SAP PRESS

SAP® Essentials

ABAP™ Development for Financial Accounting: Custom Enhancements

- Provides tutorials for the custom development of your SAP system
- Covers validations and substitutions, user exits, BTEs, BAdIs, and implicit enhancements
- Includes discussions on report development, accounting document processing, workflows, and more

Sergey Korolev

Galileo Press

Sergey Korolev

ABAP™ Development for Financial Accounting

Custom Enhancements

Galileo Press

Bonn • Boston

Dear Reader,

As you hold *ABAP Development for Financial Accounting* in your hands, you are steps away from learning how to create custom enhancements to standard ABAP code in SAP ERP Financials Financial Accounting (release 6.0) in order to address all corporate and/or country-specific business rules. Thanks to the expert guidance of Sergey Korolev, this book will teach you how to efficiently and effectively customize data flow between subsystems and external systems.

I never know what to expect when working with a first-time author, but my experience with Sergey was a pleasure from day one. His expertise is unparalleled, his dedication and effort were superhuman, and at every step of the way he remained positive, upbeat, and—yes—funny. Although I ruined many a weekend for him, he always came through…and cheerfully, at that. Working with him was truly a joy.

We appreciate your business, and welcome your feedback. Your comments and suggestions are the most useful tools to help us improve our books for you, the reader. We encourage you to visit our website at *www.sap-press.com* and share your feedback about this work.

Thank you for purchasing a book from SAP PRESS!

Kelly Grace Harris
Editor, SAP PRESS

Galileo Press
Boston, MA

kelly.harris@galileo-press.com
www.sap-press.com

Notes on Usage

Imprint

This e-book is a publication many contributed to, specifically:

Editor Kelly Grace Harris
Developmental Editor Laura Korslund
Copyeditor Julie McNamee
Cover Design Graham Geary
Photo Credit Image Copyright KellyM. Used under license from Shutterstock.com.
Production E-Book Graham Geary
Typesetting E-Book Publishers' Design and Production Services, Inc.

We hope that you liked this e-book. Please share your feedback with us and read the Service Pages to find out how to contact us.

The Library of Congress has cataloged the printed edition as follows:
Korolev, Sergey.
ABAP development for financial accounting : custom enhancements / Sergey Korolev. — 1st ed.
p. cm.
Includes index.
ISBN-13: 978-1-59229-370-4
ISBN-10: 1-59229-370-0
1. Accounting—Data processing. 2. SAP ERP. 3. ABAP/4 (Computer program language) I. Title.
HF5679.K595 2011
657.0285'5133—dc22
2010051980

ISBN 978-1-59229-370-4 (print)
ISBN 978-1-59229-739-9 (e-book)
ISBN 978-1-59229-740-5 (print and e-book)

Contents

"One can't believe impossible things," said Alice.
"I daresay you haven't had much practice," said the Queen.

Introduction

Sitting in a rapid train with more than three hours of travel ahead and nothing to watch through the window except for the black-inked void—this is a good working environment for writing an introduction to the book.

When I started working with SAP products, it was a great surprise that a big part of a developer's knowledge cannot be obtained from legitimate sources such as technical guides, but must rather be absorbed from a kind of folklore: word of mouth from a more experienced colleague, or from one or another Internet community. The most esoteric kind of knowledge was methods and ways of enhancing the system, and, during those days, I often wished I had a book or two with a more or less comprehensive description of available user-exits. It now turns out that, instead of reading such a book, I have had the opportunity to write one. Thus, for the last six months, I have been trying to convert a developer's folklore and my own experience in FI programming into a more systematic exposition.

The reader of this book should have a general knowledge of the ABAP programming language, including ABAP objects, and also have a basic understanding of Financial Accounting with SAP ERP Financials—in other words, the reader should be familiar with the phrase "general ledger account."

This book mainly covers Financial Accounting with SAP ERP Financials, and does *not* include Controlling and Asset Accounting. The structure of the book is as follows:

- **Chapter 1**
  This chapter is an introduction to enhancement technologies you can come across when fulfilling development tasks in Financial Accounting.

- **Chapter 2**
  This chapter discusses enhancement techniques for Financial Accounting master data: general ledger accounts, Accounts Receivable, and Accounts Payable.
- **Chapter 3**
  This chapter deals with the accounting document and the process of its posting. This is probably the most sensitive functionality in the system, as it has to do with actually counting money.
- **Chapter 4**
  In this chapter, we discuss methods of enhancement for some standard Financial Accounting reports.
- **Chapters 5 and 6**
  These chapters show possible ways of enhancing several inbound and outbound scenarios when the SAP system exchanges accounting data with external systems.
- **Chapter 7**
  This chapter is a quick introduction to SAP Business Workflow, which is another tool—with often underestimated capabilities and overestimated complexity—that you can use to extend system functionality.

All coding samples and screenshots were prepared using the commercially available SAP ECC 6.0 IDES system.

When Stefan Proksch (Senior Editor at Galileo Press at that time) asked me in April of 2010 if I felt capable of writing a book, I answered "Yes"—but at the same time, a part of my mind believed it was impossible. I even thought it was impossible when the first version of the table of contents was completed. Nevertheless, the book came out. Thus, the Queen was possibly right.

Sergey Korolev
Moscow, Russia

Acknowledgments

I thank Stefan Proksch for the first impulse to start this book; I also greatly thank Kelly Harris, who chased me during the writing process in a friendly yet strong manner. I also would like to thank Laura Korslund and Julie McNamee, who were the editors of the book and worked hard to turn my ugly English into something more readable.

Finally, I thank my family, who survived my absent-mindedness for the last few months.

We begin by reviewing the available enhancement techniques in the SAP system. This review will help you better understand the chapters that follow.

1 Enhancement Types

Even extremely configurable software, such as SAP ERP, can't account for all of the specific requirements of clients. Also, the future is unpredictable; that is, changes in business, in country legislation, and so on have an impact on business-specific software and how companies manage their businesses. In some situations, the standard (even configurable) applications can't cope with particular business circumstances. In these circumstances, customers might want to amend their system's behavior by implementing custom program solutions.

SAP delivers its software with full source code. In a way, it can be considered as an open source system. And before introducing any enhancement techniques, SAP customers modified the source code to implement unavoidable logic extensions. Even in the most recent versions of SAP ERP, some developing activities such as sales document user exits or pricing formula creation are system modifications (due to their formal nature).

The concept of enhancements was born as an attempt to make altering source code more controllable, while retaining considerable freedom for the customer in tailoring unique and business-specific program logic into an existing system.

The available enhancement techniques we'll discuss in this chapter are by essence program hooks, which allow the customer to couple up custom program code with the system. At the same time, these hooks are under SAP ERP control, which makes it possible for the system manufacturer to remain responsible for the whole system's behavior.

1.1 Customer Enhancements (CMOD/SMOD)

Customer enhancements (also known as customer modifications) are the oldest type of enhancement tools available in SAP (the earliest online help for SAP R/3 3.0A shows that customer enhancements were already there).

The three types of components used in customer enhancements are function module exits, menu exits, and customer subscreens. Several components of the same functional purpose are combined into an enhancement. You can display particular enhancements in Transaction SMOD.

To implement a particular exit, a developer must first create an *enhancement project* in Transaction CMOD and assign one or more enhancements to the project. When the developer activates an enhancement project, all components of all enhancements that are assigned to the specific project become active. Because the project is a development and cross-client object, it is transported by the workbench change request.

1.1.1 Function Module Exit

The customer exit function module component is sometimes called in SAP ERP in the form of the `CALL CUSTOMER-FUNCTION NNN` statement, where `NNN` is a number suffix. In runtime, the statement is converted into the call of the function module `EXIT_<program name>_nnn` where `<PROGRAM NAME>` is the name of the currently running main program. For example, the statement `CALL CUSTOMER-FUNCTION '001'` in program SAPMM06E is converted into `CALL FUNCTION 'EXIT_SAPMM06E_001'`.

The source code of any function module exit has one `INCLUDE` statement referencing an include with a name starting with prefix `ZX`, as shown in Listing 1.1.

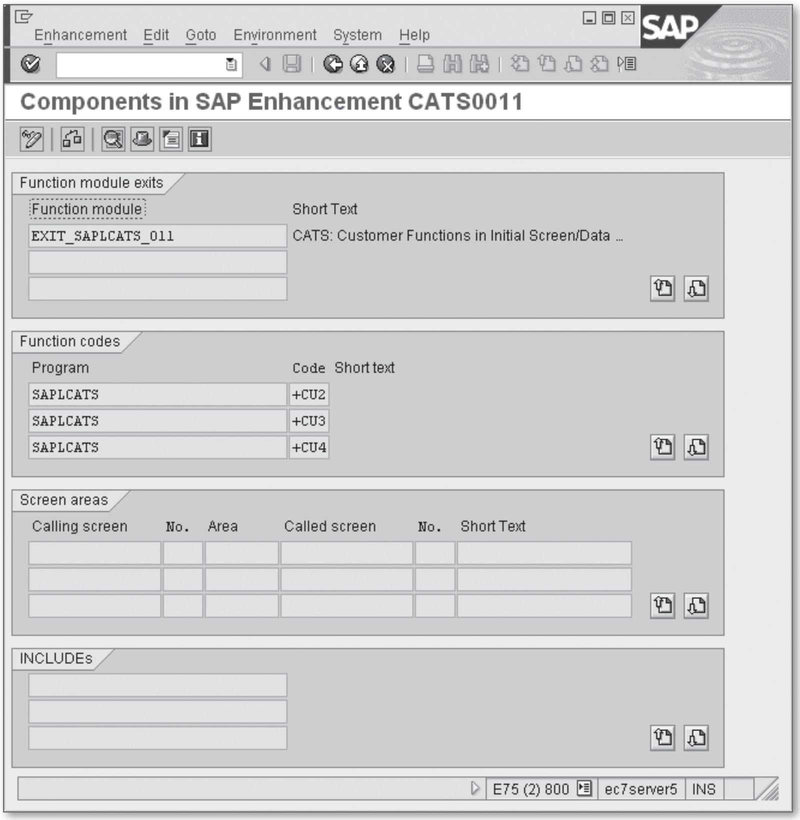
```
FUNCTION EXIT_SAPLCATS_011 .
*"----------------------------------------------------------------------
*"*"Lokale Schnittstelle:
*"       IMPORTING
*"             VALUE(SAP_FCODE) LIKE  SY-UCOMM
*"             VALUE(SAP_TCATS) LIKE  TCATS STRUCTURE  TCATS OPTIONAL
*"             VALUE(SAP_CATSFIELDS) LIKE  CATSFIELDS_COMM
*"                                STRUCTURE  CATSFIELDS_COMM OPTIONAL
*"             VALUE(SAP_CATSD) TYPE  CATSD_EXT_TAB OPTIONAL
*"             VALUE(SAP_CATSW) TYPE  CATSW_TAB OPTIONAL
*"             VALUE(SAP_PERNRLIST) TYPE  PERNR_LIST_TAB OPTIONAL
*"             VALUE(SAP_CURSOR_FIELD) TYPE  TEXT70 OPTIONAL
*"             VALUE(SAP_CURSOR_CATSDLINE) LIKE  SY-STEPL OPTIONAL
*"----------------------------------------------------------------------
INCLUDE ZXCATU12 .
ENDFUNCTION.
```

Listing 1.1 Example of Customer Exit Function Module

The prefix ZX in a name alerts the compiler that the referenced include can be nonexistent. In that case, the INCLUDE statement is ignored by the compiler. To implement enhancement logic, create a corresponding ZX include.

1.1.2 Menu Exit

A number of SAP enhancements also include menu exits. *Menu exits* are just hidden function codes included into particular SAP GUI statuses of an SAP standard program. Due to naming conventions, these function codes start with "+". For example, SAP enhancement CATS0011 includes several function codes: +CU2, +CU3, and +CU4 (see Figure 1.1). When defining an enhancement project that includes menu exits, you can assign an icon and all necessary texts to the menu entry. The menu entries become visible after you activate the project.



Components in SAP Enhancement CATS0011

Function module exits

Function module	Short Text
EXIT_SAPLCATS_011	CATS: Customer Functions in Initial Screen/Data ...

Function codes

Program	Code	Short text
SAPLCATS	+CU2	
SAPLCATS	+CU3	
SAPLCATS	+CU4	

Screen areas

Calling screen	No.	Area	Called screen	No.	Short Text

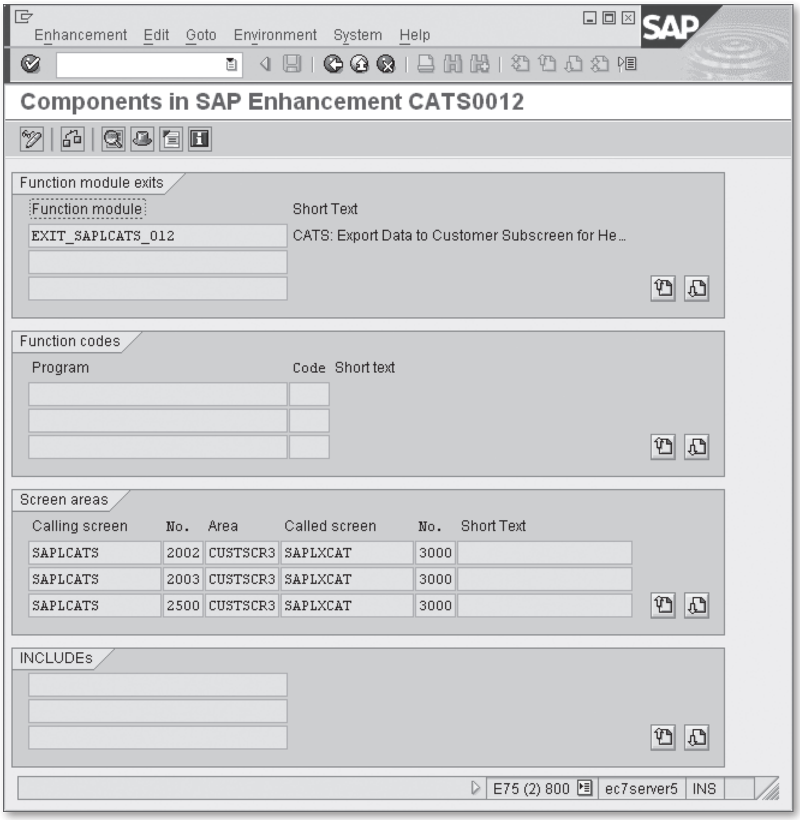
INCLUDEs

Figure 1.1 Enhancement with Menu Entries

As a rule, the enhancement should include a customer function module component where you implement the customer function code processing logic. For example, enhancement CATS0011 also includes function `EXIT_SAPLCATS_011`, which is called to process enhanced function codes.

1.1.3 Customer Exit Subscreen

Some enhancements include customer subscreen components. For example, enhancement CATS0012 includes a subscreen component (see Figure 1.2).



Components in SAP Enhancement CATS0012

Function module exits

Function module	Short Text
EXIT_SAPLCATS_012	CATS: Export Data to Customer Subscreen for He...

Function codes

Program	Code	Short text

Screen areas

Calling screen	No.	Area	Called screen	No.	Short Text
SAPLCATS	2002	CUSTSCR3	SAPLXCAT	3000	
SAPLCATS	2003	CUSTSCR3	SAPLXCAT	3000	
SAPLCATS	2500	CUSTSCR3	SAPLXCAT	3000	

INCLUDEs

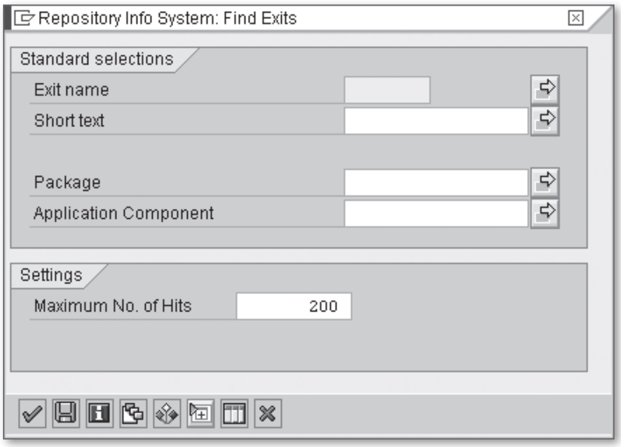
E75 (2) 800 | ec7server5 | INS

Figure 1.2 Subscreen Components of Enhancement CATS0012

The subscreen component includes a calling screen and customer subscreen number. Enhancement CATS0012 has subscreen 3000, which is included into three different screens of program SAPLCATS. When you implement the customer subscreen, you create the corresponding subscreen. In our example, this will be subscreen 3000 of program SAPLXCAT (which is actually function group `XCAT`). Usually an enhancement with a customer subscreen also includes function module components, whose purpose is to transfer data to and from the customer subscreen. In enhancement CATS0012, function module `EXIT_SAPLCATS_012` transfers data to an additional subscreen.

1.1.4 Finding Customer Enhancements

You can always find customer enhancements using either the Workbench Information System or the F4 Search Help in Transaction SMOD. Both tools use the same program. As shown in Figure 1.3, click the All Selections button (⊞) to open more fields.


Repository Info System: Find Exits

Standard selections

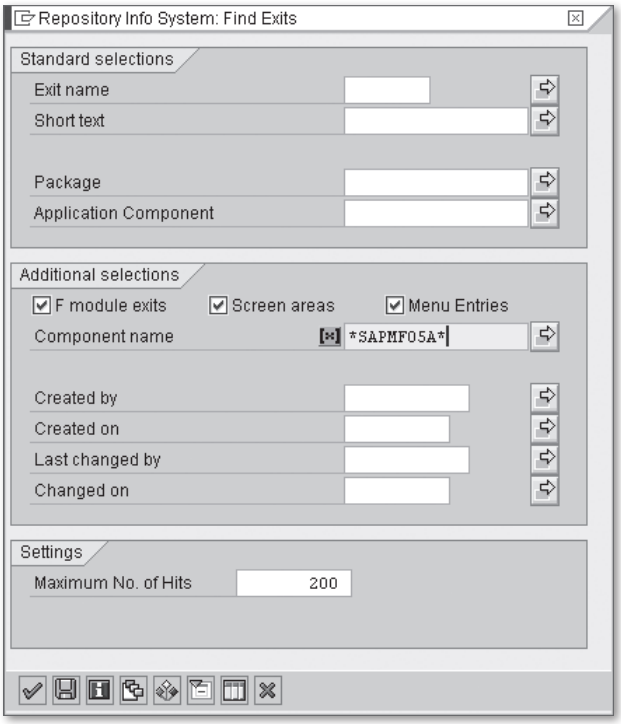
Exit name	
Short text	
Package	
Application Component	

Settings

Maximum No. of Hits	200

Figure 1.3 Search Help Screen of Transaction SMOD

The extended search help screen includes a COMPONENT NAME field where you can enter a program name with asterisks to refine the search. For example, if you want to find a customer exit in the main program SAPMF05A, you should enter "*SAPMF05A*" into the COMPONENT NAME field, as shown in Figure 1.4, and run the search. You will receive an empty result set because there are no function module exits in program SAPMF05A.



[✓] F module exits [✓] Screen areas [✓] Menu Entries

Component name: *SAPMF05A*

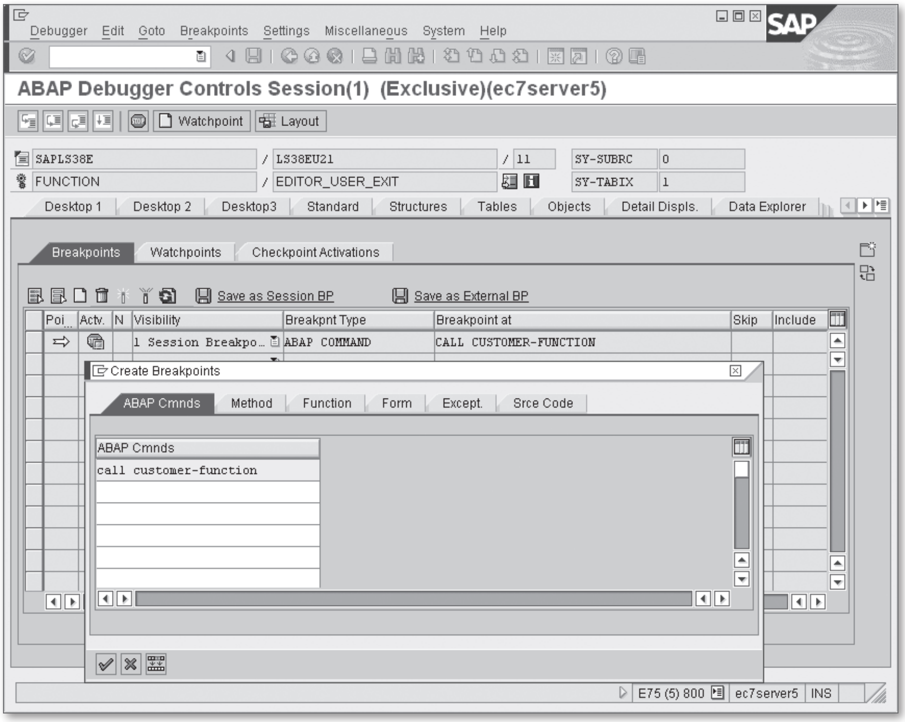
Maximum No. of Hits: 200

Figure 1.4 Extended Search Help Screen of Transaction SMOD

The most effective way of finding a customer exit for a particular transaction is by placing a generic breakpoint. First, you have to enter debugging mode via the `/H` system command, add a breakpoint at the ABAP statement `CALL CUSTOMER-FUNCTION`, run the investigated transaction, and then wait to see if the breakpoint will be hit. In the Debugger (both old and new), you select the menu path BREAKPOINTS • BREAKPOINT AT • BREAKPOINT AT STATEMENT. You then enter the statement in the ABAP CMNDS tab, as shown in Figure 1.5.

The trouble with this method is that sometimes SAP calls customer exits directly using `CALL FUNCTION`. But here you can use an additional breakpoint at function module `MODX_FUNCTION_ACTIVE_CHECK`. This function module checks if a particular customer exit function is activated and thus should always be called before the direct call of the customer exit.

You can also do a Google search for “SAP find user exit program” to find a number of publicly available ABAP utility reports for locating user exits. These reports just retrieve a package (or development class) name from a transaction or other development object, which you can use as a starting point and then search customer exits for a particular package.


Debugger Edit Goto Breakpoints Settings Miscellaneous System Help
Watchpoint
Layout
Desktop 1 Desktop 2 Desktop3 Standard Structures Tables Objects Detail Displs. Data Explorer
Breakpoints Watchpoints Checkpoint Activations
Save as Session BP
Save as External BP
ABAP Cmnds Method Function Form Except. Srce Code

ABAP Debugger Controls Session(1) (Exclusive)(ec7server5)

SAPLS38E	/ LS38EU21	/ 11	SY-SUBRC 0
FUNCTION	/ EDITOR_USER_EXIT		SY-TABIX 1

Poi...	Actv.	N	Visibility	Breakpnt Type	Breakpoint at	Skip	Include
⇨			1 Session Breakpo...	ABAP COMMAND	CALL CUSTOMER-FUNCTION		

Create Breakpoints

ABAP Cmnds
call customer-function

E75 (5) 800 | ec7server5 | INS

Figure 1.5 Defining Breakpoint at the Call Customer-Function Statement

1.1.5 Enhancements Summary

The main drawback of customer exits is that they do not comply with the publish-and-subscribe (P&S) paradigm. This means that you can only define a single include for the whole logic, which can potentially be a problem if more than one developer's work is on the same task involving that particular customer exit. However, there is a workaround for this. For example, you can create a BAdI definition and place its call into the customer exit implementation. After that, you should only use the newly created BAdI to implement additional logic.

SAP tends to supply other types of enhancements in addition to existing function module exits at the same source code locations. So, before deciding to implement a particular customer enhancement, you should investigate the calling point or

Workbench Information System for other types of user exits (BAdIs or BTEs) with the same function.

1.2 Business Transaction Events (BTE)

A Business Transaction Event (BTE) (also known as Open FI or FI Business Framework) is an enhancement technique originally created to extend FI applications. Currently, it is widely used in SAP ERP.

1.2.1 Events and Processes

The BTE framework is implemented as a publish and subscribe (P&S) interface. The P&S interface is a kind of message transferring technique in which the sender never knows the addressee of the message, and subscribers just register their interest in receiving messages of a particular type.

Technically, the framework allows developers to configure one or more function modules to be called back at various moments while running an SAP application. The subscriber function must comply with the predefined interface of the caller. SAP distinguishes two categories of such callbacks: business transaction events and business transaction event processes.

Note

BTE frameworks can be used to establish synchronous communication between remote systems using the SAP RFC (Remote Function Call) protocol.

A *BTE* notifies its subscribers of a particular phase or situation during an application run. Event subscribers are not supposed to change the application data. In practice, however, SAP allows data changing in many BTEs. The full directory of BTEs is stored in configuration table TBE01.

A *BTE process* also notifies subscribers of a particular situation in an application, but unlike in BTEs, a subscriber can or is even expected to change the supplied data. The BTE process has more sophisticated configuration options: It can be reserved only to SAP internal developments, and it has a call mode, allowing multiple or single subscribers to run. In addition, the BTE process has a default function name in its configuration, which is executed if none of the configured subscriptions are found. The full directory of BTE processes is stored in configuration table TPS01.

Both BTEs and processes can be filtered by country and application codes (an *application code* is an additional identification parameter that we'll discuss in Section 1.2.2, Configuration). However, setting the flag in the table is not enough; the flag becomes fully active if it is taken into account at the event or process call point.

Note

Both configuration tables TBE01 and TPS01 are maintainable via Transaction SM30 and have delivery class E, which means that you can add your own entries. However, in practice, you can't add entries to these tables via the standard maintenance dialog because the system requires the existence of special data elements for new entries: `EVNNNNNNNN` for BTEs and `PRNNNNNNNN` for BTE process, where `NNNNNNNN` is the event or process number. These data elements can't be created without an SAP modification key.

To work around this issue, you can create your own maintenance view referencing these tables and append your own BTE and process codes.

1.2.2 Configuration

All BTE configuration options are accessible from Transaction FIBF. Configuration activities for the BTE framework are located in the menu in SETTINGS. Each BTE configuration activity is divided into three groups:

- **SAP internal**
  This area includes the entire configuration that SAP delivers with the installation.
- **Partner**
  This configuration area belongs to SAP's software development partner that is developing its own add-on or application component.
- **Customer**
  This area belongs to each end-client/business that is developing application extensions for its own needs.

Identification

Now let's consider how to display and set up identification of the products in BTE. The menu path of these settings in Transaction FIBF is SETTINGS • IDENTIFICATION. Here you have two options: SAP APPLICATION and PARTNER.

Using the SAP APPLICATION option, you can display a list of application codes with an activation flag. The list is predefined and delivered by SAP.

By changing the ACTIVATION flag, you can turn on and off all BTEs and process subscribers delivered by SAP. SAP application codes for BTEs are stored in configuration table TBE11.

Note

The properties of several SAP applications can't be changed via the maintenance view of Table TBE11. This is hardcoded inside the maintenance dialog.

PARTNER identification is an arbitrary character code that serves as a grouping key for BTE subscriber functions. As with the SAP application, the partner code has its own activation flag, which affects all BTE subscribers assigned to it. The partner identification codes are maintained in Table TBE12.

Products

Product code is another subscriber grouping key. There are two kinds of products: products for partners and products for customers. These products are accessible through the FIBF menu via menu path SETTINGS • PRODUCT.

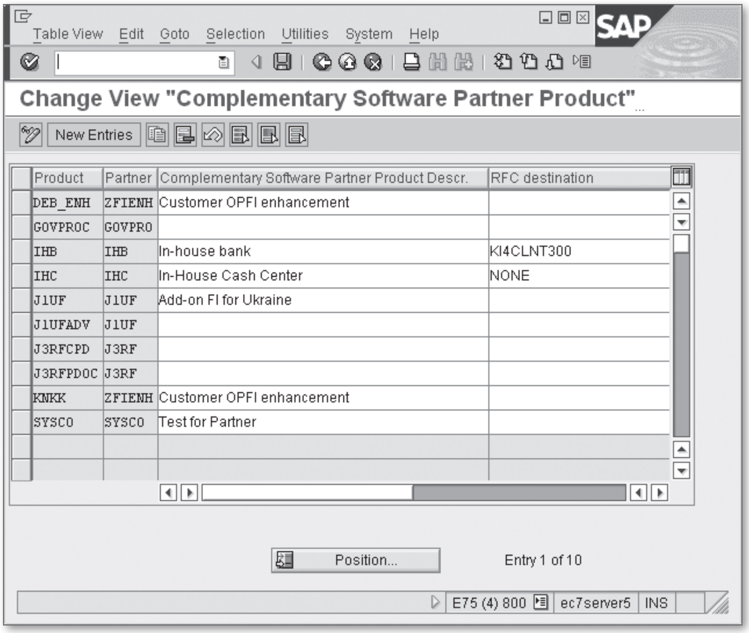
Partner Products

A partner product differs from a customer product because of the activation technique. With a partner product, you use a list of active products instead of checkboxes. You maintain the list of products by accessing the menu path SETTINGS • PRODUCT • OF A PARTNER • EDIT. Here you can see that the partner product configuration also has an RFC destination field, which is used when all of the product subscriber functions must be executed on a remote system (see Figure 1.6).

The list of active product is available when you access menu path SETTINGS • PRODUCT • OF A PARTNER • ACTIVATE. This shows a list of pairs of product code and partner code. Partner products are stored in Tables TBE22 (the list) and TBE23 (activation).

Customer Products

The customer products are given in a simple list with product code, activation flags, description, and an RFC destination name. See the sample customer product list in Figure 1.7.



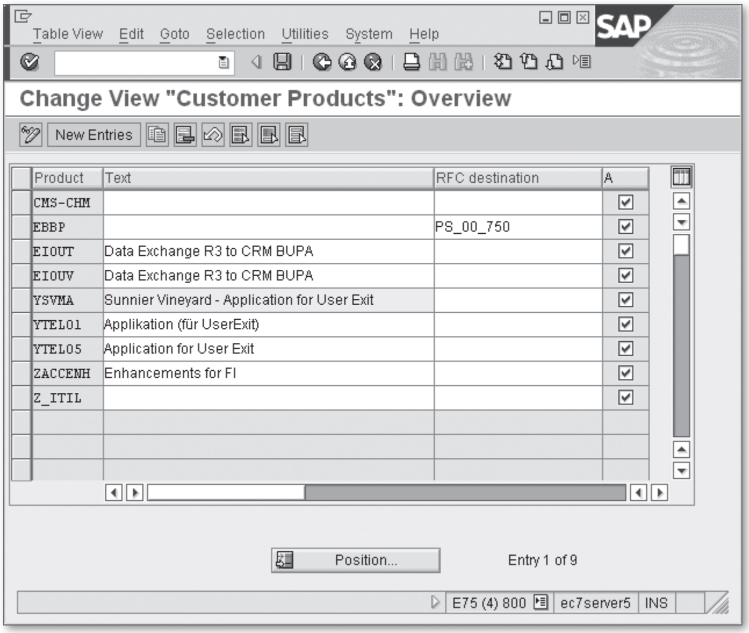
Change View "Complementary Software Partner Product"

Product	Partner	Complementary Software Partner Product Descr.	RFC destination
DEB_ENH	ZFIENH	Customer OPFI enhancement	
GOVPROC	GOVPRO		
IHB	IHB	In-house bank	KI4CLNT300
IHC	IHC	In-House Cash Center	NONE
J1UF	J1UF	Add-on FI for Ukraine	
J1UFADV	J1UF		
J3RFCPD	J3RF		
J3RFPDOC	J3RF		
KNKK	ZFIENH	Customer OPFI enhancement	
SYSCO	SYSCO	Test for Partner	

Entry 1 of 10

E75 (4) 800 ec7server5 INS

Figure 1.6 BTE Partner Products



Change View "Customer Products": Overview

Product	Text	RFC destination	A
CMS-CHM			✓
EBBP		PS_00_750	✓
EIOUT	Data Exchange R3 to CRM BUPA		✓
EIOUV	Data Exchange R3 to CRM BUPA		✓
YSVMA	Sunnier Vineyard - Application for User Exit		✓
YTEL01	Applikation (für UserExit)		✓
YTEL05	Application for User Exit		✓
ZACCENH	Enhancements for FI		✓
Z_ITIL			✓

Entry 1 of 9

E75 (4) 800 ec7server5 INS

Figure 1.7 BTE Customer Product List

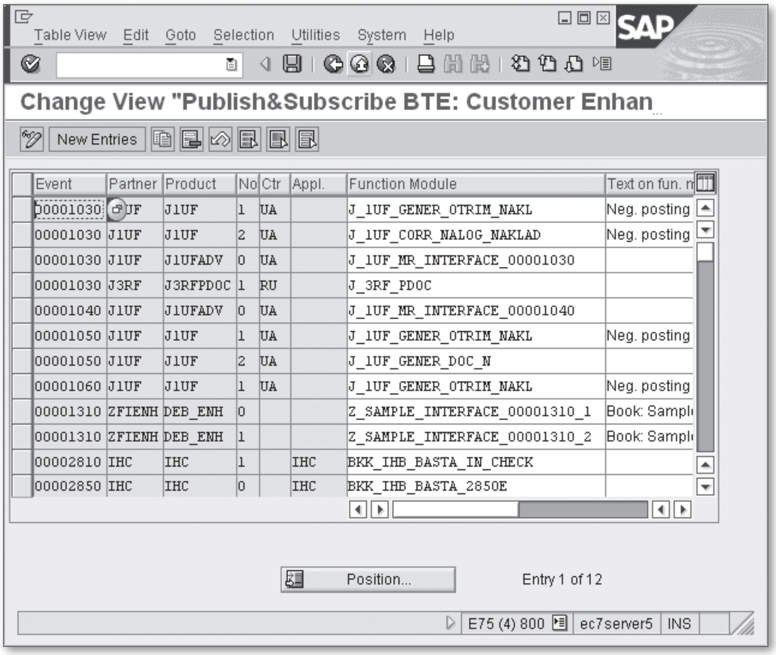
BTE Configuration

You can assign a particular function module to the BTE of your interest in the configuration. In Transaction FIBF, event settings are available via menu path SETTINGS • P/S MODULES. SAP's event list formally can't be changed because the corresponding configuration table has delivery class "S"—meaning its modification is the same as a system modification.

Note that a partner's event configuration slightly differs from that of a customer. We discuss both events in the following subsections.

Partner Events

The key field set of a partner BTE configuration includes an event number, partner code, partner product code, country code, SAP application code, and an implementation number (see Figure 1.8). The event number, partner code, and product code are obligatory items. If you don't fill in the country code or SAP application code, then this particular entry won't be sensitive to the application and country filter.



Change View "Publish&Subscribe BTE: Customer Enhan

Event	Partner	Product	No	Ctr	Appl.	Function Module	Text on fun. m
00001030	UF	J1UF	1	UA		J_1UF_GENER_OTRIM_NAKL	Neg. posting
00001030	J1UF	J1UF	2	UA		J_1UF_CORR_NALOG_NAKLAD	Neg. posting
00001030	J1UF	J1UFADV	0	UA		J_1UF_MR_INTERFACE_00001030	
00001030	J3RF	J3RFPDOC	1	RU		J_3RF_PDOC	
00001040	J1UF	J1UFADV	0	UA		J_1UF_MR_INTERFACE_00001040	
00001050	J1UF	J1UF	1	UA		J_1UF_GENER_OTRIM_NAKL	Neg. posting
00001050	J1UF	J1UF	2	UA		J_1UF_GENER_DOC_N	
00001060	J1UF	J1UF	1	UA		J_1UF_GENER_OTRIM_NAKL	Neg. posting
00001310	ZFIENH	DEB_ENH	0			Z_SAMPLE_INTERFACE_00001310_1	Book: Sampl
00001310	ZFIENH	DEB_ENH	1			Z_SAMPLE_INTERFACE_00001310_2	Book: Sampl
00002810	IHC	IHC	1		IHC	BKK_IHB_BASTA_IN_CHECK	
00002850	IHC	IHC	0		IHC	BKK_IHB_BASTA_2850E	

Figure 1.8 Partner BTE Configuration

The implementation number (with header title No) is a tool that supplies more than one subscription to the event with the same other key values.

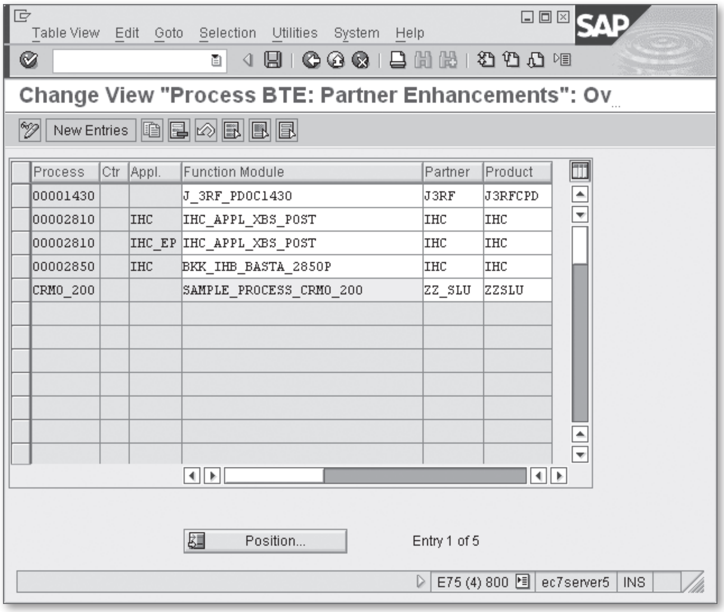
Customer Events

Customer BTE configuration is slightly different from that of partners. It doesn't have an implementation number key column, and it only references the customer product. Nevertheless, you can actually supply more than one subscription to an event by adding additional customer products.

BTE Process Configuration

BTE process settings are available via menu path Settings • Process Modules. As with events, you have three options for SAP internal applications, partner processes, and customer processes. SAP processes can't be modified, due to the delivery class of the corresponding table.

Partner and customer process configuration have similar structures. First note that each configuration entry has only three key fields: process number, country code, and SAP application code (see Figure 1.9). The product and partner code are nonkey fields here; they are necessary to make the entry active/nonactive depending on product properties.



Change View "Process BTE: Partner Enhancements": Ov...

Process	Ctr	Appl.	Function Module	Partner	Product
00001430			J_3RF_PDOC1430	J3RF	J3RFCPD
00002810		IHC	IHC_APPL_XBS_POST	IHC	IHC
00002810		IHC_EP	IHC_APPL_XBS_POST	IHC	IHC
00002850		IHC	BKK_IHB_BASTA_2850P	IHC	IHC
CRM0_200			SAMPLE_PROCESS_CRM0_200	ZZ_SLU	ZZSLU

Figure 1.9 Partner BTE Process Configuration

If the process allows multiple subscriptions, then at runtime, all of the matching process subscriber functions—SAP's, the partner's, and the customer's—will be run.

For single process subscriptions, SAP uses the following logic:

1. First, it checks if the process is marked as SAP internal; in that case, only SAP subscribed functions are executed, and the system doesn't check customer and partner subscriptions.
2. If there is at least one customer subscription, then it is executed.
3. If there is at least one partner subscription, then it is executed, if no customer subscriptions were found.
4. If the system found more than one matching partner or customer subscription for a given process, and the process does not support multiple implementations, then an error message is issued.

1.2.3 Finding Business Transaction Events

As a passive tool for searching BTEs or processes, you can use the BTE Information System accessible in Transaction FIBF via the following menus:

- Environment • Info system (P/S)
- Environment • Info system (Processes)

The BTE Information System includes event or process documentation and the sample function module, which can be copied into your own system. Not all BTEs are provided with documentation, though. There are also some events or processes that have sample function modules with incompatible interfaces, so you should always double-check the sample function and the calling point to make sure your function will be compatible with the call.

A more effective way of finding a BTE is placing a breakpoint into internal BTE function modules and running an SAP transaction or report of your particular interest. When BTE is involved, the SAP system always calls function `BF_FUNCTIONS_READ` for finding events and `PC_FUNCTIONS_READ` for processes. Thus, breakpoints in these function modules can discover the major part of available BTE exits.

There is a problem with debugging, however, because it doesn't always give you 100% accurate results. It isn't always possible to examine *all* combinations of parameter/system configuration combinations that lead to a BTE call.

1.2.4 Business Transaction Events Summary

Some BTEs and processes are interdependent; for example, if one BTE is used for setting additional function codes in a GUI status and another is used for processing such function codes, the latter will likely not work without the former. Sometimes such interdependencies are not so obvious and can be found only by investigating source code or while debugging.

Also, not all BTEs or processes seen in the source code are maintained in BTE configuration tables (you will see such examples later in this book), and thus they are not available for implementation unless you decide to modify Tables TBE01 and/or TPS01 yourself.

1.3 Business Add-In (BAdI)

A business add-in (BAdI) is an object-oriented enhancement tool introduced in SAP R/3 4.6c together with ABAP objects. A BAdI is also a type of P&S technique. Generally (depending on specific BAdI definition properties), you can subscribe more than one ABAP class to the same definition. A BAdI can be marked as SAP internal to prevent a customer from implementing the BAdI.

Simply put, a BAdI actually defines a global object-oriented interface, and the process of BAdI implementation is the creation of an ABAP class implementing that interface. A BAdI runtime framework then selects the activated implementations and runs corresponding class methods.

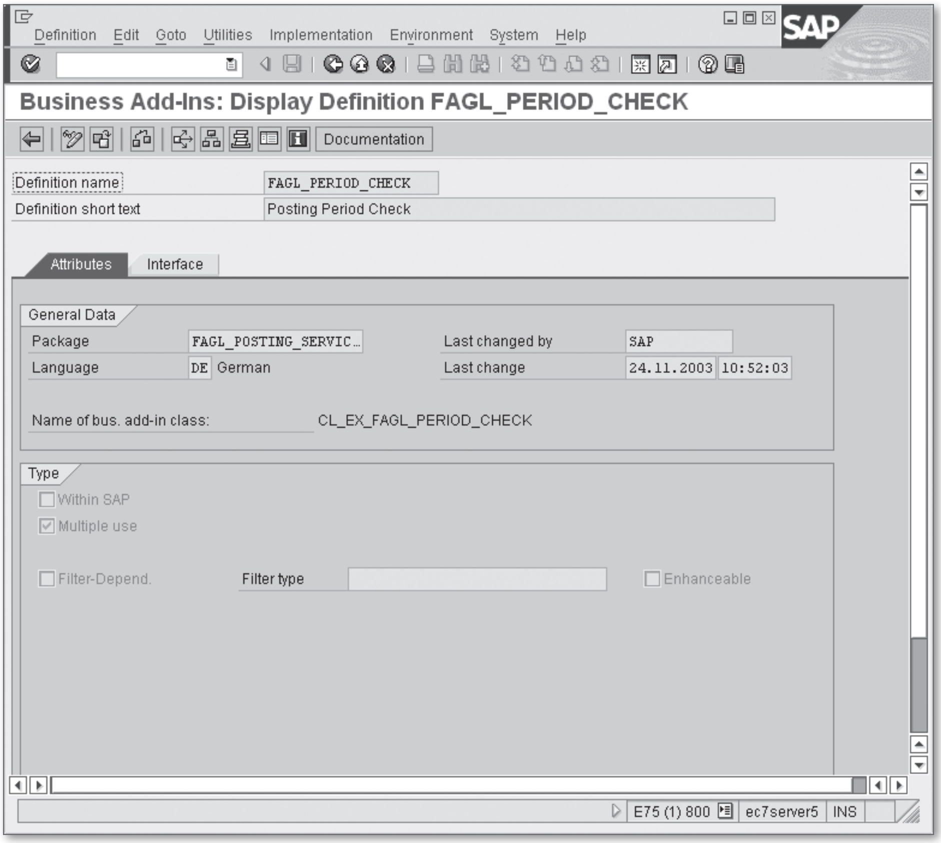
In SAP NetWeaver Application Server (SAP NetWeaver AS), there are two flavors of BAdIs available: classic and kernel-based. In both cases, the BAdI definition actually declares the ABAP object-oriented interface. When the SAP application runs, it instantiates a BAdI class and calls its methods at appropriate moments of data processing. SAP provides specific language statements for BAdI class instance creation for both flavors of BAdI definitions.

BAdI definitions are maintained in Transaction SE18. Both kinds of BAdIs can allow multiple implementations (flag MULTIPLE USE), thus allowing different developers to independently develop several extensions of the same functionality.

The differences in how classic and kernel-base BAdIs are instantiated are discussed next.

1.3.1 Classic BAdI

The classic BAdI runtime framework is implemented in the special global class CL_EXITHANDLER. To instantiate a BAdI definition, the static method GET_INSTANCE of the class CL_EXITHANDLER is used. As a result, it returns a runtime reference to the interface of the corresponding BAdI definition. This interface reference actually points to an instance of intermittent class, which hides all of the internal functionality of the classic BAdI runtime, including multiple implementations and filters (for more about filters, see Section 1.3.3, Filtered BAdIs). Figure 1.10 shows a classic BAdI definition as it looks in Transaction SE18.



Business Add-Ins: Display Definition FAGL_PERIOD_CHECK

Definition name	FAGL_PERIOD_CHECK
Definition short text	Posting Period Check

General Data

Package	FAGL_POSTING_SERVIC...	Last changed by	SAP
Language	DE German	Last change	24.11.2003 10:52:03

Name of bus. add-in class: CL_EX_FAGL_PERIOD_CHECK

Type

☐ Within SAP

☑ Multiple use

☐ Filter-Depend. Filter type ☐ Enhanceable

E75 (1) 800 ec7server5 INS

Figure 1.10 Classic BAdI Definition Example

Listing 1.2 shows an excerpt from the SAP source code with an example of a classic BAdI instantiation and interface method call.

In the listing, the name of the BAdI definition is passed to the GET_INSTANCE method via parameter EXIT_NAME; in this example, it is EHS_PS_002. The resulting BAdI instance reference is passed to the variable L_BADI_INSTANCE via changing parameter INSTANCE. By means of parameter ACT_IMP_EXISTING, the method GET_INSTANCE returns the activation flag of the BAdI into the variable EHS_PS_002_ACTIVE.

Next to the GET_INSTANCE method call, there is a BAdI method ENTRY_INQUIERY call with application-specific parameters.

```
  CALL METHOD CL_EXITHANDLER=>GET_INSTANCE
    EXPORTING
        EXIT_NAME              = ‘EHS_PS_002’
        NULL_INSTANCE_ACCEPTED = ‘X’
    IMPORTING
        ACT_IMP_EXISTING = EHS_PS_002_ACTIVE
    CHANGING
      INSTANCE = L_BADI_INSTANCE
    EXCEPTIONS
      OTHERS   = 1.

* call the report info-system
    CALL METHOD L_BADI_INSTANCE->ENTRY_INQUIERY
      EXPORTING
        I_FLG_NEW_TASK   = EHS01_TRUE
        I_QMATNR_TAB     = L_MATNR_TAB[]
      EXCEPTIONS
        ILLEGAL_REPTYPE  = 1
        ILLEGAL_RVLID    = 2
        ILLEGAL_LANGU    = 3
        NO_MATERIALS     = 4
        NO_REPORTS_FOUND = 5
        INTERNAL_ERROR   = 6
        RFC_FAILED       = 7
        OTHERS           = 8.
```

Listing 1.2 Example of Classic BAdI Instantiation and Call

1.3.2 Kernel-Based BAdI

For kernel-based BAdIs, SAP delivers special aided ABAP statements for accessing the definition: Kernel-based BAdIs were introduced together with the new Enhancement Framework.

The new statement GET BADI is used to instantiate a kernel-based BAdI definition, and the CALL BADI statement is used to call the BAdI interface method.

Listing 1.3 shows an example of a kernel-based BAdI instantiation and call. See that the L_BADI variable declaration looks like an ordinary class reference declaration; and the BAdI name CUSTOMER_ADD_DATA is used as its type. After the instantiation with the statement GET BADI, the BAdI method READ_ADD_ON_DATA call follows.

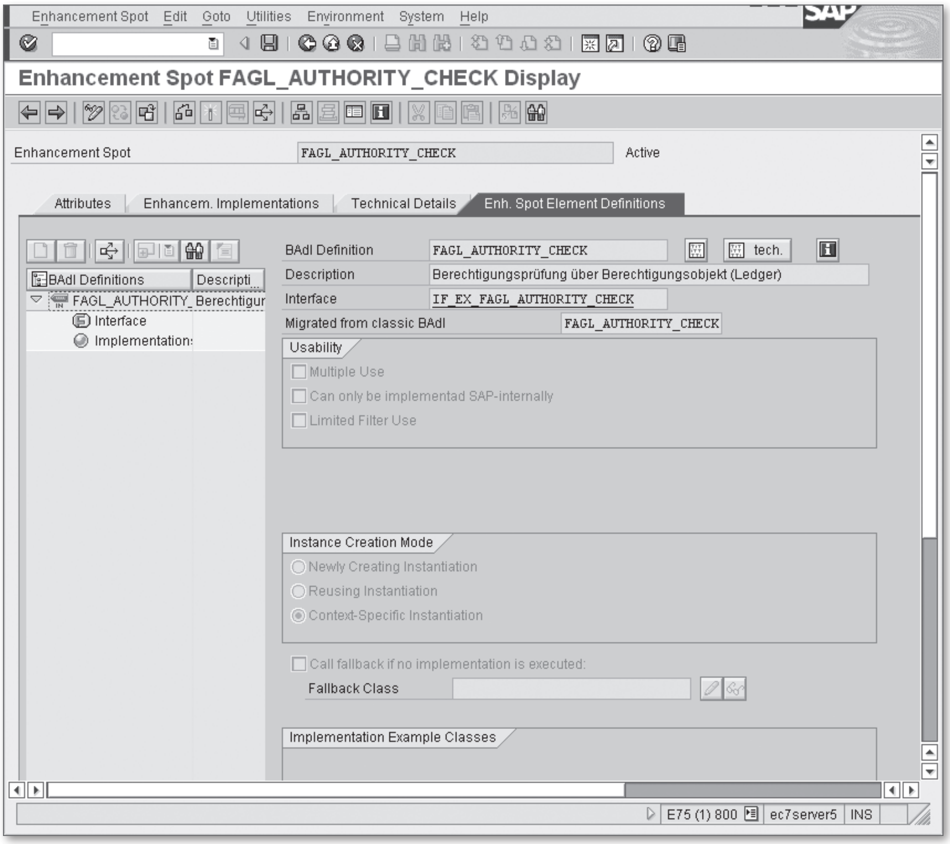
```
DATA l_badi TYPE REF TO CUSTOMER_ADD_DATA.

TRY.

  GET BADI l_badi
   CONTEXT me.

  CALL BADI l_badi->READ_ADD_ON_DATA
   EXPORTING
     I_KUNNR = I_KUNNR
     I_BUKRS = I_BUKRS
     I_VKORG = I_VKORG
     I_VTWEG = I_VTWEG
     I_SPART = I_SPART.



  CATCH CX_BADI.
ENDTRY.
```

Listing 1.3 Example of Kernel-Based BAdI Instantiation and Call

The BAdI definition name itself becomes a globally available reference type. This paradigm has an important advantage over the classic BAdI: The ABAP compiler can check the BAdI existence and parameters statically, so you can avoid runtime errors related to misspelling BAdI definition names or interface incompatibility.

Furthermore, unlike with the classic BAdI, the "where-used list" tool can be used directly for kernel-based BAdI definition.

See Figure 1.11 for an example view of a kernel-based BAdI in Transaction SE18.


Enhancement Spot Edit Goto Utilities Environment System Help
Attributes
Enhancem. Implementations
Technical Details
Enh. Spot Element Definitions
BAdI Definitions
Descripti
Interface
Implementation:
tech.
E75 (1) 800
ec7server5
INS

Enhancement Spot FAGL_AUTHORITY_CHECK Display

Enhancement Spot: FAGL_AUTHORITY_CHECK Active

FAGL_AUTHORITY_Berechtigur

BAdI Definition: FAGL_AUTHORITY_CHECK
Description: Berechtigungsprüfung über Berechtigungsobjekt (Ledger)
Interface: IF_EX_FAGL_AUTHORITY_CHECK
Migrated from classic BAdI: FAGL_AUTHORITY_CHECK

Usability
- [ ] Multiple Use
- [ ] Can only be implementad SAP-internally
- [ ] Limited Filter Use

Instance Creation Mode
- ○ Newly Creating Instantiation
- ○ Reusing Instantiation
- ● Context-Specific Instantiation

- [ ] Call fallback if no implementation is executed:

Fallback Class:

Implementation Example Classes

Figure 1.11 Kernel-Based BAdI Definition Example

Kernel-based BAdIs are an integral part of the Enhancement Framework. Each kernel-based BAdI is assigned to an enhancement spot.

1.3.3 Filtered BAdIs

Both the classic and kernel-based BAdI flavors can be filtered. A filter helps the calling application choose the appropriate BAdI implementation for the current data processing situation. For example, calculating tax is a highly country- or state-dependent matter; thus, if you define a BAdI for tax calculation, it's logical to set a country code as a BAdI filter. The calling program is responsible for supplying the appropriate filter value at the moment of the BAdI call. At runtime, the BAdI framework (either classic or kernel-based) selects only those active BAdI implementations that have matching filter values in their properties.

A classic BAdI definition has a simple filter facility: A BAdI definition can be assigned a global data element as a filter type. When implementing a filtered BAdI, you must provide one or more filter values for which your implementation will be active.

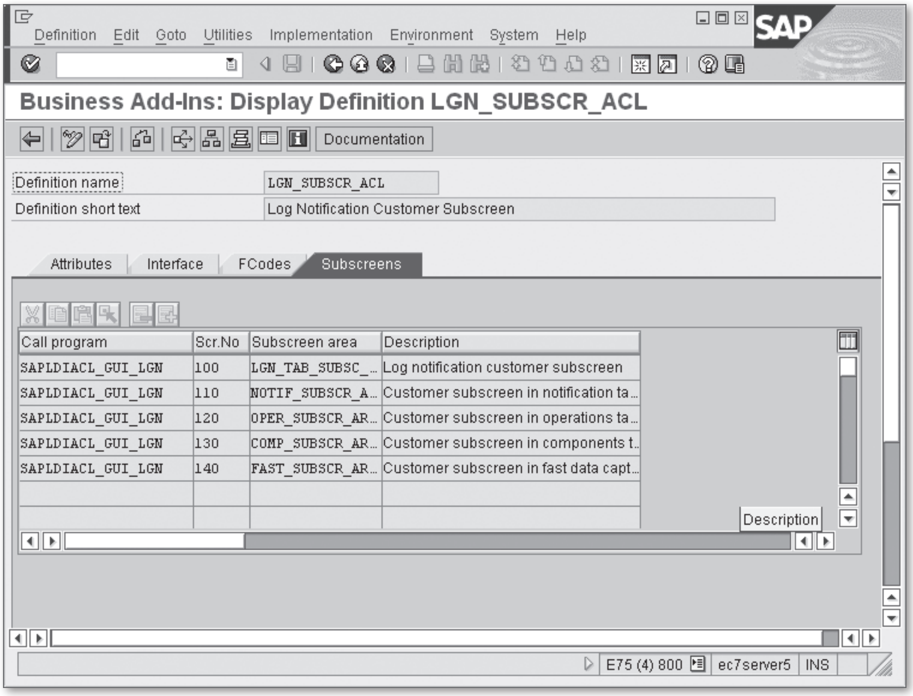
A kernel-based BAdI has a much more sophisticated filter definition technique: The filter value can be checked against data element fixed values (as with the classic BAdI); alternatively, you can define a special program for checking the filter value at runtime. When creating an implementation for a kernel-based filtered BAdI, you can create considerably complex filter conditions with several conditions connected with `AND` and `OR` logical operators.

1.3.4 BAdI Subscreen and Function Codes

Both kinds of BAdIs are also capable of screen and menu function code extending. If the BAdI is capable of extending the screen and menu codes, it will have an additional tab with available subscreen areas (see Figure 1.12). When creating the BAdI implementation, you will have to set your own program name and subscreen number for each available subscreen area.

For menu function code, the implementation will contain text labels and an icon for redefined function code. Subscreen and function code extensions actually have nothing to do with the object-oriented paradigm. In this case, the BAdI definition and implementation are just used as a placeholder for screen numbers and function codes. The calling program is responsible for properly processing the provided subscreens and defining a mechanism for processing additional subscreens and menu codes.

Later in this book, you'll see an example of a BAdI with a subscreen definition.



Business Add-Ins: Display Definition LGN_SUBSCR_ACL

Definition name	LGN_SUBSCR_ACL
Definition short text	Log Notification Customer Subscreen

Call program	Scr.No	Subscreen area	Description
SAPLDIACL_GUI_LGN	100	LGN_TAB_SUBSC_...	Log notification customer subscreen
SAPLDIACL_GUI_LGN	110	NOTIF_SUBSCR_A...	Customer subscreen in notification ta...
SAPLDIACL_GUI_LGN	120	OPER_SUBSCR_AR...	Customer subscreen in operations ta...
SAPLDIACL_GUI_LGN	130	COMP_SUBSCR_AR...	Customer subscreen in components t...
SAPLDIACL_GUI_LGN	140	FAST_SUBSCR_AR...	Customer subscreen in fast data capt...

Figure 1.12 Classic BAdI with Screen and Menu Enhancements

1.3.5 Finding BAdIs

As with other enhancement tools, the Workbench Information System is always available as a passive search method. The active method is using generic breakpoints: Defining breakpoints at the `CL_EXITHANDLER=>GET_INSTANCE` method can find a classic BAdI, or defining a breakpoint at a `GET BADI` statement can find a kernel-based BAdI.

1.3.6 BAdI Summary

A BAdI is the most flexible enhancement tool; compared to the BTE framework, the latest kernel-based BAdIs have sophisticated filter mechanisms. The object-oriented nature of the BAdI can potentially lead to more elegant decomposition when functionally close application callbacks can be united into one BAdI definition. At the same time, a BAdI isn't capable of remote functionality (unlike BTE framework); it's obvious, however, that implementing a remote BAdI (which is the same as remote object communication) can be a considerably challenging task.

1.4 Implicit Enhancements

SAP introduced the *implicit enhancement* concept in SAP NetWeaver 7.0 together with the new Enhancement Framework. Using this option, you can modify virtually any standard source code without requesting a modification key. However, there are some restrictions. For example, you can't enhance dynpro screens and system programs (those with PROGRAM STATUS = "S"). Additionally, an implicit enhancement can't be implanted into an arbitrary source code place; they are allowed only at the beginning or at the end of a programming module (function, subroutine, or class method), class declaration sections, structure declarations, and some other places. Despite these restrictions, you can almost completely redefine the logic of virtually any given standard function.

Implicit enhancements should be the last resort for developers when all other options are completely insufficient. Experienced users sometimes say that if it seems that you need a source code modification (or implicit enhancement), this is possibly due to lack of knowledge. On the other hand, when you invade a standard source code, you take over the original manufacturer responsibility. You should always think twice before altering standard code and then reconsider your options.

1.5 Summary

In this chapter, we considered the main enhancement techniques that SAP currently delivers with its systems. We also covered the history of enhancements from rude modifications to the sophisticated modern Enhancement Framework. This chapter gives you a good start before plunging into the details of Financial Accounting enhancements. In the next chapter, we examine both the data representation and user interface enhancement methods of accounting master data.

In this chapter, we examine possible ways of enhancing both the database and user interface data of different types of Financial Accounting master data: general ledger, Accounts Receivable, and Accounts Payable.

2 Master Data Enhancements

The various master data enhancements in Financial Accounting are not always simple to implement and thus require detailed explanation. As for customer and vendor master data, these enhancements have several common features worth discussing together, and, at the same time, they also have some specifics that should be shown separately. First, we'll walk through the general ledger account master data enhancement. Then, we'll discuss common features of vendor and customer data enhancements, followed by a separate consideration of customer credit control data enhancements.

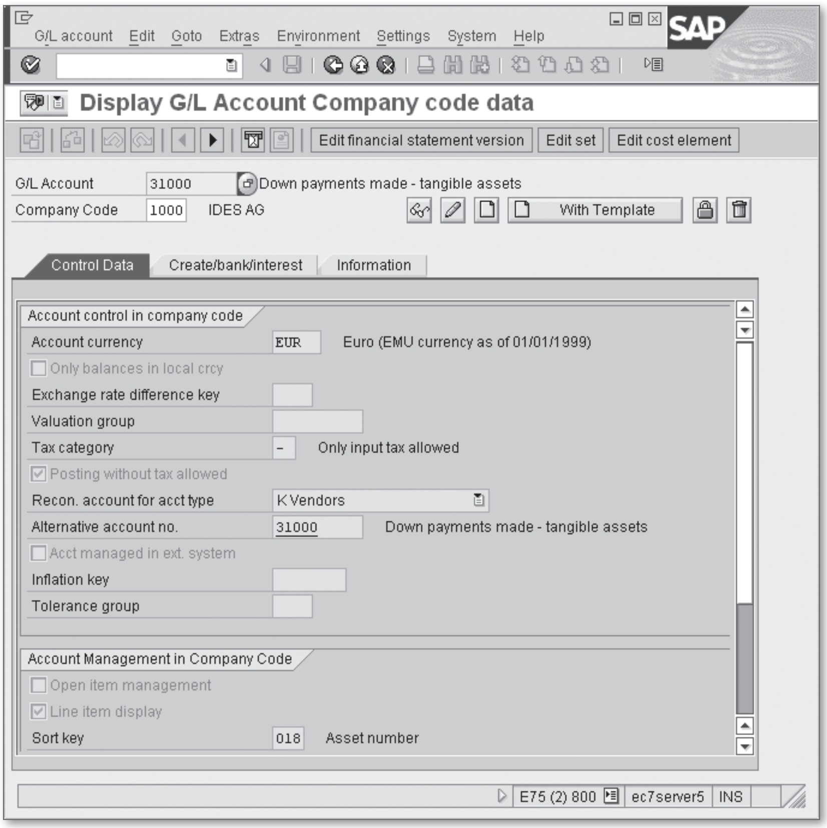
2.1 General Ledger Accounts

The task of the general ledger account master data enhancement is not common in SAP implementations. In a way, it depends on the informal business influence of an accounting department. The higher that influence is, the more enhancements are required in accounting, including general ledger account master data.

It isn't a problem to add additional fields to the general ledger account table via the append structure and then to create some customer-specific Z transaction for manipulating those data fields. A more interesting task is enhancing not only a database table but also the user interface (UI) and screen layout. It requires some tricky coding, though, and a considerable amount of time to debug standard SAP transactions. In this section, we discuss three elements of general ledger accounts: their main transaction codes, data enhancements of general ledger account master data tables, and data enhancements of UIs.

2.1.1 Main Transaction Codes for General Ledger Account Master Data

First, let's examine the structure of the main transaction codes for general ledger account master data maintenance. These codes are FS00, FSP0, and FSS0. The screen layout of all three transactions actually looks almost the same. They include a header block with the account number and header information and a tabstrip control below. You can see this in Figure 2.1.



Display G/L Account Company code data

G/L Account	31000	Down payments made - tangible assets
Company Code	1000	IDES AG

Account control in company code

Account currency	EUR	Euro (EMU currency as of 01/01/1999)
☐ Only balances in local crcy		
Exchange rate difference key		
Valuation group		
Tax category	-	Only input tax allowed
☑ Posting without tax allowed		
Recon. account for acct type	K Vendors	
Alternative account no.	31000	Down payments made - tangible assets
☐ Acct managed in ext. system		
Inflation key		
Tolerance group		

Account Management in Company Code

☐ Open item management		
☑ Line item display		
Sort key	018	Asset number

E75 (2) 800 | ec7server5 | INS

Figure 2.1 Screen Layout of Transaction FS00

All three transactions call report SAPGL_ACCOUNT_MASTER_START with a short source code. Following the program logic, you can see that the processing of general ledger account master data takes place inside function module `GL_ACCT_MASTER_MAINTAIN`.

Note

For compatibility reasons, SAP retains old transaction codes for general ledger master record maintenance (FS01/FS02/FS03). They reference the `SAPMF02H` module pool, which is a very old piece of code. If you open the starting screen's layout of these transactions in the Dialog Screen Editor, you can see that those layouts have nothing in common with what you see in the opened transaction. The secret is that SAP actually calls new transaction codes. For example, you can follow the logic by opening starting screen 401 of Transaction FS02, then opening PBO module `TRANSAKTION_INIT`, and finally opening subroutine `NEW_TRANSACTIONS`.

As a reminder, you want to seamlessly integrate your own logic and screen elements into an existing SAP transaction without modifying the standard and without any harm to the standard logic. As you can see, the modern general ledger account maintaining transactions use a tabstrip control to display different views of account data. To tailor the tab to your personal needs, you must add an additional tab with your proprietary data.

Toward the end of the source code of function module `GL_ACCT_MASTER_MAINTAIN`, you can see a subroutine call with quite a promising name: `SET_LAYOUT`. Notice that there is also another subroutine: `SET_LAYOUT_FROM_ACCOUNT`, but it's actually a wrapper for the first one. This is where you find some useful information. Inside the subroutine, you can see the call to the `TABSTRIP_INIT` function module, which implements an SAP customizable tabstrip control manipulating technique.

Note

Usage of function modules from the `ATAB` function group (such as `TABSTRIP_INIT` or `TABSTRIP_LAYOUT_READ`) is an indication that the UI of a transaction might be enhanced. Later, you'll see that this technique is used in areas other than general ledger account maintenance transactions.

2.1.2 Data Enhancement of General Ledger Account Master Data Tables

The two main database tables storing general ledger account information are SKA1 and SKB1. (Other general ledger account tables are not relevant to our discussion in this section.) SKA1 contains general account data common to all company codes, while SKB1 contains data specific for a particular company code and consequently has company code as a part of the primary key. You might also notice that each table has chart of accounts code (field KTOPL) as a primary key component, which means that an account is always a part of some chart of accounts. A chart of accounts is just a list of accounts, specific for some country and/or company code. Following the foreign key link, you can see that a chart of accounts has its own table (T004). In the following explanation, you will see that the chart of accounts has its impact on the UI of general ledger account master record maintenance transactions.

Let's suppose that you need to add some field to the SKB1 table, so you enhance the data specific to the company code. To make the task more expressive, let's also suppose that the field should reference some other customizing table and should be exposed to a user in the form of a list box element.

The next section describes the steps in this process.

Creating Domain and Data Elements

Note

Never be too rushed to create data elements for your fields with appropriate field label texts. This practice makes the design more flexible and maintainable — if you use the same field in ALV reports or other screens, you don't have to rewrite field labels for corresponding interface elements.

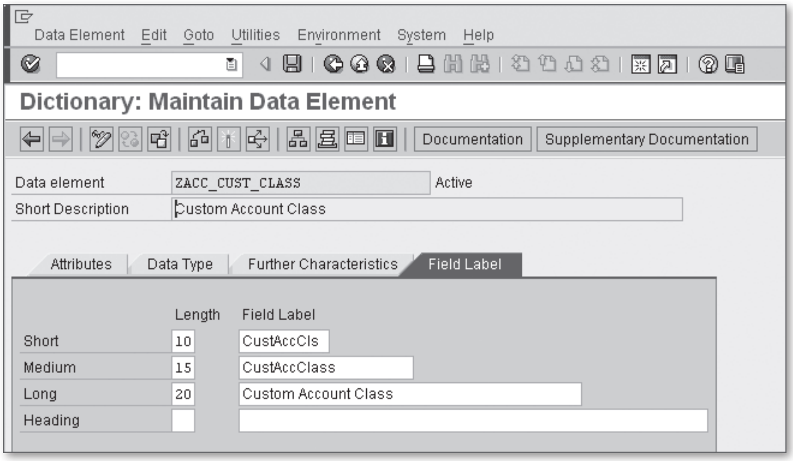
To plan the field to reference another table, you create a domain for the field because the domain contains information of its value table. It's a good practice to give the same name to the domain and data element. In the example, the domain's name is ZACC_CUST_CLASS. The description of the field is "Custom Account Class."

Note

As an alternative to a value table, you can use a list of predefined domain values. This method is less flexible, so it should be used when you have a very stable list of values.

Now let's create the data element in Transaction SE11.

1. Enter "Custom Account Class" into the Short Description input field.
2. Enter "ZACC_CUST_CLASS" into the Domain field on the Data Type tab.
3. Open the Field Label tab, and fill in all labels as shown in Figure 2.2.

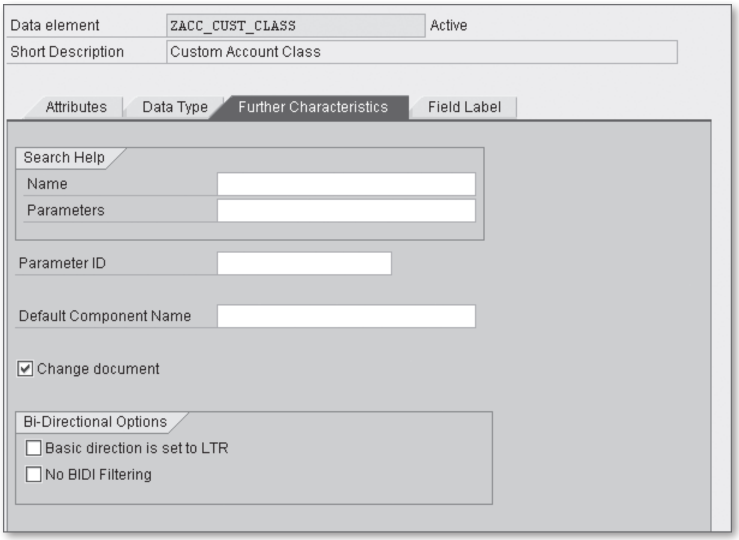

Data Element Edit Goto Utilities Environment System Help
Dictionary: Maintain Data Element
Documentation
Supplementary Documentation
Attributes
Data Type
Further Characteristics
Field Label

Data element	ZACC_CUST_CLASS	Active
Short Description	Custom Account Class	

	Length	Field Label
Short	10	CustAccCls
Medium	15	CustAccClass
Long	20	Custom Account Class
Heading		

Figure 2.2 Field Labels of the ZACC_CUST_CLASS Data Element

4. To enable change logging for the new field, check the box for CHANGE DOCUMENT IN THE FURTHER CHARACTERISTICS tab of the data element (see Figure 2.3).



Data element	ZACC_CUST_CLASS	Active
Short Description	Custom Account Class	

Search Help
Name:
Parameters:

Parameter ID:

Default Component Name:

☑ Change document

Bi-Directional Options
☐ Basic direction is set to LTR
☐ No BIDI Filtering

Figure 2.3 Turn on Change Document Flag for ZACC_CUST_CLASS

5. Reopen the DATA TYPE tab, and double-click on the DOMAIN field; the system takes you into the DOMAIN CREATION dialog.

6. In this screen, fill in the Short description field (it should remain the same as for the data element), enter "CHAR" in the Data Type field, and enter an appropriate length for the field into the No. Characters input field (1 is an excellent choice; just remember that you can express fewer than 100 different values with one character field).
7. Open the Value Range tab, and enter "ZTACC_CUST_CLASS" into the Value Table field at the bottom of the screen.
8. Double-click the Value Table field to open the Database table creation dialog.
9. Fill in the Short description field (use the same text as for data element and domain).
10. Choose a delivery class. For the example, the delivery class A (Application table—master and transaction data) is suitable enough.

Note

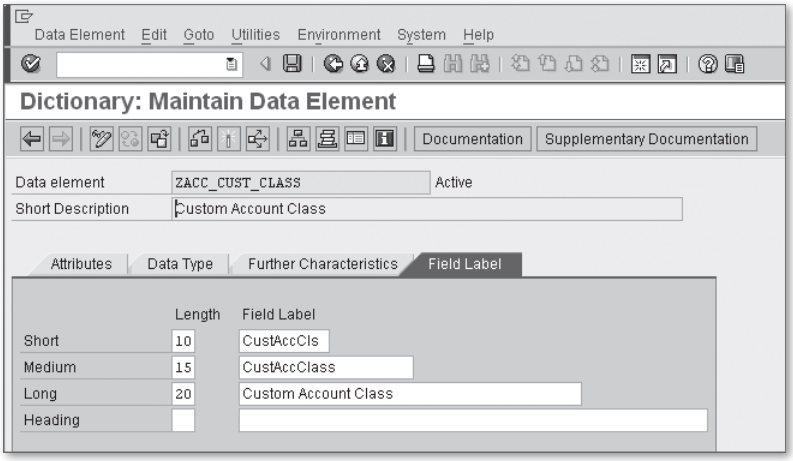
If you plan to use a table as a part of customizing, you should choose Delivery Class C—Customizing table, maintenance only by cust., not SAP import. In this case, the standard maintenance dialog for that table asks you for a transport request number every time you make changes to the table entries.

For the domain value table, its primary key must contain only one field referencing the domain (besides the client number field: MANDT). The field list should look as shown on Figure 2.4. Note that you do not include any text description field in the table.

The next mandatory step in the process of creating a table is maintaining the technical settings, which are accessible via the Technical settings toolbar button.

Here, you have to fill in two fields: Data class and Size Category. For the example, we chose Data class APPL2 and Size Category 0.

Now you're ready to activate the newly created objects: domain, data element, and value table. Click the Activate button (). The pop-up dialog window with all three inactive objects appears. Select them all, and press Enter.


Data Element Edit Goto Utilities Environment System Help
Dictionary: Maintain Data Element
Documentation
Supplementary Documentation
Attributes
Data Type
Further Characteristics
Field Label

Data element: ZACC_CUST_CLASS Active
Short Description: Custom Account Class

	Length	Field Label
Short	10	CustAccCls
Medium	15	CustAccClass
Long	20	Custom Account Class
Heading		

Figure 2.4 Field List for Table ZTACC_CUST_CLASS

Creating Text Tables for Domain Values

To make the interaction between the user and the interface easier, besides the code values of the new field, you should also provide a text description of each value by using a text table. The primary key of a text table must include a language key. The language key is just a field referencing language table T002 as a value table via domain definition. To follow the tradition, you can use existing data element `SPRAS` for the language key field.

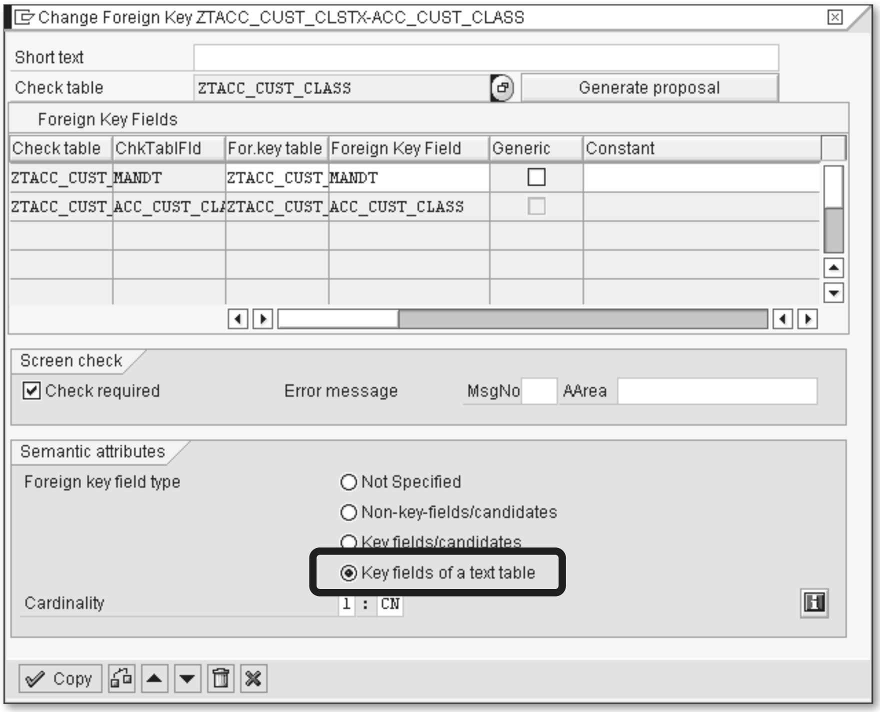
You create the text table with the name `ZTACC_CUST_CLSTX`, choose DELIVERY CLASS A for the value table, and use "Custom Account Class" as a short description of the table. You also set the same technical settings for the table with DATA CLASS APPL2 and SIZE CATEGORY 0.

For a description text field, you also can use existing data elements such as `STEXT`, `LTEXT`, or many others starting with `BEZEI`. For example, `BEZEI20` would be great, with length 20 and label "Description".

Now you have to define the relation between the previously created value table and the text table. To do this, you create a foreign key for the `ACC_CUST_CLASS` field by clicking the foreign keys button (), while the cursor is placed over the `ACC_CUST_CLASS` field.

When you define `ZTACC_CUST_CLASS` as a value table for the domain `ZACC_CUST_CLASS`, the system asks if it should propose the default values for the foreign key.

Answer "Yes." Now you will see the dialog window represented in Figure 2.5. To make the table a text table, click the circle next to KEY FIELDS OF A TEXT TABLE in the FOREIGN KEY FIELD group.


Change Foreign Key ZTACC_CUST_CLSTX-ACC_CUST_CLASS

Short text:
Check table: ZTACC_CUST_CLASS

Foreign Key Fields

Check table	ChkTablFld	For.key table	Foreign Key Field	Generic	Constant
ZTACC_CUST	MANDT	ZTACC_CUST	MANDT	☐	
ZTACC_CUST	ACC_CUST_CL	ZTACC_CUST	ACC_CUST_CLASS	☐	

Screen check
☑ Check required   Error message   MsgNo   AArea

Semantic attributes
Foreign key field type: ○ Not Specified ○ Non-key-fields/candidates ○ Key fields/candidates [✓ Key fields of a text table]
Cardinality: 1 : CN

Figure 2.5 Defining the Text Table Foreign Key

Note

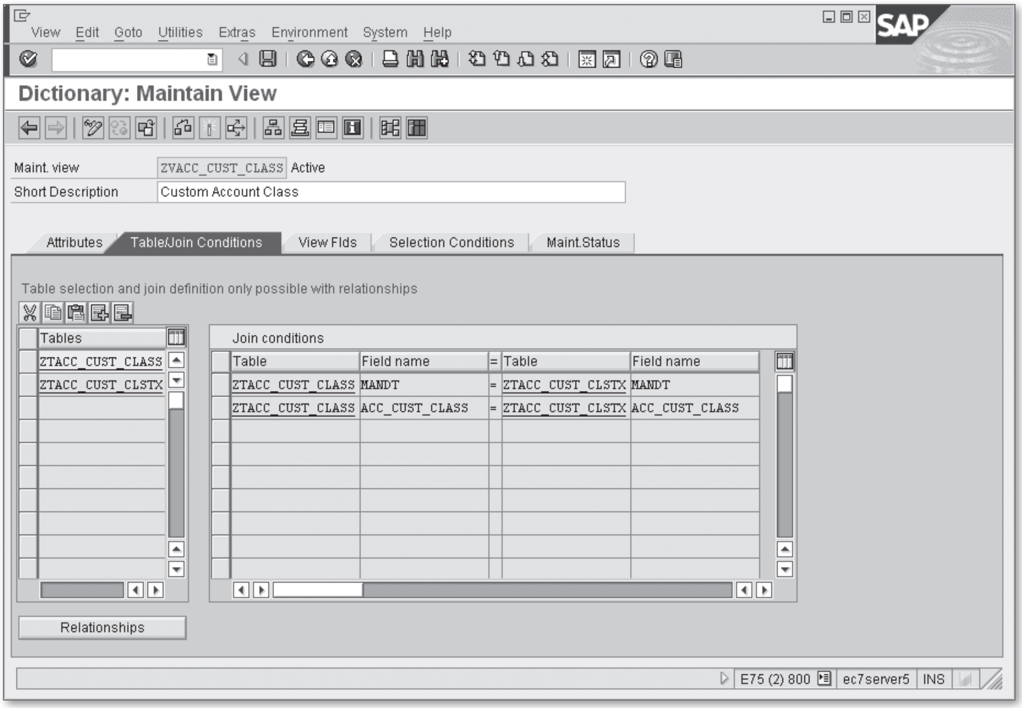
It's a good practice to define the foreign key for all fields that reference the domain with the value table because, in many cases, it helps when designing dialog screens and UI elements. In our case, we also create a foreign key for the field SPRAS that accepts all default values provided by the system.

Creating a Maintenance View for Domain Values

To make maintaining the table entries possible via Transaction SM30, you have to create a maintenance view for your two tables. Its name will be ZVACC_CUST_CLASS.

You create the maintenance view in Transaction SE11. By default, at the beginning of this procedure, the TABLE/JOIN CONDITIONS tab is open.

1. Enter the name of the view into VIEW FIELD, and click on the corresponding button.
2. After clicking the CREATE button, the system asks for the view category. Select MAINTENANCE VIEW.
3. Enter the short description of the view.
4. Enter the main table name of the view: "ZTACC_CUST_CLASS".
5. Click the RELATIONS button, and select the only available table ZTACC_CUST_CLSTX. It is available here because we previously created the foreign key relation between the two tables. Now the TABLE/JOIN CONDITIONS tab should look like Figure 2.6.



Maint. view ZVACC_CUST_CLASS Active
Short Description Custom Account Class

Table selection and join definition only possible with relationships

Tables
ZTACC_CUST_CLASS
ZTACC_CUST_CLSTX

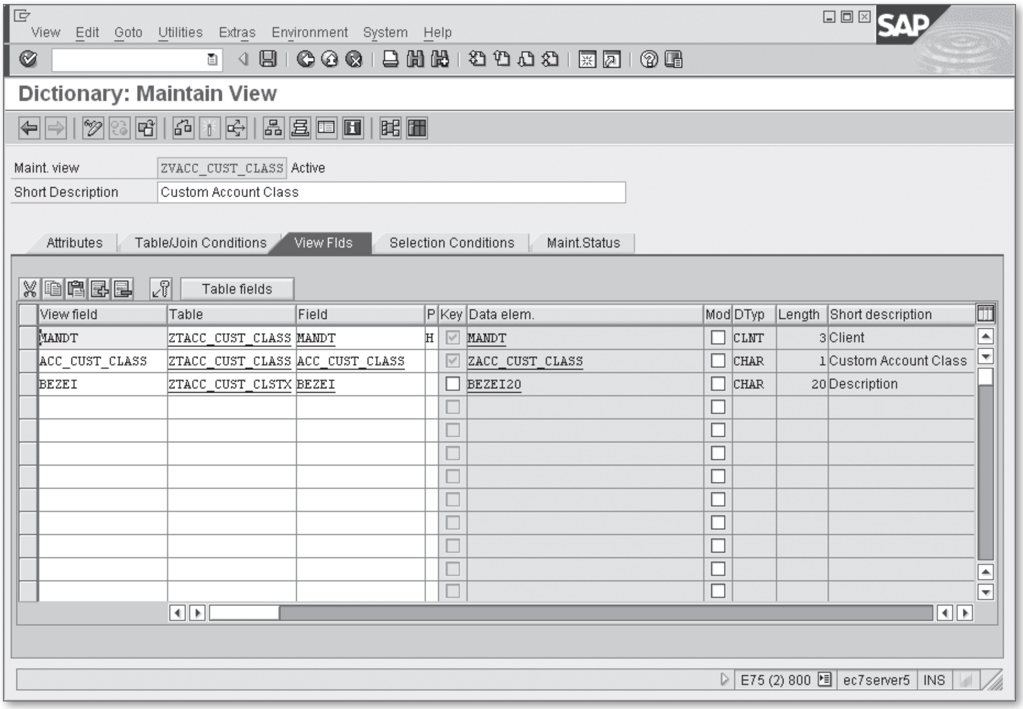
Join conditions

Table	Field name	=	Table	Field name
ZTACC_CUST_CLASS	MANDT	=	ZTACC_CUST_CLSTX	MANDT
ZTACC_CUST_CLASS	ACC_CUST_CLASS	=	ZTACC_CUST_CLSTX	ACC_CUST_CLASS

Figure 2.6 Join Conditions of ZVACC_CUST_CLASS Maintenance View

Next, you need to define the view fields via the View Flds tab. By default, the system shows all of the key fields of both tables. (Note that although it is a component of the primary key, there is no SPRAS field here because it is a language key, and the relation between the two tables is the text table relation.)

1. Click the Table fields button, and select the table ZTACC_CUST_CLSTX in the pop-up window.
2. In the next window, click the box next to the BEZEI field. The View Flds tab now should look as shown in Figure 2.7.



Dictionary: Maintain View

Maint. view ZVACC_CUST_CLASS Active
Short Description Custom Account Class

View field	Table	Field	P	Key	Data elem.	Mod	DTyp	Length	Short description
MANDT	ZTACC_CUST_CLASS	MANDT	H	✓	MANDT	☐	CLNT	3	Client
ACC_CUST_CLASS	ZTACC_CUST_CLASS	ACC_CUST_CLASS		✓	ZACC_CUST_CLASS	☐	CHAR	1	Custom Account Class
BEZEI	ZTACC_CUST_CLSTX	BEZEI		☐	BEZEI20	☐	CHAR	20	Description

Figure 2.7 Fields of the ZVACC_CUST_CLASS Maintenance View

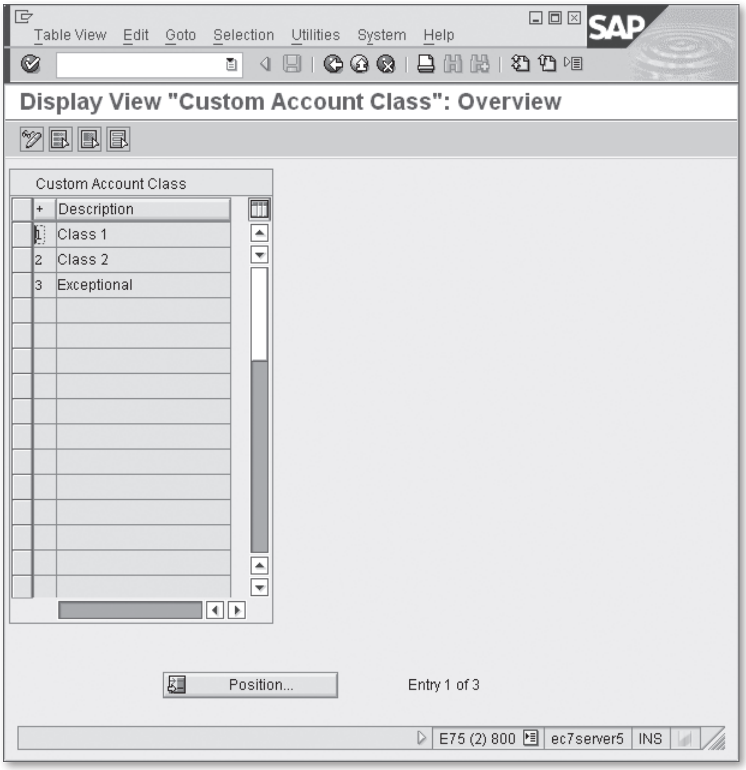
3. To finalize the view definition, create a table maintenance dialog by choosing Utilities • Table Maintenance Generator.
4. In the next screen, enter "&NC&" in the Authorization Group field and enter "ZVACC_CUST_CLASS" into the function group.

5. Click the one-step radio button in the Maintenance Type group, and enter "100" in the Overview screen field.
6. Click the CREATE button. After the Maintenance dialog appears, you can maintain the view in Transaction SM30.

Maintaining Domain Values in Transaction SM30

To maintain domain values in Transaction SM30, follow these steps:

1. In Transaction SM30, enter "ZVACC_CUST_CLASS" into the TABLE/VIEW field, and click the MAINTAIN button.
2. Click the NEW ENTRIES button, and fill in the table with arbitrary values for the key and description. An example is shown in Figure 2.8.



Display View "Custom Account Class": Overview

Custom Account Class

+	Description
1	Class 1
2	Class 2
3	Exceptional

Entry 1 of 3

Figure 2.8 ZVACC_CUST_CLASS View Contents

Now you are ready to enhance the structure of the SAP tables for the general ledger account master record.

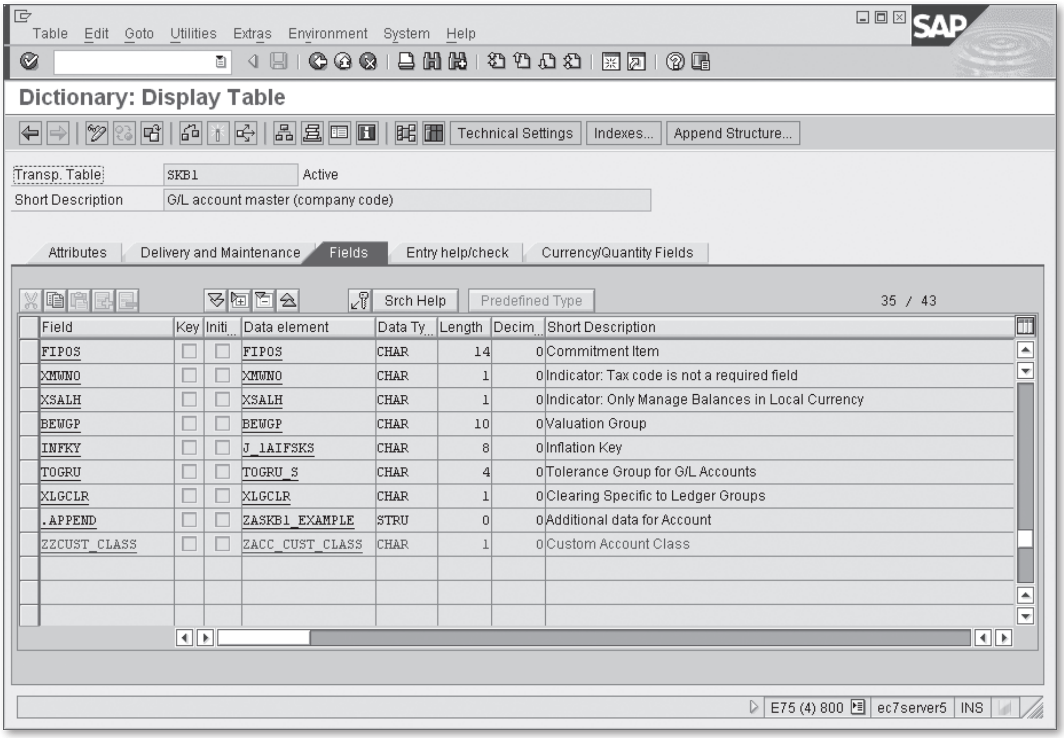
Enhancing the Database Table Structure

To enhance the company code specific data, you have to expand the SKB1 table. If you are familiar with the data enhancement techniques that SAP uses, you might know that SAP often provides customer include structures with names that start with CI_. In the case of Table SKB1, there is no such inclusion, so we'll use an append structure instead.

1. Open Table SKB1 in Transaction SE11 in display mode.
2. Click the APPEND STRUCTURES button. If there is no append structure for SKB1 in your system, an information message appears, and then the system asks for an append structure name. To comply with SAP naming conventions, name the structure starting with ZA (or YA), although you can use any name from the customer namespace. For the example, use the name is ZASKB1_EXAMPLE.
3. After entering this name, you are taken into the common structure definition window. Here you define the only field with the name ZZCUST_CLASS. You should start all additional field names with ZZ to avoid possible conflicts with SAP fields.
4. Don't forget to activate the structure. After adding the append structure, table SKB1 should look like Figure 2.9.

Enhancing the Auxiliary Structure

Unfortunately, enhancing only the SKB1 database table is not enough for the experiment because general ledger account transactions do not operate directly with SKA1 or SKB1 tables. Instead, they use intermediate in-memory structures serving as a runtime data container. The structure of the container is declared by type `ACCOUNT` in function group `GL_ACCOUNT_MASTER_MAINTAIN`. Looking into the where-used list of the type, you can see two variables of the type: `AC_NEW` and `AC_OLD`. `AC_NEW` is used for storing changed data of an edited account, and `AC_OLD` keeps track of old values.



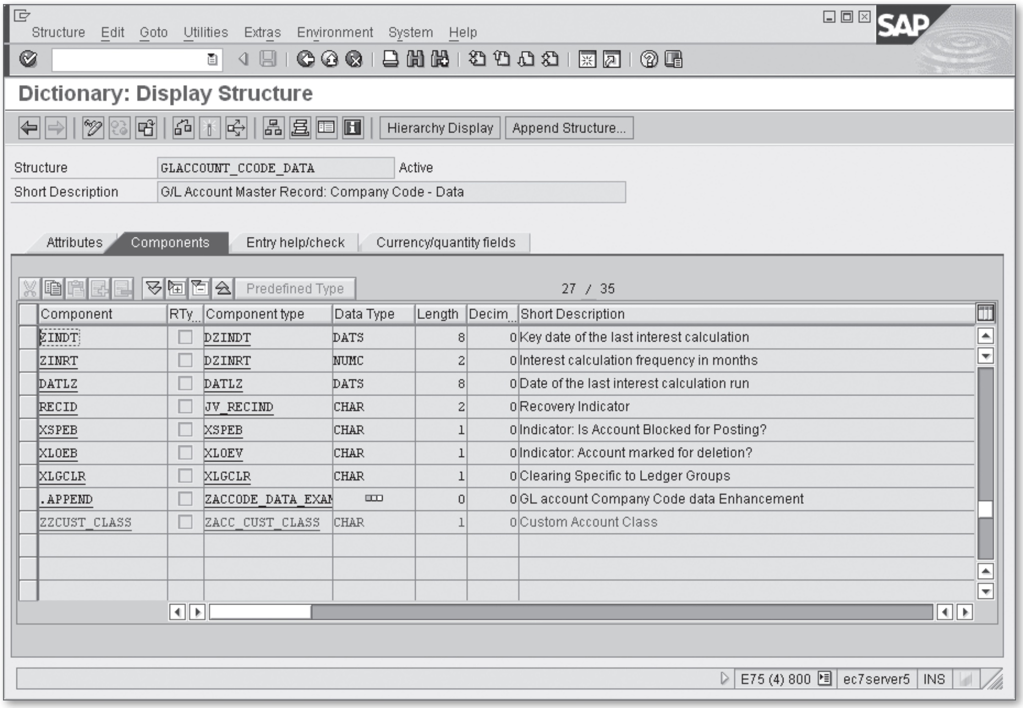
Transp. Table	SKB1 Active
Short Description	G/L account master (company code)

Field	Key	Initi...	Data element	Data Ty...	Length	Decim...	Short Description
FIPOS	☐	☐	FIPOS	CHAR	14	0	Commitment Item
XMWNO	☐	☐	XMWNO	CHAR	1	0	Indicator: Tax code is not a required field
XSALH	☐	☐	XSALH	CHAR	1	0	Indicator: Only Manage Balances in Local Currency
BEWGP	☐	☐	BEWGP	CHAR	10	0	Valuation Group
INFKY	☐	☐	J_1AIFSKS	CHAR	8	0	Inflation Key
TOGRU	☐	☐	TOGRU_S	CHAR	4	0	Tolerance Group for G/L Accounts
XLGCLR	☐	☐	XLGCLR	CHAR	1	0	Clearing Specific to Ledger Groups
.APPEND	☐	☐	ZASKB1_EXAMPLE	STRU	0	0	Additional data for Account
ZZCUST_CLASS	☐	☐	ZACC_CUST_CLASS	CHAR	1	0	Custom Account Class

Figure 2.9 Table SKB1 Additional Fields

Another element that uses the where-used list shows the company code runtime data of the account stored in the `CCODE_DATA` component of the `ACCOUNT` type. The type of this component is a global dictionary structure `GLACCOUNT_CCODE_DATA`. Looking into the source code of subroutine `ACCOUNT_CHECK_AND_SAVE`, you can see that it calls the `GL_ACCT_MASTER_SAVE` function module, which actually updates all of the database tables of the general ledger account master record. SAP uses the `MOVE-CORRESPONDING` statement to move data from the `GLACCOUNT_CCODE_DATA` structure into the SKB1 work area. Thus, if we enhance the `GLACCOUNT_CCODE_DATA` structure with fields that have the same names as the enhancement of table SKB1, we can successfully store the proprietary information in table SKB1.

After you've finished creating an append structure for database table SKB1, you can enhance the `GLACCOUNT_CCODE_DATA` dictionary structure with an additional field: `ZZCUST_CLASS` (see Figure 2.10).



Dictionary: Display Structure

Structure	GLACCOUNT_CCODE_DATA	Active
Short Description	G/L Account Master Record: Company Code - Data	

27 / 35

Component	RTy	Component type	Data Type	Length	Decim	Short Description
ZINDT	☐	DZINDT	DATS	8	0	Key date of the last interest calculation
ZINRT	☐	DZINRT	NUMC	2	0	Interest calculation frequency in months
DATLZ	☐	DATLZ	DATS	8	0	Date of the last interest calculation run
RECID	☐	JV_RECIND	CHAR	2	0	Recovery Indicator
XSPEB	☐	XSPEB	CHAR	1	0	Indicator: Is Account Blocked for Posting?
XLOEB	☐	XLOEV	CHAR	1	0	Indicator: Account marked for deletion?
XLGCLR	☐	XLGCLR	CHAR	1	0	Clearing Specific to Ledger Groups
.APPEND	☐	ZACCODE_DATA_EXAM		0	0	GL account Company Code data Enhancement
ZZCUST_CLASS	☐	ZACC_CUST_CLASS	CHAR	1	0	Custom Account Class

E75 (4) 800 | ec7server5 | INS

Figure 2.10 GLACCOUNT_CCODE_DATA Additional Field

2.1.3 Screen Layout Enhancement

Now that we've created all of the necessary data enhancements, it's time to enhance the UI of the general ledger account maintenance transaction.

In the next steps, we'll create a subscreen, configure the layout of the SAP transaction, and develop the appropriate screen flow logic.

Creating the Subscreen

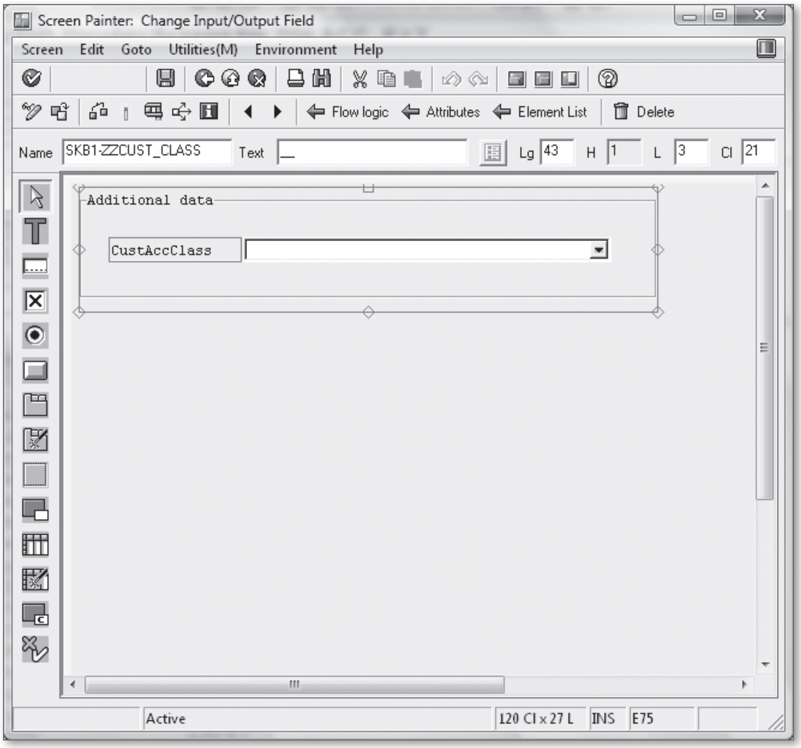
Because we don't plan to implement any sophisticated logic, the subscreen will be simply a frame with a caption and a list box.

1. Open Transaction SE80, and create a program of type "module pool" with name `ZGLACC_EXT`. The only line of code you have to add is the `TABLES` definition. The whole source code of the module pool is shown in Listing 2.1.

```
PROGRAM  zglacc_ext.

TABLES: skb1.
```

Listing 2.1 Module Pool Source Code

2. Create a screen with number 0100 inside module pool `ZGLACC_EXT`. Enter the screen description into the SHORT DESCRIPTION field, and click the SUBSCREEN button.
3. Open the graphical screen layout by clicking on the LAYOUT toolbar button.
4. Add a frame with an arbitrary caption, and create a text input field (within the frame), referencing the `SKB1-ZZCUST_CLASS` table field (see Figure 2.11). To turn the frame into a list box, double-click it, and then choose the LISTBOX entry in the Dropdown list box as shown in Figure 2.12. Set the visible length of the field to 43. After these manipulations, the layout of the screen should look just like Figure 2.12. At this step, we do not need any flow logic for the screen.


Screen Painter: Change Input/Output Field
Screen Edit Goto Utilities(M) Environment Help
Flow logic
Attributes
Element List
Delete

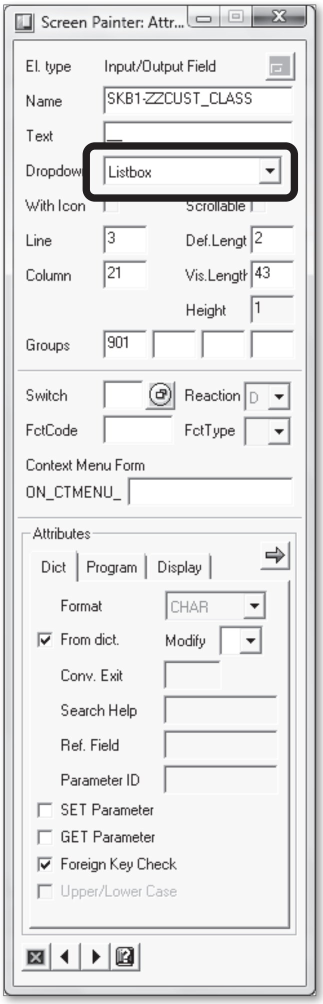
Name SKB1-ZZCUST_CLASS Text __ Lg 43 H 1 L 3 Cl 21

Additional data

CustAccClass

Active 120 Cl x 27 L INS E75

Figure 2.11 Screen Field Properties



El. type	Input/Output Field		
Name	SKB1-ZZCUST_CLASS		
Text	__		
Dropdow	Listbox		
With Icon	☐	Scrollable	☐
Line	3	Def.Lengt	2
Column	21	Vis.Lengtł	43
		Height	1
Groups	901		
Switch		Reaction	D
FctCode		FctType	
Context Menu Form			
ON_CTMENU_			

Attributes

Dict | Program | Display

Format	CHAR		
☑ From dict.		Modify	
Conv. Exit			
Search Help			
Ref. Field			
Parameter ID			
☐ SET Parameter			
☐ GET Parameter			
☑ Foreign Key Check			
☐ Upper/Lower Case			

Figure 2.12 Screen Field Attributes for the SKB1-ZZCUST_CLASS Field

5. Save your work, and activate both the screen and the module pool. Test the screen separately. If you've thoroughly defined all of the necessary attributes of the field, data element, and domain as explained earlier, you'll see a list box with a selectable entry list. The test screen should look like Figure 2.13.

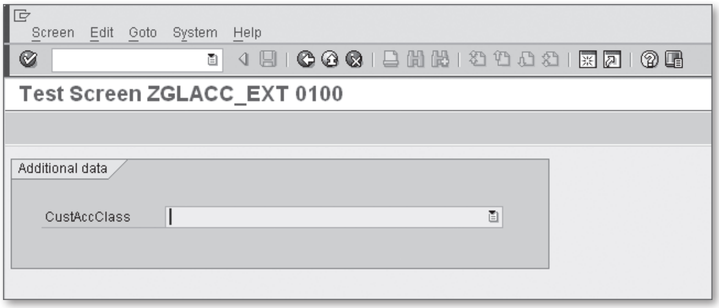


Figure 2.13 Separate Screen Test Result

Creating a Custom Tabstrip Layout

Now we need to implant the subscreen into the screen layout of general ledger account maintaining transactions. To do this, we must thoroughly investigate the source code of various functions of function group ATAB. This function group is an implementation of SAP internal dynamic tabstrip control, used in various master data maintenance transactions. The key function module that contains the necessary tables is TABSTRIP_LAYOUT_READ.

Note

ATAB's short description reads "Tabs pages in master data," which means that it's a reusable technology; in other words, after you enhance one transaction, you can do the same with any other transaction that uses the same technique. Additionally, you can use it in your own transactions.

You can see that the function reads data from various tables with names starting with TAMLAY: TAMLAYA, TAMLAYB, TAMLAY1, and TAMLAY2. Using the FIND MAINTENANCE DIALOG button in Transaction SM30, you see that these tables can be maintained via two view clusters: VC_TAMLAYA_00 and VC_TAMLAY_00.

Registering Subscreens

Now let's register the subscreen in layout configuration. The first view cluster VC_TAMLAYA_00 contains a list of subscreens used in different application areas. The application area is a key field. Recall the function module TABSTRIP_INIT in the general ledger account maintenance transaction, which has an input parameter I_APPL with an actual value GL_MASTER. The list of subscreens of application area GL_MASTER is shown in Figure 2.14.

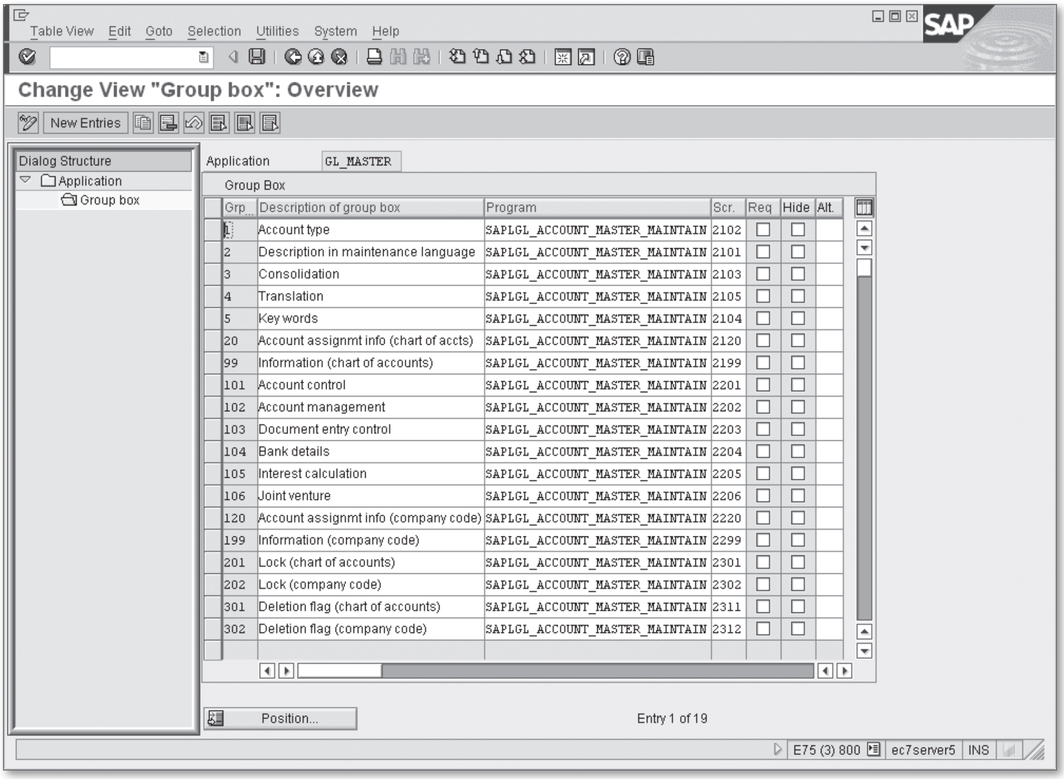


Table View Edit Goto Selection Utilities System Help
Change View "Group box": Overview
New Entries
Dialog Structure
Application
Group box
Position...
E75 (3) 800
ec7server5
INS

Application: GL_MASTER

Group Box

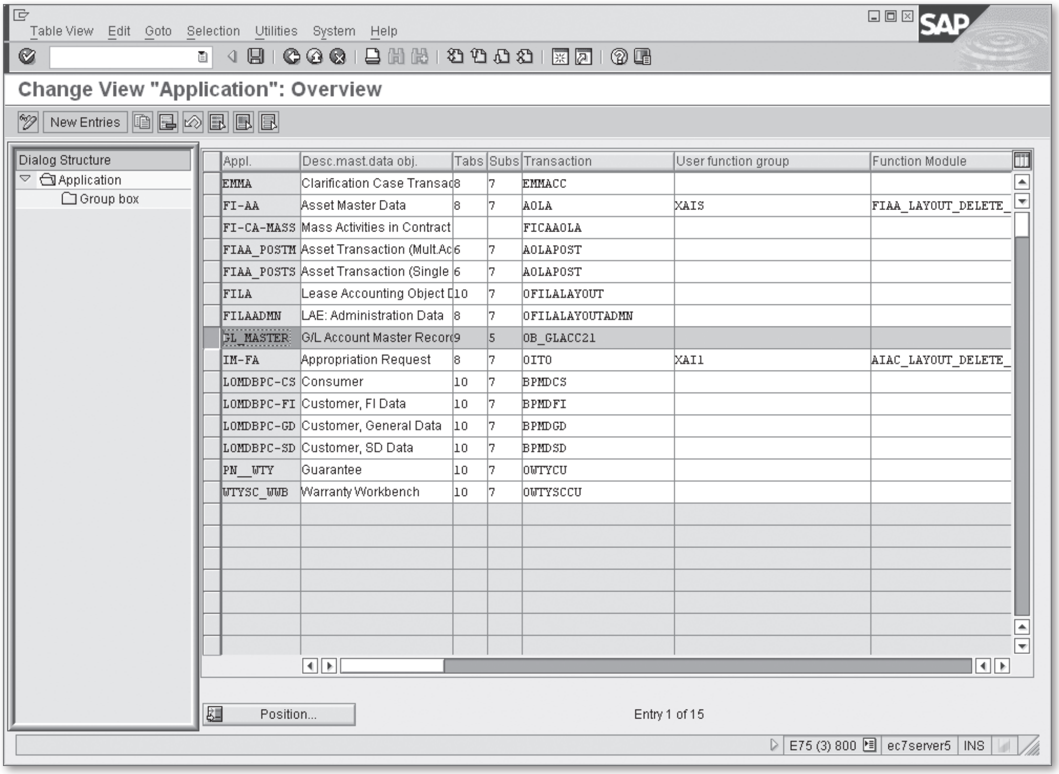
Grp	Description of group box	Program	Scr.	Req	Hide	Alt.
1	Account type	SAPLGL_ACCOUNT_MASTER_MAINTAIN	2102	☐	☐	
2	Description in maintenance language	SAPLGL_ACCOUNT_MASTER_MAINTAIN	2101	☐	☐	
3	Consolidation	SAPLGL_ACCOUNT_MASTER_MAINTAIN	2103	☐	☐	
4	Translation	SAPLGL_ACCOUNT_MASTER_MAINTAIN	2105	☐	☐	
5	Key words	SAPLGL_ACCOUNT_MASTER_MAINTAIN	2104	☐	☐	
20	Account assignmt info (chart of accts)	SAPLGL_ACCOUNT_MASTER_MAINTAIN	2120	☐	☐	
99	Information (chart of accounts)	SAPLGL_ACCOUNT_MASTER_MAINTAIN	2199	☐	☐	
101	Account control	SAPLGL_ACCOUNT_MASTER_MAINTAIN	2201	☐	☐	
102	Account management	SAPLGL_ACCOUNT_MASTER_MAINTAIN	2202	☐	☐	
103	Document entry control	SAPLGL_ACCOUNT_MASTER_MAINTAIN	2203	☐	☐	
104	Bank details	SAPLGL_ACCOUNT_MASTER_MAINTAIN	2204	☐	☐	
105	Interest calculation	SAPLGL_ACCOUNT_MASTER_MAINTAIN	2205	☐	☐	
106	Joint venture	SAPLGL_ACCOUNT_MASTER_MAINTAIN	2206	☐	☐	
120	Account assignmt info (company code)	SAPLGL_ACCOUNT_MASTER_MAINTAIN	2220	☐	☐	
199	Information (company code)	SAPLGL_ACCOUNT_MASTER_MAINTAIN	2299	☐	☐	
201	Lock (chart of accounts)	SAPLGL_ACCOUNT_MASTER_MAINTAIN	2301	☐	☐	
202	Lock (company code)	SAPLGL_ACCOUNT_MASTER_MAINTAIN	2302	☐	☐	
301	Deletion flag (chart of accounts)	SAPLGL_ACCOUNT_MASTER_MAINTAIN	2311	☐	☐	
302	Deletion flag (company code)	SAPLGL_ACCOUNT_MASTER_MAINTAIN	2312	☐	☐	

Entry 1 of 19

Figure 2.14 Layout List for the GL_MASTER Application Area

You now need to insert a record denoting the new subscreen.

1. Switch to edit mode by clicking the TOOLBAR button (✎). The system displays two messages. The first warns that the table is client independent and that your changes will affect all clients defined in your system. The second warns: "Do not make any changes (SAP data)." You don't plan to modify the SAP layout; you just want to add your input without any changes to the standard logic. With this information in mind, you can proceed further into edit mode.
2. The screen shown in Figure 2.15 displays all application areas that use the same customizable tabstrip layout technique. Select the GL_MASTER application area, and double-click on the folder icon labeled GROUP BOX on the left pane of the window. You will see the list of all subscreen used by this application area.



Appl.	Desc.mast.data obj.	Tabs	Subs	Transaction	User function group	Function Module
EMMA	Clarification Case Transac	8	7	EMMACC		
FI-AA	Asset Master Data	8	7	AOLA	XAIS	FIAA_LAYOUT_DELETE_
FI-CA-MASS	Mass Activities in Contract			FICAAOLA		
FIAA_POSTM	Asset Transaction (Mult.Ac	6	7	AOLAPOST		
FIAA_POSTS	Asset Transaction (Single	6	7	AOLAPOST		
FILA	Lease Accounting Object D	10	7	OFILALAYOUT		
FILAADMN	LAE: Administration Data	8	7	OFILALAYOUTADMN		
GL_MASTER	G/L Account Master Recor	9	5	OB_GLACC21		
IM-FA	Appropriation Request	8	7	OITO	XAI1	AIAC_LAYOUT_DELETE_
LOMDBPC-CS	Consumer	10	7	BPMDCS		
LOMDBPC-FI	Customer, FI Data	10	7	BPMDFI		
LOMDBPC-GD	Customer, General Data	10	7	BPMDGD		
LOMDBPC-SD	Customer, SD Data	10	7	BPMDSD		
PN__WTY	Guarantee	10	7	OWTYCU		
WTYSC_WWB	Warranty Workbench	10	7	OWTYSCCU		

Figure 2.15 Application Area List in the First Screen of View Cluster VC_TAMLAYA_00

3. Click the New Entries button, and enter the following information:
   - 901 for the Group Box field (we chose the 901 value so as not to interfere with the SAP standard key values)
   - Arbitrary description for the Group box (in the sandbox IDES [Internet Demonstration and Evaluation System] system, we used the word "Enhancement")
   - ZGLACC_EXT for Program
   - 0100 for Screen number
4. Leave all other fields of the line uninitialized, and save the entry.

Configuring Layout

Now that the subscreen is registered in the subscreen list of the `GL_MASTER` application area, you need to configure the new layout of the transaction. To do this, you open view cluster VC_TAMLAY_00 in Transaction SM34. When the system asks for the application area, enter "GL_MASTER".

The first screen (see Figure 2.16) of the view cluster shows available layouts for different general ledger account transactions. You can clearly understand the layout destination by its short description.

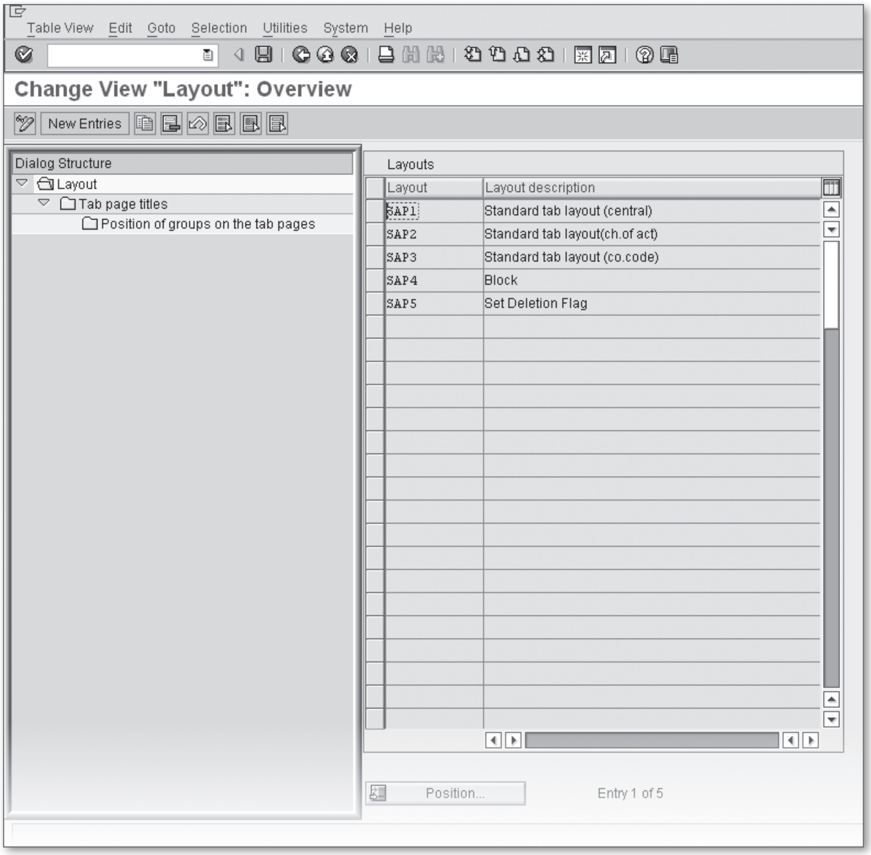


Table View Edit Goto Selection Utilities System Help
Change View "Layout": Overview
New Entries
Dialog Structure
Layout
Tab page titles
Position of groups on the tab pages
Layouts
Position...
Entry 1 of 5

Layout	Layout description
SAP1	Standard tab layout (central)
SAP2	Standard tab layout(ch.of act)
SAP3	Standard tab layout (co.code)
SAP4	Block
SAP5	Set Deletion Flag

Figure 2.16 Standard SAP Layouts for Application Area GL_MASTER

As shown in Figure 2.16, the left pane of the view window with tree control gives a notion of what can be customized here. First, you can define the number and titles of tabs of the tabstrip layout, and secondly, you can organize one or more subscreens on each tab.

Note

When implementing a dynamic tabstrip with function group `ATAB`, you cannot create a tab with more than seven subscreens on it.

Due to delivery class E of this maintenance view, you can't change the existing SAP layouts, so you should add the new one by copying one of the existing layouts. Because the plan is to add a subscreen with some company code specific data, it's logical to use layout SAP3 STANDARD TAB LAYOUT (CO.CODE) as a sample.

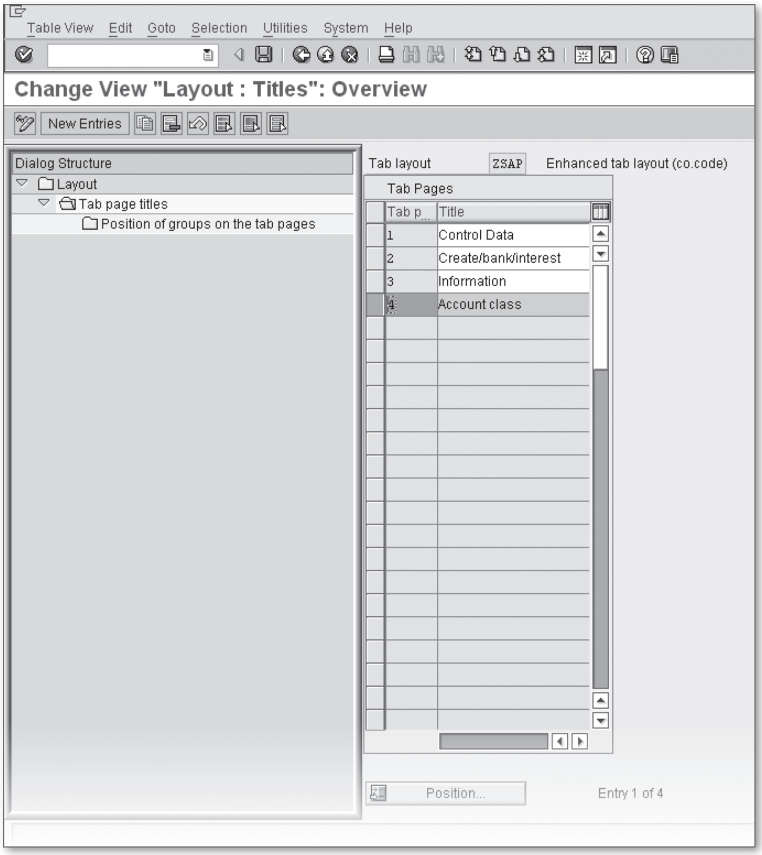
Note

If a configuration table has delivery class E, then the table contains predefined entries delivered initially by SAP. Also, you can add your own entries within your customer namespace (e.g., the key value must start with character Z or Y).

1. Select layout SAP3, and click the TOOLBAR icon (📋). When the system asks if you want to copy all depending records, choose YES.
2. Now you see a single line with key SAP3 and the short description. Change the key value to any code starting with character Z or Y, enter the short description "Enhanced tab layout (co.code)", and press `Enter`. In the IDES system, we created the layout ZSAP.

Now let's configure the layout so that the subscreen will occupy a separate tab.

1. Add a tab by adding a tab description. Select the newly created layout, and double-click the TAB PAGE TITLES folder on the left pane of the window.
2. Add a new tab number with a short description by clicking the NEW ENTRIES button (see Figure 2.17).
3. Select the tab, and double-click the POSITION OF GROUPS ON THE TAB PAGES folder. You can add up to seven subscreens to the tab. The field POSITION controls the order in which subscreens appear on a tab; the field GROUP BOX is a code name for subscreen. You can select previously registered subscreens by searching under HELP. Notice that SAP added the prefix S to the subscreen group number, so the subscreen appears with key S0901.



Change View "Layout : Titles": Overview

Dialog Structure
- Layout
  - Tab page titles
    - Position of groups on the tab pages

Tab layout ZSAP Enhanced tab layout (co.code)

Tab Pages

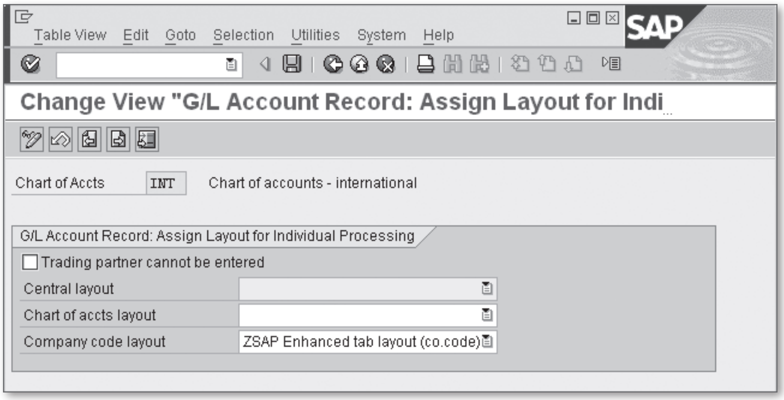
Tab p...	Title
1	Control Data
2	Create/bank/interest
3	Information
4	Account class

Entry 1 of 4

Figure 2.17 Adding a Tab to the ZSAP Tabstrip Layout

All of this work is still not enough to make the custom developed subscreen appear in a standard transaction. Another type of configuration activity has to be done in Financial Accounting (FI). There are various ways of configuring the general ledger account master data screen: The layout can be assigned either to a chart of accounts or an account group. Let's assign the layout to a whole chart of accounts by editing an entry in the maintenance view `V_T004_B`. In the sandbox system, we assigned a new layout to the INT chart of account, which should be well known to those of you who attended SAP Financials training courses.

Now you can see the layout settings for the INT chart of accounts (see Figure 2.18). Note that we made our ZSAP layout the default company code layout.



Change View "G/L Account Record: Assign Layout for Indi

Chart of Accts INT Chart of accounts - international

G/L Account Record: Assign Layout for Individual Processing

☐ Trading partner cannot be entered

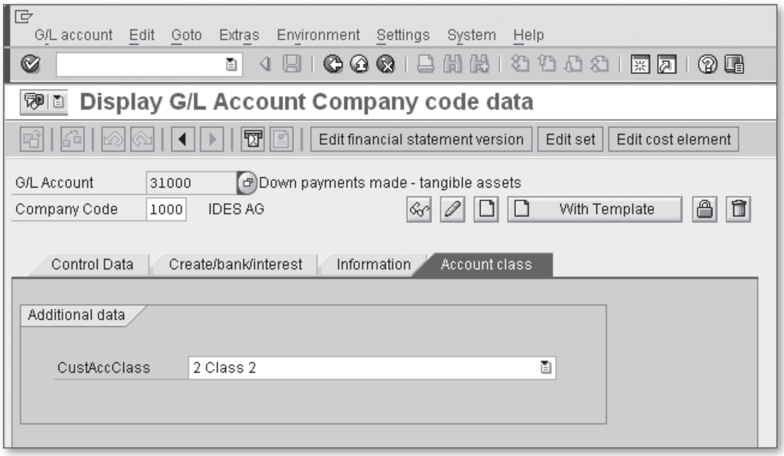
Central layout

Chart of accts layout

Company code layout ZSAP Enhanced tab layout (co.code)

Figure 2.18 INT Chart of Accounts Layout Assignment

Finally, you can start Transaction FSS0—general ledger account maintenance in company code—and see the resulting screen, which has an additional tab: ACCOUNT CLASS (see Figure 2.19).



Display G/L Account Company code data

G/L Account 31000 Down payments made - tangible assets

Company Code 1000 IDES AG

Additional data

CustAccClass 2 Class 2

Figure 2.19 Enhanced Screen of General Ledger Account Master Data

Defining Screen Flow Logic

If you play with the newly added subscreen while editing any general ledger account master data in Transaction FSS0, you'll see that the list box is still not functional because of the following:

- The field value is not saved.
- The list box behaves the same way both in edit mode and in display mode.

To make the list box work, you must open the module pool `ZGLACC_EXT` to do some programming.

First, it's important to understand that field properties function differently depending on the transaction mode. In edit or create mode, the field should be ready for input; in display mode, it should not.

Knowing the mode in which the transaction is working is very important; unfortunately, the transaction code doesn't indicate the editing mode. And here again, you need to dive into the source code. When you look into a couple of PBO (process before output) modules of screens belonging to function group `GL_ACCOUNT_MASTER_MAINTAIN`, you can see that the running transaction mode is stored in the field `activity` of the global structure variable `status`. Also, the field `activity` can take five different values (see Table 2.1).

Mode	Description
1	Display Mode
2	Edit mode
3	Create mode
4	Block mode
5	Delete mode

Table 2.1 Available Modes of a General Ledger Master Data Transaction

Unfortunately, there's no simple way of retrieving the value of the `status-activity` global variable other than the dynamic assign technique.

Note

SAP does not recommend the dynamic assign technique because you can read and even change virtually any global data of any SAP program loaded into the same session together with your program.

Screen properties are modified in PBO modules, so you have to create the following entry in the `PROCESS BEFORE OUTPUT` screen logic part of screen 0100:

```
MODULE modify_screen.
```

Note

When implementing screen flow logic modules, avoid creating long module code (not more than five to seven lines of code) due to the screen logic module's strange visibility rules: If you declare a variable inside `MODULE…ENDMODULE` boundaries, it becomes global, so it keeps its value between module runs.

The resulting logic inside the `modify_screen` module should be as shown in Listing 2.2. Note that you access the `status-activity` variable using dynamic assign.

```
FIELD-SYMBOLS: <activity> TYPE char1.

ASSIGN ('(SAPLGL_ACCOUNT_MASTER_MAINTAIN)STATUS-ACTIVITY') TO <activity> CASTING.

CHECK <activity> IS ASSIGNED.

LOOP AT SCREEN.
  CASE screen-group1.
    WHEN '901'.
      IF <activity> CA '23'. "Edit or Create mode
        screen-input = '1'.
      ELSE.
        screen-input = '0'.
      ENDIF.
    WHEN OTHERS.
      CONTINUE.
  ENDCASE.
  MODIFY SCREEN.
ENDLOOP.
```

Listing 2.2 Flow Logic in a PBO Module

Save and activate both the screen and the module pool, and then start Transaction FSS0. Try switching to edit and display mode. The field `CustAccClass` should be grayed in display mode and ready for input in edit mode.

The value of the field isn't stored in the database table SKB1. To achieve this, you must implement moving values between runtime structure `AC_NEW` (see the

Enhancing the Auxiliary Structure section earlier in this chapter for details on the `AC_NEW` structure) of function group `GL_ACCOUNT_MASTER_MAINTAIN` and your own runtime data.

As in the case of the transaction mode, here you have to use the dynamic assign technique. Now you need to implement the logic both in PBO and PAI (process after input) modules. The PBO logic moves actual account data from the runtime program variable to the screen field; the PAI logic does the reverse. In this case, you need to move the field value from the `AC_NEW` structure to your SKB1 table work area and vice versa. See Listing 2.3 and Listing 2.4 for flow logic implementation.

```
FIELD-SYMBOLS: <ac_new> TYPE account.

ASSIGN (‘(SAPLGL_ACCOUNT_MASTER_MAINTAIN)AC_NEW’) TO <ac_new> CASTING.

CHECK <ac_new> IS ASSIGNED.

skb1-zzcust_class = <ac_new>-ccode_data-zzcust_class.
```

Listing 2.3 PBO Logic for SKB1 Additional Field

```
FIELD-SYMBOLS: <ac_new> TYPE account.

ASSIGN (‘(SAPLGL_ACCOUNT_MASTER_MAINTAIN)AC_NEW’) TO <ac_new> CASTING.

CHECK <ac_new> IS ASSIGNED.

<ac_new>-ccode_data-zzcust_class = skb1-zzcust_class.
```

Listing 2.4 PAI Logic for the SKB1 Additional Field

After implementing the flow logic, you activate the module pool and screen restart Transaction FSS0 to check that the new field is successfully stored in the database table. The SAP system registers all changes made to the field via the flag CHANGE DOCUMENT in the `ZACC_CUST_CLASS` data element. You can check this by opening the INFORMATION tab of the changed general ledger account in Transaction FSS0 and clicking the CHANGE DOCUMENTS button.

2.1.4 Other Enhancements Available in General Ledger Account Master Data

As we already mentioned in the beginning of the chapter, it seems as though SAP doesn't assume there is much demand for general ledger account master data enhancement, so the set of existing enhancements is reduced to the process of checking data before saving.

Data Checks

To see the details of the data check exit, you can open subroutine `CALL_USER_EXITS` in module pool `SAPMF02H`. SAP uses two types of user exits: one BTE and an older style customer enhancement.

Each of the exits has the same set of parameters:

- Chart of account data of type `SKA1`
- Company code account data of structure `SKB1`
- One-character calling mode (`MODE`)
- Table parameter with line type `BAPIRET2`

SAP doesn't expect any exceptions from these user exits; all of the messages should be passed via an export table parameter return of type `BAPIRET2`, which is a standard SAP container for message information. If any of the returned messages have type E, A, or X, the saving process will be interrupted with the corresponding message.

These data check user exits are called twice in different scenarios:

- Just after the internal checks are performed. This moment occurs when a user performs the Check or Save command. In this case, the parameter mode contains the value "C."
- Just before the database update. The mode parameter contains the value "U."

Business Transaction Event (Open FI)

Function module `OUTBOUND_CALL_00002310_E` is the calling point for P&S BTE (publish and subscribe business transaction event) modules for the event `00002310`. The event has sample function module `SAMPLE_INTERFACE_00002310`. To implement the event, copy the sample function module to your own event, and implement your logic there. To activate the event, you have to perform customizing activities in Transaction FIBF.

Function Module Exits

The statement `CALL CUSTOMER-FUNCTION '001'` is actually translated into the function module `EXIT_SAPMF02H_001` call. This function module is the only component of the old-style enhancement `SAPMF02H`; you can examine it in Transaction SMOD.

SAP Internal BAdI

A call is also made to BAdI `FI_LIMIT_ACCOUNT` before saving account data and after other checks. The BAdI definition is marked as SAP internal, so you can't use it. SAP uses the user exit to implement logic in additional components and industry solutions.

Checking Texts

As you can see in Transaction FS00 on tabs INFORMATION (C/A) and INFORMATION (COCD), each general ledger account has two sets of texts: chart of accounts texts and company code texts. A special user exit also exists for checking detailed texts. This user exit is called after all other data of the general ledger account are checked and saved and before finalizing the `COMMIT`. You can see the logic in subroutine `CALL_USER_EXIT_TEXTS` of the `SAPMF02H` module pool.

To implement the text checking user exit, you must do the following:

- Implement the logic in a subroutine (`FORM`) of your own program. The subroutine must have seven parameters (for types and descriptions, see Table 2.2).
- Make at least one entry in the customizing table TKEEXITS where you have to set the following fields (other fields are optional):
  - `EXITID` = "`EXT_SAPMF02H_002`": Exit identification.
  - `ISACTIVE` = "X": Activation flag.
  - `REPORT`: Your report or module pool name.
  - `FORM`: Your subroutine name.

To make several entries, you can set different values in the SEQNO field.

The TKEEXITS table has no maintenance dialog, but you can maintain it directly from Transaction SE11 via menu path UTILITIES • TABLE CONTENTS • CREATE ENTRIES.

No	Name	Type	Description
1	ACCOUNT	SAKNR	Account number.
2	CHART	KTOPL	Chart of accounts.
3	COA_TEXT	Table parameter. See the YS_GLACCOUNT_TEXT type definition in Listing 2.5 for the line type of the parameter.	Chart of accounts texts.
4	CCODE	BUKRS	Company code.
5	CC_TEXT	Table parameter. See Listing 2.5 for the line type definition of the parameter.	Company code texts.
6	RETURN	Table of BAPIRET2	Return messages.
7	ACTIVE		Activation flag. Actually the parameter is ignored if there are the general ledger account maintenance transactions.

Table 2.2 Text Checking Subroutine Parameter List

```
types:  begin of ys_glaccount_text,
            id        type tdid,
            language  type spras,
            lines     type tsftext,
        end of ys_glaccount_text.
```

Listing 2.5 The Structure of the Text Table Parameter

2.1.5 General Ledger Summary

This section answered many questions about enhancing general ledger accounts. We discussed the possible methods to enhance the data, screen layout, and behavior of general ledger account master data maintenance programs.

We added our own field to a standard SAP table, created subscreens with the data, and implanted a subscreen into a standard transaction layout. However, remember that it does take a substantial amount of time to debug and investigate standard SAP code.

You learned some developmental tricks that are required to make the end results behave seamlessly. We used the dynamic assign technique to access global SAP variables, which is generally not recommended by SAP.

We expanded the standard table with an additional field. To avoid developing our own database table access code, we successfully relied upon SAP standard routines and achieved field changes logging via a standard change document object.

We also examined other available user exits, including the text-checking exit.

2.2 Accounts Payable and Accounts Receivable

Whereas enhancing general ledger master records is an infrequent task in most projects, extending Accounts Payable (AP) and Accounts Receivable (AR) is a common task. In this section, you'll see how AP and Accounts Receivable master records can be enhanced. First, we discuss general information concerning both kinds of master data, and then we'll show the enhancement techniques in more detail for AP and AR separately.

2.2.1 Maintenance Transactions

There are numerous transactions in the system for maintaining customer and vendor master records. As a rule of thumb, the four-character transaction code follows this naming convention: the first letter can be F for Financial View, V for Sales view, or X for Central view; the second letter can be D for Customers or K for Vendors; and finally the last two characters can be 01 for Create, 02 for Change, and 03 for Display. All of the resulting transactions (and some others) use the same screen sequence management technique.

The module pool `SAMF02D` contains a major portion of the program logic for customer master maintenance. `SAMF02K` is the main module pool for vendors.

2.2.2 Data Enhancements

Before proceeding with data enhancements, we should review the technical structure of the customer and vendor master record. Unlike general ledger accounts, which have a considerably smaller number of tables, customers and vendors are represented in the system with dozens of tables, due to the rich variety of business activities they are involved in. To roughly estimate the number of tables, you can open the Repository Information System and find all database tables with names starting with LF (for vendors) and KN (for customers).

Note

LF derives from the German Lieferant, and KN from Kunde.

In the SAP ERP IDES system, there are 21 tables for vendors and 31 for customers. This is a logical breakdown because customers bring revenue, so businesses should know more about these entities. We don't need to examine all of these database tables; a couple of main tables are enough to illustrate the point.

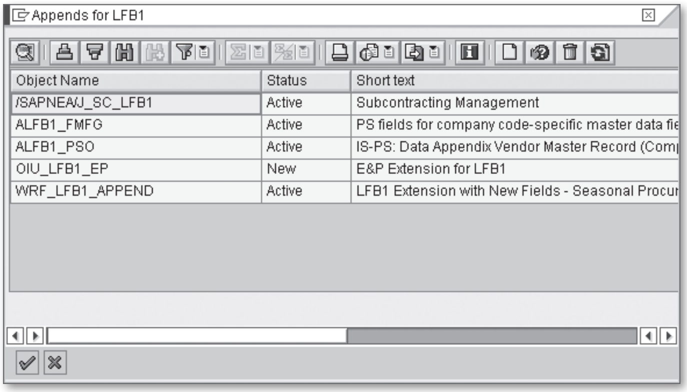
- **Main vendor tables:**
  - LFA1: Vendor master (general section)
  - LFB1: Vendor master (company code)
- **Main customer tables:**
  - KNA1: General data in customer master
  - KNB1: Customer master (company code)

As with the general ledger accounts, we will enhance company code view tables KNB1 and LFB1 with the same field: `ZZCUST_CLASS`. Using Section 2.1.2, Data Enhancement of General Ledger Account Master Data Tables, as a sample, you can enhance both tables by appending structures.

Note

Later in this chapter, you'll see that the customer and vendor enhancement technique assumes that you use your own database tables to expand master data. Such an approach requires more development efforts because you need to program the database table update code (insert, update, and delete). Furthermore, if you plan to employ change documents, you have to implement change document update code.

Several industry solutions and third-party components are installed in the IDES system, so tables LFB1 and KNB1 already have a number of append structures. In this case, after clicking the Append Structure button in Transaction SE11, a dialog box appears with the available append structures listed, as shown in Figure 2.20.

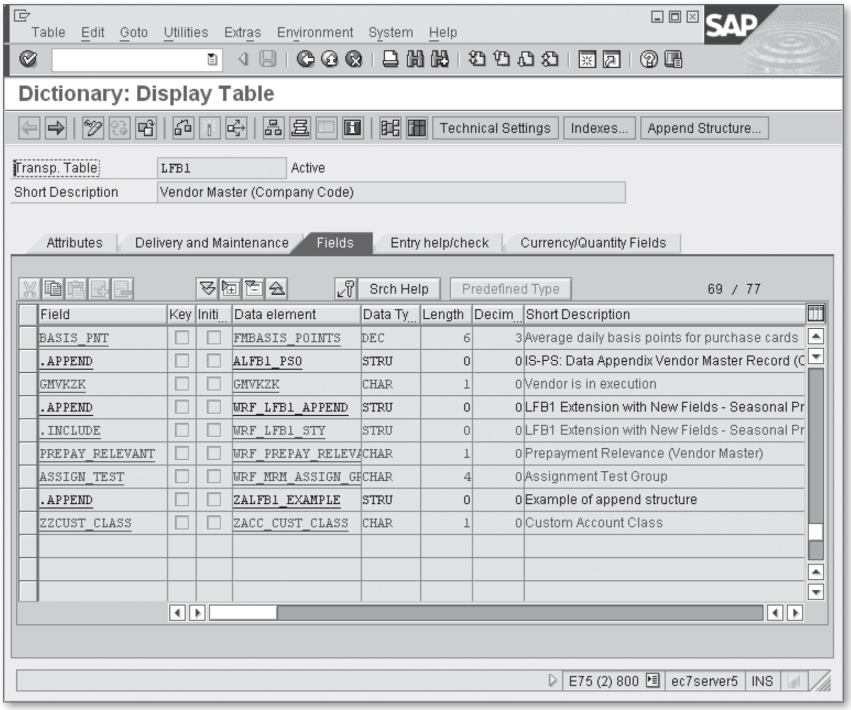


Appends for LFB1

Object Name	Status	Short text
/SAPNEA/J_SC_LFB1	Active	Subcontracting Management
ALFB1_FMFG	Active	PS fields for company code-specific master data fie
ALFB1_PSO	Active	IS-PS: Data Appendix Vendor Master Record (Com
OIU_LFB1_EP	New	E&P Extension for LFB1
WRF_LFB1_APPEND	Active	LFB1 Extension with New Fields - Seasonal Procur

Figure 2.20 List of Append Structures for Table LFB1

You then click the CREATE APPEND button (🗋) and enter the append structure name starting with Z: "ZALFB1_EXAMPLE" for LFB1 and "ZAKNB1_EXAMPLE" for KNB1. The resulting structure of Tables LFB1 and KNB1 should look as shown on Figure 2.21 and Figure 2.22. Don't forget to define the appropriate foreign keys.



Dictionary: Display Table

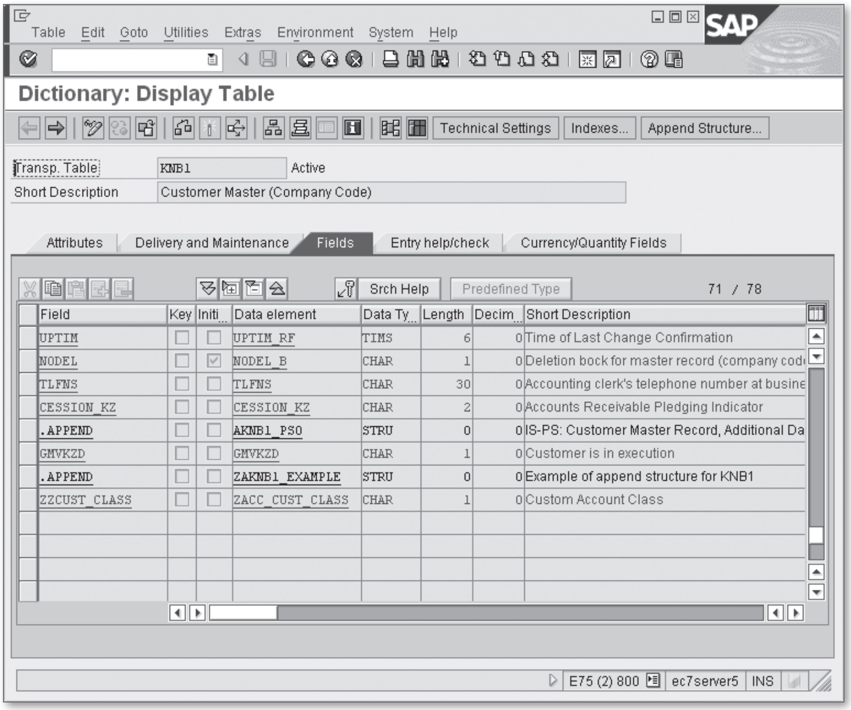
Transp. Table LFB1 Active
Short Description Vendor Master (Company Code)

69 / 77

Field	Key	Initi...	Data element	Data Ty...	Length	Decim...	Short Description
BASIS_PNT	☐	☐	FMBASIS_POINTS	DEC	6	3	Average daily basis points for purchase cards
.APPEND	☐	☐	ALFB1_PSO	STRU	0	0	IS-PS: Data Appendix Vendor Master Record (C
GMVKZK	☐	☐	GMVKZK	CHAR	1	0	Vendor is in execution
.APPEND	☐	☐	WRF_LFB1_APPEND	STRU	0	0	LFB1 Extension with New Fields - Seasonal Pr
.INCLUDE	☐	☐	WRF_LFB1_STY	STRU	0	0	LFB1 Extension with New Fields - Seasonal Pr
PREPAY_RELEVANT	☐	☐	WRF_PREPAY_RELEVA	CHAR	1	0	Prepayment Relevance (Vendor Master)
ASSIGN_TEST	☐	☐	WRF_MRM_ASSIGN_GF	CHAR	4	0	Assignment Test Group
.APPEND	☐	☐	ZALFB1_EXAMPLE	STRU	0	0	Example of append structure
ZZCUST_CLASS	☐	☐	ZACC_CUST_CLASS	CHAR	1	0	Custom Account Class

E75 (2) 800 ec7server5 INS

Figure 2.21 Enhanced Table LFB1



Dictionary: Display Table

Transp. Table	KNB1	Active
Short Description	Customer Master (Company Code)	

71 / 78

Field	Key	Initi...	Data element	Data Ty...	Length	Decim...	Short Description
UPTIM	☐	☐	UPTIM_RF	TIMS	6	0	Time of Last Change Confirmation
NODEL	☐	☑	NODEL_B	CHAR	1	0	Deletion bock for master record (company cod
TLFNS	☐	☐	TLFNS	CHAR	30	0	Accounting clerk's telephone number at busine
CESSION_KZ	☐	☐	CESSION_KZ	CHAR	2	0	Accounts Receivable Pledging Indicator
.APPEND	☐	☐	AKNB1_PSO	STRU	0	0	IS-PS: Customer Master Record, Additional Da
GMVKZD	☐	☐	GMVKZD	CHAR	1	0	Customer is in execution
.APPEND	☐	☐	ZAKNB1_EXAMPLE	STRU	0	0	Example of append structure for KNB1
ZZCUST_CLASS	☐	☐	ZACC_CUST_CLASS	CHAR	1	0	Custom Account Class

Figure 2.22 Enhanced Table KNB1

2.2.3 Screen Layout Enhancements

Unlike the general ledger accounts, where you had to dive into the source code to discover possible ways of UI enhancement, customer and vendor master data have special enhancement IMG subtrees in Financial Accounting (FI). Here you can find several nodes dealing with screen and data enhancements.

Customizing nodes can be found via the following menu path: Financial Accounting • Accounts Receivable and Accounts Payable. Then you can access the subtrees:

- For customers: Customer Accounts • Master Data • Preparations for Creating Customer Master Data • Adoption of Customer's Own Master Data Fields
- For vendors: Vendor Accounts • Master Data • Preparations for Creating Vendor Master Data • Adoption of Customer's Own Master Data Fields

Note

At first glance, the screen layout of customer master maintenance transactions such as XD02 or XD03 look very similar to that of general ledger account maintenance. Also, if you place a breakpoint into the `TABSTRIP_LAYOUT_READ` function module and open Transaction XD03, you'll see the debugger stopping at the function start. Unfortunately, you can't employ the same technique as you did for the general ledger account screen layout enhancement here because customer maintenance transactions use a single default tabstrip layout named `SAP`. This layout cannot be changed due to the delivery class of the customizing tables.

Settings and explanations of these two IMG subtrees look very similar although, in reality, vendor and customer transaction layouts are different.

Examining both IMG subtrees shows that SAP provides two BAdI definitions for the customer master (`CUSTOMER_ADD_DATA` and `CUSTOMER_ADD_DATA_CS`) and two definitions for the vendor master (`VENDOR_ADD_DATA` and `VENDOR_ADD_DATA_CS`). This is done to separate program logic dealing with screens (the CS suffix stands for customer screen) from the logic working "silently" without screen interaction. Also note that the CS-suffixed definition is filter dependent, and the filter value is the code of the customer screen group (see Section 2.3.2, Define Tabstrip Layout [Customer Screen Group]).

2.3 Accounts Receivable (Customers)

First, let's consider enhancement techniques for AR (or customer) master records. In this section, we'll create our own subscreen with its logic, configure our special screen layout to implant into the standard transaction, and look at data manipulating techniques using the available user exits.

2.3.1 Define Your Own Subscreen

First, let's design our own subscreen, which will be displayed in a customized view. Earlier, we created a special module pool for general ledger account master data enhancement, so it's logical to reuse it for the customer master record.

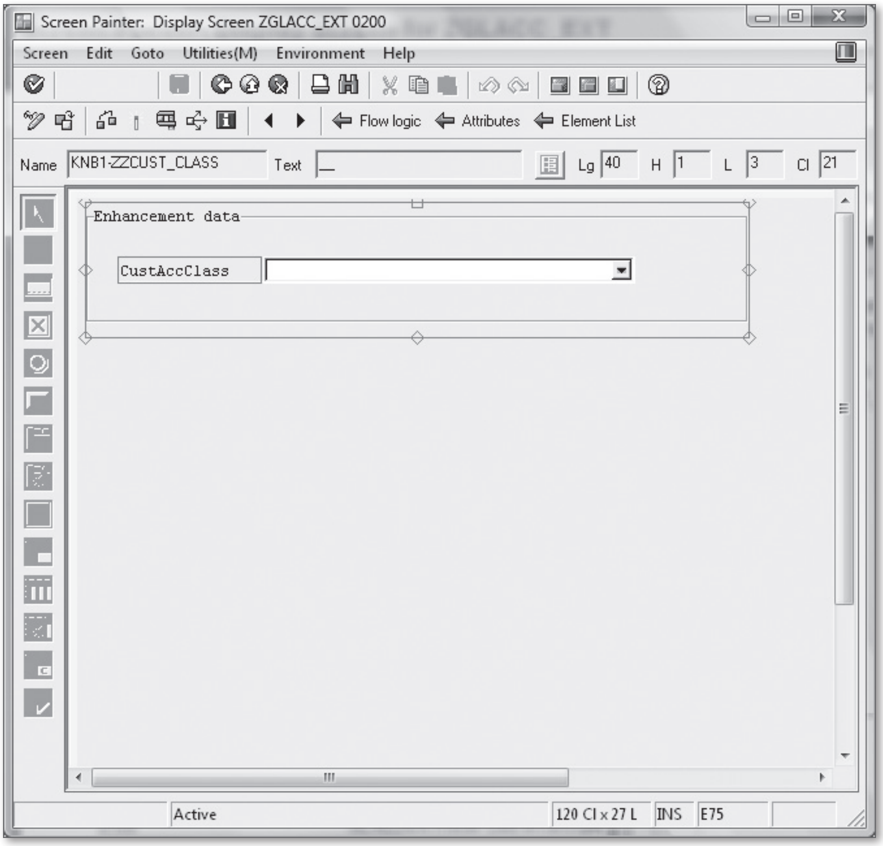
First, you need to declare the global data structures to be used for onscreen fields. Here you can use the `TABLES` statement:

```
TABLES: knb1, *knb1.
```

Note

`*KNB1` notation defines a table work area with the structure of database table KNB1. In earlier SAP code work, areas with asterisk prefixes were traditionally used as runtime storage for the data before editing, while the work area without asterisk prefixes held edited data.

You can create a subscreen with a single list box inside a frame, as you did earlier in Section 2.1.3, Screen Layout Enhancement. The resulting screen layout should look like Figure 2.23. Note that the list box references global field `KNB1-ZZCUST_CLASS`.


Screen Painter: Display Screen ZGLACC_EXT 0200
Screen Edit Goto Utilities(M) Environment Help
Flow logic
Attributes
Element List

Name KNB1-ZZCUST_CLASS Text _ Lg 40 H 1 L 3 Cl 21

Enhancement data

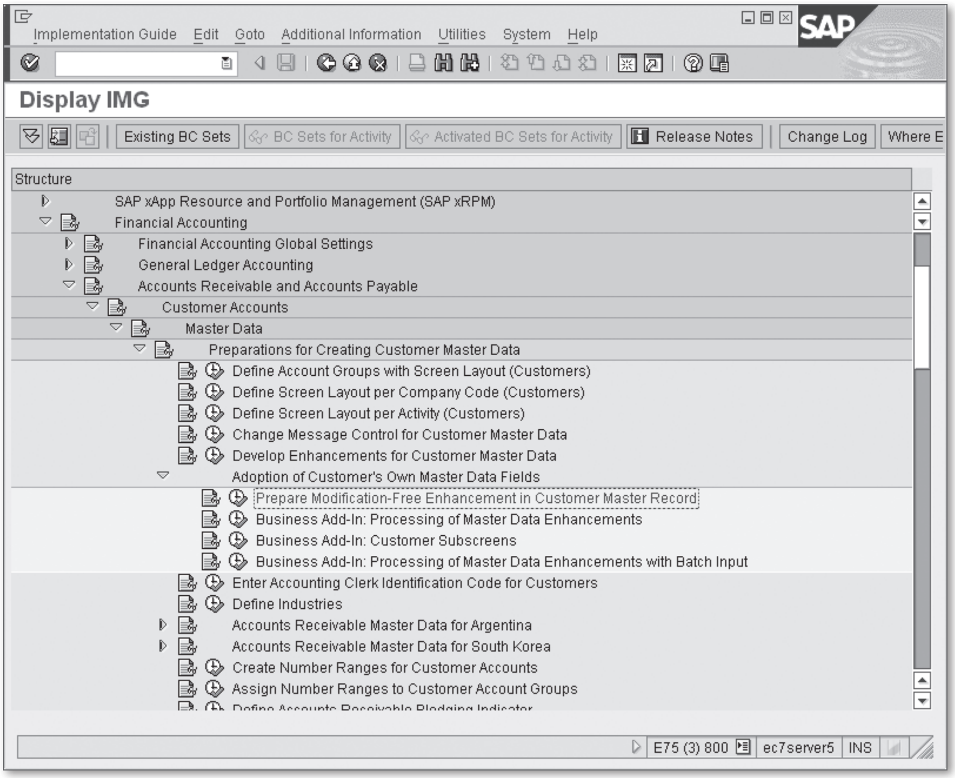
CustAccClass

Active 120 Cl x 27 L INS E75

Figure 2.23 Screen Layout in Design Time

2.3.2 Define Tabstrip Layout (Customer Screen Group)

Now you have to define customer screen groups in the SAP configuration utility IMG, which is accessible through Transaction SPRO. A screen group is just a grouping code for one or more screen tabs, which then are placed together into a standard SAP screen. See the starting IMG node for customer screen layout enhancements in Figure 2.24.



Display IMG

Structure
- SAP xApp Resource and Portfolio Management (SAP xRPM)
- Financial Accounting
  - Financial Accounting Global Settings
  - General Ledger Accounting
  - Accounts Receivable and Accounts Payable
    - Customer Accounts
      - Master Data
        - Preparations for Creating Customer Master Data
          - Define Account Groups with Screen Layout (Customers)
          - Define Screen Layout per Company Code (Customers)
          - Define Screen Layout per Activity (Customers)
          - Change Message Control for Customer Master Data
          - Develop Enhancements for Customer Master Data
          - Adoption of Customer's Own Master Data Fields
            - Prepare Modification-Free Enhancement in Customer Master Record
            - Business Add-In: Processing of Master Data Enhancements
            - Business Add-In: Customer Subscreens
            - Business Add-In: Processing of Master Data Enhancements with Batch Input
          - Enter Accounting Clerk Identification Code for Customers
          - Define Industries
          - Accounts Receivable Master Data for Argentina
          - Accounts Receivable Master Data for South Korea
          - Create Number Ranges for Customer Accounts
          - Assign Number Ranges to Customer Account Groups
          - Define Accounts Receivable Pledging Indicator

Figure 2.24 IMG Starting Node for Customer Screen Layout Enhancements

The IMG node is just another well-known view cluster maintenance dialog, and you should not experience any difficulties with its data manipulating.

As you can see in Figure 2.25, we added screen group Z0. Now you must define one or more tabs for a custom-defined screen group. To do this, select the newly defined screen group Z0, and double-click on the LABEL TAB PAGES folder in the left pane of the window.

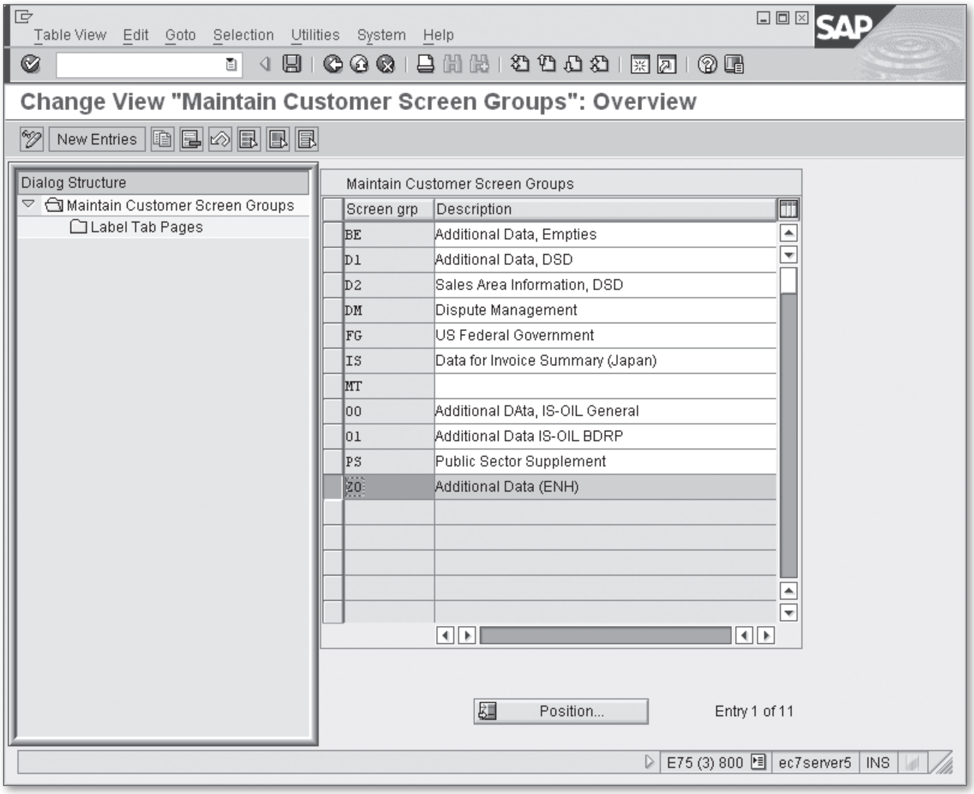


Table View Edit Goto Selection Utilities System Help
New Entries
Dialog Structure
Maintain Customer Screen Groups
Label Tab Pages
Position...
E75 (3) 800
ec7server5
INS

Change View "Maintain Customer Screen Groups": Overview

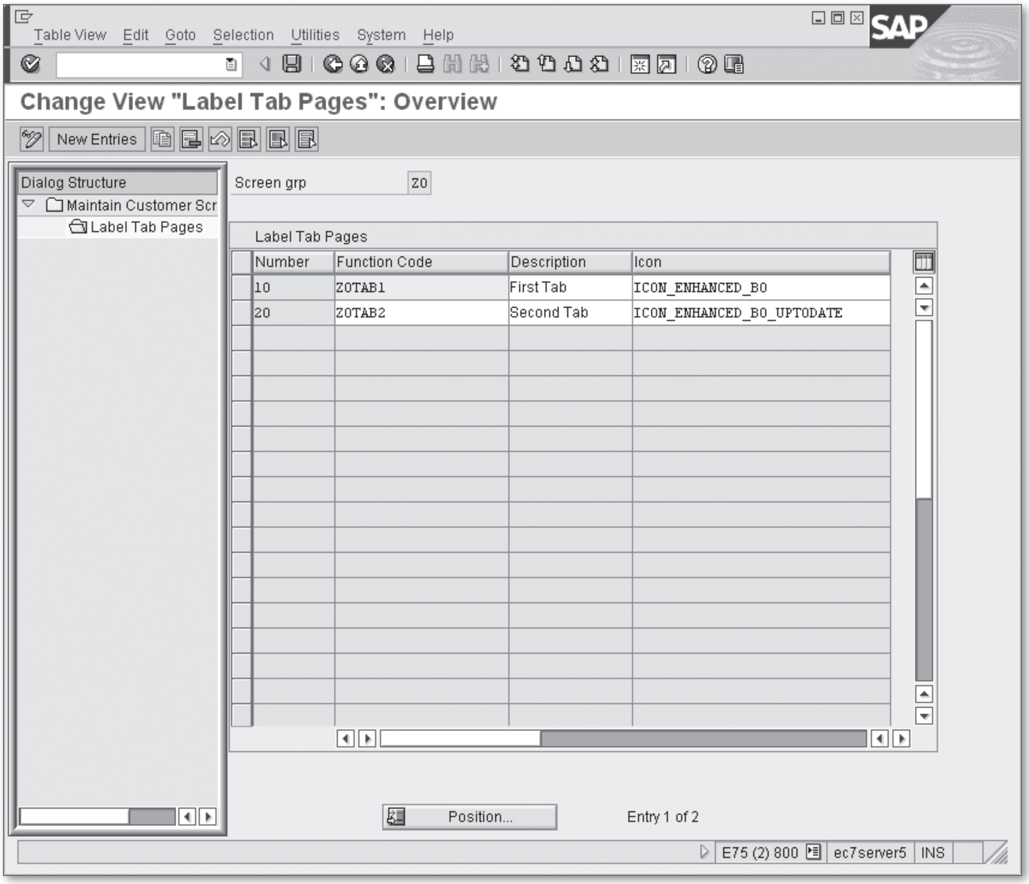
Maintain Customer Screen Groups

Screen grp	Description
BE	Additional Data, Empties
D1	Additional Data, DSD
D2	Sales Area Information, DSD
DM	Dispute Management
FG	US Federal Government
IS	Data for Invoice Summary (Japan)
MT	
00	Additional DAta, IS-OIL General
01	Additional Data IS-OIL BDRP
PS	Public Sector Supplement
Z0	Additional Data (ENH)

Entry 1 of 11

Figure 2.25 Newly Added Customer Screen Group (Z0)

On the next screen, you define two tabs with the names FIRST TAB and SECOND TAB. Note that you can supply each tab with an icon name; the screen field has a search function attached that lets you select the most applicable icon. You select here the ICON_ENHANCED_BO and ICON_ENHANCED_BO_UPTODATE icons. Also, you have to provide a unique function code for each created tab; function codes are then used to distinguish the tab selected by the user. Here you use Z0TAB1 and Z0TAB2 codes (Figure 2.26).



Change View "Label Tab Pages": Overview

Dialog Structure
- Maintain Customer Scr
  - Label Tab Pages

Screen grp: Z0

Label Tab Pages

Number	Function Code	Description	Icon
10	ZOTAB1	First Tab	ICON_ENHANCED_BO
20	ZOTAB2	Second Tab	ICON_ENHANCED_BO_UPTODATE

Entry 1 of 2

E75 (2) 800 | ec7server5 | INS

Figure 2.26 Tabs Definition Screen

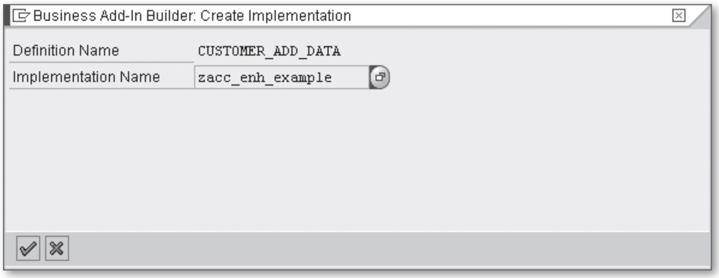
2.3.3 Activating a Screen Group via a BAdI Implementation

If you start one of the customer master data maintenance transactions immediately after creating the screen group with tabs, you won't be able to see any of the changes.

To make the group appear on the screen, you have to implement additional programming; namely, you employ a BAdI `CUST_ADD_DATA`.

1. Open IMG node Business Add-In: Processing of Master Data Enhancements. If the BAdI already has implementations in your system, you are prompted with a list of the BAdI implementations.

2. Click the New button, which takes you to the dialog box asking for the new BAdI implementation name (see Figure 2.27).


Business Add-In Builder: Create Implementation

Definition Name CUSTOMER_ADD_DATA

Implementation Name zacc_enh_example

Figure 2.27 BAdI Implementation Name Dialog Box

3. Enter an arbitrary name starting with Z or Y, and click OK (see Figure 2.27). In the IDES system, we have created an implementation with the name `ZACC_ENH_EXAMPLE`.

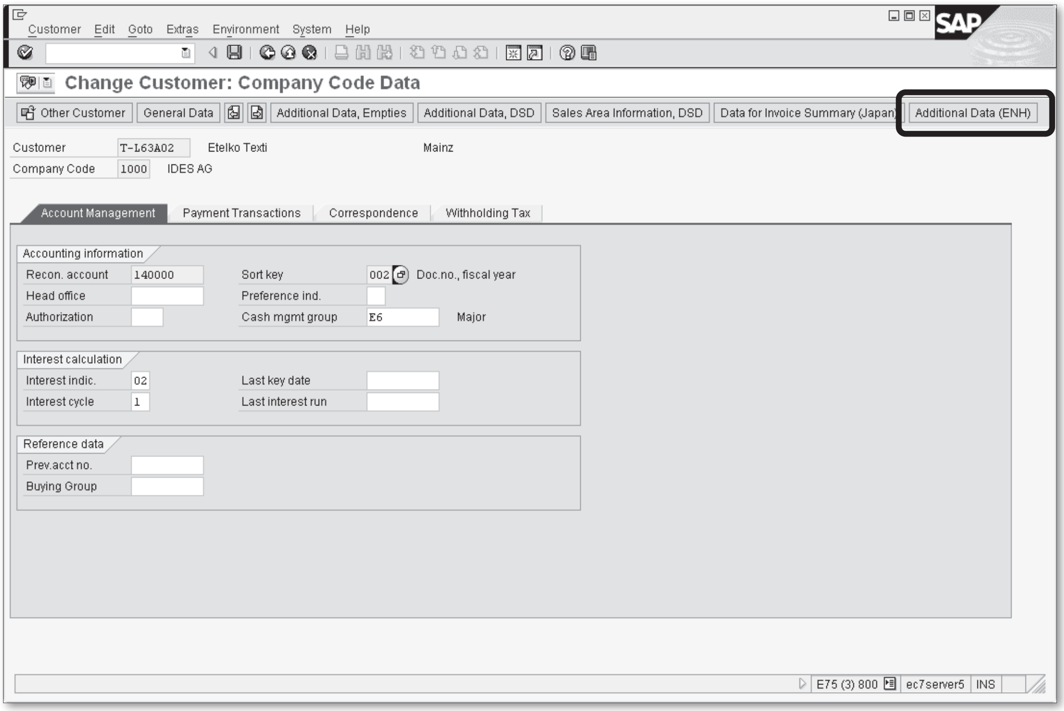
Now you can implement the `CHECK_ADD_ON_ACTIVE` method, which will be called while initializing the screen layout of the customer master data maintenance transaction. Its only task is to inform the runtime environment that the screen layout enhancement is active. The single input parameter `I_SCREEN_GROUP` contains the group name that we have already defined (`Z0`), while the output parameter `E_ADD_ON_ACTIVE` must contain "X" if the screen group is active. The source code of the method is quite simple, as shown in Listing 2.6.

```
METHOD if_ex_customer_add_data~check_add_on_active.
  IF i_screen_group = 'Z0'.
    e_add_on_active = 'X'.
  ENDIF.
ENDMETHOD.
```

Listing 2.6 CHECK_ADD_ON_ACTIVE Method Implementation

After activating the BAdI implementation, you can start the customer transaction again and see that an additional button has appeared on the toolbar (see Figure 2.28). Note that the button is titled according to the Z0 subscreen group name.

Also note that a new line appeared in the window menu at Goto • Enhancements. If you click the button Additional Data (ENH) (or choose Goto • Enhancements • Additional data [ENH]), you will see the customized view as shown in Figure 2.29.



Customer T-L63A02 Etelko Texti Mainz
Company Code 1000 IDES AG

Accounting information

Recon. account	140000	Sort key	002 Doc.no., fiscal year
Head office		Preference ind.	
Authorization		Cash mgmt group	E6 Major

Interest calculation

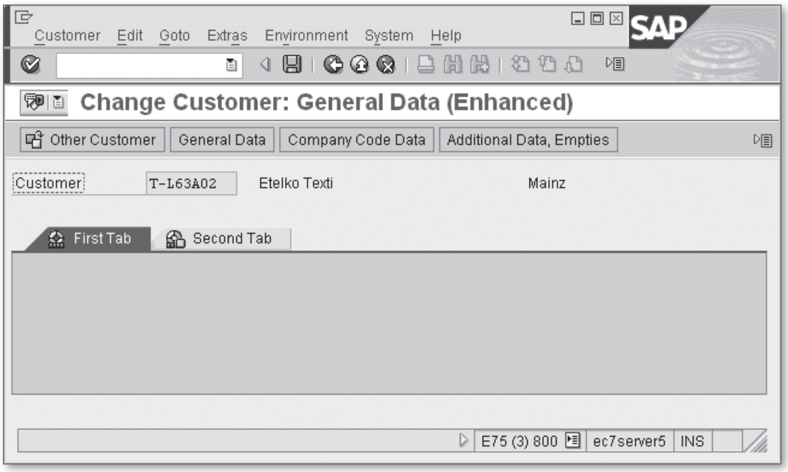
Interest indic.	02	Last key date	
Interest cycle	1	Last interest run	

Reference data

Prev.acct no.	
Buying Group	

E75 (3) 800 ec7server5 INS

Figure 2.28 New Toolbar Button Displayed on the Customer Master Data Screen


Customer Edit Goto Extras Environment System Help
Change Customer: General Data (Enhanced)
Other Customer
General Data
Company Code Data
Additional Data, Empties
First Tab
Second Tab

Customer T-L63A02 Etelko Texti Mainz

E75 (3) 800 ec7server5 INS

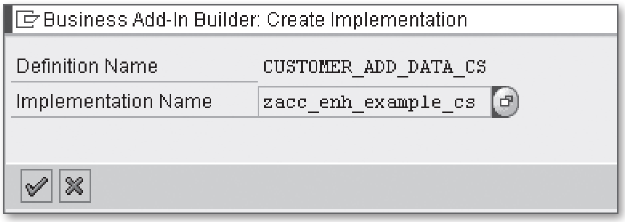
Figure 2.29 Enhanced Customer Master Screen Area

You can see the tabs we defined previously in the customizing view cluster with the correct title and the new icon. However, you can still see that both tab screens are empty, and also that the header part of the screen above the tabstrip doesn't contain any company code data (remember that we enhanced the company code data of a customer).

2.3.4 Linking Your Own Subscreen

The remedy for these issues is another BAdI implementation. Now you open IMG node Business Add-In: Customer Subscreens. Here you make an implementation of the CUSTOMER_ADD_DATA_CS BAdI definition.

As in the previous step, you'll see a dialog request for the BAdI implementation name (see Figure 2.30). Here you use implementation name ZACC_ENH_EXAMPLE_CS. As the definition is filtered by screen group code, you supply a Z0 as the filter value for the ZACC_ENH_EXAMPLE_CS implementation.


Business Add-In Builder: Create Implementation

Definition Name	CUSTOMER_ADD_DATA_CS
Implementation Name	zacc_enh_example_cs

Figure 2.30 BAdI Implementation Name Dialog Box

The BAdI has a number of methods in its interface, but for now you just need to implement one method: GET_TAXI_SCREEN. The system calls this method when initializing the screen layout. In the method implementation, you tell the system which screen you want to be displayed and also which view of the customer (company code, sales, or general) the screen is attached to.

The GET_TAXI_SCREEN method has three changing parameters to assign values to:

- E_SCREEN
  In this parameter, you assign the screen number that will be displayed on a customized tab.

- `E_PROGRAM`
  This parameter is the name of a program that the subscreen belongs to.
- `E_HEADERSCREEN_LAYOUT`
  This parameter defines the view of the customer master record, and it can accept three different values:
  - "B": For company code view.
  - "V": For sales view.
  - " " (space): For general view.

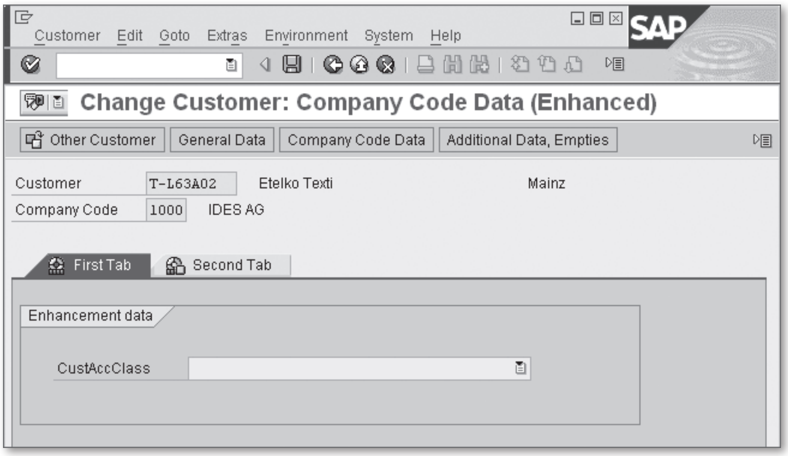
This method also has an import parameter, `I_TAXI_FCODE`, which contains a function code of the tab, so you can distinguish which one of the two tabs is selected. Let's suppose that you implant the previously defined subscreen 200 onto FIRST TAB; its corresponding function code is `Z0TAB1`.

Now that you know what the system expects from the method, you can implement it correctly as shown in Listing 2.7.

```
METHOD if_ex_customer_add_data_cs~get_taxi_screen.
  e_headerscreen_layout = 'B'.
  CASE i_taxi_fcode.
    WHEN 'Z0TAB1'.
      e_screen = '0200'.
      e_program = 'ZGLACC_EXT'.
    WHEN 'Z0TAB2'.
    WHEN OTHERS.
  ENDCASE.
ENDMETHOD.
```

Listing 2.7 GET_TAXI_SCREEN Method Implementation

After activating the method, you can reopen Transaction XD03 (or any other customer maintenance transaction), click the additional button, and see the previously developed subscreen on the first tab while the second remains empty (see Figure 2.31).



Customer T-L63A02 Etelko Texti Mainz
Company Code 1000 IDES AG
CustAccClass

Figure 2.31 The Subscreen Layout After Activating the GET_TAXI_SCREEN Method

2.3.5 Making the Screen Field Transaction Mode Aware and Updatable

The newly added screen still is not fully functional because it doesn't distinguish the edit/display mode, and data is not actually saved to the KNB1 table. To make the screen field aware of the current transaction mode (edit or display), you have to implement another BAdI method: `SET_DATA` of the `CUSTOMER_ADD_DATA_CS` BAdI definition. The method is called before displaying the enhanced layout and is used to transfer data from the standard transaction to the enhanced view.

You implement the method in the same BAdI implementation named `ZACC_ENH_EXAMPLE_CS`. The `SET_DATA` method has `I_ACTIVITY` import parameter, which contains the current mode of the running transaction. The list of possible values of the parameter `I_ACTIVITY` is as follows:

- "A": Display mode.
- "V": Edit mode.
- "H": Create mode.

There is also an `S_KNB1` import parameter containing runtime data of the customer company code data. You need this parameter to initialize the internal runtime data.

Because we use the ZGLACC_EXT module pool for manipulating additional data, it's a good idea to implement all of the main logic in this module. So, in the method implementation, you just call the external subroutine from ZGLACC_EXT as shown in Listing 2.8.

```
METHOD if_ex_customer_add_data_cs~set_data.

  PERFORM set_knb1 IN PROGRAM zglacc_ext

                   USING i_activity

                         s_knb1.

ENDMETHOD.
```

Listing 2.8 SET_DATA Method Source Code

In the same way, we implement another method—GET_DATA—to transfer data from the enhanced view back to the standard transaction (see Listing 2.9).

```
METHOD if_ex_customer_add_data_cs~get_data.
  PERFORM get_knb1 IN PROGRAM zglacc_ext CHANGING s_knb1.
ENDMETHOD.
```

Listing 2.9 GET_DATA Method Source Code

Finally, to tell the system that the data has been changed, you must implement method CHECK_DATA_CHANGED from another BAdI definition: CUSTOMER_ADD_DATA. The method has a single parameter, E_CHANGED, which tells the system if the data has been changed during the runtime. As in previous cases, we use an external subroutine call (see Listing 2.10).

```
METHOD if_ex_customer_add_data~check_data_changed.
  PERFORM check_knb1_changed IN PROGRAM zglacc_ext CHANGING e_changed.
ENDMETHOD.
```

Listing 2.10 CHECK_DATA_CHANGED Method Source Code

We are still not finished programming because we need to implement all of the declared logic in the module pool ZGLACC_EXT. We need a global variable for the current transaction mode (edit/display), as well as some PBO logic to turn the field editability on and off.

See Listing 2.11 for the full source code of logic for manipulating KNB1 data.

```
TABLES: knb1, *knb1.

DATA: BEGIN OF gs_knb1,
        loaded TYPE flag,
        aktyp TYPE aktyp,
      END OF gs_knb1.

*---------------------------------------------------------------------*
*  MODULE status_0200 OUTPUT
*---------------------------------------------------------------------*
*
*---------------------------------------------------------------------*
MODULE status_0200 OUTPUT.
  PERFORM status_0200.
ENDMODULE.                    "status_0200 OUTPUT

*&--------------------------------------------------------------------*
*&      Form  status_0200
*&--------------------------------------------------------------------*
*       text
*---------------------------------------------------------------------*
FORM status_0200 .
  LOOP AT SCREEN.
    IF gs_knb1-aktyp = 'A'. "Display
      screen-input = '0'.
    ELSE.
      screen-input = '1'.
    ENDIF.
    MODIFY SCREEN.
  ENDLOOP.
ENDFORM.                    " STATUS_0200

*&--------------------------------------------------------------------*
*&      Form  check_knb1_changed
*&--------------------------------------------------------------------*
*       text
*---------------------------------------------------------------------*
*      -->P_CHANGED  text
*---------------------------------------------------------------------*
```

```
FORM check_knb1_changed  CHANGING p_changed.
* Compare old and new value of zzcust_class field
  IF knb1-zzcust_class NE *knb1-zzcust_class.
    p_changed = ‘X’.
  ENDIF.
ENDFORM.                    “

*&---------------------------------------------------------------------*
*&      Form  set_knb1_aktyp
*&---------------------------------------------------------------------*
*       text
*----------------------------------------------------------------------*
*      -->VALUE(P_ACTIVITY)  text
*----------------------------------------------------------------------*
FORM set_knb1 USING  value(p_activity)
                     value(p_knb1) TYPE knb1.
  gs_knb1-aktyp = p_activity.
  IF gs_knb1-loaded IS INITIAL .
*   Initialize *KNB1 the very first time only
    *knb1 = p_knb1.
    gs_knb1-loaded = ‘X’.
  ENDIF.

  knb1 = p_knb1.
ENDFORM.                    “set_knb1_aktyp
*&---------------------------------------------------------------------*
*&      Form  GET_KNB1
*&---------------------------------------------------------------------*
*       text
*----------------------------------------------------------------------*
*      <--P_S_KNB1  text
*----------------------------------------------------------------------*
FORM get_knb1  CHANGING ch_knb1 TYPE knb1.
  ch_knb1 = knb1.
ENDFORM.                                                       “ GET_KNB1
```

Listing 2.11 KNB1 Enhancement Logic Implementation in Module Pool ZGLACC_EXT

2.3.6 Calling Moments of BAdI Methods

There are more methods within the BAdI definitions CUSTOMER_ADD_DATA and CUSTOMER_ADD_DATA_CS than we have examined so far, and it's useful to know when those are called. We'll explore several of these methods in the following subsections.

Initialization

Before displaying the first screen of the customer master maintenance transactions, the following methods of the BAdI definition `CUSTOMER_ADD_DATA` are called:

- `CHECK_ADD_ON_ACTIVE`
  This method tells the system that a particular screen group is implemented.
- `INITIALIZE_ADD_ON_DATA`
  This method is the initialization point for the BAdI implementation; be aware that at the moment of call, the user hasn't entered any data.

The First Screen PAI

After the user has entered values on the first visible transaction screen, the system calls one of the following methods of the BAdI definition `CUSTOMER_ADD_DATA`:

- `SET_USER_INPUTS`
  This method transfers all of the values entered on the first screen to the enhancement: customer number, organizational units (company code, sales organization, etc.). This method is called only in creation mode; thus, the customer number can be blank (for internal numbering).
- `READ_ADD_ON_DATA`
  Inside this method, you should select all additional tables (if any) depending on user inputs; this method is called only in display or change mode.

PBO Logic in the Tabstrip

In create or change mode, one of two `CUSTOMER_ADD_DATA` BAdI methods is called while processing PBO logic of all tabstrip subscreens:

- `PRESET_VALUES_CCODE`
  This method is called for company code data screens.
- `PRESET_VALUES_SAREA`
  This method is called for sales area data screens.

Both methods should primarily be used in create mode to fill in default values in the corresponding structures: KNB1 for company code data, and KNVV for sales area data.

For a particular enhanced screen tab, a set of `CUSTOMER_ADD_DATA_CS` BAdI methods is called:

- `SUPPRESS_TAXI_TABSTRIPS`
  This method hides unnecessary tabs depending on the customer number and its organizational assignment (company code, sales area, etc.).
- `SET_DATA`
  This method transfers current customer data to the logic.
- `GET_TAXI_SCREEN`
  This method tells the system the own screen number for each enhanced tab.

PAI Logic in the Tabstrip

When a user executes a command by clicking a toolbar button or selecting a menu command, the system allows the user interaction to be intercepted while displaying the enhanced screen by calling the `SET_FCODE` method of the `CUSTOMER_ADD_DATA_CS` definition.

The system also calls method `GET_DATA`, which we used in our earlier example, to transfer data from the enhancement implementation to the standard program.

Saving Data

When the user finally clicks the SAVE button, another chain of BAdI methods of the `CUSTOMER_ADD_DATA` definition is called:

- `CHECK_ALL_DATA`
  This method checks the data, as its name indicates. Note that if the program is in creation mode with internal numbering, the customer number is still undefined at the moment of call.
- `CHECK_DATA_CHANGED`
  This method tells the system that some data was changed, so it should update the database tables.

In creation mode, either method CHECK_ACCOUNT_NUMBER or method MODIFY_ACCOUNT_NUMBER will be called. The former allows checking the customer number in case of external numbering; the latter allows customer number modification after its assignment via an internal numbering technique.

- SAVE_DATA
  This method is called only if data was changed; the assumption is that all the additional tables will be saved inside the method.

Note

If you plan to implement your own change tracking via change documents, SAVE_DATA is the right place to insert system-generated includes for manipulating changes.

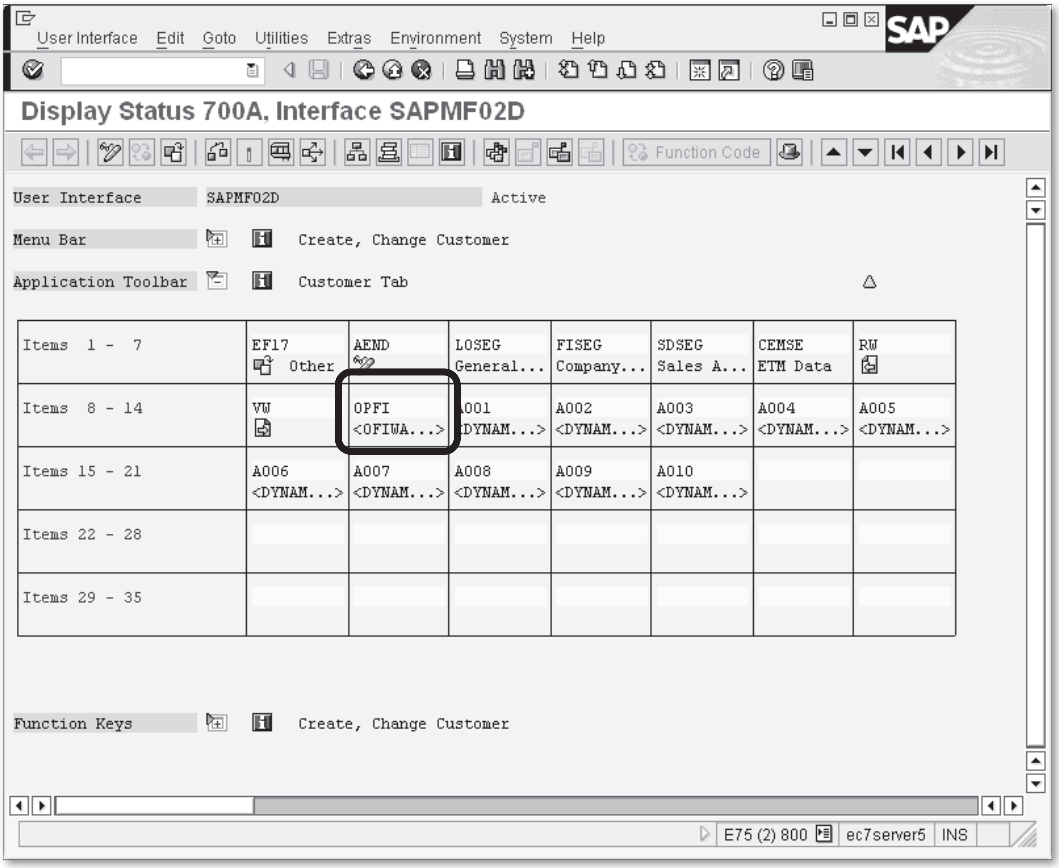
Changes Report

When a user opens a changes report for a particular customer via menu path Environment • Account changes, the system calls two BAdI methods that allow the user to tailor his own change document objects to the standard output:

- GET_CHANGEDOCS_FOR_OWN_TABLES
  This method is called to transfer all additional change document object names.
- BUILD_TEXT_FOR_CHANGE_DETAIL
  This method allows you to create your own explanatory text for a particular change item.

2.3.7 GUI Status Enhancement with Open FI (BTE)

Two BTEs can be used to make a slight alteration of the UI. If you open one of the GUI statuses of module pool SAPMF02D—700A or 700V—you notice a function code OPFI, based on the dynamic function text OFIWA-FTEXT (see Figure 2.32).



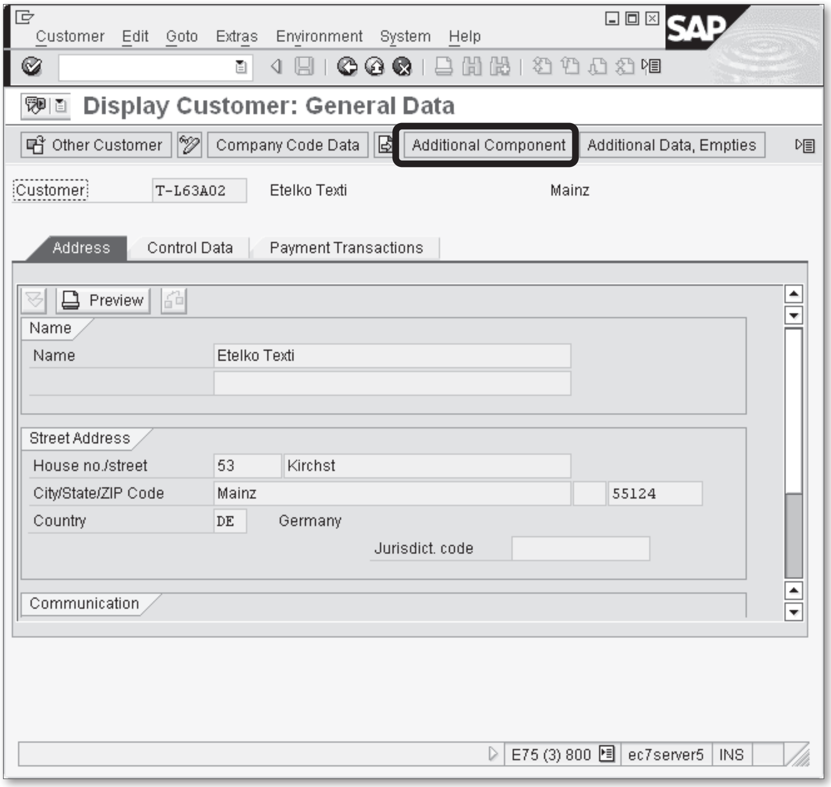
Display Status 700A, Interface SAPMF02D

User Interface	SAPMF02D Active						
Menu Bar	Create, Change Customer						
Application Toolbar	Customer Tab						
Items 1 - 7	EF17 Other	AEND	LOSEG General...	FISEG Company...	SDSEG Sales A...	CEMSE ETM Data	RW
Items 8 - 14	VW	OPFI <OFIWA...>	A001 <DYNAM...>	A002 <DYNAM...>	A003 <DYNAM...>	A004 <DYNAM...>	A005 <DYNAM...>
Items 15 - 21	A006 <DYNAM...>	A007 <DYNAM...>	A008 <DYNAM...>	A009 <DYNAM...>	A010 <DYNAM...>		
Items 22 - 28							
Items 29 - 35							
Function Keys	Create, Change Customer						

Figure 2.32 Status 700A of Module Pool SAPMF02D

This function code won't be visible and active unless you implement these two BTEs:

- `00001330`
  This event is called in the PBO logic of the starting transaction screen (see PBO module `TRANSAKTIONS_INIT` of module pool `SAPMF02D`). It allows transferring custom-defined function code text. If more than one function module is subscribed to the event, the `OPFI` function code shows the default text Additional Components as shown in Figure 2.33.



Customer T-L63A02 Etelko Texti Mainz

Name
Name Etelko Texti

Street Address
House no./street 53 Kirchst
City/State/ZIP Code Mainz 55124
Country DE Germany
Jurisdict. code

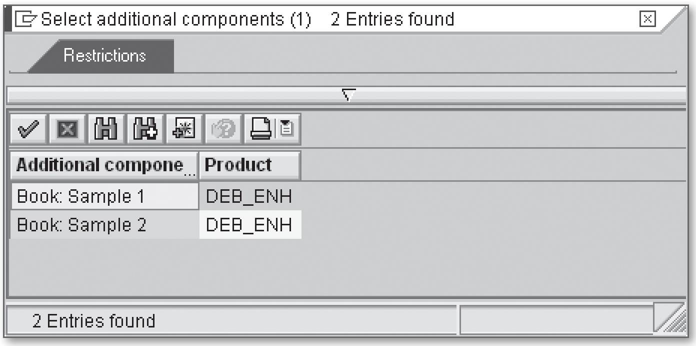
Communication

Figure 2.33 Additional Component Button in the Standard Toolbar of Transaction FD03

- `00001310`
  This event implements the reaction to function code `OPFI`. If more than one function is subscribed to that event, the system first shows a standard search help dialog for selecting a particular subscriber. For example, in the IDES system, we have defined two partner P&S modules for event `00001310`, so after clicking the ADDITIONAL COMPONENT button, the system pops up the dialog box shown in Figure 2.34.

Note

If you defined your BTE enhancement as a partner add-on, then you don't need to subscribe a function module to event `00001330` because the system can obtain function code texts from the `00001310` event subscription.


Select additional components (1) 2 Entries found

Restrictions

Additional compone...	Product
Book: Sample 1	DEB_ENH
Book: Sample 2	DEB_ENH

2 Entries found

Figure 2.34 BTE Enhancement Selection Box

Event 00001310 knows that your P&S function module can return the modification flag (using export parameter E_XCHNG) just like the method CHECK_DATA_CHANGED of the CUSTOMER_ADD_DATA BAdI.

In the next subsection, we briefly discuss other BTEs you can use to implement additional data saving.

2.3.8 Other Open FI (BTE) Events

Besides the BAdI definitions we discussed earlier, there are also a number of BTE calls in the customer master maintenance logic as described in Table 2.3.

Event Number	Calling Moment
00001360	Called in the PAI logic of the starting transaction screen after a user has entered transaction parameters (customer number, company code, etc.). The event can be used to implement additional authorization checks. This event is called in all transaction modes; while others can only be called in create and edit modes.
00001350	Called in the PAI logic of virtually all tabstrip screens in create or edit mode. This event allows for example change visibility of screen fields.
00001340	Called for in the final checks before saving (customer number can be undefined for internal numbering).
00001320	Called after the customer data has been updated and *before* calling the SAVE_DATA method of the CUSTOMER_ADD_DATA BAdI.

Table 2.3 Customer Master BTEs

Event Number	Calling Moment
00001321	Called after the customer data has been updated and *after* calling the SAVE_DATA method of the CUSTOMER_ADD_DATA BAdI, and after the customer change documents update.

Table 2.3 Customer Master BTEs (Cont.)

2.3.9 Function Module Exits

In addition to Open FI events, customer master logic contains a single customer function 001 of an old-fashioned enhancement SAPMF02D. The corresponding function module EXIT_SAPMF02D_001 is called before saving customer data, just after BTE 00001340.

The function module interface includes the full pack of the customer data, so the interface can be used. The function module doesn't include any exceptions, so you need to create an error or warning message directly if your logic encounters an error in the supplied data.

2.4 Customer Credit Management Data and Screen Enhancement

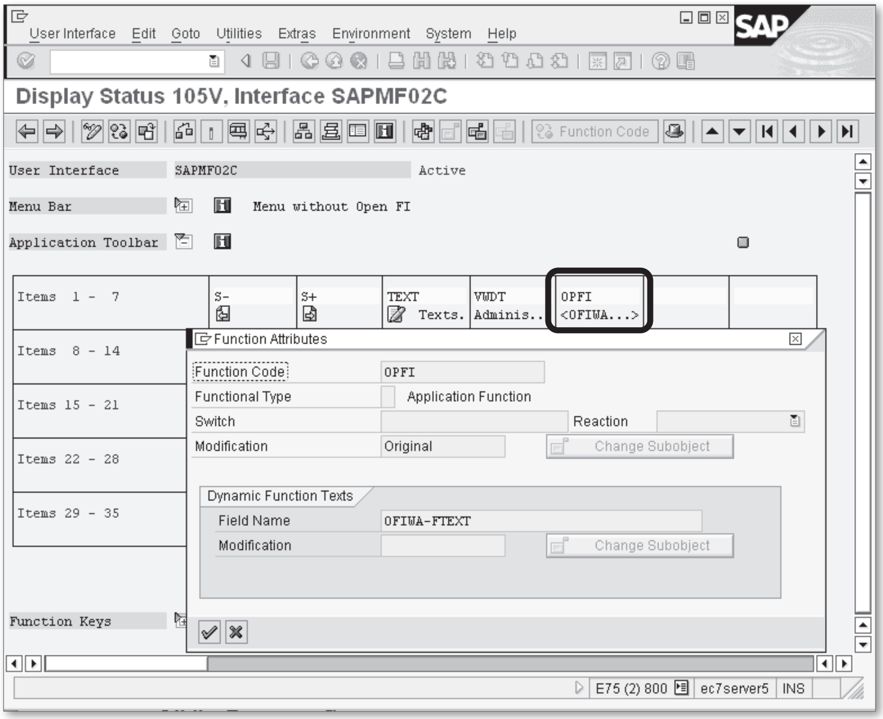
Customer credit control is an essential part of general AR functionality; however, credit control has its own subset of database tables for master data, specific organizational data, and separate master data maintenance transactions.

Note
Generally credit control helps track down customers' behavior and make decisions concerning customers' debts.

The enhancement technique of customer credit control data and the UI has its own specifics, so we decided to separate the exposition into this section. In this section, we will discuss methods of enhancements of credit control data and the screen layout of the main credit control transactions: FD32 and FD33.

2.4.1 GUI Status Enhancement

As in other customer master maintenance transactions, statuses of Transactions FD32/FD33 have one additional function code `OPFI` referencing the dynamic field `OFIWA-FTEXT` (see Figure 2.35).



Display Status 105V, Interface SAPMF02C

User Interface	SAPMF02C	Active
Menu Bar		Menu without Open FI
Application Toolbar		

Items 1 - 7	S-	S+	TEXT Texts.	VWDT Adminis..	OPFI <OFIWA...>		
Items 8 - 14							
Items 15 - 21							
Items 22 - 28							
Items 29 - 35							

Function Keys

Function Attributes

Function Code	OPFI		
Functional Type		Application Function	
Switch		Reaction	
Modification	Original	Change Subobject	

Dynamic Function Texts

Field Name	OFIWA-FTEXT	
Modification		Change Subobject

E75 (2) 800 | ec7server5 | INS

Figure 2.35 Additional Function Code in the GUI Status of the Module Pool SAPMF02C

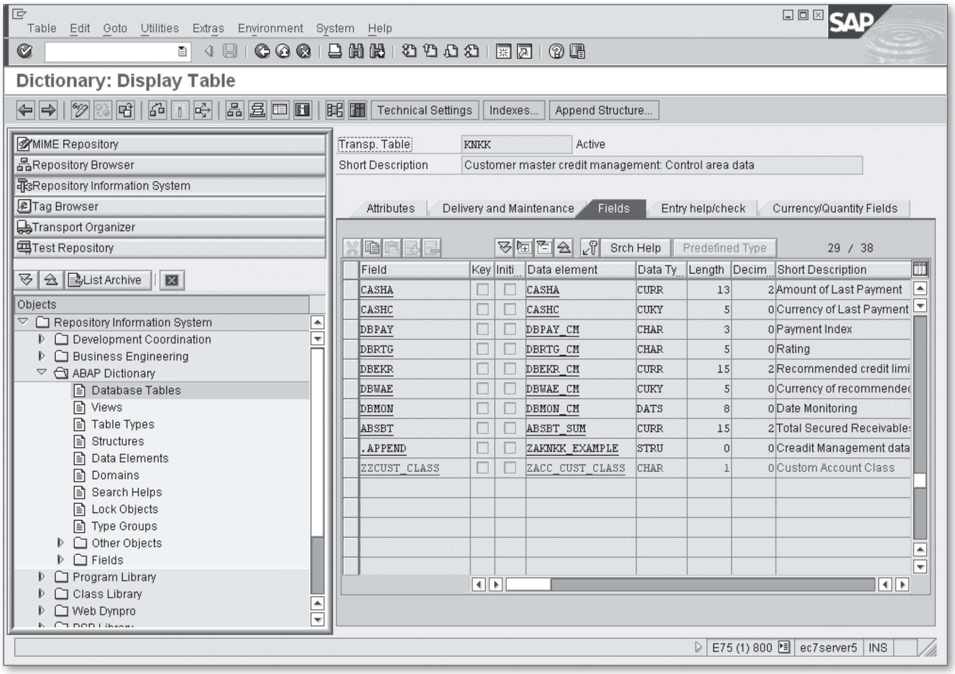
Similar to Section 2.3.7, GUI Status Enhancement with Open FI (BTE), you can use BTEs here to implement GUI status enhancement:

- `00001550` is used to obtain function code text that will be displayed in the toolbar and menu.
- `00001510` is called as a reaction to clicking an additional toolbar button or selecting an extra menu item. Just like in general customer maintenance transactions, you can subscribe more than one function module to this event, and, in this case, the system pops up a dialog box from which the user can choose a particular enhancement.

2.4.2 Data Enhancement

Customer credit management data are stored in database table KNKK. In addition to the customer number, the table has also the credit control area code as an additional key field. As with all the other standard tables we have enhanced earlier, we can enhance KNKK with an append structure and use as an example the same familiar field `ZZCUST_CLASS`.

After adding the append structure (don't forget to define a default foreign key for a new field), the table should look like Figure 2.36.


Dictionary: Display Table

Transp. Table	KNKK Active
Short Description	Customer master credit management: Control area data

Field	Key	Initi...	Data element	Data Ty...	Length	Decim...	Short Description
CASHA	☐	☐	CASHA	CURR	13	2	Amount of Last Payment
CASHC	☐	☐	CASHC	CUKY	5	0	Currency of Last Payment
DBPAY	☐	☐	DBPAY_CM	CHAR	3	0	Payment Index
DBRTG	☐	☐	DBRTG_CM	CHAR	5	0	Rating
DBEKR	☐	☐	DBEKR_CM	CURR	15	2	Recommended credit limi
DBWAE	☐	☐	DBWAE_CM	CUKY	5	0	Currency of recommended
DBMON	☐	☐	DBMON_CM	DATS	8	0	Date Monitoring
ABSBT	☐	☐	ABSBT_SUM	CURR	15	2	Total Secured Receivable:
.APPEND	☐	☐	ZAKNKK_EXAMPLE	STRU	0	0	Creadit Management data
ZZCUST_CLASS	☐	☐	ZACC_CUST_CLASS	CHAR	1	0	Custom Account Class

Figure 2.36 Enhanced Table KNKK

2.4.3 Status Screen Enhancement

Customer credit management Transactions FD32/FD33 have several views, but only one of them has screen enhancement capability. If you open the flow logic of screen 0210 of module pool `SAPMF02C`, you can see that there are many dynamic subscreen calls. Also note that external subscreen numbers and program names are obtained from global structure `RF61B`. Later, you'll see how you can use this information.

If your data enhancement is reduced to the append structure of table KNKK, the program logic is very simple and compact.

Let's begin by creating our own subscreen. First, you create module pool `ZKNKKENH`.

Note

You can't use the previously created module pool `ZGLACC_EXT` because this particular enhancement technique has technical restrictions for module pool name length. The technique is probably old enough to accept only program names with length less or equal to eight characters. This might be an indication that the technique was first introduced in SAP R/3 version 3.X, where all program names could not exceed eight characters in length.

First, you create a simple subscreen that has only one input field in the form of a list box, which references newly added field `KNKK-ZZCUST_CLASS`. In designtime, the screen should look like Figure 2.37.

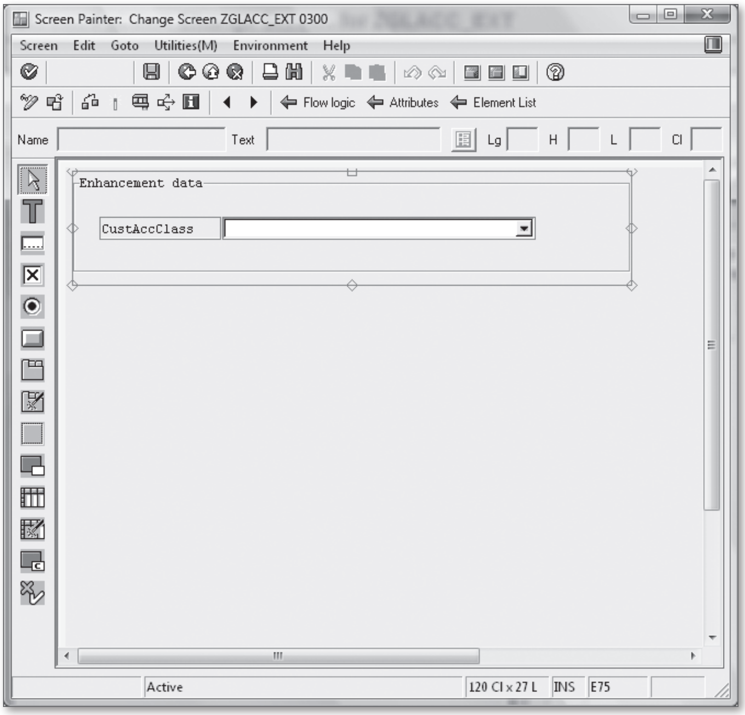


Figure 2.37 New Subscreen for Customer Credit Management Data

Its flow logic should be quite simple as shown in Listing 2.12.

```
PROCESS BEFORE OUTPUT.
  MODULE status_0300.

PROCESS AFTER INPUT.
* MODULE USER_COMMAND_0200.
```

Listing 2.12 Flow Logic of the Custom Defined Subscreen

As you can see, you don't even need a PAI module (you'll find out why later).

Now let's implement all of the necessary logic in the module pool source code. In full, it should look like Listing 2.13.

```
* Credit Control Data Enhancement
TABLES: knkk, t020.
*---------------------------------------------------------------------*
*  MODULE status_0300 OUTPUT
*---------------------------------------------------------------------*
*
*---------------------------------------------------------------------*
MODULE status_0300 OUTPUT.
  PERFORM modify_screen_0300.
ENDMODULE.                   "status_0300 OUTPUT

*&--------------------------------------------------------------------*
*&      Form  modify_screen_0300
*&--------------------------------------------------------------------*
*       text
*---------------------------------------------------------------------*
FORM modify_screen_0300.
  LOOP AT SCREEN.
    IF t020-aktyp = 'A'.
      screen-input = '0'.
    ELSE.
      screen-input = '1'.
    ENDIF.
    MODIFY SCREEN.
  ENDLOOP.
ENDFORM.                    "modify_screen_0300
```

Listing 2.13 The Source Code of Customer Credit Management Enhancement Logic

Note that we declared two table work areas: KNKK and T020. The former contains customer credit management data; the latter contains current transaction properties. Thanks to ABAP memory management, these work areas will be shared with the caller program when the subscreen is called. This ensures that we don't need any special code to transfer data to and from standard programs.

Note

Remember that sharing work areas declared with a `TABLES` statement is not possible for function groups. See further details in ABAP system help.

Now that you have created and activated the subscreen and program logic, you can dive into additional customizing to make your screen appear in standard transactions.

2.4.4 Defining and Activating Partner Products in Transaction FIBF

To define and activate partner products in Transaction FIBF, follow these steps:

1. Open Transaction FIBF. Select Settings • Identification • Partner.
2. Add a new record with the Partner name "ZFIENH" and Name of Area "FI Enhancements."
3. Tick the Active flag.
4. Save the entries.
5. Select Settings • Products • ..of a Partner • Edit.
6. Add a new record with the following field values:
   - Product: "KNKK"
   - Partner: "ZFIENH"
   - Product Description: "Customer OPFI enhancement"
7. Select Settings • Products • ..of a Partner • Activate.
8. Add the record with the newly created product and partner.

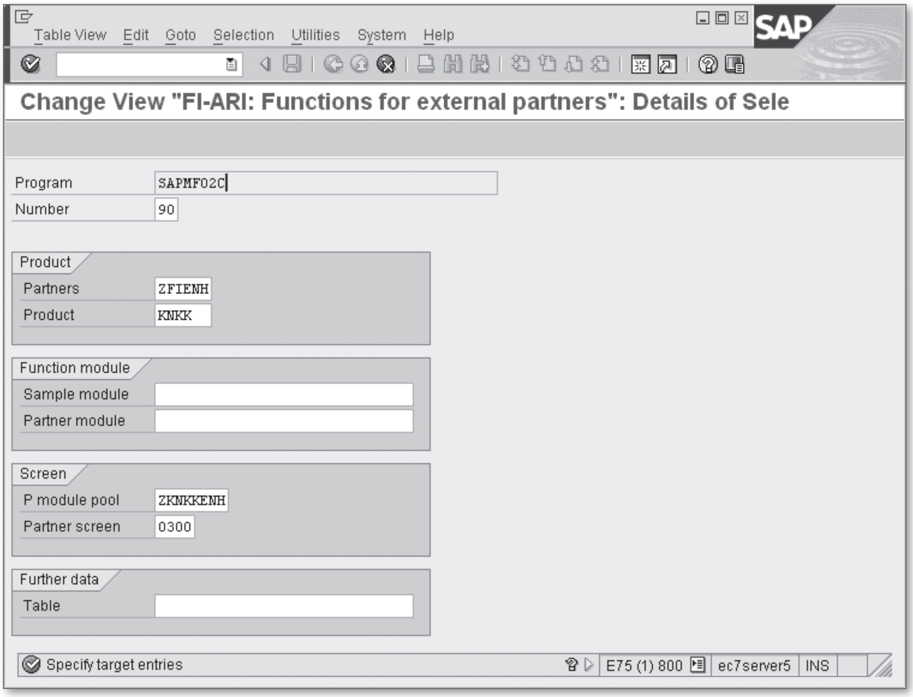
Note

As with the program name, product and partner names cannot exceed six characters.

2.4.5 Setting External Partner Functions

To set external partner functions, follow these steps:

1. After creating partner and product names, open the customizing table T061S in Transaction SM30. Before opening the table data for maintenance, the system shows this warning: "Do not make any changes (SAP data)." However, ignore this warning and proceed with editing.
2. Add an entry as shown in Figure 2.38.



Change View "FI-ARI: Functions for external partners": Details of Sele

Program	SAPMF02C
Number	90

Product

Partners	ZFIENH
Product	KNKK

Function module

Sample module	
Partner module	

Screen

P module pool	ZKNKKENH
Partner screen	0300

Further data

Table	

Specify target entries   E75 (1) 800   ec7server5   INS

Figure 2.38 Details of Table T061S Entry

3. Use the field values as shown in Table 2.4.

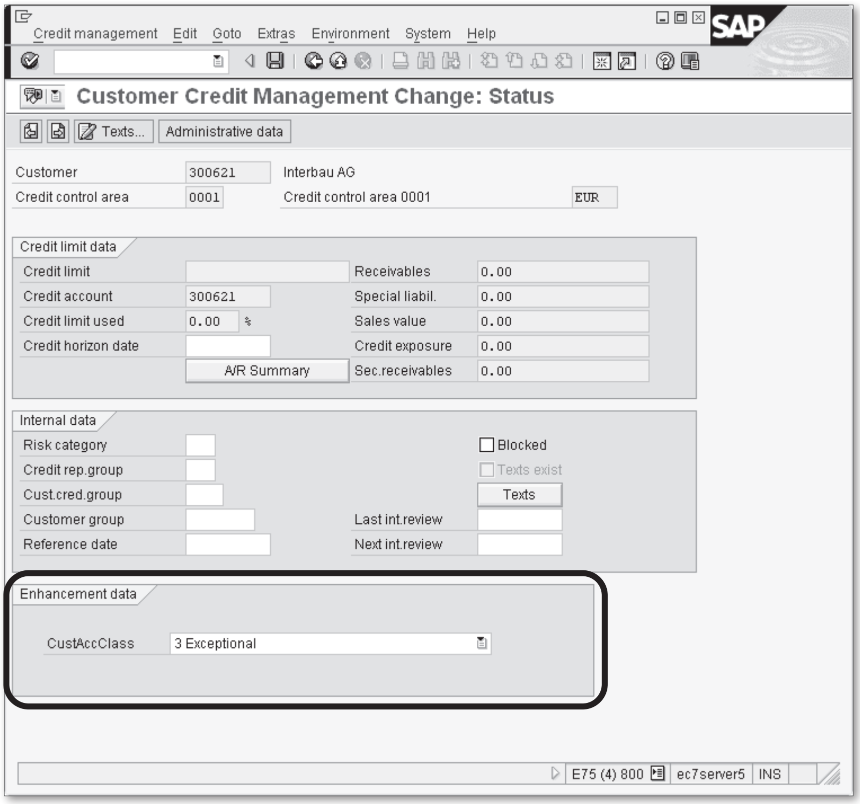
Field	Value
Program	SAMF02C
Number	90

Table 2.4 T061S Table Entry

Field	Value
Partners	ZFIENH
Product	KNKK
P Module Pool	ZKNKKENH
Partner screen	0300

Table 2.4 T061S Table Entry (Cont.)

4. Keep all other fields blank for now.
5. Save this entry, and open Transaction FD32 or FD33 with ticked Status check-box to see that the subscreen appears below the main screen. See the resulting screen in Figure 2.39.



Customer Credit Management Change: Status

Customer	300621	Interbau AG	
Credit control area	0001	Credit control area 0001	EUR

Credit limit data

Credit limit		Receivables	0.00
Credit account	300621	Special liabil.	0.00
Credit limit used	0.00 %	Sales value	0.00
Credit horizon date		Credit exposure	0.00
		Sec.receivables	0.00

Internal data

Risk category		☐ Blocked	
Credit rep.group		☐ Texts exist	
Cust.cred.group			
Customer group		Last int.review	
Reference date		Next int.review	

Enhancement data

CustAccClass: 3 Exceptional

E75 (4) 800 | ec7server5 | INS

Figure 2.39 Credit Management Status View with an Additional Subscreen

Note

Fields Partners and Product of table T061S have lengths of six characters, and field P Module Pool has a length of eight. This illustrates the technical restrictions for component names.

Number 90 is used in an effort not to interfere with possible SAP entries, keeping in mind threatening system warnings; later, you'll see how this number can be used in further enhancements.

Also note that the screen is fully functional; the value of the field `ZZCUST_CLASS` is saved and reported in change documents.

2.4.6 Further GUI Status Enhancement with Table T061V

If you open Table T061S in Transaction SE11, you'll notice the field FUNCP with a promising description: "FI-ARI: Partner function module name." The system uses it as a handler for additional function codes, which appear in GUI status after you properly customize the function texts in Table T061V (FI-ARI: Texts for external partner functions).

As most text tables do, T061V contains language code as a primary key component and also contains sequential numbers that must correspond to that of Table T061S.

Table T061V also belongs to SAP, just like Table T061S, and in the IDES system, it contains dozens of entries. However, those entries don't use sequential numbers greater than 3. That's why we used sequential number 90 in the Table T061S entry to minimize the possibility of interfering with SAP components.

Let's add an entry to the table for the English language (see Table 2.5).

Field	Value
Language key	EN
Program	SAPMF02C
Number	90
Name	Command Enhancement

Table 2.5 T061V Table Entry

Field	Value
Partners	ZFIENH
Product	KNKK

Table 2.5 T061V Table Entry (Cont.)

Now let's implement a function module to assign to the T061S table entry. Its interface must be the same as in Listing 2.14.

```
*"----------------------------------------------------------------------
*"*"Local Interface:
*"  IMPORTING
*"     VALUE(ADDRESS) TYPE  RF61H
*"     REFERENCE(IKNKK) TYPE  KNKK
*"     VALUE(PARTY) TYPE  TBE12-PARTY
*"     VALUE(PRDKT) TYPE  TBE22-PRDKT
*"     VALUE(LANGU) TYPE  SY-LANGU DEFAULT SY-LANGU
*"  EXPORTING
*"     REFERENCE(EKNKK) TYPE  KNKK
*"     VALUE(TDID) TYPE  TTXID-TDID
*"     VALUE(RCODE) TYPE  RF61B-RCODE
*"     VALUE(RTEXT) TYPE  RF61B-RTEXT
*"  TABLES
*"      TEXTLINES STRUCTURE  RF61T
*"----------------------------------------------------------------------
```

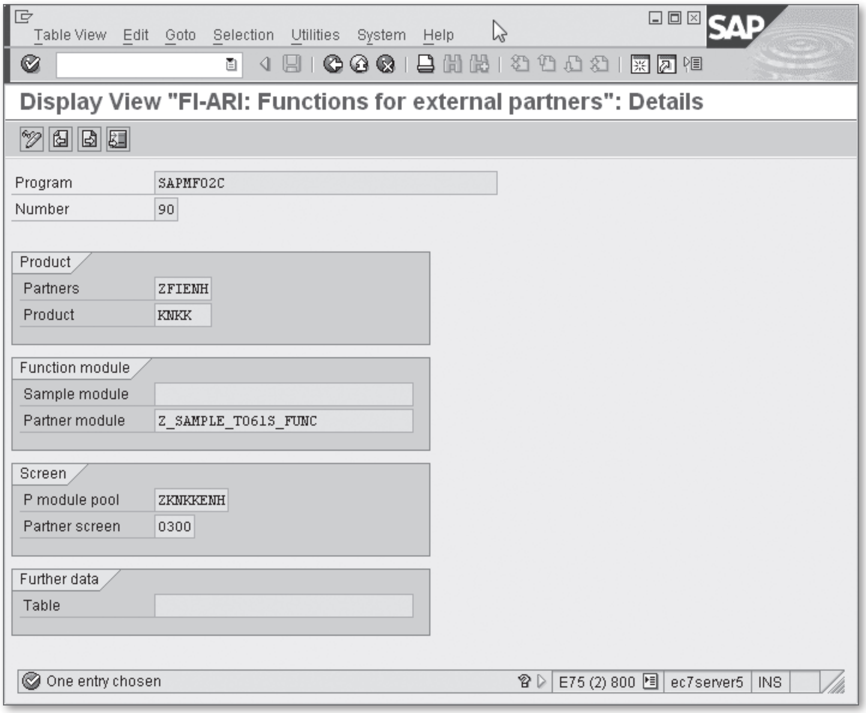
Listing 2.14 FI-ARI Function Module Interface for Customer Credit Control Data

The function module can change the contents of Table KNKK: We see exporting parameter `EKNKK` and also can provide an additional detailed text line supplying text identification via `TDID` and text lines via the `TEXTLINES` table parameter.

Note

Make sure you declare `IKNKK` and `EKNKK` parameters passing *by reference*; otherwise, you have to insert at least one assignment statement `EKNKK = IKNKK` into your function module. If you don't, you are at risk of data damage—the source structure will be cleared with uninitialized export parameter `EKNKK`.

RCODE and RTEXT act as return codes of the function. If it returns any value in either RCODE or RTEXT, the system does not update runtime data in Table KNKK after the call of the subscribed function module. See the details in the ECNN_OKCODE subroutine of the SAPMF02C module pool. In the IDES system, we create a sample function module Z_SAMPLE_T061S_FUNC and put its name into the field PARTNER MODULE of the T061S table entry (see Figure 2.40).



Display View "FI-ARI: Functions for external partners": Details

Program	SAPMF02C
Number	90

Product

Partners	ZFIENH
Product	KNKK

Function module

Sample module	
Partner module	Z_SAMPLE_T061S_FUNC

Screen

P module pool	ZKNKKENH
Partner screen	0300

Further data

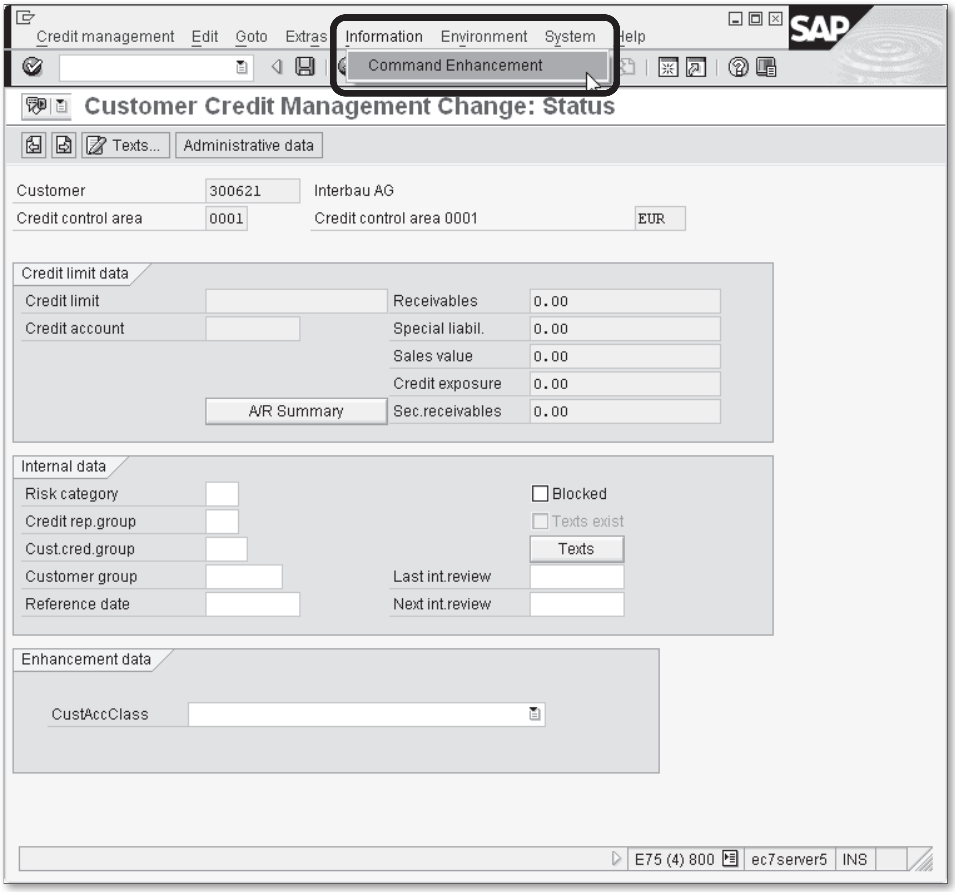
Table	

One entry chosen E75 (2) 800 ec7server5 INS

Figure 2.40 T061S Table Entry with Partner Function Module for the Customer Credit Control

After activating the function module and saving all necessary data into T061S and T061V tables, you can test Transactions FD32/FD33.

The menu INFORMATION has a submenu with the single entry COMMAND ENHANCEMENT. This function code is active only in status view together with the subscreen, which was added earlier (see Figure 2.41).


Credit management Edit Goto Extras Information Environment System Help
Command Enhancement
Texts...
Administrative data
A/R Summary
Texts

Customer Credit Management Change: Status

Customer	300621	Interbau AG	
Credit control area	0001	Credit control area 0001	EUR

Credit limit data

Credit limit		Receivables	0.00
Credit account		Special liabil.	0.00
		Sales value	0.00
		Credit exposure	0.00
		Sec.receivables	0.00

Internal data

Risk category		☐ Blocked	
Credit rep.group		☐ Texts exist	
Cust.cred.group			
Customer group		Last int.review	
Reference date		Next int.review	

Enhancement data

CustAccClass

E75 (4) 800 | ec7server5 | INS

Figure 2.41 Additional Menu Item in Credit Management Status View

2.4.7 Additional Credit Management Data User Exits

If your enhancement is more sophisticated (involving additional database tables update), you have to use BTE `00001520`. Subscribed functions of the event will only be called if event subscription `00001510` returns a CHANGED flag.

2.5 Accounts Payable (Vendors)

Now that we've discussed AR master record enhancement, it's a good time to turn to its counterpart: Accounts Payable (i.e., vendors). The architecture of vendor master enhancements looks almost the same as that of customer master enhancements. The obvious difference is that vendor maintenance transactions don't use tabstrip control; toolbar buttons (⇦ ⇨) are used to navigate a screen group. In the following subsections, we briefly review the main differences between the two techniques.

2.5.1 Screen and GUI Status Enhancement with Function Group FARI

Vendor master enhancements include the same technique as customer credit management, which is built on function group FARI and customizing tables T061S and T061V. Here, you can use the function group to add your own subscreen to the general control data screen of the vendor master.

You can also supply the function module name in the Table T061S entry, but the interface of the function will look different (see Listing 2.15).

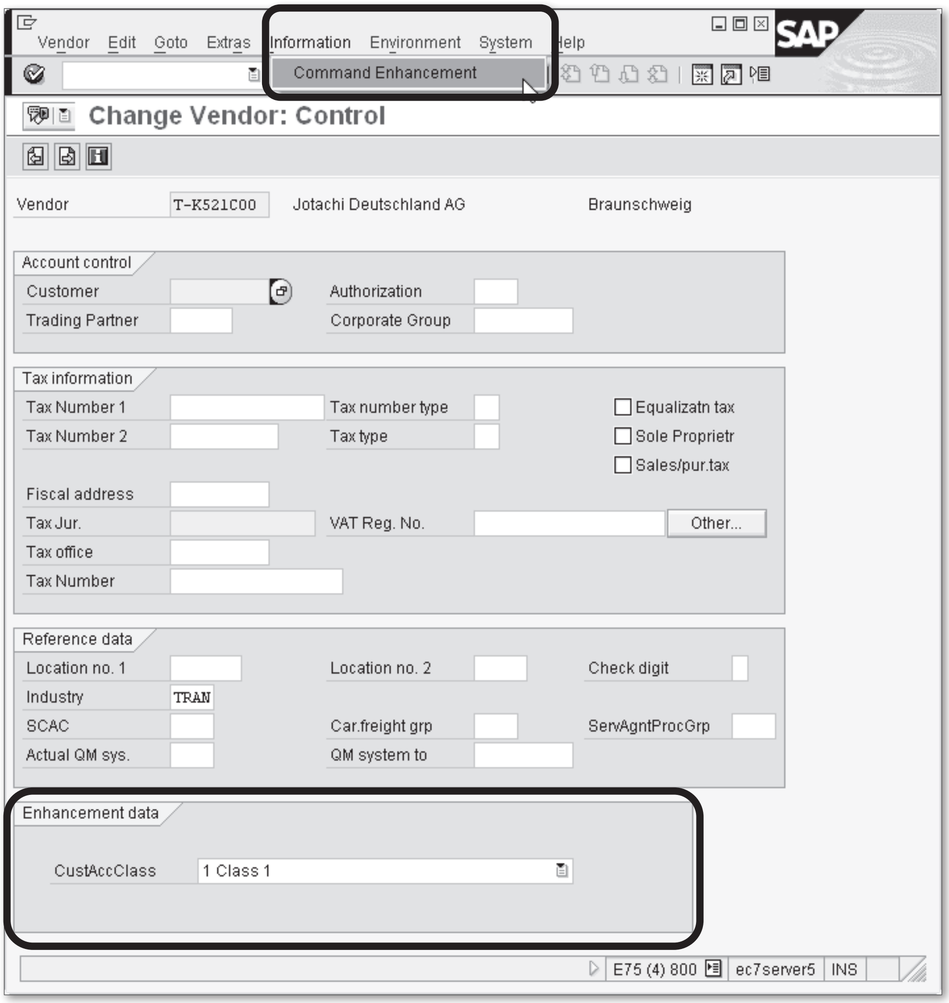
```
*"----------------------------------------------------------------------
*"*"Local Interface:
*"  IMPORTING
*"     VALUE(ADDRESS) TYPE  RF61H_K
*"     REFERENCE(ILFA1) TYPE  LFA1
*"     VALUE(PARTY) TYPE  TBE12-PARTY
*"     VALUE(PRDKT) TYPE  TBE22-PRDKT
*"     VALUE(LANGU) TYPE  SY-LANGU DEFAULT SY-LANGU
*"  EXPORTING
*"     REFERENCE(ELFA1) TYPE  LFA1
*"     VALUE(TDID) TYPE  TTXID-TDID
*"     VALUE(RCODE) TYPE  RF61B-RCODE
*"     VALUE(RTEXT) TYPE  RF61B-RTEXT
*"  TABLES
*"      TEXTLINES STRUCTURE  RF61T
*"----------------------------------------------------------------------
```

Listing 2.15 T061S Partner Function Interface for Vendors

Using the same steps as stated in Section 2.4, Customer Credit Management Data and Screen Enhancement, you can add a subscreen and additional menu entries to the vendor general control data screen. You can also use reduced program logic thanks to the `TABLES` declaration sharing.

To make the similar enhancement to the vendor master data in the IDES system, follow these steps:

1. Append Table LFA1 with the single field `ZZCUST_CLASS`.
2. Create module pool `ZLFB1ENH` with screen 300 by copying the previously created module pools (see Section 2.4).
3. Create function module `Z_SAMPLE_T061S_FUNC_K` with the required interface.
4. Use the necessary entries in Tables T061S and T061V.
5. Enhance the general control screen of the vendor master as shown in Figure 2.42.



Vendor	T-K521C00	Jotachi Deutschland AG		Braunschweig	
Customer		Authorization			
Trading Partner		Corporate Group			
Tax Number 1		Tax number type		☐ Equalizatn tax	
Tax Number 2		Tax type		☐ Sole Proprietr	
				☐ Sales/pur.tax	
Fiscal address					
Tax Jur.		VAT Reg. No.			
Tax office					
Tax Number					
Location no. 1		Location no. 2		Check digit	
Industry	TRAN				
SCAC		Car.freight grp		ServAgntProcGrp	
Actual QM sys.		QM system to			
CustAccClass	1 Class 1				

E75 (4) 800 | ec7server5 | INS

Figure 2.42 Enhanced Vendor General Data Screen with Additional Subscreen and Menu Entry

The corresponding table entries in T061S and T061V customizing tables are shown in Table 2.6 and Table 2.7.

Field	Value
Program	SAPMF02K
Number	90
Partners	ZFIENH
Product	KNKK
Partner module	Z_SAMPLE_T061S_FUNC_K
P module pool	ZLFB1ENH
Partner screen	0300

Table 2.6 T061S Table Entry for Vendor Control Data

Field	Value
Language	EN
Program	SAPMF02K
Number	90
Name	Command Enhancement
Partners	ZFIENH
Product	KNKK

Table 2.7 T061V Table Entry for Vendor Control Data

The source code for module pool ZLFB1ENH looks much like ZKNKKENH (see Listing 2.16).

```
*&-------------------------------------------------------------------*
*& Module Pool       ZLFB1ENH
*&-------------------------------------------------------------------*

PROGRAM  zlfb1enh.

TYPE-POOLS: abap.

* Vendor General Control Data Enhancement
```

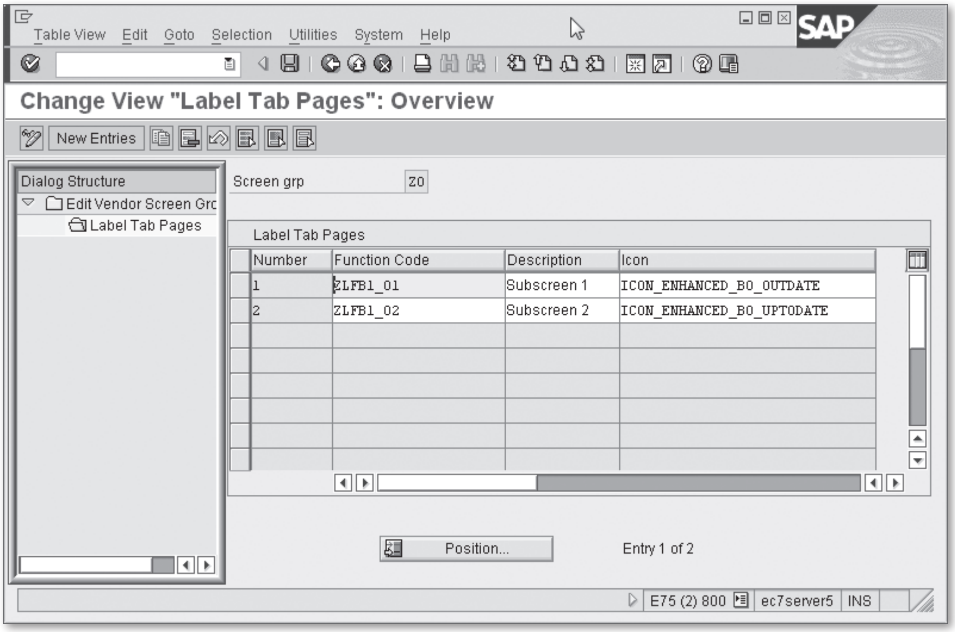
```
TABLES: lfb1, lfa1, t020.
*----------------------------------------------------------------------*
*  MODULE status_0300 OUTPUT
*----------------------------------------------------------------------*
*
*----------------------------------------------------------------------*
MODULE status_0300 OUTPUT.
  PERFORM modify_screen_0300.
ENDMODULE.                    "status_0300 OUTPUT

*&---------------------------------------------------------------------*
*&      Form  modify_screen_0300
*&---------------------------------------------------------------------*
*       text
*----------------------------------------------------------------------*
FORM modify_screen_0300.
  LOOP AT SCREEN.
    IF t020-aktyp = 'A'.
      screen-input = '0'.
    ELSE.
      screen-input = '1'.
    ENDIF.
    MODIFY SCREEN.
  ENDLOOP.
ENDFORM.                    "modify_screen_0300
```

Listing 2.16 ZLFB1ENH Source Code

2.5.2 BAdI Definitions

The main BAdI definitions are VENDOR_ADD_DATA and VENDOR_ADD_DATA_CS, which have almost the same interface as their customer counterparts. For vendor master data enhancement, complete the following steps:

1. In the IMG subtree Prepare Modification-Free Enhancement in Vendor Master Record under Vendor customizing, add screen group Z0 with two subscreens as shown in Figure 2.43.
2. Implement method CHECK_ADD_ON_ACTIVE of BAdI definition VENDOR_ADD_DATA, which reports screen group Z0 as active.



Screen grp Z0

Label Tab Pages

Number	Function Code	Description	Icon
1	ZLFB1_01	Subscreen 1	ICON_ENHANCED_BO_OUTDATE
2	ZLFB1_02	Subscreen 2	ICON_ENHANCED_BO_UPTODATE

Entry 1 of 2

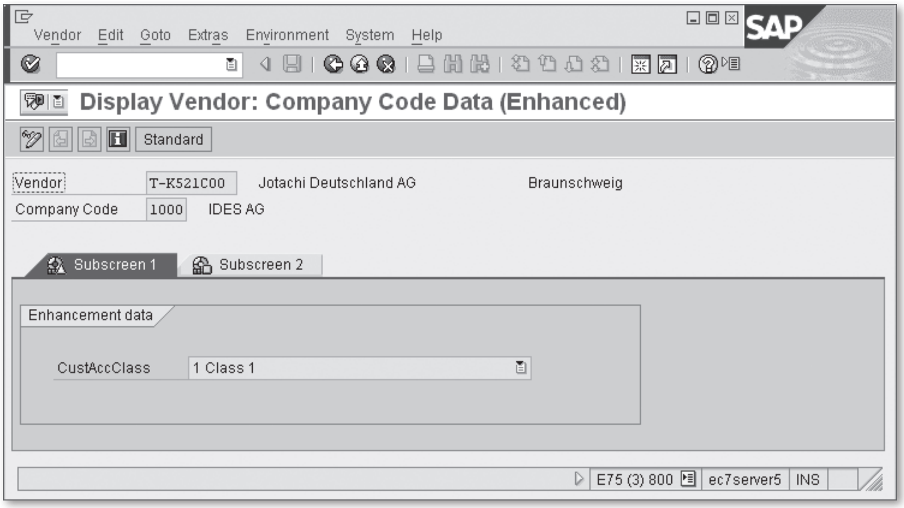
Figure 2.43 Vendor Data Screen Group Z0

3. Implement method `GET_TAXI_SCREEN` of the filtered BAdI definition `VENDOR_ADD_DATA_CS`, and assign screen group Z0 to the implementation filter value. In this method, you return `B` in the `E_HEADERSCREEN_LAYOUT` export parameter, which signals the main program that you are supposed to work with the company code view of the vendor. You also return module pool `ZLFB1ENH` and screen number `300`.

Note

Unlike the BAdI definition `CUSTOMER_ADD_DATA_CS`, here `E_HEADERSCREEN_LAYOUT` can take either `B` or `E` as possible values. `B` corresponds to the company code view of a vendor, while `E` is for the purchasing view.

After finishing all of these steps, you can add another view to the vendor master record, as shown in Figure 2.44.



Display Vendor: Company Code Data (Enhanced)

Vendor T-K521C00 Jotachi Deutschland AG Braunschweig
Company Code 1000 IDES AG

Enhancement data
CustAccClass 1 Class 1

Figure 2.44 Additional Tabstrip Control for Vendor Data

As you can see, additional screen groups are defined in the IMG for the vendor, which are displayed using the tabstrip control.

As you might remember from Section 2.3, Accounts Receivable (Customers), you have to implement more vendor BAdI definitions to achieve complete functionality of the enhancement. Necessary method implementations of the `VENDOR_ADD_DATA_CS` BAdI are listed here:

- `SET_DATA`
  Transfers data from the standard program to your enhancement together with current transaction mode (edit/create/display).
- `GET_DATA`
  Transfers data from your program back to the standard program.

You also need to implement the following method of the `VENDOR_ADD_DATA` BAdI:

- `CHECK_DATA_CHANGED`
  Tells the standard program that data was modified in the enhancement.

Other methods for these two BAdI definitions are optional and can be used in the same way as those of the similar customer BAdI definitions (for details refer to Section 2.3.6, Calling Moments of BAdI Methods).

2.5.3 Business Transaction Events

The set of available BTEs for the vendor master is also very similar to the customer's set (see Table 2.8).

Event Number	Calling Moment
00001460	Called in the PAI logic of the starting transaction screen after the user has entered transaction parameters (vendor number, company code, etc.). The event can be used to implement an additional authorization check. This event is called in all transaction modes; while others are only called in create and edit modes.
00001450	Called in the PAI logic of virtually all tabstrip screens in create or edit mode. This event allows for example change visibility of screen fields.
00001440	Called for final checks before saving (vendor number can be undefined for internal numbering).
00001410	Used as a handler for `OPFI` dynamic function code. May return the CHANGE flag for vendor data. If more than one function module is subscribed to the event, then the user requested to choose a particular add-on.
00001430	Called in the PBO logic of the starting transaction screen (see PBO module `TRANSAKTIONS_INIT` of module pool `SAPMF02K`). It allows transferring custom-defined function code text. If more than one function module is subscribed to the event, the `OPFI` function code gets the default text "Additional Components."
00001420	Called after the vendor data have been updated and *before* calling the `SAVE_DATA` method of the `VENDOR_ADD_DATA` BAdI.
00001421	Called after the vendor data have been updated and *after* calling the `SAVE_DATA` method of the `VENDOR_ADD_DATA` BAdI, and after the customer change documents update.

Table 2.8 Vendor Master BTE List

2.5.4 Function Module Exits

The vendor master has its own old-fashioned enhancement `SAPMF02K` (seen in Transaction SMOD) with the single function module component `EXIT_SAPMF02K_001`, which is called before saving vendor data. The function doesn't have exceptions in its interface (just like `EXIT_SAPMF02D_001` for customer), so you directly use the `MESSAGE` statement to show an error or warning.

2.6 Summary

As you can see, financial master data has a wide variety of user exits and allows developers to implement complex solutions, which can be tailored to the specific needs of their corporate business. At the same time, some enhancement techniques require tricky and dangerous developmental tricks such as using dynamic assign or editing data that belongs to SAP. These methods are not generally recommended. In such cases, you should thoroughly evaluate the necessity of the enhancement, the pros and cons, and the possibility of a negative impact. The presence of different enhancement techniques, which can seem redundant, are the results of long evolution and development of the SAP ERP system.

In the next chapter, we'll consider methods and tools to intervene into probably the most sensitive process of any ERP system: posting to Financial Accounting—the process where the system counts money.

Accounting documents represent a financial transaction, which is the act of transferring an amount of money from one or more accounts to one or more other accounts. In this chapter, we briefly discuss the technical structure of accounting documents and main database tables, which is where transactional information is stored. Then we will consider in detail the processing of accounting documents with available user exits.

3 Posting to Accounting

This chapter begins with the technical structure of the accounting document: tables and their relations followed by how accounting data can be enhanced. After that, we dive into program logic and walk through the process of enhancing the logic of accounting document posting both in dialog transactions and programmatically. We also touch on some internal techniques that SAP uses to update other application-specific data during posting.

3.1 The Technical Structure of an Accounting Document

If you are familiar with the common practice of representing business documents in a RDBMS (Relational Database Management System), then you might expect that almost every document model consists of at least two tables: a header table and an item table, where multiple items correspond to one header. However, the accounting document model in the SAP ERP system is represented with many more tables, which sometimes contain redundant and duplicated data. This is partly done for performance reasons but is mostly a consequence of the long evolution of the SAP system.

In the SAP ERP system, an accounting document has a compound key, consisting of the company code, document number, and fiscal year. The accounting primary key can be confusing for those new to this subject, and it has an impact on programming practice in accounting. The following is the main rule of this ABAP niche:

Whenever you develop a `SELECT` *statement against financial transaction data in SAP ERP, always check that you have included all three key fields in the* `WHERE` *clause.*

Note

Although the following anecdote is not an encouraging way to start this topic, it should help you understand the importance of accurately formulating SQL queries.

A disastrous error in one FI implementation dealt with a missing key field in the `UPDATE` SQL statement: After discussing all possible alternatives, the client decided to directly update existing accounting transactions to implement some business requirements. The updating report was implemented and then thoroughly tested in the Q & A system. The trouble was that the Q & A system was not identical to the production and contained only one company code. Additionally, the `UPDATE` statement in the report did not contain the company code in its `WHERE` clause (remember: the accounting document key consists of three fields). After the report was run, the disastrous results were not noticed at once, so in about a month, the company had to restore the data from quite an old backup file, which resulted in a sleepless month for the accounting department.

After that optimistic note, let's dive into the technical representation of an accounting document in the database. We'll discuss how the system stores a single document and what tables are used to represent aggregate (total) values in accounting.

3.1.1 The Header

The header table of the accounting document is BKPF, which contains general information about the document (e.g., company code, fiscal year, posting date, document currency, etc.).

Note

When developing user exits, you often need to know the source of the accounting document (for example, if it came from the purchasing or sales departments). While the accounting document header contains the original transaction code where the document was created (field BKPF-TCODE), it is not always correct, as sometimes the document can be generated automatically. For this reason, it's better to analyze fields BKPF-AWTYP and BKPF-AWKEY. The former contains an application specific code characterizing the source application, and the latter is the key of the source document. For example, the RMRP code in the BKPF-AWTYP field tells us that the document originated from the invoice verification process; in that case, the field BKPF-AWKEY contains the full number of the incoming invoice (document number + fiscal year).

3.1.2 Items

Raw document item data are stored in several cluster tables. A *cluster table* is a special kind of database table that isn't visible in RDBMS, as opposed to transparent tables. A cluster contains one or more tables with the same primary key and different data field structure. The most common examples of SAP cluster tables are listed here:

- **BSEG**
  Accounting line-item data (most common table in SAP ERP).
- **KONV**
  Pricing condition data.
- **CDPOS**
  Change document data.

Technically, a cluster is a table with a group of key fields common to all of the cluster member tables and some additional control fields specific to cluster administration. The actual data are stored in a long character-typed field.

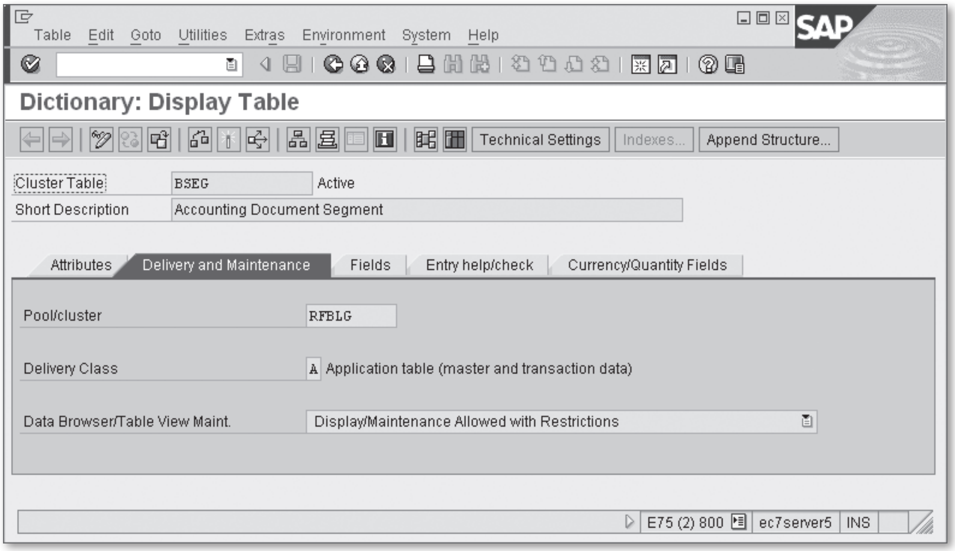
You should be aware of one main restriction and one main recommendation when working with cluster tables: You can't use such tables in a `JOIN`, and you must strictly use the full primary key when selecting data from the clustered table, or you might experience poor performance behavior.

Note

In SAP ERP, Table BSEG contains more than 300 data fields. The number of fields depends on which enhancement package is installed and which customer enhancements are implemented.

If you open Table BSEG in Transaction SE11 and then click the Delivery and Maintenance tab, you'll see that Table BSEG is assigned to cluster RFBLG (as shown in Figure 3.1).

Through the Repository Information System, which is accessible from virtually every ABAP Workbench transaction, we find that cluster RFBLG contains the tables listed in Table 3.1.



Dictionary: Display Table

Cluster Table: BSEG Active
Short Description: Accounting Document Segment

Pool/cluster: RFBLG
Delivery Class: A Application table (master and transaction data)
Data Browser/Table View Maint.: Display/Maintenance Allowed with Restrictions

E75 (2) 800 | ec7server5 | INS

Figure 3.1 Delivery and Maintenance Tab of Table BSEG

Name	Description
BSEC	One-Time Account Data Document Segment
BSED	Bill of Exchange Fields Document Segment
BSEG	Accounting Document Segment
BSES	Document Control Data (Obsolete)
BSET	Tax Data Document Segment

Table 3.1 Tables of Cluster RFBLG

Table BSEG contains a huge variety of line-item information such as account number, posting key, transaction amount, and many additional attributes of the account assignments transaction. Account assignments control the distribution of the amount to other components or subsystems (e.g., Cost Controlling, Funds Management, Asset Accounting, etc.).

If you look into the CURRENCY/QUANTITY FIELDS tab of Table BSEG in Transaction SE11, you'll notice dozens of amount fields. However, in most cases, you'll be dealing with two main amount fields:

- **WRBTR**
  Line-item amount represented in document currency.
- **DMBTR**
  Amount in local company currency.

You might expect that both of these amount fields must not be zero in an accounting document line item; however, there are situations when a document currency amount is zero, while the local currency amount is not. For example, this occurs when a company needs to count profit or loss depending on the currency rate change. Often such transactions occur at the end of the fiscal year.

Note

Do you know why the local currency amount field in an accounting document is named DMBTR? Because SAP originated in Germany, BTR stands for *betrag* (the German word for "amount") while DM stands for Deutsche Mark (German Mark, Germany's former currency). Most likely, in early releases of SAP, the Deutsche Mark was the only local currency for all clients.

Another meaningful cluster table is BSET, which contains tax data for the document. This cluster table contains the tax code, tax rate, tax amount, and tax account where the company counts its taxes. From the entry in Table BSET, you generally can't derive a single item to which this tax corresponds; there is no one-to-one correspondence between the tax and taxable items in an accounting document because the system sums up taxes with the same tax code and tax account and generates one totaling tax item for a group of taxable items.

Note

If you have SAP ERP with enabled Flexible General Ledger accounting (which is enabled by default), then you can use *splitting* to evaluate a particular tax proportion for each taxable item.

3.1.3 Parked Document Tables

In the SAP system, you can create a preliminary document called a parked document. This is a kind of a draft document that doesn't affect any accounting reports or account balances. A parked document can even have a nonzero balance; that is, the sum of all its credit line items doesn't have to be equal to the sum of its debit line items. Table 3.2 lists the set of tables for parked documents.

Name	Description
VBKPF	Document Header for Document Parking
VBSEC	Document Parking One-Time Data Document Segment
VBSEGA	Document Segment for Document Parking—Asset Database
VBSEGD	Document Segment for Customer Document Parking
VBSEGK	Document Segment for Vendor Document Parking
VBSEGS	Document Segment for Document Parking—General Ledger Account Database
VBSET	Document Segment for Taxes Document Parking

Table 3.2 Set of Tables for Parked Documents

3.1.4 Secondary Indices

As already mentioned, you can likely have serious performance problems if you don't use the entire key of accounting document when selecting data from cluster tables. At the same time, it's a common practice to build up reports using nonkey fields; for example, an accountant can be interested in selecting line items for a particular customer in a specific date interval.

To enable accounting data selection on a line-item basis with acceptable performance, SAP implemented another set of tables, which are transparent. The set is called *secondary indices*. Each line item has an attribute called *account type*, which characterizes the accounting area the account belongs to, for example, general ledger account, accounts receivable account (or customer account), or accounts payable account (vendor account). Account types in the accounting document are coded by the one-character field KOART: S for general ledger account, K for vendor account, and D for customer account.

Note

All available account types can be found in the KOART domain definition in Transaction SE11. The domain has a list of fixed values.

For these three account types, SAP implemented a set of six tables: a pair of tables for each account type.

- BSAS and BSIS for general ledger accounts
- BSAD and BSID for customer accounts
- BSAK and BSIK for vendor accounts

One table of each pair (with the letter *I* in its name) contains all line items with unclear liability (e.g., unpaid vendor invoice), and the other table of each pair (with letter *A*) contains those cleared.

Notice that each pair of tables has the same set of key fields, with two of them corresponding to the fact of debt clearance: AUGDT (date of clearance), AUGBL (clearance document number). Also notice that the AUGDT and AUGBL fields are always empty in the table with the letter *I*, and its counterpart has those fields filled with values. The union of each pair represents the whole set of line items of the corresponding account type.

The most standard accounting line-item reports are built using these tables. See, for example, the structure of such logical databases as SDF, KDF, and DDF, which are mostly used in standard SAP reports.

3.1.5 Total Tables

Another practical kind of accounting report reflects the different types of a total report: the account balance for a fiscal period or the whole year, the reports for comparing figures of different fiscal periods, and so on.

Mathematically, it's sufficient to have just the main accounting document tables (header and items) to build any fiscal period report—adding amounts item by item. But remember, in this case, to calculate the opening account balance, you have to sum up all of the account line items from the very beginning. Knowing that an ordinary SAP client can produce a few million transactions each year, you can imagine that the calculation of an account opening balance becomes a mission impossible from the performance point of view. For this reason, SAP delivers a variety of summation tables storing totals.

As of SAP ERP 6.0, SAP introduced a significant extension of accounting technology known as the *Flexible General Ledger*. From a technical point of view, this is an evolutionary step of Special Ledger technology known at least from SAP R/3 3.0. The Flexible General Ledger solution simplifies extending accounting with additional dimensions and ledgers, thus helping customers build complex accounting methodology (e.g., parallel accounting). We'll use *classic ledger* to refer to all of the legacy accounting tables and techniques used long before the appearance of the Flexible General Ledger.

The next subsection shows how total figures are stored in the system.

Total Tables of the Classic Ledger

Now let's see how the SAP system stores total figures in Financial Accounting (FI). We'll consider how totals are stored in the classic ledger and also in the new Flexible General Ledger.

GLT0

Each record of the GLT0 table contains either credit or debit total amounts for each period of one fiscal year for one company code account. Fields with the suffix VT (HSLVT, TSLVT, and others) hold the opening value for the fiscal year. If you select the record with the same key values and previous fiscal year, the sum of the VT field and all other 16 period fields gives the value of the next year. Therefore, when calculating an account balance for a given period of a particular fiscal year, you don't have to select additional GLT0 records for that account.

Amounts are stored as groups of homogeneous fields: one numbered field for each fiscal year period. You can see fields from HSL01 to HSL16, and from TSL01 to TSL16. Why 16? In accounting, a year is divided into 16 months. Actually there are 12 periods for 12 calendar months and also 4 additional periods, for end-of-year reconciliation work, which normally is done manually by accountants. In the end of a year when the 12th period is closed, no financial data can be sent from other SAP ERP components (Sales, Procurement, etc.); only accountants can post accounting documents manually, entering fiscal periods from 13 to 16.

Also notice that each GLT0 record contains a currency code as the primary component. This is document currency. Amount fields with names starting with TSL represent amounts in document currency. Other groups of fields have names starting with HSL. These fields represent amounts in the company code home currency.

The structure of GLT0 obviously violates relational normal forms, so you can't calculate the sum of period fields using pure SQL. To effectively operate with such structures, you should use special aided ABAP constructs: Cycling statements with

`ASSIGN …INCREMENT` (or obsolete `DO … VARYING`, and `WHILE … VARY`) can iterate through a group of homogeneous fields.

Note

One of the annoying examples of negligent FI/CO programming techniques is calculating a fiscal period by extracting an `MM` component from a `YYYYMMDD` internal date representation. Always convert the calendar date into a fiscal period with SAP function `DATE_TO_PERIOD_CONVERT`, even if in your company the fiscal year is equal to the calendar year. You can never predict business or legislation change; for example, at one moment or another your corporation can decide to open a branch in the USA with another company code, with the fiscal year not equal to the calendar year.

LFC1, KNC1

Total values for customers and vendors are also gathered into special tables: LFC1 for vendors and KNC1 for customers. There are also repeating groups of amount fields, although here you can see that each table record contains debit and credit values for each fiscal period, unlike GLT0 where the *Debit/Credit* indicator is a component of the primary key.

Total Tables of the Flexible General Ledger

Note

According to official SAP help, the Flexible General Ledger solution was first introduced in SAP R/3 4.6b; however, this was a pilot implementation without many modern capabilities such as document splitting, among others.

The SAP Flexible General Ledger can be switched on and off, and it's turned on by default. Using the Flexible General Ledger, several ledgers can be created with different dimensions specific to a particular accounting methodology. Technically, every Flexible General Ledger consists of several database tables for items and total values. SAP delivers only one active main ledger (0L), which always exists. And its database total table is FAGLFLEXT.

The name of a total table for a given ledger can be found in field TAB of customizing table T881. This is where attributes for all ledgers defined in the system (not only Flexible General Ledgers) are stored. You can distinguish Flexible General Ledgers from all others by whether there's a value in the GLFLEX field. Flexible General Ledgers have a value in this field, and other ledgers do not.

The structure of any ledger total table is similar to that of Table GLT0, with a number of dimension fields and repeating amounts for every period. Keep in mind, however, that the Flexible General Ledger (just as its predecessor the Special Ledger) has various dimension fields and up to 366 periods for daily ledgers. It is up to the user to create a ledger with, for example, weekly periods. Therefore, you should not make any assumptions on the amount field count in a Flexible General Ledger total table. Fortunately, SAP always stores the number of a ledger's periods for each record in the RPMAX field, so it's better to use this field in all processing routines, including GLT0, thus making your program logic more robust. Listing 3.1 shows a period amounts processing code snippet.

```
DATA: ls_faglflext TYPE faglflext.

FIELD-SYMBOLS: <tsl> TYPE faglflext-tslvt,
               <hsl> TYPE faglflext-tslvt.

*  SELECT * FROM faglflext INTO ls_faglflext
*           WHERE...
*
*      To be implemented...
*
*  ENDSELECT.

DATA inc TYPE i.

WHILE sy-subrc = 0.
  inc = sy-index—1.
  ASSIGN ls_faglflext-tsl01 INCREMENT inc TO <tsl>
                            CASTING RANGE ls_faglflext.
  ASSIGN ls_faglflext-hsl01 INCREMENT inc TO <hsl>
                            CASTING RANGE ls_faglflext.

*    To be implemented...

ENDWHILE.
```

Listing 3.1 FAGLFLEXT Record Amounts Processing

3.2 Core Program Modules of Accounting

The core functionality of the accounting component is gathered in package FBAS. Here you can find almost all module pools, reports, and function groups. From a technical point of view, an accounting document can be created in the system via two main methods: dialog transactions or program logic.

Among these programs, two main modules are module pool `SAPMF05A` and function group `RWCL`. Module pool `SAPMF05A` is an implementation of all of the main dialog transactions for entering accounting documents, whereas function group `RWCL` is in the background when other SAP functional components need to post values to accounting.

In the following subsections, we'll discuss how to enhance user interaction in classic transactions and also newer Enjoy transactions.

3.2.1 Screen Enhancement of Accounting Posting Transactions

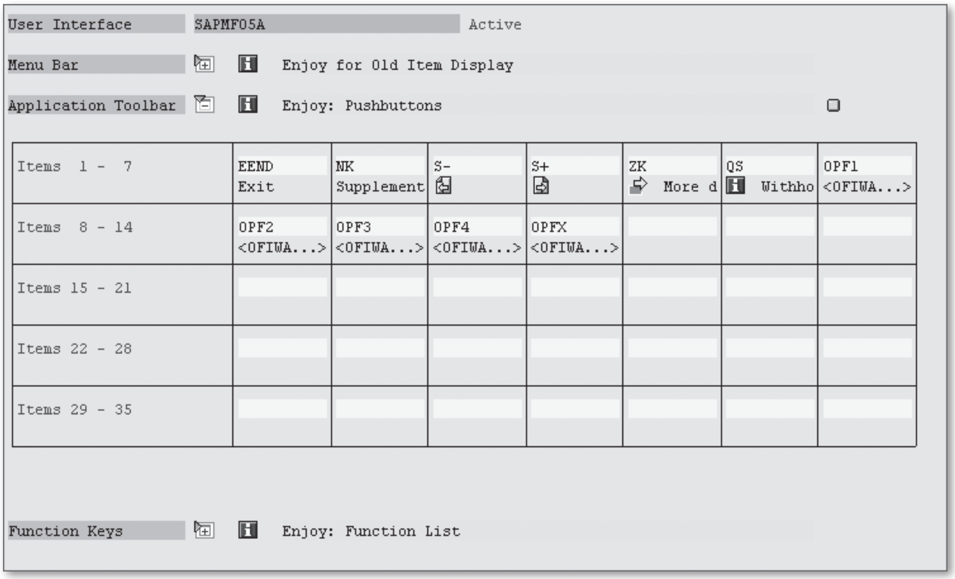
The two kinds of dialog transactions for posting accounting documents are old style transactions (e.g., F-02, F-42, etc.), and Enjoy transactions (e.g., FB50, FB60, etc.). Both kinds of transactions have screen and UI enhancement capabilities, although Enjoy transactions possess more enhancement features.

In the next subsection, we'll show how to extend the GUI status of posting transactions using BTEs.

GUI Status Enhancement with Open FI

The GUI status enhancement looks like additional entries in the menu and toolbar. For example, you can open GUI status ZBE of module pool `SAPMF05A` shown in Figure 3.2. There you can see several function codes (`OPF1`, `OPF2`, etc.) that reference dynamic text. Also, there is an additional dynamic function code `OPFI` in the menu bar.

In old-style transactions, status enhancements are available only in document line-item view, whereas in Enjoy transactions, they are visible both in document overview and line-item view. You should also notice that GUI status enhancements are not available in document edit transactions.



User Interface SAPMF05A Active

Menu Bar Enjoy for Old Item Display

Application Toolbar Enjoy: Pushbuttons

Items 1 - 7	EEND Exit	NK Supplement	S-	S+	ZK More d	QS Withho	OPF1 <OFIWA...>
Items 8 - 14	OPF2 <OFIWA...>	OPF3 <OFIWA...>	OPF4 <OFIWA...>	OPFX <OFIWA...>			
Items 15 - 21							
Items 22 - 28							
Items 29 - 35							

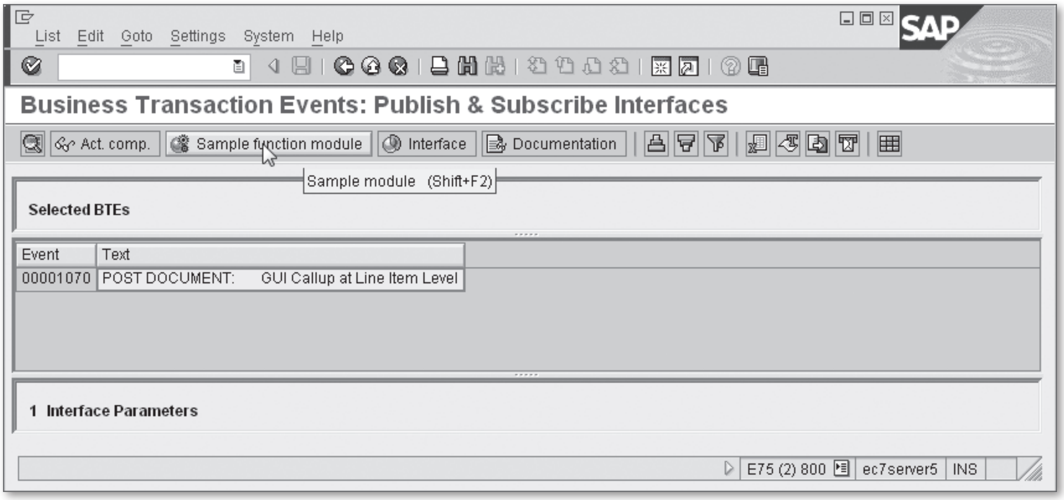
Function Keys Enjoy: Function List

Figure 3.2 Additional Function Codes in GUI Status ZBE of Module Pool SAPMF05A

To make additional function codes active, you have to implement two BTEs: `00001070` and `00001080`. The first event implements a particular action you want to perform when a user clicks a corresponding button; the second is a tool for transferring your application-specific caption for an additional button.

First, you have to create two function modules, which will then be subscribed to the events. The most convenient way of creating function modules for BTEs is copying the sample functions. Sample interface functions can be found in the Open FI information system in Transaction FIBF. Follow the menu path ENVIRONMENT • INFO SYSTEM (P/S). The resulting screen of this report for BTE `00001070` is shown in Figure 3.3. By clicking the SAMPLE FUNCTION MODULE button, you will be taken to Transaction SE37, which shows the required function module that you can copy to your own.

In the IDES system, we implemented both functions in the simplest possible way (see the source in Listing 3.2 and Listing 3.3). As you can see, we just transfer some predefined text into a button caption and show an information message as a reaction to the button click.



Event	Text
00001070	POST DOCUMENT: GUI Callup at Line Item Level

Figure 3.3 Business Transaction Events Information System Screen

```
FUNCTION z_sample_interface_00001080.
*"----------------------------------------------------------------------
*"*"Local Interface:
*"  IMPORTING
*"     REFERENCE(I_SPRAS) LIKE  SY-LANGU
*"     REFERENCE(I_AKTYP) TYPE  AKTYP
*"     REFERENCE(I_DYNCL) TYPE  DYNCL
*"  EXPORTING
*"     VALUE(E_FTEXT) LIKE  FTEXTS-FTEXT
*"----------------------------------------------------------------------
  e_ftext = 'FI Sample Enhancement'.
ENDFUNCTION.
```

Listing 3.2 00001080 BTE Implementation

```
FUNCTION z_sample_interface_00001070.
*"----------------------------------------------------------------------
*"*"Local Interface:
*"  IMPORTING
*"     REFERENCE(I_BKPF) TYPE  BKPF
*"     REFERENCE(I_BSEG) TYPE  BSEG
*"     REFERENCE(I_AKTYP) TYPE  AKTYP
*"     REFERENCE(I_DYNCL) TYPE  DYNCL
*"  EXPORTING
*"     REFERENCE(E_XCHNG) LIKE  OFIWA-XCHNG
*"----------------------------------------------------------------------
  MESSAGE 'Sample BTE event 00001070 implementation' TYPE 'I'.
ENDFUNCTION.
```

Listing 3.3 00001070 BTE Implementation

The 00001070 BTE has an export parameter `E_XCHNG`, which you might assume serves as a modification flag. However, it actually is redundant because the event is called only in the document-creation process, so there is no need to transfer the modification flag to the main program.

If you plan to employ event `00001070` for requesting additional data from a user, you have to provide functionality for saving your data together with the accounting document. As a rule, the data might be dependent on an accounting document number. Be aware that at the moment of calling event `00001070`, the document number is still undefined. The document number becomes available only in BTE `00001030`, which we'll discuss later.

Note

BTEs `00001070` and `00001080` are interdependent. The enhanced button caption becomes visible only if there are active subscriptions to both events `00001070` and `00001080` with the same *customer product*. You can also define P&S modules for a *partner product*; in this case, subscriptions must have the same sequential number in both events.

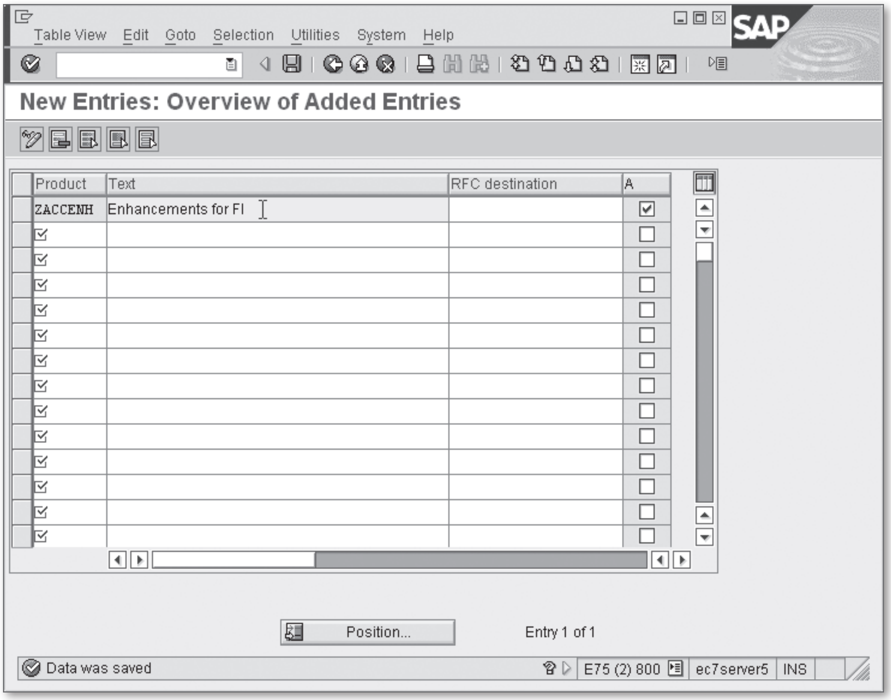
Both events have an import parameter of type `BSEG`, which corresponds to a document line item. When fired on the overview screen of an Enjoy transaction (e.g., FB60 or FB70), the system transfers to the event data of the last added line item.

Creating Products

Before linking (or subscribing) newly created function modules to BTEs, you have to create and activate a *product* in Transaction FIBF. The product is just a code for an arbitrary set of BTEs and/or process implementations. You maintain product codes and their activation status in Transaction FIBF using menu path SETTINGS • P/S MODULES • ... OF A CUSTOMER.

Linking Function Modules and BTEs

After creating function modules, you have to subscribe them to the events. In the IDES system, we created and activated a product of customer ZACCENH (see Figure 3.4).



New Entries: Overview of Added Entries

Product	Text	RFC destination	A
ZACCENH	Enhancements for FI		✓

Entry 1 of 1

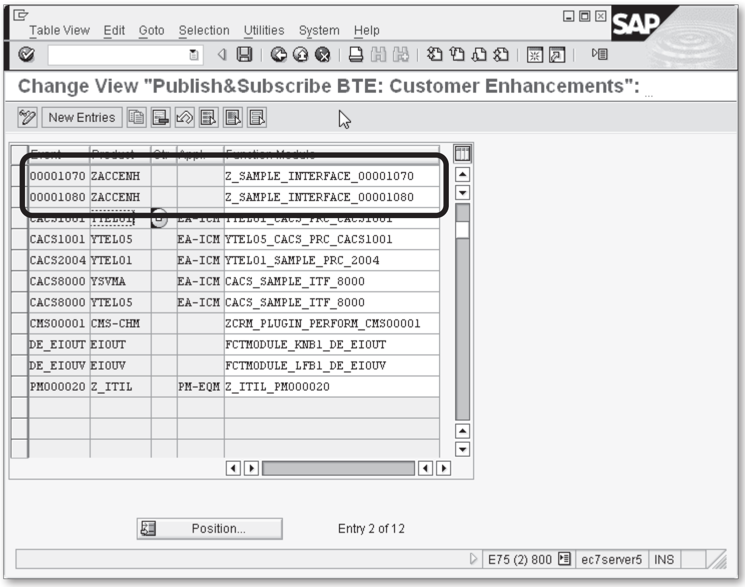
Data was saved

E75 (2) 800 | ec7server5 | INS

Figure 3.4 Customer Product ZACCENH

Here you define records for events `00001070` and `00001080` as shown in Figure 3.5.

After implementing function modules and linking them to both events in the IDES system, the resulting FB60 screen looks like Figure 3.6. The same additional button and menu also can be seen in old-style Transaction F-43 (see Figure 3.7).

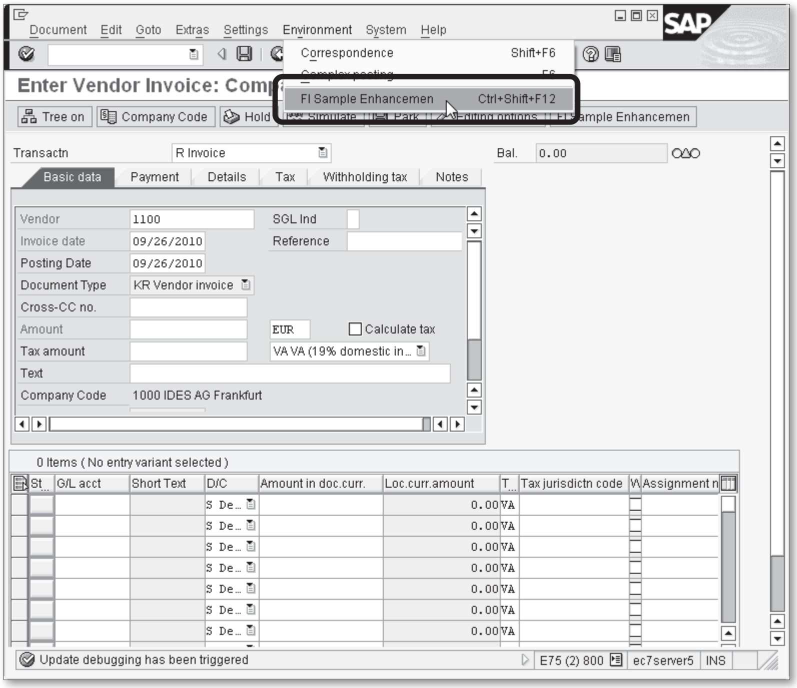


Change View "Publish&Subscribe BTE: Customer Enhancements":

Event	Product	Ctr	Appl.	Function Module
00001070	ZACCENH			Z_SAMPLE_INTERFACE_00001070
00001080	ZACCENH			Z_SAMPLE_INTERFACE_00001080
CACS1001	YTEL01		EA-ICM	YTEL01_CACS_PRC_CACS1001
CACS1001	YTEL05		EA-ICM	YTEL05_CACS_PRC_CACS1001
CACS2004	YTEL01		EA-ICM	YTEL01_SAMPLE_PRC_2004
CACS8000	YSVMA		EA-ICM	CACS_SAMPLE_ITF_8000
CACS8000	YTEL05		EA-ICM	CACS_SAMPLE_ITF_8000
CMS00001	CMS-CHM			ZCRM_PLUGIN_PERFORM_CMS00001
DE_EIOUT	EIOUT			FCTMODULE_KNB1_DE_EIOUT
DE_EIOUV	EIOUV			FCTMODULE_LFB1_DE_EIOUV
PM000020	Z_ITIL		PM-EQM	Z_ITIL_PM000020

Entry 2 of 12

Figure 3.5 BTE Linkage for Events 00001070 and 00001080



Enter Vendor Invoice: Company Code 1000

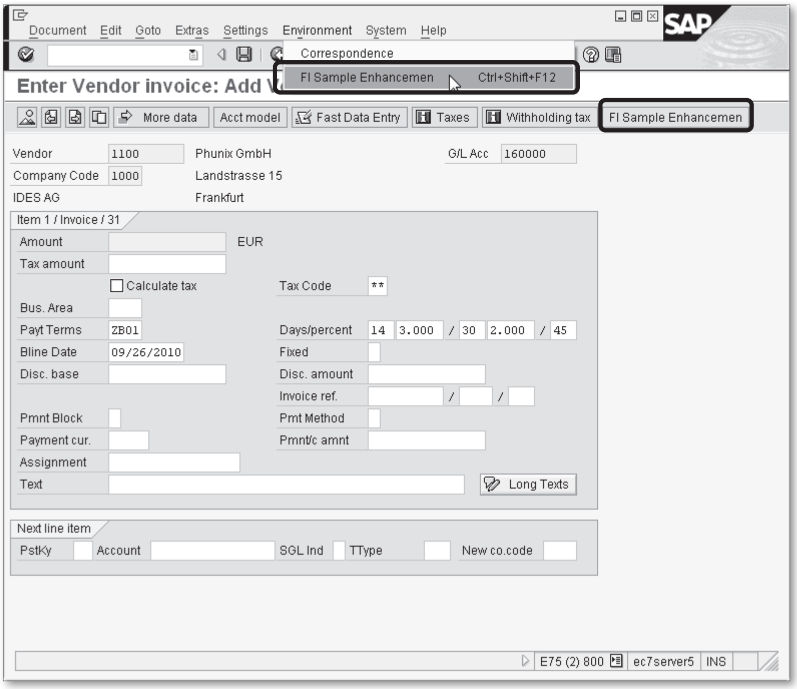
Transactn	R Invoice
Bal.	0.00
Vendor	1100
SGL Ind	
Invoice date	09/26/2010
Reference	
Posting Date	09/26/2010
Document Type	KR Vendor invoice
Cross-CC no.	
Amount	EUR
Calculate tax	☐
Tax amount	VA VA (19% domestic in...
Text	
Company Code	1000 IDES AG Frankfurt

0 Items (No entry variant selected)

St...	G/L acct	Short Text	D/C	Amount in doc.curr.	Loc.curr.amount	T...	Tax jurisdictn code	W...	Assignment n
			S De...		0.00	VA			
			S De...		0.00	VA			
			S De...		0.00	VA			
			S De...		0.00	VA			
			S De...		0.00	VA			
			S De...		0.00	VA			
			S De...		0.00	VA			

Update debugging has been triggered

Figure 3.6 Additional Menu Entry and Toolbar Button in Transaction FB60



Enter Vendor invoice: Add V

Vendor 1100 Phunix GmbH G/L Acc 160000
Company Code 1000 Landstrasse 15
IDES AG Frankfurt

Item 1 / Invoice / 31

Amount		EUR	
Tax amount			
	☐ Calculate tax	Tax Code	**
Bus. Area			
Payt Terms	ZB01	Days/percent	14 3.000 / 30 2.000 / 45
Bline Date	09/26/2010	Fixed	
Disc. base		Disc. amount	
		Invoice ref.	/ /
Pmnt Block		Pmt Method	
Payment cur.		Pmnt/c amnt	
Assignment			
Text			

Next line item

PstKy Account SGL Ind TType New co.code

E75 (2) 800 ec7server5 INS

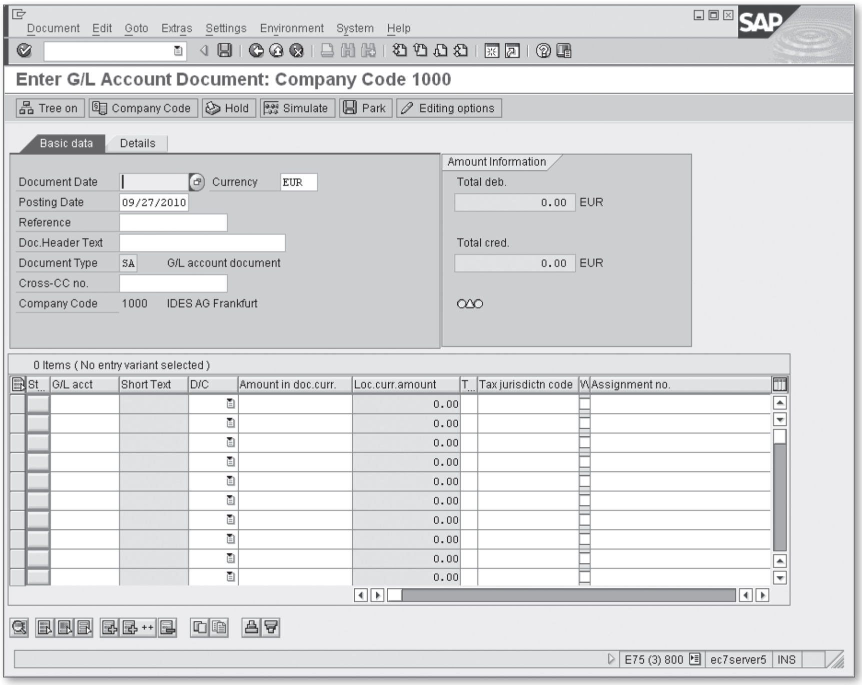
Figure 3.7 Line-item View in Transaction F-43

3.2.2 Screen Enhancement of General Ledger Posting Enjoy Transactions with BAdI

You can use BAdI definition `FI_HEADER_SUB_1300` to add your custom-defined subscreen to the BASIC DATA tab of Transaction FB50's overview screen. Figure 3.8 shows the overview screen of Transaction FB50 without enhancement.

The bottom area of the BASIC DATA tab contains an empty subscreen that can be filled with BAdI implementations. For testing purposes, in the IDES system, we created simple subscreen 0300 in module pool `ZFB50ENH` containing just text labels.

If you open BAdI definition `FI_HEADER_SUB_1300` in Transaction SE18 and open the SUBSCREENS tab, you can see that the BAdI is linked to subscreen 1300 of module pool `SAPMF05A` (see Figure 3.9). Also notice that the BAdI is filter-dependent, and its filter value is a country ISO code (see the ATTRIBUTES tab).



Enter G/L Account Document: Company Code 1000

Document Date		Currency	EUR
Posting Date	09/27/2010		
Reference			
Doc.Header Text			
Document Type	SA	G/L account document	
Cross-CC no.			
Company Code	1000	IDES AG Frankfurt	

Total deb. 0.00 EUR

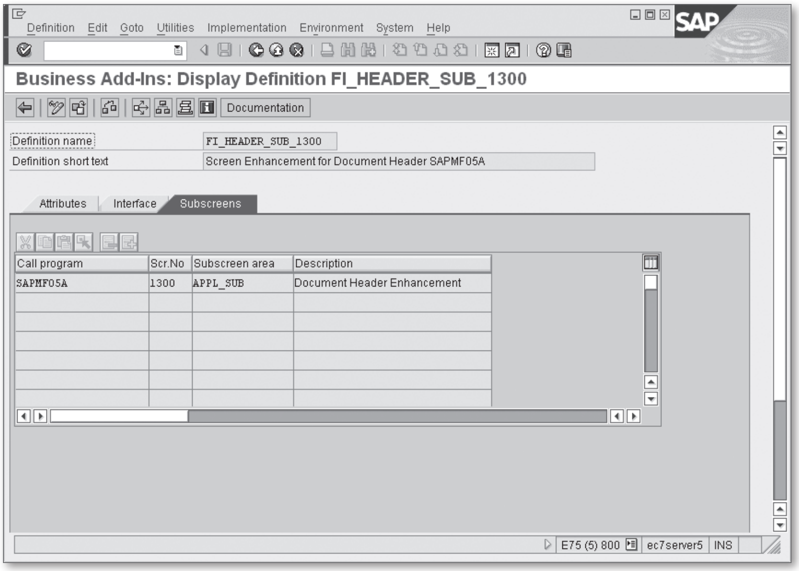
Total cred. 0.00 EUR

0 Items (No entry variant selected)

St	G/L acct	Short Text	D/C	Amount in doc.curr.	Loc.curr.amount	T.	Tax jurisdictn code	W	Assignment no.
					0.00				
					0.00				
					0.00				
					0.00				
					0.00				
					0.00				
					0.00				
					0.00				
					0.00				
					0.00				

E75 (3) 800 ec7server5 INS

Figure 3.8 Starting Screen of Transaction FB50


Definition Edit Goto Utilities Implementation Environment System Help
Documentation
Attributes
Interface
Subscreens

Business Add-Ins: Display Definition FI_HEADER_SUB_1300

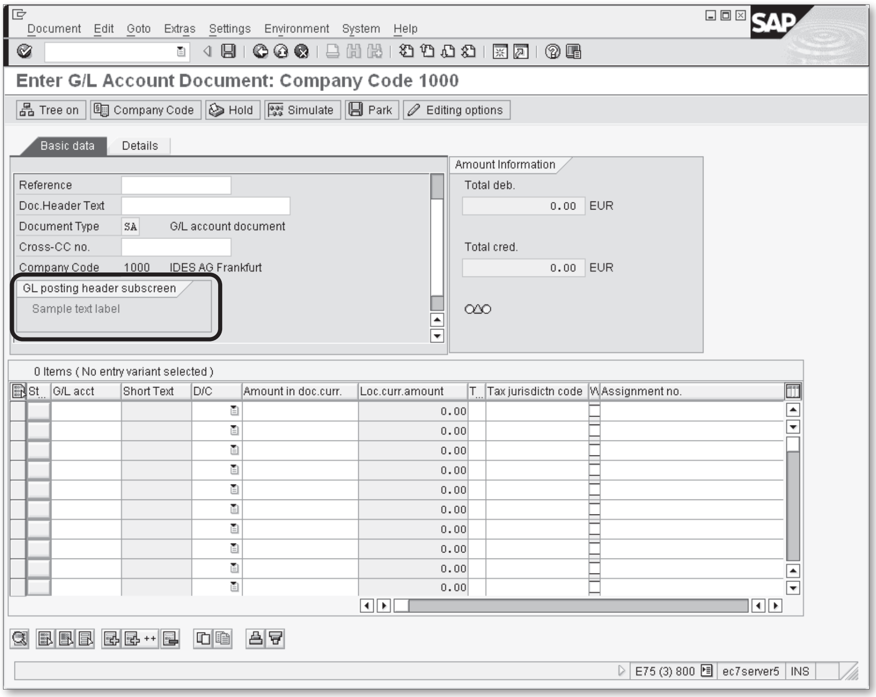
Definition name	FI_HEADER_SUB_1300
Definition short text	Screen Enhancement for Document Header SAPMF05A

Call program	Scr.No	Subscreen area	Description
SAPMF05A	1300	APPL_SUB	Document Header Enhancement

E75 (5) 800 ec7server5 INS

Figure 3.9 Subscreen Linkage of the BAdI Definition FI_HEADER_SUB_1300

As in the IDES system, we are testing enhancements using company code 1000 (well-known to SAP FI training attendees), which is assigned to Germany (ISO code DE); we also created an implementation of BAdI `FI_HEADER_SUB_1300` with filter value `DE`. Because the subscreen doesn't include any interaction logic but only static texts, we don't implement any interface methods of the BAdI. Nevertheless, we can successfully activate the implementation and check the results by opening Transaction FB50 (see Figure 3.10).



Enter G/L Account Document: Company Code 1000

Reference	
Doc.Header Text	
Document Type	SA G/L account document
Cross-CC no.	
Company Code	1000 IDES AG Frankfurt

GL posting header subscreen
Sample text label

Total deb. 0.00 EUR

Total cred. 0.00 EUR

0 Items (No entry variant selected)

St...	G/L acct	Short Text	D/C	Amount in doc.curr.	Loc.curr.amount	T...	Tax jurisdictn code	W	Assignment no.
					0.00				
					0.00				
					0.00				
					0.00				
					0.00				
					0.00				
					0.00				
					0.00				
					0.00				
					0.00				

E75 (3) 800 ec7server5 INS

Figure 3.10 Enhanced Overview Screen of Transaction FB50

Note that this BAdI implementation affects old-styled posting transactions such as FB01, FBD1, and FBD5.

If you need to implement data transfer between your own subscreen and the main program, then you have to implement the BAdI interface methods. Using Transaction FB50 as an example, these methods are called in the following moments:

- PUT_DATA_TO_SCREEN_PBO is called from within the PBO logic of the BASIC DATA tab screen (screen 1010 of the SAPMF05A program) before entering the PBO logic of your own screen.
- PUT_DATA_TO_SCREEN_PAI is called in the PAI logic of the 1010 screen just after checking and transferring basic document header data into the main program variables and before the PAI logic of your enhancement screen.
- GET_DATA_FROM_SCREEN_PAI method is called after executing your own screen PAI logic and before other standard field checks.

3.2.3 Screen Enhancement of Customer or Vendor Enjoy Transactions with BAdI

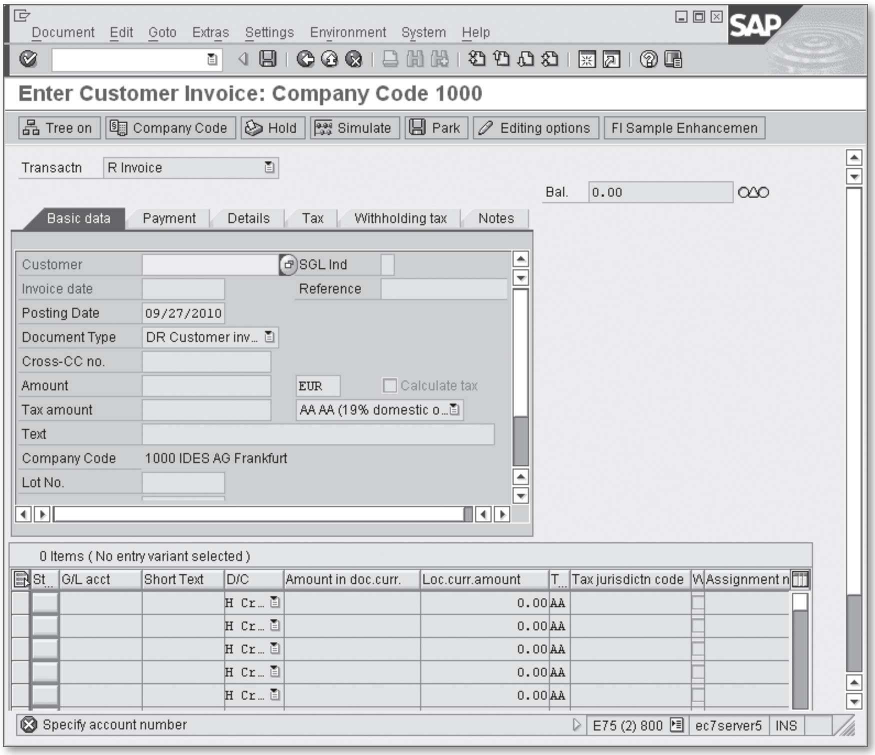
BAdIs can be used to add screen enhancements to Enjoy accounting transactions for entering customer or vendor documents such as FB60 (Incoming Invoice) or FB70 (Outgoing Invoice). Figure 3.11 shows the overview screen of Transaction FB60. The BASIC DATA tab is the area where you can implant your own subscreen. This area contains either screen 0510 for customer accounts (Transaction FB70) or 0010 for vendor accounts (Transaction FB60). Both subscreens are defined within function group FDCB, which implements screen data management tools in a variety of transactions, including accounting Enjoy transactions.

You can tailor your own subscreen to this area by implementing one of five BAdI definitions: from BADI_FDCB_SUBBAS01 to BADI_FDCB_SUBBAS05.

Note

The vendor subscreen 0010 of function group FDCB contains six dynamic subscreen areas, and there is a BAdI definition BADI_FDCB_SUBBAS06 for the sixth subarea. However, this BAdI can only be implemented by SAP itself and used only in the Materials Management Invoice Verification application.

All five BAdI definitions look the same; the only difference can be found on the SUBSCREENS tab in Transaction SE18. As you can see in Figure 3.12, the BADI_FDCB_SUBBAS01 definition is linked to the SUBBAS01 subscreen area. Other BAdI definitions are linked to the subscreen area with the corresponding number: BADI_FDCB_SUBBAS02 to SUBBAS02, BADI_FDCB_SUBBAS03 to SUBBAS03, and so on.



Enter Customer Invoice: Company Code 1000

Transactn R Invoice

Bal. 0.00

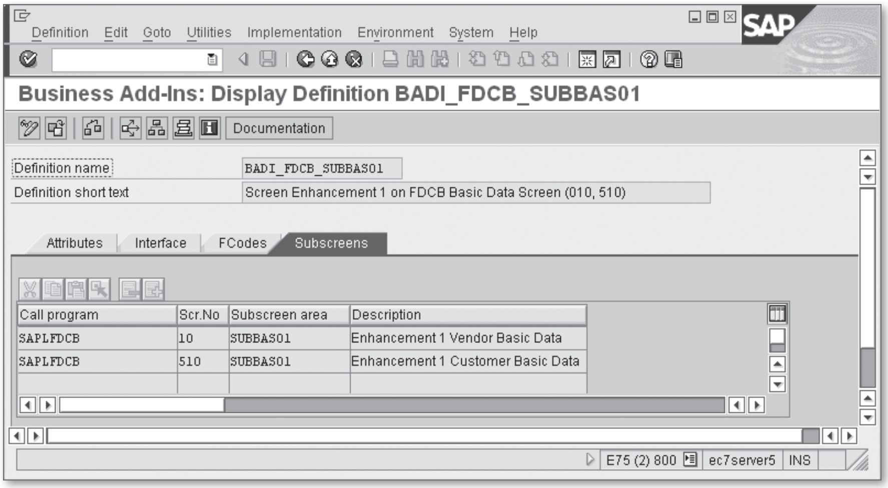
Customer		SGL Ind	
Invoice date		Reference	
Posting Date	09/27/2010		
Document Type	DR Customer inv...		
Cross-CC no.			
Amount		EUR	☐ Calculate tax
Tax amount		AA AA (19% domestic o...	
Text			
Company Code	1000 IDES AG Frankfurt		
Lot No.			

0 Items (No entry variant selected)

St...	G/L acct	Short Text	D/C	Amount in doc.curr.	Loc.curr.amount	T...	Tax jurisdictn code	W...	Assignment n
			H Cr...		0.00	AA			
			H Cr...		0.00	AA			
			H Cr...		0.00	AA			
			H Cr...		0.00	AA			
			H Cr...		0.00	AA			

Specify account number | E75 (2) 800 | ec7server5 | INS

Figure 3.11 Starting Screen of Transaction FB60


Definition Edit Goto Utilities Implementation Environment System Help
SAP
Documentation
Attributes
Interface
FCodes
Subscreens

Business Add-Ins: Display Definition BADI_FDCB_SUBBAS01

Definition name	BADI_FDCB_SUBBAS01
Definition short text	Screen Enhancement 1 on FDCB Basic Data Screen (010, 510)

Call program	Scr.No	Subscreen area	Description
SAPLFDCB	10	SUBBAS01	Enhancement 1 Vendor Basic Data
SAPLFDCB	510	SUBBAS01	Enhancement 1 Customer Basic Data

E75 (2) 800 | ec7server5 | INS

Figure 3.12 Subscreens Tab of the BAdI Definition BADI_FDCB_SUBBAS01

Note

Because a BAdI definition manages only one subscreen area, it's logical that none of the five BAdIs allows multiple implementations. Therefore, if you need to enhance a customer or vendor screen in Enjoy accounting transactions, you first have to make sure none of the five BAdIs are being implemented by some other developer or by an installed add-on.

Before implementing a BAdI, you have to design your own subscreens. In the IDES system, we created two simple subscreens in module pool `ZFB50ENH` for a customer and vendor to illustrate the technique (see Figure 3.13 and Figure 3.14).

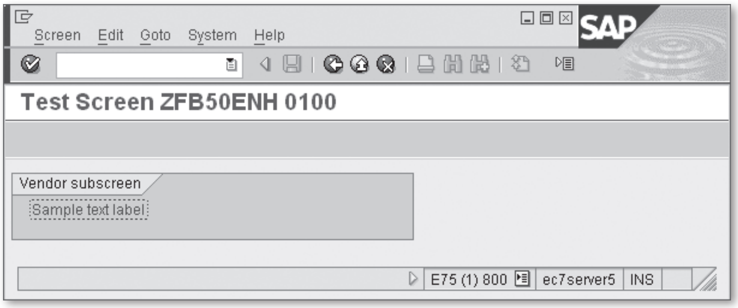


Figure 3.13 Custom-Defined Vendor Subscreen to Be Used in Transaction FB60

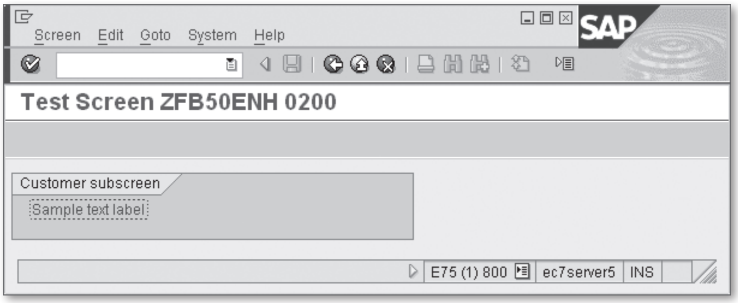
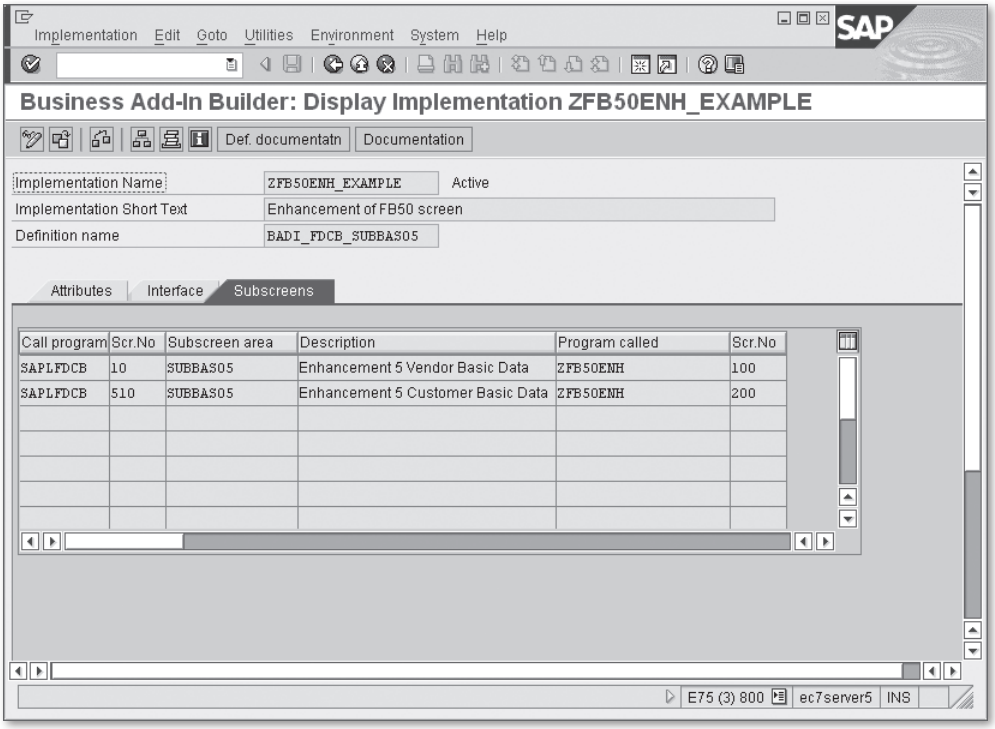


Figure 3.14 Custom-Defined Customer Subscreen to be used in Transaction FB70

Each of the BAdI definitions (from `BADI_FDCB_SUBBAS01` to `BADI_FDCB_SUBBAS05`) supports single implementation. So we can enhance the screen only if we have an unimplemented BAdI from the range. Because several installed add-ons in the system use these BAdI definitions, only `BADI_FDCB_SUBBAS05` is available.

Implementing the BAdI

We created a new implementation for BAdI BADI_FDCB_SUBBAS05 named ZFB50ENH_EXAMPLE. The key point here is defining a linkage between the host subscreen area of the standard SAP program and our own subscreen. We make corresponding settings on the SUBSCREENS tab of the BAdI implementation. In Figure 3.15, you can see that our 200 screen linked to subarea SUBBAS05 of screen 510 in program SAPLFDCB (which is actually a main program of function group FDCB) and subscreen 100 corresponds to the subarea of screen 10.



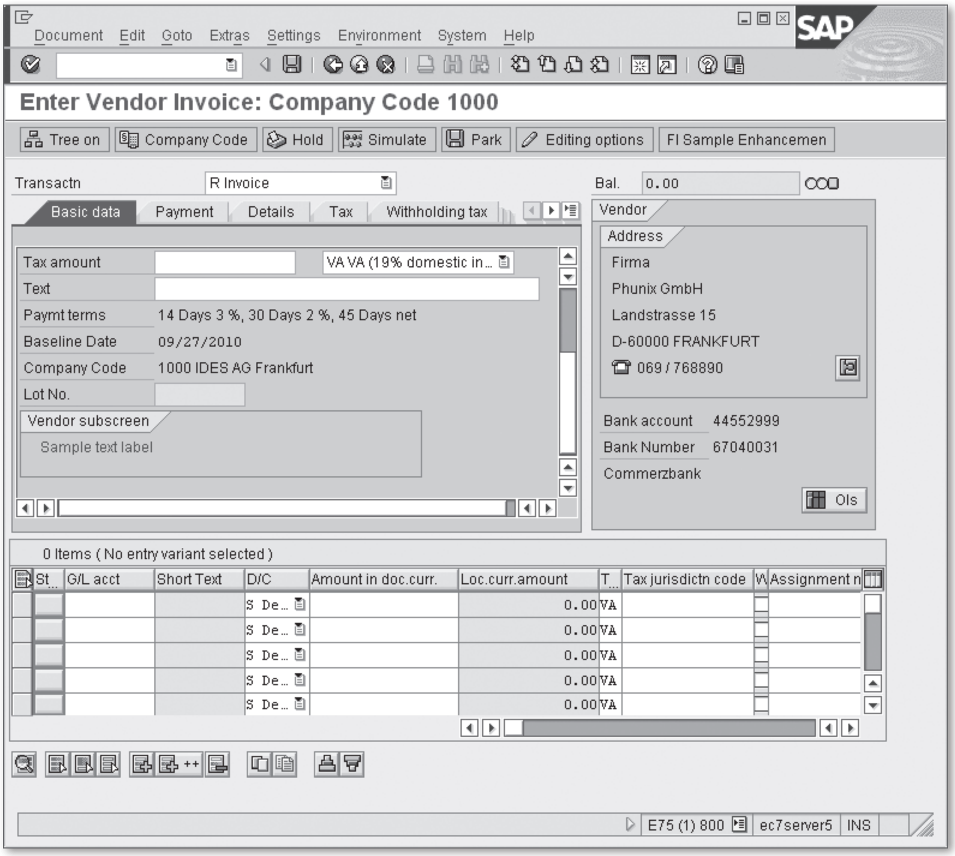
Business Add-In Builder: Display Implementation ZFB50ENH_EXAMPLE

Implementation Name	ZFB50ENH_EXAMPLE	Active
Implementation Short Text	Enhancement of FB50 screen	
Definition name	BADI_FDCB_SUBBAS05	

Call program	Scr.No	Subscreen area	Description	Program called	Scr.No
SAPLFDCB	10	SUBBAS05	Enhancement 5 Vendor Basic Data	ZFB50ENH	100
SAPLFDCB	510	SUBBAS05	Enhancement 5 Customer Basic Data	ZFB50ENH	200

Figure 3.15 Custom-Defined Subscreen Linkage with the Host Program Screen Subarea

After activating the BAdI implementation, you can test the standard transaction. Figure 3.16 shows how the look of Transaction FB60 changed. To make your newly created subscreen area more attractive, you have to scroll down to the BASIC DATA tab.



Enter Vendor Invoice: Company Code 1000

Transactn	R Invoice
Bal.	0.00

Tax amount	VA VA (19% domestic in…
Text	
Paymt terms	14 Days 3 %, 30 Days 2 %, 45 Days net
Baseline Date	09/27/2010
Company Code	1000 IDES AG Frankfurt
Lot No.	

Vendor subscreen

Sample text label

Vendor

Address

Firma

Phunix GmbH

Landstrasse 15

D-60000 FRANKFURT

069 / 768890

Bank account	44552999
Bank Number	67040031
Commerzbank	

0 Items (No entry variant selected)

St...	G/L acct	Short Text	D/C	Amount in doc.curr.	Loc.curr.amount	T...	Tax jurisdictn code	W	Assignment n
			S De...		0.00	VA			
			S De...		0.00	VA			
			S De...		0.00	VA			
			S De...		0.00	VA			
			S De...		0.00	VA			

E75 (1) 800 | ec7server5 | INS

Figure 3.16 Enhanced Overview Screen of Transaction FB60

Data Transfer

If you plan to use the enhancement for supplying additional data to the document, then you must implement data transfer to and from the main program. Each of the five BAdI definitions has two methods:

- `PUT_DATA_TO_SCREEN_OBJECT`
  Used to transfer data from the main program to your own subscreen; called in PBO.
- `GET_DATA_FROM_SCREEN_OBJECT`
  Used to transfer your own screen data back to the main program; called in PAI.

Both methods have one parameter of structure type INVFO. By default, INVFO has numerous fields from Tables BKPF, BSEG, and some other accounting document tables. The host program uses MOVE-CORRESPONDING logic to transfer data to and from the INVFO structure before or after calling the BAdI implementation. So if you use your own custom-defined fields to display and modify data, you have to make sure that INVFO includes those fields (e.g., in an append structure), together with other accounting tables you use to store custom-defined data.

Note

The SAP system uses vendor or customer line items depending on the transaction type when transferring data to and from the INVFO structure. For example, in Transaction FB60 (Incoming Invoice), as soon as a user enters a value into the Vendor number field, the SAP system generates a vendor line item that is used to fill structure INFVO with values.

3.3 Accounting Document Data Enhancement

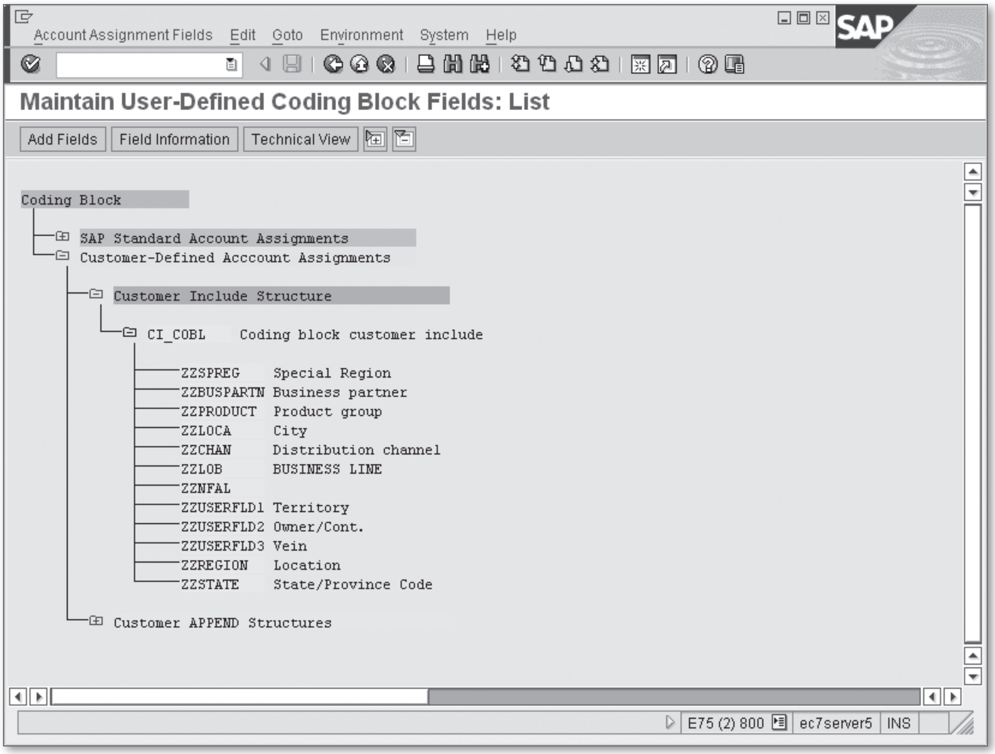
Expanding an accounting document with custom defined fields is a common task. You can hardly come across an SAP ERP implementation without additional accounting document fields. Most often, the line-item (not header) structure of the document is enhanced because line items are the entities that directly affect a particular account balance and turnover. Additional fields help businesses to build a corporate-specific finance methodology and reporting. In SAP, these fields are also known as *coding blocks* or *account assignments*.

Technically, line items can be enhanced with additional fields in two steps:

1. Enhance special customer include structure CI_COBL, which contains custom-defined fields. This can be done directly in Transaction SE11.
2. Add the same fields to Table BSEG, which is formally a modification.

You can do this using the IMG node in menu path Financial Accounting (New) • Financial Accounting Global Settings (New) • Ledgers • Fields • Customer Fields • Edit Coding Block. Here you not only can enhance data structures but also maintain line-item subscreens, which will then be used in all account document transactions.

In the IDES system used for training purposes, you can see that many custom-defined fields were added to the system *coding block*. See an example of coding block structure in Figure 3.17.


Account Assignment Fields Edit Goto Environment System Help
Maintain User-Defined Coding Block Fields: List
Add Fields
Field Information
Technical View
E75 (2) 800
ec7server5
INS

```
Coding Block
  SAP Standard Account Assignments
  Customer-Defined Acccount Assignments
    Customer Include Structure
      CI_COBL     Coding block customer include
        ZZSPREG    Special Region
        ZZBUSPARTN Business partner
        ZZPRODUCT  Product group
        ZZLOCA     City
        ZZCHAN     Distribution channel
        ZZLOB      BUSINESS LINE
        ZZNFAL
        ZZUSERFLD1 Territory
        ZZUSERFLD2 Owner/Cont.
        ZZUSERFLD3 Vein
        ZZREGION   Location
        ZZSTATE    State/Province Code
    Customer APPEND Structures
```

Figure 3.17 Example of Coding Block Structure

Note

All of the field names start with ZZ to comply with SAP naming conventions.

By introducing Flexible General Ledgers, SAP now allows you to expand totals tables. You can see the corresponding node in IMG in the following menu path: Financial Accounting (New) • Financial Accounting Global Settings (New) • Ledgers • Fields • Customer Fields • Include Fields in Totals Table. In the pre-Flexible General Ledgers era, you could build a similar functionality only by using Special Purpose Ledgers.

Note

Adding fields to accounting document line items and expanding totals tables can significantly affect overall system performance. Thus, all decisions in this area should be made after a thorough evaluation of the potential business impact.

3.4 Data Processing Enhancements during Dialog Processing

Many kinds of user exits (business transaction events and processes, and BAdI implementations) are available during the dialog processing of an accounting document. When implementing such user exits, be aware that they can be called several times while a user works with the document. For example, some user exits are called up in every PBO-PAI loop pass, which in turn can be initiated by various user actions, such as simply pressing the `Enter` key. Other user exits are called only once at the very start of the transaction.

In the following subsections, we'll discuss available user exits and the moments when they are called.

3.4.1 Data Processing BTEs

Now let's look at the available BTEs you can use to enhance accounting posting in dialog transactions.

00001140 (POST DOCUMENT: Exclude OK [Enjoy] Codes)

This P&S event is called both in the PBO and PAI modules of most accounting transactions. Its main function is to provide additional control over available GUI function codes. The `T_EXCTAB` table parameter contains an exclusion list for GUI status. The event also has table parameters `T_BKPF` and `T_BSEG`, which represent the current state of the accounting document that is being created, displayed, or edited.

In PBO processing, the `T_EXCTAB` table parameter is used as an exclusion list for the next `SET PF-STATUS` command, whereas in the PAI module, the parameter is used for checking entered function code for execution. As in PAI, the event is called after transferring screen data into the program data. You can implement sufficient logic to either prevent or allow execution of a particular function code, depending on whether program data has been changed.

00001085 (POST DOCUMENT: Functions for Line Item)

This event functionality is very similar to that of 00001080. The difference is that the `00001085` event is called in every PBO processing logic, while `00001080` is called only once. You can use event `00001085` to implement the dynamic button text assignment, depending on the document data.

00001011 (POST DOCUMENT: Checks at Line-Item Level)

This event can be employed for additional check logic after a user has entered line-item data. It is called in the PAI screen logic and has two import parameters, `I_BKPF` and `I_BSEG`, for current document header and item, respectively.

This event transfers the check result via a direct error message without a `RAISING EXCEPTION` addition. The host program shows an error message as information.

00001005 (POST DOCUMENT: Footer Input)

This event is called only for customer or vendor items during PAI logic processing. The event can be used for additional checking of the customer or vendor document items. It is called after the item posting key determination. Together with the document header and item parameters, the event also has two structure parameters of types `SKB1` and `SKA1` for passing reconciliation account data. You can display an error message by directly using the `MESSAGE` statement.

3.4.2 BTE Processes

00001110 (DOCUMENT POSTING: Check on Invoice Duplication)

This process is called only for vendor invoice posting transactions and can be used to stop standard duplicate invoice checking. It is called from within the `FI_DUPLICATE_INVOICE_CHECK` function module before performing the invoice duplicates check against table BSIP. If you have your own duplication check logic, you can implement it in the BTE process 00001110. To prevent performing standard supplication logic, your process must return nonblank values in export parameter `E_NOSTD`.

00001100 (DOCUMENT POSTING: Adjust SAP Internal Payment)

This process can be used for external determination of payment terms for the incoming vendor Invoice. The process is called only once in PAI logic processing after entering all mandatory basic data of the invoice: document date, posting date, vendor number, and so on. The process function should return the following parameters: payment terms code, payment day counts, payment block key, payment method, and payment baseline date.

3.4.3 BAdI

FAGL_PERIOD_CHECK (Posting Period Check)

This BAdI has only the method PERIOD_CHECK. Using this method, you can implement external program logic to automatically decide if posting is possible to the given period or not. Together with several import parameters such as company code, fiscal year, posting period, and others, the PERIOD_CHECK method has a changing parameter CH_SUBRC, which is used to transfer the result of external program logic. If CH_SUBRC has a value of 0, then posting is allowed to the specified period; if the value is 4, then posting is not permitted.

This BAdI method is called only once, as soon as a user has entered mandatory fields in the accounting document header, including the posting date.

FI_TRANS_DATE_DERIVE (Derive BKPF-WWERT from Other Document Header Data)

This BAdI has only one method: DERIVE_WWERT. It allows you to implement any corporate-specific logic of the valuation date calculation. The BAdI is filter-dependent, and the filter value is the country ISO code. In runtime, the filter value is the country from the company code master record.

Method DERIVE_WWERT uses the document date, posting date, and document type as import parameters. This method has one exporting parameter of type WWERT_D. Multiple implementations of this BAdI are not permitted. The BAdI method is called only once—as soon as a user enters the mandatory fields in the accounting document header.

FAGL_DERIVE_SEGMENT (Segment Derivation)

If you use the Flexible General Ledger and the segment accounting functionality, you can use the BAdI definition FAGL_DERIVE_SEGMENT to externally derive a segment code based on values from the accounting document *coding block*. The notion of a segment is used in the recently introduced Flexible General Ledger accounting as a basic entity for segmental reporting.FAGL_DERIVE_SEGMENT is filter-dependent; and code from the controlling area is the filter value in runtime. This BAdI interface has only one method, GET_SEGMENT, which is called for each cost controlling relevant line item.

FAGL_DERIVE_PSEGMENT (Partner Segment Derivation)

This BAdI is similar to FAGL_DERIVE_SEGMENT and is used to derive the partner segment from coding block data for segmental reporting. As its counterpart, the BAdI has only one method, GET_PEGMENT, which is called just after the FAGL_DERIVE_SEGMENT BAdI call. It has the same filter and the same interface as its counterpart.

TR_GET_ACCNT_ASSIGN (Determine FM Account Assignment from Coding Block)

In this case, the BAdI is used to derive Funds Management (FM) fields from other fields of the coding block. This BAdI has a single method, GET_ACCNT_ASSIGN, which is called for each line item in the PAI logic. The BAdI implementation should only be used if you employ the FM functionality.

3.4.4 Substitutions and Validations

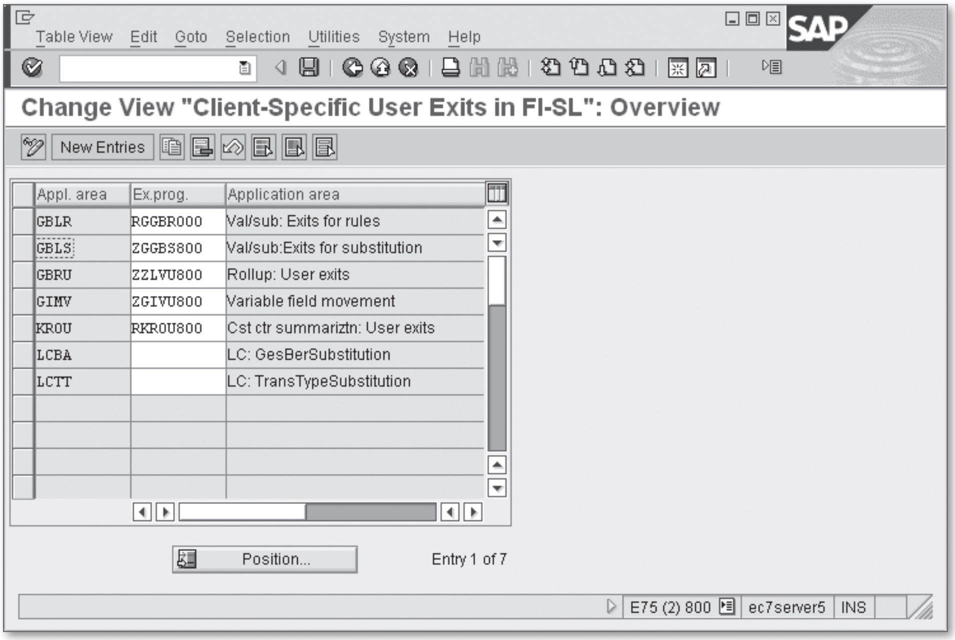
The Financial Accounting component (FI) has its own specific technique for intercepting and amending standard logic when entering accounting documents. The technique is called *validation and substitution*. Generally, validations and substitutions belong to a scope of responsibility of FI functional consultants because the technique normally doesn't involve ABAP development. However, it's capable of tailoring external programming logic in the form of external subroutine calls.

The corresponding customizing activity can be found in the IMG tree via the following menu path: Financial Accounting (New) • Financial Accounting Global Settings (New) • Tools • Validation/Substitution. The resulting screen shows a sample validation editing screen. As the names imply, validation is a tool for an additional check, and substitution is a tool for changing field values.

Each validation and substitution of an object has its name and description and can be assigned to any number of company codes. Each validation or substitution has a specific callup point: header, item, and whole document. Both validation and substitution can have an arbitrary number of steps; each step has a prerequisite, which is a preliminary execution condition. Each validation step has a check. A message appears if the check fails. Each substitution step has one or more field substitutions where you can specify a calculation for a particular single field value. Prerequisites, checks, and substitutions can contain external subroutines, which is why we included substitutions and validations in our discussion.

Defining a Subroutine Pool for FI Substitutions and Validations

Maintenance view V_T80D contains records where you assign a subroutine pool to a particular application area. SAP provides several predefined application areas for different functional modules. For FI substitutions and validations, the GBLS application area is used. Figure 3.18 shows contents of the V_T80D view in the IDES system. A common practice is to copy the standard sample subroutine pool `RGGBS000`, which is delivered by SAP, to your own system. `RGGBS000` has sufficient comments to help you develop your own subroutines.



Change View "Client-Specific User Exits in FI-SL": Overview

Appl. area	Ex.prog.	Application area
GBLR	RGGBR000	Val/sub: Exits for rules
GBLS	ZGGBS800	Val/sub:Exits for substitution
GBRU	ZZLVU800	Rollup: User exits
GIMV	ZGIVU800	Variable field movement
KROU	RKROU800	Cst ctr summariztn: User exits
LCBA		LC: GesBerSubstitution
LCTT		LC: TransTypeSubstitution

Entry 1 of 7

Figure 3.18 Subroutine Pool Assignment to Application Areas

Defining Subroutines for Substitution and Validation

When defining subroutines for substitution or validation, you have to follow some specific rules:

- The subroutine name must be four characters long.
- The subroutine can have either one parameter, or one parameter of type `GB002_015` declared in type pool `GB002`.
- The subroutine must register itself in another predefined subroutine, which can be found in `GET_EXIT_TITLES`. If you copied your subroutine pool from standard program RGGBS000, then `GET_EXIT_TITLES` contains explanation comments and subroutine samples.

Substitutable Fields

Not all accounting document fields can be substituted. The list of substitutable fields is maintained in the maintenance view VWTYGB01, referencing cross-client table GB01. Field names relevant to FI substitution are stored in this view under BOOLEAN CLASS 9. Figure 3.19 shows the substitutable fields list.

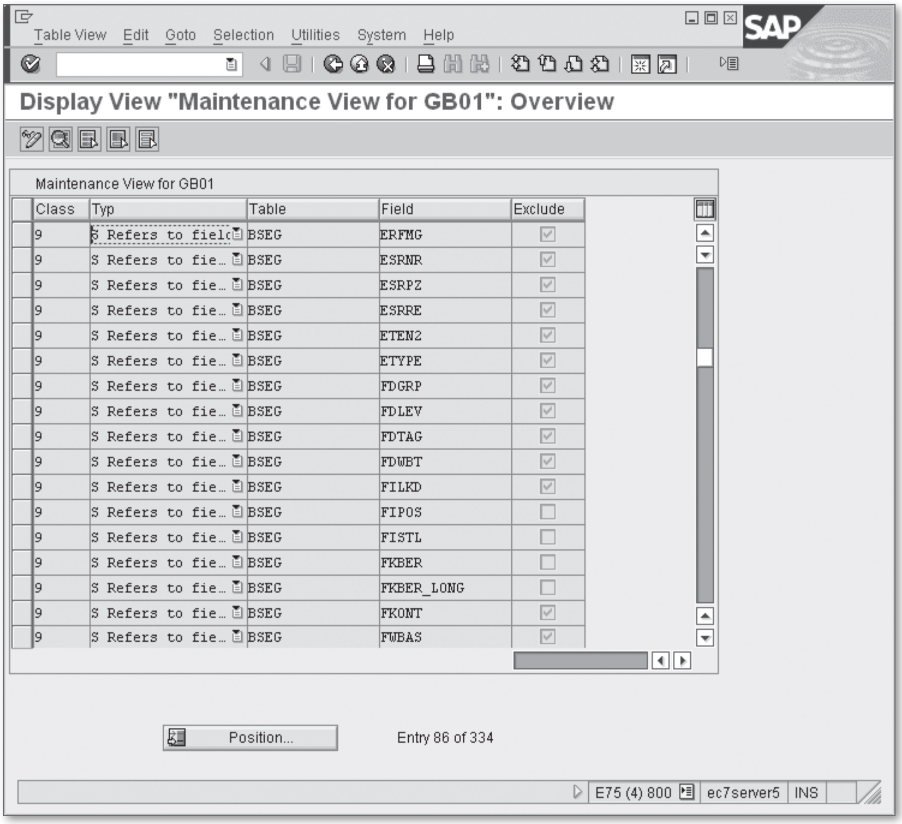
Calling Moments

In dialog transactions, validations and substitutions of the document header and items are called from within PAI logic processing. In old-styled FI transactions such as FB01, F-02, and others, documents, header substitutions, and validations are called in the PAI logic in the first transaction screen, and then item level substitutions and validations are called in the PAI logic of each line item. The whole document's substitutions and validations are called only when a user saves the document.

In Enjoy transactions such as FB50, FB60, and FB70, the main screen contains header data fields together with a table control for entering item data. Header and item substitutions and validations are called each time the system processes PAI logic.

Note

When you are running substitutions and validations, the accounting document number is still undefined. You can only specify the document number during the document saving process. At each callup point (header, item, and whole document), all relevant substitutions are executed before validations.



Display View "Maintenance View for GB01": Overview

Maintenance View for GB01

Class	Typ	Table	Field	Exclude
9	S Refers to field	BSEG	ERFMG	✓
9	S Refers to fie…	BSEG	ESRNR	✓
9	S Refers to fie…	BSEG	ESRPZ	✓
9	S Refers to fie…	BSEG	ESRRE	✓
9	S Refers to fie…	BSEG	ETEN2	✓
9	S Refers to fie…	BSEG	ETYPE	✓
9	S Refers to fie…	BSEG	FDGRP	✓
9	S Refers to fie…	BSEG	FDLEV	✓
9	S Refers to fie…	BSEG	FDTAG	✓
9	S Refers to fie…	BSEG	FDWBT	✓
9	S Refers to fie…	BSEG	FILKD	✓
9	S Refers to fie…	BSEG	FIPOS	☐
9	S Refers to fie…	BSEG	FISTL	☐
9	S Refers to fie…	BSEG	FKBER	☐
9	S Refers to fie…	BSEG	FKBER_LONG	☐
9	S Refers to fie…	BSEG	FKONT	✓
9	S Refers to fie…	BSEG	FWBAS	✓

Position...   Entry 86 of 334

E75 (4) 800 | ec7server5 | INS

Figure 3.19 Substitutable Field List (Table GB01)

3.5 Data Processing Enhancements during Document Saving

Saving a newly created accounting document is a complex process, consisting of the following phases:

1. Generate additional line items (e.g., for tax calculations).
2. Perform final checks of the whole document.
3. Assign the document number.
4. Update the database tables.

At each phase, the system calls different kinds of user exits.

In the next subsection, we'll see business transaction eventsprocesses and BAdIs that are available when processing accounting document saves.

3.5.1 BTE Events

First, let's walk through the available BTE events.

00001005 (POST DOCUMENT: Footer Input) and 00001011 (POST DOCUMENT: Checks at Line-Item Level)

Events `00001005` (POST DOCUMENT: Footer Input) and `00001011` (POST DOCUMENT: Checks at Line-Item Level) are called for each automatically created line item. (See also Section 3.4.1, Data Processing BTEs).

00001020 (POST DOCUMENT: Prior to Final Checks)

The event is called before the standard final check of the whole document. At the moment of call, all amounts in the document are correct, and all necessary items are generated (e.g., tax items). The document number is unavailable at this moment.

00001025 (POST DOCUMENT: Final Checks Completed)

This event is called after all standard checks, substitutions, and validations are performed for the whole document. The document number is unavailable at this moment.

Note

Some of you might remember that after the first introduction of this BTE event the source code of the calling function `OPEN_FI_PERFORM_00001025_E` used ABAP memory for storing transient variables, which made possible uncontrolled updates of accounting document data. To the great disappointment of the FI development community, SAP has closed this loophole via SAP Note 530655 and the subsequent support package.

Despite this disappointment, we must admit that altering document data at this point is not a good practice, and can lead to inconsistencies in posting data. For example, by the time of the 1025 event call the system may already have performed actions such as availability control in Funds Management, generating Cost Controlling documents, or forming Profitability Analysis data (and many others). So, any significant change in the document account assignment at this moment would definitely cause problems with data consistency (not even considering changing amounts or adding new items). To make things worse, such a flaw is not always apparent right away, and might not be discovered until much later.

00001030 (POST DOCUMENT: Posting of Standard Data)

This event is called after registering the `POST_DOCUMENT` update function module call and just before the final `COMMIT`.

Note

BTE `00001030` is the only user exit where the accounting document number is available.

3.5.2 BTE Processes

Now, let's consider the BTE processes you can use to intervene in the posting process.

00001120 (DOCUMENT POSTING: Field Substitution Header/Items)

Process 00001120 allows you to implement true programming substitution logic. Unlike standard validations and substitutions in process 00001120, all substitutable fields are passed via dictionary structures:

- `BKDF_SUBSTS`
  For recurring documents.
- `BKPF_SUBST`
  For document header fields.
- `BSEG_SUBST`
  For document line-item fields.

You transfer values to and from this intermediate area by using the `MOVE-CORRESPONDING` statement. You can easily add additional fields to any of these structures using append structures.

Process 00001120 is called at the end of the final document checks, just before running the standard FI substitutions and validations. The document number is unavailable at the moment of call.

00001130 (POST DOCUMENT: SAP-Internal Field Substitution)

Process 00001130 has a similar interface to process 00001120, but it's reserved for SAP's internal needs. The document number is unavailable at the moment of call.

00001150 (OPEN FI EXIT 00001150: Get Offsetting Account)

This process is called after the standard substitution and validation. This substitution capability is used via dictionary structure `ACCIT_SUBST`. The process is used for the derivation of the offset account type and offset account number (fields `ACCIT-GKOAR` and `ACCIT-GKONT`). The document number is unavailable at the moment of call.

00001170 (POST RESIDUAL ITEMS: Deactivate No. Range Buffers)

Process 00001170 is called at the moment of assigning the document number. The process function receives the company code, fiscal year, and number range as import parameters. Using export parameters `E_NO_BUFFER` and `E_RANGE`, the process function can switch off number range buffering or even change the number range according to business-specific logic. Switching off number range buffering provides even sequential document numbering, which is required by some accountants.

3.5.3 BAdIs

Numerous BAdI methods are called during the document saving process. However, some can be used only by SAP; others belong to other application modules and are out of the scope of our discussion.

3.6 SAP Internal Techniques for Processing Accounting Data Flow (RWIN)

SAP uses a P&S technique called the RW interface (RWIN) for posting accounting values from different SAP components (Sales, Logistics, Payroll) to a variety of accounting components, including SAP General Ledger and Cost Controlling, Profitability Analysis, Funds Management (budgeting), and many others.

Like most other P&S interfaces, RWIN has a control Table TRWPR (Table 3.3) that stores function module names to be called at particular moments of accounting document generation.

Name	Key	Description
PROCESS	✓	Transaction type for which CO interface is accessed
EVENT	✓	Phase of processing at which the RWIN function is called up
SUBNO	✓	Sequence number
COMPONENT		Component in ACC interface
KZ_BLG		Indicator: Function module operates in document
FUNCTION		Name of function module

Table 3.3 TRWPR Table Structure

The interface function call is coded by PROCESS, EVENT, and SUBNO fields. The PROCESS field codes a particular business process, which generates values in accounting; for example, it can be goods receipt posting, vendor invoice posting, and so on. The EVENT field is a phase of the processing (e.g., document item check), and SUBNO is a number of the process step within one event. All function modules assigned to a particular event must have the same predefined interface.

Note

The main trouble with RWIN is that it has no documentation, so if you want to use it you have to spare a considerable amount of time for reading SAP source code and debugging just to make sure you do the right things. At the same time, the initial stage of investigation for this interface can be rather easy. To discover all the events and functions of the corresponding RWIN process, you just have to place a breakpoint at function module RWIN_CHECK_SUBSET and start the transaction you want to check for the RWIN presence.

Both core accounting generation modules SAPMF05A and RWCL use the RWIN interface. However, they use slightly different sets of P&S modules: RWCL mainly uses the process DOCUMENT, whereas SAPMF05A uses processes BELEG and BELEGPOS, among others.

In dialog transactions, RWIN works only at the moments of either simulating a document or saving it permanently. At the same time, the whole process of generating an accounting document inside function group `RWCL` is built on RWIN.

Note

Table TRWPR is marked with delivery class "S", which means that this data belongs to SAP, and you should not change the data in any way. Nevertheless, you can append your own entries to the table via the standard maintenance dialog, ignoring the corresponding caution message. Remember, though, to comply with the function module interface and assign a subnumber to your entry from the 900-999 interval. Additionally, you should never delete any entries from this table. Generally, RWIN can be used as a last resort if no other enhancement capabilities are available.

3.6.1 RWIN Summary

The presence of RWIN should greatly simplify the introduction of the Flexible General Ledger in SAP ERP. The way we see it, SAP developers just had to add another bunch of RWIN compatible function modules to the existing interface and voilà: The new technology is up and running without touching anything in old stable interface! Well, at least the ideal process should look like this.

All in all, you likely won't need to use RWIN, unless you are developing a brand new industry solution. Regardless, it's useful to know how to find the program code where accounting data are updated because the source code is the most accurate technical documentation.

3.7 Differences in Data Processing between Dialog Transactions and Program Functions

In practice, the SAP ERP system usually generates accounting documents automatically; for example, when posting incoming vendor invoices in Materials Management, or posting goods issues for outbound delivery in Sales and Distribution. On the other hand, some customers employ BAPI functionality in their proprietary developments to automatically post accounting documents.

In all of these scenarios, function group RWCL is used, which represents the core logic of accounting document creation: functions `AC_DOCUMENT_CREATE`, `AC_DOCUMENT_POST`, and `AC_DOCUMENT_GENERATE`. This function group is also used for mass creation of accounting documents (direct input option). As compared to the manual creation of an accounting document through a dialog transaction, the process of automatic generation of an accounting document has a similar set and sequence of user exits.

Note

Generating accounting documents via programming can be complex because there is no way to automatically generate items for tax calculation. All data must be prepared in your program.

In the next section, we'll walk through the available BAdIs and business transaction events/processes you can use while generating an accounting document programmatically.

3.7.1 Additional BAdI AC_DOCUMENT

Function group `RWCL` uses additional BAdI `AC_DOCUMENT` in the process of generating an accounting document. The BAdI method `CHANGE_INITIAL` is called at the very beginning before checking input data by RWIN components. The `CHANGE_AFTER_CHECK` method is called after the RWIN check.

3.7.2 BTEs That Are Not Called

During the process of document generation within the `RWCL` function group, none of the BTEs relevant to dialog interaction are called; such as `00001011`, `00001140`, `00001070`, `00001080`, or `00001085`.

3.7.3 Ending BTE 00001050 (POST DOCUMENT: Accounting Interface)

This event is called instead of event `00001030` after assigning the document number.

3.8 Summary

As you can see, SAP provides a generous set of different kinds of user exits during the process of creating and editing an accounting document. The available variety of techniques provides you with a rich programming toolbox, so you can intercept and amend virtually any phase of an accounting posting. Remember, however, that the power has its reverse side, and possible errors in core accounting data processing can have a great negative impact.

In the next chapter, we'll talk about some enhancements techniques in financial reporting.

The main and final goal of any SAP ERP implementation is producing accurate, timely, and complete financial reports.

4 Enhancements in Reports

Financial Accounting with SAP ERP Financials has many different reports, and corporate SAP developers produce even more reports that are specifically tailored to particular business requirements. It's often useful to examine standard reports for enhancement capability because enhancements can significantly reduce development and maintenance efforts. The most common practice in report development is to take a similar report delivered by SAP, copy it into a custom program, and amend the report according to corporate needs.

In the following sections, we'll discuss how you can enhance the financial reports that are used in the everyday practice of virtually any accounting department. Financial reports are line-item reports, which show posting documents for a given period of time.

4.1 Technical Architecture of the Line-Item Report

There are line-item reports for all main account types: general ledger account, vendor account, and customer account. The transactions for these reports are listed here:

- **FBL1N**
  Vendor line-items report.
- **FBL3N**
  General ledger account line-items report.
- **FBL5N**
  Customer line-items report.

Note

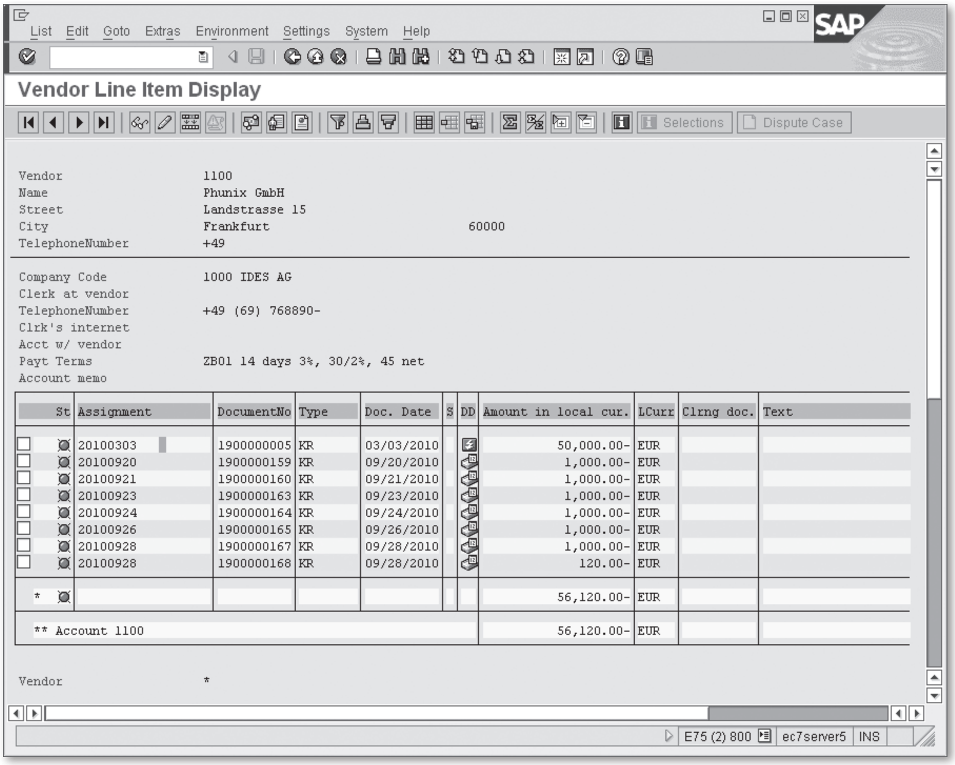
General ledger accounts have a new version of line-item report Transaction FAGLL03 that incorporates the new Flexible General Ledger functionality.

The technical architecture of all three reports looks very similar. All reports are based on the corresponding logical database: SDF for general ledger accounts, KDF for customer accounts (account receivables), and LDF for vendors (account payables).

Note

A common belief about logical databases is that they are slow, inefficient, and obsolete. However, this is not true. Many financial reports are built on logical databases, and almost all HR reports are based on special-aided logical databases. As with other programming tools, the myth originated from inappropriate usage, leading to the poor performance of the software product.

Figure 4.1 shows how a typical line-item report looks—in this case, showing the open items of a vendor. Line-item reports for the other two account types (general ledger accounts and customers) look about the same.



```
Vendor                         1100
Name                           Phunix GmbH
Street                         Landstrasse 15
City                           Frankfurt                                60000
TelephoneNumber                +49

Company Code                   1000 IDES AG
Clerk at vendor
TelephoneNumber                +49 (69) 768890-
Clrk's internet
Acct w/ vendor
Payt Terms                     ZB01 14 days 3%, 30/2%, 45 net
Account memo

     St Assignment             DocumentNo Type       Doc. Date  S DD Amount in local cur. LCurr Clrng doc. Text

[ ]     20100303               1900000005 KR         03/03/2010           50,000.00- EUR
[ ]     20100920               1900000159 KR         09/20/2010            1,000.00- EUR
[ ]     20100921               1900000160 KR         09/21/2010            1,000.00- EUR
[ ]     20100923               1900000163 KR         09/23/2010            1,000.00- EUR
[ ]     20100924               1900000164 KR         09/24/2010            1,000.00- EUR
[ ]     20100926               1900000165 KR         09/26/2010            1,000.00- EUR
[ ]     20100928               1900000167 KR         09/28/2010            1,000.00- EUR
[ ]     20100928               1900000168 KR         09/28/2010              120.00- EUR

   *                                                                      56,120.00- EUR

   ** Account 1100                                                        56,120.00- EUR

Vendor                         *
```

Figure 4.1 Typical Look of a Line-Item Report

It isn't a coincidence that all three reports use the same function group FI_ITEMS for report output. Function module FI_ITEMS_DISPLAY is used for ALV (ABAP List Viewer) output. By default, the standard output is formatted as an ALV List, but the user can switch to ALV Grid using menu path SETTINGS • SWITCH LIST.

Note

If you plan to create your own FI line-item reports, it's a good idea to reuse the FI_ITEMS_DISPLAY function module because it has a basic drill-down functionality and can also be enhanced.

All of the enhancements reside in the function group FI_ITEMS, so if you implement the enhancements, you affect all reports that employ the same functionality. In the following subsections, we discuss the available enhancements and their implementations.

4.1.1 Header and Footer Output Enhancement

You can change the appearance of the result list's header and footer by utilizing BTE event 00001640 (LINE ITEM DISPLAY: Additional Header Lines). The event is called only when the output is formatted using ALV List. While in ALV Grid, the event is not called.

The event is called several times during output: during processing of the list's header and footer, and on the top-of-page event. In the IDES system we created, a function module with an interface is compatible with BTE event 00001640. Listing 4.1 shows the sample implementation.

```
FUNCTION z_sample_interface_00001640.
*"----------------------------------------------------------------------
*"*"Local Interface:
*"  IMPORTING
*"     VALUE(I_RFXPO) LIKE  RFXPO STRUCTURE  RFXPO
*"     VALUE(I_KNA1) LIKE  KNA1 STRUCTURE  KNA1
*"     VALUE(I_LFA1) LIKE  LFA1 STRUCTURE  LFA1
*"     VALUE(I_SKA1) LIKE  SKA1 STRUCTURE  SKA1
*"  EXPORTING
*"     VALUE(E_SUPPRESS_STANDARD) LIKE  BOOLE-BOOLE
*"  TABLES
*"      T_LINES STRUCTURE  EPTEXT
*"----------------------------------------------------------------------
```

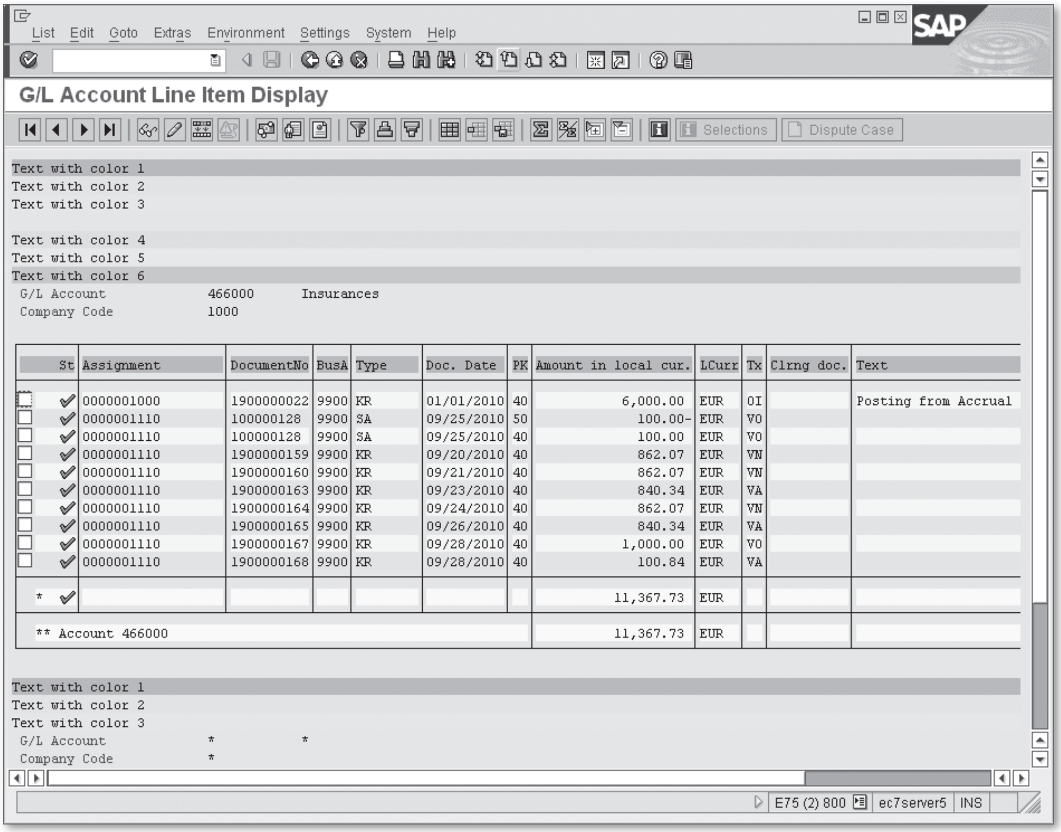
```
  CASE i_rfxpo-koart.
    WHEN 'S'. "GL Accounts
      t_lines-color = '4'.
      t_lines-text = 'Text with color 4'.
      APPEND t_lines.

      t_lines-color = '5'.
      t_lines-text = 'Text with color 5'.
      APPEND t_lines.

      t_lines-color = '6'.
      t_lines-text = 'Text with color 6'.
      APPEND t_lines.

    WHEN 'D'. "Customers

    WHEN 'K'. "Vendors

    WHEN space. "Either Top or Bottom
      t_lines-color = '1'.
      t_lines-text = 'Text with color 1'.
      APPEND t_lines.

      t_lines-color = '2'.
      t_lines-text = 'Text with color 2'.
      APPEND t_lines.

      t_lines-color = '3'.
      t_lines-text = 'Text with color 3'.
      APPEND t_lines.
    WHEN OTHERS.
  ENDCASE.
ENDFUNCTION.
```

Listing 4.1 Source Code of 00001640 Event Implementation Component

The I_RFXPO import parameter of the sample function (see Listing 4.1) contains control fields of the report, which you can use to distinguish the moments of call. For example, you analyze field I_RFXPO-KOART to decide if the call was from the header, footer, or top-of-page.

You can't precisely and completely determine what kind of transaction the call came from using the `I_RFXPO` parameter. Therefore, the colored header and footer will also appear in the customer line-item and vendor line-item reports. However, this should be enough for testing needs.

Figure 4.2 shows the resulting output after activating our BTE event implementation.


List Edit Goto Extras Environment Settings System Help
G/L Account Line Item Display
Selections
Dispute Case
E75 (2) 800
ec7server5
INS

```
Text with color 1
Text with color 2
Text with color 3

Text with color 4
Text with color 5
Text with color 6
 G/L Account            466000      Insurances
 Company Code           1000

     St Assignment        DocumentNo BusA Type     Doc. Date  PK Amount in local cur. LCurr Tx Clrng doc. Text

☐    ✓  0000001000        1900000022 9900 KR       01/01/2010 40             6,000.00 EUR   OI            Posting from Accrual
☐    ✓  0000001110        100000128  9900 SA       09/25/2010 50              100.00- EUR   VO
☐    ✓  0000001110        100000128  9900 SA       09/25/2010 40              100.00  EUR   VO
☐    ✓  0000001110        1900000159 9900 KR       09/20/2010 40              862.07  EUR   VN
☐    ✓  0000001110        1900000160 9900 KR       09/21/2010 40              862.07  EUR   VN
☐    ✓  0000001110        1900000163 9900 KR       09/23/2010 40              840.34  EUR   VA
☐    ✓  0000001110        1900000164 9900 KR       09/24/2010 40              862.07  EUR   VN
☐    ✓  0000001110        1900000165 9900 KR       09/26/2010 40              840.34  EUR   VA
☐    ✓  0000001110        1900000167 9900 KR       09/28/2010 40            1,000.00  EUR   VO
☐    ✓  0000001110        1900000168 9900 KR       09/28/2010 40              100.84  EUR   VA

   * ✓                                                                    11,367.73  EUR

   ** Account 466000                                                      11,367.73  EUR

Text with color 1
Text with color 2
Text with color 3
 G/L Account            *           *
 Company Code           *
```

Figure 4.2 The Result of BTE 00001640 Implementation

You can see that the structure of table parameter `T_LINES` includes a one-character `COLOR` field, whose meaning is obvious from its name. Digits from 1 to 7 are acceptable for this field; they correspond to the standard list output color you use in a `WRITE` statement.

There is also an export parameter, E_SUPPRESS_STANDARD, which allows you to suppress standard header/footer output. To do this, just return X in its value.

4.1.2 Menu Enhancement with BTE Events

The GUI statuses of a classical line-item report have an additional function code OPFI in the ENVIRONMENT menu. The function code references dynamic text OFIWAFTEXT.

To initialize the function code text, you have to develop a function module with the appropriate logic and subscribe it to the BTE event 00001620. The function interface is shown in Listing 4.2.

```
*"----------------------------------------------------------------------
*"*"Lokale Schnittstelle:
*"       IMPORTING
*"              VALUE(I_SPRAS) LIKE  SY-LANGU
*"       EXPORTING
*"              VALUE(E_FTEXT) LIKE  FTEXTS-FTEXT
*"----------------------------------------------------------------------
```

Listing 4.2 The Interface of BTE Event 00001620

The function code text is returned in export parameter E_FTEXT. If there is more than one event implementation, then the function text will be changed to the predefined text "Additional components."

Function code execution has to be implemented via BTE 00001610. The interface is given in Listing 4.3.

```
*"----------------------------------------------------------------------
*"*"Lokale Schnittstelle:
*"       IMPORTING
*"              VALUE(I_KUNNR) LIKE  KNB1-KUNNR
*"              VALUE(I_BUKRS) LIKE  KNB1-BUKRS
*"              VALUE(I_BELNR) LIKE  BKPF-BELNR DEFAULT '000000000'
*"              VALUE(I_BUZEI) LIKE  BSEG-BUZEI DEFAULT '000'
*"              VALUE(I_GJAHR) LIKE  BKPF-GJAHR DEFAULT '0000'
*"----------------------------------------------------------------------
```

Listing 4.3 Interface of BTE Event 00001610

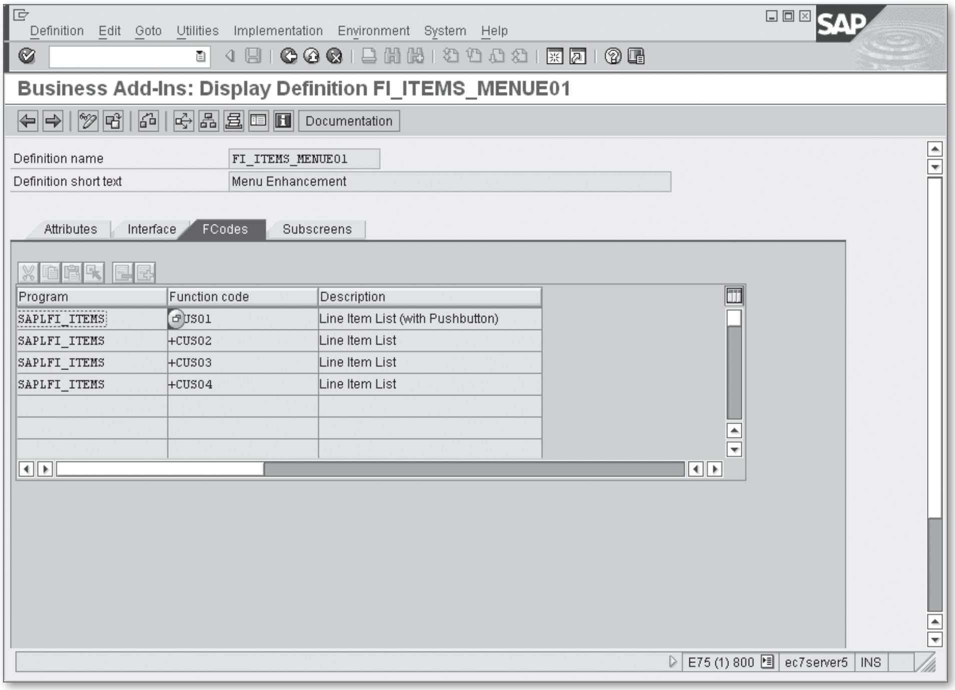
If you define more than one implementation of event 00001610, the user will see a pop-up dialog box with available event implementations.

This pair of BTE events can be used for implementing additional drill-down capabilities in line-item reports. As you can see from the `00001610` interface, the event is called for a particular selected line item.

Also note that your implementations will affect all reports that use the `FI_ITEMS` function group for displaying line-item data.

4.1.3 Menu Enhancement with BAdI

In addition to BTE events, there are two definitions to be used for GUI status enhancement: `FI_ITEMS_MENUE01` and `FI_ITEMS_MENUE02`.



Business Add-Ins: Display Definition FI_ITEMS_MENUE01

Definition name: FI_ITEMS_MENUE01

Definition short text: Menu Enhancement

Program	Function code	Description
SAPLFI_ITEMS	+CUS01	Line Item List (with Pushbutton)
SAPLFI_ITEMS	+CUS02	Line Item List
SAPLFI_ITEMS	+CUS03	Line Item List
SAPLFI_ITEMS	+CUS04	Line Item List

E75 (1) 800 | ec7server5 | INS

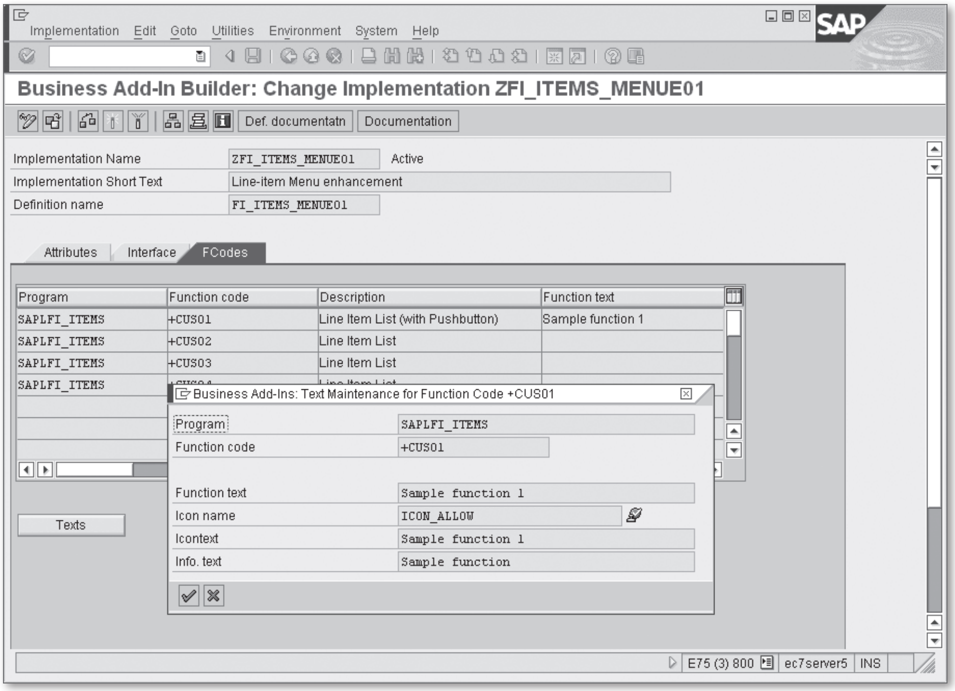
Figure 4.3 BAdI FI_ITEMS_MENUE01 Function Codes

If you open any of these definitions in Transaction SE18, you can see several function codes displayed on the FCodes tab, as shown in Figure 4.3. The `FI_ITEMS_MENUE01` definition has function codes from `+CUS01` to `+CUS04`, and `FI_ITEMS_MENUE02` has codes from `+CUS05` to `+CUS08`. All function codes are assigned to program SAPLFI_ITEMS, which is the main program of function group `FI_ITEMS`. Each function code starts with +; this signals that the function code can be enhanced and won't appear in the menu unless there is an active implementation of the BAdI definition.

The same function codes appear in the default GUI status in function group `FI_ITEMS`, which is used in line-item reports. (See GUI status `ALV_ITEMS_AR`.) You can see function codes from `+CUS01` to `+CUS08` in the EXTRAS menu; also, two function codes `+CUS01` and `+CUS05` appear in the toolbar. Unlike in BTEs, none of these BAdIs can have multiple implementations.

Let's create a sample implementation of `FI_ITEMS_MENUE01` in the IDES system. For testing purposes, it's enough to develop a reaction to only one function code just to see how it works.

First, you set function code attributes on the FCODES tab in the BAdI implementation. For `+CUS01` function code, you enter "Sample function 1" in both the FUNCTION TEXT box and the ICON TEXT box, enter "ICON_ALLOW" in the ICON NAME box, and enter "Sample function" in the INFO.TEXT box. See Figure 4.4 for all function code properties. You leave all other function code properties blank.



Business Add-In Builder: Change Implementation ZFI_ITEMS_MENUE01

Implementation Name	ZFI_ITEMS_MENUE01	Active
Implementation Short Text	Line-item Menu enhancement	
Definition name	FI_ITEMS_MENUE01	

Program	Function code	Description	Function text
SAPLFI_ITEMS	+CUS01	Line Item List (with Pushbutton)	Sample function 1
SAPLFI_ITEMS	+CUS02	Line Item List	
SAPLFI_ITEMS	+CUS03	Line Item List	
SAPLFI_ITEMS			

Business Add-Ins: Text Maintenance for Function Code +CUS01

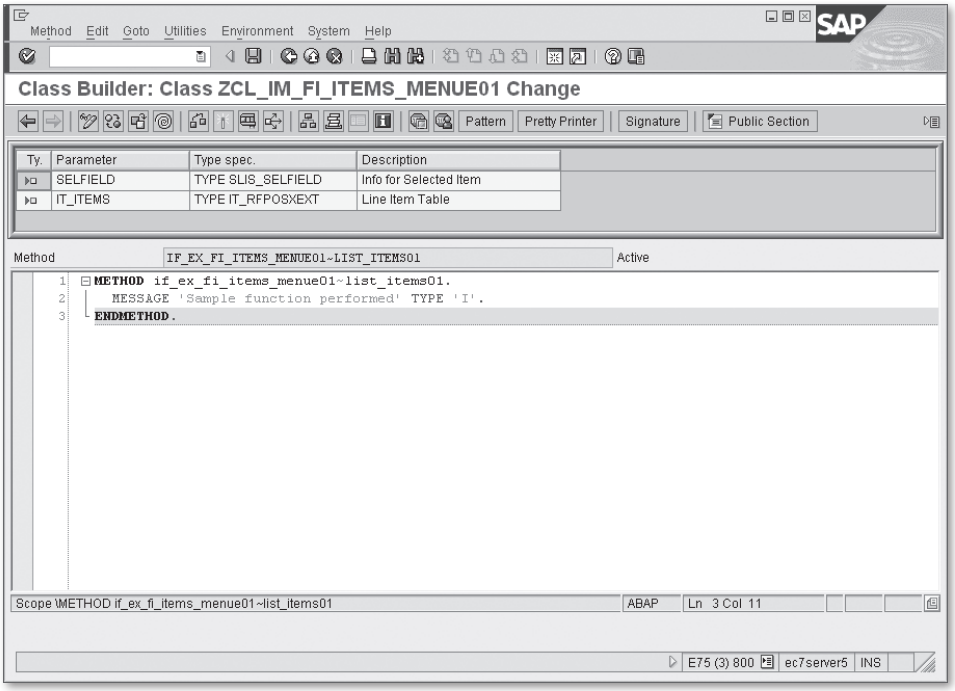
Program	SAPLFI_ITEMS
Function code	+CUS01
Function text	Sample function 1
Icon name	ICON_ALLOW
Icontext	Sample function 1
Info. text	Sample function

E75 (3) 800 ec7server5 INS

Figure 4.4 Function Code Properties in the BAdI Implementation

Now you must develop a reaction for the +CUS01 function code. You have to implement interface method LIST_ITEMS01. For the purpose of this example, we just show a senseless information message. See the method implementation in Figure 4.5.

As you can see, the method has the SELFIELD import parameter containing the selected field's data. The same type is used in function REUSE_ALV_GRID_DISPLAY for passing information on the selected ALV line and field. The import table parameter IT_ITEMS contains the current ALV output table.


Method Edit Goto Utilities Environment System Help
Class Builder: Class ZCL_IM_FI_ITEMS_MENUE01 Change
Pattern
Pretty Printer
Signature
Public Section

Ty.	Parameter	Type spec.	Description
▸▫	SELFIELD	TYPE SLIS_SELFIELD	Info for Selected Item
▸▫	IT_ITEMS	TYPE IT_RFPOSXEXT	Line Item Table

Method IF_EX_FI_ITEMS_MENUE01~LIST_ITEMS01 Active

```
METHOD if_ex_fi_items_menue01~list_items01.
  MESSAGE 'Sample function performed' TYPE 'I'.
ENDMETHOD.
```

Scope \METHOD if_ex_fi_items_menue01~list_items01 ABAP Ln 3 Col 11

E75 (3) 800 ec7server5 INS

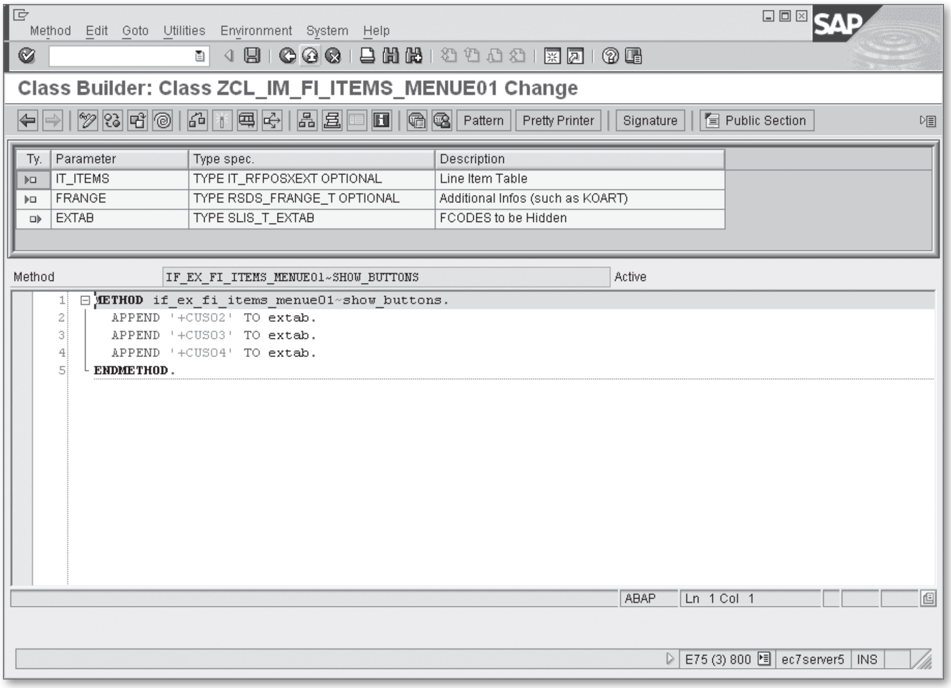
Figure 4.5 LIST_ITEMS01 Method Implementation

To finalize the BAdI implementation, you have to develop interface method SHOW_BUTTONS, which is designed to tell the host program which custom-defined buttons are hidden and which are not. The method has export table parameter EXTAB, which contains the function codes to be excluded from the GUI status. Because you've only defined a reaction for the first function code +CUS01, you should hide all other inactive codes. See the method source code in Figure 4.6.

The method SHOW_BUTTONS also has import parameters:

- IT_ITEMS is the output table of the ALV framework.
- FRANGE is the calling report selection parameter.

Using these parameters, you can implement much more complex logic for deactivating GUI status function codes.

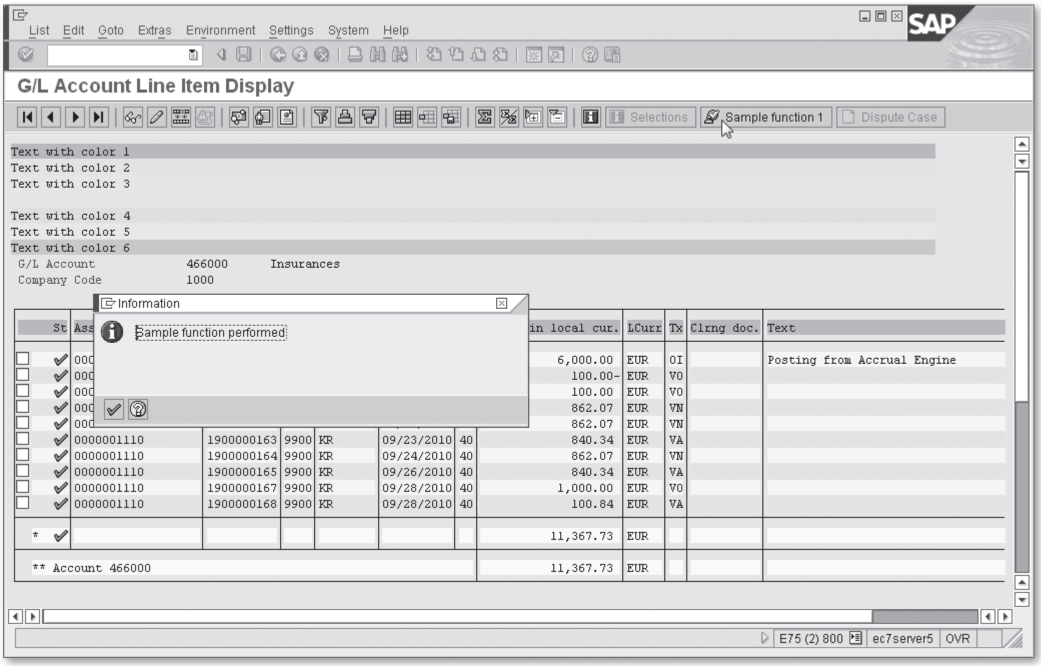

Method Edit Goto Utilities Environment System Help
Class Builder: Class ZCL_IM_FI_ITEMS_MENUE01 Change
Pattern
Pretty Printer
Signature
Public Section
Method
IF_EX_FI_ITEMS_MENUE01~SHOW_BUTTONS
Active
ABAP
Ln 1 Col 1
E75 (3) 800
ec7server5
INS

Ty.	Parameter	Type spec.	Description
▶□	IT_ITEMS	TYPE IT_RFPOSXEXT OPTIONAL	Line Item Table
▶□	FRANGE	TYPE RSDS_FRANGE_T OPTIONAL	Additional Infos (such as KOART)
□▶	EXTAB	TYPE SLIS_T_EXTAB	FCODES to be Hidden

```
METHOD if_ex_fi_items_menue01~show_buttons.
  APPEND '+CUS02' TO extab.
  APPEND '+CUS03' TO extab.
  APPEND '+CUS04' TO extab.
ENDMETHOD.
```

Figure 4.6 SHOW_BUTTONS Method Source Code

Figure 4.7 shows the general ledger account line-item report (Transaction FBL3N) after activation of the BAdI implementation. Notice the ADDITIONAL TOOLBAR button with the previously assigned icon and text.


List Edit Goto Extras Environment Settings System Help
G/L Account Line Item Display
Selections
Sample function 1
Dispute Case
Information
E75 (2) 800
ec7server5
OVR

```
Text with color 1
Text with color 2
Text with color 3

Text with color 4
Text with color 5
Text with color 6
 G/L Account            466000      Insurances
 Company Code           1000

Sample function performed

   St Ass                                            in local cur. LCurr Tx Clrng doc. Text
☐  ✓ 000                                                 6,000.00  EUR   OI            Posting from Accrual Engine
☐  ✓ 000                                                   100.00- EUR   VO
☐  ✓ 000                                                   100.00  EUR   VO
☐  ✓ 000                                                   862.07  EUR   VN
☐  ✓ 000                                                   862.07  EUR   VN
☐  ✓ 0000001110    1900000163 9900 KR  09/23/2010 40       840.34  EUR   VA
☐  ✓ 0000001110    1900000164 9900 KR  09/24/2010 40       862.07  EUR   VN
☐  ✓ 0000001110    1900000165 9900 KR  09/26/2010 40       840.34  EUR   VA
☐  ✓ 0000001110    1900000167 9900 KR  09/28/2010 40     1,000.00  EUR   VO
☐  ✓ 0000001110    1900000168 9900 KR  09/28/2010 40       100.84  EUR   VA

 *  ✓                                                   11,367.73  EUR

 ** Account 466000                                      11,367.73  EUR
```

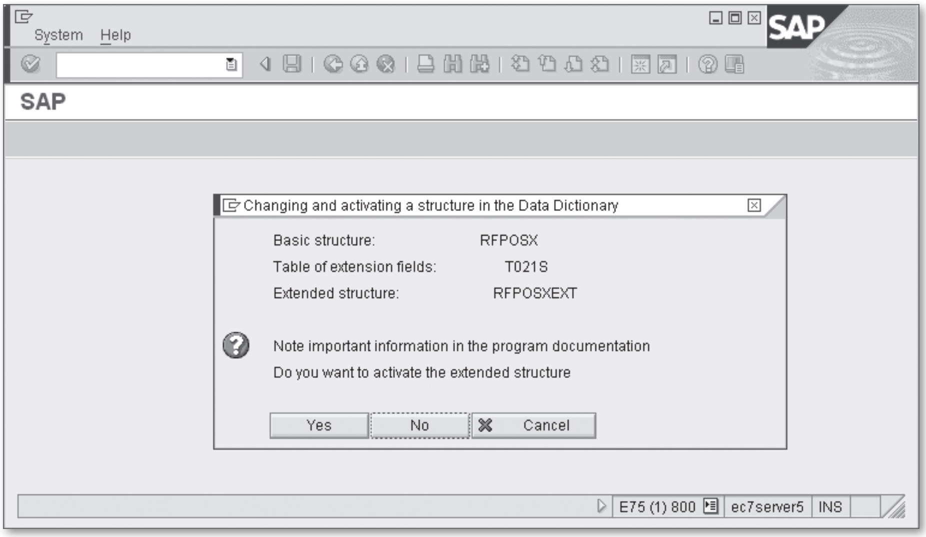
Figure 4.7 The General Ledger Account Line-Item Report After Activating the BAdI Implementation

4.1.4 Output Layout Enhancement

Besides GUI status enhancements, the line-item report provides a set of BTE events for output layout enhancement. Using these events, you can output additional fields into line items or even change the contents of the standard output.

If you plan to add your own calculated fields, you first have to extend the ABAP Data Dictionary structure `RFPOS` and `RFPOSX` with the same fields. Then, you have to run report RFPOSXEXTEND, which combines fields from Tables RFPOSX and those maintained in the customizing view V_T021S (or V_FAGL_T021S) into the automatically generated structure `RFPOSXEXT`. Maintenance views V_T021S and V_FAGL_T021S refer to the same table (T021S) and represent the IMG activity DEFINE SPECIAL FIELDS FOR LINE ITEM DISPLAY.

Figure 4.8 shows the starting screen of report RFPOSXEXTEND.



Changing and activating a structure in the Data Dictionary

Basic structure: RFPOSX
Table of extension fields: T021S
Extended structure: RFPOSXEXT

Note important information in the program documentation
Do you want to activate the extended structure

Figure 4.8 Starting Screen of RFPOSXEXTEND Report

After extending structures `RFPOS` and `RFPOSX`, you implement BTE events `0000163` and `00001650`. Event `00001630` is called before sequential processing of the line item's output table. Event 00001650 is called for each selected line item.

In `00001630` event implementation, you should prepare your own arbitrary data selection; for example, you select data from all necessary database tables into internal tables. Inside event `00001650`, you use previously selected data for efficient custom field calculation.

Listing 4.4 shows the subscription function interface for event `00001630`. Table parameter `T_KONTAB` contains all of the selected account data. Table parameter `T_SLBTAB` contains a selected list of company codes.

```
*”----------------------------------------------------------------------
*”*”Lokale Schnittstelle:
*”       IMPORTING
*”             VALUE(I_KNA1) LIKE  KNA1 STRUCTURE  KNA1 OPTIONAL
*”             VALUE(I_LFA1) LIKE  LFA1 STRUCTURE  LFA1 OPTIONAL
*”             VALUE(I_SKA1) LIKE  SKA1 STRUCTURE  SKA1 OPTIONAL
*”       TABLES
*”              T_KONTAB STRUCTURE  RFEPK
*”              T_SLBTAB STRUCTURE  RFEPB
*”----------------------------------------------------------------------
```

Listing 4.4 Event 00001630 Interface

Import parameters I_LFA1, I_KNA1, and I_SKA1 will be filled only if an account (general ledger account, customer, or vendor) is selected on the report selection screen; otherwise, these import parameters will be blank.

Listing 4.5 shows the functional interface of event 00001650.

```
*"----------------------------------------------------------------------
*"*"Lokale Schnittstelle:
*"       IMPORTING
*"             VALUE(I_POSTAB) LIKE  RFPOS STRUCTURE  RFPOS
*"       EXPORTING
*"             VALUE(E_POSTAB) LIKE  RFPOS STRUCTURE  RFPOS
*"----------------------------------------------------------------------
```

Listing 4.5 Event 00001650 Interface

Both import and export parameters have the same type. When passing the parameters to this function module, SAP uses the MOVE-CORRESPONDING logic. Make sure the first statement in this implementation is E_POSTAB = I_POSTAB to avoid standard data corruption.

4.2 New SAP General Ledger Account Line-Item Report Enhancements

As we mentioned earlier, general ledger accounts have an advanced version of the line-item report for the new Flexible General Ledger. Its transaction code is FAGLL03. The obvious difference when compared to the classic report is that a user can choose the ledger. And, according to ledger they select, the report will pick items from the corresponding ledger table. Technically, the reports differ in how they gather item data and output results.

Item selection is done by using functional modules of group FAGL_ITEMS_SELECT: FAGL_GET_ITEMS_BSAS, FAGL_GET_ITEMS_BSIS, FAGL_GET_ITEMS_BSEG, and some others. Data output is performed by the FAGL_ITEMS_DISPLAY function module.

FAGL_ITEMS_DISPLAY has similar enhancement capabilities to module FI_ITEMS_DISPLAY, which is used in classical reports; however, instead of some BTE events, the new module employs a BAdI.

4.2.1 Header and Footer Output Enhancement

The header and footer enhancement in Transaction FAGLL03 uses the same BTE event (`00001640`) as the classical line-item report; after you subscribe to that event, your subscription will work both in classic and new report versions. See implementation details in Section 4.1.1, Header and Footer Output Enhancement, earlier in this chapter.

4.2.2 Extended Authorization Check

Because report FAGLL03 can be used to display data from different Flexible General Ledgers, SAP provided a BAdI you can use to implement more complex authorization checks for a particular ledger. The BAdI definition `FAGL_AUTHORITY_CHECK` has a single interface method, `CHECK_LEDGER_AUTHORITY`, and supports only one active implementation (flag MULTIPLE USE is turned off). The method has company code, ledger, ledger group, and activity import parameters and also some others (you can see the full list of parameters in the definition of the method `CHECK_LEDGER_AUTHORITY` of the interface `IF_EX_FAGL_AUTHORITY_CHECK`. The parameter `I_ACTVT` (activity) shows the operation mode (display or change) that the user intends to work with the ledger in. If you've ever worked with authorization, you might know the most commonly used activities to be protected: 01 for create, 02 for change, 03 for display, and so on.

If the check fails, the method must raise a classical exception, `NO_AUTHORITY` (via the system variable `SY-SUBRC`). The exception can be raised without an accompanying message because the calling program ignores it. Also, if you activated the BAdI implementation, then the standard authorization check for ledger (authorization object `F_FAGL_LDR`) is ignored.

Note

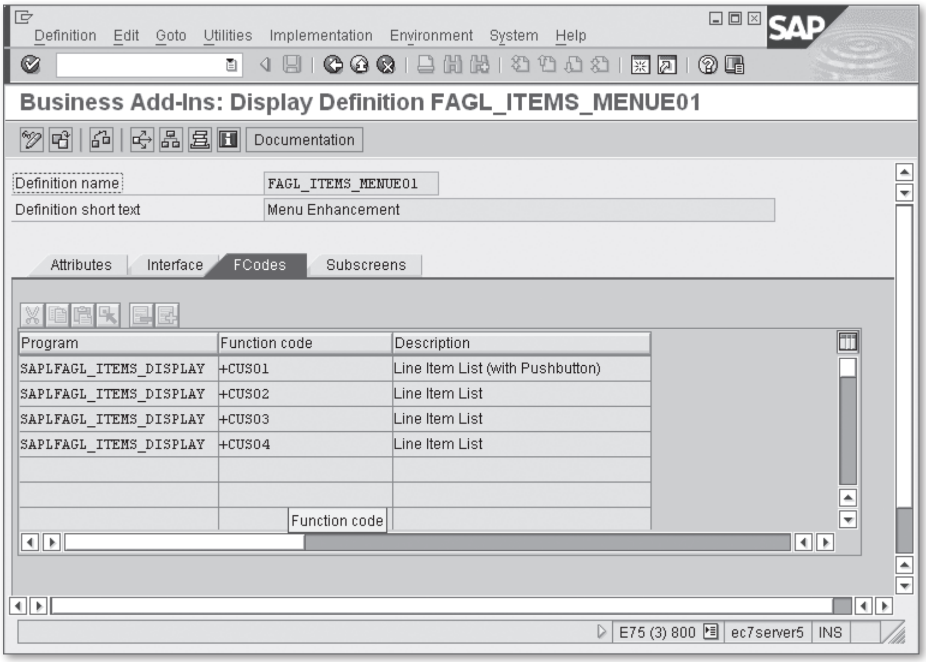
A full list of authorized activities can be found in Table TACT. There are almost 200 values.

4.2.3 Menu Enhancement

Unlike classic line-item reports, Transaction FAGLL03 doesn't provide BTE events for menu enhancement. It uses two BAdI definitions: `FAGL_ITEMS_MENUE01` and `FAGL_ITEMS_MENUE02`, which have the same structure as the classical. Figure

4.9 shows the function codes of the BAdI. Notice that they reference program SAPLFAGL_ITEMS_DISPLAY.

For these BAdI definitions, you can use exactly the same technique as for the classic report BAdI. See the details in Section 4.1.3, Menu Enhancement with BAdI.



Business Add-Ins: Display Definition FAGL_ITEMS_MENUE01

Definition name	FAGL_ITEMS_MENUE01
Definition short text	Menu Enhancement

Program	Function code	Description
SAPLFAGL_ITEMS_DISPLAY	+CUS01	Line Item List (with Pushbutton)
SAPLFAGL_ITEMS_DISPLAY	+CUS02	Line Item List
SAPLFAGL_ITEMS_DISPLAY	+CUS03	Line Item List
SAPLFAGL_ITEMS_DISPLAY	+CUS04	Line Item List

Figure 4.9 BAdI FAGL_ITEMS_MENUE01 Function Codes

4.2.4 Enhancing the Output Layout

The new line-item report uses BAdI definition `FAGL_ITEMS_CH_DATA` for changing the output of the layout, rather than using BTE events `00001630` and `00001650`. Prior to starting the implementation, you have to extend the dictionary structure `FAGLPOSX`, which is used as the ALV output table in a new line-item report. As you can see in Transaction SE11, structure `FAGLPOSX` has customer include `CI_FAGLPOSX` in its definition. By default, the structure is empty, and you can extend it with an arbitrary number of fields.

Because SAP doesn't provide a BAdI or user exit for changing the ALV field catalog for the line-item report, you should extend the include structure with fields supplied with thoroughly defined text labels. The BAdI definition has only one method (CHANGE_ITEMS) with a single changing parameter of table type FAGLPOSX_T.

4.3 Summary

This chapter provided a brief overview of enhancing reports, rather than a comprehensive guide. The SAP Financials functionality provides you with a huge variety of other reports and reporting tools, including country-specific tax reports, dozens of banking reports, and many others. Not all of them can be enhanced using user exits like those we have discussed in this chapter, so sometimes you're better off copying the report into your own Z-report and amending it according to your business needs. Nevertheless, this chapter gave you insight into how to investigate a report's enhancement capability, which sometimes can save a lot of time and effort.

In the next two chapters, we'll discuss the available user exits in the process of accounting data exchange with external systems.

As today's corporate ERP system landscape becomes more and more distributed, you have to be prepared for different kinds of data that can flow to and from external systems. With this in mind, the focus of this chapter is inbound scenarios in Financial Accounting.

5 Inbound Scenarios in Financial Accounting

In this chapter, we consider data processing scenarios when the SAP system receives accounting data from external systems. This can be master data from legacy systems or posting data from, for example, an external payroll system. This chapter describes how you can intervene in this process using various user exits.

5.1 Master Data Migration and Distribution

There could be no SAP ERP implementation project without an initial data migration procedure. Imagine how painful it would be if a company started its trading activity by implementing SAP ERP and then entered its existing customers and vendors one by one. As a rule, the moment a company implements SAP ERP, the customer/vendor list (which is in some other legacy system) has to be prepared. There are also scenarios in which accounting master data are loaded from external systems on a regular basis.

In the following subsections, we'll discuss several ways to load master data into an SAP system and how to seamlessly penetrate the standard data flow to address specific requirements.

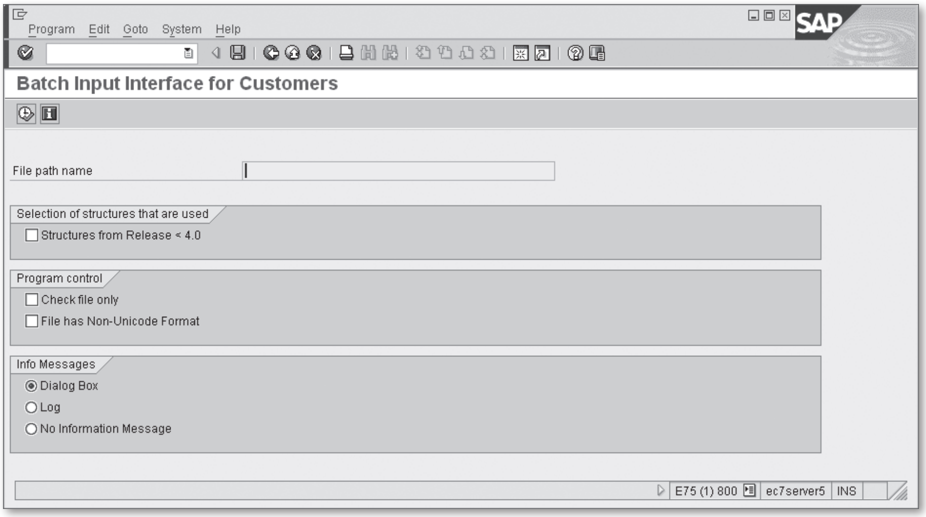
5.1.1 Batch Input

If you are familiar with the SAP Legacy System Migration Workbench (LSMW) and have completed data migration projects, you probably recognize these standard SAP programs for the mass uploading of customer and vendor master records: reports RFBIDE00 and RFBIKR00.

Both reports have the same selection screen as shown in Figure 5.1. Input data for the report must be presented as a flat file located on the application server.

Note

You can also pass a logical file name into the report by passing it through invisible parameter `LDS_NAME`, which can be used in a `SUBMIT` statement. In this case, the value of the visible file path name parameter is ignored.



Batch Input Interface for Customers

File path name

Selection of structures that are used
☐ Structures from Release < 4.0

Program control
☐ Check file only
☐ File has Non-Unicode Format

Info Messages
● Dialog Box
○ Log
○ No Information Message

E75 (1) 800 | ec7server5 | INS

Figure 5.1 Selection Screen of Report RFBIDE00

By default, the maximum length of an input file line is 2,000 characters—this is the length of dictionary structure `BDIFIBIWA`. If your input file has longer lines, you can extend structure `BDIFIBIWA` by using customer include structure `CI_BDIFIBIWA`.

Keep in mind, however, that structure `BDIFIBIWA` only defines the length of an input file line, whereas the actual structure of the data being processed is defined according to the first 31 characters of the line (see the structure shown in Figure 5.2).

The first character of each file line is a *record type*, which can take one of three values: 0, 1, or 2. Record type 0 marks the beginning of a session, record type 1 is the beginning of one customer (or vendor) data for one transaction code, and record type 2 is a data record. The next 30 characters of a file line contain a dictionary structure name. For record type 0, the structure name is always `BGR00`; for record type 1, the structure name is always `BKN00` for customers and `BLF00` for vendors.

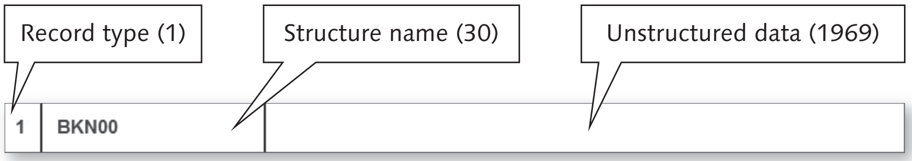


Figure 5.2 The Structure of a Flat File Line

In the record with structure `BGR00`, you can denote the transaction code that will be used to process the data. The record with structure `BKN00` contains the customer number and corresponding organizational assignment, such as company code, sales market data, credit control area, and so on. In the record with structure `BLF00`, the data contains information for the vendor number, company code, purchasing organization, and so on.

File lines with record type 2 can contain standard and nonstandard structures. Standard batch-input structures mainly comply with the following naming convention: character B followed by one of the master data table names. For example, `BKNA1` is a batch-input structure for Table KNA1, `BLFA1` is the batch-input counterpart for LFA1, and so on.

Note

The full list of all standard batch-input structures and supported transactions can be found in SAP online help for reports RFBIDE00 and RFBIKR00.

In the next subsection, you'll see learn to extend data and amend its processing using BAdIs. You'll also see a step-by-step example of loading extended data with a standard SAP program.

Data Enhancement

You can enhance batch-input data either by defining your own fields in the corresponding customer include, which you can find in all standard batch-input structures (e.g., `CI_BKNA1` in `BKNA1`), or by defining your own data structures.

If you choose the second option, follow the same conventions found in the standard structure:

- The first two fields of the customer include should be the same as in the standard structure (STYPE and `TABNAME`).
- All fields must be characters (no numbers).

Note

To make the customer-defined batch-input structure available in SAP LSMW, you must insert a corresponding entry in the customizing table SXDA2.

Using BAdIs

If your custom-defined fields are part of an additional screen layout (see Chapter 2, Master Data Enhancements), then you have to apply user exits to make the system process additional data in customer or vendor loading reports.

Customer loading report RFBIDE00 uses the following BAdI definitions and methods:

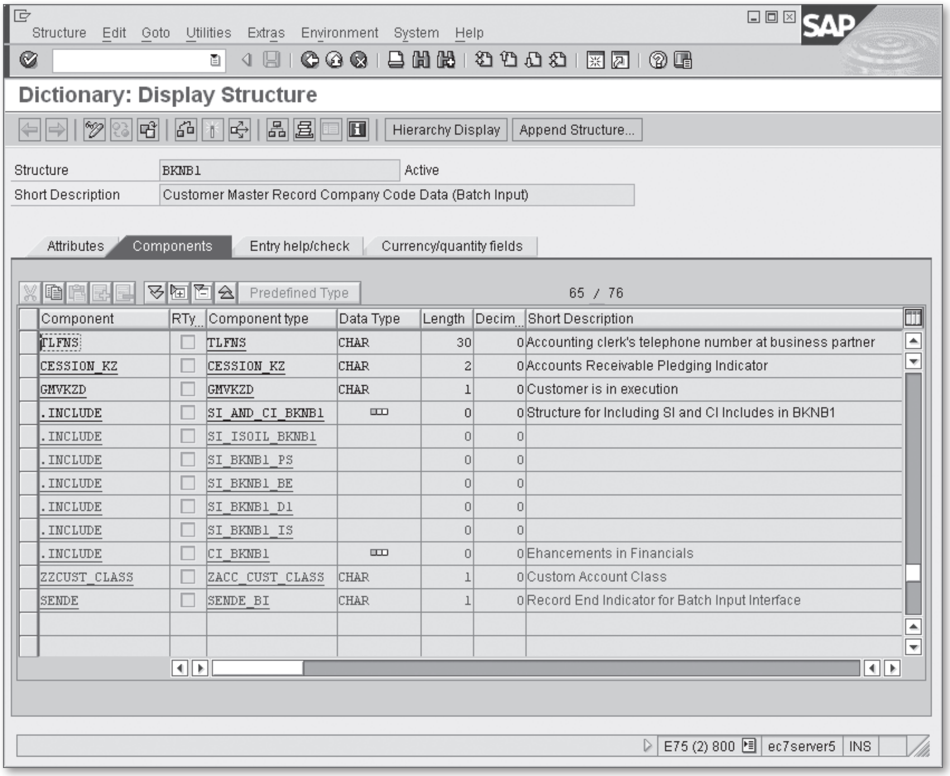
- **Definition:** `CUSTOMER_ADD_DATA`
  - Method `CHECK_ADD_ON_ACTIVE` is called in the initialization phase of the report. Other BAdI methods are called only if at least one add-on is active.
- **Definition:** `CUSTOMER_ADD_DATA_BI`
  - Method `CHECK_DATA_ROW` is called for any nonstandard file line with record type 2 and an unknown structure name. The method can be used to check the input contents for nonstandard structures.
  - Method `FILL_FT_TABLE_USING_DATA_ROWS` is called at the end of transactions processing (only for Transactions XD01 and XD02). The method can be used to amend or extend generated batch-input screens and field sequences to incorporate add-on screens and fields.

Vendor loading report RFBIKR00 uses the following BAdI definitions: `VENDOR_ADD_DATA` and `VENDOR_ADD_DATA_BI`. Method names and their purposes are the same as in report RFBIDE00; and logical method `FILL_FT_TABLE_USING_DATA_ROWS` is only called for Transactions XK01 and XK02.

Example

To illustrate the enhancement usage in our IDES system, let's incorporate the example from Chapter 2, where we enhanced customer master data, into the standard loading program, RFBIDE00. We extended the company code view of the customer master data by an additional field: Custom Account Class (with technical name `KNB1-ZZCUST_CLASS`).

First, we extended the dictionary structure (`BKNB1`) by defining the customer include (`CI_BKNB1`). As a result, the `BKNB1` definition in Transaction SE11 should look like Figure 5.3.



Dictionary: Display Structure

Structure BKNB1 Active
Short Description Customer Master Record Company Code Data (Batch Input)

65 / 76

Component	RTy	Component type	Data Type	Length	Decim	Short Description
TLFNS	☐	TLFNS	CHAR	30	0	Accounting clerk's telephone number at business partner
CESSION_KZ	☐	CESSION_KZ	CHAR	2	0	Accounts Receivable Pledging Indicator
GMVKZD	☐	GMVKZD	CHAR	1	0	Customer is in execution
.INCLUDE	☐	SI_AND_CI_BKNB1	▭	0	0	Structure for Including SI and CI Includes in BKNB1
.INCLUDE	☐	SI_ISOIL_BKNB1		0	0	
.INCLUDE	☐	SI_BKNB1_PS		0	0	
.INCLUDE	☐	SI_BKNB1_BE		0	0	
.INCLUDE	☐	SI_BKNB1_D1		0	0	
.INCLUDE	☐	SI_BKNB1_IS		0	0	
.INCLUDE	☐	CI_BKNB1	▭	0	0	Ehancements in Financials
ZZCUST_CLASS	☐	ZACC_CUST_CLASS	CHAR	1	0	Custom Account Class
SENDE	☐	SENDE_BI	CHAR	1	0	Record End Indicator for Batch Input Interface

E75 (2) 800 | ec7server5 | INS

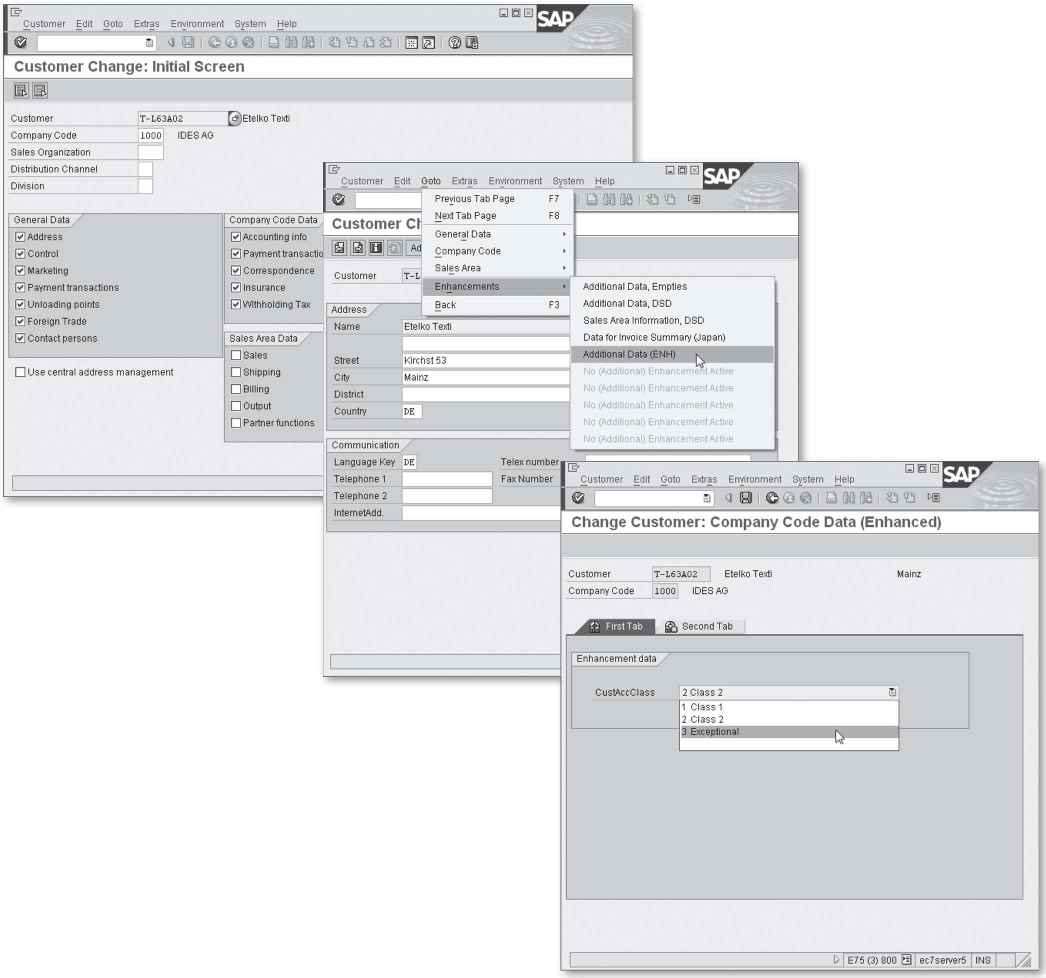
Figure 5.3 Extended BKNB1 Dictionary Structure

When preparing the example for Chapter 2, we implemented BAdI `CUSTOMER_ADD_DATA`. Now we need to use BAdI definition `CUSTOMER_ADD_DATA_BI`. Because we haven't created our own batch-input structure, but extended a standard structure instead, we don't need to implement the `CHECK_DATA_ROW` method. We do need to code an addition to the screen and field sequence, which will save our data into the customer master record. To do this, we need to examine how the screen sequence might look by using an old batch-input recording, which can be found in Transaction SHDB.

We record the following actions of Transaction XD02 with the following steps:

1. Enter the customer number and company code.
2. Select the enhanced screen layout (defined in Chapter 2).
3. Change the value in the CustAccClass field (no matter from which to which; we just need a value change).
4. Save.

Figure 5.4 shows the combined sequence of screenshots of these steps.



Customer Change: Initial Screen

Customer	T-L63A02 Etelko Texti
Company Code	1000 IDES AG
Sales Organization	
Distribution Channel	
Division	

General Data: ☑ Address ☑ Control ☑ Marketing ☑ Payment transactions ☑ Unloading points ☑ Foreign Trade ☑ Contact persons

Company Code Data: ☑ Accounting info ☑ Payment transactio ☑ Correspondence ☑ Insurance ☑ Withholding Tax

Sales Area Data: ☐ Sales ☐ Shipping ☐ Billing ☐ Output ☐ Partner functions

☐ Use central address management

Customer Ch

Customer	T-L
Name	Etelko Texti
Street	Kirchst 53
City	Mainz
District	
Country	DE
Language Key	DE
Telephone 1	
Telephone 2	
InternetAdd.	
Telex number	
Fax Number	

Change Customer: Company Code Data (Enhanced)

Customer	T-L63A02 Etelko Texti Mainz
Company Code	1000 IDES AG

Enhancement data

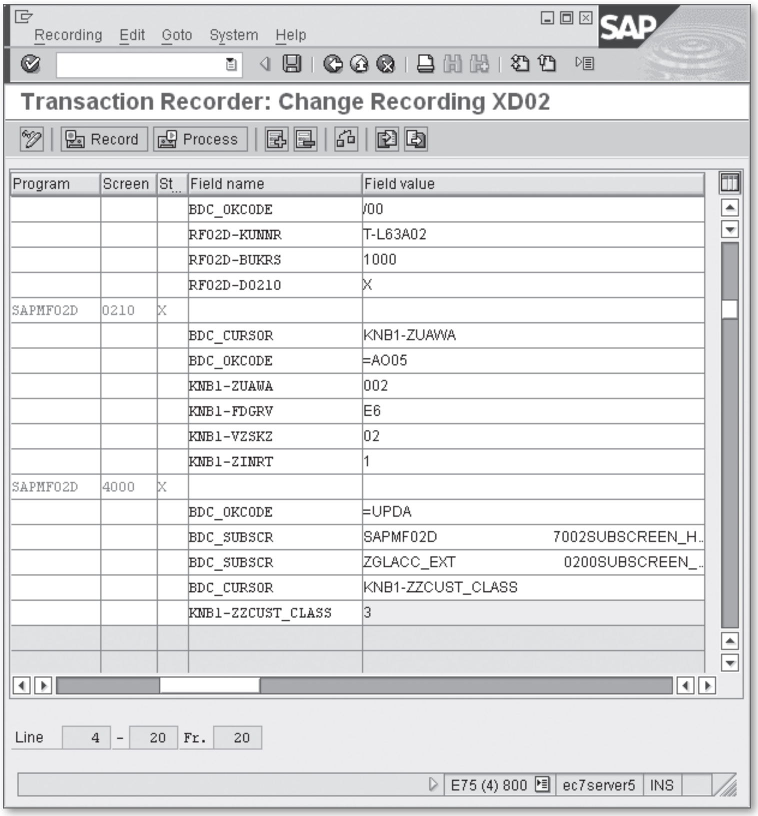
CustAccClass 2 Class 2

1 Class 1
2 Class 2
3 Exceptional

E75 (3) 800 ec7server5 INS

Figure 5.4 Recorded Screen Sequence of Transaction XD02

The result of the recording is shown in Figure 5.5.



Transaction Recorder: Change Recording XD02

Program	Screen	St..	Field name	Field value
			BDC_OKCODE	/00
			RF02D-KUNNR	T-L63A02
			RF02D-BUKRS	1000
			RF02D-D0210	X
SAPMF02D	0210	X		
			BDC_CURSOR	KNB1-ZUAWA
			BDC_OKCODE	=AO05
			KNB1-ZUAWA	002
			KNB1-FDGRV	E6
			KNB1-VZSKZ	02
			KNB1-ZINRT	1
SAPMF02D	4000	X		
			BDC_OKCODE	=UPDA
			BDC_SUBSCR	SAPMF02D 7002SUBSCREEN_H..
			BDC_SUBSCR	ZGLACC_EXT 0200SUBSCREEN_..
			BDC_CURSOR	KNB1-ZZCUST_CLASS
			KNB1-ZZCUST_CLASS	3

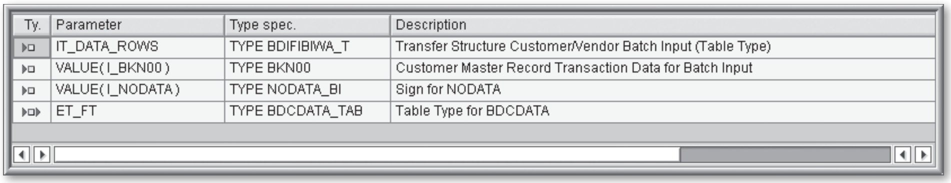
Line 4 - 20 Fr. 20

E75 (4) 800 | ec7server5 | INS

Figure 5.5 The Recording of Transaction XD02

As you analyze the recording, you see that on the starting data screen SAPMFD02/200, we executed function code `AO05`, which has taken us into the enhanced screen layout. There we entered a value of 3 into the field KNB1-ZZCUST_CLASS and clicked Save (function code `UPDA`).

Now we are ready to implement the code of method `FILL_FT_TABLE_USING_DATA_ROWS`.



Ty.	Parameter	Type spec.	Description
	IT_DATA_ROWS	TYPE BDIFIBIWA_T	Transfer Structure Customer/Vendor Batch Input (Table Type)
	VALUE(I_BKN00)	TYPE BKN00	Customer Master Record Transaction Data for Batch Input
	VALUE(I_NODATA)	TYPE NODATA_BI	Sign for NODATA
	ET_FT	TYPE BDCDATA_TAB	Table Type for BDCDATA

Figure 5.6 The Interface of Method FILL_FT_TABLE_USING_DATA_ROWS

Figure 5.6 shows the interface of method `FILL_FT_TABLE_USING_DATA_ROWS`. You can see that we have current `BKN00` data (with customer number and other organizational assignment data) as input parameter `I_BKN00`; we also have all file lines related to the current transaction in input parameter `IT_DATA_ROWS`. Finally, we have one export table typed parameter, `ET_FT`, which we will amend according to our logic. `ET_FT` has line type of `BDCDATA` structure, which is a well-known structure used in batch-input statement `CALL TRANSACTION USING`.

The algorithm should do the following:

- Find the first entry of structure `BKNB1` in the file data.
- Insert function code `AO05` into the previous screen: BDC data.
- Start a new screen in BDC data.
- Set new field values according to `BKNB1` contents that were found.

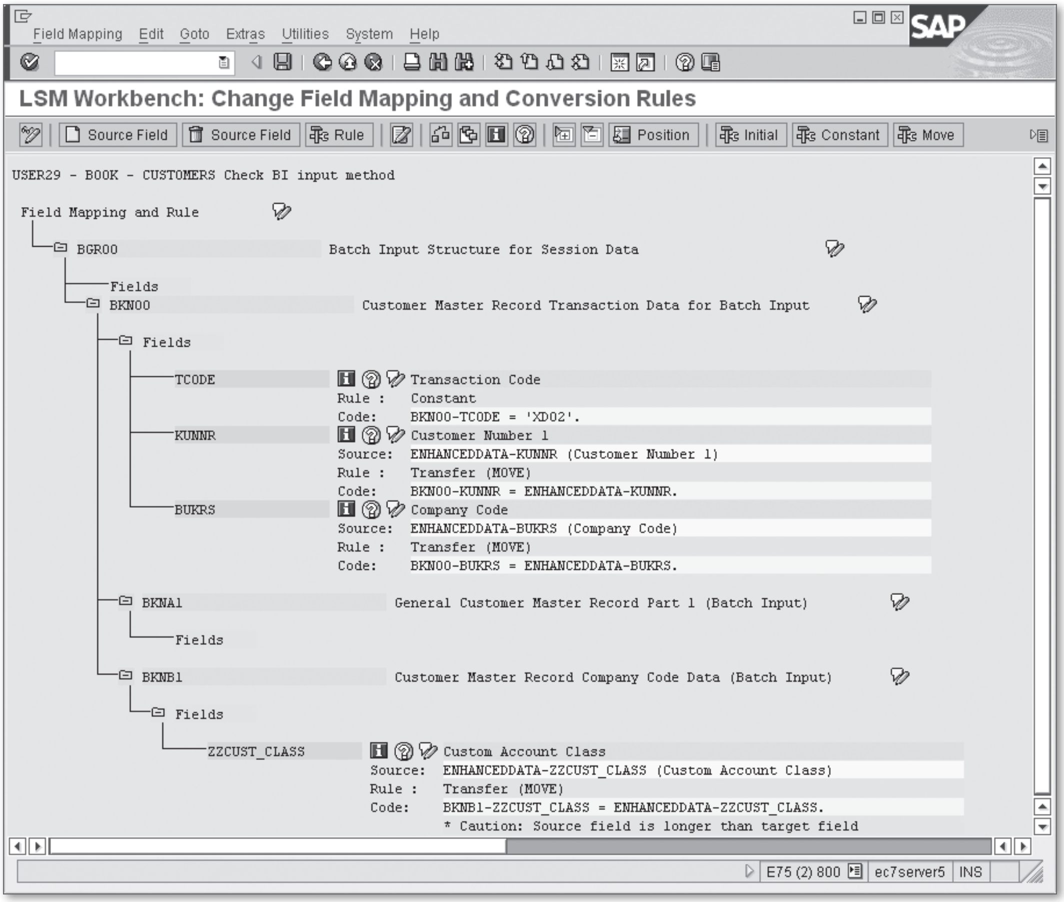
Always keep in mind that there can be other active BAdI implementations, so you shouldn't include any function codes in the batch input because this can end the transaction. In our example, we don't insert the function code `UPDA`, which is seen in our sample recording (refer back to Figure 5.5).

Listing 5.1 shows the source code of our method implementation.

```
METHOD if_ex_customer_add_data_bi~fill_ft_table_using_data_rows.
  FIELD-SYMBOLS: <wa> TYPE bknb1.

  DATA: ft TYPE bdcdata.
  LOOP AT it_data_rows ASSIGNING <wa> CASTING.
    CHECK <wa>-stype = '2' AND <wa>-tbnam = 'BKNB1'.

*   Insert function code to select Enhanced screen layout
*   This will be added to the last processed screen in BDC data
    CLEAR ft.
    ft-fnam = 'BDC_OKCODE'.
    ft-fval = '=AO05'.
    APPEND ft TO et_ft.

*   Start new screen
    CLEAR ft.
    ft-program = 'SAPMF02D'.
    ft-dynpro = '4000'.
    ft-dynbegin = 'X'.
    APPEND ft TO et_ft.

*   Enter field value on the custom defined screen
    CLEAR ft.
    ft-fnam = 'KNB1-ZZCUST_CLASS'.
    ft-fval = <wa>-zzcust_class.
    APPEND ft TO et_ft.

    EXIT.
  ENDLOOP.
ENDMETHOD.
```

Listing 5.1 Method FILL_FT_TABLE_USING_DATA_ROWS Source

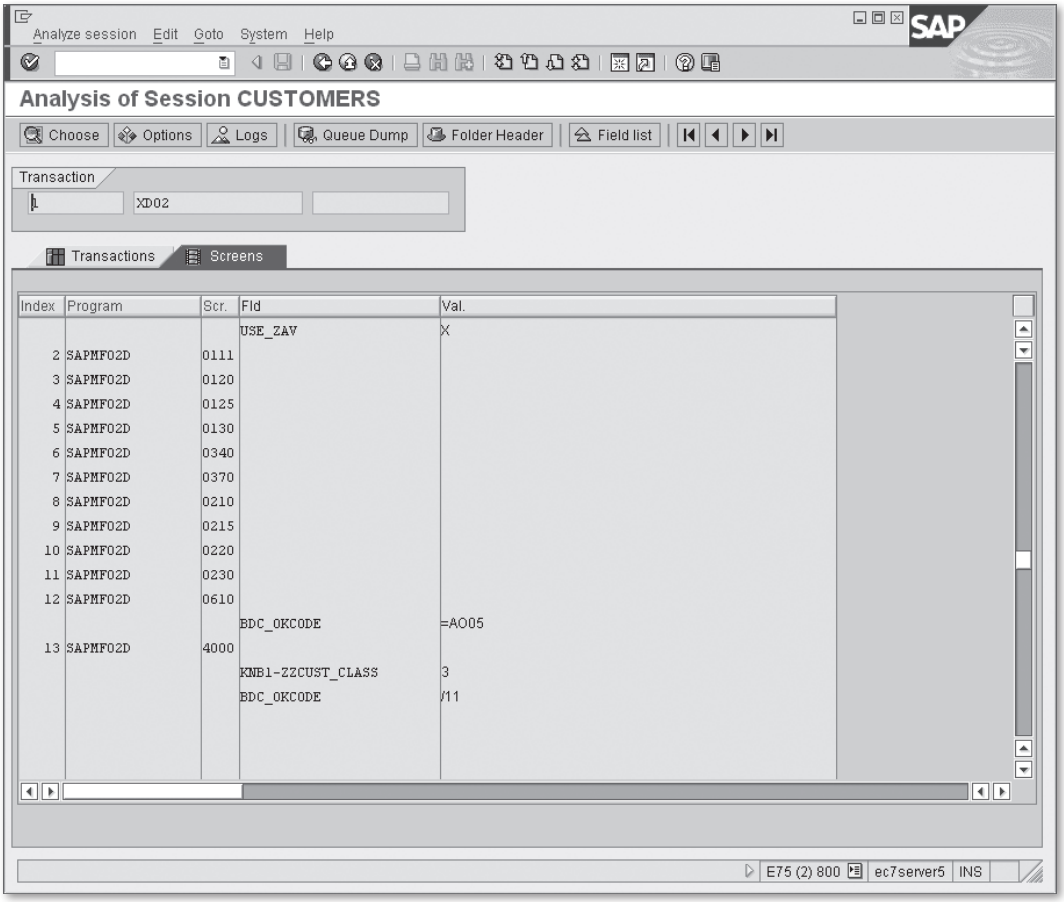
After activating the BAdI implementation, we can now test the new fields with a small SAP LSMW project. The goal of this project is to update field `KNB1-ZZCUST_CLASS` using the batch-input loading program RFBIDE00. After defining the appropriate target object and source structure, you can see in the SAP LSMW field-mapping step that our field is included in the target structure (see Figure 5.7). Note that all uninitialized fields are turned off to make the view more compact.


Field Mapping Edit Goto Extras Utilities System Help
LSM Workbench: Change Field Mapping and Conversion Rules
Source Field
Source Field
Rule
Position
Initial
Constant
Move
E75 (2) 800
ec7server5
INS

```
USER29 - BOOK - CUSTOMERS Check BI input method

Field Mapping and Rule
 └─ BGR00                 Batch Input Structure for Session Data
     ├──Fields
     └─ BKN00             Customer Master Record Transaction Data for Batch Input
         ├─ Fields
         │   ├──TCODE          Transaction Code
         │   │                 Rule :   Constant
         │   │                 Code:    BKN00-TCODE = 'XD02'.
         │   ├──KUNNR          Customer Number 1
         │   │                 Source:  ENHANCEDDATA-KUNNR (Customer Number 1)
         │   │                 Rule :   Transfer (MOVE)
         │   │                 Code:    BKN00-KUNNR = ENHANCEDDATA-KUNNR.
         │   └──BUKRS          Company Code
         │                     Source:  ENHANCEDDATA-BUKRS (Company Code)
         │                     Rule :   Transfer (MOVE)
         │                     Code:    BKN00-BUKRS = ENHANCEDDATA-BUKRS.
         ├─ BKNA1             General Customer Master Record Part 1 (Batch Input)
         │   └──Fields
         └─ BKNB1             Customer Master Record Company Code Data (Batch Input)
             └─ Fields
                 └──ZZCUST_CLASS    Custom Account Class
                                    Source:  ENHANCEDDATA-ZZCUST_CLASS (Custom Account Class)
                                    Rule :   Transfer (MOVE)
                                    Code:    BKNB1-ZZCUST_CLASS = ENHANCEDDATA-ZZCUST_CLASS.
                                             * Caution: Source field is longer than target field
```

Figure 5.7 LSMW Field Mapping View for the Customer Master

The Create Batch Input Session step in the SAP LSMW project is actually a call of the program RFBIDE00. We tested it with only one record in the input file to update customer T-L63A02 in company code 1000. Now change the CustAccClass field to 3. After generating the batch-input session, we can inspect it in Transaction SM35. Figure 5.8 shows the screen list of the session with an opened field value list. Our added field is in its place.



Transaction: 1 XD02

```
Index Program   Scr.  Fld                 Val.
                      USE_ZAV             X
    2 SAPMF02D  0111
    3 SAPMF02D  0120
    4 SAPMF02D  0125
    5 SAPMF02D  0130
    6 SAPMF02D  0340
    7 SAPMF02D  0370
    8 SAPMF02D  0210
    9 SAPMF02D  0215
   10 SAPMF02D  0220
   11 SAPMF02D  0230
   12 SAPMF02D  0610
                      BDC_OKCODE          =AO05
   13 SAPMF02D  4000
                      KNB1-ZZCUST_CLASS   3
                      BDC_OKCODE          /11
```

Figure 5.8 Batch-Input Session Analysis in Transaction SM35

5.1.2 HR Master Data

In some HR payroll instances, an employee has his own HR master record, which generates a corresponding vendor master record or customer master record for that employee in the financials department of the company. From the formal accounting point of view, when the company pays the salary to that employee, he should be treated as a company vendor because that employee sells his services to the company (in the form of an everyday job). If HR Payroll and FI are installed as separate systems, you must set up a task of regularly distributing HR employee data into an FI system to form vendor or customer master records.

In brief, the standard process of HR data distribution, which is based on ALE (application link enabling) technology, looks as follows:

1. Several structures of employee data (called *infotypes*) from the external HR system are copied into the FI system, in the form of an IDoc (depending on the HR system version, it can be an IDoc type from `HRMD_A01` to `HRMD_A07`).
2. The receiving FI system regularly runs report RPRAPA00, which prepares the locally available HR data for loading with the standard report RFBIKR00.
3. Inside report RPRAPA00, a BAdI definition `BADI_EXITS_RPRAPA00` is used to intercept the standard logic when preparing a data file for the following run of the report RFBIKR00. The list of available BAdI methods is shown in Table 5.1.

Method	Description
`SET_VALUES_FOR_BLFBW`	Exit for `BLFBW`: Vendor master, withholding tax types
`SET_VALUES_FOR_BLF00`	Exit for `BLF00`: Vendor master
`SET_VALUES_FOR_BLFA1`	Exit for `BLFA1`: Vendor master, general data part 1
`SET_VALUES_FOR_BLFBK`	Exit for `BLFBK`: Vendor master, bank details
`SET_VALUES_FOR_BLFB1`	Exit for `BLFB1`: Vendor master, company code data
`SET_VALUES_FOR_BLFB5`	Exit for `BLFB5`: Vendor master, dunning data
`SET_VALUES_FOR_BGR00`	Exit for `BGR00`: Batch-input structure for session data

Table 5.1 Interface Methods of the BAdI Definition BADI_EXITS_RPRAPA00

Each method has an employee number (`PERNR`) as an input parameter and a respective batch-input structure as a changing parameter. The structure name is clearly shown by the method name.

Because report RPRAPA00 works on the local HR data, you can use standard HR functionality to access employee infotypes. All the BAdI methods are called in the end of each employee number processing.

5.1.3 ALE/IDoc

The batch-input data loading techniques discussed earlier are based on a file as a data carrier. This is a somewhat outdated technology, and while it is robust and stable, it's less flexible and less secure compared to ALE/IDoc technology. IDoc processing logic is completely separated from the data transferring media, which is much more suitable to the modern distributed environments with its variety of data transferring protocols. In essence, ALE/IDoc technology is more welcome in modern integration projects involving B2B (business to business), A2A (application to application), and mobile scenarios.

When it comes to making a decision on what type of technology to employ in an integrating project of almost any nature, we recommend choosing IDocs over files. ALE/IDoc technology is highly configurable, and depending on corporate-specific requirements, you can completely intercept the IDoc processing of any individual type.

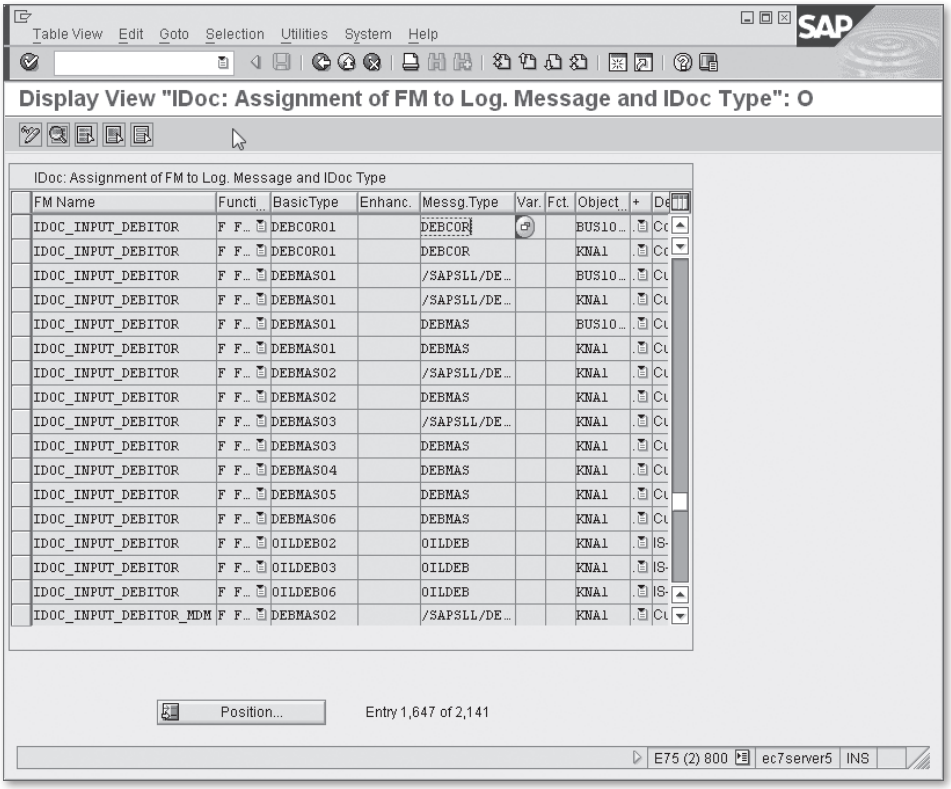
The Structure of an IDoc in a Nutshell

The structure of an IDoc is identified by its basic type, which is an ordered set of segments. For simplicity, the notion of an IDoc segment can be treated as an equivalent of the dictionary structure. Basic type defines not only a simple order of its segments but also their hierarchy relations, cardinality, and necessity. In other words, the basic type defines the syntax of IDoc, which is controlled by the runtime ALE system layer. The IDoc basic type structure can be displayed using Transaction WE30.

For the sake of simplicity, we can also say that a pair of objects—logical message code and basic type—together define IDoc processing logic via assignment to a specific ABAP function module, workflow template, or task. These assignments are stored in configuration table EDIFCT, which is accessible via Transaction WE57.

SAP delivers the following logical messages for master data distribution via ALE: `CREMAS` and `CRECOR` for vendors, and `DEBMAS` and `DEBCOR` for customers. Figure 5.9 shows the IDoc processing module configuration for customer-related messages and IDoc types.

If you look into the default IDoc configuration table EDIFCT (via Transaction WE57), you can see that standard processing logic for inbound IDoc transferring customer and vendor master data is hidden in two function modules: `IDOC_INPUT_DEBITOR` and `IDOC_INPUT_CREDITOR`. These function modules are assumed to process IDoc basic types from `CREMAS01` to `CREMAS05`, and from `DEBMAS01` to `DEBMAS06`, `CRECOR01`, and `DEBCOR01`. In this notation, the numeric suffix is the version of the IDoc structure.



Display View "IDoc: Assignment of FM to Log. Message and IDoc Type": O

IDoc: Assignment of FM to Log. Message and IDoc Type

FM Name	Functi...	BasicType	Enhanc.	Messg.Type	Var.	Fct.	Object...	+	De
IDOC_INPUT_DEBITOR	F F...	DEBCOR01		DEBCOR			BUS10...	.	Cc
IDOC_INPUT_DEBITOR	F F...	DEBCOR01		DEBCOR			KNA1	.	Cc
IDOC_INPUT_DEBITOR	F F...	DEBMAS01		/SAPSLL/DE...			BUS10...	.	Cu
IDOC_INPUT_DEBITOR	F F...	DEBMAS01		/SAPSLL/DE...			KNA1	.	Cu
IDOC_INPUT_DEBITOR	F F...	DEBMAS01		DEBMAS			BUS10...	.	Cu
IDOC_INPUT_DEBITOR	F F...	DEBMAS01		DEBMAS			KNA1	.	Cu
IDOC_INPUT_DEBITOR	F F...	DEBMAS02		/SAPSLL/DE...			KNA1	.	Cu
IDOC_INPUT_DEBITOR	F F...	DEBMAS02		DEBMAS			KNA1	.	Cu
IDOC_INPUT_DEBITOR	F F...	DEBMAS03		/SAPSLL/DE...			KNA1	.	Cu
IDOC_INPUT_DEBITOR	F F...	DEBMAS03		DEBMAS			KNA1	.	Cu
IDOC_INPUT_DEBITOR	F F...	DEBMAS04		DEBMAS			KNA1	.	Cu
IDOC_INPUT_DEBITOR	F F...	DEBMAS05		DEBMAS			KNA1	.	Cu
IDOC_INPUT_DEBITOR	F F...	DEBMAS06		DEBMAS			KNA1	.	Cu
IDOC_INPUT_DEBITOR	F F...	OILDEB02		OILDEB			KNA1	.	IS-
IDOC_INPUT_DEBITOR	F F...	OILDEB03		OILDEB			KNA1	.	IS-
IDOC_INPUT_DEBITOR	F F...	OILDEB06		OILDEB			KNA1	.	IS-
IDOC_INPUT_DEBITOR_MDM	F F...	DEBMAS02		/SAPSLL/DE...			KNA1	.	Cu

Entry 1,647 of 2,141

E75 (2) 800 | ec7server5 | INS

Figure 5.9 The Contents of Table EDIFCT

Both function modules work the same way. They first analyze the system type; if it's an ERP system, the function modules call an ERP-specific function: `ERP_IDOC_INPUT_CREDITOR` for a vendor and `ERP_IDOC_INPUT_DEBITOR` for a customer. There is also a function call for a standalone HR system, but it's quite simple. Because HR doesn't need any advanced customer or vendor master data manipulations, you'll find just a direct update of the corresponding tables.

The main secret of standard IDoc processing logic is that it updates or creates individual master record by means of batch input. If you dive into the source code of `ERP_IDOC_INPUT_DEBITOR` or `ERP_IDOC_INPUT_CREDITOR`, you'll find the corresponding `CALL TRANSACTION` statement. In a way, they repeat the logic of reports RFBIDE00 and RFBIKR00; but instead of a flat file, these functions process IDocs, and each segment can be treated as an equivalent of a file line. You can also see that after processing IDoc segments, the function gathers information into an internal table of structure BDIFIBIWA.

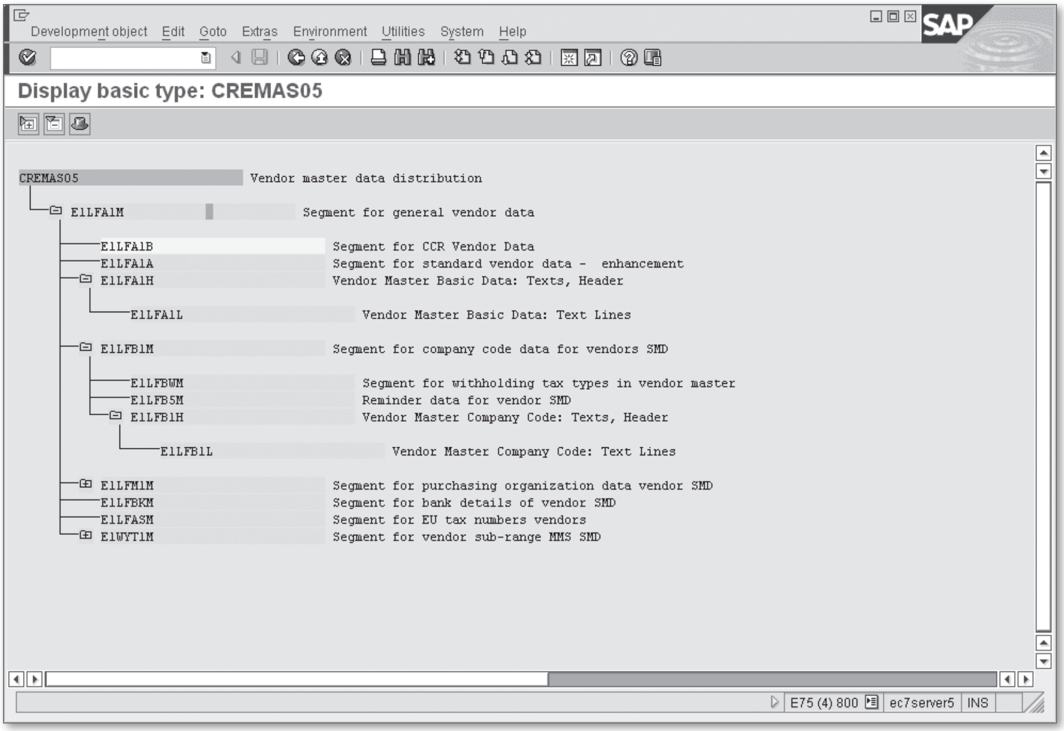
Note

In IDoc processing, SAP provides calling moments for the same BAdI definition as in RFBIDE00 and RFBIKR00.

Next, we'll discuss working with IDoc data structures—segments—and how you can affect the processing logic in standard SAP functions.

Working with Segments

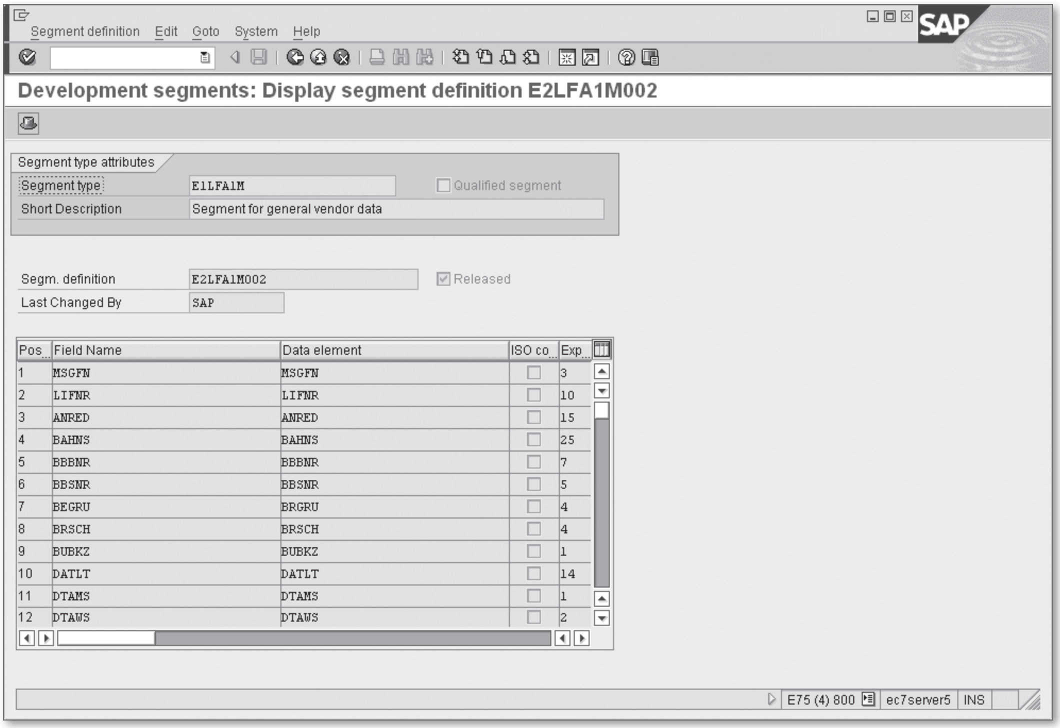
The structure of the IDoc type you are planning to process can be displayed in Transaction WE30. Figure 5.10 shows the structure of IDoc basic type CREMAS05. As you can see, there are three levels of segment hierarchy.



```
CREMAS05                Vendor master data distribution
  E1LFA1M               Segment for general vendor data
    E1LFA1B             Segment for CCR Vendor Data
    E1LFA1A             Segment for standard vendor data -  enhancement
    E1LFA1H             Vendor Master Basic Data: Texts, Header
      E1LFA1L           Vendor Master Basic Data: Text Lines
    E1LFB1M             Segment for company code data for vendors SMD
      E1LFBWM           Segment for withholding tax types in vendor master
      E1LFB5M           Reminder data for vendor SMD
      E1LFB1H           Vendor Master Company Code: Texts, Header
        E1LFB1L         Vendor Master Company Code: Text Lines
    E1LFM1M             Segment for purchasing organization data vendor SMD
    E1LFBKM             Segment for bank details of vendor SMD
    E1LFASM             Segment for EU tax numbers vendors
    E1WYT1M             Segment for vendor sub-range MMS SMD
```

Figure 5.10 The Structure of IDoc Type CREMAS05

By double-clicking on an arbitrary segment name, you can drill down to the segment editor where you can see the list of segment fields. You can see an example of segment structure in the segment editor in Figure 5.11.



Development segments: Display segment definition E2LFA1M002

Segment type attributes

Segment type	E1LFA1M	☐ Qualified segment
Short Description	Segment for general vendor data	

Segm. definition	E2LFA1M002	✓ Released
Last Changed By	SAP	

Pos	Field Name	Data element	ISO co	Exp
1	MSGFN	MSGFN	☐	3
2	LIFNR	LIFNR	☐	10
3	ANRED	ANRED	☐	15
4	BAHNS	BAHNS	☐	25
5	BBBNR	BBBNR	☐	7
6	BBSNR	BBSNR	☐	5
7	BEGRU	BRGRU	☐	4
8	BRSCH	BRSCH	☐	4
9	BUBKZ	BUBKZ	☐	1
10	DATLT	DATLT	☐	14
11	DTAMS	DTAMS	☐	1
12	DTAWS	DTAWS	☐	2

E75 (4) 800 | ec7server5 | INS

Figure 5.11 The Structure of the Segment E1LFA1M

When you develop a brand new segment, the final point of the development is the act of releasing the segment. At the moment of release, the system generates a dictionary structure with the same name and all of the segment's fields, which means that the segment can be used officially. All standard segments also have a dictionary structure of the same name. So if an IDoc type defined in your system contains a segment `E1LFA1M`, you can declare a variable in your program of the type `E1LFA1M`.

IDoc has a single primary key field—its 16-digit number. We recommend accessing an individual IDoc by the standard function module `IDOC_READ_COMPLETELY`. Besides the control data (which are outside of our current discussion), the function returns all of the IDoc segments in the form of an internal table of structure, `EDIDD`.

Each record contains exactly one segment; the segment's name is stored in field SEGNAM, while segment data are located in an unstructured field, SDATA. An example of a code snippet for IDoc segment processing is provided in Listing 5.2.

```
DATA: lt_edidd TYPE TABLE OF edidd,
      IDoc_number TYPE edidc-docnum,
      ls_e1lfa1m_segment TYPE e1lfa1m,
      ls_e1lfb1m_segment TYPE e1lfb1m.

FIELD-SYMBOLS: <edidd> TYPE edidd.

CALL FUNCTION 'IDOC_READ_COMPLETELY'
  EXPORTING
    document_number = IDoc_number
  TABLES
    int_edidd       = lt_edidd
  EXCEPTIONS
    OTHERS          = 3.

IF sy-subrc <> 0.
  MESSAGE ID sy-msgid TYPE sy-msgty NUMBER sy-msgno
          WITH sy-msgv1 sy-msgv2 sy-msgv3 sy-msgv4.
ENDIF.

LOOP AT lt_edidd ASSIGNING <edidd>.
  CASE <edidd>-segnam.
    WHEN 'E1LFA1M'.
      ls_e1lfa1m_segment = <edidd>-sdata.
*     Processing...

    WHEN 'E1LFB1M'.
      ls_e1lfb1m_segment = <edidd>-sdata.
*     Processing...

      ......

    WHEN OTHERS.
*     Processing non-standard segments...

  ENDCASE.
ENDLOOP.
```

Listing 5.2 IDoc Segment Processing Code

Note that you can freely use direct assignment between unstructured field EDIDD-SDATA and the structured field of the segment despite the Unicode. This is possible because IDoc segment structure contains only character fields; EDIDD-SDATA is character typed as well.

Available BAdIs in Customer Data IDoc Processing

SAP standard batch-input program RFBIDE00 is called from within the function module ERP_IDOC_INPUT_DEBITOR. First, it calls method CHECK_ADD_ON_ACTIVE of BAdI definition CUSTOMER_ADD_DATA. All other methods of the BAdI definition CUSTOMER_ADD_DATA_BI are only called if there is at least one active add-on.

During the IDoc processing, function ERP_IDOC_INPUT_DEBITOR invokes the following methods of the BAdI definition CUSTOMER_ADD_DATA_BI:

- PASS_NON_STANDARD_SEGMENT
  This method is called when the system encounters an unknown segment during the main loop of IDoc segments processing. This call allows you to convert a nonstandard segment into an internal structure for later processing. The segment name and segment data are passed to the method as import parameters.
- MODIFY_BI_STRUCT_FROM_STD_SEG
  This method is called after fulfilling all standard processing for each standard segment. The method uses the segment name and segment data as import parameters, and one changing parameter with an already known structure, BDIFIBIWA. By the moment of the call, structure BDIFIBIWA is filled with standard values, and you can change it according to your requirements.
- FILL_BI_TABLE_WITH_OWN_SEGMENT
  This method is called when all standard batch-input data are saved into the internal table of structure BDIFIBIWA. This method has a changing table parameter with this structure and an import parameter of dictionary structure CUSTOMER_ORG_DATA. When this method is called, you should process the data that were prepared earlier and saved by the PASS_NON_STANDARD_SEGMENT method.
- CHECK_DATA_ROW
  When all segments are processed and all of the data gathered into the batch-input table of structure BDIFIBIWA, the system checks the data before starting the batch input. This method is called for each line of batch-input data if it contains the name of a nonstandard structure. The method has import parameter of structure BDIFIBIWA and a flag parameter for passing the data check status ("X"

for success and a blank space for failure). If some of the data have not passed the check, the method can return an error message through the corresponding export parameters.

- `FILL_FT_TABLE_USING_DATA_ROWS`
  This method is called just before calling the transaction in batch-input mode. It allows the user to make final alterations into the batch-input screen and field value sequence. Note that this method is only called if the transaction to be called is either XD01 or XD02. This method has a changing table parameter typed with structure `BDCDATA`.

Available BAdIs in Vendor Data IDoc Processing

Function module `ERP_IDOC_INPUT_CREDITOR` works with BAdIs in a slightly different way: It calls BAdI `VENDOR_ADD_DATA` and method `CHECK_ADD_ON_ACTIVE` after gathering information into an intermediary internal table of structure `BDIFIBIWA`, instead of at the beginning of IDoc processing.

The following methods of BAdI definition `VENDOR_ADD_DATA_BI` are called during the processing of the vendor master IDoc:

- `PASS_NON_STANDARD_SEGMENT`
- `MODIFY_BI_STRUCT_FROM_STD_SEG`
- `FILL_BI_TABLE_WITH_OWN_SEGMENT`
- `CHECK_DATA_ROW`
- `FILL_FT_TABLE_USING_DATA_ROWS`

Enhancement Spots

Function group `VV02` has two entries of enhancement spot `ES_SAPLVV02CORE`. One source code plug-in entry of this spot is located in the top include of the function group and allows you to use your own includes here. Another spot entry can be found at the beginning of the function code `ERP_IDOC_INPUT_DEBITOR`. On the vendor side, there is an enhancement spot—`ES_SAPLKD02`—with the same functionality.

Function Module Exits

There are also components of old-styled function module exit `VSV00001`, which you can examine in Transaction SMOD. Customer function `EXIT_SAPLKD02_001` is called after the vendor data IDoc is completely processed and allows you to save additional data in the database. Customer function EXIT_SAPLVV02_001 has the same purpose; it is called after processing the customer data IDoc.

5.2 Postings Inbound Scenarios

Now let's examine how accounting document data can come from the external world and what we can do with it.

5.2.1 Batch-Input or Direct Input

As with master data, an initial stage of an SAP ERP implementation project virtually always requires loading initial accounting transaction data. A traditional tool for this activity is the standard SAP report RFBIBL00. Input data for the report are provided in the form of a flat file located on the application server. The report is suitable for use with SAP LSMW, which effectively hides all the file preparation issues.

Internally, the report uses function modules of group FIPI, which are listed in Table 5.2.

Name	Description
POSTING_INTERFACE_CLEARING	Post with clearing (FB05) using internal posting interface.
POSTING_INTERFACE_DOCUMENT	Post document using the internal posting interface.
POSTING_INTERFACE_END	The ending function of the group. Should be called in the end of the process.
POSTING_INTERFACE_RESET_CLEAR	Reset clearing via posting interface.
POSTING_INTERFACE_REVERSE_DOC	Cancel document via posting interface.
POSTING_INTERFACE_START	Initial information for internal accounting interface.

Table 5.2 FIPI Function Group Modules

These functions actually make postings through batch input by generating sessions or calling a transaction directly. The function modules also have detailed system documentation. Unfortunately, report RFBIBL00 does not contain a user-exit call, although you can rely on the user exits available inside the transactions that are called during processing.

5.2.2 Payroll Results

Note that the payroll result posting interface is fully equipped with specific user exits. However, it's worth seeing the overall process outline so that you can understand where and when the process should (or should not) be intercepted, depending on your business requirements.

If the company has SAP HR Payroll implemented, then in every payroll period (weekly or monthly), there must be an interface running that posts payroll results to the Financials department. SAP recommends implementing HR as a separate system to improve data security because payroll data are among the most sensitive corporate data.

If you are implementing a payroll results posting from SAP HR into SAP FI, then in the end, the posting will be performed with the same tools.

The whole process of HR payroll posting looks like this:

1. The responsible person in HR creates a payroll posting run with report RPCIPE00. The report creates a preliminary posting document stored in Tables PPDHD, PPDIT, and others.
2. Someone then checks and approves all of the resulting posting documents (they are not accounting documents) by editing particular payroll runs with Transaction PCP0.
3. Finally, someone runs report RPCIPP00 to transfer values into accounting.

The last step can be performed either via ALE/IDoc interfaces (if HR Payroll works as a separate system), or locally—by direct call of an accounting BAPI. By default, all of HR Payroll IDocs are processed in the receiving system by the same BAPI. Let's trace the chain.

The HR system generates three types of postings:

- **Employee expenses**
  For example, travel and accommodation when on a business trip.
- **Employee vendor items**
  For example, an employee can be treated as a corporate vendor or service provider to justify salary payment; thus the document is generated as an Account Payables item.

- **Employee customer items**
  If an employee has debts that are not settled, he might appear in the role of a corporate customer; the document is generated as an Account Receivables item.

If an HR Payroll component is implemented as a separate system, then it generates three types of IDocs: ACC_EMPLOYEE_PAY02, ACC_EMPLOYEE_REC02, and ACC_EMPLOYEE_EXP02. In the receiving system, these IDocs are linked by default via the ALE/BAPI-generated interface to the following function modules:

- IDOC_INPUT_ACC_EMPLOYEE_EXP for employee expenses
- IDOC_INPUT_ACC_EMPLOYEE_PAY for employee payments
- IDOC_INPUT_ACC_EMPLOYEE_REC for employee debts

The accounting documents are generated with BAPI calls:

- BAPI_ACC_EMPLOYEE_EXP_POST for employee expenses
- BAPI_ACC_EMPLOYEE_PAY_POST for employee payments
- BAPI_ACC_EMPLOYEE_REC_POST for employee debts

Finally, each of the BAPIs call function modules AC_DOCUMENT_CREATE and AC_DOCUMENT_POST as a low-level accounting interface utility. Thus, you can employ any user exit (BAdI or BTE) appearing in the AC_DOCUMENT_CREATE function module (see Chapter 3, Posting to Accounting), including substitutions and validations.

At the call point of a user exit during the document generation, you can distinguish SAP standard HR Payroll postings from any others by the contents of the field BKPF-GLVOR:

- HRP1 for employee expenses
- HRP3 for employee payments (Account Payables)
- HRP2 for employee debts (Accounts Receivable)

5.2.3 Postings via IDoc

The SAP system delivers dozens of IDoc types to be used for posting different flavors of accounting documents: direct posting to a general ledger account, posting of incoming vendor invoice, and so on. You can find corresponding IDoc types in Transaction WE30 (Executing the Search Help with Mask ACC*). However, if you look into the processing function modules, you'll notice that they aren't equipped with user exits. If you thoroughly trace the chain of calls, you'll see that this chain is ended at the same function modules mentioned in the previous section: AC_DOCUMENT_CREATE and AC_DOCUMENT_POST. Thus, you should rely on already-known user exits discussed in Chapter 3.

5.2.4 Electronic Bank Statement

The process of loading a bank statement file consists of two phases: importing the bank statement file in Transaction FF_5, and posting the bank statement through Transaction FEBP.

Importing the Bank Statement File

Loading program RFEBKA00, which is linked to Transaction FF_5, parses incoming bank files according to a selected format, such as Multicash or SWIFT MT940, which are widely used in bank communication. Each individual file format is parsed in an external program, although the code in report RFEBKA00 that is responsible for choosing the format parsing program is quite static; there is just a CASE statement with no configuration.

However, if you look into the source code of format SWIFT MT940 parsing routine program RFEKA400, you can discover an old-fashioned user exit, EXIT_RFEKA400_001, belonging to function module exit FEB00004. The enhancement can be used for preprocessing raw file data, which is passed to the user exit in the form of a table parameter with a length of 512 unstructured lines. Listing 5.3 shows the interface of the user exit.

```
FUNCTION EXIT_RFEKA400_001.
*"----------------------------------------------------------------------
*"*"Lokale Schnittstelle:
*"       TABLES
*"              T_RAW_DATA STRUCTURE  RAW_DATA
*"       EXCEPTIONS
*"              ERROR_OCCURED
*"----------------------------------------------------------------------


INCLUDE ZXF01U06 .


ENDFUNCTION.
```

Listing 5.3 EXIT_RFEKA400_001 Interface

You can also see that `EXIT_RFEKA400_001` has one exception, which signals the host program to stop processing the file any further.

Report RFEBKA00 gathers parsed data into the following bank statement database tables:

- FEBKO (electronic bank statement header records)
- FEBEP (electronic bank statement line items)
- FEBRE (reference record for electronic bank statement line item)

Posting the Bank Statement

When you link report RFEBKA30 to Transaction FEBP, it interprets data in bank statement tables and makes an accounting posting. A bank statement is a list of operations of what the bank did with your money on your behalf, such as company payments to vendors, bank charges for its services, interest payments, payments from your customers, and so on. All of these operations should be correctly reflected in the company's financial accounting to make sure that the money flow is consistent and correct.

At the same time, the bank's statement can use different identification for the same objects presented in your system; also, it's possible that some valuable data in the context of your SAP ERP system may be omitted in the statement for one reason or another. During the interpretation phase, report RFEBKA30 is trying to fill these gaps automatically, for example, to determine the business partner number for the bank transaction or even more important to determine the clearing reference (e.g., payment against invoice) document numbers.

Report RFEBKA30 actually is only a wrapper for another report, RFEBBU10, which performs the interpretation. The algorithm runs through header-item relation of two tables, FEBKO and FEBEP. For each FEBEP internal loop run, the report calls different user exits that can help discover missing statement data.

Now let's walk through the available BTEs you can employ during the processing of a bank statement.

BTE 00002810 and Process 00002820

First, the system calls BTE `00002810` (you can see its interface in Listing 5.4). The event has a pair of parameters for the header record and for the line item of the bank statement that is being processed. The parameter with suffix `EXT` contains fields with external data (records that were sent by the bank), whereas suffix INT signifies that this data is internal. As a result of its run, each function module that is subscribed to the 00002810 event must return a registration flag in one of two export parameters: E_REGISTER_AREA_1 or E_REGISTER_AREA_2.

```
*"----------------------------------------------------------------------
*"*"Lokale Schnittstelle:
*"  IMPORTING
*"     VALUE(I_FEBKO_EXT) LIKE  FEBKOXT_BF STRUCTURE  FEBKOXT_BF
*"     VALUE(I_FEBEP_EXT) LIKE  FEBEPXT_BF STRUCTURE  FEBEPXT_BF
*"     VALUE(I_FEBKO_INT) LIKE  FEBKOIN_BF STRUCTURE  FEBKOIN_BF
*"     VALUE(I_FEBEP_INT) LIKE  FEBEPIN_BF STRUCTURE  FEBEPIN_BF
*"     VALUE(I_TESTRUN) TYPE  XFLAG OPTIONAL
*"  EXPORTING
*"     VALUE(E_REGISTER_AREA_1) LIKE  BOOLE-BOOLE
*"     VALUE(E_REGISTER_AREA_2) LIKE  BOOLE-BOOLE
*"     VALUE(E_SUPPR_STD_AREA_1) LIKE  BOOLE-BOOLE
*"     VALUE(E_SUPPR_STD_AREA_2) LIKE  BOOLE-BOOLE
*"  TABLES
*"      T_FEBRE STRUCTURE  FEBRE_BF
*"      T_FEBCL STRUCTURE  FEBCL_BF
*"----------------------------------------------------------------------
```

Listing 5.4 The Interface of BTE 00002810

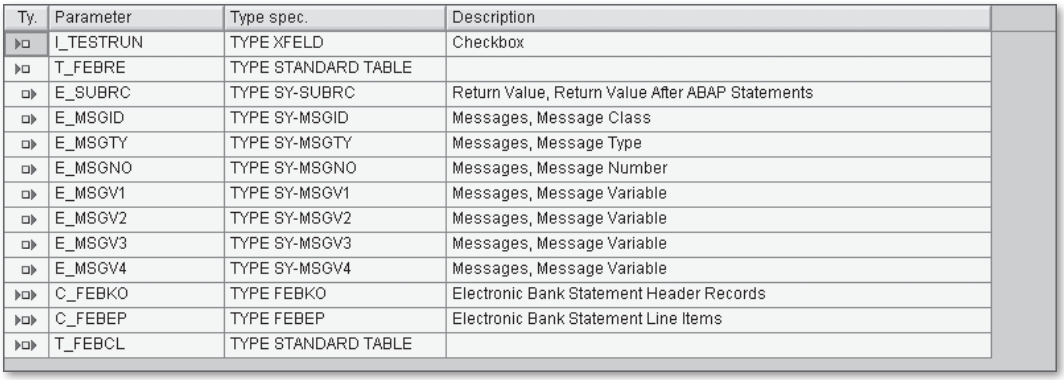
Note that subscribers to event 00002810 are called from within function FEB_OPEN_FI_CALL_1. This function allows only one application ID to be registered for each of the two areas. The application ID in the BTE framework is used to distinguish SAP internal and partner application areas. Customer-defined P&S modules and processes can have blank application IDs. Therefore, you should make sure that for this particular line item of the bank statement, your function is the only one registered, or an error will be reported. Another pair of event flag parameters, E_SUPPR_STD_AREA_1 and E_SUPPR_STD_AREA_1, will prevent execution of interpretation algorithm if they are assigned X.

Process 00002820 is called just after the event and only for registered application IDs. You can see the process interface in Listing 5.5. Note that there are export parameters to allow changing values in bank statement headers and items. Note that your changed data will be taken into account only if you assign X to the export parameter E_UPDATE_FEB.

```
*"----------------------------------------------------------------------
*"*"Lokale Schnittstelle:
*"  IMPORTING
*"     VALUE(I_FEBKO_EXT) LIKE  FEBKOXT_BF STRUCTURE  FEBKOXT_BF
*"     VALUE(I_FEBEP_EXT) LIKE  FEBEPXT_BF STRUCTURE  FEBEPXT_BF
*"     VALUE(I_FEBKO_INT) LIKE  FEBKOIN_BF STRUCTURE  FEBKOIN_BF
*"     VALUE(I_FEBEP_INT) LIKE  FEBEPIN_BF STRUCTURE  FEBEPIN_BF
*"     VALUE(I_TESTRUN) TYPE  XFLAG OPTIONAL
*"  EXPORTING
*"     VALUE(E_FEBKO_EXT) LIKE  FEBKOXT_BF STRUCTURE  FEBKOXT_BF
*"     VALUE(E_FEBEP_EXT) LIKE  FEBEPXT_BF STRUCTURE  FEBEPXT_BF
*"     VALUE(E_FEBKO_INT) LIKE  FEBKOIN_BF STRUCTURE  FEBKOIN_BF
*"     VALUE(E_FEBEP_INT) LIKE  FEBEPIN_BF STRUCTURE  FEBEPIN_BF
*"     VALUE(E_UPDATE_FEB) LIKE  BOOLE-BOOLE
*"  TABLES
*"      T_FEBRE STRUCTURE  FEBRE_BF
*"      T_FEBCL STRUCTURE  FEBCL_BF
*"----------------------------------------------------------------------
```

Listing 5.5 The Interface of BTE Process 00002820

Besides header and item data, you can also fill in clearing data in table parameter T_FEBCL.



Ty.	Parameter	Type spec.	Description
▶□	I_TESTRUN	TYPE XFELD	Checkbox
▶□	T_FEBRE	TYPE STANDARD TABLE	
□▶	E_SUBRC	TYPE SY-SUBRC	Return Value, Return Value After ABAP Statements
□▶	E_MSGID	TYPE SY-MSGID	Messages, Message Class
□▶	E_MSGTY	TYPE SY-MSGTY	Messages, Message Type
□▶	E_MSGNO	TYPE SY-MSGNO	Messages, Message Number
□▶	E_MSGV1	TYPE SY-MSGV1	Messages, Message Variable
□▶	E_MSGV2	TYPE SY-MSGV2	Messages, Message Variable
□▶	E_MSGV3	TYPE SY-MSGV3	Messages, Message Variable
□▶	E_MSGV4	TYPE SY-MSGV4	Messages, Message Variable
▶□▶	C_FEBKO	TYPE FEBKO	Electronic Bank Statement Header Records
▶□▶	C_FEBEP	TYPE FEBEP	Electronic Bank Statement Line Items
▶□▶	T_FEBCL	TYPE STANDARD TABLE	

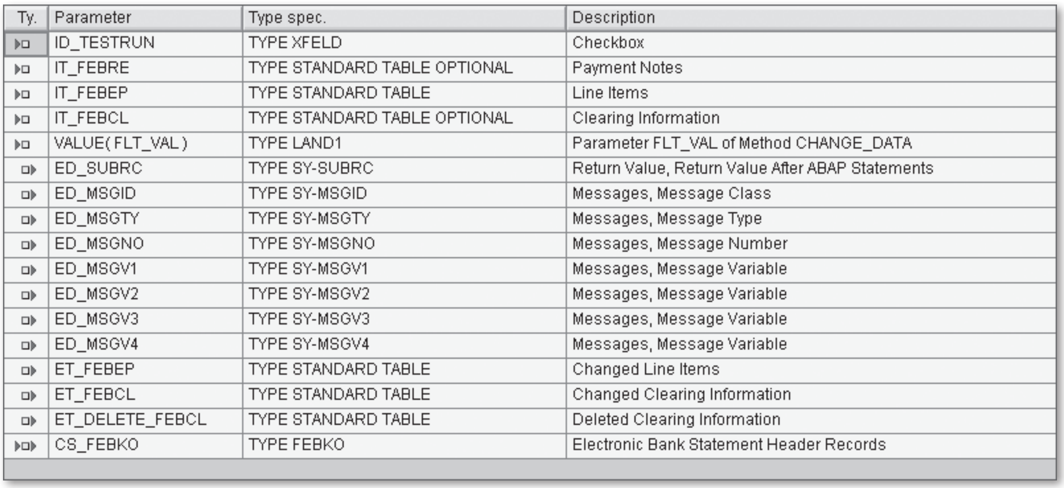
Figure 5.12 The Signature of Method CHANGE_DATA of BAdI FIEB_CHANGE_BS_DATA

BAdI Definitions

Progressing to business transaction events and processes, the system calls BAdI definition FIEB_CHANGE_BS_DATA and method CHANGE_DATA. Figure 5.12 shows the interface (or signature) of the method. Notice that the method has three changing parameters: C_FEBKO and C_FEBEP for the header and item of the bank statement, and table parameter T_FEBCL for clearing data from the statement.

The method can also return error code and error message attributes to be reported in the log and prevent the statement from being processed further.

Another BAdI definition, FIEB_CHANGE_STATEMNT, is called after all of the interpretation is executed, and the system has done everything it can. You can see the interface of the BAdI method CHANGE_DATA in Figure 5.13.



Ty.	Parameter	Type spec.	Description
▶□	ID_TESTRUN	TYPE XFELD	Checkbox
▶□	IT_FEBRE	TYPE STANDARD TABLE OPTIONAL	Payment Notes
▶□	IT_FEBEP	TYPE STANDARD TABLE	Line Items
▶□	IT_FEBCL	TYPE STANDARD TABLE OPTIONAL	Clearing Information
▶□	VALUE(FLT_VAL)	TYPE LAND1	Parameter FLT_VAL of Method CHANGE_DATA
□▶	ED_SUBRC	TYPE SY-SUBRC	Return Value, Return Value After ABAP Statements
□▶	ED_MSGID	TYPE SY-MSGID	Messages, Message Class
□▶	ED_MSGTY	TYPE SY-MSGTY	Messages, Message Type
□▶	ED_MSGNO	TYPE SY-MSGNO	Messages, Message Number
□▶	ED_MSGV1	TYPE SY-MSGV1	Messages, Message Variable
□▶	ED_MSGV2	TYPE SY-MSGV2	Messages, Message Variable
□▶	ED_MSGV3	TYPE SY-MSGV3	Messages, Message Variable
□▶	ED_MSGV4	TYPE SY-MSGV4	Messages, Message Variable
□▶	ET_FEBEP	TYPE STANDARD TABLE	Changed Line Items
□▶	ET_FEBCL	TYPE STANDARD TABLE	Changed Clearing Information
□▶	ET_DELETE_FEBCL	TYPE STANDARD TABLE	Deleted Clearing Information
▶□▶	CS_FEBKO	TYPE FEBKO	Electronic Bank Statement Header Records

Figure 5.13 The Signature of Method CHANGE_DATA of the BAdI FIEB_CHANGE_STATEMNT

Customer-Defined Interpretation Algorithm

After calling BTEs and the first BAdI, the system runs the interpretation proper. Each bank statement item can have its own interpretation algorithm, which is defined by the field FEBEP-INTAG value. Therefore, the individual item algorithm can be set during a user exit run: either BTE or BAdI.

A full list of interpretation algorithm numbers and descriptions can be found in the INTAG_EB domain fixed values. INTAG_EB is numeric 3. It is assumed that all SAP system algorithms belong to the range of INTAG values from 000 to 899, and everything above 900 is a customer-defined interpretation.

To implement a customer-defined interpretation, you have to create a function module with a predefined name structure—Z_FIEB_NNN_ALGORITHM—where NNN is the algorithm number.

This function module must have the interface shown in Listing 5.6.

```
FUNCTION Z_FIEB_901_ALGORITHM.
*"----------------------------------------------------------------------
*"*"Local Interface:
*"  IMPORTING
*"     REFERENCE(I_NOTE_TO_PAYEE) TYPE  STRING
*"     VALUE(I_COUNTRY) TYPE  T001-LAND1
*"  TABLES
*"      T_AVIP_IN STRUCTURE  AVIP
*"      T_AVIP_OUT STRUCTURE  AVIP
*"      T_FILTER1
*"      T_FILTER2
*"----------------------------------------------------------------------

ENDFUNCTION.
```

Listing 5.6 Sample Interpretation Algorithm Function

Based on the payment note passed to the function in parameter I_NOTE_TO_PAYEE and document references in T_AVIP_IN, the algorithm is expected to produce reasonable results in table structure T_AVIP_OUT, which has the structure of the payment advice line item. Table structure T_AVIP_OUT is then used to update the clearing reference data for the statement item.

Function Module Exit

After the interpretation algorithm and just before the second BAdI call, the system invokes a component (function module) EXIT_RFEBBU10_001 of the old-fashioned function module exit FEB00001. Its interface is shown in Listing 5.7.

```
FUNCTION EXIT_RFEBBU10_001.
*"----------------------------------------------------------------------
*"*"Lokale Schnittstelle:
*"  IMPORTING
*"     VALUE(I_FEBEP) LIKE  FEBEP STRUCTURE  FEBEP
*"     VALUE(I_FEBKO) LIKE  FEBKO STRUCTURE  FEBKO
*"     VALUE(I_TESTRUN) TYPE  XFLAG
*"  EXPORTING
*"     VALUE(E_FEBEP) LIKE  FEBEP STRUCTURE  FEBEP
*"     VALUE(E_FEBKO) LIKE  FEBKO STRUCTURE  FEBKO
*"     VALUE(E_MSGTEXT) LIKE  FEBMKA-MESSG
*"     VALUE(E_MSGTYP) LIKE  FEBMKA-MSTYP
*"     VALUE(E_UPDATE) LIKE  FEBMKA-MSTYP
*"  TABLES
*"      T_FEBCL STRUCTURE  FEBCL
*"      T_FEBRE STRUCTURE  FEBRE
*"----------------------------------------------------------------------

  INCLUDE ZXF01U01.

ENDFUNCTION.
```

Listing 5.7 The Interface of EXIT_RFEBBU10_001

This is another point where you can intercept the standard flow of the bank statement processing.

5.3 Summary

In this chapter, we discussed several inbound interfaces of Financial Accounting, which cover some of the general corporate activities. Thanks to the SAP design in all of these scenarios, you can find ways to seamlessly tailor the standard process for specific corporate needs.

In the next chapter, you'll see what user-exit techniques are available for development in outbound scenarios when the system sends accounting data to external systems.

To preserve symmetry, there must be outbound interfaces in Financial Accounting: Communication with the outside world cannot be just one-way. This chapter covers the methods of communicating financial data to external systems.

6 Outbound Scenarios in Financial Accounting

In this chapter, we'll consider several important scenarios during which data from Financial Accounting (FI) are transferred to external systems, including distributing master data, sending dunning notifications, and using the payment program. Master data distribution occurs when implementing an external payroll system; dunning procedures are common in companies selling their products or services; and, finally, no company can do without a bank account, and the communication with banks is fulfilled using the payment program.

6.1 Master Data Distribution

In this section, we'll discuss some techniques for extracting (or sending) accounting master data to external systems. An example of such a scenario is extracting legacy data from an old SAP system when a company is upgrading. Various scenarios also involve centralized SAP NetWeaver Master Data Management (SAP NetWeaver MDM), when a master data change is initiated in the accounting system and must be distributed over the entire system landscape.

Let's start by discussing two techniques of accounting master data distribution: generating files for batch input and using ALE/IDoc technology.

6.1.1 Batch Input

The programs mentioned in Chapter 5, Inbound Scenarios in Financial Accounting, for loading FI master data with a batch-input technique have counterpart programs that export data in a batch-input format. Table 6.1 shows these three reports, which can be used to copy data between two company codes or SAP systems.

Report Name	Description
RFBIDE10	Transfer Customer Master Data from Source Company Code: Send
RFBIKR10	Transfer Vendor Master Data from Source Company Code: Send
RFBISA10	Copy General Ledger Account Master Data: Send

Table 6.1 Reports for Sending Master Data to an External System in Batch-Input Format

These reports are listed only for your information; they do not include any enhancements or user exits. However, the logic of the reports is clear and simple, so it's not a big problem for an average ABAP developer to make a copy of any of the reports and tailor that copy to specific corporate needs.

6.1.2 ALE/IDoc tools

Another set of reports generates master data IDocs for higher-level communication with external systems. For all three main kinds of FI master data—general ledger accounts, customers, vendors—the ALE/IDoc interface is built using a very similar approach. One report is for creating and sending an IDoc of a particular type, and one report is for requesting an IDoc from an external system. The requesting report actually sends a special type of IDoc of basic type `ALEREQ01`, which contains a high-level application object identification and a set of criteria similar to an ABAP range construct.

Next, we'll see what tools can be used to distribute accounting master data (general ledger accounts, customers, and vendors) and what user exits are available there.

General Ledger Account Master Record

Sending Transaction BD18 is linked to report RBDSEGLM with this simple logic: The report selects well-known general ledger account tables—SKA1, SKB1, SKAT—and generates an IDoc. The report can only send one of two predefined logical messages: `GLCORE` and `GLMAST`. By default, the `GLCORE` message is mapped to the `GLCORE01` IDoc type, and the `GLMAST` message is mapped to the `GLMAST01` IDoc.

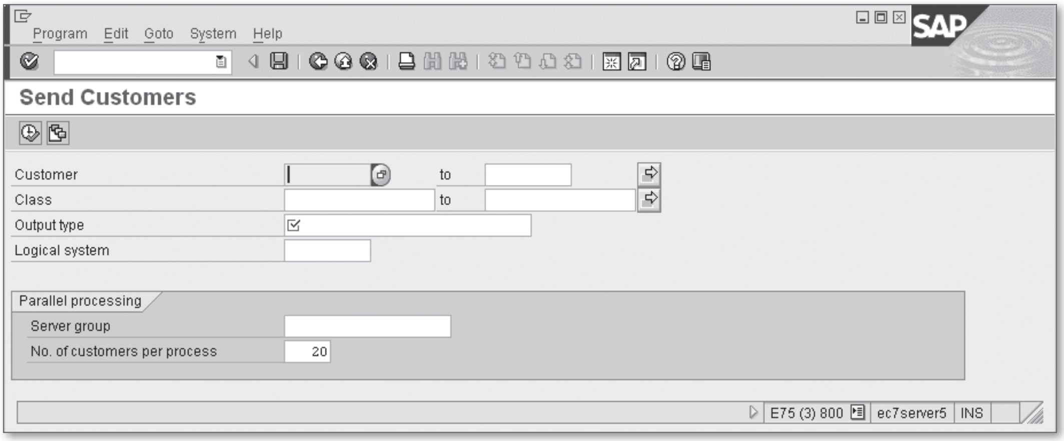
These IDoc types differ in data volume to be sent. `GLCORE01` includes only very basic general ledger account data, whereas `GLMAST` contains much more data.

The GLCORE IDoc is generated by the MASTERIDOC_CREATE_GLCORE function module, and the GLMAST IDoc is generated by the MASTERIDOC_CREATE_GLMAST function.

Only function MASTERIDOC_CREATE_GLCORE has predefined enhancement capabilities: You can use the source code enhancement point of the spot ES_SAPLKS03. The enhancement point is situated exactly before sending the generated IDoc to the ALE runtime system via the MASTER_IDOC_DISTRIBUTE function module. The MASTERIDOC_CREATE_GLMAST function module has several source code enhancement points of the spot ES_SAPLKS03_1; however, it's marked as SAP internal, so you can't legally implement it.

Sending Customers

Customer master records can be sent to an external system by Transaction BD12, which is linked to report RBDSEDEB. Figure 6.1 shows the selection screen of the report.



Send Customers

Customer		to	
Class		to	
Output type	✓		
Logical system			

Parallel processing

Server group	
No. of customers per process	20

E75 (3) 800 | ec7server5 | INS

Figure 6.1 Selection Screen of Report RBDSEDEB

By default, the report generates and sends one of two IDoc messages: DEBMAS and DEBCOR. However, unlike the general ledger account master sending program, report RBDSEDEB can be enhanced using a source code plug-in technique via enhancement spot ES_RBDSEDEB. The spot has three enhancement options:

- Dynamic enhancement point RBDSEDEB_01 at the event AT SELECTION-SCREEN ON VALUE-REQUEST FOR MESTYP allows you to add your own message types to the selection list.

- Dynamic enhancement section RBDSEDEB_02 at the report event AT SELECTION-SCREEN ON MESTYP allows you to check the entered message type.
- Dynamic enhancement section RBDSEDEB_03 contains a call to the ALE runtime system for sending the generated IDoc. Using the enhancement section, you can completely redefine the logic of sending the IDoc to an external system.

Note

If you plan to use the enhancement spot ES_RBDSEDEB, you should become familiar with the whole source code of report RBDSEDEB and thoroughly evaluate the possible impact of your coding on other applications.

Message DEBCOR is used to send only a customer's most basic data: number, name, and address. Standard SAP function module MASTERIDOC_CREATE_DEBCOR, which generates message DEBCOR, has only one enhancement point, MASTERIDOC_CREATE_DEBCOR_G2, of the spot ES_SAPLVV01, which is located before the actual sending of the created IDoc to the ALE runtime system.

The program logic of sending customer master records into an external system with message DEBMAS has another level of enhancement options, which is located in function module MASTERIDOC_CREATE_DEBMAS. This function module employs source code plug-ins, BAdI definitions, and customer enhancements.

Let's walk through the available user exits: BAdI method calls, function module exits, and enhancement spots.

BAdI Definition CUSTOMER_ADD_DATA_BI

To use this BAdI inside the MASTERIDOC_CREATE_DEBMAS function module, you have to implement its interface method, FILL_ALE_SEGMENTS_OWN_DATA. The method is called after each IDoc segment has filled with data.

Function Module Exit VSV00001

Function modules EXIT_SAPLVV01_001 in the form of CALL CUSTOMER-FUNCTION is called just like a BAdI interface method after forming each segment data.

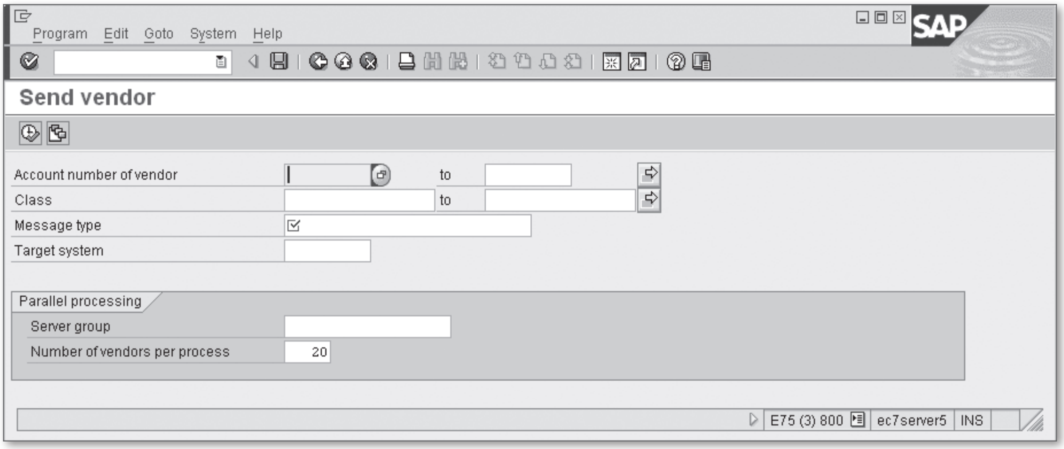
Enhancement Spot ES_SAPLVV01

The spot includes several source code plug-ins inside the MASTERIDOC_CREATE_DEBMAS function module and VV01 function group:

- Using static enhancement points `LVV01TOP_01` and `LVV01TOP_02`, you can declare your own global data for the function group `VV01`.
- Dynamic enhancement points `MASTERIDOC_CREATE_DEBMAS_06` and `MASTERIDOC_CREATE_DEBMAS_G2` allow you to implement last-minute additions to the generated IDoc just before it's sent to the ALE runtime.
- By means of *dynamic enhancement point* `MASTERIDOC_CREATE_DEBMAS_04`, you can add some logic before processing customer data; for example, you can redefine IDoc control parameters.
- Other available dynamic enhancement points include `MASTERIDOC_CREATE_DEBMAS_01`, `MASTERIDOC_CREATE_DEBMAS_02`, `MASTERIDOC_CREATE_DEBMAS_03`, and `MASTERIDOC_CREATE_DEBMAS_05`, which can be used to implement additional logic for adding your application-specific segments to the IDoc. For example, in the IDES system, Industry Solution IS-Oil is installed, and its specific logic is implemented here using enhancement spot `ES_SAPLVV01`.

Sending Vendors

The process of sending vendor master data looks very similar to that of sending customer master data. Transaction BD14 is linked to report RBDSECRE, which has almost the same look and feel as report RBDSEDEB. The selection screens of both reports look almost identical, as you can see by looking at Figure 6.2.



Send vendor

Account number of vendor		to	
Class		to	
Message type			
Target system			

Parallel processing

Server group	
Number of vendors per process	20

E75 (3) 800 | ec7server5 | INS

Figure 6.2 Selection Screen of the Report RBDSECRE

However, in the source code of report RBDSECRE, you can see obvious differences from that of RBDSEDEB: The vendor sending report completely lacks any enhancements. For example, you can't expand the list of available messages without modification. The report by default can generate two main logical messages, `CRECOR` and `CREMAS`, as well as their reduced versions.

The report RBDSECRE calls function module `MASTERIDOC_CREATE_CRECOR` for generating message `CRECOR`, and `MASTERIDOC_CREATE_CREMAS` for the message `CREMAS`.

Message `CRECOR` is linked by default to the IDoc basic type `CRECOR01` and includes only basic vendor data, such as vendor number, name, and address. Only the `MASTERIDOC_CREATE_CRECOR_G2` enhancement point is available in the `MASTERIDOC_CREATE_CRECOR` function module: It is located before the call of `MASTER_IDOC_DISTRIBUTE`. At the moment of the call, all of the IDoc data are already prepared. As you might expect, the `MASTERIDOC_CREATE_CREMAS` function module has more enhancement options available.

BAdI Definition VENDOR_ADD_DATA_BI

As with the customer sending report, you have to implement the interface method `FILL_ALE_SEGMENTS_OWN_DATA` to use the BAdI inside the `MASTERIDOC_CREATE_CREMAS` function module. This method is called after each IDoc segment has filled with data.

Function Module Exit VSV00001

Function module `EXIT_SAPLKD01_001` is called just like the BAdI interface method after forming each data segment.

Enhancement Spot ES_SAPLKD01

The enhancement spot contains the following points:

- Static enhancement point `LKD01TOP_01` can be used to declare global variables.
- Dynamic enhancement point `MASTERIDOC_CREATE_CREMAS_G2` can be used to implement additional logic to be executed just before sending the prepared IDoc via the ALE runtime.

- Dynamic enhancement points MASTERIDOC_CREATE_CREMAS_01, EHP_MASTERIDOC-CREATE_CREMAS_01, and EHP_MASTERIDOCCREATE_CREMAS_02 can be used to add your own segment to the IDoc.

6.2 Dunning

Dunning is the process of notifying business partners of overdue payments. The simplest way to use a dunning procedure is to print out previously configured dunning letters and send them to owing customers or vendors.

The process of dunning in SAP uses sophisticated configuration tools to make the procedure completely automatic (well, almost). However, as always, not every instance or unique specification can be foreseen. To make room for custom-defined additions to the standard dunning procedure, SAP offers a number of user exits that we'll discuss in the following subsections.

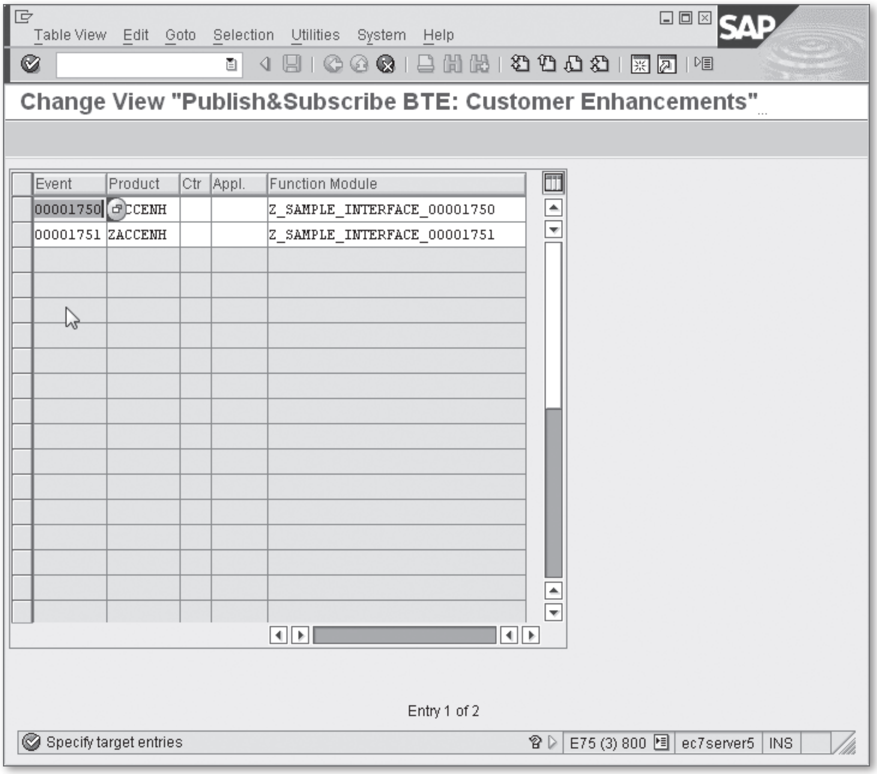
6.2.1 BTEs in Transaction F150

Transaction F150 (Dunning) is an entry point to the main dunning procedure activities: configuring the dunning activity, printing individual dunning notices, or scheduling an automatic dunning run. The transaction code contains some enhancement capabilities in the form of BTE calls. Using these BTEs, you can enhance the user interface of the transaction. The GUI status of the transaction has a function code OPFI, which by default is hidden. BTE 00001750 defines the text of the function code, and BTE 00001751 implements your specific processing of function code OPFI. Note that the additional command is available only on the PARAMETER tab of Transaction F150.

Secret Function Codes

Transaction F150 contains secret function codes that allow enabling and disabling of Open FI events processing. The "secret" means that these function codes aren't available in the transaction toolbar or menu; you can only execute them by directly entering the code into the GUI window command field. Code OFI shows the status of Open FI events for dunning; OFI+ enables Open FI processing; and OFI- disables Open FI.

For the sake of demonstration, we implemented both events in the IDES system. The configuration of BTEs is shown in Figure 6.3.



Change View "Publish&Subscribe BTE: Customer Enhancements"

Event	Product	Ctr	Appl.	Function Module
00001750	ZACCENH			Z_SAMPLE_INTERFACE_00001750
00001751	ZACCENH			Z_SAMPLE_INTERFACE_00001751

Entry 1 of 2

Specify target entries | E75 (3) 800 | ec7server5 | INS

Figure 6.3 BTE Configuration for Transaction F150

Listing 6.1 shows a sample implementation of event 00001750. We assign a predefined text to the export parameter E_FTEXT. Note, however, a real-life implementation should take into account the language key, which is supplied as import parameter I_SPRAS.

```
FUNCTION Z_SAMPLE_INTERFACE_00001750.
*"-----------------------------------------------------------------
*"*"Local Interface:
*"  IMPORTING
*"     VALUE(I_SPRAS) LIKE  SY-LANGU
*"  EXPORTING
*"     VALUE(E_FTEXT) LIKE  FTEXTS-FTEXT
*"----------------------------------------------------------------------

  E_FTEXT = 'Dunning enhancement'(001).
ENDFUNCTION.
```

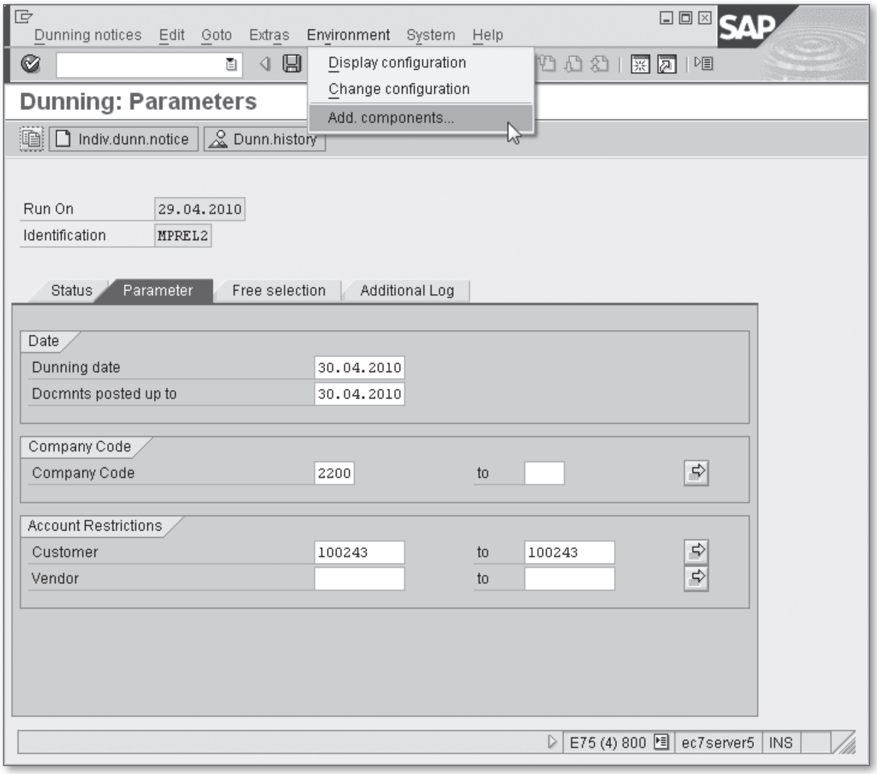
Listing 6.1 Sample Implementation of BTE 00001750

Listing 6.2 shows the test implementation of event 00001751. The combination of I_LAUFD and I_LAUFI import parameters is the unique key of the dunning run. The one-character import parameter I_AKTYP shows the mode of the running transaction. It can take as a value either V for editing mode or A for display mode, depending on the status of the dunning run. If I_AKTYP = A, you can show your additional parameters for the dunning run without modification.

```
FUNCTION z_sample_interface_00001751.
*"----------------------------------------------------------------------
*"*"Local Interface:
*"  IMPORTING
*"     VALUE(I_AKTYP) LIKE  OFIWA-AKTYP OPTIONAL
*"     VALUE(I_LAUFD) LIKE  MAHNV-LAUFD
*"     VALUE(I_LAUFI) LIKE  MAHNV-LAUFI
*"----------------------------------------------------------------------
  MESSAGE 'Sample 00001751 implementation called'(002) TYPE 'I'.
ENDFUNCTION.
```

Listing 6.2 Sample Implementation of BTE 00001751

There can be more than one implementation of both 00001750 and 00001751 events. For example, in the IDES system, there is an SAP TR-LO (loan management) component installed, which implements an addition to the dunning functionality. Therefore, the system supplies a generic text for the OPFI function code (see the menu selected in Figure 6.4).



Dunning: Parameters

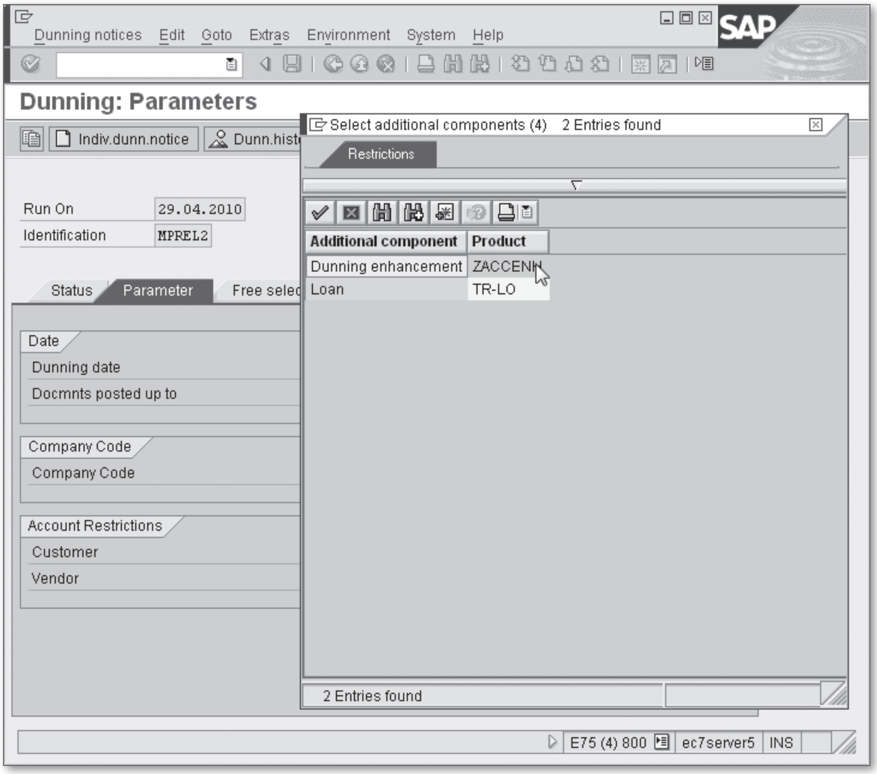
Run On	29.04.2010		
Identification	MPREL2		
Dunning date	30.04.2010		
Docmnts posted up to	30.04.2010		
Company Code	2200	to	
Customer	100243	to	100243
Vendor		to	

Figure 6.4 Additional Menu Command in Dunning Parameter Configuration Screen

When selecting the menu command, the system displays a dialog box that instructs you to choose a particular component (see Figure 6.5).

You can use these BTEs to supply additional parameters for a dunning run. Remember that a dunning run is identified by the date (RUN ON) and an additional arbitrary identification code (IDENTIFICATION). These fields are mandatory, so both fields must be filled in by the time you execute the BTE implementation. The storage for your additional parameters is completely your responsibility.

You can use previously saved additional data during the dunning run itself by using another set of user exits, which we'll discuss in the next section.



Run On	29.04.2010
Identification	MPREL2

Dunning date
Docmnts posted up to
Company Code
Customer
Vendor

Select additional components (4) 2 Entries found

Additional component	Product
Dunning enhancement	ZACCENH
Loan	TR-LO

2 Entries found

Figure 6.5 Additional Components of the Dunning Parameters Configuration

6.2.2 BTEs during the Dunning Run

A dunning run consists of two phases: dunning data selection and printout. Dunning data selection is performed by SAP report SAPF150S2; while the printout phase is executed by SAPF150D2. The user can select to perform each phase independently or both in one run.

Data Selection Phase

The data selection phase processes vendor and/or customer open items and stores dunning data in several tables. As a result, the report SAPF150S2 generates a number of records in Tables MHNK and MHND by means of function module `GENERATE_DUNNING_DATA`.

During the run of report SAPF150S2, you can use several BTEs.

Event 00001703

This event should actually be called a process because it allows you to change data. Listing 6.3 shows the interface of the event. Despite the fact that table parameters are declared as untyped, they all actually have a structure of a `RANGE` (or `SELECT-OPTION`).

```
FUNCTION SAMPLE_INTERFACE_00001703.
*“----------------------------------------------------------------------
*“*“Lokale Schnittstelle:
*“  TABLES
*“      T_SEL_CC
*“      T_SEL_CUST
*“      T_SEL_VEN
*“      T_LOG_CUST
*“      T_LOG_VEND
*“      T_SEL_FILTER
*“----------------------------------------------------------------------
ENDFUNCTION.
```

Listing 6.3 The Interface of BTE 00001703

The elements in the listing are defined as follows:

- `T_SEL_CC`
  A range for the company code.
- `T_SEL_CUST`
  A range of customer numbers to be processed.
- `T_SEL_VEN`
  A range of vendor numbers to be processed.
- `T_LOG_CUST`
  A range of customer numbers to be processed with trace.
- `T_LOG_VEND`
  A range of vendor numbers to be processed with trace.
- `T_SEL_FILTER`
  A free selection list of field values. The list corresponds to the `IFLDTAB` dictionary structure. Each record contains a full field name (`FLDNA`). An example field name

can be KNA1-STCD1. Fields FLDL1 and FLDL2 contain a comma-delimited list of possible values. There is also the flag IGNOR, which works as an exclusion mark for matching table entries, and the flag UPPCT for case-insensitive values.

Inside the event implementation, you can amend the parameters and thus change the selection criteria for customer or vendor open items to be dunned.

Process 00001053—DUNNING: Set a One-Time Account

This process is called inside the GENERATE_DUNNING_DATA function module when the function processes a one-time account item. The process allows you to generate your own one-time account group key.

Process 00001060—DUNNING: Dunning Check MHND

This process allows the system to decide if a particular item should or should not be dunned. As seen in Listing 6.4, the process has one import parameter of structure MHND (dunning data) and three flags to return, which controls further processing of the item. Additionally, the process can return messages to be shown in the resulting log via table parameter T_FIMSG.

```
FUNCTION SAMPLE_PROCESS_00001060.
*"----------------------------------------------------------------------
*"*"Lokale Schnittstelle:
*"       IMPORTING
*"             VALUE(I_MHND) LIKE  MHND STRUCTURE  MHND
*"       TABLES
*"              T_FIMSG STRUCTURE  FIMSG
*"       CHANGING
*"             VALUE(C_XFAEL) LIKE  MHND-XFAEL
*"             VALUE(C_XZALB) LIKE  MHND-XZALB
*"             VALUE(C_MANSP) LIKE  MHND-MANSP
*"----------------------------------------------------------------------
ENDFUNCTION.
```

Listing 6.4 The Interface of Process 00001060

Event 00001762—Dunning

This event should be a process because it has changing parameters. This event can be used to fill additional fields in a dunning item (MHND_EXT). Listing 6.5 shows the interface of the event.

```
FUNCTION SAMPLE_INTERFACE_00001762.
*"----------------------------------------------------------------------
*"*"Lokale Schnittstelle:
*"  CHANGING
*"     REFERENCE(CS_MHND_EXT) LIKE  MHND_EXT STRUCTURE  MHND_EXT
*"----------------------------------------------------------------------

ENDFUNCTION.
```

Listing 6.5 The Interface of BTE 00001762

Note

Besides its activation in the general BTE configuration, event 00001762 should also be activated in the runtime by calling function module SET_EXIT_ACTIVE of function group F150.

Process 00001061—DUNNING: Delete Indicator MHND

Using this process, you can completely remove a dunning item from the processing list. The process has the import parameter of the structure MHND and a changing flag C_DEL_DU.

Note

When implementing the logic of process 00001061, you should take into account the imported value of the parameter C_DEL_DU. If it's already marked with an X, you might not need any further processing.

Event 00001763—Dunning

This event is called before Phase III of data selection. At this stage, all of the dunning run data have already been prepared, and the system is ready to calculate the minimal dunning amount and interest charges.

Note

Besides its activation in general BTE configuration, event 00001763 should also be activated in runtime by calling function module SET_EXIT_ACTIVE of function group F150.

This event has the full pack of prepared dunning data as table parameters:

- `CT_MHNK`
  Dunning account entry.
- `CT_MHND_EXT`
  Dunning data items.

Process 00001068—DUNNING: Activate Group Interest Calculation
This process can be used to enable group interest calculation. Import parameters of the process are the following:

- Dunning procedure code (field `MHNK-MAHNA`)
- Application code (field `MHNK-APPLK`)

Note that field `MHNK-APPLK` should be the field at the moment of other BTEs or process calls. The process returns the group interest calculation flag in export parameter `E_GROUP_INTEREST`.

Process 00001074—DUNNING: Carry Out Group Interest calculation
If process `00001068` previously activated group interest, then process `00001074` is supposed to perform this particular calculation.

In the IDES system, there is only one SAP function module, `FI_PSO_PROCESS_00001074` that is subscribed to this process. The function belongs to the SAP Public Sector Industry Solution.

Process 00001076—DUNNING: Interest Calculation for PA
Process `00001076` is called if the group interest calculation is disabled and when the system has calculated interest in its standard way. Another prerequisite for this process call is switch `FM_CI_CORE_SFWS_2`: The process is called if the switch is turned on.

As part of the SAP Public Sector Industry Solution in the IDES system, SAP function module `FI_PSO_PROCESS_00001076` is subscribed to this process.

Process 00001050—DUNNING: Read Additional Fields for MHNK
The process can be used for filling additional fields in dunning account entries (Table MHNK). You can extend this table using customer include structure `CI_MHNK`.

The process also has a flag parameter `MIN_IT`, which shows that a dunning account entry being processed contains the lowest possible dunning level.

Event 00001764—Dunning: Alternative Check for Account Balance

This is the final event available in a selection process. You can use it to make the final decision whether the dunning item being processed should be dunned at all. The interface of the event is shown in Listing 6.6.

```
FUNCTION SAMPLE_INTERFACE_00001764.
*”----------------------------------------------------------------------
*”*”Lokale Schnittstelle:
*”  IMPORTING
*”     REFERENCE(I_WAERS) LIKE  MHNK-WAERS
*”  EXPORTING
*”     REFERENCE(EB_PROCESSED) LIKE  BOOLE-BOOLE
*”  TABLES
*”      T_MHND_EXT STRUCTURE  MHND_EXT
*”      T_T047B STRUCTURE  T047B
*”      T_FIMSG STRUCTURE  FIMSG
*”  CHANGING
*”     VALUE(CB_DUNN_IT) LIKE  BOOLE-BOOLE
*”     REFERENCE(CS_MHNK) LIKE  MHNK STRUCTURE  MHNK
*”----------------------------------------------------------------------
ENDFUNCTION.
```

Listing 6.6 The Interface of BTE 00001764

As a result of its run, the event implementation should set the flag CB_DUNN_IT if the items presented by table parameter T_MHND_EXT must be dunned; and also flag EB_PROCESSED to tell the system that the event has taken over the processing. If the event returned space in the parameter EB_PROCESSED, then the system runs the standard amount check algorithm.

Dunning Printout Phase

When dunning data has been gathered, checked, and calculated by report SAPF150S2, it's time to run the next phase of the dunning process: printout. The phase is performed by standard report SAPF150D2, which outputs dunning letters to a customer or a vendor on the list. SAP offers some functionality to implement output on different types of media such as hardcopy, fax, or email. However, you can use numerous BTEs available in report SAPF150D2 to thoroughly tailor the process to your specific business needs.

In the following subsections, we discuss these events in the order of their appearance during processing.

Event 00001705—DUNNING: Start of Dunning Notice Printout

Event 00001705 is called before selecting and sorting dunning data. You can use this event to amend selection criteria, which were passed on to the report from Transaction F150. The event interface is shown in Listing 6.7.

```
FUNCTION SAMPLE_INTERFACE_00001705.
*"----------------------------------------------------------------------
*"*"Lokale Schnittstelle:
*"       IMPORTING
*"             VALUE(I_LAUFD) LIKE  F150V-LAUFD
*"             VALUE(I_LAUFI) LIKE  F150V-LAUFI
*"             VALUE(I_UPDATE) LIKE  BOOLE-BOOLE
*"       TABLES
*"              T_SEL_DEBI
*"              T_SEL_KRED
*"       CHANGING
*"             VALUE(E_ITCPO) LIKE  ITCPO STRUCTURE  ITCPO
*"             VALUE(E_DIRECTION) TYPE  C
*"----------------------------------------------------------------------
ENDFUNCTION.
```

Listing 6.7 The Interface of Event 00001705

T_SEL_DEBI and T_SEL_KRED are passed on as untyped table parameters; however, they both have the structure of SELECT-OPTIONS of RANGE.

The I_UPDATE import flag shows that the report was submitted in an update mode; that is, it updates dunning data after printing. Besides possible changes in selection criteria, events T_SEL_DEBI and T_SEL_KRED can return printing parameters in export structure E_ITCPO and also the sorting direction in E_DIRECTION. The direction can either be A for ascending or D for descending.

Process 00001020—DUNNING: Following Reading, Prior to Printing

This process allows the user to change dunning data before printing. The interface includes changeable dunning header structure E_MHNK and dunning items in table structure T_MHND. See a sample interface in Listing 6.8.

```
FUNCTION SAMPLE_PROCESS_00001020.
*"----------------------------------------------------------------------
*"*"Lokale Schnittstelle:
*"       IMPORTING
*"             VALUE(I_MAHNV) LIKE  MAHNV STRUCTURE  MAHNV
*"             VALUE(I_F150V) LIKE  F150V STRUCTURE  F150V
*"       TABLES
*"              T_MHND STRUCTURE  MHND
*"       CHANGING
*"             VALUE(E_MHNK) LIKE  MHNK STRUCTURE  MHNK
*"----------------------------------------------------------------------
ENDFUNCTION.
```

Listing 6.8 BTE Process 00001020 Interface

Event 00001719—DUNNING: Additional Activities Before Printing and Event 00001720—DUNNING: Printing Dunning Notice

These events are called one after the other and have identical interfaces, as shown in Listing 6.9.

```
FUNCTION SAMPLE_INTERFACE_00001719.
*"----------------------------------------------------------------------
*"*"Lokale Schnittstelle:
*"       IMPORTING
*"             VALUE(I_MAHNV) LIKE  MAHNV STRUCTURE  MAHNV
*"             VALUE(I_F150V) LIKE  F150V STRUCTURE  F150V
*"             VALUE(I_MHNK) LIKE  MHNK STRUCTURE  MHNK
*"             VALUE(I_ITCPO) LIKE  ITCPO STRUCTURE  ITCPO
*"             VALUE(I_UPDATE) LIKE  BOOLE-BOOLE
*"             VALUE(I_MOUT) LIKE  BOOLE-BOOLE
*"             VALUE(I_OFI) LIKE  BOOLE-BOOLE
*"       TABLES
*"              T_MHND STRUCTURE  MHND
*"              T_FIMSG STRUCTURE  FIMSG
*"       CHANGING
*"             VALUE(E_COMREQ) LIKE  BOOLE-BOOLE
*"             VALUE(E_RETCODE) TYPE  C
*"----------------------------------------------------------------------
ENDFUNCTION.
```

Listing 6.9 The Interface of BTEs 00001719 and 00001720

Event implementations should not change dunning data. If any of the events return `X` in export parameter `E_COMREQ`, then the system should issue a database `COMMIT` after event execution. If there is no active implementation of event `00001720`, the system calls standard function module `FI_PRINT_DUNNING_NOTICE`, which implements the default printing functionality via SAPscript. If you need to print a dunning letter as a SAP Smart Form, then implement event `00001720` and use function module

FI_PRINT_DUNNING_NOTICE_SMARTF. For SAP Interactive Forms by Adobe, you can use function module FI_PRINT_DUNNING_NOTICE_PDF.

You can also use event 00001720 to implement dunning output on other media such as EDI, email, or even SMS (text messages).

Process 00001030—DUNNING: Determine Form

This process allows you to implement logic for choosing the printout form and is called from within module GET_DUNNING_CUSTOMIZING. The GET_DUNNING_CUSTOMIZING function is indirectly called from FI_PRINT_DUNNING_NOTICE, FI_PRINT_DUNNING_NOTICE_PDF, and FI_PRINT_DUNNING_NOTICE_SMARTF function modules. However, if you don't use the printing functions mentioned previously for a particular dunning data, process 00001030 is irrelevant. Its interface is shown in Listing 6.10.

```
FUNCTION SAMPLE_PROCESS_00001030.
*"--------------------------------------------------------------------
*"*"Lokale Schnittstelle:
*"       IMPORTING
*"             VALUE(I_MHNK) LIKE  MHNK STRUCTURE  MHNK
*"       CHANGING
*"             VALUE(C_FORNR) LIKE  T047E-FORNR
*"             VALUE(C_LISTN) LIKE  T047E-LISTN
*"             VALUE(C_XAVIS) LIKE  T047E-XAVIS
*"             VALUE(C_ZLSCH) LIKE  T047E-ZLSCH
*"--------------------------------------------------------------------
ENDFUNCTION.
```

Listing 6.10 BTE Process 00001030 Interface

Process 00001040—DUNNING: Determine Output Device

Process 00001040 is called indirectly from within function modules FI_PRINT_DUNNING_NOTICE, FI_PRINT_DUNNING_NOTICE_PDF, and FI_PRINT_DUNNING_NOTICE_SMARTF. The process function interface is shown in Listing 6.11.

```
FUNCTION SAMPLE_PROCESS_00001040.
*"--------------------------------------------------------------------
*"*"Lokale Schnittstelle:
*"       IMPORTING
*"             VALUE(I_KNA1) LIKE  KNA1 STRUCTURE  KNA1
*"             VALUE(I_KNB1) LIKE  KNB1 STRUCTURE  KNB1
*"             VALUE(I_LFA1) LIKE  LFA1 STRUCTURE  LFA1
*"             VALUE(I_LFB1) LIKE  LFB1 STRUCTURE  LFB1
*"            VALUE(I_MHNK) LIKE  MHNK STRUCTURE  MHNK
*"            VALUE(I_F150D2) LIKE  F150D2 STRUCTURE  F150D2
*"            VALUE(I_T047E) LIKE  T047E STRUCTURE  T047E
*"            VALUE(I_UPDATE) LIKE  BOOLE-BOOLE
*"       TABLES
*"             T_FIMSG STRUCTURE  FIMSG
*"       CHANGING
*"            VALUE(C_FINAA) LIKE  FINAA STRUCTURE  FINAA
*"            VALUE(C_ITCPO) LIKE  ITCPO STRUCTURE  ITCPO
*"            VALUE(C_ARCHIVE_INDEX) LIKE  TOA_DARA
*"                            STRUCTURE  TOA_DARA DEFAULT SPACE
*"            VALUE(C_ARCHIVE_PARAMS) LIKE  ARC_PARAMS
*"                            STRUCTURE  ARC_PARAMS DEFAULT SPACE
*"----------------------------------------------------------------------
ENDFUNCTION.
```

Listing 6.11 BTE Process 00001040 Interface

A subscribed function module is expected to return printing device properties in structure C_FINAA and printing parameters in structure C_ITCPO. It can also return archive parameters in structures C_ARCHIVE_INDEX and C_ARCHIVE_PARAMS. The implementation should acknowledge the I_UPDATE flag: If the flag is blank, then the event is called for test printing; otherwise, the printing is productive. The purpose of other parameters should be clear by their definitions.

If you don't use the printing functions mentioned previously, then process 00001040 is irrelevant.

6.2.3 Dunning Summary

The dunning process has a wide choice of enhancement capabilities at many different levels. By possessing such tools, you can implement sophisticated dunning output automation, which allows you to incorporate virtually any technique and media.

6.3 Payment Program

The payment program is another example of an outbound accounting interface. The payment program processes vendor or customer open items, which must be paid via a bank account. Each specific payment run is identified by two key fields: run date (LAUFD) and character ID (LAUFI). This is similar to the dunning process.

The process of an automatic payment run is divided into two phases:

1. **Generating the payment proposal**
   Technically, payment proposal data is stored in database tables REGUH and REGUP, and payment run parameters are located in FB table REGUV.
2. **Payment posting and generating payment media**
   Payment media can be just a set of printed payment notices for a bank, a set of EDI messages, or a file in a specific bank format.

The main entry point for the payment run is Transaction F110. You can set parameters for the payment run and schedule payment proposal creation and payment posting creation.

The method of payment media creation (printout form, EDI message, or file) is defined in *Payment Method*, which is a set of configuration parameters stored in a number of customizing tables T042*. There are two options for media creation: via Print Workbench and DME (data medium exchange) engine, or by a classical printing program (the names of these programs traditionally start with RFF).

Technically, the control logic of Transaction F110 is mainly implemented in module pool `SAPF110V`, while payment proposal and payment creation is performed in report SAPF110S.

We'll now discuss available enhancement options in the payment control utility (Transaction F110) and in payment program SAPF110S.

6.3.1 User Exits in Transaction F110

Unlike the dunning control utility (Transaction F150), the user interface of Transaction F110 cannot be enhanced. Nevertheless, a number of enhancements are available, which we'll highlight in the following subsections.

BAdI Definition FI_F110_SCHEDULE_JOB

The only interface method of the BAdI is called when the user saves payment run schedule parameters. Parameters are passed to the method in structure `F110V`. Depending on the check logic, the method returns an `X` (if everything is okay) or a space (if not okay) in the export parameter `E_PARAM_OK`.

Payment Release List BTEs

Transaction F110 also has two BTEs that work only if the SAP application *Payment Release List* is activated. This application can be activated and configured using Transaction FPRL_CUSTOMIZING, which opens a subset of IMG settings. In the IDES system, however, these events do not have entries in BTE configuration tables, so they might be reserved for future use.

Event 00002105

This event is called just before opening a pop-up screen with payment run schedule parameters. The interface of the event is shown in Listing 6.12.

```
FUNCTION SAMPLE_INTERFACE_00002105.
*"----------------------------------------------------------------------
*"*"Lokale Schnittstelle:
*"  IMPORTING
*"     REFERENCE(I_LAUFD) TYPE  LAUFD
*"     REFERENCE(I_LAUFI) TYPE  LAUFI
*"     REFERENCE(I_XVORL) TYPE  XVORL
*"----------------------------------------------------------------------
ENDFUNCTION.
```

Listing 6.12 BTE 00002105 Interface

Parameters I_LAUFD and I_LAUFI identify the payment run being processed. Flag I_XVORL shows the mode of the run: If it's X, then the run is for payment proposal creation; otherwise, the run is for payment posting and printing.

Process 00001819

This process is called after the run is executed, including printing payment media. Note that process 00001819 doesn't have any changing or export parameters. The interface of the process is shown in Listing 6.13.

```
FUNCTION OPEN_FI_PERFORM_00001819_P.
*"----------------------------------------------------------------------
*"*"Lokale Schnittstelle:
*"  IMPORTING
*"     REFERENCE(I_XVORL) TYPE  XVORL
*"     REFERENCE(I_LAUFD) TYPE  LAUFD
*"     REFERENCE(I_LAUFI) TYPE  LAUFI
*"     REFERENCE(IS_JOBNAME) OPTIONAL
*"     REFERENCE(I_JOBCOUNT) TYPE  C OPTIONAL
```

```
*"----------------------------------------------------------------------
ENDFUNCTION.
```

Listing 6.13 BTE Process 00001819 Interface

In addition to the payment run identification and run mode, this process also includes the background job name and count.

6.3.2 User Exits in Payment Program SAPF110S

The payment run in the payment program is identified according to schedule parameters by run date and identifier, which were set in Transaction F110. During its run, the payment program processes vendor and customer open items to be paid, posts payment documents to accounting to represent bank transactions, and updates the payment run information into database tables REGUH and REGUP.

Process 00001820—PAYMENT PROGRAM: Item Selection

This process is called when the system is processing customer or vendor open items. The interface of the process is shown in Listing 6.14.

```
FUNCTION SAMPLE_PROCESS_00001820.
*"----------------------------------------------------------------------
*"*"Lokale Schnittstelle:
*"  IMPORTING
*"     REFERENCE(I_BSID) TYPE  BSID OPTIONAL
*"     REFERENCE(I_BSIK) TYPE  BSIK OPTIONAL
*"     REFERENCE(I_KOART) LIKE  BSEG-KOART
*"     REFERENCE(I_BUDAT) LIKE  F110C-BUDAT OPTIONAL
*"     REFERENCE(I_NEDAT) LIKE  F110V-NEDAT OPTIONAL
*"     REFERENCE(I_FDEBI) LIKE  F110V-FDEBI OPTIONAL
*"     REFERENCE(I_TRACE) LIKE  TRCOPT STRUCTURE  TRCOPT OPTIONAL
*"  EXPORTING
*"     REFERENCE(E_NO_FREE_SELECTIONS) TYPE  C
*"  TABLES
*"      T_FLDTAB_1820 STRUCTURE  F110_FLDTAB_1820 OPTIONAL
*"  CHANGING
*"     REFERENCE(C_ZLSPR) LIKE  BSEG-ZLSPR
*"     REFERENCE(C_ZLSCH) LIKE  BSEG-ZLSCH
*"----------------------------------------------------------------------
ENDFUNCTION.
```

Listing 6.14 The Interface of BTE Process 00001820

Depending on the I_KOART parameter that represents the type of account being processed, either the I_BSIK (for I_KOART = 'K') or I_BSID (for I_KOART = 'D') structure will be filled with values. I_BSIK is a single item of a vendor; I_BSID is a line item of a customer. As a result of the process logic, changing parameters C_ZLSPR and C_ZLSCH can be returned. C_ZLSPR is a payment blocking indicator, whereas C_ZLSCH is the new value for the payment method.

Process 00001830—PAYMENT PROGRAM: Edit Group

When processing payment data, some line items are gathered into groups by the payment program, which will then be posted as a single accounting document. Process 00001830 can be used to cancel processing of the whole group of items or of individual items in the group (see Listing 6.15).

```
FUNCTION SAMPLE_PROCESS_00001830.
*"----------------------------------------------------------------------
*"*"Lokale Schnittstelle:
*"  IMPORTING
*"     REFERENCE(I_BUDAT) LIKE  F110C-BUDAT OPTIONAL
*"     REFERENCE(I_NEDAT) LIKE  F110V-NEDAT OPTIONAL
*"     REFERENCE(I_FDEBI) LIKE  F110V-FDEBI OPTIONAL
*"     REFERENCE(I_TRACE) LIKE  TRCOPT STRUCTURE  TRCOPT OPTIONAL
*"  TABLES
*"      T_REGUP STRUCTURE  REGUP_1830
*"  CHANGING
*"     REFERENCE(C_REGUH) TYPE  REGUH_1830
*"----------------------------------------------------------------------
ENDFUNCTION.
```

Listing 6.15 00001830 BTE Process Interface

The group header is represented as changing parameter C_REGUH of type REGUH_1820. Group items are passed to the process as a table parameter T_REGUP of structure REGUP_1830. If the process fills C_REGUH-XIGNO with X, then the whole group is ignored. The same rule is relevant to group items: All items with XIGNO = 'X' will be ignored.

Process 00001810—PAYMENT PROGRAM: Individual Bank Determination

This process allows the interception of the standard bank determination logic. Depending on the import parameter values, the process can amend the house bank list (T_HBANK) and the partner bank list (T_PBANK). If payment is relevant for bank chains, then this process can return up to three corresponding bank data in export parameters: `E_KORRESPBANK`, `E_KORRESPBANK2`, and `E_KORRESPBANK3`. The sample interface of process `00001810` is shown in Listing 6.16.

```
FUNCTION SAMPLE_PROCESS_00001810.
*"----------------------------------------------------------------------
*"*"Lokale Schnittstelle:
*"       IMPORTING
*"             VALUE(I_RZAWE) LIKE  REGUH-RZAWE
*"             VALUE(I_WAERS) LIKE  REGUH-WAERS
*"             VALUE(I_RWBTR) LIKE  REGUH-RWBTR
*"             VALUE(I_RBETR) LIKE  REGUH-RBETR
*"             VALUE(I_KUNNR) LIKE  REGUH-KUNNR
*"             VALUE(I_LIFNR) LIKE  REGUH-LIFNR
*"             VALUE(I_ZBUKR) LIKE  REGUH-ZBUKR
*"             VALUE(I_SRTGB) LIKE  REGUH-SRTGB
*"             VALUE(I_SRTBP) LIKE  REGUH-SRTBP
*"             VALUE(I_HBKID) LIKE  ZHLG1-HBKID OPTIONAL
*"       EXPORTING
*"             VALUE(E_KORRESPBANK) LIKE  F110_KBANK
*"                             STRUCTURE  F110_KBANK
*"             VALUE(E_KORRESPBANK2) LIKE  F110_KBANK
*"                             STRUCTURE  F110_KBANK
*"             VALUE(E_KORRESPBANK3) LIKE  F110_KBANK
*"                             STRUCTURE  F110_KBANK
*"       TABLES
*"              T_HBANK STRUCTURE  IHBANK
*"              T_PBANK STRUCTURE  F110_PBANK
*"----------------------------------------------------------------------
ENDFUNCTION.
```

Listing 6.16 The Interface of BTE Process 00001810

Payment Release List BTE Processes

Some BTE processes are only activated if the Payment Release List application is active. We discuss these in the following subsections.

Process 00001821

This process allows you to split a single payment item (represented by an entry in table REGUP) into several items. Its interface is shown in Listing 6.17.

```
FUNCTION SAMPLE_PROCESS_00001821.
*"----------------------------------------------------------------------
*"*"Lokale Schnittstelle:
*"  IMPORTING
*"     REFERENCE(I_XVORL) TYPE  XVORL
*"     REFERENCE(I_LAUFD) TYPE  LAUFD
*"     REFERENCE(I_LAUFI) TYPE  LAUFI
*"     REFERENCE(IS_HEADER) TYPE  HEADER_1821_PRL
*"     REFERENCE(IS_REGUP) TYPE  REGUP
*"  CHANGING
*"     REFERENCE(CT_ITEMS) TYPE  REGUP_T_1821_PRL
ENDFUNCTION.
```

Listing 6.17 00001821 BTE Process Interface

The source `REGUP` structure is passed by import parameter `IS_REGUP`. The result of splitting logic is expected to be returned in changing table parameter `CT_ITEMS`.

Process 00001831

This process can be used to verify the payment method for a group of payments. It is represented by a header (`REGUH`) and several items (`REGUP`). As shown in Listing 6.18, payment group header information is passed by parameter `IS_REGUH`, and the group items are passed by the `IT_REGUP` table parameter. Export parameter `E_RZAWE` should contain the newly determined payment method.

```
FUNCTION SAMPLE_PROCESS_00001831.
*"----------------------------------------------------------------------
*"*"Lokale Schnittstelle:
*"  IMPORTING
*"     REFERENCE(I_XVORL) TYPE  XVORL
*"     REFERENCE(I_LAUFD) TYPE  LAUFD
*"     REFERENCE(I_LAUFI) TYPE  LAUFI
*"     REFERENCE(IS_ZHLG1) TYPE  ZHLG1
*"     REFERENCE(IS_REGUH) TYPE  REGUH
*"     REFERENCE(IT_REGUP) TYPE  FI_T_REGUP
*"  EXPORTING
*"     REFERENCE(E_RZAWE) TYPE  RZAWE
*"----------------------------------------------------------------------
ENDFUNCTION.
```

Listing 6.18 00001831 BTE Process Interface

Process 00001809

This process can be used to implement bank determination logic specific to the Payment Release List functionality. It is called just before the general BTE process 00001810. Listing 6.19 shows that unlike process 00001810, this process doesn't have corresponding bank export parameters.

```
FUNCTION SAMPLE_PROCESS_00001809.
*"----------------------------------------------------------------------
*"*"Lokale Schnittstelle:
*"  IMPORTING
*"     REFERENCE(I_XVORL) TYPE  XVORL
*"     REFERENCE(I_LAUFD) TYPE  LAUFD
*"     REFERENCE(I_LAUFI) TYPE  LAUFI
*"     REFERENCE(IS_REGUH) TYPE  REGUH
*"     REFERENCE(IS_ZHLG1) TYPE  ZHLG1
*"  TABLES
*"      T_HBANK STRUCTURE  IHBANK
*"      T_PBANK STRUCTURE  F110_PBANK
*"----------------------------------------------------------------------
ENDFUNCTION.
```

Listing 6.19 00001809 BTE Process Interface

Process 00001815

Using this process, you can change payment group header information before posting a corresponding accounting document. As shown in Listing 6.20, the process should return changed header information in export parameter ES_REGUH_SF. The logic should be based on the payment group data being processed.

```
FUNCTION SAMPLE_PROCESS_00001815.
*"----------------------------------------------------------------------
*"*"Lokale Schnittstelle:
*"  IMPORTING
*"     REFERENCE(I_XVORL) TYPE  XVORL
*"     REFERENCE(I_LAUFD) TYPE  LAUFD
*"     REFERENCE(I_LAUFI) TYPE  LAUFI
*"     REFERENCE(IS_ZHLG1) TYPE  ZHLG1
*"     REFERENCE(IS_REGUH) TYPE  REGUH
*"     REFERENCE(IT_REGUP) TYPE  FI_T_REGUP
*"  EXPORTING
*"     REFERENCE(ES_REGUH_SF) TYPE  REGUH_CSF_PRL
*"----------------------------------------------------------------------
ENDFUNCTION.
```

Listing 6.20 00001815 BTE Process Interface

Process 00001811

Process `00001811` is only used when the SEPA supporting functionality is active in the system. Using this process, you can implement specific logic for SEPA mandate determination. It is called at the end of the bank determination logic in the payment program.

Note

SEPA is a pan-European method of electronic payment and is said to greatly reduce the complexities of international payments inside the European Union.

You can see in Listing 6.21 that prepared SEPA mandates are passed to the process by a changing table parameter: `CT_MANDATES`. Besides the mandates, the process can return a table of processing messages that will be reported in the payment program log.

```
FUNCTION SAMPLE_PROCESS_00001811.
*"----------------------------------------------------------------------
*"*"Local Interface:
*"  IMPORTING
*"     REFERENCE(I_LAUFD) TYPE  LAUFD
*"     REFERENCE(I_LAUFI) TYPE  LAUFI
*"     REFERENCE(I_XVORL) TYPE  XVORL
*"     REFERENCE(IS_REGUH) TYPE  REGUH
*"     REFERENCE(IS_REGUP) TYPE  REGUP
*"     REFERENCE(I_ZIBAN) TYPE  DZIBAN
*"  EXPORTING
*"     REFERENCE(ET_MESSAGES) TYPE  BAPIRET1_LIST
*"  CHANGING
*"     REFERENCE(CT_MANDATES) TYPE  SEPA_TAB_DATA_MANDATE_DATA
*"----------------------------------------------------------------------
ENDFUNCTION.
```

Listing 6.21 BTE 00001811 Process Interface

6.4 Summary

The information we've discussed in this chapter has again proved the usefulness of the system source code as complete and final system documentation (unless something has been changed by an enhancement package or SAP note implementation). The outbound interface we considered in this chapter is just a subset of the techniques that Financial Accounting can use. Other interfaces in SAP industry solutions or more specific add-ons are also available. In any case, you should always examine existing solutions from the enhancement point of view before starting the development of a brand new interface. Also, when designing your own application, you should always think of those who will maintain and support it after you finish the development (even if it will be you).

In the final chapter, we'll look at another tool that can help extend the system functionality but that is often underestimated and reputed as too complex: SAP Business Workflow.

In this chapter, we briefly discuss SAP Business Workflow, which is another way to extend standard system functionality.

7 Workflow as a User Exit

SAP Business Workflow is a tool for automating a business process when several people have to fulfill different interactive operations during a process flow. The most common scenario to implement with SAP Business Workflow is an approval procedure of different kinds, such as payment approval, master data change, creation approval, and so on.

As an example, in Financial Accounting customizing, you can mark specific vendor or customer master record fields as sensitive so that any change made by one user to such a field requires confirmation from another user who has sufficient authorization. It's logical to make the system notify the authorized user of a vendor (or customer) account change that must be confirmed or rejected. This is the task for SAP Business Workflow.

You can also use SAP Business Workflow functionality for performing background tasks, which is in a way closer to a common enhancement practice we were discussing in this book—when something works silently without user intervention. For example, on one project, the customer requires that after creating a vendor master record, which belongs to a particular account group, the corresponding customer master record must be created with the same name, address, and some other fields. This activity must also be performed without user interaction.

Note

For more information on SAP Business Workflow, we recommend *Practical Workflow for SAP*, Second Edition, by Ginger Gatling et al. (SAP PRESS, 2009). You may also want to visit an excellent SDN blog series by Jocelyn Dart concerning ABAP programming in the SAP Business Workflow framework: at *www.sdn.sap.com/irj/scn/weblogs?blog=/pub/u/4075*.

In the following sections, we briefly review the main techniques of linking standard system activities with custom-defined applications, as well as the main concepts and objects of SAP Business Workflow.

7.1 Workflow Events: Linking System Actions with External Applications

Events in SAP Business Workflow can be compared to nerves in the human body: They transfer signals from one application to another making the whole design alive. Next, we'll consider how events are handled; what tools you can use to create them, and what you should be aware of when developing applications for SAP Business Workflow.

7.1.1 Event Handling

The SAP Business Workflow runtime system is informed of different business activities such as vendor account creation or an incoming invoice parking by means of a workflow event. A workflow event is a kind of P&S interface resembling BTEs (or a BAdI definition with multiple implementations). Unlike BTEs, you don't subscribe function modules to workflow events. Instead, you use special intermittent development entities, workflow templates and standard tasks, which can be subscribed to a particular workflow event.

Note

In earlier SAP releases, the workflow event was part of a business object (BO) definition. BO is an old incarnation of the object oriented (OO) paradigm in SAP. A BO definition is a development object maintained in Transaction SWO1. As of SAP NetWeaver AS 6.40, a workflow event can be defined as a component of an ABAP global class.

A standard task is a single-step workflow, whereas the workflow template can contain several standard tasks connected by different routes. The workflow template is also called a multiple-step workflow. Each workflow event can have one or more subscribers, which adds great flexibility to the design.

If you look at the runtime framework of BTEs, you can see that they are synchronous, similar to a subroutine call. The main task passes complete control to the BTE subscription function until the end of its execution. And, unless you use remote BTEs (with RFC destination), the whole execution of the BTE is performed within the runtime context of the original task. If there is more than one subscription function, then they are executed sequentially. See the schematic control flow during a BTE call in Figure 7.1.

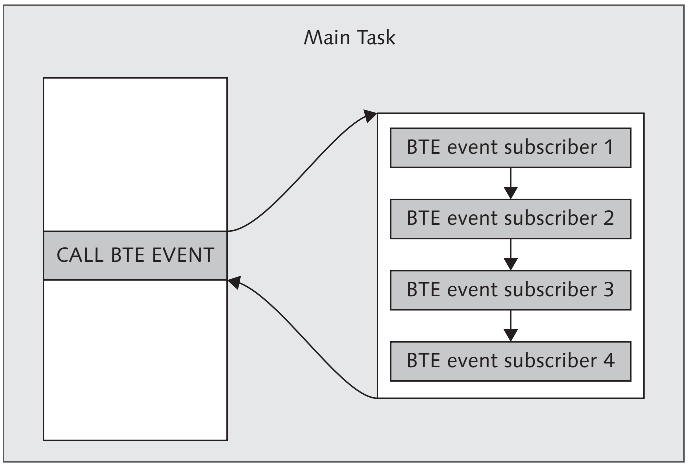


Figure 7.1 Runtime Control Flow During BTE Call

On the contrary, a workflow event is linked to the host task asynchronously, so after firing the event, the host task continues its execution. Each event subscriber is executed independently in its own separate task as schematically shown in Figure 7.2.

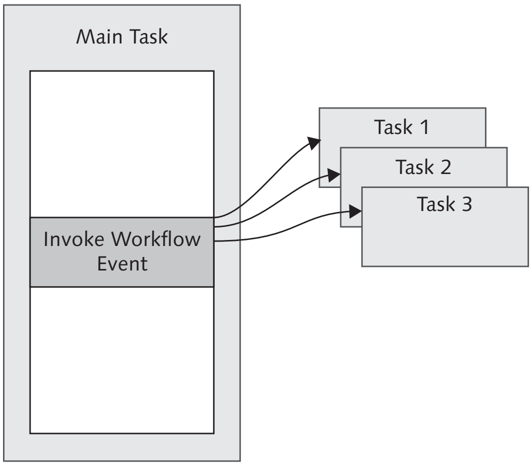
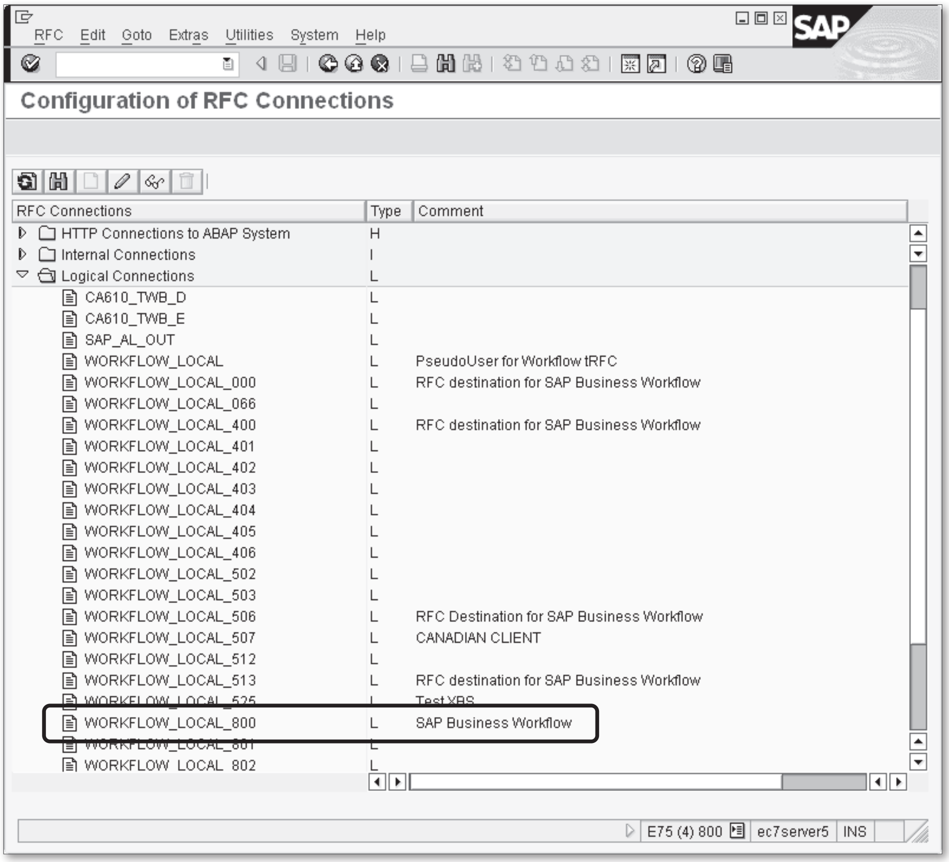


Figure 7.2 Runtime Control Flow During Workflow Event Processing

For a BTE, the call of the event is just a synchronous function module call (or synchronous RFC call). Workflow events handling is arranged in a more sophisticated manner. For simplicity, you can assume that each workflow event handler is started using a background RFC call in the form of `CALL FUNCTION IN BACKGROUND TASK AS SEPARATE UNIT`. The RFC destination points to the same working system and has a predefined special user account that is part of SAP Business Workflow Customizing. If workflow functionality was ever used in your system, you can find special RFC destinations in Transaction SM59 under the LOGICAL CONNECTIONS subtree with the name of format `WORKFLOW_LOCAL_NNN`, where `NNN` is a client number (see Figure 7.3).



Configuration of RFC Connections

RFC Connections	Type	Comment
HTTP Connections to ABAP System	H	
Internal Connections	I	
Logical Connections	L	
CA610_TWB_D	L	
CA610_TWB_E	L	
SAP_AL_OUT	L	
WORKFLOW_LOCAL	L	PseudoUser for Workflow tRFC
WORKFLOW_LOCAL_000	L	RFC destination for SAP Business Workflow
WORKFLOW_LOCAL_066	L	
WORKFLOW_LOCAL_400	L	RFC destination for SAP Business Workflow
WORKFLOW_LOCAL_401	L	
WORKFLOW_LOCAL_402	L	
WORKFLOW_LOCAL_403	L	
WORKFLOW_LOCAL_404	L	
WORKFLOW_LOCAL_405	L	
WORKFLOW_LOCAL_406	L	
WORKFLOW_LOCAL_502	L	
WORKFLOW_LOCAL_503	L	
WORKFLOW_LOCAL_506	L	RFC Destination for SAP Business Workflow
WORKFLOW_LOCAL_507	L	CANADIAN CLIENT
WORKFLOW_LOCAL_512	L	
WORKFLOW_LOCAL_513	L	RFC destination for SAP Business Workflow
WORKFLOW_LOCAL_525	L	Test XBS
WORKFLOW_LOCAL_800	L	SAP Business Workflow
WORKFLOW_LOCAL_801	L	
WORKFLOW_LOCAL_802	L	

E75 (4) 800 | ec7server5 | INS

Figure 7.3 Workflow-Specific RFC Destinations

7.1.2 Event Creation Options

Workflow events can be created in the system with different techniques:

- Some predefined workflow events are created in the system automatically by SAP applications.
- Workflow event creation can be linked to change documents.
- Workflow events can be created programmatically using standard SAP function module `SWE_EVENT_CREATE` from any custom-defined application (e.g., from within an enhancement implementation).

System workflow events (including those generated by change documents) are actually created as part of an asynchronous update process. This update process is the recommended technique for event creation in customer applications. In other words, you should create a workflow event by means of the `CALL FUNCTION IN UPDATE TASK` statement. The technique guarantees that the event will be created only upon a successful `COMMIT` execution.

7.1.3 Application Development Implications

There are a number of main implications of the SAP Business Workflow event handling for application development, such as the following:

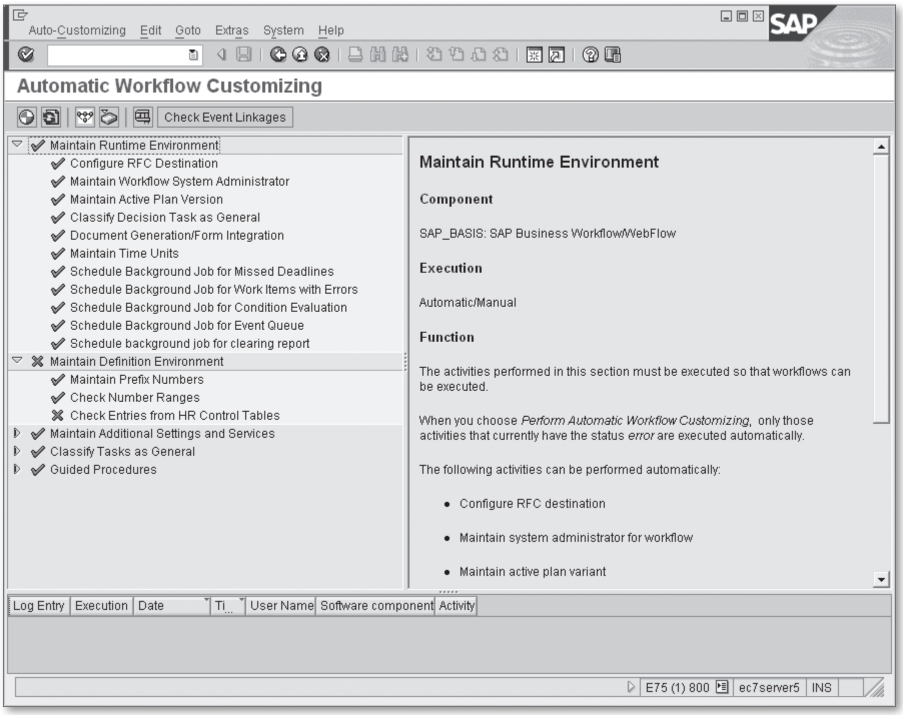
- Due to the asynchronous character of event handling, the process has less impact on user productivity to compare with synchronous user exits.
- If an application error occurs during workflow event handling it doesn't affect the source application; on the other hand, bug investigation in workflow applications can be more complex again due to the asynchronous nature of the process.
- Because workflow event handling is executed in its own memory context, it isn't possible to access memory areas of the host application during event processing.
- The event handler can report a temporary error (e.g., if some resource is blocked by another application). In this case, the workflow runtime system can restart the handler automatically in a predefined period of time.
- Some event processing options, such as the event queue, are configured independently of the customer application. This can impact the processing delay, which can vary from milliseconds to minutes.

7.2 Practical Example

As an example, let's create a linkage between vendor master record creation and our own external application. To keep pace with modern technology, we'll do it from scratch using ABAP objects as a foundation for our developments.

7.2.1 Prerequisites

Certainly this small chapter cannot be a comprehensive guide to SAP Business Workflow customizing and design. We assume that all of the necessary configuration activity is already performed in your system.


Auto-Customizing Edit Goto Extras System Help
SAP
Check Event Linkages
Log Entry | Execution | Date | Ti... | User Name | Software component | Activity
E75 (1) 800 | ec7server5 | INS

Automatic Workflow Customizing

- ✓ Maintain Runtime Environment
  - ✓ Configure RFC Destination
  - ✓ Maintain Workflow System Administrator
  - ✓ Maintain Active Plan Version
  - ✓ Classify Decision Task as General
  - ✓ Document Generation/Form Integration
  - ✓ Maintain Time Units
  - ✓ Schedule Background Job for Missed Deadlines
  - ✓ Schedule Background Job for Work Items with Errors
  - ✓ Schedule Background Job for Condition Evaluation
  - ✓ Schedule Background Job for Event Queue
  - ✓ Schedule background job for clearing report
- ✗ Maintain Definition Environment
  - ✓ Maintain Prefix Numbers
  - ✓ Check Number Ranges
  - ✗ Check Entries from HR Control Tables
- ✓ Maintain Additional Settings and Services
- ✓ Classify Tasks as General
- ✓ Guided Procedures

Maintain Runtime Environment

Component

SAP_BASIS: SAP Business Workflow/WebFlow

Execution

Automatic/Manual

Function

The activities performed in this section must be executed so that workflows can be executed.

When you choose *Perform Automatic Workflow Customizing*, only those activities that currently have the status *error* are executed automatically.

The following activities can be performed automatically:

- Configure RFC destination
- Maintain system administrator for workflow
- Maintain active plan variant

Figure 7.4 Transaction SWU3—The Starting Point of SAP Business Workflow Configuration

To check SAP Business Workflow configuration and also perform automatic Customizing, use Transaction SWU3. The required settings are shown in Figure 7.4. The rule of thumb for this transaction is that you should assure that all elements of the subtree MAINTAIN RUNTIME ENVIRONMENT are marked with green ticks and at least the two upmost elements of the MAINTAIN DEFINITION ENVIRONMENT subtree (Maintain Prefix Numbers and Check Number Ranges) should also be green. If this is the case in your system, then you can create SAP Business Workflow development objects and run workflows.

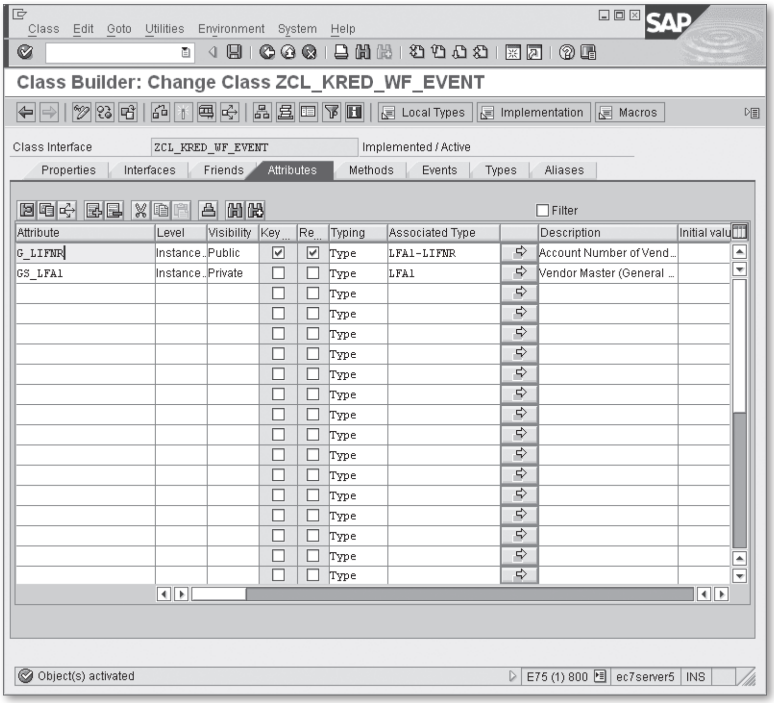
7.2.2 Workflow-Enabled Class

In the IDES system, we created a demo class named `ZCL_KRED_WF_EVENT`. To make it workflow-enabled, we must add to its definition `IF_WORKFLOW` interface, which in reality is a combination of two other interfaces: `BI_OBJECT` and `BI_PERSISTENT`.

Attributes

Because we plan to implement manipulations with the vendor master record, we add public read-only attribute `G_LIFNR` of type `LIFNR` and mark it as a key attribute. This attribute will be a unique identifier of our class instance in the runtime.

We also create a private attribute, `GS_LFA1` of type `LFA1`, which will store the corresponding vendor record in the runtime. Figure 7.5 shows the Attributes section of the class as it is seen in Transaction SE24.



Class Builder: Change Class ZCL_KRED_WF_EVENT

Class Interface ZCL_KRED_WF_EVENT Implemented / Active

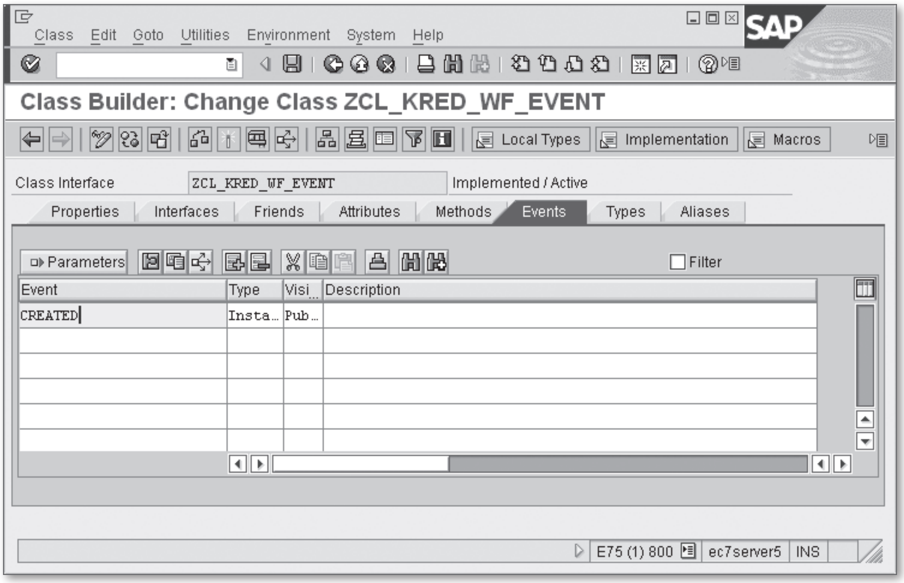
Attribute	Level	Visibility	Key	Re...	Typing	Associated Type		Description	Initial valu
G_LIFNR	Instance..	Public	☑	☑	Type	LFA1-LIFNR	⇨	Account Number of Vend...	
GS_LFA1	Instance..	Private	☐	☐	Type	LFA1	⇨	Vendor Master (General ...	
			☐	☐	Type		⇨		
			☐	☐	Type		⇨		
			☐	☐	Type		⇨		
			☐	☐	Type		⇨		
			☐	☐	Type		⇨		
			☐	☐	Type		⇨		
			☐	☐	Type		⇨		
			☐	☐	Type		⇨		
			☐	☐	Type		⇨		
			☐	☐	Type		⇨		
			☐	☐	Type		⇨		
			☐	☐	Type		⇨		
			☐	☐	Type		⇨		
			☐	☐	Type		⇨		
			☐	☐	Type		⇨		

Object(s) activated | E75 (1) 800 | ec7server5 | INS

Figure 7.5 Attributes of the Workflow-Enabled Class

Events

Now we define the public event CREATED, which will be the workflow event we hope to invoke upon creation of the vendor master record. Figure 7.6 shows the EVENTS tab of our class definition.



Class Builder: Change Class ZCL_KRED_WF_EVENT

Class Interface ZCL_KRED_WF_EVENT Implemented / Active

Event	Type	Visi...	Description
CREATED	Insta...	Pub...	

Figure 7.6 Event Definition of the Workflow-Enabled Class

Methods

When the workflow runtime system tries to execute event handlers, it passes an object reference (in our task it's a vendor number) in specific internal format. To convert the reference from and to an internal system representation, we have to implement two methods of interface BI_PERSISTENT: FIND_BY_LPOR and LPOR (Local Persistent Object Reference). LPOR consists of object identification (in our case, it's a vendor number), object type category (for ABAP classes, it's CL), and object type name (ZCL_KRED_WF_EVENT for our class).

The BI_PERSISTENT~FIND_BY_LPOR method is static, and its goal is to find the corresponding business object in the system, create an instance of our class, and return the instance as a result. LPOR has a structure of SIBFLPOR. In this case, the system passes an internal representation of the vendor number to the method, and we then have to extract the vendor number and create an instance of our class. See the implementation in Listing 7.1.

```
METHOD bi_persistent~find_by_lpor.
  DATA: local_ref TYPE REF TO zcl_kred_wf_event.
  CREATE OBJECT local_ref EXPORTING i_lpor = lpor.
  result = local_ref.
ENDMETHOD.
```

Listing 7.1 BI_PERSISTENT~FIND_BY_LPOR Method Implementation

`BI_PERSISTENT~LPOR` is an instance method, and it must return an internal representation of the business object identification in the form of a `LPOR`. In this case, we have to convert the vendor number into a `LPOR`. The source code of the method is shown in Listing 7.2.

```
METHOD bi_persistent~lpor.
  result-catid = ‘CL’.
  result-typeid = ‘ZCL_KRED_WF_EVENT’.
  result-instid = g_lifnr.
ENDMETHOD.
```

Listing 7.2 BI_PERSISTENT~LPOR Method Implementation

Another interface method we need to implement is `BI_PERSISTENT~REFRESH`. It should refresh runtime instance data from the corresponding database data. In our example, it can select a corresponding record from the Table LFA1 database (see Listing 7.3).

```
METHOD bi_persistent~refresh.
  SELECT SINGLE * INTO gs_lfa1 FROM lfa1 WHERE lifnr = g_lifnr.
ENDMETHOD.
```

Listing 7.3 BI_PERSISTENT~REFRESH Method Implementation

Now we implement the instance `CONSTRUCTOR` method with only one parameter: `LPOR`. For our task, the implementation of the `CONSTRUCTOR` can look like Listing 7.4. Here we just initialize our key attribute `G_LIFNR` and call method `BI_PERSISTENT~REFRESH` to complete the task.

```
METHOD constructor.
  g_lifnr = i_lpor-instid.
  me->bi_persistent~refresh( ).
ENDMETHOD.
```

Listing 7.4 CONSTRUCTOR Implementation

Note

In the available IDES system (which is SAP ERP with EhP4) the installed methods BI_PERSISTENT~FIND_BY_LPOR and BI_PERSISTENT~REFRESH don't have any exceptions. They should have exceptions, however, because these methods should be capable of reporting errors to the workflow runtime system in case; for example, the object can't be found. At the time of this writing, the reason for the lack of exceptions in these methods remains unclear.

Functional Method

At this part of the process, we should implement a demo method, which we'll run in the background as a reaction to vendor creation. We won't create a real application; instead, we'll just throw some message for demonstration purposes.

We name the demo method BACKGROUND_METHOD. This will be an instance and public method. We also declare that the method can throw a class-oriented exception: CX_BO_APPLICATION. This is a standard exception root for workflow-enabled methods. In the implementation (Listing 7.5), we only throw exception CX_BO_ERROR (which is a subclass of CX_BO_APPLICATION) with a method name.

```
METHOD background_method.
  RAISE EXCEPTION TYPE cx_bo_error
    EXPORTING class_name = 'BACKGROUND_METHOD'.
ENDMETHOD.
```

Listing 7.5 BACKGROUND_METHOD Implementation

Now the class is ready to be used in workflow event linkage.

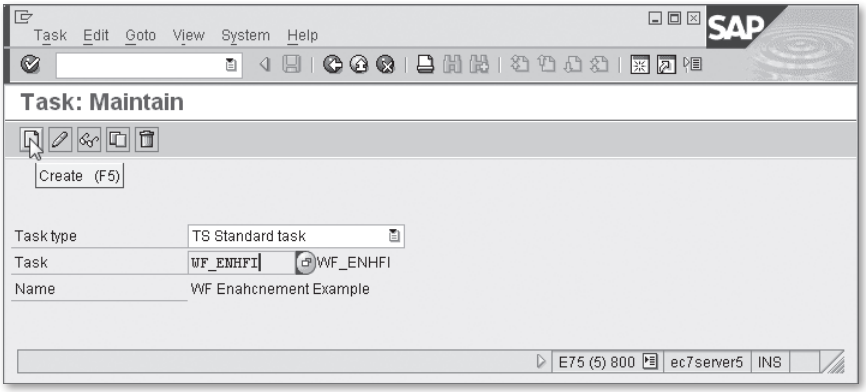
7.2.3 Standard Task

The standard task is an intermittent object that contains additional properties of an executable code, which makes it accessible from the context of the workflow runtime system. Standard tasks are maintained in Transaction PFTC.

Note

A standard task is the development cross-client object. When you save the object, the system requests a package name for the standard task.

For the sake of our example, we need a standard task to execute our BACKGROUND_METHOD in the context of the workflow. On the first screen of Transaction PFTC, we choose the STANDARD TASK value in the TASK TYPE list box, enter "WF_ENHFI" into the TASK field, and finally click the CREATE toolbar button as shown in Figure 7.7.



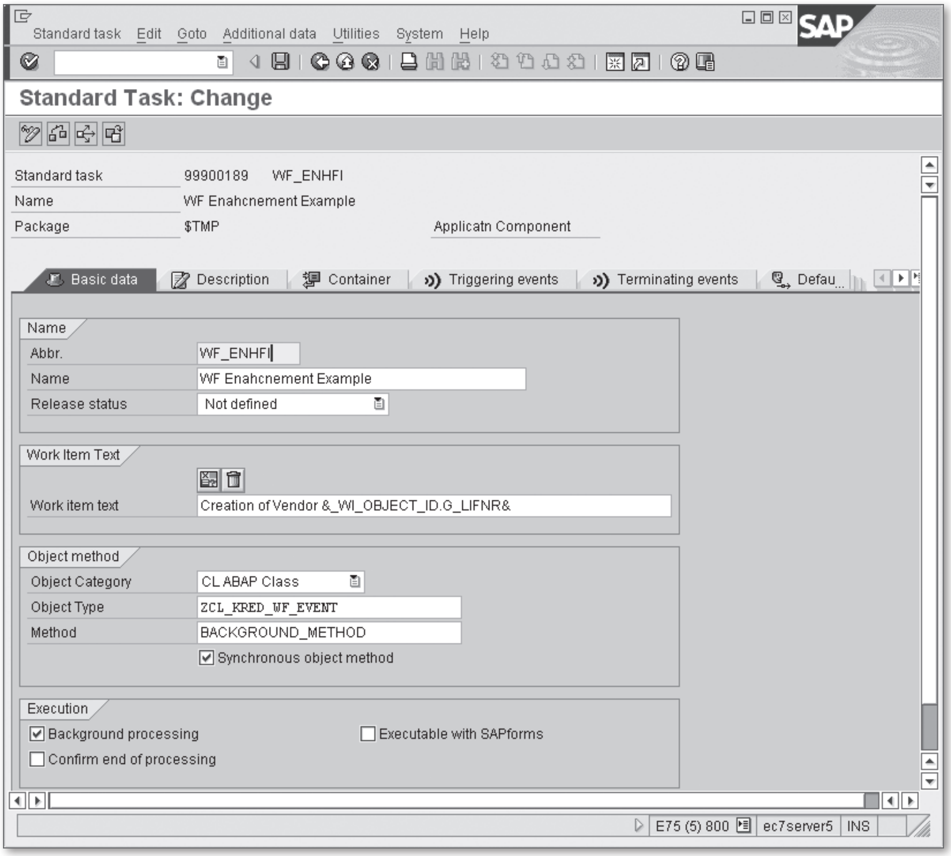
Task type	TS Standard task
Task	WF_ENHFI WF_ENHFI
Name	WF Enahcnement Example

Figure 7.7 PFTC Transaction Starting Screen

Basic Data

On the next screen, we enter arbitrary information into the descriptive text fields (ABBR., NAME, WORK ITEM TEXT) as shown in Figure 7.8. The main fields we have to fill in are located in the box OBJECT METHOD. Here we enter "ABAP Class" as OBJECT CATEGORY, "ZCL_KRED_WF_EVENT" as OBJECT TYPE, and "BACKGROUND_METHOD" as METHOD. We also need to tick the SYNCHRONOUS OBJECT METHOD and BACKGROUND PROCESSING checkboxes below the method name.

After saving the task, the system assigns its number automatically. Our task has number 99900189. The fully qualified identifier of the task will be TS99900189.



Standard task 99900189 WF_ENHFI
Name WF Enahcnement Example
Package $TMP
Applicatn Component

Name
Abbr. WF_ENHFI
Name WF Enahcnement Example
Release status Not defined

Work Item Text
Work item text Creation of Vendor &_WI_OBJECT_ID.G_LIFNR&

Object method
Object Category CL ABAP Class
Object Type ZCL_KRED_WF_EVENT
Method BACKGROUND_METHOD
☑ Synchronous object method

Execution
☑ Background processing
☐ Executable with SAPforms
☐ Confirm end of processing

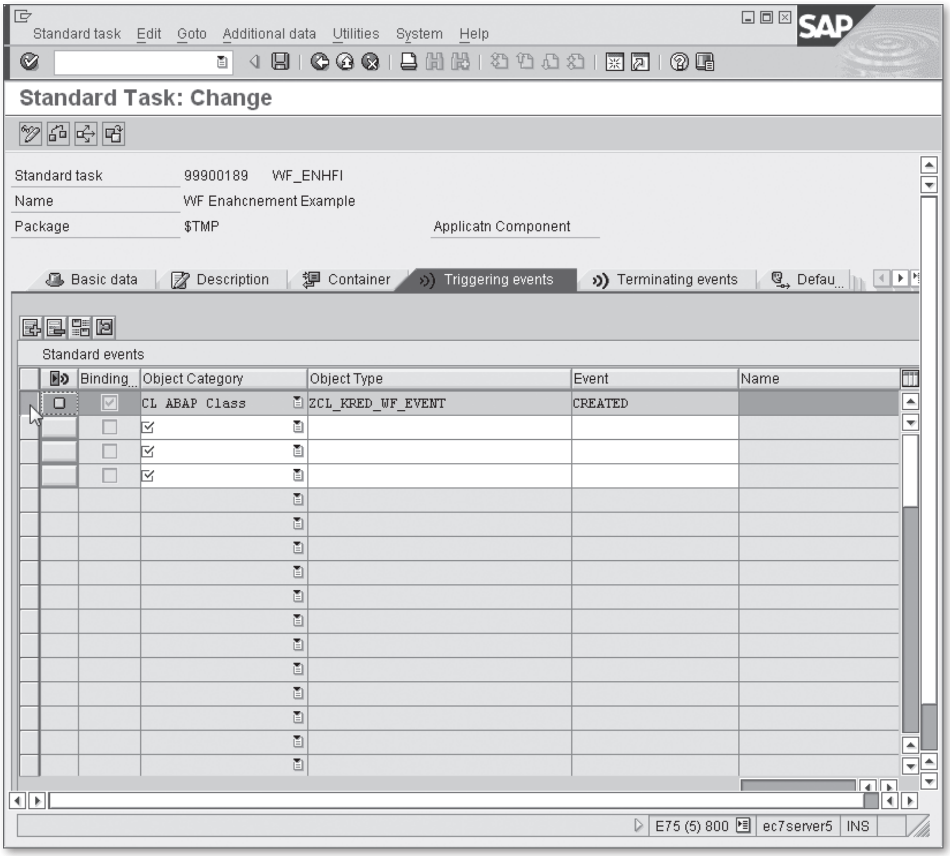
E75 (5) 800 ec7server5 INS

Figure 7.8 Standard Task Basic Attributes

Triggering Events

The next data tasks we have to define are located on the TRIGGERING EVENTS tab. Here we specify an event the task will start upon. For our example, we specify the event `CREATED` from our class `ZCL_KRED_WF_EVENT` (see Figure 7.9). Note that we also specify "ABAP Class" as the OBJECT CATEGORY.

To finalize the triggering event setting, we click on the activation button. This button is gray but turns green with activation. This is a part of customizing; when clicking the event activation, a pop-up window with the CUSTOMIZING TRANSPORT REQUEST selection appears.



Standard task	99900189 WF_ENHFI
Name	WF Enahcnement Example
Package	$TMP
Applicatn Component	

Standard events

Binding	Object Category	Object Type	Event	Name
✓	CL ABAP Class	ZCL_KRED_WF_EVENT	CREATED	

Figure 7.9 Standard Task Triggering Events

After entering the triggering event linkage, we can save the task. It's now ready to use in SAP Business Workflow.

Note

For the sake of simplicity, in this example, we omit parameter-passing options for the method and event (called *binding* here). For our example, system default settings should be enough.

7.2.4 Event Creation

Now that we've prepared all of the development objects, we can link the vendor master creation to our newly created objects in Transaction SWEC. In this transaction, we maintain the linkage between the system change documents and workflow events. The vendor master has its predefined change document object `KRED`. Follow these steps (see Figure 7.10):

1. Enter "KRED" as the value for CHANGE DOC.OBJECT.
2. Enter "ABAP class" in the OBJECT CATEGORY field.
3. Enter "ZCL_KRED_WF_EVENT" as the OBJECT TYPE.
4. Enter "CREATED" as the EVENT type.
5. Tick the CREATE radio-button.

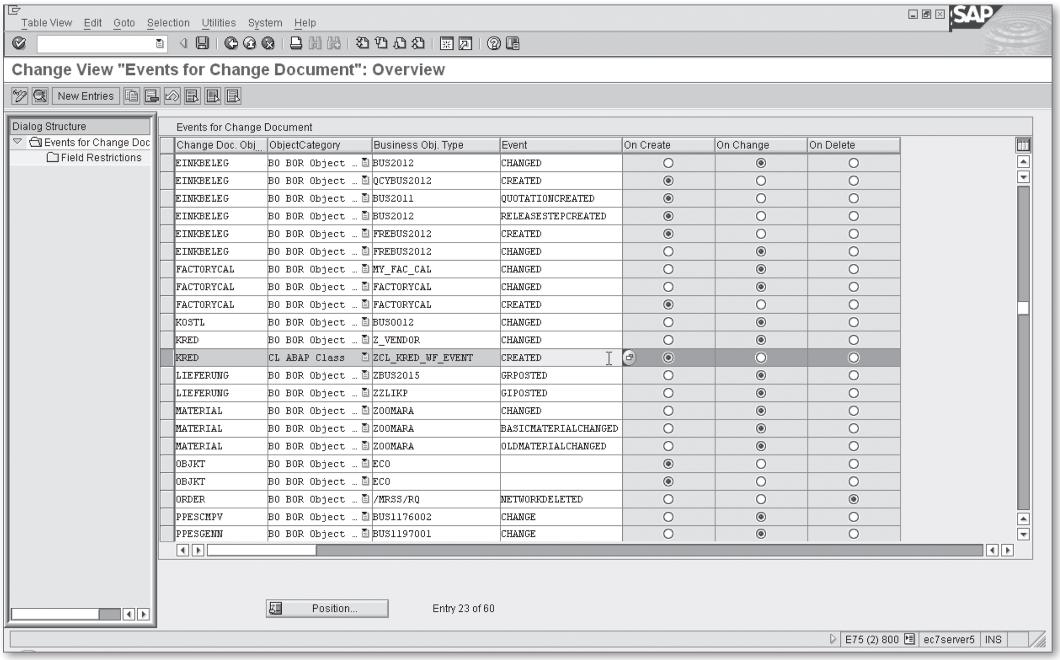


Table View Edit Goto Selection Utilities System Help
Change View "Events for Change Document": Overview
New Entries
Dialog Structure
Events for Change Doc
Field Restrictions
Events for Change Document
Position...
Entry 23 of 60
E75 (2) 800
ec7server5
INS

Change Doc. Obj.	ObjectCategory	Business Obj. Type	Event	On Create	On Change	On Delete
EINKBELEG	BO BOR Object ...	BUS2012	CHANGED	○	●	○
EINKBELEG	BO BOR Object ...	QCYBUS2012	CREATED	●	○	○
EINKBELEG	BO BOR Object ...	BUS2011	QUOTATIONCREATED	●	○	○
EINKBELEG	BO BOR Object ...	BUS2012	RELEASESTEPCREATED	●	○	○
EINKBELEG	BO BOR Object ...	FREBUS2012	CREATED	●	○	○
EINKBELEG	BO BOR Object ...	FREBUS2012	CHANGED	○	●	○
FACTORYCAL	BO BOR Object ...	MY_FAC_CAL	CHANGED	○	●	○
FACTORYCAL	BO BOR Object ...	FACTORYCAL	CHANGED	○	●	○
FACTORYCAL	BO BOR Object ...	FACTORYCAL	CREATED	●	○	○
KOSTL	BO BOR Object ...	BUS0012	CHANGED	○	●	○
KRED	BO BOR Object ...	Z_VENDOR	CHANGED	○	●	○
KRED	CL ABAP Class	ZCL_KRED_WF_EVENT	CREATED	●	○	○
LIEFERUNG	BO BOR Object ...	ZBUS2015	GRPOSTED	○	●	○
LIEFERUNG	BO BOR Object ...	ZZLIKP	GIPOSTED	○	●	○
MATERIAL	BO BOR Object ...	ZOOMARA	CHANGED	○	●	○
MATERIAL	BO BOR Object ...	ZOOMARA	BASICMATERIALCHANGED	○	●	○
MATERIAL	BO BOR Object ...	ZOOMARA	OLDMATERIALCHANGED	○	●	○
OBJKT	BO BOR Object ...	ECO		●	○	○
OBJKT	BO BOR Object ...	ECO		●	○	○
ORDER	BO BOR Object ...	/MRSS/RQ	NETWORKDELETED	○	○	●
PPESCMPV	BO BOR Object ...	BUS1176002	CHANGE	○	●	○
PPESGENN	BO BOR Object ...	BUS1197001	CHANGE	○	●	○

Figure 7.10 Change Document Linkage to Workflow Events

By following these steps, you tell the system to invoke the event when the vendor master record is created.

Note

You can also set field restrictions for each event linkage to further refine event starting conditions.

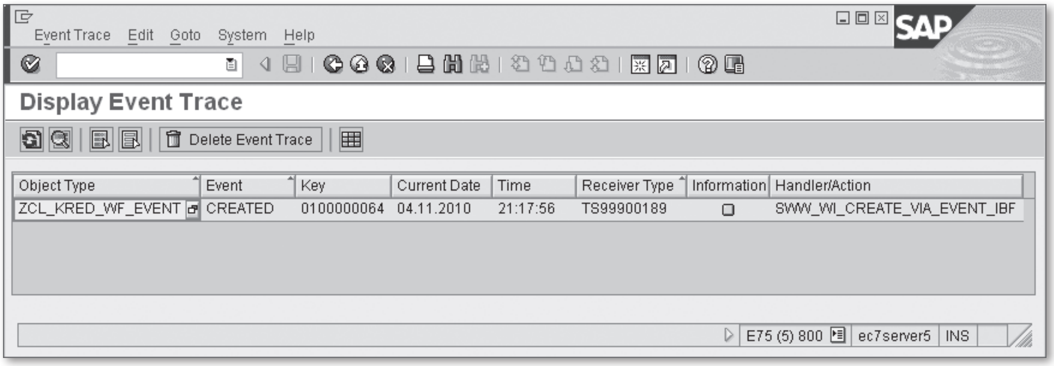
7.2.5 Now Test!

For testing purposes, it's recommended to first turn on the workflow event trace through Transaction SWELS.

Note

The event trace should be used mainly in the test system. Unrestricted event traces can consume considerable database space.

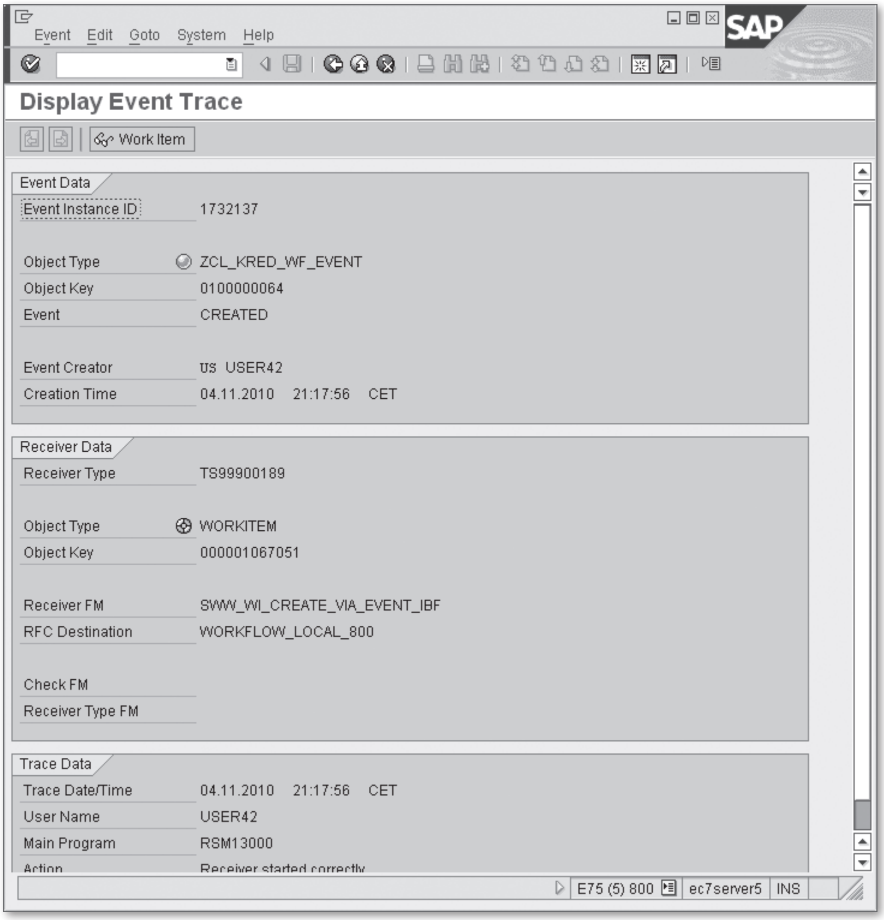
Now let's create a vendor master record and see if our event linkage is working. In the IDES system, we created a copy of T-K521C00 vendor with number 0100000064. After vendor creation, look into the workflow event trace in Transaction SWEL. See the result in Figure 7.11.



Object Type	Event	Key	Current Date	Time	Receiver Type	Information	Handler/Action
ZCL_KRED_WF_EVENT	CREATED	0100000064	04.11.2010	21:17:56	TS99900189	☐	SWW_WI_CREATE_VIA_EVENT_IBF

Figure 7.11 Event Trace After Vendor Creation

The event was invoked successfully. By double-clicking on the event line, you can see more details (see Figure 7.12).



Event Data

Event Instance ID	1732137
Object Type	ZCL_KRED_WF_EVENT
Object Key	0100000064
Event	CREATED
Event Creator	US USER42
Creation Time	04.11.2010 21:17:56 CET

Receiver Data

Receiver Type	TS99900189
Object Type	WORKITEM
Object Key	000001067051
Receiver FM	SWW_WI_CREATE_VIA_EVENT_IBF
RFC Destination	WORKFLOW_LOCAL_800
Check FM	
Receiver Type FM	

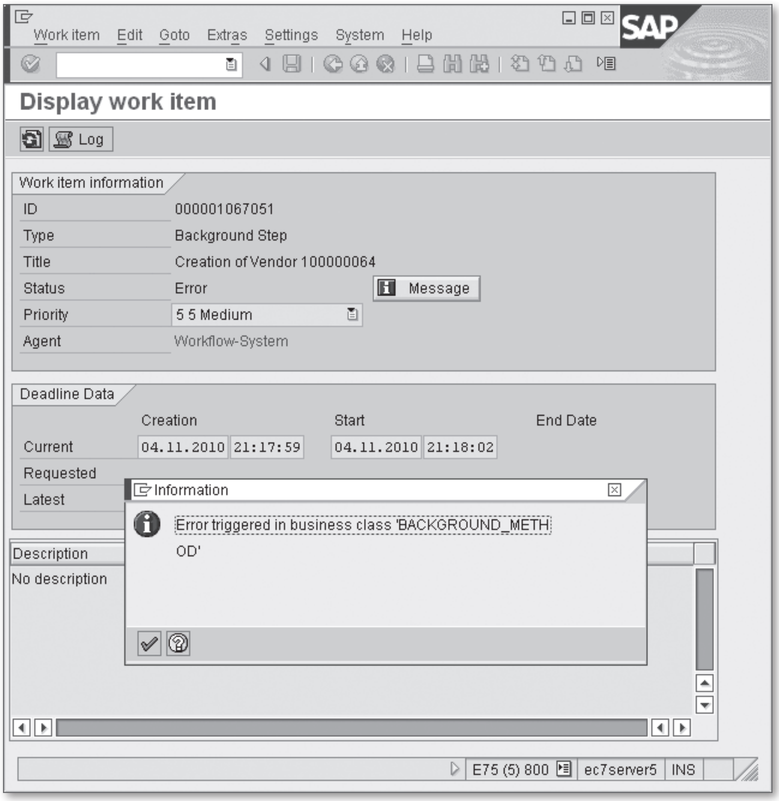
Trace Data

Trace Date/Time	04.11.2010 21:17:56 CET
User Name	USER42
Main Program	RSM13000
Action	Receiver started correctly

E75 (5) 800 ec7server5 INS

Figure 7.12 Event Trace Detailed Information

To see even more details, click the WORK ITEM toolbar button and see workflow-specific information. Figure 7.13 shows that the work item is in error status because our method `BACKGROUND_METHOD` did nothing except throw an exception. The information box in Figure 7.13 shows the result of our implementation.



Display work item

Work item information

ID	000001067051
Type	Background Step
Title	Creation of Vendor 100000064
Status	Error
Priority	5 5 Medium
Agent	Workflow-System

Deadline Data

	Creation		Start		End Date
Current	04.11.2010	21:17:59	04.11.2010	21:18:02	
Requested					
Latest					

Description
No description

Error triggered in business class 'BACKGROUND_METHOD'

E75 (5) 800 | ec7server5 | INS

Figure 7.13 Work Item Information

Now we've achieved the goals of this chapter: We linked a workflow event to a particular business action (vendor creation) and managed to execute a newly defined method in background mode.

7.3 Summary

While you might think that the process we walked you through in this chapter seemed cumbersome (and don't forget that we omitted a lot of details to make the picture relatively simple), this process gives you another degree of design freedom. Using standard and custom-defined workflow events, you can further expand potential design capabilities of the system, including sophisticated, automatic, or interactive document chains; B2B and A2A scenarios; and complex data distribution models that can be developed by using SAP Business Workflow functionality.

The Author

Sergey Korolev graduated from Moscow State University in the former USSR, and has worked as a software engineer since 1984, exploring many different flavors of software development: system programming for mainframes, proprietary graphical user interfaces, image recognition software, and business software. In 1999, he started his SAP career as an ABAP and Workflow developer and consultant. Since 2007, he has been working as a freelancer providing services for various international clients, including SAP itself.

Being a cappuccino addict, he often spends his working time in coffeeshops (provided the client offers WIFI access). When out of town, he particularly enjoys visiting small local jazz clubs.

Index

Service Pages

The following sections contain notes on how you can contact us.

Praise and Criticism

We hope that you enjoyed reading this book. If it met your expectations, please do recommend it, for example, by writing a review on http://www.sap-press.com. If you think there is room for improvement, please get in touch with the editor of the book: kelly.harris@galileo-press.com. We welcome every suggestion for improvement but, of course, also any praise!

You can also navigate to our web catalog page for this book to submit feedback or share your reading experience via Facebook, Google+, Twitter, email, or by writing a book review. Simply follow this link: http://www.sap-press.com/H3171.

Supplements

Supplements (sample code, exercise materials, lists, and so on) are provided in your online library and on the web catalog page for this book. You can directly navigate to this page using the following link: http://www.sap-press.com/H3171. Should we learn about typos that alter the meaning or content errors, we will provide a list with corrections there, too.

Technical Issues

If you experience technical issues with your e-book or e-book account at SAP PRESS, please feel free to contact our reader service: customer@sap-press.com.

Legal Notes