20000070		30000125	31.12.2009	X			X	contract has been settled
7	10001560	20000071		30000126	31.12.9999					
8	10001560	20000071		30000127	31.12.2009	X			X	contract has been settled
9	10000289	20000072		30000128	01.01.2004	X			X	contract has been settled
10	10000289	20000072		30000129	01.01.2004	X			X	contract has been settled
11	10000289	20000073		30000131	31.12.2009	X			X	contract has been settled
12	10000289	20000073		30268583	31.12.9999					
13	10000289	20000073		30000130	31.12.9999					
14	10000289	20000073		30314889	31.12.9999					
15	10000289	20000074		30000132	01.01.2004	X			X	contract has been settled
16	10000289	20000074		30000133	01.01.2004	X			X	contract has been settled

Figure 4.45: ALV grid list with analyze results

This standard ALV grid control contains some helpful functions, which are offered on the standard icon-strip:

- Details of a marked data set
- Sorting marked data sets as descending or ascending
- Filtering data sets: the user has to choose a column to define the filter and then enter a value for the filter
- Print preview of the list
- Shows the data in an Excel List, if Microsoft Excel is installed on the system
- Shows the data in Microsoft Word if this application is installed (personalized letterhead is provided)
- Allows the output list to send within the SAP system and to an e-mail account
- Builds bar charts from marked rows and columns
- Enables the user to select, change, and store layouts of the output list (see the explanations of the variables I\_SAVE and IS\_VARIANT above).

Using the pull-down menu **LIST, THEN** the function **EXPORT**, stores the list in a local file or exports to a table calculation and to a word processing application.

To sum up: By using ALV grid controls, you can provide useful output to the user with some simple developments. If the developer uses a template, he can provide the data output quickly – faster than using the WRITE command.

#### 4.11.2 ALV grid list in dialog programming

ALV grid lists in dialog processing have to be developed using standard SAP classes and their methods instead of function modules. In comparison to the function modules for the list processing, the methods for ALV grid lists in dialog programming have considerably more functions. For example, the ALV grid control reacts to a double click in the output list.

##### React on a double click in an ALV grid list



I use the example program ZCU\_ABAP\_ALV\_GRID\_SCREEN. In the selection screen, you can enter business partners, contract accounts, and contracts. The program selects business partners, contract accounts, and contracts and checks whether the contracts are valid. The results list is printed in the form of an ALV grid list. With a double click on a contract number in the ALV grid list, the standard SAP program to display contracts is called and displays the selected contract with its attributes. If the user ends the display of the contract, the program returns to the ALV grid list.

You can find a row of programs in the standard SAP development package “SALV\_OM\_OBJECTS”. These programs can be displayed by using the term “SALV\_DEMO\*” and the “F4” help key in **TRANSACTION SE80**.

The following development steps are necessary for the ALV grid list in dialog programming:

1. Instead of function modules, use the statement CALL SCREEN <screen\_number> to create a screen, e.g., CALL SCREEN 0100 (don't use the number 1000, this number is reserved for the selection screen!).

2. For the treatment of the double click on the screen, you have to define and implement a local class.
3. Declare the necessary variables.
4. Before calling the screen, a standard SAP object for the event “Double\_Klick” has to be generated and registered.

The course logic of a screen contains the following steps:

- ▶ Process before output (PBO) module, regularly this module contains the ABAP coding for the screen title and the PF-Status (that means function codes in the screen which can be chosen by the user).
  - ▶ Screen with the elements to be displayed on the screen
  - ▶ Process after input (PAI) module, the ABAP coding for the reaction on input from the user (that means get the function codes, which are defined in a PBO module and develop the requested function).
5. Create an area for the output of the ALV grid list on the screen with the layout editor.
  6. Define a title for screen and a *PF-status* in a PBO module. This PF-status contains the pull-down menu, the button strip, the icon strip, and the allocation of the function keys.
  7. In a second PBO module, fill in an ALV grid container for the delivery of the data to be indicated.
  8. In a third PBO module, an internal table with data to display is dispatched to screen.
  9. In a PAI module, the developer has to react to the user’s input.
  10. Finally, a method has to define for the requested treatment of the event “Double\_klick.”

These steps are described in detail next.

We define and implement the local class for the treatment of the double click with an include. This include can be used in other programs. Name this include ZCU\_ABAP\_LOCAL\_CLASS\_DOB\_KLICK and use this include in front of the declaration part for the ALV grid objects (see Figure 4.46).

```

[+] *-----
|   * Include of class for treatment of double click on ALV-Grid
|   *
[+] INCLUDE zcu_abap_lcl_class_double_cl.

```

Figure 4.46: Include for the definition and implementation of a local class

This include has the coding shown in Figure 4.47.

```

[+] *&-----*
|   * & Include          ZCU_ABAP_LCL_CLASS_DOUBLE_CL
|   *-----*
|   *-----*
|   * &   Class definition for treatment of the event double-click
|   * &   ALV-Grid-Control
|   *-----*
[+] CLASS lcl_alv_grid DEFINITION.

    PUBLIC SECTION.
        METHODS on_dbclick
            FOR EVENT double_click
            OF cl_gui_alv_grid
            IMPORTING e_row.

    ENDCLASS.           "lcl_alv_grid DEFINITION
[+] *-----*
|   *     CLASS lcl_alv_grid IMPLEMENTATION
|   *-----*
|   *     Implementation for treatment of double-click
|   *-----*
[+] CLASS lcl_alv_grid IMPLEMENTATION.

    METHOD on_dbclick.
        *   Call Transaktion ES22 to display contract of the line
        *   in case of double-click on ALV-Grid-Line
        READ TABLE      gt_output INDEX e_row-index INTO gs_output.
        SET PARAMETER ID 'VTG'  FIELD gs_output-vertrag.
        CALL TRANSACTION 'ES22' AND SKIP FIRST SCREEN.
        CLEAR e_row.

    ENDMETHOD.          "on_dbclick

ENDCLASS.           "lcl_alv_grid IMPLEMENTATION

```

Figure 4.47: Local class for the treatment of the double click on a line of the ALV grid list

This method guarantees that the selected contract number is received for calling **TRANSACTION** ES22 (display contract). When using the command call transaction for ES22 the contract number is passed to the **PARAMETER ID** VTG. Find out the name of the **PARAMETER ID** by hitting the “F1” help key on the **CONTRACT** selection screen of **TRANSACTION** ES22. Clicking the **Technical Information** button will show the **PARAMETER ID** in **FIELD DATA** (see Figure 4.48).



Figure 4.48: Find the parameter ID

For the output of the ALV grid list on the screen, the declarations shown in Figure 4.49 are necessary.

```

*-----*
* Objekt declarations for ALV Grid Control
*-----*

DATA: go_custom_container      TYPE REF TO cl_gui_custom_container,
      go_sap_grid            TYPE REF TO cl_gui_alv_grid,
      go_sap_dbclick          TYPE REF TO lcl_alv_grid,
      gs_alv_layout           TYPE lvc_s_layo,
      gs_alv_variant          TYPE disvariant,
      gv_alv_save(1)          TYPE c,
      ok_code                 TYPE sy-ucomm.

```

Figure 4.49: Declarations of variables for the ALV grid on the screen

Now we come to the necessary preliminary work that happens before the command “CALL SCREEN” (see Figure 4.50).

The work includes the default to store layout variants, the creation of an object for the double click on the screen with the name “go\_sap\_dbclick,” and registering the

“double\_click” as a screen event.

```
-----  
*->      Form  OUTPUT_RESULT  
*->  
*      Output of analyse result in ALV-Grid  
*-----  
* -->  gt_output Output-table  
*-----  
FORM output_result.  
*-----  
*      Numbers of datasets  
*-----  
    DESCRIBE TABLE gt_output LINES gv_number.  
*-----  
* define assignments for ALV-Grid-Control  
* with this settings display-layout can be saved  
*-----  
    gs_alv_variant-report = sy-cprog.  
    gv_alv_save = 'A'.  
*-----  
* Create an object for the event of double-click  
* on the ALV-Grid-Control  
*-----  
    CREATE OBJECT go_sap_dbclick.  
*-----  
* Register of event double-click  
*-----  
    SET HANDLER go_sap_dbclick->on_dbclick FOR ALL INSTANCES.  
    CALL SCREEN 0100.  
ENDFORM.          "OUTPUT_RESULT
```

Figure 4.50: Source code before the command CALL SCREEN

Next, the screen is called with the command CALL\_SCREEN 0100 for the ALV grid list output.

With a double click on the number “0100,” the ABAP editor creates a screen in the current program.

A screen with three tabs appears (see Figure 4.51):

- ▶ **ATTRIBUTES** of the screen, include the screen title **SCREEN TYP** (normal, sub-screen, or modal dialog box), and the number of the **NEXT SCREEN**
- ▶ **ELEMENT LIST** of the screen, identifies which objects are displayed on the screen

- **Flow Logic** on the screen, calls different subroutines in front of the screen display as well as after the input of commands of the user

Screen number **100** Active

Attributes Element list Flow logic

Short Description	Testscreen for display ALV-Grid-List	
Original Language	EN English	Package ZCU_ABAP_EDUCATION
Last changed on/at	04.03.2015	13:25:54
Last Generation	04.03.2015	13:25:54

<b>Screen Type</b>	<b>Settings</b>
<input checked="" type="radio"/> Normal <input type="radio"/> Subscreen <input type="radio"/> Modal dialog box <input type="radio"/> Selection screen	<input type="checkbox"/> Hold Data <input type="checkbox"/> Switch Off Runtime Compress <input type="checkbox"/> Template - non-executable <input type="checkbox"/> Hold Scroll Position <input type="checkbox"/> Without Application Toolbar

<b>Other Attributes</b>		
Next Screen	100	
Cursor Position		
Screen Group		
Lines/Columns	Occupied 32	157
	Mainten. 33	157
Context Menu FORM ON CTMENU		<b>Properties</b>

Figure 4.51: Attributes of the ALV grid screens

In this example, choose the “Normal” screen with the name **NEXT SCREEN 100**. So that after completing the screen, the same screen 100 is displayed to the user again by clicking the “**Enter**” key if no PAI module is developed with other steps.

Now we change to the screen layout with a click on the **Layout** button (see Figure 4.52).

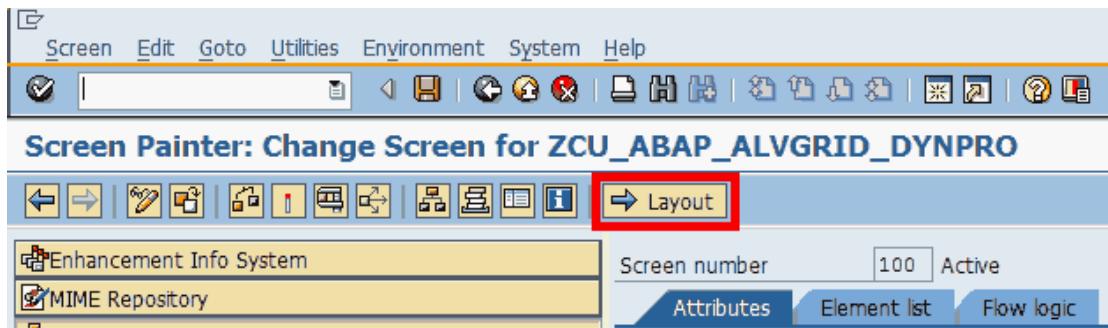


Figure 4.52: Call the layout painters

The system displays a graphical layout painter. The toolbar of this painter is on the left side of the screen (see Figure 4.53).

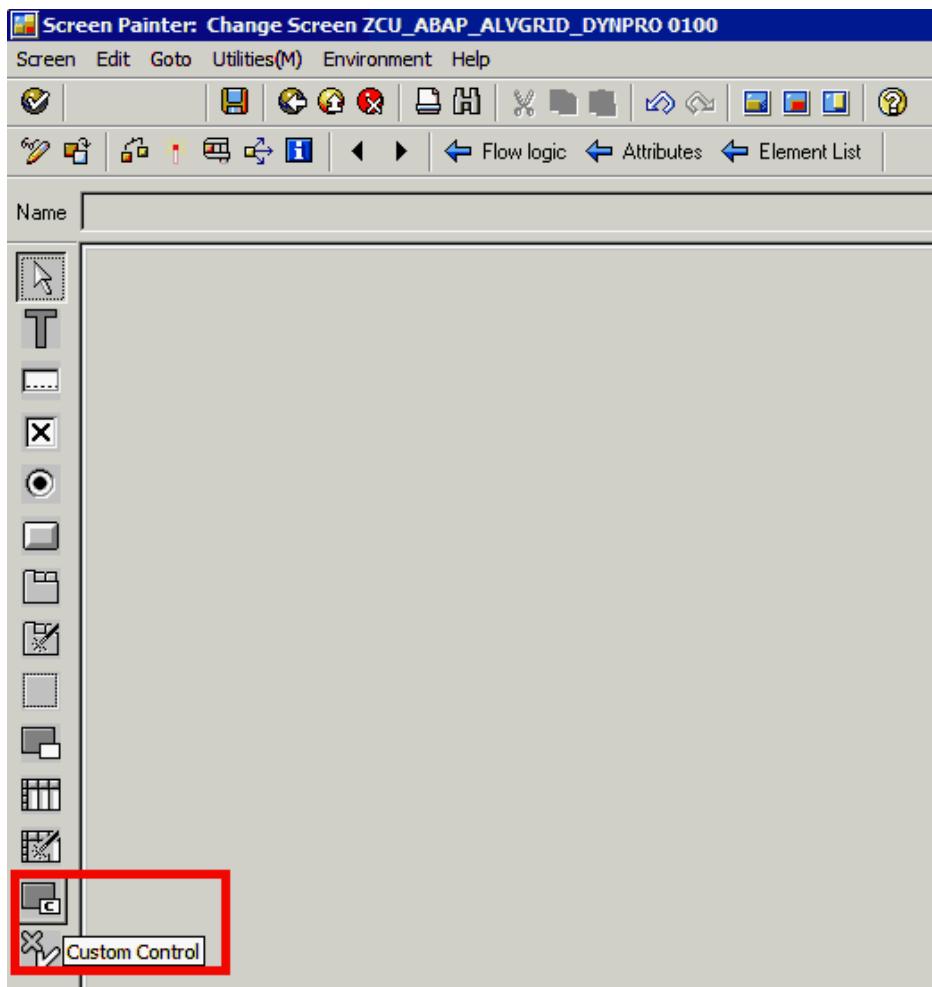


Figure 4.53: Custom control element in the toolbar

We use the last icon (**CUSTOM CONTROL**, identified in Figure 4.53) to create a screen area for the place of the “Custom Control.” Using this tool, we can draw an area for the output of the ALV grid control (see Figure 4.54).

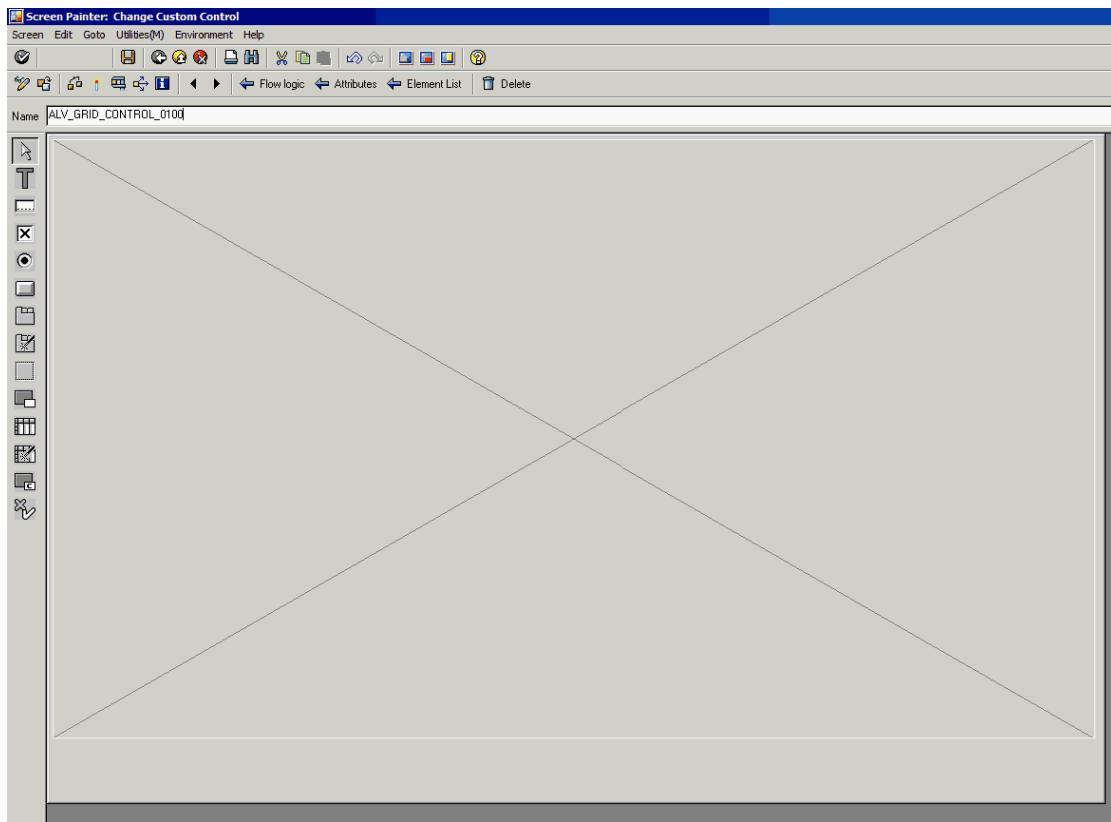


Figure 4.54: Area for the output of a custom control element on the screen

In the input line, give this screen area the name ALV\_GRID\_CONTROL\_0100.

For the output of the amount of shown data sets in the list, we now use the icon found in the tool strip. This text field gets the name “Numbers of Selected Data Sets,” and it is placed in a free area beneath the ALV grid control.

The output of the amount of selected data is missing.

Click the icon from the upper icon strip to open a new modal screen. This screen allows access to dictionary elements and internal variables. Look for the global program variable GV\_NUMBER, which is displayed under the ALV grid list. Enter this variable name into the **TABLE-/FIELDNAME** field and click the button (see Figure 4.55).

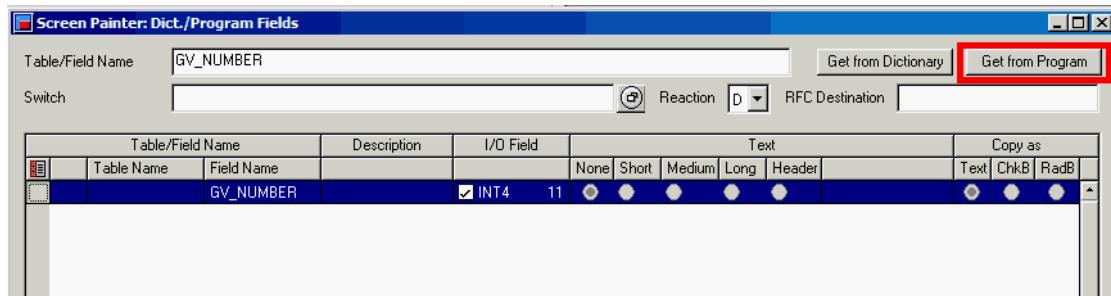


Figure 4.55: Choose program variables for the screen

The searched variable is shown in a list – if it exists – in the program as a global variable. Mark this variable in the first column, then drag and drop it into a free place under the custom control element.

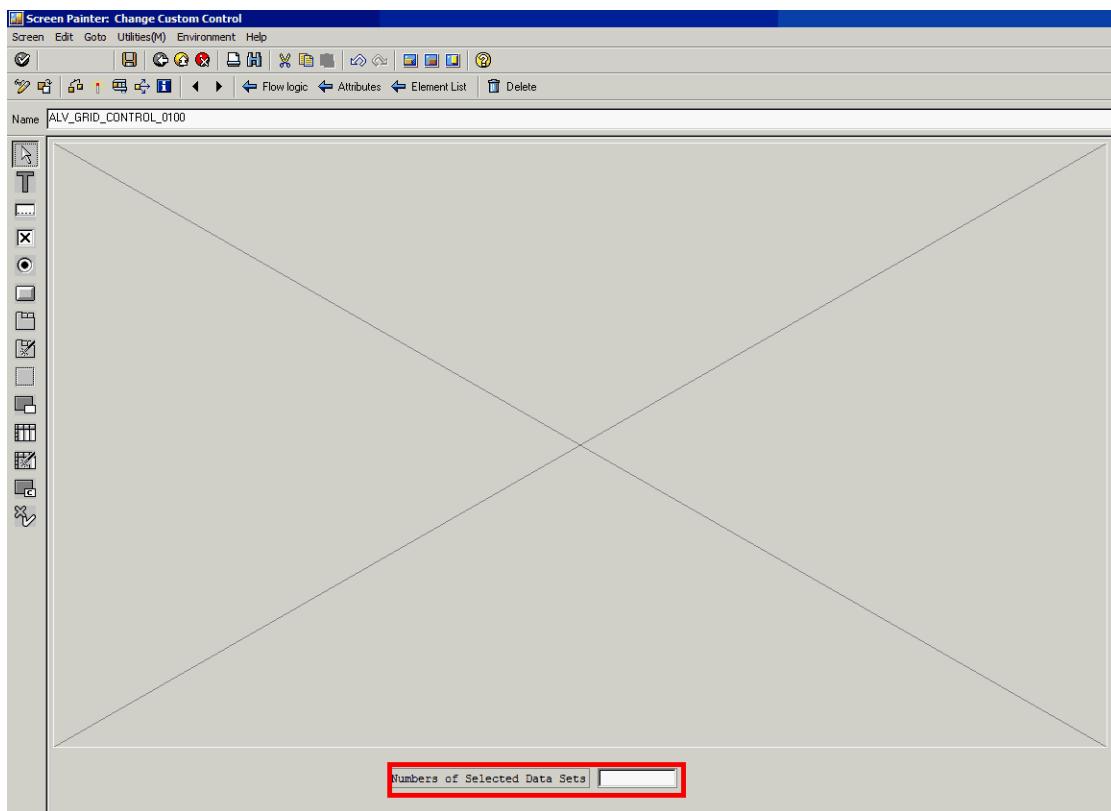


Figure 4.56: Placement of a program field on the screen

Close the modal screen for the dictionary and program fields.

The field for output of the “Numbers of Selected Data Sets” (GV\_NUMBER) is ready for input, but it should be an output field. With a double click on this field, the system opens a modal window for changing the attributes of this field (see Figure 4.57).

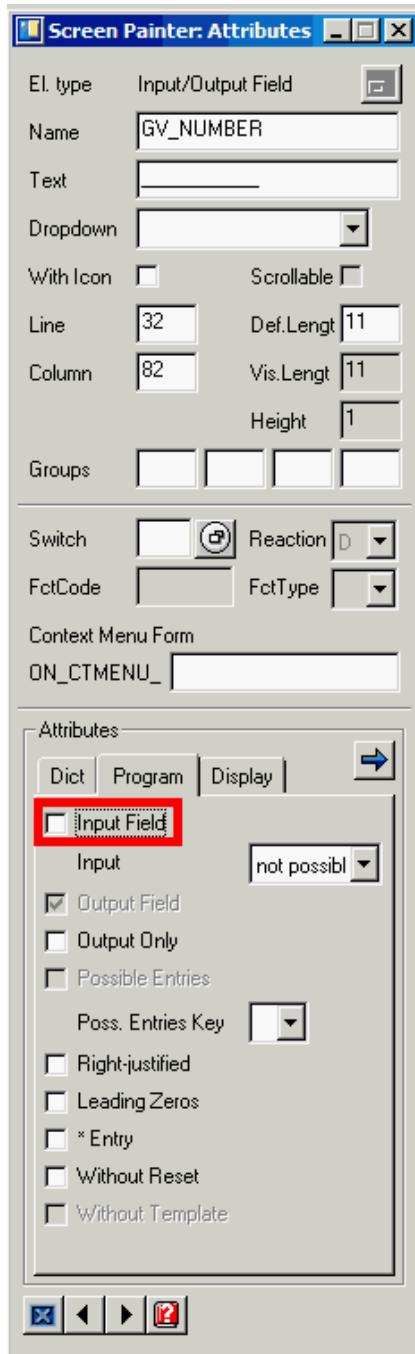


Figure 4.57: Field attributes of a program field

Remove the check mark in front of the field **INPUT FIELD**. The field changes to grey, meaning the variable GV\_NUMBER is no longer ready for input and serves only as output. Close the modal attribute screen. Click on the icon in the Screen Painter to activate the changes on the screen. Close the Screen Painter window.

Now move to the second tab of the screen display with the name **ELEMENT LIST**. You can see all the screen elements that have to be defined and placed on the screen with the help of the Screen Painter (see Figure 4.58).

H...	M...	Name	Type...	Line	Co...	De...	Vis...	He...	Sc...	Format	In...	O...	Out...	Di...	Dict...	Property list
+		ALV GRID CONTROL_0100	CCtrl	1	1	156	156	30								
		NUMBER	Text	32	51	29	29	1								Properties
		GV_NUMBER	I/O	32	82	11	11	1	INT4			✓				Properties
		OK_CODE	OK	0	0	20	20	1	OK							Properties

Figure 4.58: Variable OK\_CODE as an element on the screen

Enter a new screen element named OK\_CODE to the element list. In this field in the program, enter the function code, which is released by an action on the screen. The possible function codes are defined in the PF-status (see below). The variable OK\_CODE gets the typification SY-UCOMM in the declaration part of the program:

DATA: ok\_code TYPE sy-ucomm.

Now move to the third tab of the screen display named **FLOW LOGIC**. The standard SAP program generates the flow logic shown in Figure 4.59.

```

1  *-----*
2  PROCESS BEFORE OUTPUT.
3
4  *  MODULE status_0100.
5
6  *-----*
7  PROCESS AFTER INPUT.
8
9  *  MODULE user_command_0100.
10

```

Figure 4.59: Flow logic of a screen

Remove the commentary sign (\*) in front of the screen modules. With a double click on the module name, the corresponding module (*PBO module*, *PAI module*) is created automatically in the program code.

PBO modules consist of ABAP code, which prepare the output of a screen.

```
*->-----  
*&      Module STATUS_0100 OUTPUT  
*&-----  
*      Definition PF-Status of Screen 0100  
*-----  
MODULE status_0100 OUTPUT.  
  
SET PF-STATUS 'PF_100'.  
  
ENDMODULE.           " STATUS 0100 OUTPUT
```

Figure 4.60: Definition of the PF-status for a screen

In module STATUS\_PF\_<screen\_nr> the PF-status is defined: it contains the allocation of the functional keys as well as the button and icon strips on the screen (*GUI status*). For this example, we need a simple PF-status (with the icons (BACK), (EXIT), and (CANCEL)). With the output of the standard ALV grid list, the user gets enough menus and functional icons (see Figure 4.61).

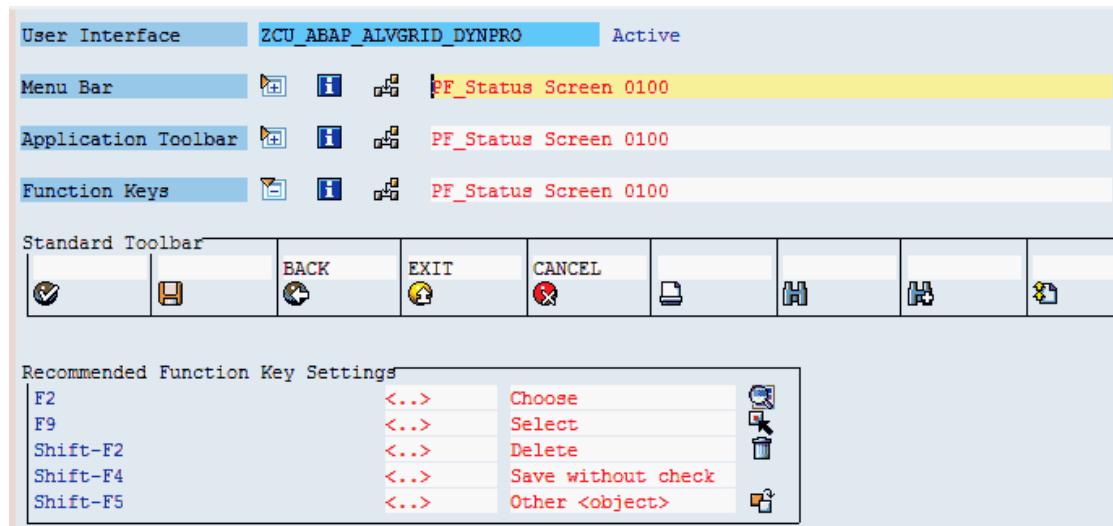


Figure 4.61: Elements of PF-status

For the ALV grid list, you need two more PBO modules.

The first module is needed to create two object instances: one for the ALV grid container and one for the ALV grid list (see Figure 4.62).

```

*&-----+
*&      Module  CREATE_ALV_OBJECT  OUTPUT
*&-----+
*      Create Container-Object for ALV-Grid-Control
*-----+
MODULE create_alv_object OUTPUT.

IF go_custom_container IS INITIAL.

CREATE OBJECT go_custom_container
EXPORTING
  container_name = 'ALV_GRID_CONTROL_0100'.

CREATE OBJECT go_sap_grid
EXPORTING
  i_parent = go_custom_container.

ENDIF.

ENDMODULE.          " CREATE_ALV_OBJECT  OUTPUT

```

Figure 4.62: Creating instances of an ALV container object and an ALV grid object

The second module is needed to move the selected data sets of the output table **GT\_OUTPUT** to the ALV grid object “go\_sap\_grid.”

By using the structure “is\_layout,” you can make default settings for the layout of the ALV grid list. In this example (see Figure 4.63), set the selection mode. A column on the left side of the ALV grid list appears. With this column, the user can select data sets. Now optimize the column width and enter a title for the ALV grid list.

```

*&-----*
*& Module TRANSFER_ALV_DATA OUTPUT
*&-----*
* PBO data transfer of internal Tabelle gt_output to ALV-Grid Control
* to save display variants on ALV-Grid-Control the parameter
* IS_VARIANT and I_SAVE must be passed.
*-----*
MODULE transfer_alv_data OUTPUT.

* define layout parameters
* Buttons on the left edge of ALV-Grid-List for selection of rows
gs_alv_layout_sel_mode = 'A'.
* Optimierung der Spaltenbreite
gs_alv_layout_cwidth_opt = 'X'.
* Titel des ALV-Grid
gs_alv_layout_grid_title = 'Demo program ALV-Grid on Dynpro'.

CALL METHOD go_sap_grid->set_table_for_first_display
EXPORTING
  i_structure_name = 'ZCU_ABAP_ASSIGN'
  is_variant      = gs_alv_variant
  is_layout       = gs_alv_layout
  i_save          = gv_alv_save
CHANGING
  it_outtab      = gt_output.

ENDMODULE.          " TRANSFER_ALV_DATA OUTPUT

```

Figure 4.63: Move selected data sets from the internal table GT\_OUTPUT to the ALV grid list on the screen

Figure 4.64 shows the coding of the flow logic of the screen “0100.”

```

1 *-----
2 PROCESS BEFORE OUTPUT.
3
4 MODULE status_0100.
5
6 MODULE create_alv_object.
7
8 MODULE transfer_alv_data.
9
10 *-----
11 PROCESS AFTER INPUT.
12
13 MODULE user_command_0100.
14
15 *-----

```

Figure 4.64: Ready flow logic of the ALV grid screen

To address user requests on the screen, develop a PAI module (see Figure 4.65). In our example, the icons defined in the PF-Status (BACK), (EXIT), and (CANCEL) have to be caught. The user ends the program when clicking one of these icons.

```

*->-----*
*     Module  USER_COMMAND_0100  INPUT
*-----*
*     Reaction on user input
*-----*
MODULE user_command_0100 INPUT.
CASE ok_code.
    WHEN 'BACK'
    OR   'EXIT'
    OR   'CANCEL'.
        LEAVE PROGRAM.
ENDCASE.
ENDMODULE.                                     " USER COMMAND 0100  INPUT

```

Figure 4.65: PAI module of the ALV grid screen

Now the program is ready to run.

Figure 4.66 shows the result of a program test.

Lfd-Nr.	BPartner	Cont. Acct	Loevn VK Contract	MvOut date	Vt. abger.	Loevm...	Vertrag abgerechnet	Vertrag ungültig	Ergebnis
1	10000360	20115104	30348961	31.07.2012	X			X	Contract has been settled
2	10000361	20004783	30010341	31.12.9999					
3	10000361	20004783	30010342	31.12.2009	X			X	Contract has been settled
4	10000363	20004858	30107783	25.10.2009	X			X	Contract has been settled
5	10000363	20004858	30010475	25.10.2009	X			X	Contract has been settled
6	10000364	20004879	30150247	29.06.2011	X			X	Contract has been settled
7	10000364	20004879	30010510	29.06.2011	X			X	Contract has been settled
8	10000364	20004879	30329939	29.06.2011	X			X	Contract has been settled
9	10000366	20004883	30010517	31.12.9999					
10	10000366	20004883	30107788	31.12.9999					

Figure 4.66: Output of the ALV grid list screen

With a double click on any contract number, the program calls TRANSACTION ES22 to display the selected contract (see Figure 4.67).

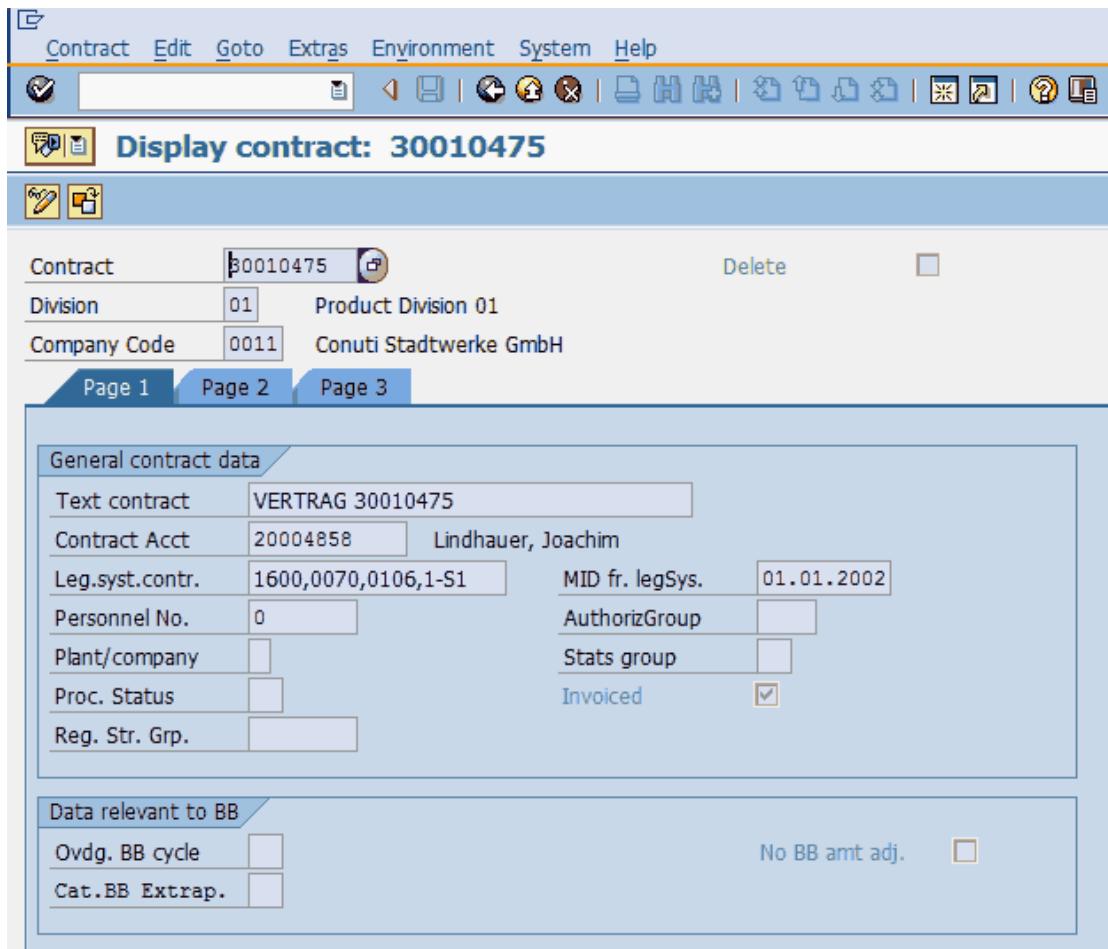


Figure 4.67: Contract details in transaction ES22

If the user leaves this transaction by clicking the (BACK) or (EXIT) icon, the program goes back to the ALV grid list display.

In this example, use the event “double\_klick.” You can create more events, such as:

- ▶ before\_user\_command: Change standard function keys.
- ▶ set\_frontend\_fieldcatalog: Change the output structure.
- ▶ set\_frontend\_layout: Define the layout of the output table.
- ▶ set\_filter\_criteria: Define filters for the output data.
- ▶ print\_top\_of\_list: Print the output at the top of the ALV grid list.
- ▶ button\_klick: Catch the mouse click on a button of the ALV grid control.
- ▶ double\_klick: Catch a double click on a row of the ALV grid control.
- ▶ hotspot\_klick: Catch a previously defined hotspot click on defined columns.

- ▶ `user_command`: Catch function codes.
- ▶ `toolbar`: Change, delete, or insert GUI elements to the toolbar.
- ▶ `menu_button`: Define menus of buttons in the toolbar.
- ▶ `after_user_command`: Catch own or standard function codes.

(Note that building ALV grid lists in a screen is more expensive than creating ALV grid lists with function modules, but it offers more functionality)

## 4.12 Download and upload data

Often, data from SAP has to be *downloaded* in an external file to work with the data in other applications. Similarly, an *upload* of data from a third-party application into SAP is also common. The files for data download or upload can be stored on a network directory of the SAP application server or on a local directory of the workstation.

There are different data formats for each type. The following are the most common:

- ▶ *ASC format*: A table transfers as text. During the transfer conversion, exits will run. The output format depends on other parameters, such as CODEPAGE, TRUNC\_TRAILING\_BLANKS, and TRUNC\_TRAILING\_BLANKS\_EOL. These are used for processing data in applications such as Microsoft Excel or Microsoft Word. Without any format specified, the SAP standard uses this format as the default format.
- ▶ *BIN format*: The data transfers as binary code. There is no other data process or code page conversion. The data is interpreted line by line and not in columns. The length of the data is given in the parameter BIN\_FILESIZE. The table is typified as type “X,” and especially in Unicode systems, the change of structured data into binary data can cause errors.
- ▶ *DAT format*: This data transfer occurs in columns. Like in the ASC format, the text format is used during the transfer. In contrast to the ASC format, no conversion exits are used, and tabulator signs separate columns. This format builds files, which can be uploaded to SAP with the standard function modules gui\_upload or ws\_upload. Use this format if you have to download data from one SAP system and upload to another SAP system.

### 4.12.1 Download data

#### Download of data from an SAP system



For the download of data out of the SAP database table ZGENERALCONTRACT I've developed the program ZCU\_ABAP\_DOWNLOAD\_ZGENERALCONT. With this

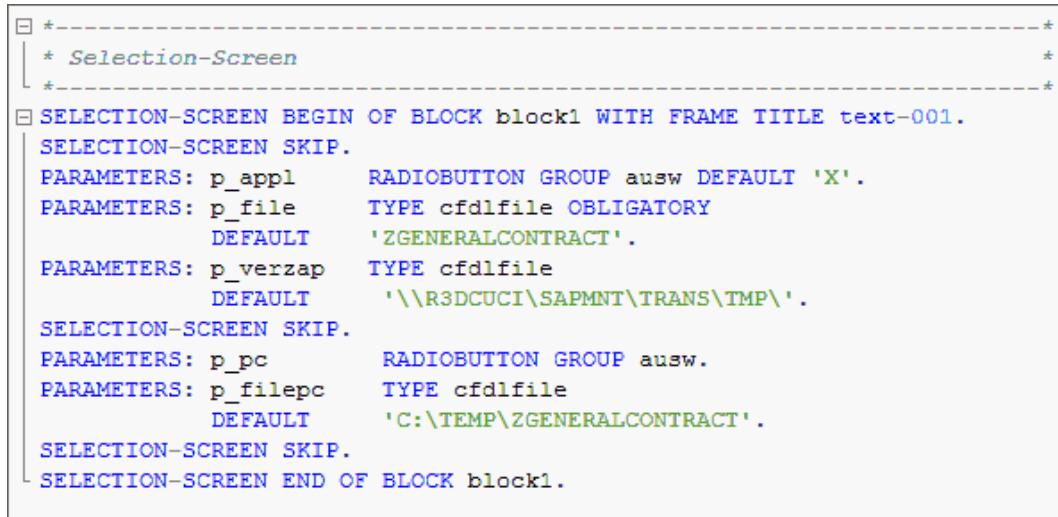
program, the data is downloaded in ASC format to the directory of the workstation or the directory of the application server.

The program owns the following functional parts:

- ▶ Selection screen
- ▶ Data selection
- ▶ Download to the application server
- ▶ Download to the workstation
- ▶ Output of the program result

Like its functional parts, the main program is structured.

The selection screen shows a choice of directories and filenames for the download on the application server or the workstation (see Figure 4.68).



```
□ *-----  
| * Selection-Screen  
| *-----  
□ SELECTION-SCREEN BEGIN OF BLOCK block1 WITH FRAME TITLE text-001.  
| SELECTION-SCREEN SKIP.  
| PARAMETERS: p_appl      RADIOBUTTON GROUP ausw DEFAULT 'X'.  
| PARAMETERS: p_file      TYPE cfdlfile OBLIGATORY  
|             DEFAULT 'ZGENERALCONTRACT'.  
| PARAMETERS: p_verzap    TYPE cfdlfile  
|             DEFAULT '\\R3DCUCI\SAPMNT\TRANS\TMP\'.  
| SELECTION-SCREEN SKIP.  
| PARAMETERS: p_pc        RADIOBUTTON GROUP ausw.  
| PARAMETERS: p_filepc   TYPE cfdlfile  
|             DEFAULT 'C:\TEMP\ZGENERALCONTRACT'.  
| SELECTION-SCREEN SKIP.  
□ SELECTION-SCREEN END OF BLOCK block1.
```

Figure 4.68: Selection screen for the input of target directory and target file for download

With **TRANSACTION AL11**, call the default directory. With this transaction, the user has access to the network file system of the SAP server. It is common for the download directory to have the name `\<system-name>\SAPMNT\TRANS\TMP\`.

This directory is mapped to a network directory, so the user has access to the downloaded file. Ask your SAP Basis Team for directories specific to your company.

The standard directory for the download on your workstation is typically the directory C:\TEMP. Not all users are working directly on the file system of the intranet. Sometimes a user needs remote access via the Internet to the SAP system. The access to the download directory is guaranteed for the remote working user through the standard SAP function module WS\_FILENAME\_GET at the program time AT SELECTION-SCREEN ON VALUE-REQUEST FOR <filename>. With this function, the user can choose a directory by clicking the “F4” help key (see Figure 4.69).

```

-----+
| * At Selection-Screen On Value-Request*
+-----+
AT SELECTION-SCREEN ON VALUE-REQUEST FOR p_filepc.

CALL FUNCTION 'WS_FILENAME_GET'          "#EC *
  EXPORTING
    def_path      = 'C:\TEMP'
    mask         = ',*.*,*.*.'
  IMPORTING
    filename     = p_filepc
  EXCEPTIONS
    inv_winsys   = 1
    no_batch     = 2
    selection_cancel = 3
    selection_error = 4
    OTHERS        = 5.

IF sy-subrc <> 0.
  MESSAGE e001(zcu_isu) WITH 'Failed to open workstation directory'.
ENDIF.
```

Figure 4.69: Standard SAP function module for choosing the target directory of a file to be downloaded

The main program has a simple structure (see Figure 4.70).

```
□ *-----  
| * Start-of-selection  
| *-----  
  
START-OF-SELECTION.  
  
    PERFORM data_selection.  
  
    □ IF p_appl = 'X'.  
        PERFORM download_appl_server.  
    ○ ELSEIF p_pc = 'X'.  
        PERFORM download_pc.  
    ENDIF.  
  
    PERFORM output_result.  
  
□ *-----
```

Figure 4.70: Structure of the main program for download data

You can download data to the application server with the ABAP code shown in Figure 4.71.

```

*&-----*
*&      Form  DOWNLOAD_APPL_SERVER
*&-----*
*      Download of table ZGENERALCONTRACT to application server
*-----*
*  -->  gt_generalcontract  Table which should be download on
*                  Appl.-Server
*-----*
FORM download_appl_server .

  CONCATENATE p_verzap
              p_file
            INTO p_file.

* Open file on application server and download data setsn
  OPEN DATASET p_file FOR OUTPUT IN TEXT MODE.
  IF sy-subrc <> 0.
    WRITE: /'Error opening filei', p_file.
    EXIT.
  ELSE.
    LOOP AT gt_generalcontract INTO gs_generalcontract.
      TRANSFER gs_generalcontract TO p_file.
    ENDLOOP.
    CLOSE DATASET p_file.
  ENDIF.

* Success-/Error flag for download to application srver
  IF sy-subrc = 0.
    gv_success = 'X'.
  ELSE.
    gv_error   = 'X'.
  ENDIF.

ENDFORM.          " DOWNLOAD_APPL_SERVER

```

Figure 4.71: Source code for download data to the application server

You can download the directory to a workstation with the standard SAP function module GUI\_DOWNLOAD. You can easily build the code for using this function (see Figure 4.73) by clicking on the **Pattern** button in the ABAP editor (see Figure 4.72).

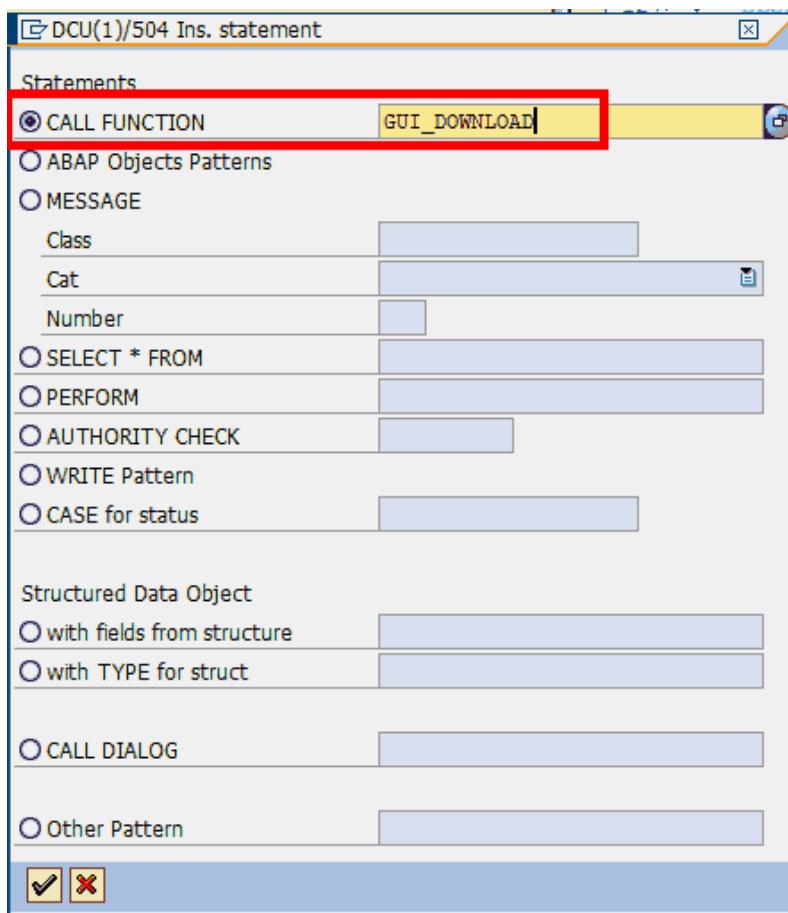


Figure 4.72: Using the pattern function in the ABAP editor

```

FORM download_pc .

DATA: lv_pc_file TYPE string.

lv_pc_file = p_filepc.

CALL FUNCTION 'GUI_DOWNLOAD'
  EXPORTING
    filename           = lv_pc_file
    filetype          = 'ASC'
  TABLES
    data_tab          = gt_generalcontract
  EXCEPTIONS
    file_write_error   = 1
    no_batch           = 2
    gui_refuse_filetransfer = 3
    invalid_type       = 4
    no_authority        = 5
    unknown_error       = 6
    header_not_allowed = 7
    separator_not_allowed = 8
    filesize_not_allowed = 9
    header_too_long     = 10
    dp_error_create     = 11
    dp_error_send       = 12
    dp_error_write      = 13
    unknown_dp_error    = 14
    access_denied       = 15
    dp_out_of_memory    = 16
    disk_full           = 17
    dp_timeout          = 18
    file_not_found      = 19
    dataprovider_exception = 20
    control_flush_error = 21
    OTHERS              = 22.

IF sy-subrc <> 0.
  gv_error      = 'X'.
  gv_errorcode = sy-subrc.
ELSE.
  gv_success = 'X'.
ENDIF.

ENDFORM.          " DOWNLOAD PC

```

Figure 4.73: Source code for the download on a local workstation directory

To download to a workstation with the standard SAP function module GUI\_DOWNLOAD, name the file, the file type, and the table to be output (see Figure 4.73).

For the end of the program, you have to inform the user with WRITE statements, whether the download has been successful or has failed.

```

FORM output_result .

  WRITE: / 'Output of Program Result'.
  WRITE: / 'User',      (15) sy-uname.
  WRITE: / 'Programm', (15) sy-repid.
  WRITE: / 'Date',     (15) sy-datum.
  WRITE: / 'Time',     (15) sy-uzeit.

  IF p_appl = 'X'.
    IF gv_success = 'X'.
      WRITE: / 'Data sets have been successfully downloaded to application server.'.
      WRITE: / 'There have been downloaded a total of', gv_number_data_sets, 'data sets.'.
      ULINE.
      WRITE: / '-----END OF PROGRAM-----'.
    ELSEIF gv_error = 'X'.
      WRITE: / 'Error downloading table ZGENERALCONTRACT to application server.'.
      ULINE.
      WRITE: / '-----END OF PROGRAM-----'.
    ENDIF.
  ELSEIF p_pc ='X'.
    IF gv_success = 'X'.
      WRITE: / 'Data sets have been successfully downloaded to workstation.'.
      WRITE: / 'There have been downloaded a total of', gv_number_data_sets, 'data sets.'.
      ULINE.
      WRITE: / '-----END OF PROGRAM-----'.
    ELSEIF gv_error = 'X'.
      WRITE: / 'Error downloading table ZGENERALCONTRACT to workstation.'.
      WRITE: / 'Errorcode = ', gv_errorcode.
      ULINE.
      WRITE: / '-----END OF PROGRAM-----'.
    ENDIF.
  ENDIF.

ENDFORM.          " OUTPUT_RESULT

```

Figure 4.74: Source code to inform the user about the program results

If the developer owns such a program as a template, he can quickly develop a program to download data from SAP systems. It goes even quicker if you use *dynamic programming* to output data from any database table. Dynamic programming is explained in [Section 4.18](#).

#### 4.12.2 Upload data

Similar to the download methods, you can develop a template program for the upload of data from any application to an SAP system. Refer to the example program **ZCU\_ABAP\_UPLOAD\_ZGENERALCONT**.

The upload of data with this program only runs with CHAR fields. The table ZGENERALCONTRACT owns three date fields with the DATS format. These fields have to be converted to CHAR fields in the SAP program. To convert the fields, you can use an internal program structure or you can build a help structure in the Data Dictionary.

For the sample program, I created a help structure in the Data Dictionary named ZGENERALCONTRACT\_CHAR (see Figure 4.75). In the tab named **COMPONENTS**, you can see that I've changed the date fields from **LENGTH "8"** to CHAR fields of the **LENGTH "12"**.

Component	Typing Method	Component Type	Data Type	Length	Decim..	Short Description
MANDT	Types	MANDT	CLNT	3	0	Client
GENERAL_CONTRACT_NUMBER	Types	ZZ_GENERAL_CONT...	CHAR	10	0	First Customer - General Contract Number
GENERAL_CONTRACT_TITLE	Types	ZZ_GENERAL_CONT...	CHAR	20	0	First Customer - General Contract Title
GENERAL_CONTRACT_KEEPER	Types	ZZ_GENERAL_CONT...	CHAR	12	0	First Customer - General Contract Keeper
ERDAT	Types	ZZ_ERDATS_CHAR	CHAR	12	0	Creation Date
ERNAM	Types	ERNAM	CHAR	12	0	Name of Person who Created the Object
AEDAT	Types	ZZ_AEDATS_CHAR	CHAR	12	0	Change Date (CHAR)
AENAM	Types	AENAM	CHAR	12	0	Name of Person Who Changed Object

Figure 4.75: Help structure for upload data of the database table ZGENERALCONTRACT

Be careful not to upload data sets twice to the database table. Check the table contents before uploading to confirm whether a data set with the same key field **GENERAL\_CONTRACT\_NUMBER** already exists in the database table. If such a data set doesn't exist, you can store an uploaded data set to the database table with a simple **INSERT** statement. If such a data set does exist, check whether the data set in the database has changed. If there are changes, use the **MODIFY** statement. If there is no difference between the uploaded data set and the data set stored in the database, you can ignore the uploaded data set.

The program for the data upload has the following flow logic.

1. View selection screen
2. Upload data from application server
3. Upload data from workstation
4. Check uploaded data sets against the data sets stored in the database table
5. Insert or modify data in the database table
6. Output the program result

Figure 4.76 shows the ABAP code for the corresponding selection screen.

```
*-
* Selection-Screen
*-
SELECTION-SCREEN BEGIN OF BLOCK block1 WITH FRAME TITLE text-001.
SELECTION-SCREEN SKIP.
PARAMETERS: p_appl      RADIOBUTTON GROUP ausw DEFAULT 'X'.
PARAMETERS: p_file      TYPE cfdlfile OBLIGATORY
             DEFAULT 'ZGENERALCONTRACT'.
PARAMETERS: p_verzap    TYPE cfdlfile
             DEFAULT '\\R3DCUCI\SAPMNT\TRANS\TMP\'.
SELECTION-SCREEN SKIP.
PARAMETERS: p_pc        RADIOBUTTON GROUP ausw.
PARAMETERS: p_filepc    TYPE cfdlfile
             DEFAULT 'C:\TEMP\ZGENERALCONTRACT'.
SELECTION-SCREEN SKIP.
SELECTION-SCREEN END OF BLOCK block1.
```

Figure 4.76: Selection screen for data upload

Similar to the download data program, the user can hit “F4” help to choose the source file. You can use the standard SAP function module WS\_FILENAME\_GET again.

The upload program uses simple flow logic.

- ▶ Upload file from the application server or workstation
- ▶ Check the uploaded data
- ▶ Insert or modify data on the corresponding database table
- ▶ Output the program result

```

*&-----*
*&      Form  UPLOAD_APPL_SERVER
*&-----*
*      Upload data file from application server
*-----*
*  --> p_verzap  Directory of application server
* <-- p_file     Directory-/name of file to be uploaded
*-----*
FORM upload_appl_server .

CONCATENATE p_verzap
            p_file
        INTO p_file.

* Open file on Appl.-Server
OPEN DATASET p_file FOR INPUT IN TEXT MODE ENCODING DEFAULT.

IF sy-subrc = 0.
DO.
    READ DATASET p_file INTO gs_generalcontract_char.
    IF sy-subrc = 0.
        MOVE-CORRESPONDING gs_generalcontract_char TO gs_generalcontract.
        APPEND gs_generalcontract TO gt_generalcontract.
    ELSE.
        EXIT.
    ENDIF.
ENDDO.
CLOSE DATASET p_file.
ELSE.
    CONCATENATE 'Error opening file'
                p_file
            INTO gv_message.
MESSAGE i001(zcu_isu) WITH gv_message.
LEAVE PROGRAM.
ENDIF.

* Number uploaded data sets
DESCRIBE TABLE gt_generalcontract LINES gv_number_upload.

ENDFORM.          " UPLOAD_APPL SERVER

```

Figure 4.77: Source code for the upload of data from the application server

As noted in the introduction, the upload structure GS\_GENERALCONTRACT\_CHAR only owns CHAR fields. The following MOVE-CORRESPONDING converts the date fields to the DATS format of the database.

The ENCODING option at the end of the OPEN DATASET statement is only necessary in Unicode systems.

To upload data from the workstation, use the standard SAP function code GUI\_UPLOAD (see Figure 4.78).

```

*&
*&      Form  UPLOAD_PC
*&
*      Upload of selected file from workstation
*-----
*  --> p_filepc          Directory and filename of workstation
* <-- gt_generalcontract Uploaded data sets
*-----
FORM upload_pc .

DATA: lv_filename TYPE string.

lv_filename = p_filepc.

CALL FUNCTION 'GUI_UPLOAD'
  EXPORTING
    filename           = lv_filename
    filetype          = 'ASC'
  TABLES
    data_tab          = gt_generalcontract
  EXCEPTIONS
    file_open_error   = 1
    file_read_error   = 2
    no_batch          = 3
    gui_refuse_filetransfer = 4
    invalid_type      = 5
    no_authority      = 6
    unknown_error     = 7
    bad_data_format   = 8
    header_not_allowed = 9
    separator_not_allowed = 10
    header_too_long   = 11
    unknown_dp_error  = 12
    access_denied     = 13
    dp_out_of_memory  = 14
    disk_full         = 15
    dp_timeout        = 16
    OTHERS            = 17.

  IF sy-subrc <> 0.
    MESSAGE e002(zcu_isu) WITH 'Error upload from workstation with errorcode:' sy-subrc.
  ENDIF.
* Number of uploaded data sets
  DESCRIBE TABLE gt_generalcontract LINES gv_number_upload.

ENDFORM.          " UPLOAD PC

```

Figure 4.78: Source code for upload data from the local workstation

In this program, you have to convert the parameter filename because the standard SAP function module expects a type string variable for the filename.

After data upload, use a form routine to check the uploaded data against the stored data in the database table.

### Prevent the insertion of double data sets

No matter if it is an upload or download, you need to prevent the program from inserting the same data set twice to the database. If data is uploaded twice, the program crashes.



In the form routine CHECK\_DATA (see Figure 4.79), the uploaded data is stored in the internal table GT\_INSERT, GT MODIFY, or GT\_DO NOTHING depending on the check result.

```
FORM check_data.

DATA: ls_generalcontract_db TYPE zgeneralcontract.

IF gv_number_upload > 0.

REFRESH gt_insert.
REFRESH gt_modify.
REFRESH gt_do_nothing.

gv_number_insert      = 0.
gv_number_modify      = 0.
gv_number_do_nothing = 0.

LOOP AT gt_generalcontract INTO gs_generalcontract.
  SELECT SINGLE *
    FROM zgeneralcontract
    INTO ls_generalcontract_db
    WHERE general_contract_number = gs_generalcontract-general_contract_number.

  IF sy-subrc <> 0.
    APPEND gs_generalcontract TO gt_insert.
    CONTINUE.
  ELSE.
    IF ls_generalcontract_db = gs_generalcontract.
      APPEND gs_generalcontract TO gt_do_nothing.
      CONTINUE.
    ELSE.
      APPEND gs_generalcontract TO gt_modify.
      CONTINUE.
    ENDIF.
  ENDIF.
  ENDLOOP.

ENDIF.

DESCRIBE TABLE gt_generalcontract LINES gv_number_upload.
DESCRIBE TABLE gt_insert          LINES gv_number_insert.
DESCRIBE TABLE gt_modify         LINES gv_number_modify.
DESCRIBE TABLE gt_do_nothing     LINES gv_number_do_nothing.

ENDFORM.                      " CHECK DATA
```

Figure 4.79: Source code for checking the data before insertion or changing the database table

Figure 4.80 shows the code of the routine UPDATE\_TABLE. This routine processes the uploaded records depending on the check result.

```
/*&-----  
*&      Form  UPDATE_TABLE  
*&-----  
*      Update of DB-Table ZGENERALCONTRACT  
*-----  
*  --> GT_INSERT      Table for Insert ZGENERALCONTRACT  
*  --> GT MODIFY       Table for Modify ZGENERALCONTRACT  
*  --> GV_ANZAHL_INSERT Numbers of data sets to be insert  
*  --> GV_ANZAHL_MODIFY Numbers of data sets to be changed  
*-----  
FORM update_table.  
  
    gv_number_insert_error = 0.  
    gv_number_modify_error = 0.  
  
    * insert data set  
    IF gv_number_insert > 0.  
        LOOP AT gt_insert INTO gs_insert.  
            INSERT INTO zgeneralcontract VALUES gs_insert.  
            IF sy-subrc <> 0.  
                gv_number_insert_error = gv_number_insert_error + 1.  
            ENDIF.  
        ENDLOOP.  
    ENDIF.  
  
    * change data set  
    IF gv_number_modify > 0.  
        LOOP AT gt_modify INTO gs_modify.  
            MODIFY zgeneralcontract FROM gs_modify.  
  
            IF sy-subrc <> 0.  
                gv_number_modify_error = gv_number_modify_error + 1.  
            ENDIF.  
        ENDLOOP.  
    ENDIF.  
  
ENDFORM.          " UPDATE_TABLE
```

Figure 4.80: Source code for the insertion of new records or changing existing data sets

At the end of the program, some simple WRITE statements inform the user about the upload results (see Figure 4.81).

```

FORM output_result.

* data sets had been uploaded
IF gv_number_upload > 0.
  WRITE: / 'Output Program Result:'.
  WRITE: / 'User:',      (15) sy-uname.
  WRITE: / 'Programm:', (15) sy-repid.
  WRITE: / 'Date:',     (15) sy-datum.
  WRITE: / 'Time:',     (15) sy-uzeit.
  WRITE: / 'Number of uploaded data sets:', gv_number_upload.
  IF gv_number_insert > 0.
    WRITE: / 'Number of inserted data sets:', gv_number_insert.
  ENDIF.
  IF gv_number_modify > 0.
    WRITE: / 'Number of changed data sets:', gv_number_modify.
  ENDIF.
  IF gv_number_do_nothing > 0.
    WRITE: / 'Number of data sets which not has been inserted or changed:',
           gv_number_do_nothing.
  ELSEIF gv_number_do_nothing = 0.
    WRITE: / 'All uploaded data sets have been inserted or changed'.
  ENDIF.
  IF gv_number_insert_error > 0.
    WRITE: / 'Numbers of insert error', gv_number_insert_error.
  ENDIF.
  IF gv_number_modify_error > 0.
    WRITE: / 'Number of modify error', gv_number_modify_error.
  ENDIF.

* No data sets had been uploaded
ELSEIF gv_number_upload = 0.

  WRITE: / 'Output Program Result:'.
  WRITE: / 'User:',      (15) sy-uname.
  WRITE: / 'Programm:', (15) sy-repid.
  WRITE: / 'Date:',     (15) sy-datum.
  WRITE: / 'Time:',     (15) sy-uzeit.

  WRITE: / 'No data sets were uploaded.'.
  ULINE.
  WRITE: / '-----END OF PROGRAM-----'.

ENDIF.

ENDFORM.          " OUTPUT RESULT

```

Figure 4.81: Source code for user information

Even this program can be used as a template to upload more data sets to different database tables. To use this program for uploading to other database tables, you only have to change a few coding lines.

#### 4.13 ABAP code lines in tables

If you have to check customer developments for customer-specific programming standards, sometimes you have to check a row of programs. All code lines of ABAP programs are stored in the database table **REPOSRC**. The coding is in the field **DATA** and is stored in coded hexadecimal signs. To load the code lines of programs in an internal table of a program, you have to use the standard SAP function module **READ\_SOURCE**. Look at the sample program **ZCU\_ABAP\_SOURCECODE** in Figure 4.82.

```

CALL FUNCTION 'READ_SOURCE'
  EXPORTING
    scanned          = space
    content_save_flag = space
    pemode           = space
    edit_control     = space
    new_source       = space
    komprmode        = space
    overflow_area    = space
    msg               = space
    incl              = space
    row               = space
    col               = space
    wrd               = space
    change_mod_flag  = space
    transport_key    = gs_transport_key
  TABLES
    content          = gt_content
    content_c         = gt_content_c
    lineindex         = gt_lineindex
    mod_tab           = gt_mod_tab
    smodilog_abap   = gt_smodilog_abap
    page              = gt_page
    linenum           = gt_linenum
    step              = gt_step
    tk                = gt_tk
    stm               = gt_stm
  CHANGING
    status_flag       = gv_status_flag
    extend_mod        = gv_extend_mod
    edit              = gs_edit
  EXCEPTIONS
    cancelled         = 1
    not_found         = 2
    read_protected   = 3
    OTHERS             = 4.

  IF sy-subrc = 1.
    MESSAGE e001(zcu_isu) WITH 'Reading of program lines was aborted'.
  ELSEIF sy-subrc = 2.
    MESSAGE e001(zcu_isu) WITH 'Program lines of selected program could not be found.'  

                                pa_progn.
  ELSEIF sy-subrc = 3.
    MESSAGE e001(zcu_isu) WITH 'Program is read-protected.'.
  ELSEIF sy-subrc = 4.
    MESSAGE e001(zcu_isu) WITH 'Systemerror reading program lines.'.
  ENDIF.

```

Figure 4.82: Calling the standard SAP function module READ\_SOURCE

Important for the call of this function module is the correct type assignment of the parameters used (see Figure 4.83).

```

TYPE-POOLS: smodi.

DATA: gt_content          TYPE rswsourcet,
      gt_content_c       TYPE rswsourcet,
      gt_mod_tab         TYPE smodi_mod_tab,
      gt_page            TYPE rswsourcet,
      gt_tk              TYPE sedi_tk,
      gt_lineindex        TYPE TABLE OF edlineindx,
      gt_smmodilog_abap  TYPE TABLE OF smmodilog,
      gt_linenum          TYPE TABLE OF edlinenum,
      gt_step             TYPE TABLE OF edstep,
      gt_stm              TYPE TABLE OF sstmnt,
      gs_transport_key    LIKE trkey,
      gv_status_flag(1)   TYPE c,
      gv_extend_mod(1)    TYPE c,
      gs_edit             TYPE s38e,
      gv_output           TYPE string.

```

Figure 4.83: Declarations for using the standard SAP function module READ\_SOURCE

The single lines are stored in lines of the internal table GT\_CONTENT. The lines can be read with a LOOP statement such as:

```
LOOP AT gt_content INTO gv_content
```

Therefore, the variable “gv\_content” must have the type XSTRING.

## 4.14 Send e-mail from SAP

To send e-mail out of SAP, an e-mailing server for the SAP system has to be configured with transaction **SCOT** (*Business Communication Services – Administration*).

To use existing table contents to create an e-mail out of SAP, you have two options: add as text in the e-mail or attach as a file in the appendix of the e-mail.

For sending e-mails out of SAP, the standard SAP function module **SO\_NEW\_DOCUMENT\_SEND\_API1** has to be used. Before calling this function module, four elements of the e-mail have to be defined:

- ▶ The recipient of the e-mail
- ▶ The attributes of the e-mail
- ▶ The document type of the e-mail
- ▶ The content of the e-mail

To explain the e-mailing dispatch, I created the demo program **ZCU\_ABAP\_SEND\_MAIL**. This program sends the content of the table **ZGENERALCONTRACT** as text.

To create the e-mail, we need the declarations listed in Figure 4.84.

```
DATA: lt_receiver      TYPE TABLE OF somlreci1,
      ls_receiver      TYPE somlreci1,
      lt_mail          LIKE solistil OCCURS 5000 WITH HEADER LINE,
      ls_mail          TYPE solistil,
      ls_doc_data      TYPE sodocchgi1.
```

Figure 4.84: Declarations for sending e-mails

First, store the recipient (receiver) of the e-mail in the internal table “**lt\_empfaenger**” (see Figure 4.85).

```
REFRESH lt_receiver.
ls_receiver-rec_type  = 'U'.
ls_receiver-receiver  = 'thomas.stutenbaeumer@conuti.de'.
ls_receiver-express   = 'X'.
APPEND ls_receiver TO lt_receiver.
```

Figure 4.85: Creating the receiver list

Define the attributes of the e-mail (see Figure 4.86).

```
* Define features of sending document-----*
ls_doc_data-obj_name      = 'General Contracts'.
ls_doc_data-obj_langu      = sy-langu.
ls_doc_data-sensitivty    = 'P'.
ls_doc_data-obj_prio       = '5'.
ls_doc_data-no_change      = 'X'.
```

Figure 4.86: Definition of e-mail attributes

In the example program, the e-mail gets the name “General Contracts”. The e-mail sends in the current SAP system language. The sensitivity describes whether the e-mail can be read by other users and whether the e-mail can be passed to other recipients. There are the standard options:

- ▶ Sensitivity ‘O’ = normal
- ▶ Sensitivity ‘F’ = functional, the e-mail can be passed to other recipients
- ▶ Sensitivity ‘P’ = personal, the e-mail cannot be read by anyone but the recipient

The object priority describes the receiver priority. In this way the e-mail gets specific signals in the e-mail box of the recipient, which tags the e-mail as urgent. You can use priorities from 1 to 9. Number 1 has the highest priority and 9 the lowest.

If the sign no\_change = ‘X’ is used, only the author can change the content of the e-mail.

In the last step of the e-mail creation, the records of the table ZGENERALCONTACT are written as text to the e-mail content (see Figure 4.87).

```

REFRESH lt_mail.

CONCATENATE 'Mandant'
    'General Contract Number'
    'General Contract Title'
    'General Contract Keeper'
    'Date of Creation'
    'Created by'
    'Date of Change'
    'Change by'
    INTO ls_mail-line SEPARATED BY ','.
APPEND ls_mail TO lt_mail.

LOOP AT gt_output INTO gs_output.

CLEAR      ls_mail.

CONCATENATE  gs_output-mandt
    gs_output-general_contract_number
    gs_output-general_contract_title
    gs_output-general_contract_keeper
    gs_output-erdat
    gs_output-ernam
    gs_output-aedat
    gs_output-aenam
    INTO ls_mail-line SEPARATED BY ','.

APPEND ls_mail TO lt_mail.

ENDLOOP.

```

Figure 4.87: Write records to the content of the e-mail

If the necessary parameters are defined, the e-mail can be sent using the standard SAP function module.

```

CALL FUNCTION 'SO_NEW_DOCUMENT_SEND_API1'
EXPORTING
  document_data          = ls_doc_data
  document_type          = 'TXT'
TABLES
  object_content         = lt_mail
  receivers              = lt_receiver
EXCEPTIONS
  too_many_receivers     = 1
  document_not_sent      = 2
  document_type_not_exist = 3
  operation_no_authorization = 4
  parameter_error        = 5
  x_error                = 6
  enqueue_error          = 7
  OTHERS                 = 8.

```

Figure 4.88: Calling the standard SAP function module for sending e-mails

Remember that you have to develop an understandable error message if the function module causes an error (`sy-subrc <> 0`). Likewise, you need to inform the user about a successful e-mail dispatch.

If the function module runs with an error, the e-mail is not given to the output queue. The program RSCCONN01 starts the e-mail dispatch. This program has to be defined as a periodical job in the SAP system. If this job is not defined, you have to dispatch your e-mail manually with transaction **SCOT**.

Use the pull-down menu **ADMINISTRATION • OUTBOUND SEND REQUESTS** in transaction **SCOT** (see Figure 4.89).

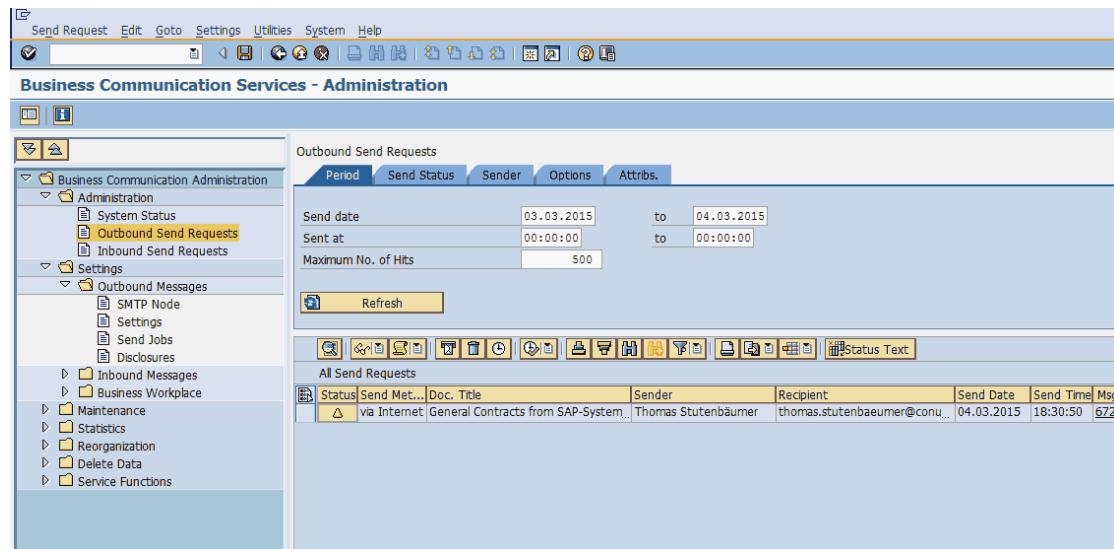


Figure 4.89: Outbound end requests in transaction SCOT

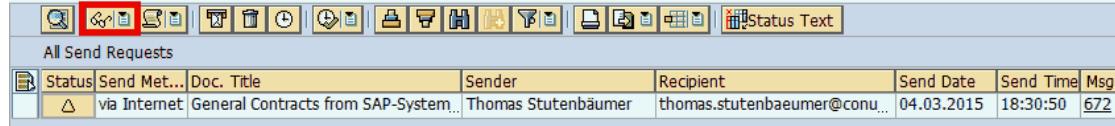
Mark the e-mail to be sent in the first row of the screen **ALL SEND REQUESTS**. Then choose the pull-down menu **SEND REQUEST** and then function **START SEND PROCESS FOR SELECTION**.

#### Be careful when sending e-mail manually



Pay attention that you don't accidentally send all e-mails from an SAP test system. There could be test e-mails of an electronic order system in the outbound send request queue.

With transaction **SCOT**, you can display the content of e-mails. To do this, mark an e-mail in the outbound send request. Then click the  icon to display the e-mail content (see Figure 4.91).

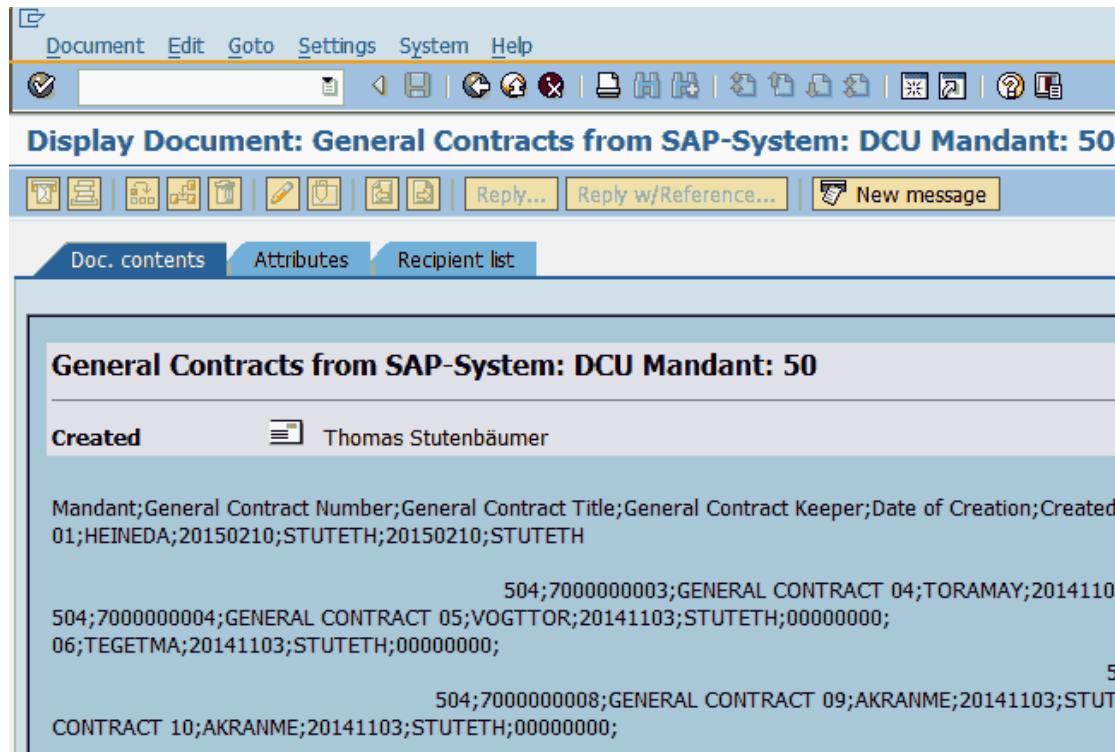


A screenshot of the SAP transaction SCOT interface. The top bar contains various icons and a status text field. Below is a table titled 'All Send Requests' with columns: Status, Send Met..., Doc. Title, Sender, Recipient, Send Date, Send Time, and Msg. One row is selected, showing 'via Internet' as the status, 'General Contracts from SAP-System...' as the document title, 'Thomas Stutenbäumer' as the sender, 'thomas.stutenbaeumer@conu...' as the recipient, '04.03.2015' as the send date, '18:30:50' as the send time, and '672' as the message ID.

Status	Send Met...	Doc. Title	Sender	Recipient	Send Date	Send Time	Msg
via Internet		General Contracts from SAP-System...	Thomas Stutenbäumer	thomas.stutenbaeumer@conu...	04.03.2015	18:30:50	672

Figure 4.90: A send request in the overview

By using the other tabs in this display, you can look at the e-mail **ATTRIBUTES** and **RECIPIENT LIST**.



A screenshot of the SAP transaction SCOT interface showing the details of an outbound e-mail request. The top navigation bar includes Document, Edit, Goto, Settings, System, and Help. The main title is 'Display Document: General Contracts from SAP-System: DCU Mandant: 50'. Below the title is a toolbar with various icons. The main content area shows the document title 'General Contracts from SAP-System: DCU Mandant: 50'. Under 'Created' it lists 'Thomas Stutenbäumer'. The document content is displayed in a large text area, showing several lines of text separated by semi-colons, such as 'Mandant;General Contract Number;General Contract Title;General Contract Keeper;Date of Creation;Created 01;HEINEDA;20150210;STUTETH;20150210;STUTETH' and '504;7000000003;GENERAL CONTRACT 04;TORAMAY;20141103;504;7000000004;GENERAL CONTRACT 05;VOGTTOR;20141103;STUTETH;00000000;06;TEGETMA;20141103;STUTETH;00000000;504;7000000008;GENERAL CONTRACT 09;AKRANME;20141103;STUTETH;CONTRACT 10;AKRANME;20141103;STUTETH;00000000;'.

Figure 4.91: Display details of an outbound e-mail request

With transaction **SCOT**, you can also create an e-mail with a free eligible **RECIPIENT** by clicking the  button (see Figure 4.92).

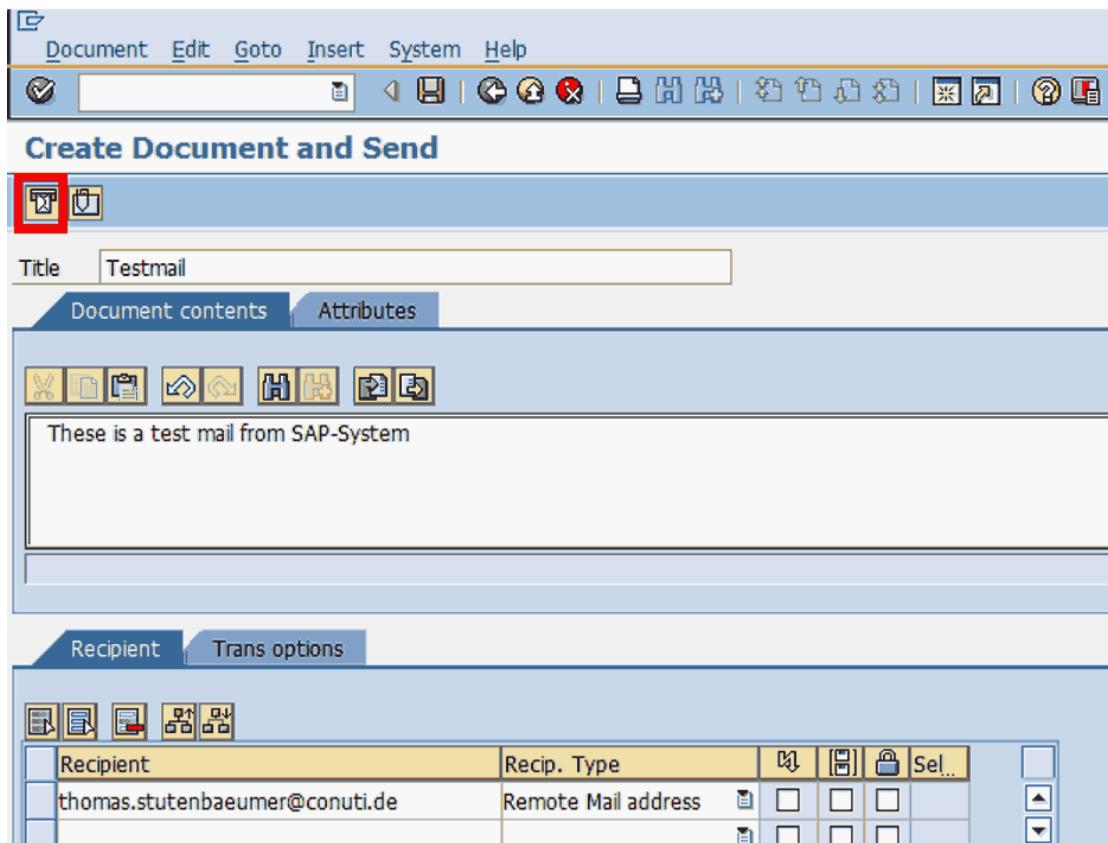


Figure 4.92: Create an e-mail by using transaction SCOT

With a click on the icon, the e-mail is sent.

The transaction can also be used to delete e-mails out of the outbound queue. Mark the e-mail that has to be deleted, and click the icon in the overview of the outbound request (see Figure 4.89).

If you want to delete the e-mail, use another instance of transaction **SCOT** (see Figure 4.93).

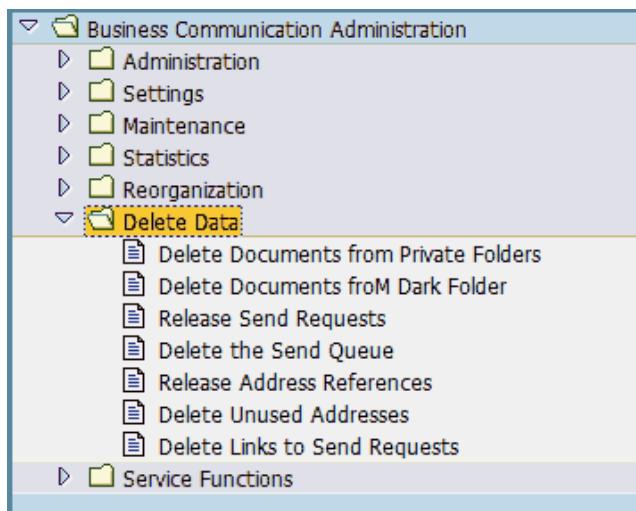


Figure 4.93: Delete data from business communication services

#### 4.15 “Dirty assign”

The “dirty assign” can be a great tool – or it can cause considerable damage. It depends on the form using this method. Use the dirty assign method only when you don’t have any other way to solve a problem.

The dirty assign, allows the developer to access the memory content of global variables of earlier programming objects in the ABAP stack. With this method, you have access to variables in user exits that are not normally available.

##### Function of “dirty assign”



In the following example, I use a user exit in front of storing an expiration date after creating the exit with the standard SAP transaction. You can make customer-specific checks for an expiration date in the user exit EXIT\_SAPLEC55\_012.

Figure 4.94 shows the source code to define the fixed expiration date “12-31” of the current year, ignoring the expiration date input by the user.

```

IF   x_moveout_ext-okcode = 'SAVE'
AND sy-tcode           = 'EC55E'. "Create move-out

* set move-out-date on 31.12. of current year
FIELD-SYMBOLS: <auszug>      TYPE isu06_moveout,
                <s_eausv>    TYPE eausv,
                <s_t_mo_d>    TYPE isu06_mo_eausvd,
                <s_t_mo_tcd>  TYPE isu06_mo_eausvtcd,
                <s_t_mo_kto>  TYPE isu06_mo_eausvkto.

DATA: lv_year(4)          TYPE c,
      lv_move_out_date(8) TYPE c.

ASSIGN ('(SAPLEC55)OBJ') TO <auszug>.
IF sy-subrc = 0.
  LOOP AT <auszug>-t_eausv ASSIGNING <s_eausv>.
    lv_year = <s_eausv>-auszdat(4).
    CONCATENATE lv_year
      '1231'
      INTO lv_move_out_date.
    <s_eausv>-auszdat = lv_move_out_date.
  ENDLOOP.
  LOOP AT <auszug>-t_mo_d ASSIGNING <s_t_mo_d>.
    lv_year = <s_eausv>-auszdat(4).
    CONCATENATE lv_year
      '1231'
      INTO lv_move_out_date.
    <s_t_mo_d>-auszdat = lv_move_out_date.
  ENDLOOP.
  LOOP AT <auszug>-t_mo_tcd ASSIGNING <s_t_mo_tcd>.
    lv_year = <s_eausv>-auszdat(4).
    CONCATENATE lv_year
      '1231'
      INTO lv_move_out_date.
    <s_t_mo_tcd>-auszdat = lv_move_out_date.
  ENDLOOP.
  LOOP AT <auszug>-t_mo_kto ASSIGNING <s_t_mo_kto>.
    lv_year = <s_eausv>-auszdat(4).
    CONCATENATE lv_year
      '1231'
      INTO lv_move_out_date.
    <s_t_mo_kto>-auszdat = lv_move_out_date.
  ENDLOOP.
ENDIF.
ENDIF.

```

Figure 4.94: User exit EXIT\_SAPLEC55\_012

The dirty assign method has the following notation:

FIELD-SYMBOLS: <field\_symbol>

TYPE (Type of global\_variable).

ASSIGN ('(Program\_name)global\_variable')

TO <field\_symbol>.

```

IF sy-subrc = 0.
  <field_symbol>-(field_name) = .....
ENDIF.

```

### Ask SY-SUBRAC directly after ASSIGN command



You always have to ask for the SY-SUBRC after using the ASSIGN command. If an ASSIGN command was not successful and you use the assigned field symbol in the following code lines, the program causes a hard crash (dump).

With the help of the dirty assign method, you can access the global structure “OBJ” of the previous called program SAPLEC55. The **ABAP AND SCREEN STACK** of the debugger displays the function module ISU\_MOVEOUT\_CREATE at line 11 (see Figure 4.95).

ABAP and Screen Stack								
Sta...	Stac...	S...	Event Type	Event	Program	Navi...	Include	Line
⇒	18		FUNCTION	EXIT_SAPLEC55_012	SAPLXeca		ZXECAU01	71
	17		FORM	CF012_CUSTOMER_INPUT	SAPLEC55		LEC55F50	32
	16		FUNCTION	ISU_O_MOVE_OUT_INPUT	SAPLEC55		LEC55U02	86
	15		MODULE (PAI)	INPUT_CHECK	SAPLEC55		LEC55I10	362
	14		PAI MODULE	INPUT_CHECK				29
	13		PAI SCREEN	0200	SAPLEC55			29
	12		FUNCTION	ISU_O_MOVE_OUT_DIALOG	SAPLEC55		LEC55U14	35
	11		FUNCTION	ISU_S_MOVE_OUT_CREATE	SAPLEC55		LEC55U08	130
	10		FUNCTION	ISU_REMOVAL_MOVEOUT_CREATE	SAPLEC60		LEC60U01	42
	9		FORM	MOVE_OUT_CREATE	SAPLEC60		LEC60F01	897
	8		MODULE (PAI)	PAI_100_ACTIONS	SAPLEC60		LEC60I01	167
	7		PAI MODULE	PAI_100_ACTIONS				25
	6		PAI SCREEN	0100	SAPLEC60			25
	5		FUNCTION	ISU_REMOVAL_FO_CONTROL	SAPLEC60		LEC60U11	113
	4		EVENT	START-OF-SELECTION	EMOVEINOUT		EMOVEINOUT	27
	3		PAI MODULE	SYST-ABRUN				6
	2		PAI SCREEN	1000	SAPMSSY0			6
	1		TRANSACTION	EC55E(EC55E)				0

Figure 4.95: ABAP stack by a move out

When you double click the name of the function module, the debugger goes to the place in the function module where the program actually stands. In the tab for global variables (**Globals**), you can see the complex structure of OBJ in the first few rows (see Figure 4.96).

The screenshot shows the ABAP Debugger interface. On the left, there is a code editor window displaying ABAP code. On the right, there is a 'Global Variables' table. The 'Globals' tab is selected in the top navigation bar. A red box highlights the 'Globals' tab. The table has columns: Va., St., Variable Name, Va., Val., and Technical Type. The 'Variable Name' column lists various global variables, and the 'Technical Type' column indicates their structure.

Va...	St...	Variable Name	Va...	Val...	Technical Type
		OBJ		Structure: deep	Structure: deep(14264)
		OBJTT		Structure: deep	Structure: deep(32)
		MOLOG	{0:288*\CLASS=CL_EZLOG}		Ref to CL_EZLOG
		OBJA		Structure: deep	Structure: deep(1200)
		OBJP		Structure: deep	Structure: deep(12904)
		T_OBJC	[1x11(9376)] Standard Tab.	Standard Table[1x11(93...	
		T_OBJC_WA		Structure: deep	Structure: Table[9376]
		T_OBJI	[1x10(3424)] Standard Tab.	Standard Table[1x10(34...	
		T_OBJI_WA		Structure: deep	Structure: deep(3424)
		OBJS		Structure: deep	Structure: deep(6080)
		GT_BCONTS_TO_CRM	[0x17(2608)] Standard Tab.	Standard Table[0x17(26...	
		T_EAUSV_WA	003000109	Structure: flat, charlike(1...	
		BCONT_WA			Structure: flat, charlike(1...

Figure 4.96: Global variables in a previous processing block

With the dirty assign method, the developer has access to the memory content of the OBJ structure in the downstream user exit. If the dirty assign has been successful, the expiration dates are changed in the tables.

### Only use global variables in the dirty assign method



Using the dirty assign method gives you access to the global variables of a previous processing block. It is not possible to use local variables.

“Dirty” refers to the fact that the memory content of a structure can be changed, even though you normally have no access to the variable.

### Using the dirty assign method



Use this method only if you have no other options to solve a problem, e.g., “Import to Memory” and “Export from Memory.” It is useful to document the application of the dirty assign in a central location.

## 4.16 Wait for update

In the example program to change the meter reading unit of installations ZCU\_FIRST\_CUSTOMER\_MR\_UNIT, records have to be deleted before inserting new data sets with the same key fields.

Figure 4.97 shows time slices of corresponding records for an installation where the assigned **METER READING UNIT** J01-002 is valid. The key fields of this table include the **INSTALLATION** number and the **VALID TO** date.

The screenshot shows the SAP Table Entry interface with the title bar "Table Entry Edit Goto System Help". Below the title bar is a toolbar with various icons. The main area displays the title "EANLH: Display of Entries Found". A sub-header "Install.Time Slice" is visible. There are input fields for "Table to be searched" (EANLH), "Number of hits" (4), "Runtime" (0), and "Maximum no. of hits" (500). Below these is a toolbar with icons for search, print, and details. The main data area is a table with the following data:

Installation	Valid to	Valid from	Rate category	Billing class	Meter Reading Unit	Temperature area
60001567	31.12.2005	01.01.2002	EJPP	PK	J01-002	0001
60001567	31.12.2006	01.01.2006	EJPP	PK	J01-002K	0001
60001567	31.12.2009	01.01.2007	EJPP	PK	J01-002	0001
60001567	31.12.9999	01.01.2010	EJPG	PK	J01-002	0001

Figure 4.97: Time slices in the database table EANLH

From 01-01-2015 a new meter reading unit shall be used with date-to 12-31-9999. Therefore, the existing time slice has to be separated; the last existing installation time slice must get the to-date 12-31-2014. After this, a new time slice with valid from-date 01.01.2015 and valid to-date 31.12.9999 as well as the new meter reading unit has to be inserted.

The **VALID TO** date is a key field. The existing record has to be deleted and two new data sets have to be created: One time slice for the validity period of the “old” meter reading unit and one time slice for the validity period of the new meter reading unit (see Figure 4.98).

Installation	Valid to	Valid from	Rate category	Billing class	MR Unit	Temperature area
60001567	31.12.2005	01.01.2002	EJPP	PK	J01-002	0001
60001567	31.12.2006	01.01.2006	EJPP	PK	J01-002K	0001
60001567	31.12.2009	01.01.2007	EJPP	PK	J01-002	0001
60001567	31.12.2014	01.01.2010	EJPG	PK	J01-002	0001
60001567	31.12.9999	01.01.2015	EJPG	PK	J01-007	0001

Figure 4.98: Two new time slices in the database table EANLH

First, delete the “old” existing time slice. Then insert a new time slice with the key field valid-to 12-31-9999.

The source code for this adjustment of the time slices looks like that in Listing 4.6:

```
DELETE FROM eanlh
WHERE anlage = gs_eanlh_old-installation
AND bis = gs_eanlh_old-to
AND ab = gs_eanlh_old-from.
```

Listing 4.6: Deletion of time slices

After the deletion of the time slice with the “old” meter reading unit and the valid-to 12-31-9999 (see Listing 4.6), we have to insert both new time slices: One for the validity period of the “old” meter reading unit and one for the validity period of the “new” meter reading unit (see Listing 4.7).

```
INSERT INTO eanlh VALUES gs_eanlh_old.
INSERT INTO eanlh VALUES gs_eanlh_new.
```

Listing 4.7: Insertion of time slices

If the DELETE and INSERT commands are processed directly one after the other, the database error “DUPLICATE KEY” will appear. The old record has been deleted in the main memory, but the deletion has not yet been stored in the database. Also, when using the COMMIT WORK command between the DELETE and INSERT commands, make sure to avoid incorrect processing. The COMMIT WORK command means that changes in the database tables are committed, but this

command does not recognize the end of the change with the *update task* to the database.

Only the standard SAP function C14\_WAIT\_ON\_COMMIT after the COMMIT WORK command waits until the DELETE command is correct and finally processed in the database. The update task sends a signal to the function module C14\_WAIT\_ON\_COMMIT after finally deleting the record of the database. Only then is the calling program processed with the next command.

## 4.17 Write change documents

If sensitive data is changed in the SAP system, such as data of the contract account or bank details of customers, the changes are stored in *change documents*. The change documents option permits changes to the corresponding object. It also shows which user made the changes.

The program ZCU\_FIRST\_CUSTOMER\_ACCOUNT\_CLA for the adjustment of the account classes of contract accounts is a classic example of where change documents should be stored.

To write change documents for a certain SAP object, the developer has to find out the name of this SAP internal change object (in our case, the contract account). You can get the necessary information by using transaction **SCDO** (change documents objects – overview) (see Figure 4.99).

Object	Text
MKK_VKONT	FI-CA: Change Documents for the Contract Account
MKK_VT	FI-CA: Change Documents for Provider Contract
MKK_VT_PRD	FI-CA: Change Documents for Product
MM_SERVICE	Change Document: External Services Management
MPLAN	Maintenance Plans

Figure 4.99: Change document objects

For the contract account, the **CHANGE DOCUMENT OBJECT** MKK\_VKONT is used. If a searched object is not available for the object contract account, you have to create an object by using the **TRANSACTION SCDO**.

With the pull-down menu **CHANGE DOC. OBJECT • DISPLAY**, the attributes of this object are displayed.

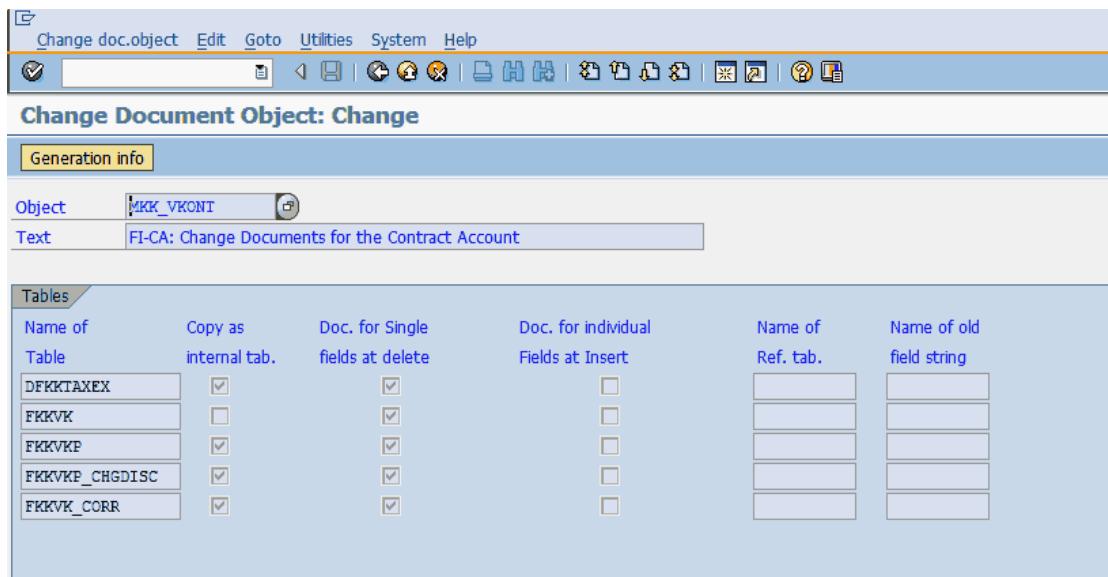


Figure 4.100: Attributes of a change document object

Figure 4.100 shows that for the change document object MKK\_VKONT, five tables are available. Changes to these tables are transferred from the update task as an internal table. Also, the deletion of data sets keeps data in the records, but the insertion of a data set does not.

To write change documents in ABAP programs, there are three steps:

1. Open the change document by using the standard SAP function module CHANGEDOCUMENT\_OPEN
2. Write the change document by changing the content of one field by using the function module CHANGEDOCUMENT\_SINGLE\_CASE and by changing the content of more than one field using the function module CHANGEDOCUMENT\_MULTIPLE\_CASE
3. Close the change document with the function module CHANGEDOCUMENT\_CLOSE

To change the account class, use the function modules as shown in Figure 4.101, Figure 4.102, and Figure 4.103.

```

1. Open Change Document
ls_cdhdr-objectclas = 'MKK_VKONT'.
ls_cdhdr-objectid   = gs_output-vkont.
ls_cdhdr-tcode       = 'ZISU_KONTOKLASSE'.

CALL FUNCTION 'CHANGEDOCUMENT_OPEN'
  EXPORTING
    objectclass      = ls_cdhdr-objectclas
    objectid        = ls_cdhdr-objectid
  EXCEPTIONS
    sequence_invalid = 1
    OTHERS           = 2.
  IF sy-subrc <> 0.
    gs_output_switch-note = 'Error: Change document could not be saved.'.
    MODIFY gt_output_switch FROM gs_output_switch INDEX sy-tabix
                                TRANSPORTING note.
    CONTINUE.
ENDIF.
```

Figure 4.101: Function module to open a change document

```

2. Write Change Document
ls_cdpos-tabname     = 'FKKVKP'.

CALL FUNCTION 'CHANGEDOCUMENT_SINGLE_CASE'
  EXPORTING
    change_indicator    = 'U'
    docu_delete         = ''
    docu_insert          = ''
    refarea_new         = ls_fkkvkp_new
    refarea_old          = ls_fkkvkp_old
    reftablename        = ls_cdpos-tabname
    tablename            = ls_cdpos-tabname
    workarea_new        = ls_fkkvkp_new
    workarea_old         = ls_fkkvkp_old
  EXCEPTIONS
    nametab_error       = 1
    open_missing         = 2
    position_insert_failed = 3
    OTHERS               = 4.

  IF sy-subrc <> 0.
    gs_output_switch-note = 'Error: Change document could not be saved.'.
    MODIFY gt_output_swtrch FROM gs_output_switch TRANSPORTING note.
    CONTINUE.
ENDIF.
```

Figure 4.102: Function module to write a change document

```

3. Close Change Document
CALL FUNCTION 'CHANGEDOCUMENT_CLOSE'
  EXPORTING
    date_of_change      = sy-datum
    objectclass         = ls_cdhdr-objectclas
    objectid            = ls_cdhdr-objectid
    tcode               = ls_cdhdr-tcode
    time_of_change     = sy-uzeit
    username            = sy-uname
  IMPORTING
    changenumber        = lv_changenr
  EXCEPTIONS
    header_insert_failed = 1
    no_position_inserted = 2
    object_invalid       = 3
    open_missing          = 4
    position_insert_failed = 5
    OTHERS                = 6.
  IF sy-subrc <> 0.
    gs_output_switch-note = 'Error: Change document could not be saved.'.
    MODIFY gt_output_switch FROM gs_output_switch TRANSPORTING note.
    CONTINUE.
  ENDIF.

ENDIF.

Wait on Commit.
COMMIT WORK.
CALL FUNCTION 'C142_WAIT_ON_COMMIT'.

```

Figure 4.103: Function module to close a change document

If a contract account has been changed by the program ZCU\_FIRST\_CUSTOMER\_ACCOUNT\_CLAS, a change document is stored.

In the following example, the contract class of the **CONTRACT ACCT** “20032187” has been changed to the **ACCOUNT CLASS** “0007” (First Customer). The contract account can be displayed by using transaction **CAA2** (see Figure 4.104).

Contract Account Edit Goto Extras Environment System Help

Contract Account Change: General data

Contract Acct: 20032187 Cont. Acct Cat.: 01 IS-U Contract Acc.

Partner/Address: 10008302 Schweizer Firma Postfach 374473 / 79934 Unte... Fix

Valid From: 05.03.2015

General data Payments/Taxes Dunning/Correspondence Title Missing

**Account management data**

Cont. acct name	Mittelspannungsvertrag
Acct in legacy	2165,0100,0910,4
Trading Partner	<input type="checkbox"/> Deletion Flag
Acct.Relation.	Account holder
Clerk Respons.	
AuthorizGroup	Posting Lock <input type="checkbox"/>
Restriction	<input type="checkbox"/>
Tolerance group	Tolerance group 0001
Clearing Cat.	0001
Planning group	CM Extra Days <input type="checkbox"/>
Interest key	Interest lock <input type="checkbox"/>
Tech. reg. strc	
Man. OS invoic.	No. outsortings <input type="checkbox"/>
Business Area	Commercial RSG <input type="checkbox"/>
Trans. currency	Inv.outs.chk gr SVK
Debtor BB req.	YAP participatn <input type="checkbox"/>
Payment Terms	BB Procedure 0
Act determ. ID	CP BB request <input type="checkbox"/>
Terms of payment 0001 Account class 0007	
Residential customers	

Figure 4.104: Contract account with a changed contract class

If you use the pull-down menu **EXTRAS • ACCOUNT CHANGES** a sub-screen is displayed that shows all recorded changes to this contract account (see Figure 4.105).

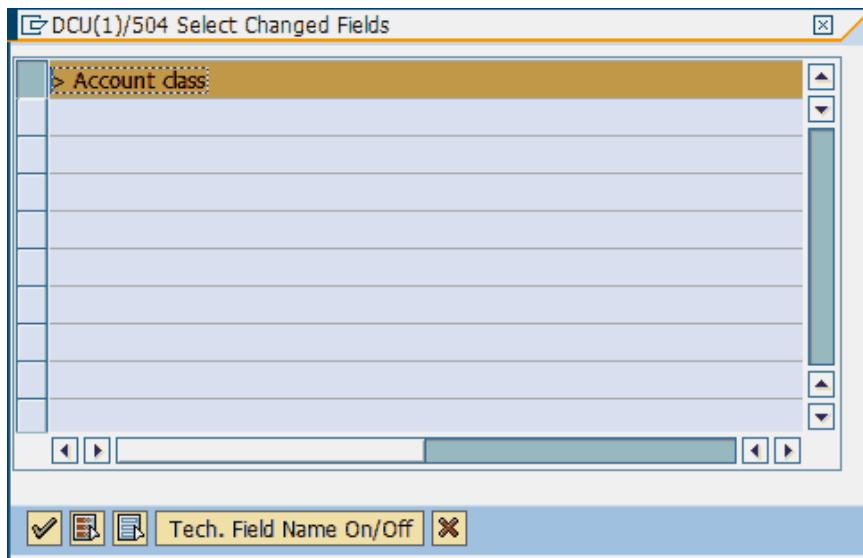


Figure 4.105: Overview of changes made to a contract account

In this case, only the contract class has been changed. If you want to see details of this change, mark this data set in the first column and use the icon.

The following screen shows the recorded details (see Figure 4.106). You can find out the **DATE** and **TIME** of the change, the number of the contract account in the **OBJECT VALUE** field, the user name in the **CHGD BY** field, and the **OLD VALUE** and **NEW VALUE**.

Date	Time	Object value	Chgd By	Fld Name	Tech.Field Name	New value	Old value	Additional Key/s
22.11.2014	09:31:49	000020032187	STUTETH	Account class	FKKVKP-KTOKL	0007	SVK	0010008302

Figure 4.106: Details of a change document for a contract account

The changed documents records are stored in the database table CDHDR (for header information of a change document, see Figure 4.107) and CDPOS (detailed information about the change, see Figure 4.108).

You can find the table key number of the change document by using the object and full contract account number (leading zeros) in **TRANSACTION** SE16N with the table CDHDR.

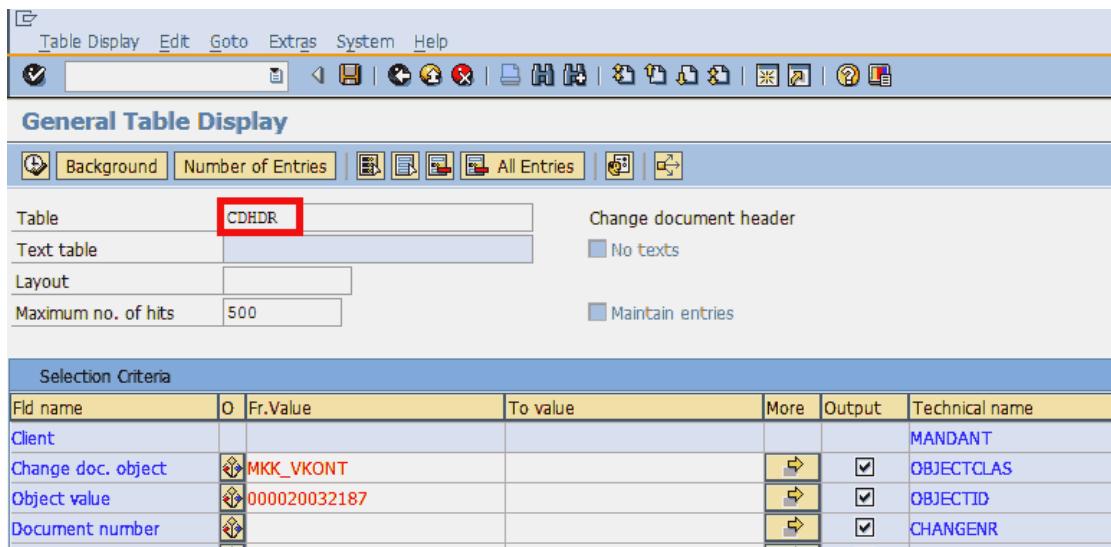


Figure 4.107: Select header information from change documents from the database table CDHDR

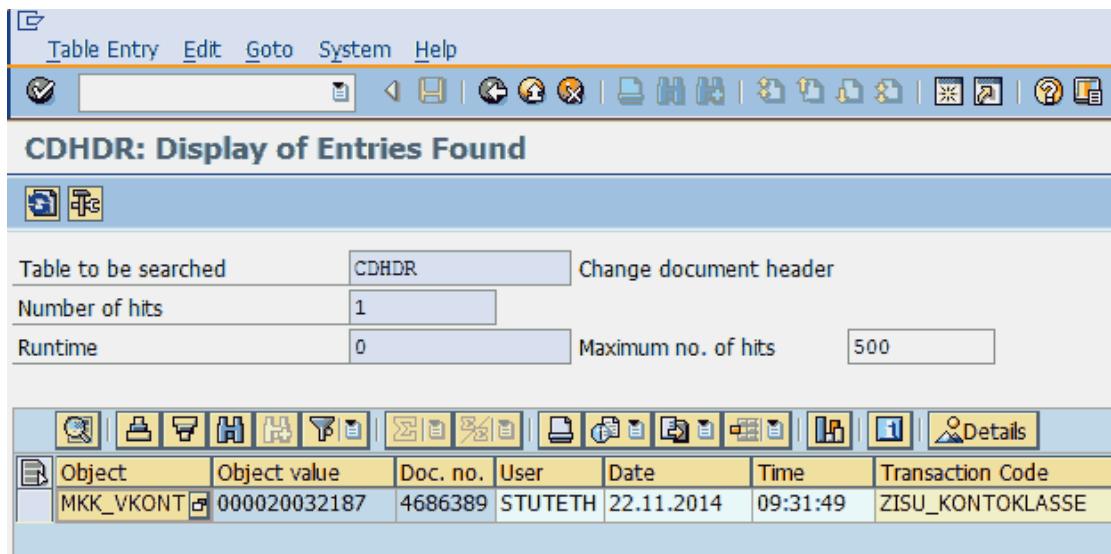


Figure 4.108: Header information of change documents in table CDHDR

Use the displayed table key number to find the change position within database table CDPOS (see Figure 4.109).

The screenshot shows the SAP GUI interface for displaying entries found in table CDPOS. The title bar reads "CDPOS: Display of Entries Found". The toolbar includes standard SAP icons for Table Entry, Edit, Goto, System, and Help. Below the toolbar, there are input fields for "Table to be searched" (CDPOS), "Number of hits" (1), "Runtime" (0), and "Maximum no. of hits" (500). A toolbar below these fields contains various icons for search, sort, and document management. The main area displays a table with one row of data:

Object	Object value	Doc. no.	Table	Table Key	Field	Change ID	Text flag	Unit	Unit	CUKY	CUKY	New value	Old value
MKK_VKONT	000020032187	4686389	FKVKP	5040000200321870010008302	KTOKL	U	1			0007	SVK		

Figure 4.109: Position information of change documents in table CDPOS

## 4.18 Dynamic SELECT statement

If you don't know a table name for a data selection as you write the program but need it at program runtime, you have to use dynamic programming statements. With dynamical statements, you can call function modules, where names are known only at the program runtime (e.g., after selection of the name from a customer table). These techniques are used by calling client-specific user exits (The method is described in the *Practical Guide to SAP ABAP: Part 2*).

Dynamic programming techniques are prohibited for security reasons in some companies.

### Dynamic SELECT statements



As an example, use the program ZCU\_ABAP\_DOWNLOAD\_TABLEDATA. In the selection screen, the user has to input the name of a database table. The program checks whether this table exists in the Data Dictionary. Then the program outputs the records from the selected table to a selectable file into an application server directory.

This program can be used to download the records of any SAP database tables.

In the selection screen, the user has to enter the name of the database table and the name of the downloaded file (see Figure 4.110).

```

*-----*
* Selection-Screen
*-----*

SELECTION-SCREEN BEGIN OF BLOCK block1 WITH FRAME TITLE text-001.
SELECTION-SCREEN SKIP.
PARAMETERS: pa_tabn TYPE tabname OBLIGATORY.
SELECTION-SCREEN SKIP.
PARAMETERS: p_file      TYPE cfdlfile OBLIGATORY.
PARAMETERS: p_verzap   TYPE cfdlfile
             DEFAULT '\\R3DCUCI\SAPMNT\TRANS\TMP\'.
SELECTION-SCREEN SKIP.

PARAMETERS: p_test AS CHECKBOX DEFAULT 'X'.
SELECTION-SCREEN END OF BLOCK block1.

*-----*
* AT LINE-SELECTION
*-----*

AT LINE-SELECTION.

READ LINE pa_tabn.

READ CURRENT LINE LINE VALUE INTO gv_tabname.

```

Figure 4.110: Selection screen and transfer of the database table name to the program

The program checks whether the selected table exists in the SAP system. If the selected table is not known in the system, the program ends with a corresponding error message. If the table is known in the system, the program assigns a dynamic type for the selected database table to the internal table <GT\_TABELLE> as well as to the internal structure <GS\_TABELLE> (see Figure 4.111).

```

□ *-----*
| * START-OF-SELECTION *
|-----*
START-OF-SELECTION.

gv_tabname = pa_tabn.

SELECT SINGLE *
  FROM dd03m
  INTO gs_dd03m
  WHERE tabname = gv_tabname.

□ IF sy-subrc <> 0.
  MESSAGE i005(zcu) WITH gv_tabname.
  LEAVE PROGRAM.
ENDIF.

* Dynamical type assign to internal table
CREATE DATA dref TYPE TABLE OF (gv_tabname).
ASSIGN dref->* TO <gt_tabelle>.

CREATE DATA dref TYPE (gv_tabname).
ASSIGN dref->* TO <gs_tabelle>.

```

Figure 4.111: Dynamic assign of a table type

After a security message asks if the user wants to print out all records of the table, the dynamic SELECT statement is processed (see Figure 4.112).

```

SELECT *
  FROM (gv_tabname)
  INTO TABLE <gt_tabelle>.

IF sy-subrc <> 0.
  CONCATENATE 'Table'
    pa_tabn
    'has not records.'
  INTO gv_message.
  MESSAGE i011(zcu) WITH gv_message.
  LEAVE PROGRAM.
ENDIF.

```

Figure 4.112: Dynamic selection of records from an SAP database table

As described in [Section 4.12.1](#) (data download), the selected records will download to a file in a directory of the application server. If the download has finished, you can display all records of the database table in the file by using **TRANSACTION AL11**.

#### 4.19 Number ranges and transaction SNRO

If new records shall be created, which ignore SAP internal sequential numbering, the numbering has to be addressed.

##### SAP internal number ranges



Consider the case of creating a new utility installation. You start by checking whether a number range object is available for installations. The standard SAP programs do this automatically. If you want to see the current number of the last installation, you can do this by using **TRANSACTION SNRO**.

Use the “F4” help key to display all object names in this transaction. Remove the restriction of 500 records in the filter. Then search by clicking on the  icon for the term “Installation” (see Figure 4.113).

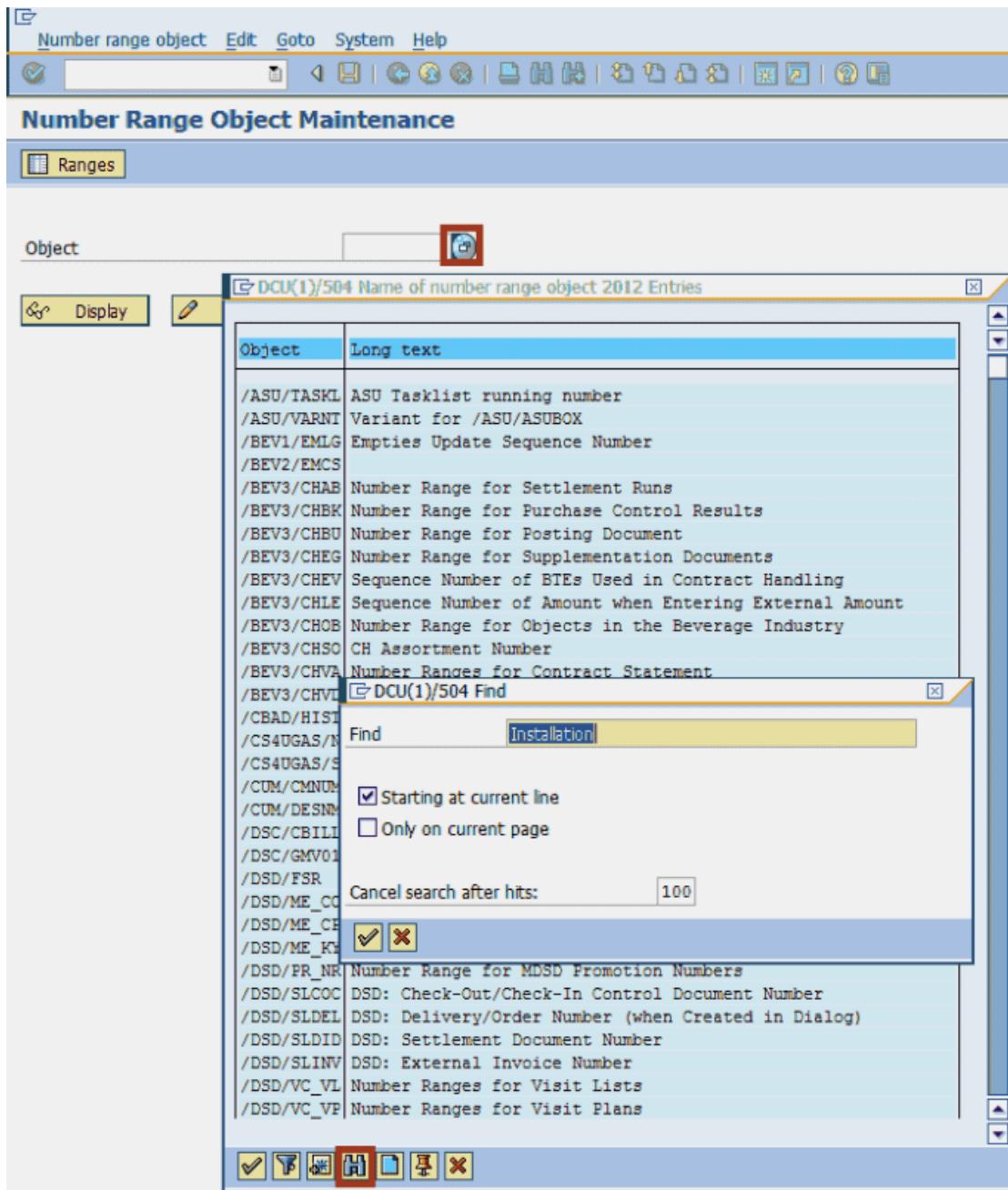


Figure 4.113: Find number range objects in transaction SNRO

Figure 4.114 shows the corresponding hit list of the search help.

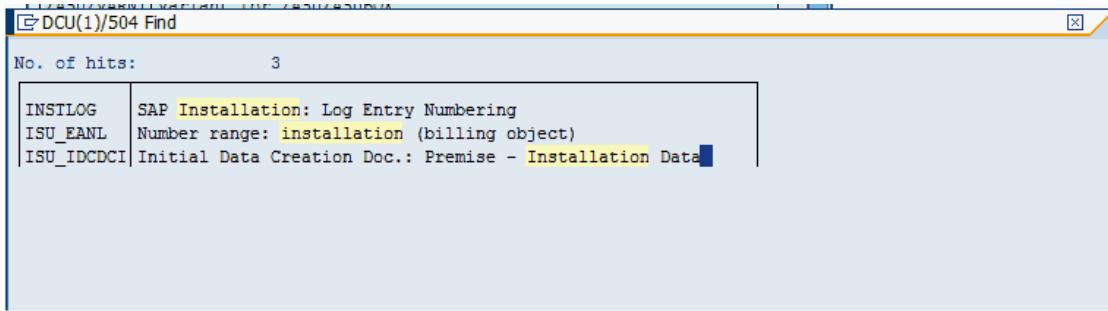


Figure 4.114: Hit list of the “F4” help for number range objects

For more options, use the hit list (**OBJECT**) named ISU\_EANL (see Figure 4.115).

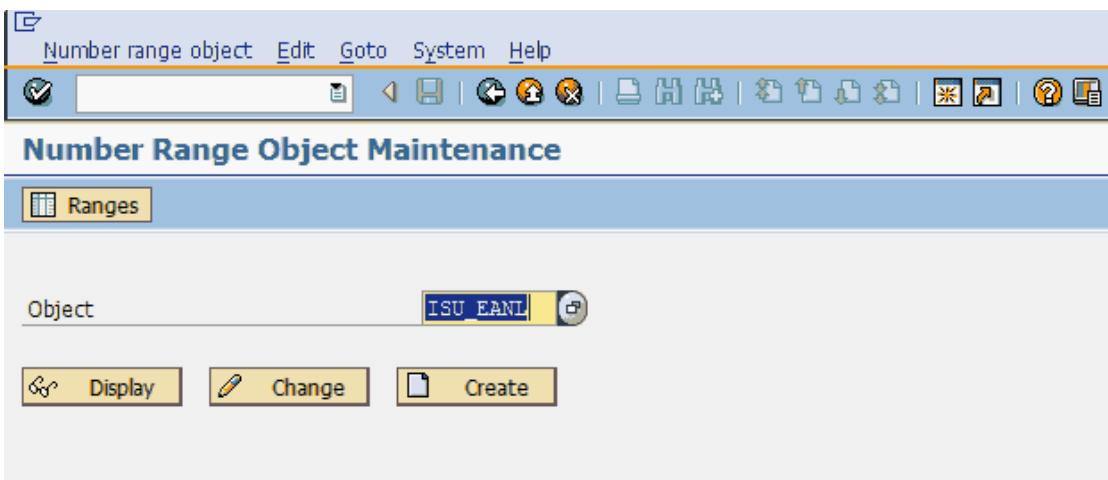


Figure 4.115: Select the number range object ISU\_EANL

By clicking the **Display** button, the attributes of this number range object are displayed (see Figure 4.116).

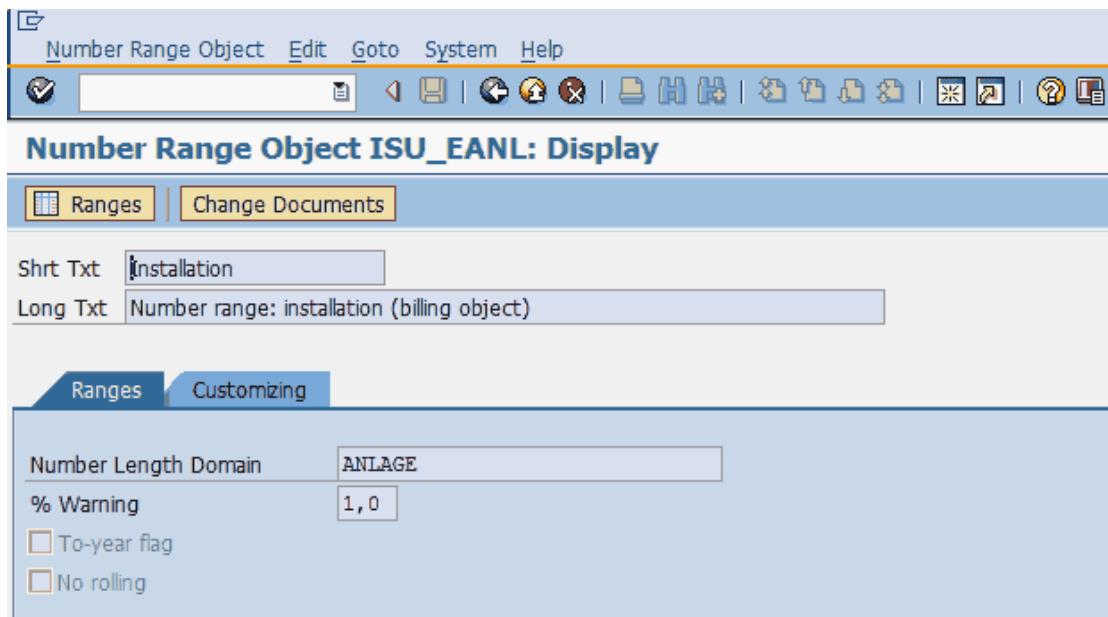


Figure 4.116: Attributes of a number range object

By clicking the **Ranges** button, the number range and the current (**NUMBER RANGE**) **NR STATUS** are displayed (see Figure 4.117).

Maintain Intervals: Installation				
No	From No.	To Number	NR Status	Ext
b1	0060000000	0069999999	60094774	<input type="checkbox"/>

Figure 4.117: Number interval of a number range object

The number of a utility installation in this system has a number between 0060000000 and 0069999999. The last created installation has the number 60094773. Because no check mark is entered in the **EXT** column, the installation number has to be created with the help of the number range.

To reserve a new installation number, the standard SAP function module **NUMBER\_GET\_NEXT** is used (see Figure 4.118). This function module can select one or more numbers out of a named number range. To explain the function of this function module, I've developed a program named **ZCU\_ABAP\_NUMBERRANGE\_INSTALL**.

```

*-----*
*      Form  GET_INSTALLATION_NUMBER
*-----*
*          Get a new installation number out of number range ISU_EANL
*-----*
* -->  gv_norange_object      Number range object
* -->  gv_norange_number      Number range status
* -->  pa_number              Number of Installation number to reserve
* <->  gv_installation_number New Installationnumber of number range
*                                ISU_EANL
*                                Number of reserved installation numbers
*-----*
FORM get_installation_number .

CALL FUNCTION 'NUMBER_GET_NEXT'
  EXPORTING
    nr_range_nr      = gv_norange_number      " = 01
    object           = gv_norange_object       " = ISU_EANL
    quantity         = pa numb                " Default '1'
  IMPORTING
    number           = gv_installation_number " TYPE anlage
    quantity         = gv_number
  EXCEPTIONS
    interval_not_found = 1
    number_range_not_intern = 2
    object_not_found = 3
    quantity_is_0     = 4
    quantity_is_not_1 = 5
    interval_overflow = 6
    buffer_overflow   = 7
    OTHERS            = 8.

IF sy-subrc <> 0.
  MESSAGE e003(zcu_isu) WITH 'Error: failed to get a new installation number'
                           'Error-Code:' sy-subrc.
ENDIF.

ENDFORM.                      " GET_INSTALLATION_NUMBER

```

Figure 4.118: Call the function module NUMBER\_GET\_NEXT

With this coding, you can create a defined number of installation numbers. The reserved number is given in the parameter PA\_NUMB. The return parameter GV\_NUMBER contains the amount of reserved numbers.

If more than one number was reserved, the last reserved number will be returned through the parameter GV\_INSTALLATION\_NUMBER.

The reserved numbers lie in the interval:

[GV\_INSTALLATION\_NUMBER – GV\_NUMBER+1;  
 GV\_INSTALLATION\_NUMBER].

With each use of the function module, the number range is increased. If no holes are created in the number range, e.g., a reserved number was not used because of an

error, the number range must be re-entered. You can use the standard SAP function module NUMBER\_RANGE\_OBJECT\_UPDATE. The number level of a number range object is stored in the field NRLEVEL of the database table NRIV.

## 4.20 Call transaction method and transaction SM35

In some cases, you have to use standard SAP transactions to delete or insert records. This is necessary if, for instance, data sets can be deleted only if no following action has been processed. In other words, a new record can be inserted only if the customized data is correct.

(Note that many standard SAP transactions contain many checks, and recreating these checks in each program is expensive.)

So, for these requests, use the SAP transaction in your own program. In our example, we have such a situation for the deletion of old and the creation of new billing plans. Payment plans may be deleted only if no bookings have been done. New billing plans may only be created if the new meter reading unit is correctly customized and is assigned to a *portion*. In addition, the portion must have appointment sentences.

A *portion* is a summary of several meter readings. To create billing plans, you have to use **TRANSACTION EA61**; for the deletion of billing plans, use **TRANSACTION E61D**.

I will explain both requests (creation and deletion) by using the example program **ZCU\_FIRST\_CUSTOMER\_BILLINGPLAN**.

To develop such a program, you have to do some preparation work:

- ▶ Record a transaction
- ▶ Generate a program out of this transaction record
- ▶ Transfer commands of the transaction record to your program

### 4.20.1 Recording a transaction

We start with recording **TRANSACTION EA61** to create a billing plan.

For a transaction recording, use **TRANSACTION SM35** (Batch Input: Session Overview) (see Figure 4.119).

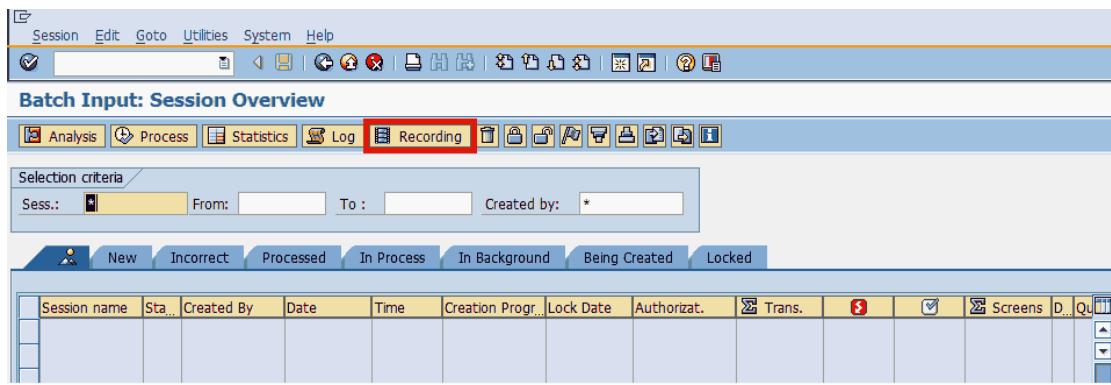


Figure 4.119: Transaction SM35 – Batch input session overview

Click the **Recording** button to change to the menu for recording SAP transactions. Before recording, you have to find an example. This example must be processed by the transaction, which has to be recorded without any errors.

Generally, you will not find a running example in the development system. Therefore, you have to record the transaction in the corresponding test system.

Start the record of **TRANSACTION EA61** by clicking the **New recording** button (see Figure 4.120).

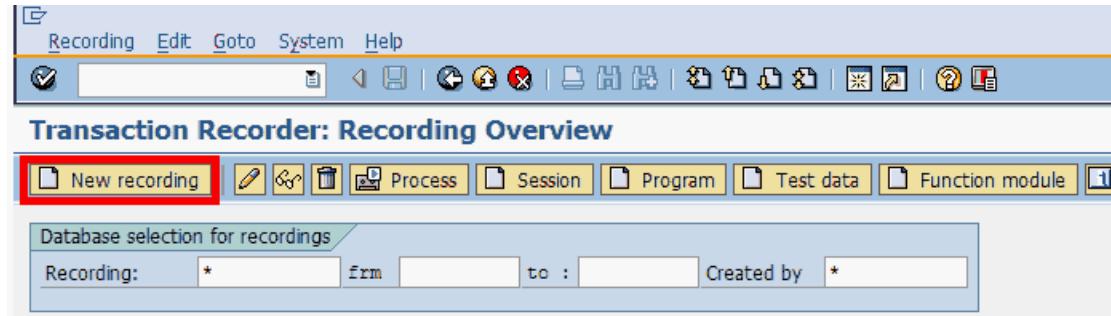


Figure 4.120: Create a new transaction recording

The transaction recording tapes all movements in the transaction, so it is useful to test the accuracy of the example before recording.

The new recording starts with the selection screen of **TRANSACTION EA61** (see Figure 4.121). The displayed **UPDATE MODE** and **CATT MODE** can be used consistently. The recording should be processed in **DEFAULT SIZE**. After the input of these parameters, start the recording by clicking the **Start recording** button.

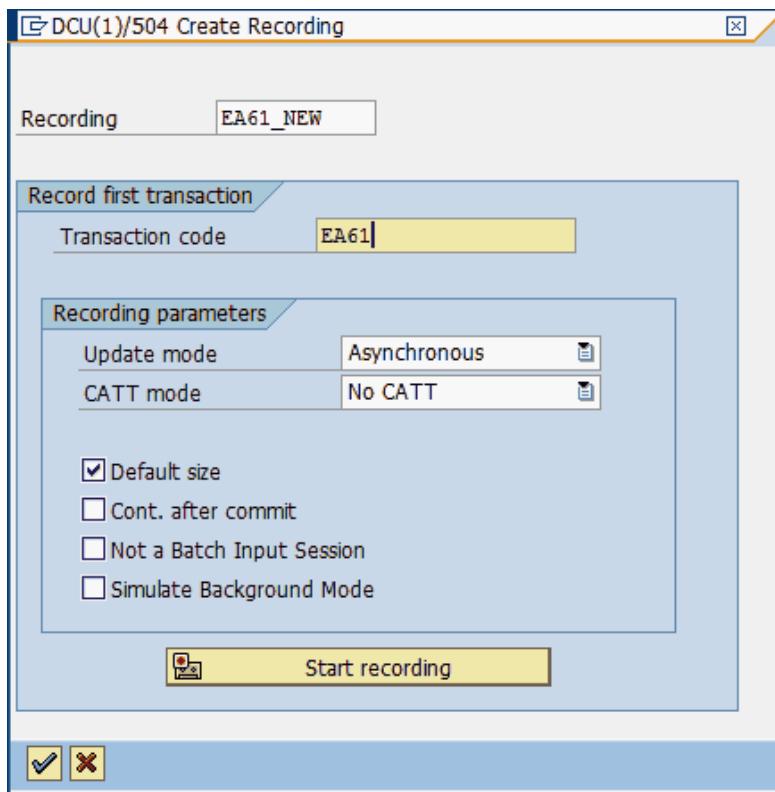


Figure 4.121: Start recording transaction EA61

The recording starts with the selection screen of **TRANSACTION EA61**. Enter the **BUSINESS PARTNER**, **CONTRACT ACCOUNT**, or **CONTRACT** with a free selected date in the billing plan period (**DATE OF BB PERIOD**) (see Figure 4.122).

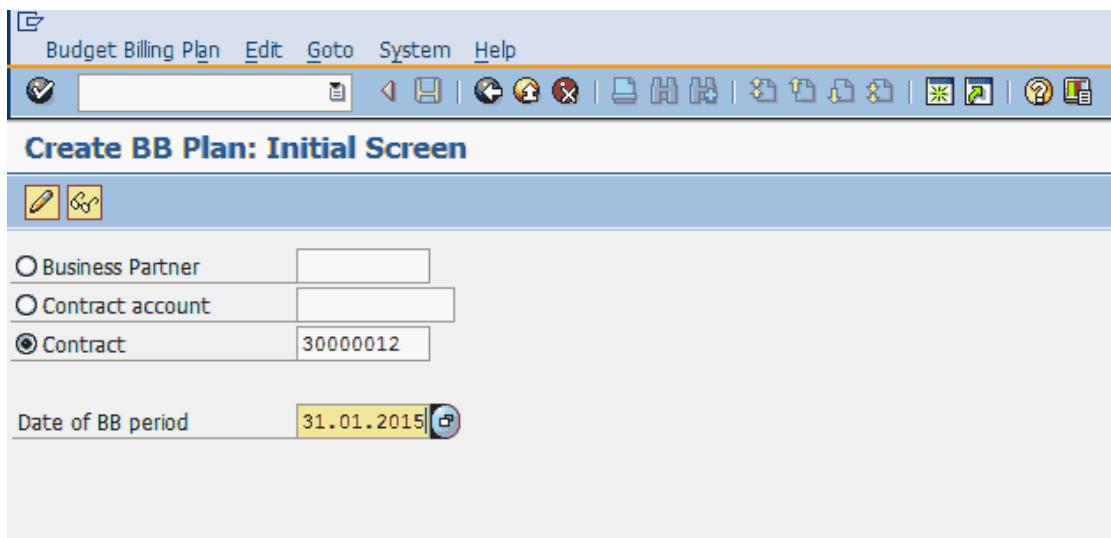
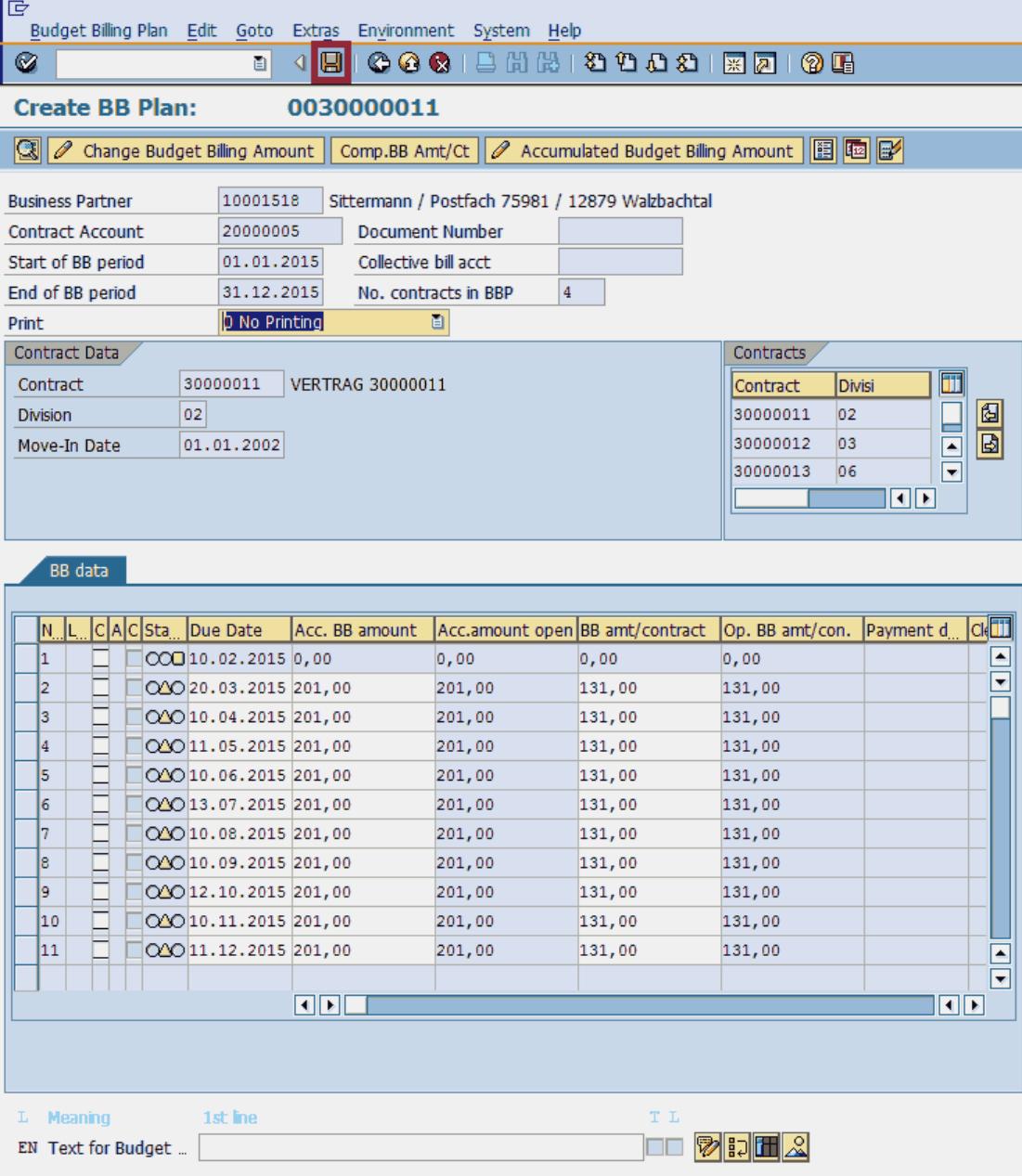


Figure 4.122: Selection screen of EA61 in transaction recording

Confirm the input by hitting the “Enter” key to create a billing plan using the standard SAP transaction. Save this new billing plan by clicking the  icon (see Figure 4.123).



N.	L.	C	A	C	Sta...	Due Date	Acc. BB amount	Acc.amount open	BB amt/contract	Op. BB amt/con.	Payment d...	Cl
1				OO	10.02.2015	0,00	0,00	0,00	0,00	0,00		
2				OΔO	20.03.2015	201,00	201,00	131,00	131,00	131,00		
3				OΔO	10.04.2015	201,00	201,00	131,00	131,00	131,00		
4				OΔO	11.05.2015	201,00	201,00	131,00	131,00	131,00		
5				OΔO	10.06.2015	201,00	201,00	131,00	131,00	131,00		
6				OΔO	13.07.2015	201,00	201,00	131,00	131,00	131,00		
7				OΔO	10.08.2015	201,00	201,00	131,00	131,00	131,00		
8				OΔO	10.09.2015	201,00	201,00	131,00	131,00	131,00		
9				OΔO	12.10.2015	201,00	201,00	131,00	131,00	131,00		
10				OΔO	10.11.2015	201,00	201,00	131,00	131,00	131,00		
11				OΔO	11.12.2015	201,00	201,00	131,00	131,00	131,00		

Figure 4.123: Newly created billing plan

After the new billing plan has been saved, **TRANSACTION EA61** ends. After this, the SAP system shows the recorded program steps in change mode (see Figure 4.124).

**Transaction Recorder: Change Recording EA61\_NEW**

Line	Program	Screen	St...	Field name	Field value
1			T	EA61	
2	SAPLEA61	0100	X		
3				BDC_CURSOR	REA61-DATUM
4				BDC_OKCODE	/00
5				REA61-VERTRAG_RB	X
6				REA61-VERTRAG	30000012
7				REA61-DATUM	31.01.2015
8	SAPLEA61	0300	X		
9				BDC_CURSOR	REA61-PRINT_POSS
10				BDC_OKCODE	=SAVE
11				REA61-PRINT_POSS	0
12				BDC_SUBSCR	SAPLEENO 1001NOTICE
13				BDC_SUBSCR	SAPLEA61 0301MAIN

Figure 4.124: Record of transaction EA61

Save this recording by clicking the icon again. With a click on the icon, you return to the recording overview (see Figure 4.125).

**Transaction Recorder: Recording Overview**

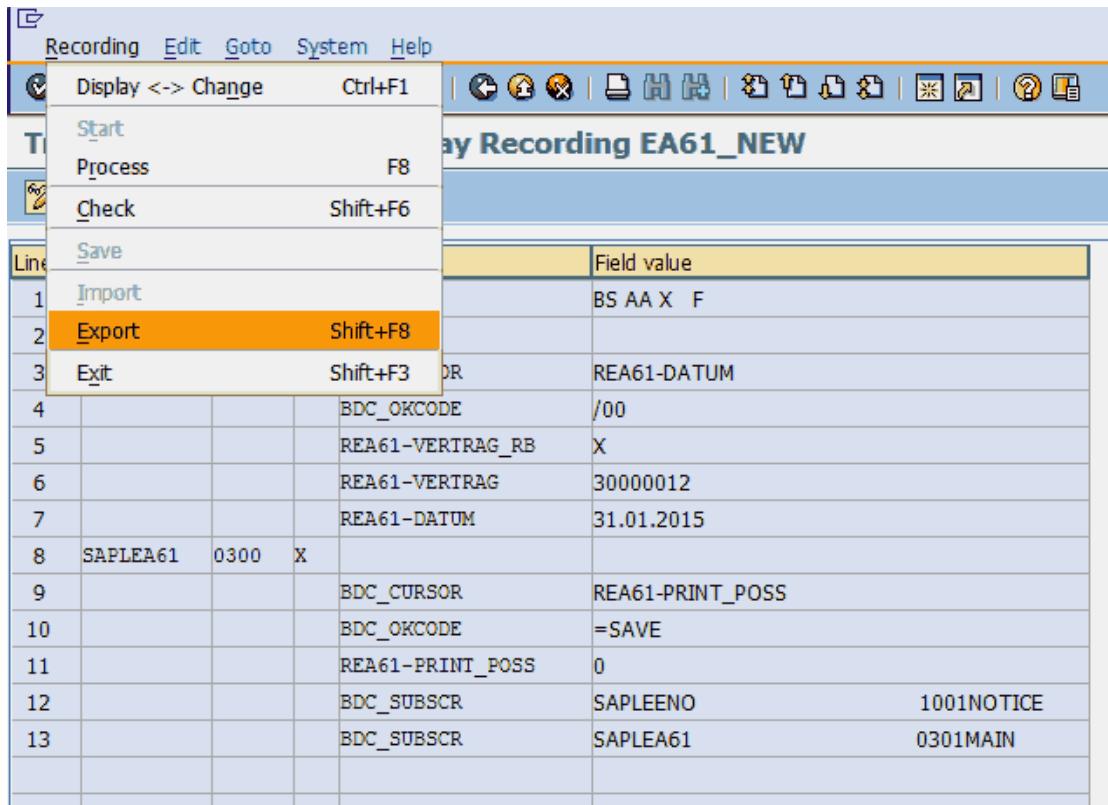
Recording:	CreatedBy	Date	Time	Transact.	Screens
EA61_NEW	STUTETH	05.03.2015	14:45:23	1	2

Figure 4.125: Saved transaction recording

The recording of **TRANSACTION EA61** is ready.

If the recording could not be created on the development system due to missing test data, you have to transfer the recording from the test system to the development system.

Therefore, call the newly created recording “EA61\_NEW” with a double click on its name in display mode. The recording can be stored in a local file using the pull-down menu **RECORDING • EXPORT** (see Figure 4.126).



The screenshot shows the SAP transaction recording interface for EA61\_NEW. The menu bar at the top includes Recording, Edit, Goto, System, and Help. Below the menu bar is a toolbar with various icons. The main area displays a table of recorded data. The table has columns for Line, Field, and Value. The first few rows show fields like BS\_AA\_X\_F, REA61-DATUM, BDC\_OKCODE, etc. Row 2 is highlighted with a yellow background, and the 'Export' option in the menu bar above it is also highlighted in yellow. The menu bar also lists other options like Start, Process, Check, Save, Import, Exit, and Display <-> Change.

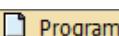
Line	Field	Value	
1	BS_AA_X_F	BS AA X F	
2	REA61-DATUM		
3	BDC_OKCODE	/00	
4	REA61-VERTRAG_RB	X	
5	REA61-VERTRAG	30000012	
6	REA61-DATUM	31.01.2015	
7	SAPLEA61 0300 X		
8	BDC_CURSOR	REA61-PRINT_POSS	
9	BDC_OKCODE	=SAVE	
10	REA61-PRINT_POSS	0	
11	BDC_SUBSCR	SAPLEENO 1001NOTICE	
12	BDC_SUBSCR	SAPLEA61 0301MAIN	
13			

Figure 4.126: Export a transaction recording

A record can only be exported or imported if it is opened in display or change mode. So, before you can import the record, create a new empty recording and open it.

#### 4.20.2 Generate a program from a recording

The next step is to create source code out of the transaction record. This source code can be used later in a customer program.

Mark the recording “EA61\_NEW” in the recording overview (see Figure 4.125). The SAP system automatically generates an ABAP program out of the recording when you click the  **Program** button.

Type the program name in the following display and click the radio button beside **TRANSFER FROM RECORDING** (see Figure 4.127).

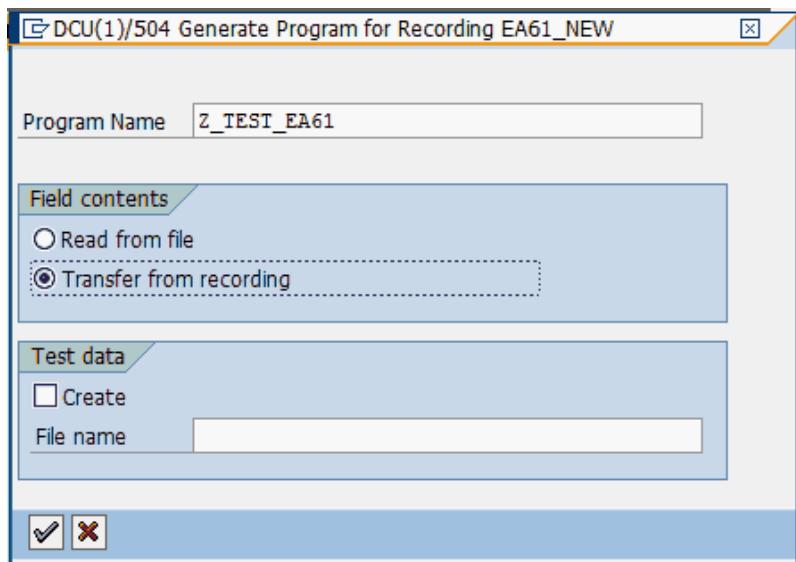


Figure 4.127: Generate a program from a transaction recording

Confirm your selection using the icon. In the next screen, you have to name the program in the **TITLE** field, and fill in other known attributes (see Figure 4.128).

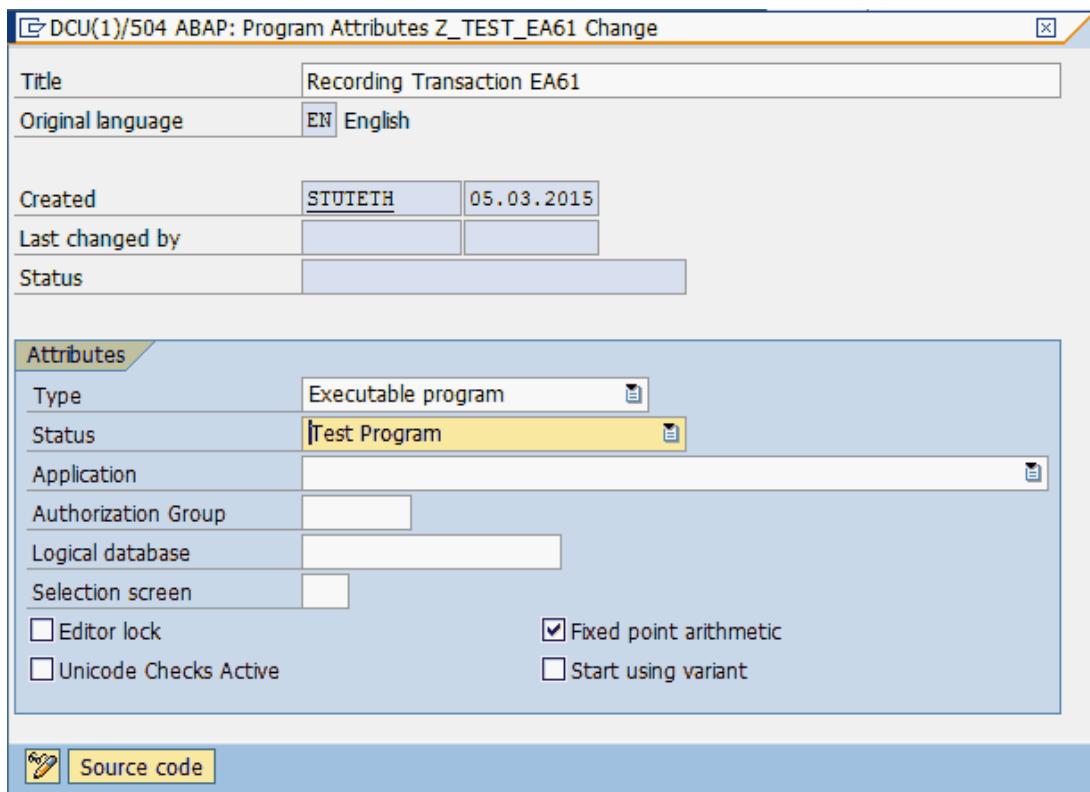


Figure 4.128: Input program attributes

Display the generated program by clicking the button (see Figure 4.129).

The screenshot shows the SAP ABAP Editor interface. The title bar reads "ABAP Editor: Change Report Z\_TEST\_EA61". The toolbar contains various icons for file operations like Open, Save, Print, and Help. Below the toolbar, a menu bar includes "Program", "Edit", "Goto", "Utilities", "Environment", "System", and "Help". The main area is a code editor with the following content:

```
1 report z_test_ea61
2         no standard page heading line-size 255.
3
4 include bdcrcx1.
5
6 start-of-selection.
7
8 perform open_group.
9
10 perform bdc_dynpro      using 'SAPLEA61' '0100'.
11     perform bdc_field    using 'BDC_CURSOR'
12                     'REA61-DATUM'.
13     perform bdc_field    using 'BDC_OKCODE'
14                     '/00'.
15     perform bdc_field    using 'REA61-VERTRAG_RB'
16                     'X'.
17     perform bdc_field    using 'REA61-VERTRAG'
18                     '30000012'.
19     perform bdc_field    using 'REA61-DATUM'
20                     '31.01.2015'.
21     perform bdc_dynpro      using 'SAPLEA61' '0300'.
22     perform bdc_field    using 'BDC_CURSOR'
23                     'REA61-PRINT_POSS'.
24     perform bdc_field    using 'BDC_OKCODE'
25                     '=SAVE'.
26     perform bdc_field    using 'REA61-PRINT_POSS'
27                     '0'.
28     perform bdc_transaction using 'EA61'.
29
30 perform close_group.
```

Figure 4.129: Source code of the generated program from a transaction recording

#### 4.20.3 Take over commands of transaction recording

We want to develop a program that can create billing plans for a large amount of selected contracts. In the following section, I will explain which code lines of the generated program you can use for the new development.

##### ABAP commands of the recording



The program, which has been generated from a transaction recording, contains three relevant command types (see Figure 4.129):

1. BDC\_SCREEN: Calls a screen.
2. BDC\_OKCODE: Processes a program function, such as saving or entering (This process runs in the program after filling in the necessary fields; in the source code of the program these commands are positioned directly after the BDC\_SCREEN command).
3. BDC\_FIELD: Fills a field on a screen.

In principle, every transaction recording runs in the called order.

BDC\_CURSOR puts the cursor in the screen on a defined field. This command does not need to be transferred to our customer program.

BDC\_OKCODE is defined through the PF-status of the current called screen status. Some standard BDC codes exist, such as “/00” for enter or “=SAVE” to save input data.

See [Appendix E](#) for a list of well-known BDC\_OKCODES.

To transfer the commands from the recording to our customer program, change the input values, such as those for the contract with parameters.

Follow these steps:

- ▶ The form routines OPEN\_GROUP and CLOSE\_GROUP are needed only for batch input; they are not needed for the call transaction method. Therefore, this routine must not transfer to our customer program.
- ▶ The form routine BDC\_TRANSACTION is substituted with the ABAP command CALL TRANSACTION.
- ▶ The form routines BDC\_SCREEN and BDC\_FIELD are needed in the customer program to create and transfer the necessary parameters for **TRANSACTION EA61**.

- ▶ The dates in the transaction recordings have the internal ABAP format YYYYMMDD of the data type DATS. For the call transaction method, use the external date format MM-DD-YYYY. You need to convert the date values from the internal format to the external date format (depending on the settings of your own data, by using the pull-down menu **SAP EASY MENU • SYSTEM • USER PROFILE • OWN DATA** and the tab **DEFAULTS**).

Figure 4.130 shows the source code of the form routine for creating billing plans.

```

FORM create_billingplan  CHANGING p_gs_output
                           TYPE zcu_firstcustomer_billplan_ana.

DATA: lv_year(4)      TYPE c,
      lv_month(2)    TYPE c,
      lv_day(2)       TYPE c,
      lv_date(10)     TYPE c.

* Date in billing period
lv_year = pa_datap(4).
lv_month = pa_datap+4(2).
lv_day   = pa_datap+6(2).

CONCATENATE lv_month
           lv_day
           lv_year
      INTO lv_date SEPARATED BY '-'.

PERFORM bdc_dynpro      USING 'SAPLEA61' '0100'.
PERFORM bdc_field        USING 'BDC_OKCODE'
                           '/00'.

PERFORM bdc_field        USING 'REA61-KUNDE_RB'
                           ''.

PERFORM bdc_field        USING 'REA61-GPART'
                           p_gs_output-gpart.
PERFORM bdc_field        USING 'REA61-VKONT'
                           p_gs_output-vkonto.
PERFORM bdc_field        USING 'REA61-VERTRAG_RB'
                           'X'.
PERFORM bdc_field        USING 'REA61-VERTRAG'
                           p_gs_output-vertrag.
PERFORM bdc_field        USING 'REA61-DATUM'
                           lv_date.
PERFORM bdc_dynpro      USING 'SAPLEA61' '0300'.
PERFORM bdc_field        USING 'BDC_OKCODE'
                           '=SAVE'.

PERFORM bdc_field        USING 'REA61-PRINT_POSS'
                           '1'.

PERFORM bdc_dynpro      USING 'SAPLSPO1' '0100'.
PERFORM bdc_field        USING 'BDC_OKCODE'
                           '=NO'.

CALL TRANSACTION 'EA61' USING bdcdata
                      MODE bdc_mode
                      UPDATE upd_mode
                      MESSAGES INTO messtab.

```

Figure 4.130: Using commands of the recorded transaction for call transaction methods

The form routines BDC\_SCREEN and BDC\_FIELD consist of the source code shown in Figure 4.131.

```

*-----*
*      Start new screen
*-----*

□ FORM bdc_dynpro USING program dynpro.
  CLEAR bdcdata.
  bdcdata-program = program.
  bdcdata-dynpro = dynpro.
  bdcdata-dynbegin = 'X'.
  APPEND bdcdata.
- ENDFORM.                               "bdc_dynpro

*-----*
*      Insert field
*-----*

□ FORM bdc_field USING fnam fval.
  CLEAR bdcdata.
  bdcdata-fnam = fnam.
  bdcdata-fval = fval.
  APPEND bdcdata.
- ENDFORM.                               "bdc_field

```

Figure 4.131: Form routines BDC\_SCREEN and BDC\_FIELD

You can use different update and calling options for the CALL TRANSACTION command (see Figure 4.132).

```

* Declarations for Call Transaction
DATA: bdcdata  LIKE bdcdata    OCCURS 0 WITH HEADER LINE,
       messtab  LIKE bdcmsgcoll OCCURS 0 WITH HEADER LINE,
       bdc_mode TYPE c VALUE 'E',
□ *      Mode E = Stop in case of error
  *      Mode N = Background processing (not visible)
  *      Mode A = Online-processing (visible).
  upd_mode TYPE c VALUE 'S'.
□ *      Mode A = Asynchron (analog: without Commit)
  *      Mode S = Synchron (analog: Commit Work and Wait on Commit)
  *      Mode L = Local booking (analog: command SET UPDATE TASK LOCAL)

```

Figure 4.132: Update and calling modes in call transaction method

For testing the customer program, use the BDC\_MODE = “A.” With this configuration, the system displays all steps of the called transactions (here **EA61**). For a production run, the parameter BDC\_MODE should have the value “N.” With this option, all steps of the called transaction are invisible. Error messages from the called transaction are returned to the calling program in table MESSTAB. You have to install an appropriate error message and a success handling message for the user.

Use the same method to develop a form routine for the deletion of budget billing plans (**TRANSACTION E61D**). The customer program

ZCU\_FIRST\_CUSTOMER\_BILLINGPLAN can delete billing plans for the old meter reading unit and create new billing plans for the new meter reading unit. By using the call transaction method, the program uses a standard SAP transaction and so it guarantees a consistent database.

# 5 What has been implemented at this time

After reviewing the practice requirement explained in [Chapter 1](#), you can develop the following solutions with the described methods:

- ▶ Create the customer-specific database table ZGENERALCONTRACT.
- ▶ Enhance the customer to include CI\_EVER for the assignment of general contracts to utility contracts.
- ▶ Create and display a search help for the field **GENERAL\_CONTRACT\_NUMBER**.
- ▶ Create a search help for the selection of general contracts in the standard SAP transaction for the creation and changing of contracts.

With the exception of the check table for the account class, all basic programs for storing new data are done with the Data Dictionary.

By following the instructions in this text, you can develop ABAP programs to:

- ▶ Upload general contract data.
- ▶ Adjust contract accounts to a new account class.
- ▶ Adjust meter reading units with the help of new time slices for the installations.
- ▶ Create specific customer contacts by using the new account class.
- ▶ Adjust billing plans.

Details on how to develop solutions for all explained requirements, methods, and procedures for the modification-free enhancement of standard

SAP programs are missing. These methods are explained in a second volume of the *Practical Guide to SAP ABAP: Part 2*.

The second part contains explanations to improve the performance of ABAP programs, a short instruction for customizing SAP software, basic know how about the SAP transport management system, methods to find and remove errors in ABAP programs, hints for the handling of SAP notes, and orientation for SAP Support Portal.

In [Appendix B](#), you can find a list of the sample programs. These sample programs contain the coding of the methods and procedures explained in this book.

You can download this example program from the Espresso-Tutorials, <http://abap.espresso-tutorials.com>.



**You have finished the book.**

**Sign up for our newsletter!**



Want to learn more about new e-books?

Get exclusive free downloads and SAP tips.

Sign up for our newsletter!

Please visit us at [newsletter.espresso-tutorials.com](http://newsletter.espresso-tutorials.com) to find out more.

# A Author



**Thomas Stutenbäumer** has worked for more than 25 years as an IT professional for public utilities. After his study of mathematics at Westfälischen-Wilhelms-Universität in Münster, Germany he was responsible for the reorganization of the IT organization of a medium-sized public utility. He has worked for different professional organizations, published articles on the applications of new information technologies, and appeared as a speaker at utility industry events. For more than 10 years, he has developed ABAP programs and has modified standard SAP code. Since 2013, Thomas has been a senior consultant at the enterprise ConUti GmbH.

# B Sample programs

## Program name

### Attributes

ZCU\_ABAP\_ALVGRID\_SCREEN

ALV grid output in dialog programming

ZCU\_ABAP\_ALV\_GRID\_TEMPLATE

Template for creating ALV grid output in list reporting

ZCU\_ABAP\_ASSIGN\_COMMAND

Use of field symbols in LOOP “LOOP AT gt\_table ASSIGNING <gs\_struktur>”

ZCU\_ABAP\_DOWNLOAD\_TABLEDATA

Dynamic SELECT statement

ZCU\_ABAP\_DOWNLOAD\_ZGENERALCONT

Download of records of SAP tables on directory of application server or workstation

ZCU\_ABAP\_BACKGROUND\_DEBUGGING

Background debugging

ZCU\_ABAP\_JOIN\_COMMAND

Select with JOIN option

ZCU\_ABAP\_SEND\_MAIL

Send e-mail from SAP system

ZCU\_ABAP\_NUMBERRANGE\_INSTALL

Use of number ranges

ZCU\_ABAP\_SOURCE\_CODE

Read source code of ABAP programs

ZCU\_ABAP\_TIME\_STAMP

Use of data type TIMESTAMP

ZCU\_ABAP\_UPLOAD\_ZGENERALCONTR

Upload of records from application server or workstation

ZCU\_INCLUDE\_APPLICATION\_USAGE

Include log use of custom applications

ZCU\_FIRST\_CUSTOMER\_MR\_UNIT

Different cases for treading of time slices, ALV grid output with modules

ZCU\_FIRST\_CUSTOMER\_BILLINGPLAN

Call transaction, ALV grid output with modules

ZCU\_FIRST\_CUSTOMER\_CONTACT

Create business partner contacts

ZCU\_FIRST\_CUSTOMER\_ACCOUNT\_CLA  
ENQUE and DEQUEUE modules, change documents

# C Useful SAP transactions

<b>Transaction</b>	<b>Function</b>
AL11	SAP directories
FQVENTS	Management of events
SCC4	Changes and transport of objects
SE06	System change options
SCDO	Change documents objects – Overview
SE11	ABAP dictionary – Initial screen
SE24	Class builder – Initial screen; create, display, change, and delete classes
SE37	Function builder – Initial screen; create, display, change, and delete modules
SE38	ABAP editor – Initial screen; create, display, change, and delete programs and includes
SE80	Object navigator
SE91	Message maintenance – Initial screen
SE93	Maintain transaction
SM04	List of logged-in users
SM12	Select log entries
SM35	Batch-input – Session overview
SM50	Process overview

ST22

ABAP runtime errors

# D Useful SAP database tables

## Table Name

### Table Content

CDHDR

Change document header

CDPOS

Change document item

DEVACCESS

Table for development user (development key)

E070

Change and transport system – Header of requests/tasks

E071

Change and transport system – Object entries of requests/tasks

E07T

Change and transport system – Short text of requests/tasks

NRIV

Number range intervals

SE16N\_CD\_DATA

Table display – Change documents data (Logged data of changed records with transaction SE16N)

S16N\_CD\_KEY

Table display – Change documents data (Header data of logged changed records with transaction SE16N)

T001

Company codes (system and clients)

T100

SAP messages  
(change message type, e.g., E -> W)

TADIR

Directory of repository objects

TFK\*

Customizing tables for FI-CA

TSTC

SAP transaction codes

REPOSCR

Report source (source code of ABAP programs)

# **E BDC\_OKCODE**

## **BDC\_OKCODE**

### **Meaning**

/00

Enter

ENTE

Enter

ENTR

Enter

UPDA

Update data, save data

UPD

Update data, save data

SAVE

Save data

SICH

Save data

BU

Book/save data

BACK

One screen back (analogy /3)

MAIN

Show main menu

EEND

Finish program resp. transaction

YES

Choose “Yes” in popup screen

PI

Select cursor position

DELZ

Delete cursor

P++

Jump to last page

P+

One page down

P--

Jump to first page

P--

One page up

/3

Function key F3 = back

/15

Function key F15 = finish

# **F Disclaimer**

This publication contains references to the products of SAP SE.

SAP, R/3, SAP NetWeaver, Duet, PartnerEdge, ByDesign, SAP BusinessObjects Explorer, StreamWork, and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP SE in Germany and other countries.

Business Objects and the Business Objects logo, BusinessObjects, Crystal Reports, Crystal Decisions, Web Intelligence, Xcelsius, and other Business Objects products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of Business Objects Software Ltd. Business Objects is an SAP company.

Sybase and Adaptive Server, iAnywhere, Sybase 365, SQL Anywhere, and other Sybase products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of Sybase, Inc. Sybase is an SAP company.

SAP SE is neither the author nor the publisher of this publication and is not responsible for its content. SAP Group shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP Group products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.

# More Espresso Tutorials eBooks



Sydnie McConnell, Martin Munzel:  
[First Steps in SAP®, 2nd edition](#)

- ▶ Learn how to navigate in SAP ERP
- ▶ Learn SAP basics including transactions, organizational units, and master data
- ▶ Watch instructional videos with simple, step-by-step examples
- ▶ Get an overview of SAP products and new development trends

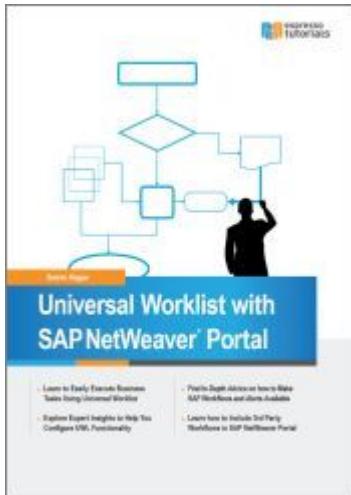


Antje Kunz:  
[SAP® Legacy System Migration Workbench \(LSMW\)](#)

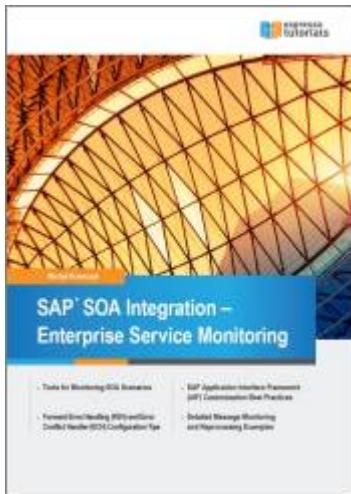
- ▶ Data Migration (No Programming Required)
- ▶ SAP LSMW Explained in Depth
- ▶ Detailed Practical Examples
- ▶ Tips and Tricks for a Successful Data Migration

Darren Hague:  
[Universal Worklist with SAP® NetWeaver Portal](#)

- ▶ Learn to Easily Execute Business Tasks Using Universal Worklist
- ▶ Explore Expert Insights to Help You Configure UWL Functionality

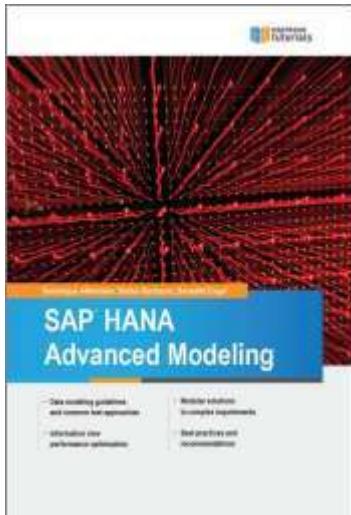


- ▶ Find In-Depth Advice on how to Make SAP Work-flows and Alerts Available
- ▶ Learn how to Include 3rd Party Workflows in SAP NetWeaver Portal



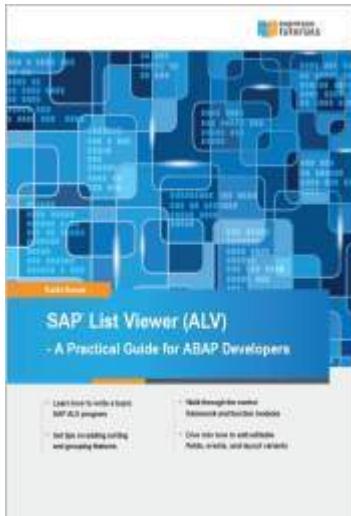
Michał Krawczyk:  
[SAP® SOA Integration - Enterprise Service Monitoring](#)

- ▶ Tools for Monitoring SOA Scenarios
- ▶ Forward Error Handling (FEH) and Error Conflict Handler (ECH)
- ▶ Configuration Tips
- ▶ SAP Application Interface Framework (AIF) Customization Best Practices
- ▶ Detailed Message Monitoring and Reprocessing Examples



Dominique Alfermann, Stefan Hartmann, Benedikt Engel:  
[SAP® HANA Advanced Modeling](#)

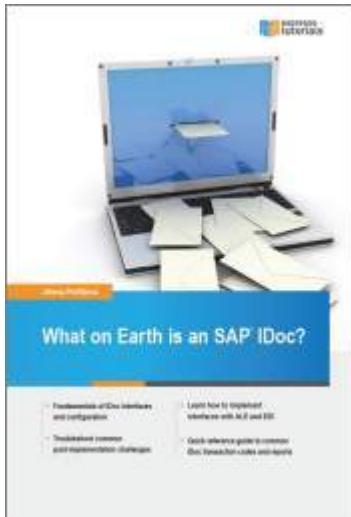
- ▶ Data modeling guidelines and common test approaches
- ▶ Modular solutions to complex requirements
- ▶ Information view performance optimization
- ▶ Best practices and recommendations



Kathi Kones

[SAP List Viewer \(ALV\) – A Practical Guide for ABAP Developers](#)

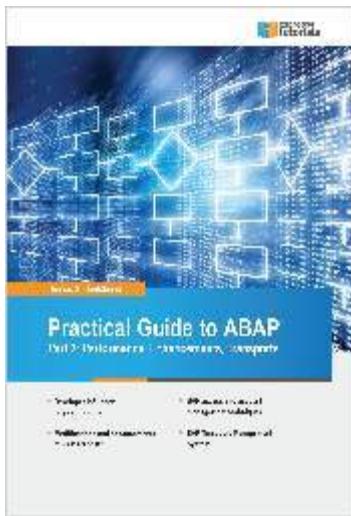
- ▶ Learn how to write a basic SAP ALV program
- ▶ Walk through the object-oriented control framework and function modules
- ▶ Get tips on adding sorting and grouping features
- ▶ Dive into how to add editable fields, events, and layout variants



Jelena Perfiljeva

[What on Earth is an SAP IDoc?](#)

- ▶ Fundamentals of inbound and outbound IDoc interfaces and configuration
- ▶ Learn how to implement interfaces with ALE and EDI
- ▶ Troubleshoot common post-implementation challenges
- ▶ Quick reference guide to common IDoc transaction codes and reports



Thomas Stutenbäumer

[Practical Guide to ABAP. Part 2: Performance, Enhancements, Transports](#)

- ▶ Developer influence on performance
- ▶ Modifications and enhancements to SAP standard
- ▶ SAP access and account management techniques
- ▶ SAP Transport Management System

