SAP PRESS

SAP® Essentials

ABAP® Development for Sales and Distribution in SAP®: Exits, BAdIs, and Enhancements

- Learn how to enhance and modify Sales and Distribution in SAP
- Benefit from real-life examples and detailed code listings
- Determine which enhancement technique is right for which situation

Michael Koch

Galileo Press

SAP® Essentials

Expert SAP knowledge for your day-to-day work

Whether you wish to expand your SAP knowledge, deepen it, or master a use case, SAP Essentials provide you with targeted expert knowledge that helps support you in your day-to-day work. To the point, detailed, and ready to use.

SAP PRESS is a joint initiative of SAP and Galileo Press. The know-how offered by SAP specialists combined with the expertise of the Galileo Press publishing house offers the reader expert books in the field. SAP PRESS features first-hand information and expert advice, and provides useful skills for professional decision-making.

SAP PRESS offers a variety of books on technical and business related topics for the SAP user. For further information, please visit our website: http://www.sap-press.com.

Dirk Herzog
ABAP Development for SAP NetWeaver BW (3rd Edition)
2012, 280 pp. (hardcover)
978-1-59229-424-4

Jürgen Schwaninger
ABAP Development for Materials Management in SAP
2011, 270 pp. (hardcover)
978-1-59229-373-5

Sergey Korolev
ABAP Development for Financial Accounting
2011, 252 pp. (hardcover)
978-1-59229-370-4

Rich Heilman, Thomas Jung
Next Generation ABAP Development (2nd Edition)
2011, 735 pp. (hardcover)
978-1-59229-352-0

Michael Koch

ABAP® Development for Sales and Distribution in SAP®

Exits, BAdIs, and Enhancements

Galileo Press

Bonn • Boston

Dear Reader,

One size fits all? Not in today's business climate! With this book, you have the opportunity to walk along with two ABAP developers, Christine and Sean, as they work together to find custom enhancements that meet a business's SD custom requirements. Thanks to the integration of practical instruction and a fictional business case, you'll feel as if an ABAP expert is standing by your side as you explore each real-world scenario. Not only will you understand how to find the best way to customize a system, you'll also understand how to piece together different experience levels and knowledge bases to get the most out of your team.

I had the pleasure of working with Michael Koch on his first (and hopefully not last) book. His enthusiasm for the project was extremely catching, and he even came up with the unique idea of adding an illustration to the book! He constantly impressed me as he juggled his consulting jobs (which of course give him the valuable experiences and expertise we're always looking for to write such a book!), with writing and meeting the deadlines.

We at SAP PRESS are always eager to hear your opinion. What do you think about *ABAP Development for Sales and Distribution in SAP*? As your comments and suggestions are our most useful tools to help us make our books the best they can be, we encourage you to visit our website at *www.sap-press.com* and share your feedback.

Thank you for purchasing a book from SAP PRESS!

Laura Korslund
Editor, SAP PRESS

Galileo Press
Boston, MA

publishing@galileo-press.com
http://www.sap-press.com

Notes on Usage

This e-book is **protected by copyright**. By purchasing this e-book, you have agreed to accept and adhere to the copyrights. You are entitled to use this e-book for personal purposes. You may print and copy it, too, but also only for personal use. Sharing an electronic or printed copy with others, however, is not permitted, neither as a whole nor in parts. Of course, making them available on the Internet or in a company network is illegal as well.

For detailed and legally binding usage conditions, please refer to the section Legal Notes.

This e-book copy contains a **digital watermark**, a signature that indicates which person may use this copy:

Imprint

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

Editor Laura Korslund
Acquisitions Editor Kelly Grace Harris
Copyeditor Julie McNamee
Cover Design Graham Geary
Photo Credit iStockphoto.com/Ceneri
Production E-Book Kelly O'Callaghan
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:
Koch, Michael.
ABAP development for sales and distribution in SAP : exits, BADIs, and enhancements / Michael Koch. — 1st ed.
p. cm.
ISBN 978-1-59229-420-6 — ISBN 1-59229-420-0
1. SAP ERP. 2. Sales management—Computer programs. 3. Marketing—Management—Computer programs. I. Title.
HF5438.4.K634 2013
658.800285'53—dc23
2012027202

ISBN 978-1-59229-420-6 (print)
ISBN 978-1-59229-603-3 (e-book)
ISBN 978-1-59229-604-0 (print and e-book)

1st edition 2012

Contents

PART II Enhancements in Delivery Processing

PART III Enhancements in Billing

PART IV Conclusion

Preface

Today, SAP ERP counts for more than 65,000 installations worldwide. Experts estimate that 90-95% of these installations use Sales and Distribution (SD), so we can assume that roughly 60,000 businesses around the globe use it. In other words, SAP SD is a true ERP success story.

There are many reasons for this success, but one that repeatedly ranks the highest is the level of configuration and customization that can be achieved within SD. Both sales and distribution form integral parts of most modern businesses. Therefore, the ability to mold the system to key business processes—where deemed necessary—can be paramount. While configuration enables a business to add system settings within the boundaries of the programming code defined by SAP, customization goes deeper and requires additional code in ABAP, SAP's programming language.

However, customization of standard software is often frowned upon. Experts perceive any deviation from the standard as a potential heachache when it comes to future upgrades. This leaves customers in a dilemma, as shown in the following light-hearted illustration.

A Customer's Enhancement Dilemma

Over the past 20 years, however, SAP has worked hard to allow its customers to slightly adjust key processes if necessary, yet still remain on the upgrade path. This is enabled by different kinds of technologies such as exits, business add-ins (BAdIs), and the more recent introduction of the Enhancement Framework. These technologies support ABAP code customizations and play a major role when modifying SAP's standard ABAP delivery.

About This Book

This book is predominantly based on my own consulting experience and tasks I have worked on over the years. I have tried to include helpful, real-world examples of varying difficulty and complexity.

Chapter 1 provides background information and comparisons of the different enhancement techniques used in this book. After this, Chapter 2 through Chapter 11 will show you how to do the following:

- Add new field validations for a sales order.
- Capture additional data in custom fields within Transactions VA01/02.
- Automatically create a SAP CRM activity after billing cancellation.
- Filter pricing data within a SAP standard web service for order confirmation.
- Add custom fields within SD pricing.
- Set a delivery block from new validation within a code enhancement.
- Use a BAdI to keep track of delivery KPIs within a little custom reporting system.
- Enhance a standard delivery monitor report to include custom KPI data.
- Split invoice items into separate documents by using a VOFM routine.
- Change the reference field and numbering range while a new invoice is created.

Chapter 12 then summarizes the techniques and enhancement methods. Lastly, Chapter 13 concludes with an outlook on ERP enhancements and their role in the future.

Within the appendices you will find an overview of all SD-related exits and BAdIs, as well as one of the longer coding examples (which can also be found online).

Who Benefits from This Book

This book will be beneficial to the following groups of people:

- ABAP developers (beginner, intermediate)
- SD consultants (all levels of experience)
- Support consultants (all levels of experience)
- SAP architects (all levels of experience)
- Project team members who require a deeper understanding of how to enhance standard SAP ERP

System Prerequisites

All examples in this book have been created and tested in a SAP ECC 6.06 system on SAP NetWeaver AS ABAP 7.31. However, none of the shown examples require such a recent release level (as of July 2012) and any SAP ECC 6.0 and SAP NetWeaver AS ABAP 7.x and above will suffice. If you work on earlier releases, some of the newer enhancement techniques such as BAdIs or the Enhancement Framework might not be available.

If in doubt, refer to the overviews in Chapter 1 to check whether an enhancement point or BAdI is available within the application release you are using.

Fictional Project

Software development doesn't take place in a vacuum, and most of the time there are underlying pains and purposes for it. The same goes for any enhancements to SAP systems, which invariably often stem from business requirements. Therefore, simply showcasing the most common exits and enhancements with instructions on how to implement them doesn't reach far enough. It's important to tell a business story. It's equally important for IT professionals to realize this concept and have a grasp of the business story behind a proposed change.

You'll find that this book mirrors the "real world" as much as possible. Throughout the course of this book, I have used stories and anecdotes from a fictional project to better illustrate the discussed topics.

You'll come across a junior ABAP developer and a senior SD consultant, who will accompany you on your learning journey. So let's meet the business, Sean, and Christine…

The Byrell Corporation

Byrell Corporation has been trading for more than 50 years as a logistics business with depots and warehouses in various locations on the British Isles. The company has been growing steadily over the past 40 years, in particular in the past 20, which among other things led to the board's decision to implement SAP R/3 and replace the various legacy systems previously in place. Apart from the traditional haulage business, Byrell also offers storage and warehousing solutions, based on its SAP systems and custom developments.

In 2004, Byrell decided to add SAP CRM alongside its SAP ERP system to offer its customers a better integration of opportunity management and quotations.

Byrell envisages an upgrade to the latest SAP ECC Enhancement Pack in the near future and has already instructed its SAP team to prepare for this.

Sean, Senior SD Consultant

Sean has been with the Byrell Corporation for more than 30 years now. He started with Byrell when he was in his early twenties, working as a packer in the warehouse. Over the years, his bosses recognized his technical aptitude and gave him the option to cross over into IT, where he played a pivotal role in the implementation of SAP R/3 3.1i in 1995. Since then, he has worked on other major upgrades, all the way to the current SAP ECC 6.0 release. Apart from his upgrade project work, Sean deals with smaller SD support and improvements, liaising between business and IT.

Sean acknowledges that SAP ERP has changed a lot since its first implementation at Byrell back in 1995. He is well versed with how to enhance the SD module using user exits, but newer enhancement methods such as business add-ins often raise a few questions. He also realizes that over the same period of time, the ABAP language has become more sophisticated. A lot of the code that he looks at during his support work is not always understandable for him. For some time now, he has vowed to refresh his ABAP knowledge, but work pressures and deadlines have prevented this so far.

Sean's role is to understand the functional side of new development requirements, write specifications for Christine, and apply any necessary SD configuration, if needed.

Christine, Junior ABAP Developer

Christine started at Byrell last year as part of the company's graduate program. She came straight from the university, where she studied computing. Christine's first real programming language was Java at college, which introduced her to object-oriented (OO) programming right from the start. During some of her college and university assignments, she also picked up XHTML, Cascading Style Sheets (CSS), and a little bit of JavaScript. Christine loves coding and spends a lot of her spare time in front of computers.

Last year after Christine started at Byrell, she was sent to a couple of introductory ABAP training courses, which taught her all about report programming, screen (Dynpro) development, and ABAP OO. She was delighted that there was such a thing as object orientation in ABAP and has been using her new OO skills on many occasions. Recently Christine has also attended a course on the SAP Enhancement Framework, because her skills will be required to help Sean with some of the SD development requests coming out of support issues. In addition, there is also an SAP ECC upgrade project on the horizon, which might require some of Christine's time.

Christine's role is to provide technical consultancy input to new development requirements and to create ABAP code per Sean's functional specifications.

Acknowledgments

Working on this book has been rewarding, insightful and taxing, but most of all it has been an extreme honor.

First of all, a big thank you goes out to the SAP Mentor initiative, which allowed me to reach out and receive assistance from such great individuals as Mark "wolf pack leader" Finnern, Matthias Steiner, and Thomas Jung, who helped me to get in touch with the right people within SAP for this project. Most of all, I would like to thank Martin Lang of SAP, who was instrumental in providing access to systems where I could code and test the solutions for this book.

Moreover, this book would have never come to fruition without the initiative, hard work, determination, trust, and patience of Kelly Harris and Laura Korslund of SAP PRESS. Writing a book such as this is bound to come with a few ups and downs. Kelly and Laura's experience and assistance certainly helped in coping with some of them.

I would also like to thank friends and colleagues, namely Peter Richardson, Zoe Gill, Arpit Oberoi, Ben McGrail, Stefan Karaivanov, Thomas Otter, and Nick Watkins, whose feedback, professional advice, criticism, and ideas were essential during the entire duration of writing this book. In particular I would like to thank Cath Laursen and Arran McMillan for their assistance with questions around SAP CRM.

Special thanks also go out to Stuart Trotter of Rockpool Children's Books (*http://rockpoolchildrensbooks.com*), who helped me to transfer my idea for the "SAP SD" illustration from my brain and onto paper and screen.

Finally, I want to express my love and gratefulness to my wife Sally and our beautiful daughters Milly and Hannah, who have all been helpful and supportive throughout the realization of this project. Sally, Milly, and Hannah—a lot of this book is also due to you! Thank you for giving me your understanding and patience to fulfill this dream.

The Enhancement and Switch Framework, along with exits and routines, are used to enhance SAP Sales and Distribution systems. In this chapter, you'll learn the differences between each of the enhancement techniques and where to find them.

1 Introduction to Enhancements in Sales and Distribution

Before introducing the different ways you can enhance the existing Sales and Distribution (SD) functionality with ABAP code, you need to understand that any customization applied to the SAP standard code should always be the last resort and the end result of vigorous contemplation. Why?

SD is standard software, like many enterprise resource planning (ERP) solutions. Most businesses introduced SD to reap the benefits of standard software. Here is a non-exhaustive list of the typical benefits a business can gain from using SD:

- Stability
- Cost efficiency
- Upgradability
- Leveraging best practices

In other words, by creating custom code enhancements to your SD component, you are introducing new software objects that might have a detrimental effect on some or all of the benefits just listed. Therefore, it's important to analyze, understand, and communicate the potential impact of any code customization.

Before undertaking any custom enhancements using ABAP code, developers need to consider the following five important questions:

1. Could SAP standard functionality solve your problem?
2. Have you searched SAP Marketplace (*http://service.sap.com*) for SAP Notes, which might help you with your problem?

3. Can the same be achieved using Customizing?
4. Is there already an existing customization object, in which the envisaged change could be incorporated?
5. Have you discussed the problem with functional consultants and other team members?

Important

You should only implement code customizations if you're satisfied that the answers to these questions do not suggest an alternative solution.

Routines, user exits, and customer exits (in some parts) leave a fair amount of responsibility with the customer. User exits, for example, have no defined interfaces, meaning all fields and tables are available to the developer. Therefore, any incorrect use of the available fields and tables can lead to undesired results.

It's important for developers to explore alternatives before embarking on the implementation of any customer-defined routine or user exit. Developers must keep the checkpoints in mind that were laid out in the introduction of this chapter before proceeding with implementing an enhancement.

Tips & Tricks

For further information, refer to SAP Note 381348, "Using User Exit, Customer Exit, VOFM in SD."

1.1 Enhancing Standard SD Functionality

SAP offers more than one method or technology to customize and enhance its off-the-shelf SD functionality. To give you an overview, Figure 1.1 shows a mind map of these methods and technologies. A flag indicates that the particular method is covered in this book.

As shown in the mind map, the main emphasis of this book is on exits, the Enhancement Framework, and customer-defined routines. This is not to say that all other techniques shown (Customizing, personalization, and business transaction events) are of no importance. However, in terms of custom code enhancements, these methods are not as relevant as exits, BAdIs, and customer-defined routines. Also,

there are situations where modifications cannot be avoided (see Chapter 2 and Chapter 13 for details).

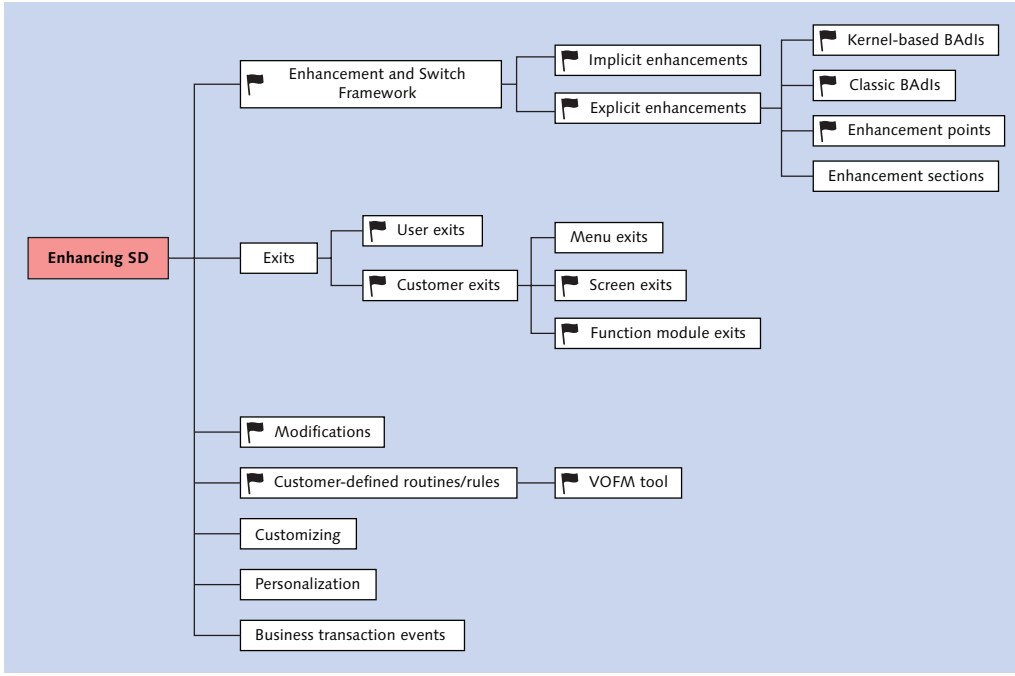


Figure 1.1 SD Enhancements Mind Map

In the following sections, we introduce exits, the Enhancement Framework, and customer-defined routines in more detail. This chapter will conclude with a comparison of enhancement techniques, which should help you make the right enhancement choice in your own projects.

1.1.1 Exits

Up to SAP release 4.6, exits were the only way to truly enhance SAP standard coding. (From release 4.6A onward, business add-ins (BAdIs) were introduced, which are discussed in Chapter 2, Section 2.3.)

SAP created exits when it developed the standard code and put these into places where customers need more flexibility. Exits are empty or inactive when delivered and can be completed by customers as needed. Exits provide the following advantages, especially when compared to modifications:

- Developed objects do not affect standard coding and are held separately, in their own customer namespace.
- During an upgrade project, most exits remain and do not affect the upgraded system (there are some exceptions, see the comparison tables in Section 1.4 of this chapter for details).

In the following sections, we'll discuss the different types of exits and distinguish them by explaining their implementation and features.

More often than not, user exits and customer exits are mixed up or even regarded as the same thing. In fact, the differences between them are more evolutionary, yet important to understand.

User Exits

User exits are preplanned exit routines in standard SD coding, which means SAP has intentionally left them empty within the standard code. User exits can only be found within the SD component. Historically, when regarded from a technical perspective, user exits required modifications. However, this changed with the emergence of the Enhancement Framework, which we'll discuss in Section 1.3. User exits are sometimes also referred to as *form exits* because they are actually form subroutines.

Finding and Accessing User Exits

As an example, let's find a user exit that is called just before a sales document is saved. There are two different ways to find user exits:

- **Via the Implementation Guide (IMG)**
  Go to Sales And Distribution • System Modifications • User Exits • User Exits In Sales • User Exits In Sales Document Processing. Click on the node to open the ABAP Editor. In the Program field, enter "MV45AFZZ" as the name of the include program that holds the crucial user exit in this case. Click on the Display button to view the different form exit routines (see Figure 1.2).
- **Via the assigned package**
  You can also find all user exits by displaying all include programs within package VMOD. Run Transaction SE80, choose Package from the dropdown box, enter "VMOD" as the name of the package, and press Enter. If you expand the Includes folder, you can find all SD user exit include programs holding the different user exits (see Figure 1.3).

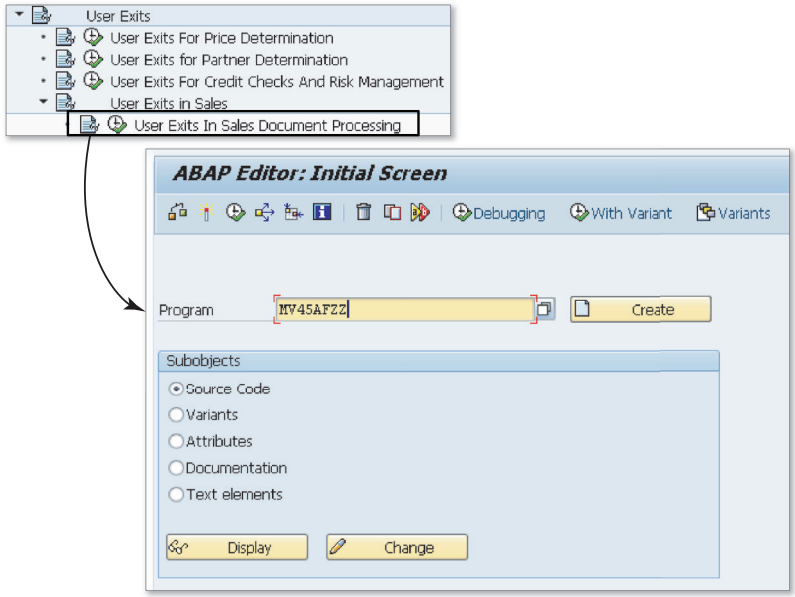


Figure 1.2 Finding User Exits with the IMG

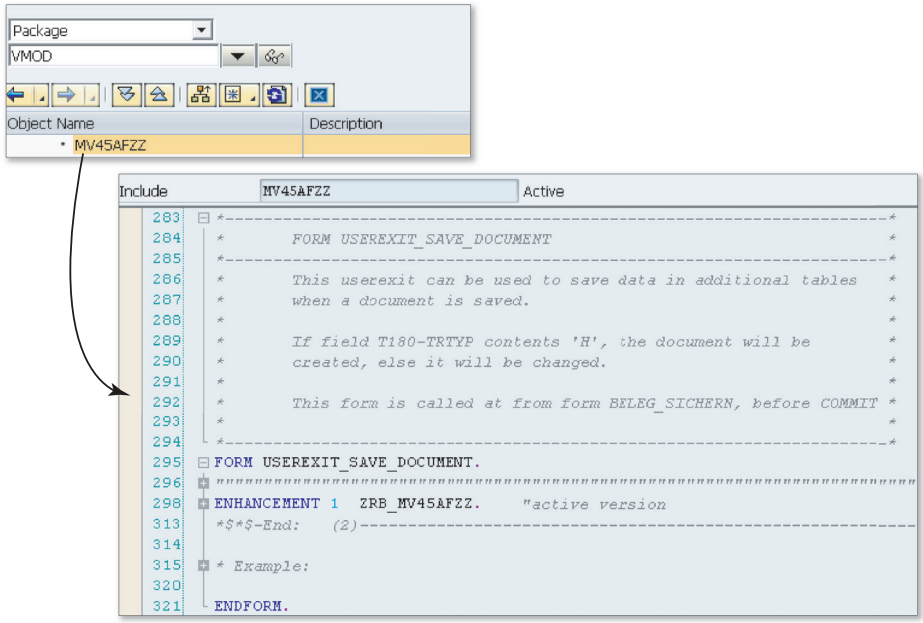


Figure 1.3 Include MV45AFZZ with User Exit USEREXIT_SAVE_DOCUMENT

Note

User exits are identified by the prefix USEREXIT.

In the past, the actual user exit coding had to be done using a modification, and it was advisable to use include programs that were assigned to a module pool within the customer's namespace (beginning with Y* or Z*). Within this include program is where you could code for any changes that you wanted to make to the system.

Example

This listing shows an example of a user exit using the include technique, which points to a program with the customer namespace—in this case ZZ_USEREXIT_SAVE_DOCUMENT.

```
FORM USEREXIT_SAVE_DOCUMENT.
  INCLUDE ZZ_USEREXIT_SAVE_DOCUMENT.
ENDFORM.
```

Note

These include programs are always processed during the program flow, and there is no built-in way to switch them on or off.

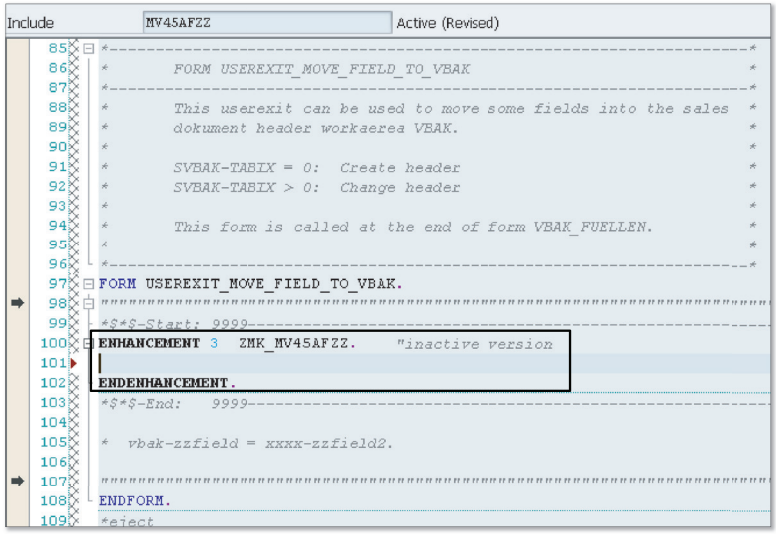


Figure 1.4 Implicit Enhancement Placed within User Exit USEREXIT_MOVE_FIELD_TO_VBAK

However, with the introduction of the SAP Enhancement Framework, it's possible to fill user exits with modification-free, switchable code enhancements. Figure 1.4 shows an implicit enhancement placed into form routine `USEREXIT_MOVE_FIELD_TO_VBAK` using the Enhancement Framework.

Note

SAP's Enhancement Framework and implicit enhancements are introduced in more detail in Section 1.3.

Customer Exits

SAP Online Help describes customer exits as follows:

> *The enhancement concept allows you to add your own functionality to SAP's standard business applications without having to modify the original applications. SAP creates customer exits for specific programs, screens, and menus within standard applications. These exits do not contain any functionality. Instead, the customer exits act as hooks. You can hang your own add-on functionality onto these hooks.*

Customer exits are part of SAP's enhancement concept, which offers the following advantages:

- They don't affect standard code.
- Because customer exits are created within the customer's namespace, no standard code is modified.
- They don't affect software upgrades.
- Strict naming conventions prevent clashing development objects (customer objects) or even loss of enhancements when the SAP standard code is upgraded.
- They can be switched on or off.
- They have defined exit interfaces.

Using Transaction CMOD, customers can switch individual customer exits on or off, which can be very helpful if a specific enhancement is causing problems. A green light indicates the exit is active. Figure 1.5 shows a Delivery Monitor user exit with two function module exits assigned to it.

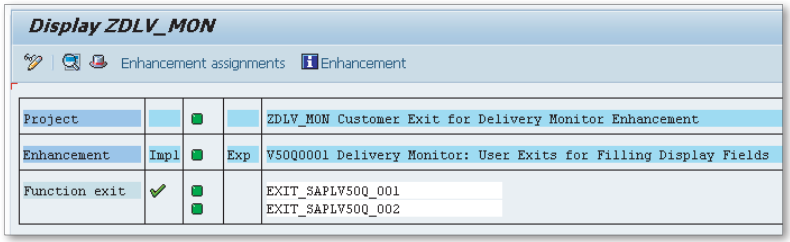


Figure 1.5 Customer Exit V50Q0001 for Delivery Monitor with Two Active Function Module Exits

Tips & Tricks

The upcoming "Finding and Accessing Customer Exits" section shows how to find and browse customer exits in more detail.

Customer exits wrap the enhancement functionality, which prevents any other changes to global variables. Defined interfaces make a significant change to enhancements and their overall stability (see Figure 1.6).

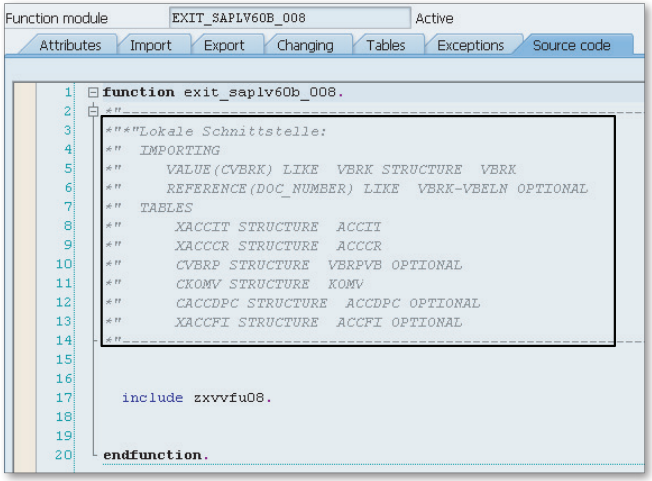


Figure 1.6 A Customer Exit with Strict Input/Output Parameters

Note

Although the function module exit was created by SAP, the include `ZXVVFU08` resides in the customer namespace.

Types of Customer Exits

There are three types of customer exits:

- **Screen exits**
  Screen exits add fields to screens in applications. SAP creates screen exits by placing special subscreen areas on a standard screen and calling a customer subscreen from the standard screen's flow logic.
- **Menu exits**
  Menu exits add items to the dropdown menus in standard SAP applications. You can use these menu items to call up your own screens or to trigger entire add-on applications.

  SAP creates menu exits by defining special menu items in the Menu Painter. These special entries have function codes that begin with "+" (a plus sign). You specify the menu item's text when activating the item in an add-on project.

  Figure 1.7 shows an assignment of a menu exit within sales activities to view contact history.

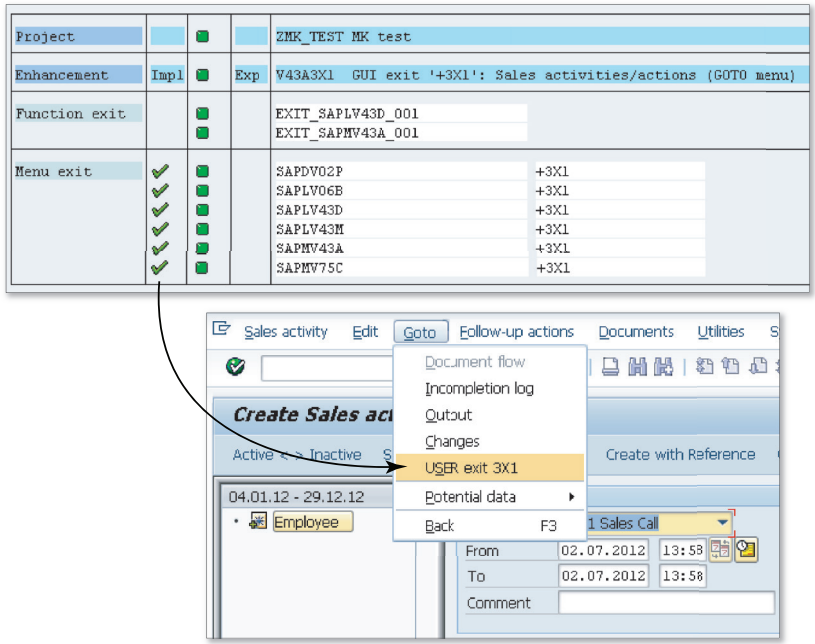


Figure 1.7 Customer Menu Exit V43A3X1 Implemented and Showing as a Menu Item in Transaction CV01N (Create Sales Activity)

Note

Keep in mind that defining the menu exit by itself is not enough. The screen menu event also has to be evaluated using the corresponding function module exits of exit `V43A3X1`. (More details on screen exits can be found in Chapter 3.)

- **Function module exits**

  Function module exits add functions to applications. They play a role in both menu and screen exits. For example, when adding a new menu item to a standard dropdown menu, a function module exit is used to define the actions that should take place after your menu is activated. Function module exits also control the data flow between standard programs and screen exit fields.

  SAP's application developers created function module exits by writing calls to customer functions in the source code of standard programs. These calls have the syntax `CALL CUSTOMER-FUNCTION `nnn`` (where `` `nnn` `` is a three-digit number).

  The `CALL CUSTOMER-FUNCTION` statement is only executed if the enhancement project was activated. Also, the INCLUDE program within the customer function module needs to be implemented. Multiple calls of the same function module are all activated at the same time.

  Figure 1.8 shows a simplified code layout for a function module exit. It comes in two parts, a `PROGRAM` part and a `FUNCTION-POOL`. Exit function modules all belong to function groups whose names begin with "X".

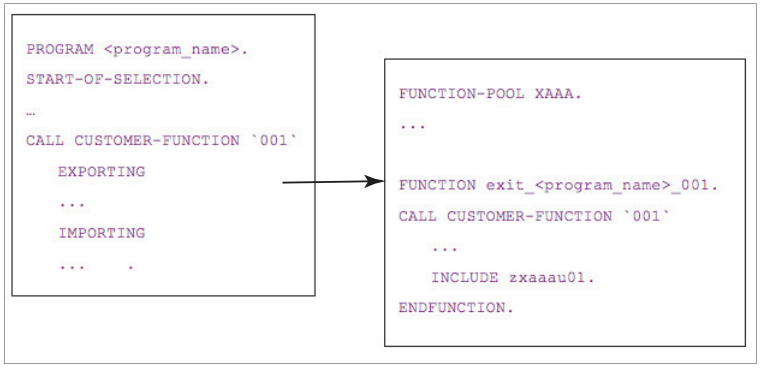


Figure 1.8 CALL CUSTOMER FUNCTION Calls Function Module exit_<program_name>_001 within INCLUDE zxaaau01

The following naming convention applies to exit function modules (each part is separated by an underscore):

- **Prefix:** EXIT
- **Name:** Name of the program that calls the function module
- **Suffix:** Three-digit number

An example of a correctly named exit function module is `EXIT_SAPLV43D_001`, for instance.

Finding and Accessing Customer Exits

You can find and access customer exits in three ways:

- **Find customer exits by searching for "call customer-function"**
  If you know the SAP transaction you want to check for a potential customer exit implementation, display the STATUS screen by selecting SYSTEM • STATUS. Double-click on PROGRAM (SCREEN) SAPMV45A to go to the module pool screen (see Figure 1.9). This shows an example from Transaction VA01 (Sales Order Create).

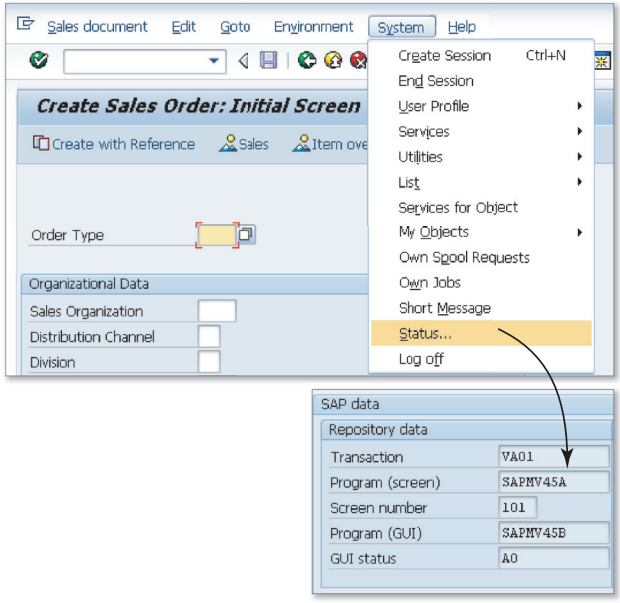


Figure 1.9 Navigating to ABAP SAPMV45A from the Status Menu Option

Press Ctrl + F to bring up the search box, and search for "call customer-function" in the code of the main ABAP program. The results of the global search will show all occurrences of customer exit calls (see Figure 1.10).

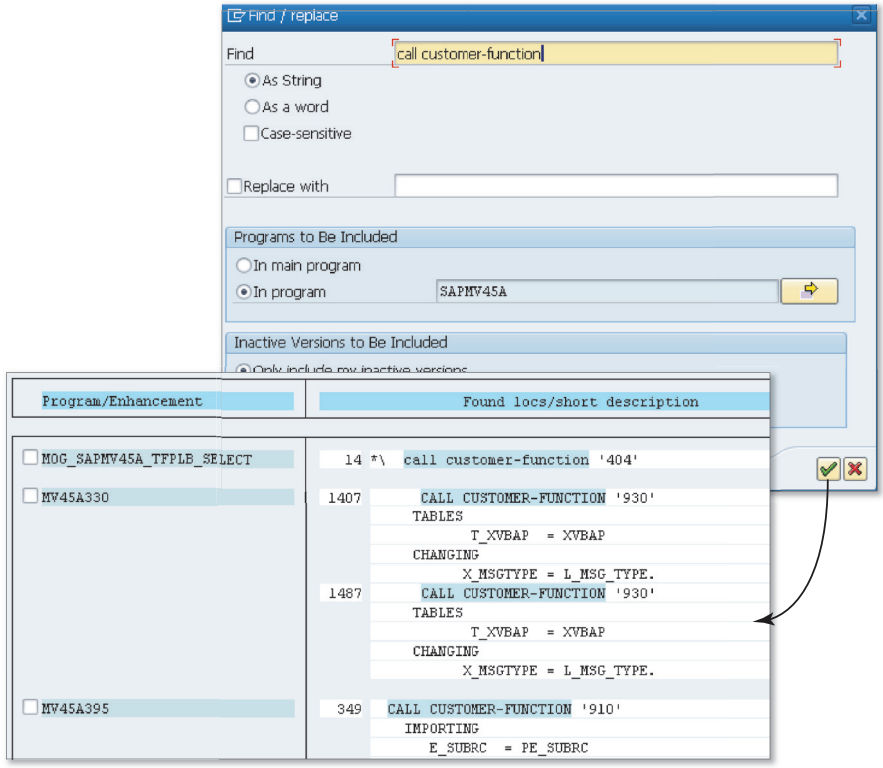


Figure 1.10 Searching for String "call customer-function" within the ABAP Program

- **Find customer exits from Transaction SMOD**

  This is a good method of searching if you know the package name of the customer exit you are looking for. Transaction SMOD is the central point for SAP customer exits, enhancements, and documentation. Its menu system includes a search function that can also be used to find customer exits.

  In Transaction SMOD, select Utilities • Find. In the following selection screen, enter "VF" in the Package field and then click on Execute (F8), which will show you a list of customer exits (see Figure 1.11).

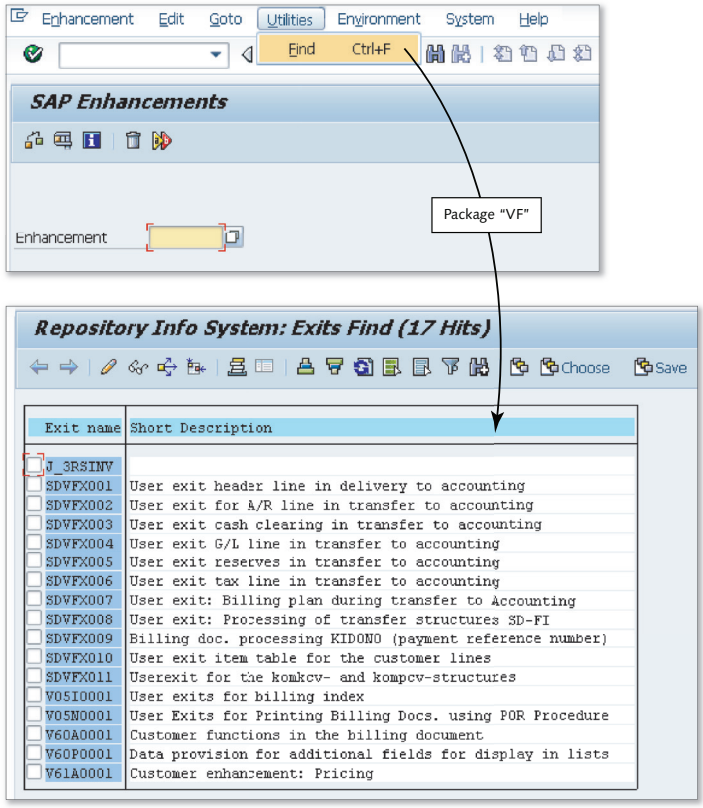


Figure 1.11 Finding Customer Exits via Transaction SMOD

- **Find customer exits via the Repository Infosystem (SE80)**
  Similarly, call Transaction SE80, and select the Repository Infosystem button on the top left. Open the Enhancements and Customer Exits folders. If you double-click on the Enhancements tag, you can search using the selection screen on the right.

1.1.2 Projects

Transaction CMOD is used to collate all customer enhancements into enhancement projects, which act like an enhancement envelope for a set of specific enhancement tasks. At this point, it's important to note that user exits are not part of this enhancement paradigm and therefore aren't required to be organized under a project umbrella.

Within standard code, only customer exits that were created by SAP can be enabled and used with Transaction CMOD. Moreover, it isn't possible to have different versions of exits to allow for filtering, for example. However, this feature was introduced with the Enhancement Framework (see Section 1.3) and can be leveraged by BAdIs and enhancement spots.

For a more detailed comparison of the different enhancement techniques, see Section 1.4.

1.2 Customer Routines

Customer routines represent yet another way of enhancing standard code, but there are some differences when comparing them to customer exits.

Similar to customer exits, SAP developers defined these routines and placed them at previously planned locations in the code.

Some facts about routines include the following:

- They can be amended using Transaction SE38 and can therefore have defined import and export parameters.
- They are not held within projects or components.
- They are switchable.
- Transaction VOFM is a central tool to search, maintain, and manage routines.
- From a technical point-of-view, routines require modifications or implicit enhancements.

There is something to be said about the ease with which a routine can be amended; you can easily change them using the standard ABAP Editor (SE38). In addition, routines can be activated and deactivated within a central tool. Also, they do not have to be defined within a project, which makes implementation easier. On the other hand, when analyzed from a technical angle, routines require (just like user exits) modifications or implicit enhancements. Enhancements are the technique of choice for new routine developments.

We'll go over the different VOFM routine types in the following subsections and also explain where these are maintained.

Transaction VOFM

Transaction VOFM is the central tool to search and manage routines. Here you can browse, implement, and activate all four different types of routines. Figure 1.12 shows an overview of all routine type menu options in Transaction VOFM.

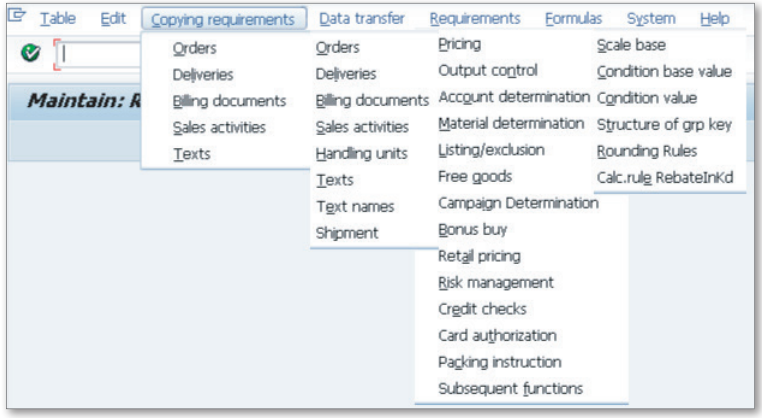


Figure 1.12 Transaction VOFM Dropdown Menus for Routine Types

Copying Requirements

Copying requirements routines check certain requirements as a precondition for the copying process. You use copy requirements to establish whether a quotation should be copied into a standard order or whether text items are transferred during the copying of a delivery into a billing document. Copy requirements are an easy way to implement company-specific business rules to allow or prevent subsequent document creation.

Data Transfer

Data transfer routines define which fields are to be copied over to a new subsequent document. For example, the item type of an order is copied over to the delivery.

Requirements

Requirements routines check whether a condition applies to a document. For example, a discount pricing condition might apply if an order is placed online and therefore has a different order type.

Formulas
Formula routines are an important part of pricing conditions and allow you to perform calculations under certain conditions. For example, a formula can be used to double a discount when an order is placed from a particular sales area.

1.3 The Enhancement and Switch Framework

Similar to customer and user exits, business add-ins (BAdIs) are enhancements to the standard version of the system. SAP's Enhancement Framework aims to extend and replace existing modification and enhancement methods, such as user exits.

The Enhancement Framework is a new approach to enhancing repository elements (development objects, data dictionary elements). Its main purpose is to bring all enhancement concepts under one roof and create a single place for all enhancement requirements. However, this doesn't mean that previously mentioned modification and enhancement concepts are replaced by the Enhancement Framework. In fact, these are still available and supported, but SAP advises customers to use the Enhancement Framework when undertaking new enhancement tasks. Moreover, with the Enhancement Framework, SAP offers a solution for a truly modification-free delivery of enhancements, which can reduce efforts during an upgrade project.

SAP and its partners and customers can use the Enhancement Framework to do the following:

- Enhance existing SAP standard programs
- Add industry or custom solutions to the SAP ERP core without disruption
- Deliver enhancement packages for SAP ERP and the SAP Business Suite

As you can see, the Enhancement Framework adds two additional dimensions here. In addition to enhancing standard objects, it also facilitates the integration of custom solutions as well as delivery of enhancements. These two additional options are often overlooked, yet represent a great opportunity for SAP and its partners to create add-on solutions for SAP ERP and SAP Business Suite core products without affecting upgrades. In this book, however, we will focus on the enhancement of standard objects.

When using the Enhancement Framework, there are basically two enhancement options available: explicit enhancements and implicit enhancements. Let's explore each option in the following subsections.

1.3.1 Explicit Enhancements

Explicit enhancements have been created and inserted into the code by the original developer. Although this may sound similar to preplanned customer exits and routines, discussed earlier in Section 1.1.1, explicit enhancements have one crucial advantage: they can exist on multiple layers. This means that before these enhancements came into existence, only SAP was able to create explicit enhancements. From version 4.6 onwards, SAP also introduced the ability for partners and customers to insert their own exits. In other words, you can create your own enhancement options, which can be very useful when designing a third-party solution on top of SAP ERP core systems. See Figure 1.13 for an overview of a multilevel enhancement landscape, starting with the core development on top, going all the way down to the enhancements on the customer level.

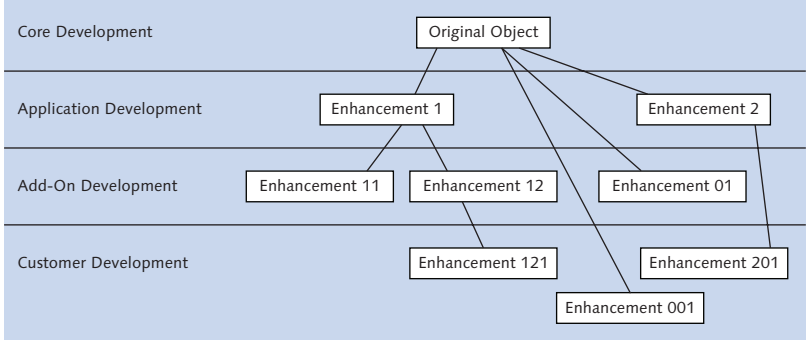


Figure 1.13 Multilevel Enhancement Landscape, Enabled by the Enhancement Framework

Explicit enhancements come in different shapes and sizes, including enhancement points, enhancement sections, and BAdIs.

Enhancement Points

Similar to user exits, enhancement points allow developers to add one or more pieces of custom logic as a code plug-in to standard code. When the program is

generated, all code plug-ins with a switch setting of "on" or "stand-by" are included in the code at this position.

Single enhancement points in ABAP code can be declared like this:

```
ENHANCEMENT-POINT enh_id SPOTS spot1 spot2
```

Every enhancement point has to be given an enhancement ID (`enh_id`) and be assigned to at least one enhancement spot (`spot1`, etc.), which is the administrational unit of an explicit enhancement. Figure 1.14 shows an example of `enhancement point beleg_sichern_10`, called while saving an SD sales document.

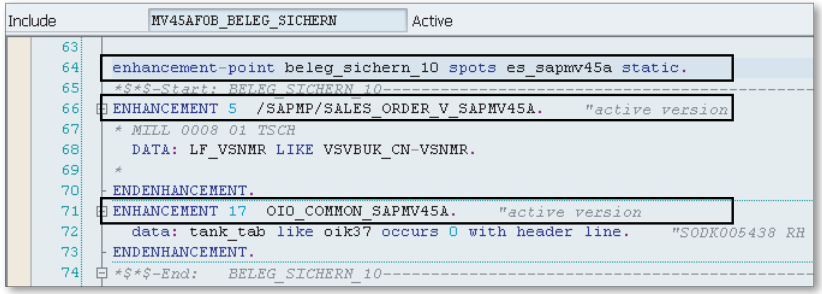


Figure 1.14 Enhancement Point beleg_sichern_10 Featuring Two Active Enhancements

Enhancement Sections

You can also enhance an entire section of code, when it is surrounded by `ENHANCEMENT-SECTION` and `END-ENHANCEMENT-SECTION` commands. If the code is executed, any enhancement implementation for this enhancement section with a switch set to "on" will be performed instead of the code between the `ENHANCEMENT-SECTION` and `END-ENHANCEMENT-SECTION` commands. In the editor, the new enhancement is shown after the `END-ENHANCEMENT-SECTION` command.

The code layout for the declaration of the section is as follows:

```
ENHANCEMENT-SECTION enh_id SPOTS spot1 spot2 ...
...
END-ENHANCEMENT-SECTION.
```

Figure 1.15 shows an example of `ENHANCEMENT OIO_COMMON_SAPMV45A`, which is performed instead of `ENHANCEMENT-SECTION beleg_sichern_21`.

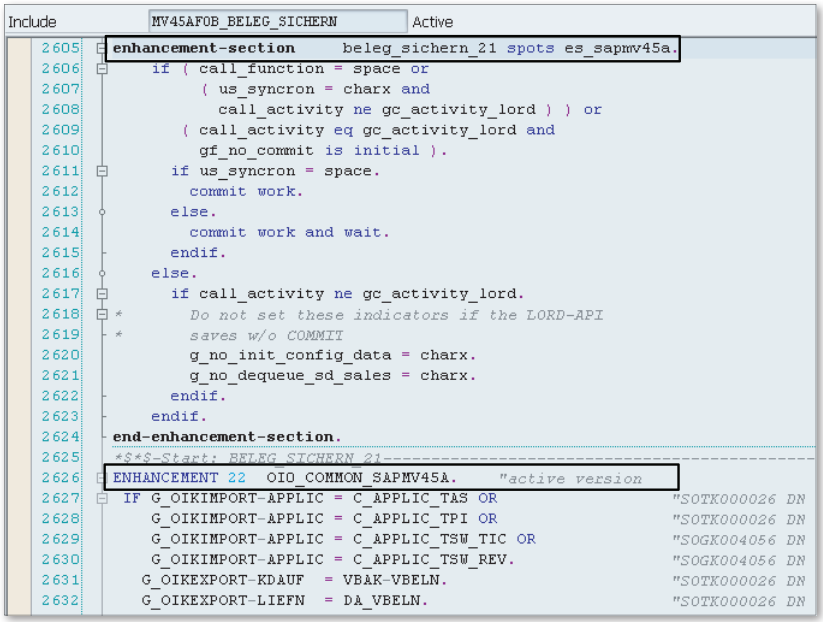


Figure 1.15 Enhancement Section beleg_sichern_21 with Replacing Enhancement

Just like the enhancement spots, every enhancement section also has to be given an enhancement ID (`enh_id`) and be assigned to at least one enhancement spot (`spot1`, etc.).

Business Add-Ins (BAdIs)

BAdIs are probably the most commonly known explicit enhancements. They were introduced in SAP Basis release 4.6 to replace function module exits. From SAP NetWeaver 7.0 onward, BAdIs form part of the Enhancement Framework.

When searching for BAdIs, you'll come across classic BAdIs (introduced before SAP NetWeaver 7.0) and kernel-based BAdIs (introduced after SAP NetWeaver 7.0). Kernel-based BAdIs are an evolutionary step forward from classic BAdIs. They are fully integrated into the Enhancement Framework and perform better than classic BAdIs.

Note

Refer to SAP online documentation for further information on this topic, especially migrating classic BAdIs to kernel-based BAdIs.

BAdIs act like "hooks" for development objects within code, and they come in two parts: definition and implementation. The definition contains a class interface and methods. See Figure 1.16 for an example of BAdI definition `BADI_SD_SALES`, which also displays some of the BAdI methods (`DELETE_DOCUMENT`, `READ_DOCUMENT`, etc.).

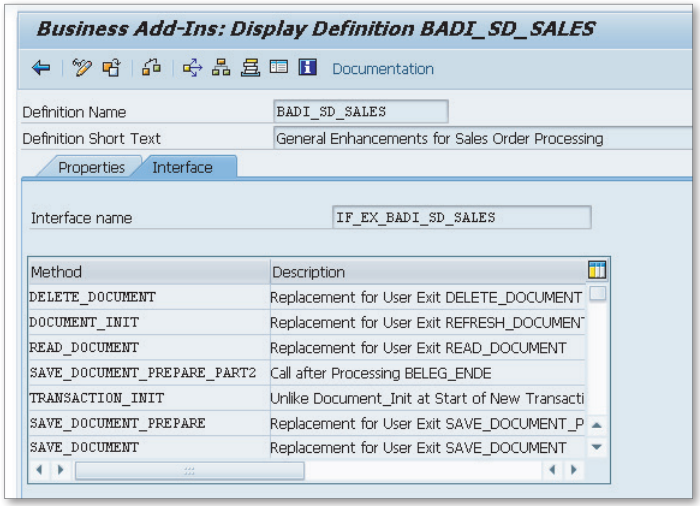


Figure 1.16 Example of a BAdI Definition: BADI_SD_SALES

A BAdI implementation uses the defined methods and holds the enhancement program code. You can also assign a filter to a BAdI implementation to facilitate different enhancement requirements for countries or sales organizations.

1.3.2 Implicit Enhancements

Implicit enhancements give developers the ability to create enhancements at the beginning and end of most code sections. Here, *implicit* means enhancements come as predefined; that is, more or less already existing in code. They therefore have no enhancements spots assigned to them.

On the whole, implicit enhancements can be applied in the following places, according to SAP Online Help:

- At the end of an include statement. There are some restrictions; for example, they can't be applied at the end of a method include.
- At the end of a `PUBLIC-SECTION`, `PROTECTED-SECTION`, `PRIVATE-SECTION` of a class.

- At the end of the implementation part of a class (before the ENDCLASS, which belongs to CLASS … IMPLEMENTATION).
- At the end of an interface definition (before the ENDINTERFACE).
- At the end of a structure definition (before TYPES END OF, DATA END OF, CONSTANTS END OF, and STATICS END OF).
- At the beginning and end of a procedure (FORM, FUNCTION, METHOD), that is, after commands FORM, FUNCTION, and METHOD, and before statements ENDFORM, ENDFUNCTION, and ENDMETHOD.
- At the end of the CHANGING-, IMPORTING-, EXPORTING-parameter list of a method. These enhancement options are located in the middle of a statement.
- Parameter interfaces of function modules, which can be enhanced with parameters.
- Attributes and parameter interfaces of global classes, which can be enhanced with attributes or parameters.

Figure 1.17 shows an example of an implicit enhancement within user exit USEREXIT_FILL_VBRK_VBRP within include program RV60AFZC. Note that the code between the ENHANCEMENT and ENDENHANCEMENT commands was inserted without registering the SAP include and modifying it. This is because implicit enhancements are automatically delivered at the beginning and end of ABAP forms, for example.

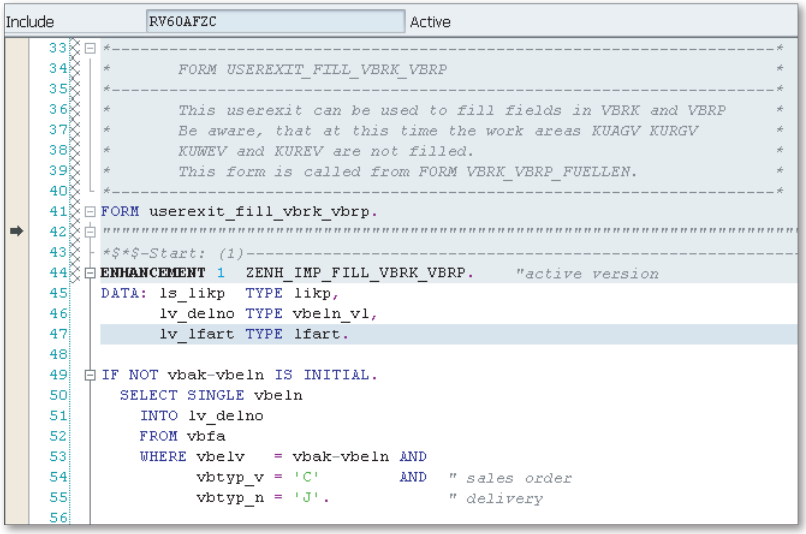


Figure 1.17 Example of an Implicit Enhancement in Include RV60AFZC

Several chapters in Part 2 of this book will demonstrate in more detail how to create implicit enhancements.

1.3.3 The Switch Framework

Now that we've covered the Enhancement Framework, let's briefly talk about the Switch Framework and how it is related to the Enhancement Framework.

Enhancements maintained by the Enhancement Framework can be switched using the Switch Framework. SAP makes extensive use of the Switch Framework as a new way to allow customers to switch on new functionality (e.g., industry solutions). Once created, all enhancements on all levels (SAP, partner, and customer) are switchable, but to be part of the Switch Framework, a switch has to be assigned. The example in Figure 1.18 shows switch EA-CP (for Consumer Products), which was assigned to an implementation of BAdI definition `BADI_SD_SALES`. The switch is currently on, so the implementation is executed.

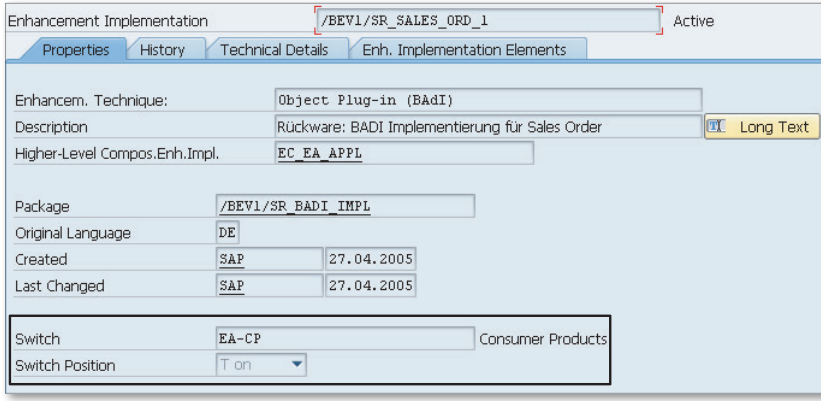


Figure 1.18 Enhancement Switch EA-CP for an Implementation of BAdI BADI_SD_SALES

If no switch is assigned to an enhancement, it can't be controlled by the Switch Framework. However, the enhancement is still compiled and part of the development object it enhances. Therefore, if an enhancement is activated but has no switch assignment, it will be called unconditionally.

As mentioned previously, switches are used extensively by SAP and its partners to provide upgrades and control the operation of large chunks of software functionality.

By the same token, customers can also use switch techniques for their own in-house developments.

Note

Refer to SAP Online Help (*http://help.sap.com*) or the SAP Marketplace (*http://service.sap.com*) for further details on development component switches.

1.4 Comparison of Enhancement Methods

Before we move on, let's reflect on the enhancement methods that we've discussed in this chapter by comparing their individual strengths and weaknesses.

Table 1.1 displays the discussed enhancement techniques. The comparison criteria were chosen because they represent the benefits to the implementing customer when developing, supporting, or upgrading them. Table 1.2 provides an overview and more details on the comparison criteria used.

	Modifications	User Exits	Customer Exits	Routines	Enhancement Framework (BAdIs)
Not affecting standard code (modification-free)			√	√	√
Not affecting upgrades		√	√	√	√
In customer namespace			√	√	√
Switchable			√	√	√
Filtering possible					√
Not preplanned					√
Defined interface			√		√
Multilevel enhancement					√

Table 1.1 Comparison of Enhancement Techniques

Criteria	Description	Comment
Not affecting standard code	SAP standard code is not altered as a result of the enhancement.	Strictly speaking, only customer exits, routines, and BAdIs fall into this category. User exits initially require a modification of standard code, so they technically affect the core.
Not affecting upgrades	Enhancement does not affect upgrade development work (except for testing).	All techniques—except modifications—allow for enhancements to "survive" an upgrade.
In customer namespace	Enhancements reside in customer's `Y*` or `Z*` namespaces.	In close conjunction with the first criteria. User exits *can* exist partially in customer namespaces if `Y*` or `Z*` includes are used.
Switchable	Enhancements can easily be switched on and off.	Only customer exits, routines, and the Enhancement Framework can provide this. Other solutions can only be switched using a custom solution.
Filtering possible	Enhancements can be called based on parameters.	One of the true domains of the Enhancement Framework! Again, only custom solutions can provide something similar for other techniques.
Not preplanned	Enhancement can be placed in other places than where defined by SAP.	No method allows for true freedom of placement here, but implicit enhancements do give some freedom here.
Defined Interface	Enhancements have a clear, restrictive input/output signature.	Only customer exits and the Enhancement Framework have this feature. User exit forms can use parameters, but this is not enforced.
Multilevel enhancement	Partners and customers can create their own enhancements on top of the SAP core.	Once again, only the Enhancement Framework trumps here. All other techniques do not support this.

Table 1.2 Comparison Criteria

In conclusion, there is no silver bullet when it comes to enhancement techniques. However, the Enhancement Framework unites the majority of advantages—a winner by points, so to speak.

1.5 Summary

Generally, newer SAP software benefits most from the Enhancement Framework. The SD component in particular can be seen as a victim of its age because early user exits, for example, could not be created using more sophisticated methods such as BAdIs. In other words, there is no clear-cut answer as to what the best enhancement technique is for SD.

However, it's paramount to always keep an open mind about *all* different enhancement techniques offered. The Enhancement Framework should be the preferred technique of choice, but sometimes only a user or customer exit might be in exactly the right place. Other reasons not to use the Enhancement Framework may include the following:

- Data required in the envisaged enhancement isn't accessible in BAdI parameters.
- An already existing user exit is available that could simply be extended.
- Performance reasons dictate the use of a different method.

Therefore, finding the right technique for an enhancement in SD is always a matter of weighing options and considering trade-offs. This chapter provided an overview of the most relevant SD enhancement techniques and their strengths and weaknesses to help you made this decision.

In your daily work as a consultant or developer, keep the five questions at the beginning of this chapter in mind when considering enhancements.

The next part of this book presents different development scenarios. You'll learn how to apply each of the enhancement methods outlined in the mind map (refer to Figure 1.1) earlier in this chapter. Most examples are taken from real-life scenarios. However, some had to be amended slightly, due to their initial complexity, to keep the focus on the enhancement technique.

PART I
Enhancements in Sales Order Processing

You can leverage a user exit as an enhancement, while at the same time using the Enhancement Framework to provide more flexibility in your solution.

2 Validating Sales Order Data

Sometimes enhancements can be very useful to automate an existing manual or labor-intensive process. For some of Byrell's clients, sales orders tend to change a lot before the actual delivery is created. Therefore, underlying payment terms data might not reflect the correct payment terms position anymore.

In this chapter, you'll learn how to perform additional validations and checks when creating or changing a sales order. Christine and Sean will use a custom table and a user exit to dynamically change payment terms while an order is created or changed. Additionally, Christine develops an exit routine to change payment terms depending on sales order net value.

2.1 Business Scenario

You've already learned in the Preface that Byrell Corporation is a logistics business. For customers to pay for their received services, Byrell provides several payment terms, currently all based on order value. Up to now, the teams who pick and dispatch goods in the warehouse made it part of their final check to ensure that the correct payment terms were chosen. This was always regarded as an archaic system, and the request to automate this process has been on the "To Do" list for a while, but there has never been the right opportunity for this development.

Due to a higher sales order and picking volume in recent days, the warehouse team hasn't always been able to perform these order value checks, leading to friction between warehouse and customer services teams, as Byrell customers complained about incorrect or missing discounts. As such, it has now become urgent to find a system-driven solution to fix the problem.

Sean was called into a meeting with the two teams to address the problem. During the meeting, he discovered that there are four different order value categories linked with four different payment methods, as displayed in Table 2.1.

Sales Order Value	Payment Method	Explanation
0 – 500 EUR	0001	Payable immediately due net
501 – 1000 EUR	0002	Within 14 days, 2% cash discount Within 30 days, due net
1001 – 2500 EUR	0003	Within 14 days, 2% cash discount Within 20 days, 1% cash discount Within 30 days, due net
> 2501 EUR	0004	Within 14 days, 3% cash discount Within 20 days, 2% cash discount Within 30 days, due net

Table 2.1 Order Value Categories

To fix the problem, Sean decides that whenever a sales order is created or changed, the payment method should be redetermined by the system. In addition, the user will receive an information message in the status bar about the change of payment method. One further requirement is that the redetermination should occur when a user presses the Enter key, or in other words, every time the sales order is validated.

2.2 Finding the Right Enhancement

Sean now has enough information to start writing a functional specification (FS), which he will then pass on to Christine, a junior ABAP developer who has recently joined the SD team. Sean knows that the redetermination of payment terms has to be done using an exit routine. However, before he begins with the FS, he has to do a little bit of research to find the right location for the exit.

Byrell's SD solution already has several implemented user exits in ABAP include `MV45AFZZ`. Because it's an enhancement technique he is comfortable with, Sean decides to take advantage of one of the user exits already being used.

Note

User exits are the best choice for any validation or amendment when creating or changing a sales order document, because of their pre-defined location within the standard code.

To begin this process, he takes a closer look at the PAYMENT TERMS field of the sales order header, which you can see in Figure 2.1.

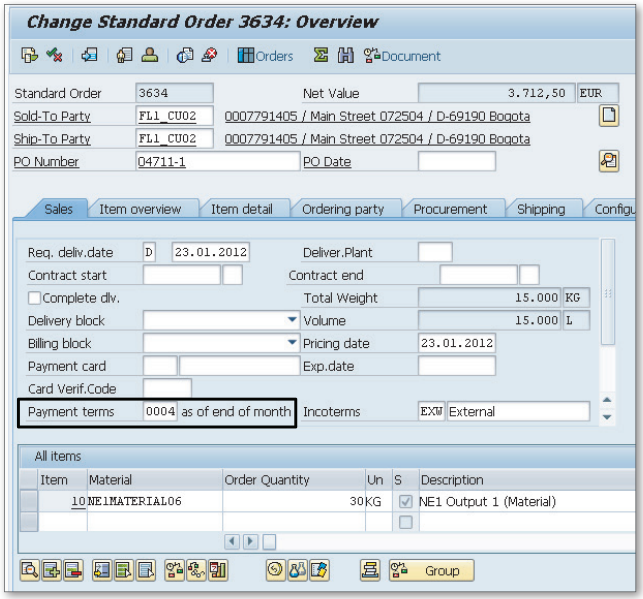


Figure 2.1 Payment Terms field in the Sales Order Header (Transaction VA01/02/03)

Sean has to find out the database table and field name for the payment terms. This will help him in finding a suitable user exit to change the payment terms value. He gets additional technical information about the field by following these steps:

1. Place the cursor in the PAYMENT TERMS field.
2. Press the F1 key.
3. In the popup window, click on TECHNICAL INFORMATION.

In this case, a new popup window reveals the database table and field name (VBKD-ZTERM; see Figure 2.2).

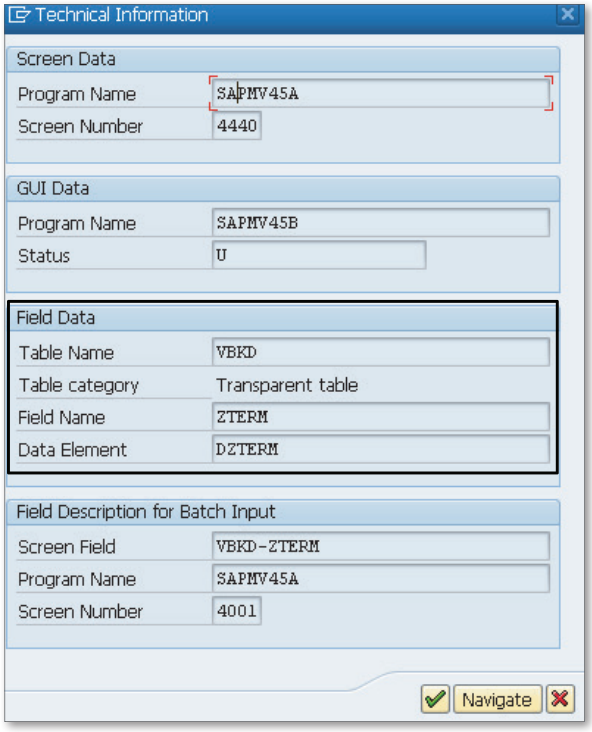


Figure 2.2 Technical Information for Payment Terms Field (ZTERM)

Database Table VBKD rings a bell for Sean, as he remembers that there is a user exit form routine to populate VBKD fields. He double-checks in include `MV45AFZZ` by pressing Ctrl+F and searching it for string `VBKD`. He finds the exit `USEREXIT_MOVE_FIELD_TO_VBKD`, which isn't in use yet. This exit is shown in Listing 2.1.

```
*---------------------------------------------------------------------*
*  FORM USEREXIT_MOVE_FIELD_TO_VBKD                                   *
*---------------------------------------------------------------------*
*  This user exit can be used to move some fields into the sales  *
*  document business data work area VBKD                              *
*                                                                     *
*       SVBKD-TABIX = 0:  Create data                                 *
*       SVBKD-TABIX > 0:  Change data                                 *
*                                                                     *
*       This form is called at the end of form VBKD_FUELLEN.      *
*                                                                     *
*---------------------------------------------------------------------*
```

```
FORM USEREXIT_MOVE_FIELD_TO_VBKD.
*  VBKD-zzfield = xxxx-zzfield2.

ENDFORM.
```

Listing 2.1 User Exit to Change Fields for Sales Document Business Data

This user exit is called as part of the population of structure `VBKD`, which is exactly what Sean wants to achieve because he wants to influence the payment terms field ZTERM.

Sean then turns his attention to the other steps the exit needs to perform to fulfill the user requirements. He decides to store the entries of Table 2.1 in a customer Z table rather than hard code them. This allows easier adjustments in the future, should customer services change their minds about the order value categories in use.

Sean now has the two key aspects for his development. He has found the right user exit to influence the value of field VBKD-ZTERM, and he will create a new database table in the data dictionary to hold the new order value categories.

To explain the process to Christine, he specifies the following steps in the functional specification:

1. Create a new database Z table—including maintenance routines—matching the layout of order value categories (refer to Table 2.1).
2. Use subroutine `USEREXIT_MOVE_FIELD_TO_VBKD` in ABAP include `MV45AFZZ` to perform an evaluation of the following steps:
   - Derive order value categories from the database (only select from the database if the internal table is empty).
   - Use order value categories to determine payment terms.
   - Issue an information message if the payment terms were changed.

Sean passes the functional specification on to Christine, who can now perform these steps.

2.3 Implementing the Solution

In this section, we'll explain the steps involved in implementing the solution per Sean's functional specification.

2.3.1 Create a New Database Z Table

When looking closer at the order value categories table (refer to Table 2.1), Christine notices that it might be better to split the *to* and *from* ranges (e.g., from €0 to €500) into separate fields and make both mandatory. In terms of conditions in his code, she decides to treat the *from* entry as ">=" (greater or equal) and the *to* entry as "<=" (less than or equal). The finished table (including data elements) looks like what is shown in Figure 2.3.

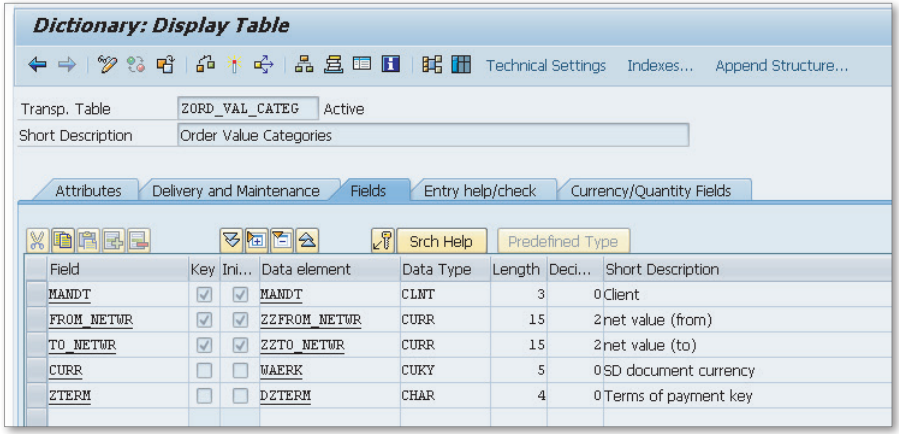


Figure 2.3 Table ZORD_VAL_CATEG with Custom Data Elements

2.3.2 Generate Table Maintenance

Christine uses the table maintenance generator (Transaction SE54) to create a small program suite to enable authorized users to adjust the table entries themselves in the development system. Figure 2.4 shows the generation parameters.

2.3.3 Create Parameter Transaction

Christine also has to create a Z transaction code to assign the table maintenance. To do this, she uses Transaction SE93 to create a new parameter transaction called `ZORD_VAL_CATEG` (see Figure 2.5). Upon accessing the transaction, she types "ZORD_VAL_CATEG" in the Transaction code field, clicks on the Create button, and comes to the Change Parameter Transaction screen (see Figure 2.5).

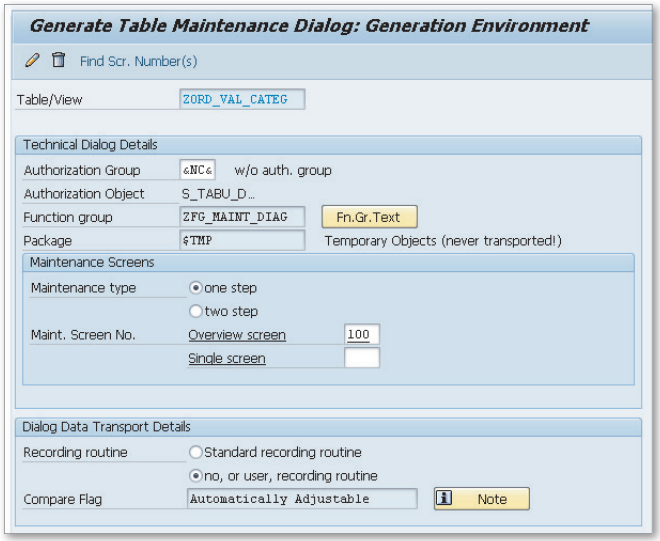


Figure 2.4 Table Maintenance Dialog for ZORD_VAL_CATEG

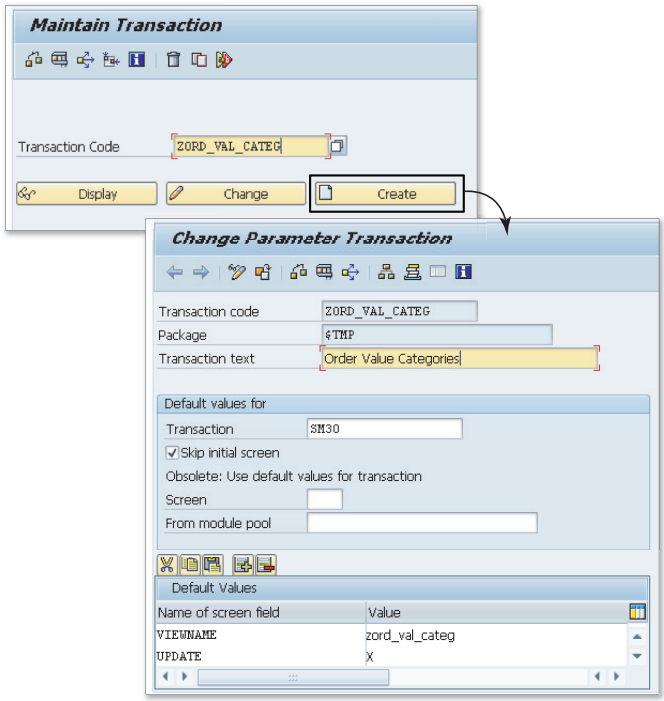


Figure 2.5 Create a Parameter Transaction to Call Transaction SM30

The transaction code also makes it easier to assign to a role profile later, ensuring the correct group of users can amend entries. Christine also fills in the fields at the bottom: she populates the table name field, and then clicks on the CHANGE button in Transaction SM30 (Database Table Maintenance), to alter the table entries.

2.3.4 Creating an Enhancement Point Implementation

Christine then begins with the actual coding of the exit. Her first step is to find the right place to add the variable declarations for the order value categories and other work variables she needs. She could declare these locally in the exit themselves, but she would have to reselect all entries from the Z table. Because the exit is called each time the user triggers field validation (e.g., when the Enter key is pressed), it's a better practice to declare the variables for the user exit in a `TOP` include to make sure items from the value category table do not have to be reselected. Figure 2.6 shows includes `MV45ATOP` and `MV45ATZZ`.

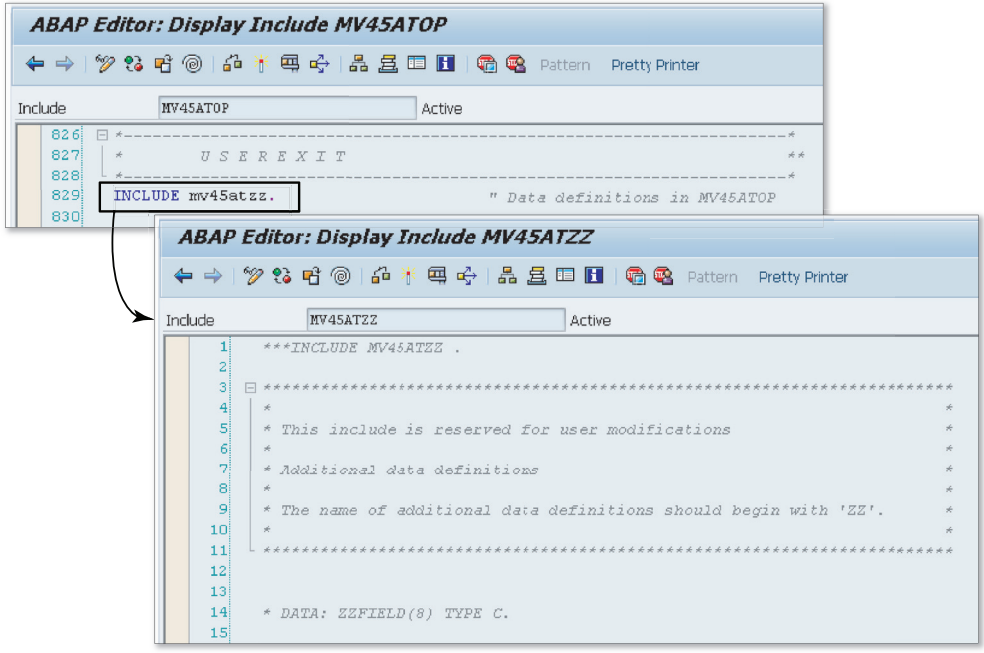


Figure 2.6 Program Includes MV45ATOP and MV45ATZZ

Christine inspects the enhancement point right under the `INCLUDE MV45ATZZ` statement and creates an enhancement implementation here.

To create an enhancement point implementation at this position in the code, Christine performs the following steps:

1. Enable the enhancement mode by clicking on the ENHANCEMENT button ( ).
2. Position the cursor on the enhancement statement in the code, right-click, and select ENHANCEMENT OPERATIONS • CREATE IMPLEMENTATION. Alternatively, she could have also achieved this by following the menu path EDIT • ENHANCEMENT OPERATIONS • CREATE IMPLEMENTATION.
3. When the ABAP Editor prompts to define or specify the actual enhancement implementation, create a new implementation by clicking on the CREATE button, enter the name "ZCB_SAPMV45A" in the ENHANCEMENT IMPLEMENTATION field, give it a description in the SHORT TEXT field, and then click on the green checkmark icon.
4. On the SELECT OR CREATE ENHANCEMENT IMPLEMENTATION screen, click on the checkmark icon again. This implementation will serve as a holding bucket for all enhancements created for this development task.
5. After confirming the last popup window, the ABAP Editor displays the new enhancement implementation, ready for input as shown in Figure 2.7.

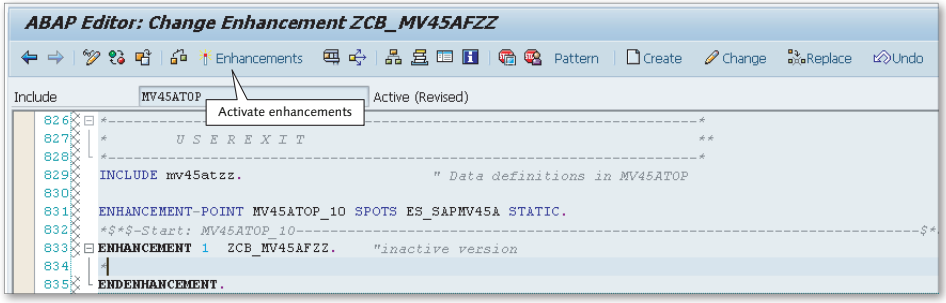


Figure 2.7 Enhancement Implementation Ready for Input in the ABAP Editor

6. Now, create the declaration, as shown in Listing 2.2.

```
ENHANCEMENT 1  ZCB_MV45AFZZ. "active version
DATA gt_ord_value_categ TYPE STANDARD TABLE OF zord_val_categ.
ENDENHANCEMENT.
```

Listing 2.2 Variable Declaration in MV45ATOP

7. Finally, activate the enhancement by clicking on the ENHANCEMENTS button (see Figure 2.7).

2.3.5 Implementing an Implicit Enhancement

Christine now turns her attention to the actual code in the user exit and implements the exit code as an implicit enhancement.

What If There's an Existing Modification?

There's no clear-cut answer for what to do if an enhancement already exists, but generally, it depends on the time that's available for development and testing. If you need to extend or change an old modification, then generally it's advisable to transfer the modification into an implicit or explicit enhancement, where possible, because of the improved support of the Enhancement Framework. However, a code migration such as this must be developed and tested. Unfortunately, project time pressures don't always allow for this. If possible, it's good practice to migrate old modifications to the Enhancement Framework.

Christine goes back to include program `MV45AFZZ`, where she locates subroutine `USEREXIT_MOVE_FIELD_TO_VBKD` and follows these steps:

1. Switch again to enhancement mode by clicking on the Enhancement button ( ).
2. Right-click on the dotted line, and select Enhancement Operations • Show Implicit Enhancements.
3. Place the cursor on the first line with the arrow next to it, and right-click on Enhancement Operations • Create Implementation.
4. Then create a code enhancement by selecting the enhancement implementation created earlier for the declaration (`ZCB_MV45AFZZ`). See Figure 2.8 for these steps.

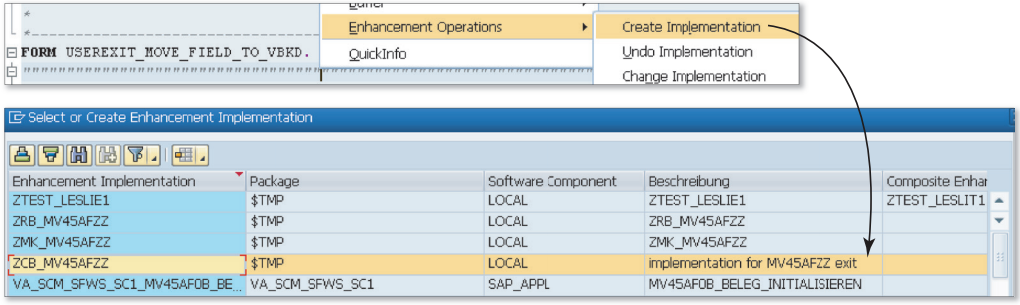


Figure 2.8 Create Implicit Enhancement Implementation within Form USEREXIT_MOVE_FIELD_TO_VBKD

2.3.6 Coding the User Exit

Christine then adds the following code to this newly created implicit enhancement, as shown in Listing 2.3.

```
FORM USEREXIT_MOVE_FIELD_TO_VBKD.
ENHANCEMENT 1  ZZ_MV45AFZZ.    "active version
DATA: ls_ord_value_categ TYPE zord_val_categ,
      lv_zterm_changed   TYPE abap_bool VALUE abap_false.

IF ( sy-tcode    = 'VA01'     OR
     sy-tcode    = 'VA02' )   AND
   xvbak-netwr > 0.      " no reason to change if net value is 0

* Step 1 - fill global order value category table if empty
  IF gt_ord_value_categ[] IS INITIAL.
    SELECT *
      FROM zord_val_categ
      INTO TABLE gt_ord_value_categ.

    IF sy-subrc <> 0.
      MESSAGE E001(ZMSG_MV45A). " No Order Value Categories found.
    ENDIF.
  ENDIF.

* Step 2 - loop at entries for document currency and find
*          matching payment terms
  LOOP AT gt_ord_value_categ INTO ls_ord_value_categ
       WHERE curr = xvbak-waerk.
    IF xvbak-netwr >= ls_ord_value_categ-from_netwr AND
       xvbak-netwr <=  ls_ord_value_categ-to_netwr.
      IF xvbkd-zterm <> ls_ord_value_categ-zterm.
        vbkd-zterm = ls_ord_value_categ-zterm. " populate pt
        lv_zterm_changed = abap_true.
        EXIT.
      ENDIF.

* Step 2.5 - if "to" field empty and net value >= from value
*            then assume open end
    ELSEIF ls_ord_value_categ-from_netwr <= xvbak-netwr AND
           ls_ord_value_categ-to_netwr   =  0.
      IF xvbkd-zterm <> ls_ord_value_categ-zterm.
        vbkd-zterm = ls_ord_value_categ-zterm. " populate pt
```

```
        lv_zterm_changed = abap_true.
        EXIT.
      ENDIF.

    ENDIF.
  ENDLOOP.

* Step 3 - give out message about change of payment terms
  IF lv_zterm_changed = abap_true.
    MESSAGE S000(ZMSG_MV45A). "   Payment Terms adjusted
  ENDIF.

ENDIF.
ENDENHANCEMENT.

*  VBKD-zzfield = xxxx-zzfield2.
ENDFORM.
```

Listing 2.3 Coding for USEREXIT_MOVE_FIELD_TO_VBKD

Within this implicit enhancement, Christine first defines a local structure to hold the order value categories (`ls_ord_value_categ`) and a Boolean variable to remember whether payment terms were changed or not (`lv_zterm_changed`). The latter will help with sending a message to users informing them if a change was made.

Most importantly, the enhancement is only processed if called during sales order creation (Transaction VA01), sales order change (Transaction VA02), and if a sales order's net value is greater than zero.

Let's go over the different coding steps.

Step 1

Christine checks whether the globally declared internal Table GT_ORD_VALUE_CATEG already holds the entries from database Table ZORD_VAL_CATEG and if necessary fetches these from the database. This check is important because the user exit is called several times during validation, and she wants to avoid unnecessary database selections.

Step 2

The code loops around all entries of Table GT_ORD_VALUE_CATEG with the sales order header currency, checking for two scenarios:

- Sales order net value is between the "to" and "from" values of the internal table.
- Sales order net value is above the upper order value limit, which in the table is identified by a "from" value greater than zero and a "to" value equal to zero.

In both of these cases, the payment term value is changed if the order net value is found to be within one of the categories. Note that the code only changes the payment term if the new terms are different from the old ones, thus avoiding unnecessary changes and messages.

Step 3

Finally, step 3 evaluates the `lv_zterm_changed` flag and gives out a success message.

2.4 Summary

In this chapter, Sean and Christine leveraged a user exit to automate a previously manual process (checking order value to find the correct payment terms).

The user exit coding was added using implicit enhancements and by implementing an enhancement point for data declaration. Using Enhancement Framework techniques (implicit enhancements and enhancement points), rather than modifications, provides a solution that is more flexible (in terms of additional implementations) and doesn't require code registration.

They also used a custom Z table to hold cross-reference order category values, rather than hard-coding them. This also helps to prevent further code change in the future if these categories need to be amended or extended.

In the next chapter, you'll learn how to customize tabs in an end-to-end example for a small customer survey enhancement. This example will show you how to use several different enhancement methods in one comprehensive enhancement.

You can combine both older and modern enhancement methods to create a more flexible solution.

3 Capturing and Saving Additional Data Fields in Sales Order Processing

Transactions VA01/02/03 for sales orders provide two additional, customizable tabs for the customer to use as desired. For example, the tabs can be added if the customer service and marketing departments think they might need to amend the catalog of questions in the future. Another future requirement might also be to enable flexible questions, making them manageable without system changes.

This chapter shows how Christine and Sean enable one of these tabs. They come up with a simple, yet effective way to capture additional data within the tab, thereby giving the business an opportunity to capture vital customer-relevant data. This is accomplished by using the Enhancement Framework and user exits in tandem. In addition, you'll see how Christine uses persistent objects to interact with the database.

3.1 Business Scenario

As a part of its sales and logistics services, Byrell accepts sales orders on behalf of its partners and then fulfills them. Byrell's marketing and customer service departments want to gain better insight into end-customer satisfaction and have therefore decided to ask customers to rate their experience. Most customers place their orders by phone, so it was deemed easiest to capture responses as part of this overall process.

Sean has been briefed by both departments about the kind of data that needs to be captured in SAP. The current survey data needed is provided in Table 3.1.

Number	Requirements
1	Overall customer satisfaction (rate from 0–10).
2	Other comments (text field).
3	Customers should be able to leave more than one entry in the satisfaction survey but not more than one within 60 days. Byrell sales advisers should only be prompted to ask customers to take the survey if the last feedback is at least 60 days old.

Table 3.1 Byrell Survey Requirements

3.2 Finding the Right Enhancement

Sean knows that the standard transactions (VA01/02/03) for sales orders in SD have tabs for additional data (see Figure 3.1), which allow for the collection of order-relevant header and item data. However, Byrell's system doesn't currently use these.

Figure 3.1 Additional Data Tabs A and B in Transactions VA01/02/03

About Additional Data in Sales Order Processing

SAP introduced Additional data tabs in release 4.6c in the main sales order transactions (VA01/VA02/VA03), which allow customers to specify supplementary data in header and item sales order tables. There are ten fields in the Additional data A tab page (Customer and Condition groups 1 – 5 fields in the header and Material and Condition group 1 – 5 fields in the item). These fields do not affect the functions in the standard version of SD and can be used freely for entering data or for making modifications. Figure 3.1 shows an example of the Additional data B tab in the sales order header table, which is blank because it's freely designable by customers without restrictions.

Sales order header (VBAK) fields are already predefined in the Additional Data A tab, but they can't be used because the database table Sean will define to store

the survey data has to be custom (within the Y or Z namespace). This is necessary for two reasons: First, the survey text field has to be appended to the sales order header table, which isn't recommended from a performance perspective due to its size. Second, other developments and reports will access this custom survey table. Therefore, it's better to create a new table in the customer name space for it. Because the ADDITIONAL DATA A tab is already predefined to use VBAK fields, Sean will use the ADDITIONAL DATA B tab instead.

3.2.1 Using the Additional Data B Tab

Sean decides to use the sales order header tab ADDITIONAL DATA B, which is freely definable. However, Sean isn't completely sure about the technical requirements in this situation and arranges to meet with Christine to establish how his envisaged survey screen needs to be inserted into the ADDITIONAL DATA B tab.

3.2.2 Using Implicit Enhancements

During her meeting with Sean, Christine examines ABAP program `SAPMV45A`, screen 8309 (the ADDITIONAL DATA B tab screen) and notices that she won't be able to simply enhance the standard tab with the Enhancement Framework because there are no explicit enhancements or hooks for implicit enhancements within the screen flow logic. Instead, she will have to modify the standard flow for screen 8310 to call the screen modules required. But it's not all bad news: the necessary PBO (process before output) and PAI (process after input) modules for her new survey in screen 8310 can be defined using implicit enhancements.

Sean already produced a field layout overview for the custom survey database table for a development specification (see Table 3.2).

Field Name	Key	Description	Data Element
MANDT	X	Client	MANDT
KUNNR	X	Customer number	KUNNR
DATE_LAST	X	Date of last survey	DATE
TIME_LAST	X	Time of last survey	TIME
CUST_SATISF		Customer satisfaction	NUMC (2)

Table 3.2 Sean's Suggested Database Table Layout for Survey Data

Field Name	Key	Description	Data Element
TEXT1		Other comments 1	CHAR (256)
TEXT2		Other comments 2	CHAR (256)
TEXT3		Other comments 3	CHAR (256)
TEXT4		Other comments 4	CHAR (256)
TEXT5		Other comments 5	CHAR (256)

Table 3.2 Sean's Suggested Database Table Layout for Survey Data (Cont.)

However, Christine isn't happy with Sean's suggested field layout. Her main criticism is that the layout shown in Table 3.2 doesn't cater to potentially longer texts in the survey form. As it stands, the table can only hold a maximum of 5 text lines with 256 characters each.

As an alternative, Christine suggests the following two-table layout, which will enable call handlers to enter as many text lines as they want to (see Table 3.3 and Table 3.4).

Field Name	Key	Description	Data Element
MANDT	X	Client	MANDT
KUNNR	X	Customer number	KUNNR
DATE_LAST	X	Date of last survey	DATE
TIME_LAST	X	Time of last survey	TIME
TEXT_KEY	X	Key to second text table	CHAR (10)
CUST_SATISF		Customer satisfaction	NUMC (2)

Table 3.3 Table ZCUST_SATISF_T: Main Database Table for Customer Satisfaction Results

Field Name	Key	Description	Data Element
MANDT	X	Client	MANDT
TEXT_KEY	X	Key to second text table	CHAR (10)
LINE	X	Survey text line number	NUM (3)
TEXT		Survey text line	CHAR (256)

Table 3.4 Table ZCUST_SATISF_TT: Database Table for Customer Satisfaction Texts

For the TEXT_KEY field that will connect the two tables, Christine and Sean agree to use the concatenated entries for KUNNR, DATE_LAST, and TIME_LAST.

3.2.3 Using Persistent Objects

As far as reading and creating database table entries is concerned, Christine suggests using something that is new to Sean: persistent objects. Christine explains that the persistence service can be thought of as a software layer between the ABAP program and database, which allows you to save data and load it again when needed, without the need to code any `SELECT` or `INSERT` statements.

All she needs to do is declare the Data Dictionary objects as shown in Table 3.3 and Table 3.4, and then code the methods to call the persistence service, which can be reused by all other applications using the survey database tables. The persistence classes themselves come with automatically generated methods and agent classes, which are called when database table entries have to be read or saved.

Note

The code for this example will give you all of the details for a persistency-enabled survey data solution. If you want to find out more about this exciting topic, refer to the following resources:

- *Next Generation ABAP*, 2nd edition, by Rich Heilman and Thomas Jung (SAP PRESS 2011)
- SAP Help (*http://help.sap.com*), "Concepts Used in Persistence"
- *Object Services in ABAP* by Christian Assig, Aldo Hermann Fobbe, and Arno Niemietz (SAP PRESS 2010)

3.2.4 USEREXIT_SAVE_DOCUMENT

Additionally, user exit `USERXIT_SAVE_DOCUMENT` must be amended to incorporate the handling of saving survey data to the database tables described in Table 3.3 and Table 3.4.

After her meeting with Sean, Christine creates a short task list of steps that need to be undertaken to create the survey tab.

Development Task List

- Create data elements, structures, and database tables as specified in Table 3.3 and Table 3.4.
- Create persistent classes for these database tables.
- Create database table lock entries for `ZCUST_SATISF_T` and `ZCUST_SATISF_TT`.
- Modify screen flow in program SAPMV45A, screen 8309 to call new `ZZ*` modules.
- Create new screen flow modules for PBO and PAI using the implicit enhancement technique.
- Amend user exit `USEREXIT_SAVE_DOCUMENT` to save survey data.

We'll now go through each of these steps and implement the solution.

3.3 Implementing the Solution

Christine is now ready to get started with her development task list.

3.3.1 Declarations of Data Elements, Database Tables, and Table Types

Using Transaction SE11, Christine creates four data elements, which will later be used in the survey database table (see Table 3.5).

Data Element	Data Type
`ZTEXT_KEY`	CHAR (10)
`ZCUST_SATISF`	NUM (2)
`ZSURV_LINE`	NUM (3)
`ZSURV_TEXT`	CHAR (256)

Table 3.5 Survey Data Elements

She then creates the following data dictionary objects:

- Database Tables ZCUST_SATISF_T and ZCUST_SATISF_TT with the layout as specified in Table 3.3 and Table 3.4 (see Figure 3.2).

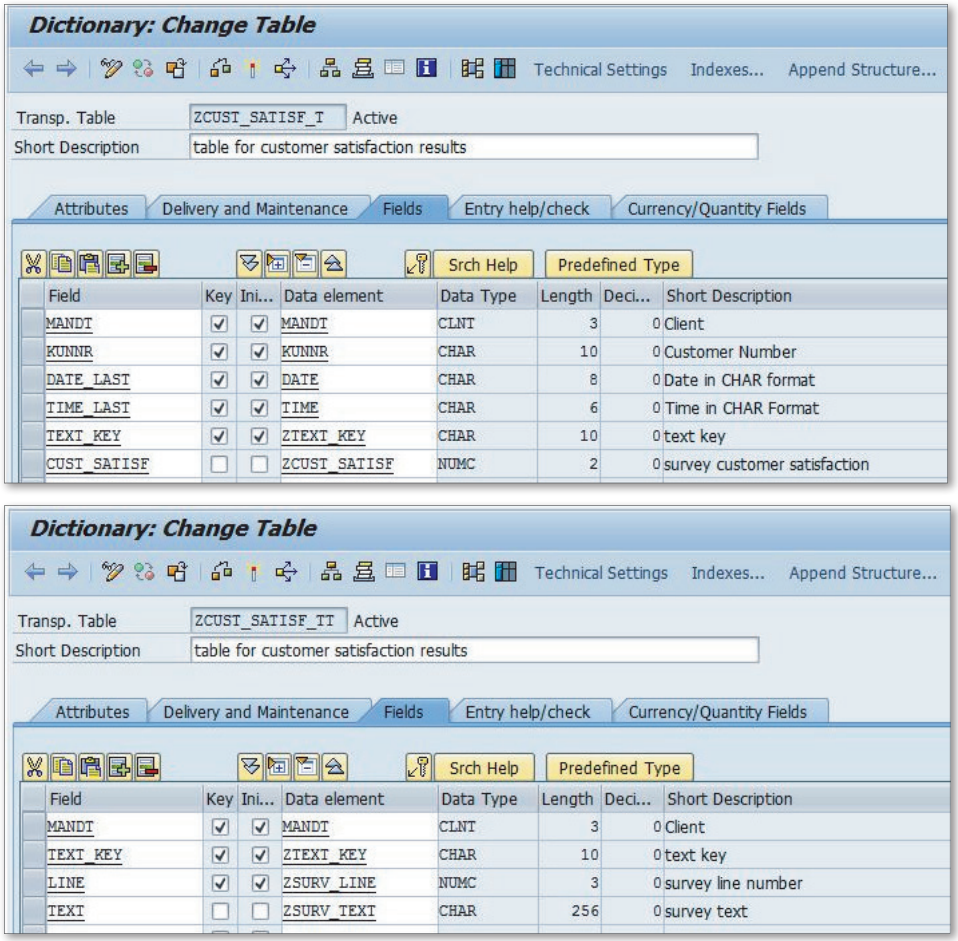


Figure 3.2 Database Tables ZCUST_SATISF_T and ZCUST_SATISF_TT

- A foreign key relationship using field TEXT_KEY (see Figure 3.3) in database Table ZCUST_SATISF_TT, as this shared by tables ZCUST_SATISF_T and ZCUST_SATISF_TT.
- Table types ZSURV_T and ZSURV_TEXT_T with the layout as specified in Table 3.3 and Table 3.4.

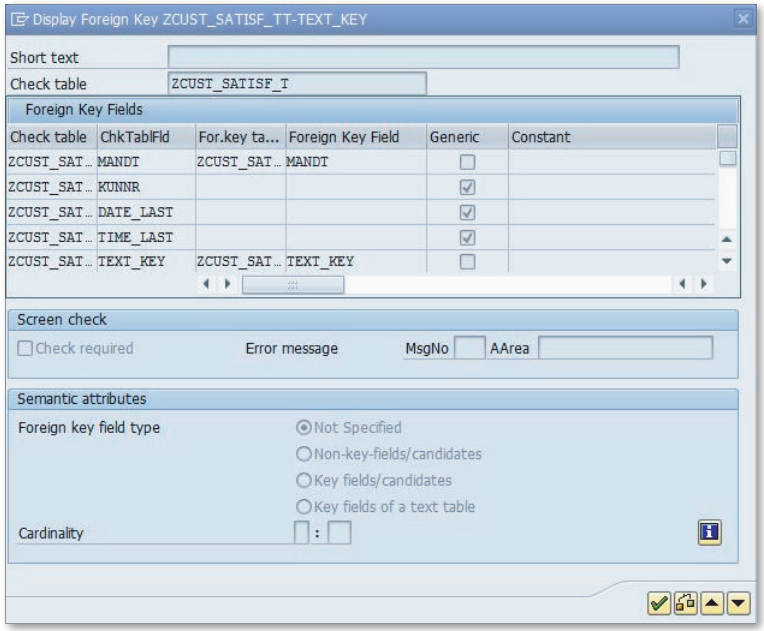


Figure 3.3 Foreign Key Relationship for Field TEXT_KEY

3.3.2 Create Persistency Classes for Survey Data

Christine is ready for the second step in her list. Persistency classes enable end users to read and save survey data, which will be generated for each of the two survey database tables. In addition, Christine will create a survey class, which is a data model class that interacts with these persistency classes. This survey class will be the one that all other applications use to read and save survey data.

First, Christine uses the Class/Interface view in Transaction SE80 to create the persistency classes for Tables ZCUST_SATISF_T and ZCUST_SATISF_TT.

When she enters class "ZCL_SURVEY_PERS" in the Object Type Name field and clicks the Search button, the Create Object popup screen appears stating that the class doesn't exist and asking whether she wants to create the object. She clicks Yes to create the object.

A new popup is displayed where Christine gives her new class a description, and then clicks the Persistent Class radio button. Instantiation is changed automatically

to *Protected*, which is the default setting for persistency classes. After assigning package and transport requests, she is then presented with the new persistency class as shown in Figure 3.4.

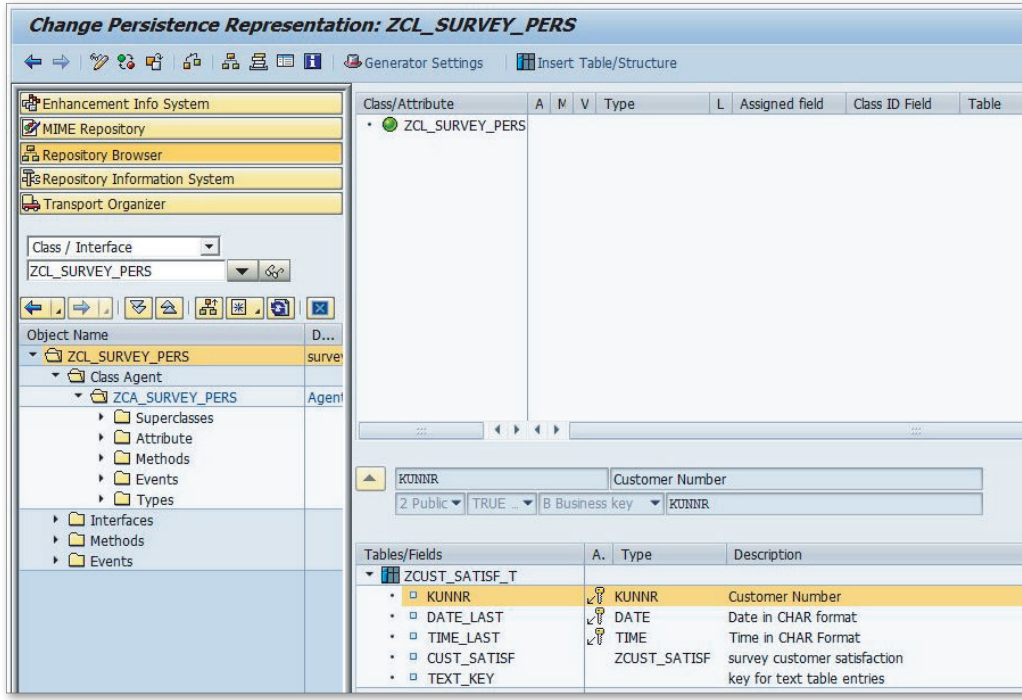


Figure 3.4 Empty Persistency Class ZCL_SURVEY_PERS

Christine enters change mode by clicking the DISPLAY/CHANGE button, in order to add new fields to her persistent class. She then assigns the database table fields to the persistency class by double-clicking on each field and then pressing the up arrow button. After all fields are assigned, the screen will look like the one shown in Figure 3.5.

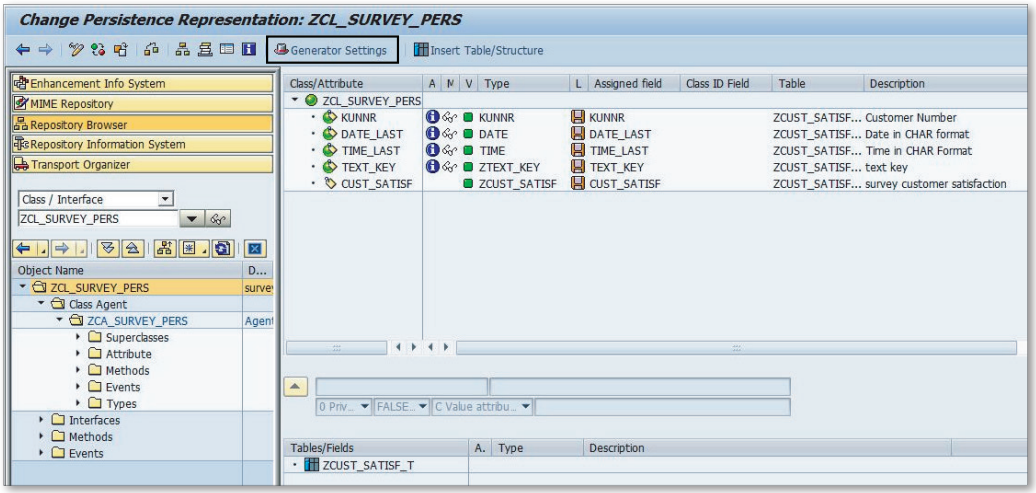


Figure 3.5 Persistency Class ZCL_SURVEY_PERS with Assigned Fields and Generator Settings

It's now time to generate the required methods, but because Christine also wants to generate code to run queries, she first has to adjust the generator settings. She clicks the GENERATOR SETTINGS button and ensures the boxes shown in Figure 3.6 are ticked.

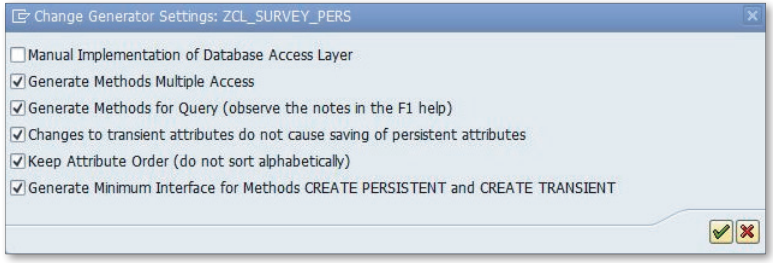


Figure 3.6 Persistency Class Generator Settings

She then saves her work and returns to the Class Builder using the BACK button. All necessary methods will be generated now that the class is active.

Christine then repeats the same steps for the second persistency class, `ZCL_SURVEY_TEXT_PERS`.

3.3.3 Create Database Table Lock Entries

Christine's preparations are almost done. Her next step is to generate the lock object for Tables ZCUST_SATISF_T and ZCUST_SATISF_TT, which will prevent other users from changing data while it's being processed.

Using Transaction SE11, she creates a new lock object named `EZ_SURVEY` for primary Table ZCUST_SATISF_T. In the Lock Mode field, she chooses E_Write Lock. Because there's a foreign key relationship between these two tables, Christine decides to specify a secondary table and thus generates only one set of function modules to set and unset the required database write locks (see Figure 3.7).

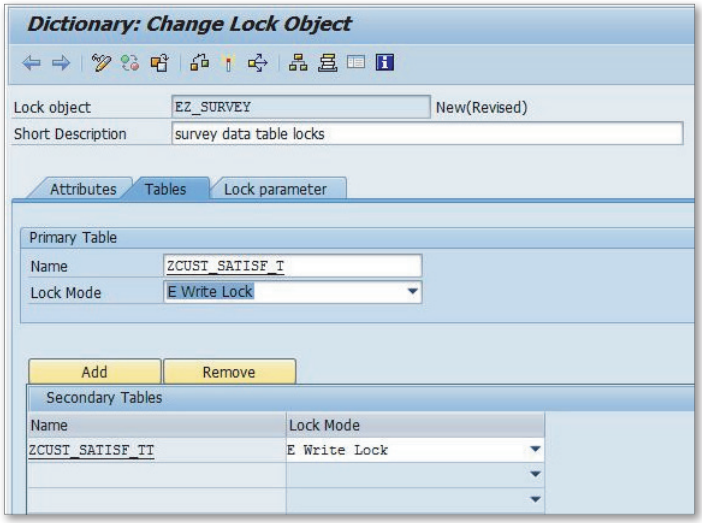


Figure 3.7 Lock Object Details

3.3.4 Creating Modification and Implicit Enhancements for a Custom Screen

Although Christine uses the Enhancement Framework heavily, she regrettably has to create a code modification for PBO and PAI modules of custom screen 8309. Modifications are always a last resort for Christine. In this case, she has no other choice, because screen flow isn't supported by the Enhancement Framework.

Figure 3.8 shows the popup window in which she will enter the source code registration code.

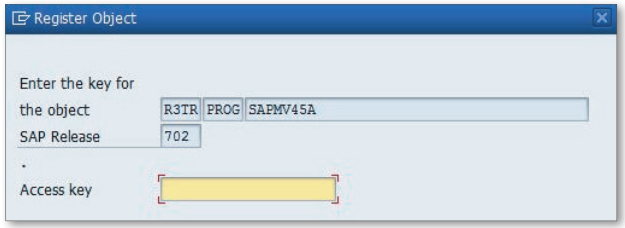


Figure 3.8 Object Registration Dialog for SAPMV45A

After the object is registered via the SAP Marketplace at *http://service.sap.com* (separate login and authorization required) as shown in Figure 3.8, Christine enters the change mode of the ABAP Editor by clicking on the DISPLAY/CHANGE button. She then clicks on the INSERT button to create a code modification.

With the modification now allowing her to adjust the screen flow, she declares three screen flow modules called `prepare_screen` and `get_survey_data` for PBO and `pai` for PAI as shown in Figure 3.9.

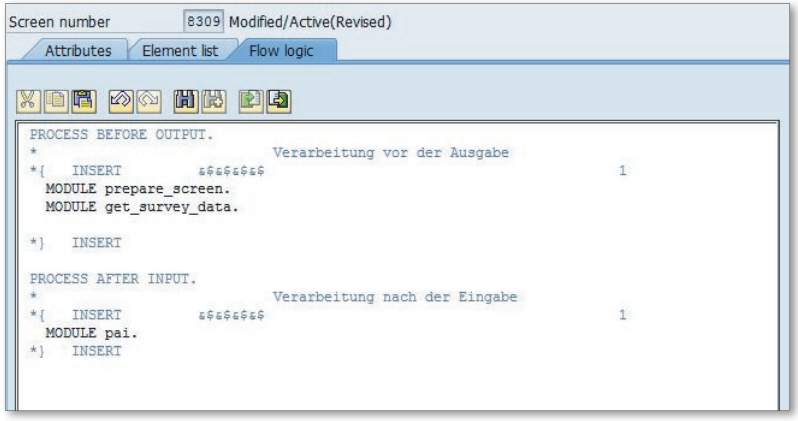


Figure 3.9 PBO and PAI Modules with Modifications

3.3.5 Create the Implicit Enhancement for PBO Modules

Christine then turns her attention to ABAP include `MV45AO0Z`, where she will create the coding for her PBO modules `prepare_screen` and `get_survey_data`. The implicit enhancement is created by following these steps:

1. Click on the ENHANCEMENT button ( ).
2. Right-click on the code, and choose ENHANCEMENT OPERATIONS • SHOW IMPLICIT ENHANCEMENT OPTIONS.
3. Position the cursor on the highlighted implicit enhancement, right-click, and choose ENHANCEMENT OPERATIONS • CREATE IMPLEMENTATION.
4. The new implementation appears as shown in Figure 3.10. Mark the implementation on the following screen, and click CONTINUE.

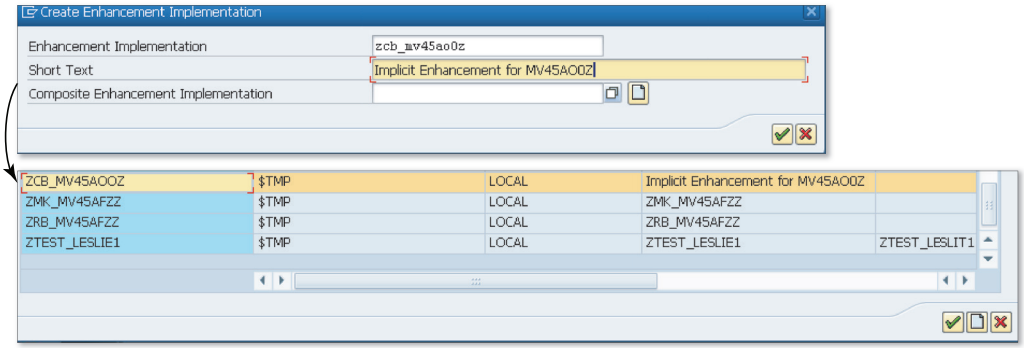


Figure 3.10 Creating and Selecting the Enhancement Implementation

Now a new implicit enhancement method appears in the ABAP Editor, and Christine adds the code detailed in Listing 3.1.

```
MODULE prepare_screen OUTPUT.
  DATA: lv_enable TYPE c VALUE ' '.
  IF go_editor IS NOT BOUND.

*   create control container
    CREATE OBJECT go_editor_container
      EXPORTING
        container_name = 'CONTAINER'.

*   create calls constructor, which initializes, creates and links
```

```
*   TextEdit Control
    CREATE OBJECT go_editor
      EXPORTING
        parent                     = go_editor_container
        wordwrap_mode              = cl_gui_textedit=>wordwrap_at_
fixed_position
        wordwrap_position          = cv_line_length
        wordwrap_to_linebreak_mode = cl_gui_textedit=>true.

  ENDIF.
ENDMODULE.                 " prepare_screen  OUTPUT


MODULE get_survey_data OUTPUT.
  DATA: ls_survey    TYPE zcust_satisf_t,
        lv_delta_min TYPE mcwmit-be_ae,
        lv_days(8)   TYPE n,
        lv_screen    TYPE screen,
        ls_text      TYPE zcust_satisf_tt,
        lt_text      TYPE STANDARD TABLE OF char256.

  gv_do_survey = abap_false.
  FREE go_survey.

  CREATE OBJECT go_survey
    EXPORTING
      iv_kunnr = vbak-kunnr.

  READ TABLE go_survey->gt_survey[] INTO ls_survey INDEX 1.

* establish whether open for change or not

  IF sy-subrc = 0.
    CALL FUNCTION 'L_MC_TIME_DIFFERENCE
      EXPORTING
        date_from  = ls_survey-date_last
        date_to    = sy-datum
      IMPORTING
        delta_time = lv_delta_min.

    lv_days = lv_delta_min / 1440.        " 1440 minutes in a day, get
no of days
    IF lv_days >= 60.
```

```
      gv_do_survey = abap_true.
      CALL METHOD go_editor->set_readonly_mode( readonly_mode = 0 ).

    ELSE.

*     set dropbox and text editor to read-only
      gv_do_survey = abap_false.
      LOOP AT screen INTO lv_screen.
        IF lv_screen-group1 = 'DBX'.
          lv_screen-input = 0.
          MODIFY screen FROM lv_screen.
        ENDIF.
      ENDLOOP.

      CALL METHOD go_editor->set_readonly_mode.

*     populate existing survey info into fields
      gv_satisfaction = ls_survey-cust_satisf.
      LOOP AT go_survey->gt_text INTO ls_text
              WHERE text_key = ls_survey-text_key.
        APPEND ls_text-text TO lt_text.
      ENDLOOP.
      CALL METHOD go_editor->set_text_as_r3table
        EXPORTING
          table = lt_text.
    ENDIF.
  ENDIF.
ENDMODULE.
```

Listing 3.1 Screen Flow Modules prepare_screen and get_survey_data

Note

The code for the `go_survey` class that Christine created can be found in Appendix B.

Module `prepare_screen` is a PBO module, which creates container and editor objects that will be displayed on custom screen 8309 (the declaration and changes to screen 8309 are described later in this chapter).

Note that from the input parameters of object go_editor, Christine sets the editor to perform a word wrap at a specific line length (256 characters). She also sets the word wrap to linebreak mode, which means words that exceed the end of the line are moved to the beginning of a new line.

The module get_survey_data in the code listing uses function module L_MC_TIME_DIFFERENCE to establish whether the 60 days since the last survey have passed, in which case the survey rating field and text box are opened up for entry. However, if the last entry isn't longer than 60 days, the program won't allow any new survey entries and sets both fields to read-only mode.

3.3.6 Create the Implicit Enhancement for Module PAI

Similarly, Christine now creates an implicit enhancement in include MV45AI0Z for the coding of module PAI. She uses the same enhancement implementation as before. After the implicit enhancement is created, she adds the code shown in Listing 3.2.

```
MODULE pai.
  REFRESH gt_text.

* retrieve table from control
  CALL METHOD go_editor->get_text_as_r3table
    IMPORTING
      table       = gt_text
      is_modified = gv_modified.
ENDMODULE.
```

Listing 3.2 Screen Flow for Module PAI

Upon PAI, the code in Listing 3.2 retrieves the text from the displayed editor and puts it back into Table GT_TEXT. Flag gv_modified indicates whether or not the text has been changed since the last processing by the user.

You've probably noticed that Christine's PAI module contains no coding for saving the entered survey data. This is being performed in a user exit we already discussed in Chapter 2: USEREXIT_SAVE_DOCUMENT. This user exit is located with the other form exits in ABAP include MV45AFZZ. Christine wants the survey to be saved together with all other sales document data, so it makes sense to use USEREXIT_SAVE_DOCUMENT for this.

Therefore, Christine creates an implicit enhancement within USEREXIT_SAVE_DOCUMENT in ABAP include MV45AFZZ (using the previously detailed implementation steps) and adds the code shown in Listing 3.3.

```
DATA: lv_modified TYPE i.
  REFRESH gt_text.
```

```
* retrieve table from control
  CALL METHOD go_editor->get_text_as_r3table
    IMPORTING
      table       = gt_text
      is_modified = lv_modified.

  IF lv_modified = 1.
    PERFORM save_survey.
  ENDIF.
```

Listing 3.3 Coding for userexit_save_document

This code saves the new survey data to the database tables using the persistent classes. Method `get_text_as_r3table` of class `go_editor` is called once again to retrieve the latest changes from the text editor and move these into Table GT_TEXT. If the text was changed (flag `lv_modified` is equal to 1), then subroutine `save_survey` is executed.

She then adds the subroutine `save_survey` at the end of this ABAP include `MV45AFZZ` because it's a separate, new form routine (see Listing 3.4).

```
FORM save_survey.
  DATA: lo_agent_survey      TYPE REF TO zca_survey_pers,
        lo_survey            TYPE REF TO zcl_survey_pers,
        lo_agent_survey_text TYPE REF TO zca_survey_text_pers,
        lo_survey_text       TYPE REF TO zcl_survey_text_pers,
        lv_result            TYPE REF TO object,
        lv_text_key          TYPE ztext_key,
        lv_text(256),
        lv_date              TYPE sy-datum,
        lv_time              TYPE sy-uzeit,
        lv_line(3)           TYPE n.

  lv_date = sy-datum.
  lv_time = sy-uzeit.

  CONCATENATE vbak-kunnr
              lv_date
              lv_time
         INTO lv_text_key.

* set database table locks
  CALL FUNCTION 'ENQUEUE_EZ_SURVEY'
```

```
    EXPORTING
      mode_zcust_satisf_t  = 'E'
      mode_zcust_satisf_tt = 'E'
      mandt                = sy-mandt
      kunnr                = vbak-kunnr
      date_last            = lv_date
      time_last            = lv_time
      text_key             = lv_text_key
    EXCEPTIONS
      foreign_lock         = 1
      system_failure       = 2
      OTHERS               = 3.

  IF sy-subrc <> 0.
    MESSAGE e001 WITH ,Locking error'.
  ENDIF.

  lo_agent_survey =  zca_survey_pers=>agent.

* create entry for zcust_satisf_t
  TRY.
      lo_agent_survey->create_persistent(
        EXPORTING
          i_kunnr      = vbak-kunnr
          i_date_last  = lv_date
          i_time_last  = lv_time
          i_text_key   = lv_text_key
      RECEIVING
        result = lv_result ).
    CATCH cx_os_object_existing.
      RAISE error.
  ENDTRY.

* create survey instance from class agent
  lo_survey = zca_survey_pers=>agent->get_persistent(
            i_kunnr      = vbak-kunnr
            i_date_last  = lv_date
            i_time_last  = lv_time
            i_text_key   = lv_text_key ).

  lo_survey->set_cust_satisf( i_cust_satisf = gv_satisfaction ).
```

```
  IF gv_modified = 1.      " only if text was changed

*   survey text processing
    lo_agent_survey_text =  zca_survey_text_pers=>agent.

*   create entry for zcust_satisf_tt
    LOOP AT gt_text INTO lv_text.
      lv_line = sy-tabix.

      TRY.
          lo_agent_survey_text->create_persistent(
            EXPORTING
              i_text_key   = lv_text_key
              i_line       = lv_line
          RECEIVING
            result = lv_result ).
        CATCH cx_os_object_existing.
          RAISE error.
      ENDTRY.

*   create survey text instance from class agent
      lo_survey_text = zca_survey_text_pers=>agent->get_persistent(
                i_text_key   = lv_text_key
                i_line        = lv_line ).

      lo_survey_text->set_text( i_text = lv_text ).
    ENDLOOP.
  ENDIF.

  CALL FUNCTION ‚DEQUEUE_EZ_SURVEY‘
    EXPORTING
      mode_zcust_satisf_t  = ‚E‘
      mode_zcust_satisf_tt = ‚E‘
      mandt                = sy-mandt
      kunnr                = vbak-kunnr
      date_last            = lv_date
      time_last            = lv_time
      text_key             = lv_text_key
      line                 = lv_line.

ENDFORM.                     " SAVE_SURVEY
```

Listing 3.4 Subroutine save_survey

The code in Listing 3.4 performs the following steps:

- Creates a text key entry by concatenating customer number, date, and time into local variable `lv_text_key`
- Sets a database lock by calling function module `ENQUEUE_EZ_SURVEY`
- Uses static class agent `zca_survey_pers=>agent` to create new persistence of the survey data using customer number, date, time, and `text_key`
- Sets the satisfaction level using persistency method `lo_survey->set_cust_satisf`
- Checks whether survey text was changed and, if so, then processes the following for each text line:
  - Uses static class agent `zca_survey__text_pers=>agent` to create new persistence of the survey text data using `text_key`
  - Uses persistency method `lo_survey_text->set_text` to set text line and text line entry
- Unsets the database lock by calling function module `DEQUEUE_EZ_SURVEY`

Create Global Declarations

Before Christine adds the new screen layout element, she needs to create all of the global declarations for the variables and objects used (see Listing 3.5). One place for this is ABAP include `MV45ATZZ`, which is reserved for global user data definitions in sales order processing.

```
* customer survey declarations
CONSTANTS: cv_line_length TYPE i VALUE 256.
DATA: go_editor           TYPE REF TO cl_gui_textedit,
      go_editor_container TYPE REF TO cl_gui_custom_container,
      gt_text             TYPE STANDARD TABLE OF char256,
      go_survey           TYPE REF TO zcl_survey,
      gv_do_survey        TYPE abap_bool,
      gv_satisfaction     TYPE zcust_satisf,
      gv_modified         TYPE i.
```

Listing 3.5 Customer Survey Declarations

Note

You've probably noticed that in Chapter 2, Christine did not use ABAP include `MV45ATZZ` for her declarations, but used Enhancement Point `MV45ATOP_10` within ABAP include `MV45ATOP` instead. Both approaches are acceptable. There are sometimes multiple ways to achieve the same result for SD enhancements.

3.3.7 Adding Layout Changes to Screen 8309 of SAPMV45A

Finally, Christine turns to screen 8309 of ABAP include `SAPMV45A` itself. She adds a dropdown box (`GV_SATISFACTION`) and a custom container (`CONTAINER`) to the screen, using the layout attributes shown in Figure 3.11.

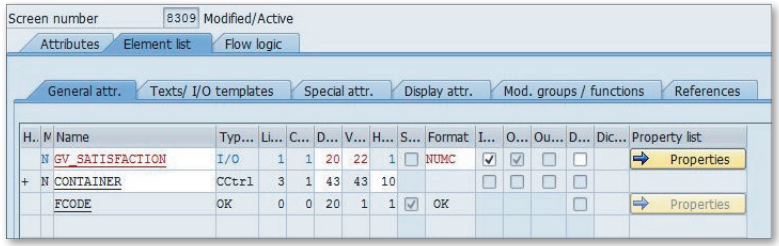


Figure 3.11 Layout Parameters for the Container and the Editor Box in Screen 8309

This concludes all necessary changes. The final result of their efforts can be seen in Figure 3.12, where a satisfaction dropdown box and the text editor are displayed in the ADDITIONAL DATA B tab in screen 8309.

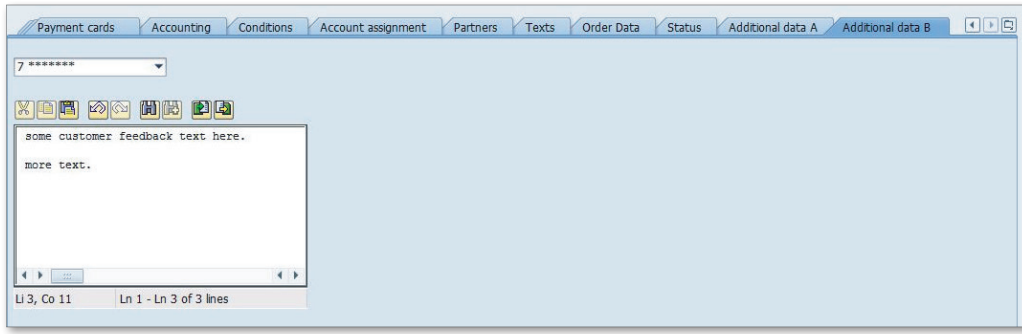


Figure 3.12 Custom Survey Screen in Sales Order Create Transaction VA01

You can also see that the editor class (`go_editor`) contains additional goodies such as Cut/Copy/Paste and Upload/Download buttons, which come as part of the editor class object.

3.4 Summary

This chapter showed you a complete end-to-end example for a small customer survey enhancement using the freely customizable tabs in the sales order capture Transactions VA01/02/03.

To accomplish this, Christine and Sean used early enhancement techniques such as a screen exit, modification, and user exit, as well as modern methods such as implicit enhancements and an enhancement point.

Under the hood, Christine's code made use of persistent objects, which manage the entire database interaction for her. All she had to do was define her survey and text tables in the data dictionary and then assign them to a persistency class. All of the methods that interact directly with the database were generated for her. All she had to do was create a data model class (`ZCL_SURVEY`) that calls on the persistence management classes and agents.

Should Byrell ever decide to extend this survey solution, Christine will be able to build on the persistency layer she has created and regenerate the necessary methods, which is a big benefit.

In the next chapter, you'll learn more about finding suitable enhancements using an iterative search process when dealing with SD order billing.

Finding the right enhancement can be a lengthy process. To find the right one, it's critical to first know the requirements, and also know when and where to execute the enhancement.

4 Creating an SAP CRM Activity after SD Order Billing

In real-life situations, when a consultant needs to find an enhancement to meet a business requirement, he very often starts with a simple search in the Repository Infosystem, for example. If the offered standard enhancement options don't fulfill the business requirements, then the consultant has to investigate deeper around the standard code that is called to find other options such as implicit enhancements or explicit enhancement spots.

This chapter provides an example of this iterative process. Sean and Christine go through many enhancement options before they find the best approach to automatically trigger the creation of an SAP CRM activity after a sales order was billed in SD. After some time spent searching, they'll provide additional functionality to some of Byrell's existing customer service tools.

4.1 Business Scenario

The customer services department at Byrell wants to be more proactive when it comes to customer delivery performance. Its managers are asking for a solution that would enable Byrell's call center staff to individually call specific customers to find out how well Byrell's delivery and billing process has worked. The department already uses functionality within SAP Customer Relationship Management (SAP CRM) to create activities for business partners to keep on top of their tasks.

Sean runs a workshop for the customer services team during which he learns that a future solution should really leverage existing functionality around SAP CRM activities. This would fit best with the customer service department's existing

activities and would enable the team to organize and plan the outgoing customer calls using existing tools.

During the workshop, the team agrees that a SAP CRM activity should be created for only *some* of the sales orders that are billed successfully. The customer service department wants to contact some customers before the invoice is sent, rather than after. The customer services team believes this is the most proactive and professional solution.

Furthermore, created activities should contain the business partner and sales document number, which are important when contacting the customer.

After the workshop, Sean reflects on his findings and thinks that a solution might be required that triggers the creation of an activity before billing has taken place.

4.2 Finding the Right Enhancement

Sean asks Christine to investigate the system to find the different enhancement options within billing because she is more experienced with the Enhancement Framework. As a guide, he gives her a "must-have" list for the enhancement:

- Called immediately after billing has been executed
- Read access to invoice header

Christine uses the Repository Infosystem (Transaction SE80) to search for suitable BAdIs first. She first opens the Business Add-Ins node and then double-clicks Definitions in the left column. On the following screen, she enters "VF_BADI" in the Package field and executes the search (see Figure 4.1).

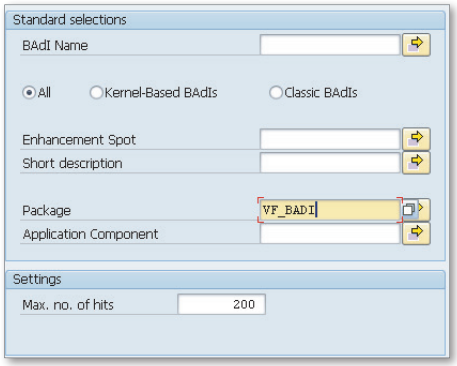


Figure 4.1 Search for All BAdIs within Package VF_BADI

When she clicks on the arrow button to the right of the field, the result set includes a range of billing-related BAdIs (see Figure 4.2).

Name of a BAdI Definition	Enhancement Spot	Description
BADI_SD_ACCOUNTING	BADI_SD_ACCOUNTING	
BADI_SD_ADV_RET	BADI_SD_ADV_RET	BADI for advanced returns management
BADI_SD_BILLING	BADI_SD_BILLING	
BADI_SD_BILLING_ITEM	BADI_SD_BILLING_ITEM	
BADI_SD_GM	BADI_SD_GM	
BADI_SD_PEROP	BADI_SD_PEROP	Period of Performance
BADI_SD_VF48	BADI_SD_VF48	
EHP_BADI_BIL_CCARD	EHP_SD_BILLING_CREDITCARD	BADI for Authorizing Credit Cards in Billing Documents

Figure 4.2 All BAdIs within Package VF_BADI

However, after double-clicking on `BADI_SD_BILLING`, Christine notices that it's flagged as Can only be implemented SAP-internally (Figure 4.3) under the Enh. Spot Element Definitions tab, which means it can't be used by customers. Christine and Sean will have to find a different BAdI or extension.

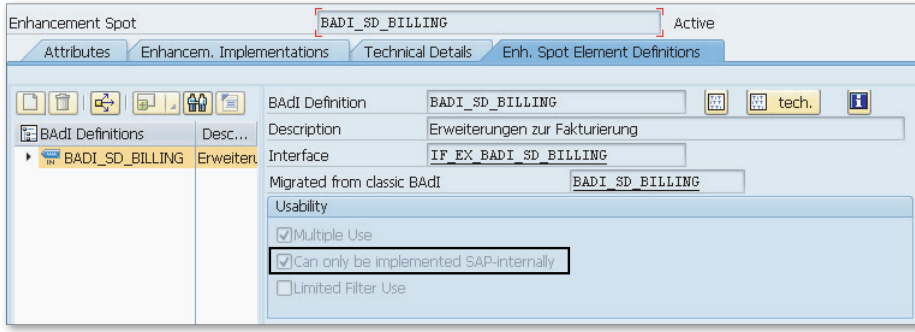


Figure 4.3 BADI_SD_BILLING Flagged for SAP Internal Use Only

The other BAdIs that came up in her search within the Repository Infosystem are also either for internal SAP use or not suitable for this particular exercise, as the BAdI descriptions reveal.

Christine therefore turns her attention to billing-related customer exits by clicking on the Customer Exits – Enhancements on the left side of the Repository Infosystem. She once again searches for all objects within the billing package VF. All found customer exits are for very specific parts of the billing and invoicing process, but customer exit `EXIT_SAPLV60A_002` stands out because it addresses Customer Functions in the Billing Document (Figure 4.4).

Exit name	Short Description
J_3RSINV	
SDVFX001	User exit header line in delivery to accounting
SDVFX002	User exit for A/R line in transfer to accounting
SDVFX003	User exit cash clearing in transfer to accounting
SDVFX004	User exit G/L line in transfer to accounting
SDVFX005	User exit reserves in transfer to accounting
SDVFX006	User exit tax line in transfer to accounting
SDVFX007	User exit: Billing plan during transfer to Accounting
SDVFX008	User exit: Processing of transfer structures SD-FI
SDVFX009	Billing doc. processing KIDONO (payment reference number)
SDVFX010	User exit item table for the customer lines
SDVFX011	Userexit for the komkcv- and kompcv-structures
V05I0001	User exits for billing index
V05N0001	User Exits for Printing Billing Docs. using POR Procedure
V60A0001	Customer functions in the billing document
V60P0001	Data provision for additional fields for display in lists
V61A0001	Customer enhancement: Pricing

Figure 4.4 Customer Exits Assigned to Package VF

She double-clicks into it and then navigates through to the source code of the function module `EXIT_SAPLV60A_002` by once again double-clicking on it (see Figure 4.5).

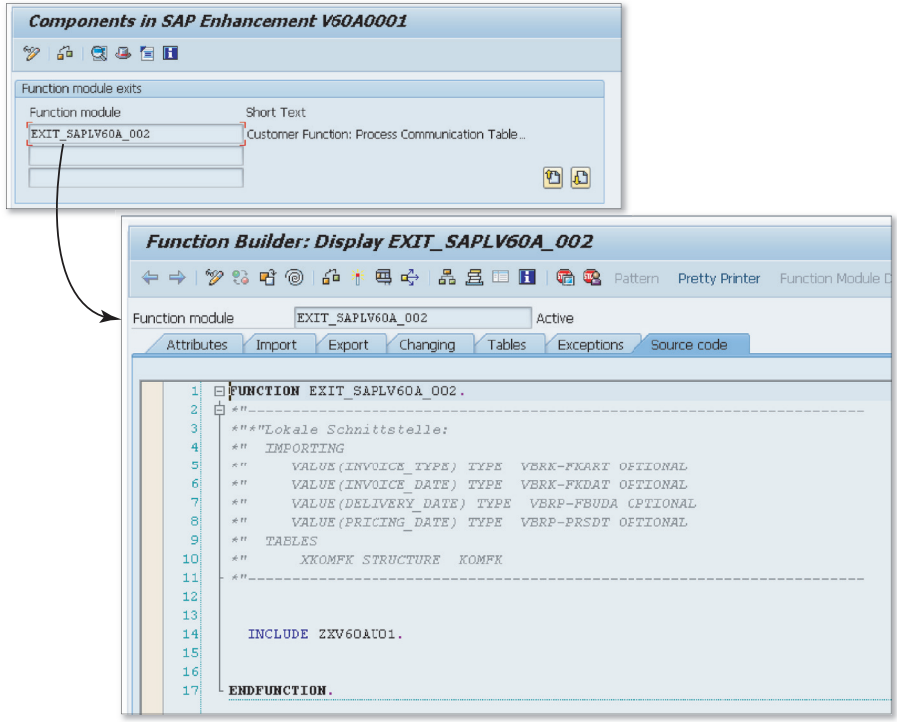


Figure 4.5 Navigating from Customer Exit to Function Module

Christine now needs to know where in the billing document process this customer exit is called. This is of particular importance because the SAP CRM activity needs to be created after billing has been executed successfully. In other words, the point at which the remote function call (RFC) module is called is crucial here. She performs a where-used search (see Figure 4.6), limiting it to Programs and Classes to show her where the customer exit has been placed in the standard code.

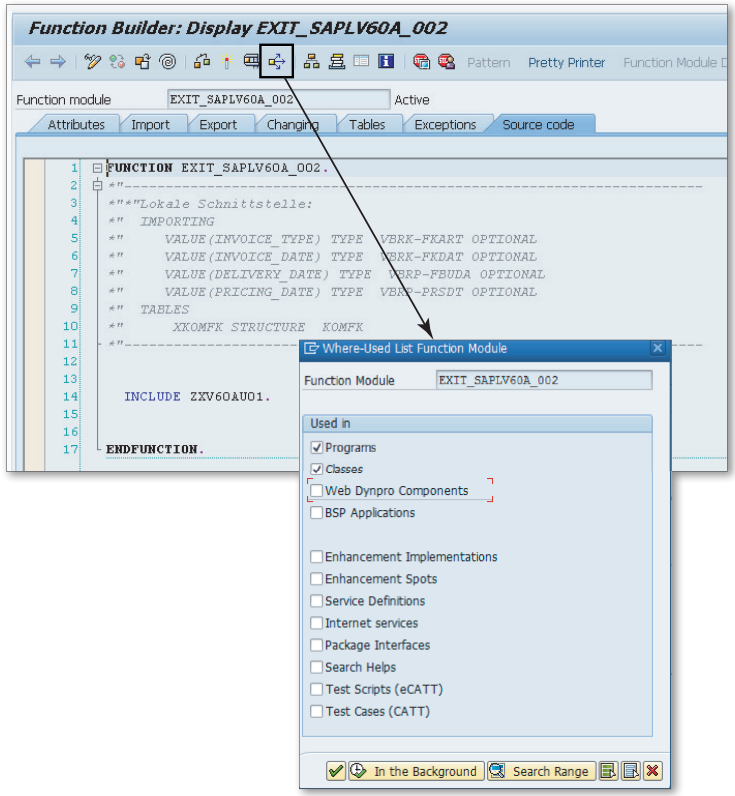


Figure 4.6 Where-Used Search for Function Module EXIT_SAPLV60A_002

She looks at the result by double-clicking on the module description, which shows her the Function Builder: Display RV_INVOICE_CREATE screen. The result set is a single entry for Program LV60AU01, which is a part of function group V60A and links to Function Module RV_INVOICE_CREATE (Figure 4.7).

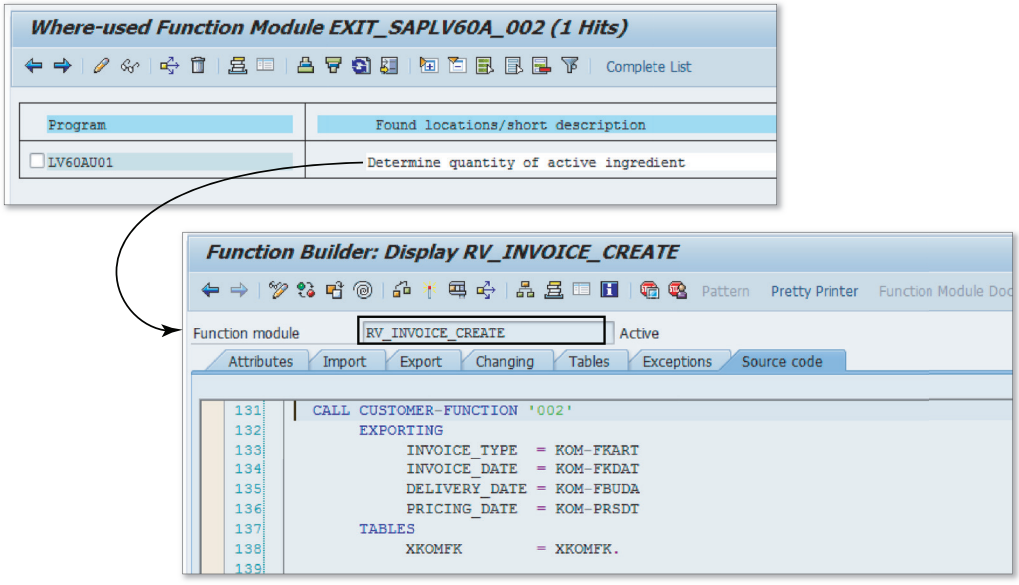


Figure 4.7 Double-Click on the Description Navigates to RV_INVOICE_CREATE

Note

This development task needs to create an SAP CRM activity after billing is performed successfully. Therefore, there is good news and bad news. First, the bad news: the customer exit `EXIT_SAPLV60A_002` isn't going to serve this particular task because it's called *before* the document is fully processed and committed to the database. The good news is that `RV_INVOICE_CREATE` has to be the place where the enhancement will eventually be added, but Sean has to find the best location for it.

Christine looks further down in `RV_INVOICE_CREATE` and can see other SAP enhancements and even method calls for BAdI `BADI_SD_BILLING` (Figure 4.8). But as she and Sean found out earlier, this can't be used because it's flagged as SAP internal.

Christine speaks to Sean, who after some analysis of function module `RV_INVOICE_CREATE`, suggests creating an implementation for an enhancement spot she found right after `RV_INVOICE_REFRESH` has been called, where she could evaluate `TABLE` parameter `XVBFS` (see Figure 4.9).

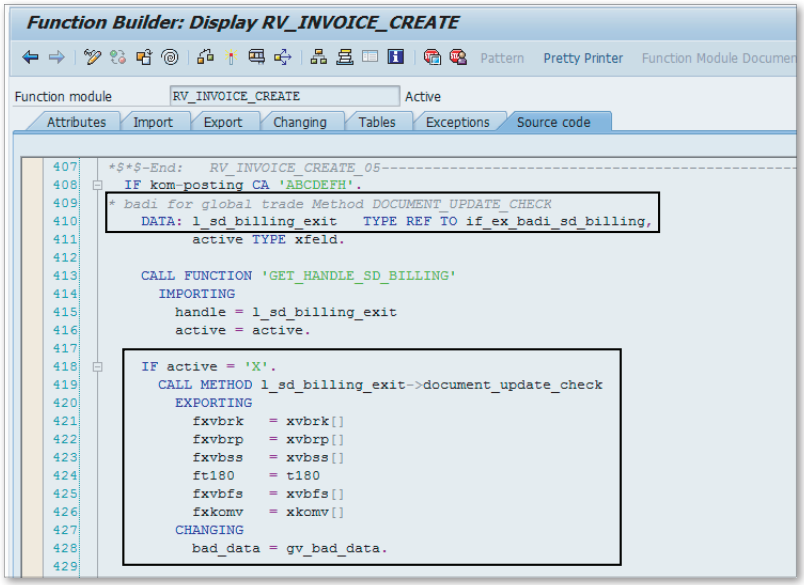


Figure 4.8 Billing BAdI Method DOCUMENT_UPDATE_CHECK in Standard Code

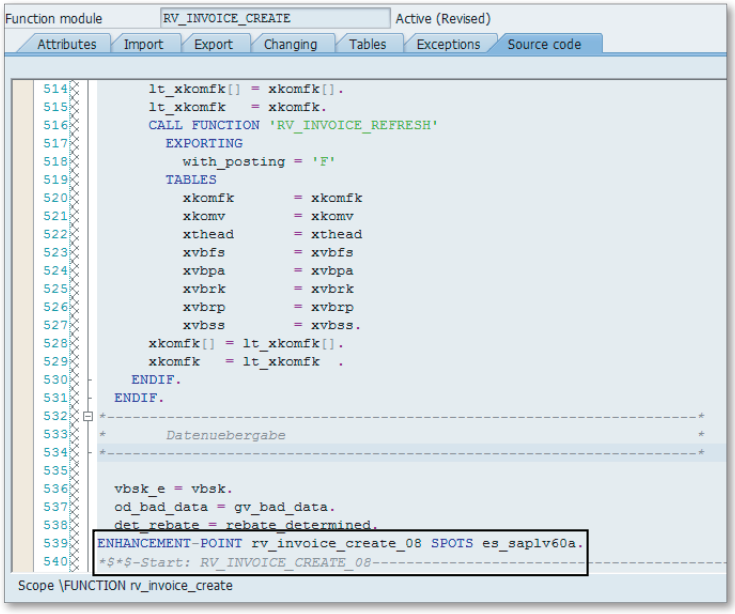


Figure 4.9 Enhancement Point RV_INVOICE_CREATE_08 after Function Module Call RV_INVOICE_REFRESH

This internal table holds the error log that the function module passes back to its caller, which should give reliable information regarding whether or not billing was performed successfully.

Function module `RV_INVOICE_REFRESH` is the central function module that encapsulates the code responsible for committing data to the database, so it's the perfect place to insert the RFC to SAP CRM.

Now that Christine and Sean have found the right place to implement the RFC function module, it's important to consider some basic facts about RFCs.

4.3 Communication between SAP Systems via RFC

One way for SAP ERP and SAP CRM to communicate is to establish an RFC link between the two systems. This section briefly touches on the basics of RFC connections and how to create an RFC-enabled function module.

What Is a Remote Function Call?

A remote function call (RFC) is an execution of a function module in a system different from the caller's. The remote function can also be called from within the same system (as a remote call), but usually the caller and receiver will be in different systems.

In the SAP system, the ability to call remote functions is provided by the RFC interface system. An RFC allows for remote calls between two SAP systems (R/3 or R/2), or between an SAP system and a non-SAP System.

Source: SAP Help

4.3.1 ABAP-to-ABAP Communications

Any ABAP program can call a remote function using the `CALL FUNCTION...DESTINATION` statement. The `DESTINATION` parameter tells the SAP system that the called function runs in a system other than the caller's. RFC communication with the remote system happens as part of the `CALL FUNCTION` statement.

RFC functions running in an SAP system must be actual function modules and must be registered in the SAP system as a Remote-Enabled Module (see Figure 4.10).

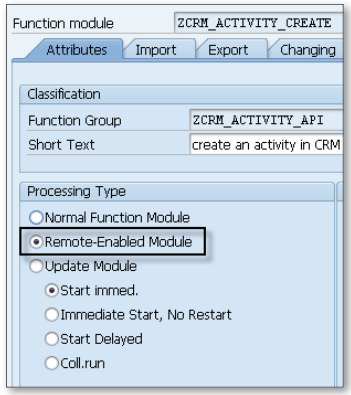


Figure 4.10 Declaration of an ABAP Function Module as Remote

4.3.2 Setting Up RFC Connections

To connect two SAP systems using the RFC interface, you must maintain the technical parameters and security settings in Transaction SM59 (Configuration of RFC Connections) as shown in Figure 4.11.

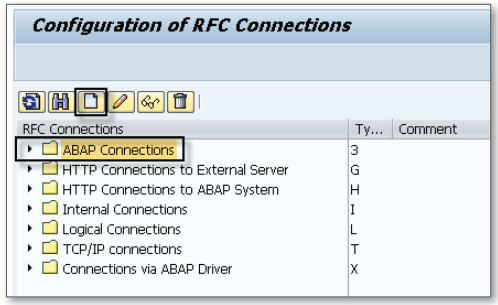


Figure 4.11 ABAP Connections in Transaction SM59

Follow these steps to create an RFC connection:

1. Access Transaction SM59 and click the CREATE button.
2. On the following screen, populate the relevant RFC destination, connection type (which in this case is "3" for ABAP connection), description, and system number. Refer to your BASIS administrator if you're unsure about any of these fields within your own system landscape. See Figure 4.12 for details of the TECHNICAL SETTINGS tab.

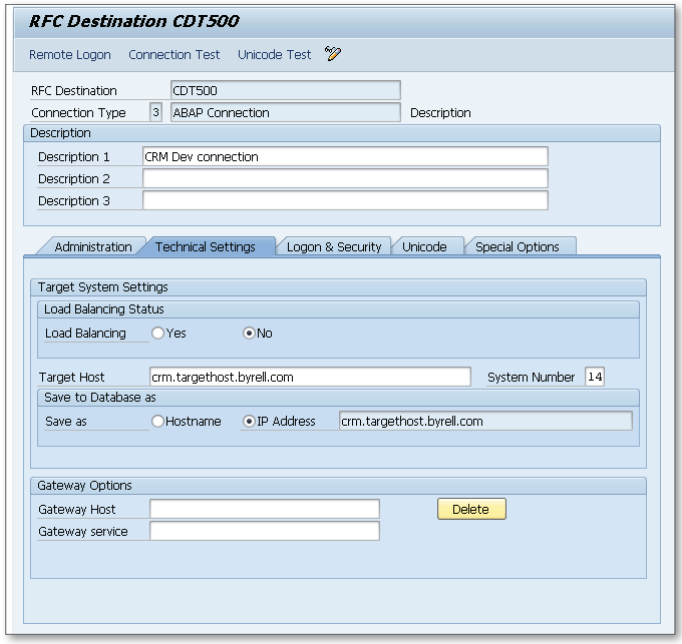


Figure 4.12 Technical RFC Settings in Transaction SM59

3. Specify the settings for logon and security by selecting the Logon & Security tab and ticking the Current User box (see Figure 4.13). On the one hand, this ensures that the function module call in the target system is executed by the same user ID that calls it from the source system. On the other hand, it means that the user ID in question has to be maintained in both systems. Consult your BASIS administrator if you're unsure about this.

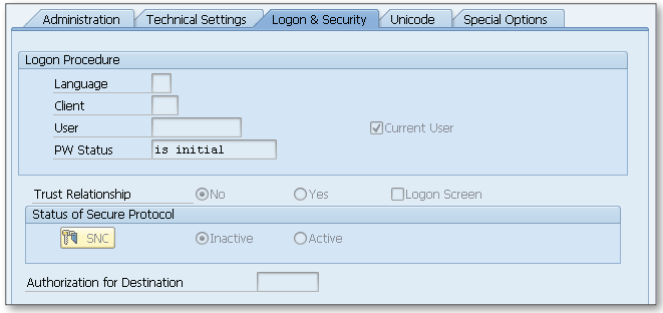
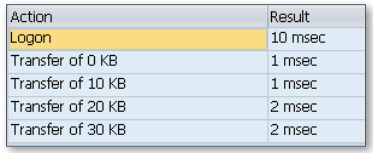


Figure 4.13 Logon & Security Tab in Transaction SM59

4. Next, save your changes, and then test your connection setup using the Connection Test button in the menu bar.

   After a few seconds, the following test log should appear, indicating that the connection is working fine (see Figure 4.14). You now have a working RFC connection.



Action	Result
Logon	10 msec
Transfer of 0 KB	1 msec
Transfer of 10 KB	1 msec
Transfer of 20 KB	2 msec
Transfer of 30 KB	2 msec

Figure 4.14 RFC Connection Test Log Results

General Note on RFCs

Check with your BASIS administrator before setting up any RFC destinations in your own system to see if there is an existing destination in your landscape you can use.

Covering RFCs in their entirety is beyond the scope of the book; for more information on this topic, please refer to the following resources:

- SAP Help (*http://help.sap.com*)
- *SAP Interface Programming* by Michael Wegelin and Michael Englbrecht (SAP PRESS 2010)

4.4 Implementing the Solution

Christine now has all of the information she needs to start her work. Together with Sean, she has created an action list for the realization phase, covering all of the necessary steps for this development:

1. Create a new function group in the SAP CRM system.
2. Create an RFC function module in the SAP CRM system.
3. Specify RFC function module parameters.
4. Implement ABAP logic to create an SAP CRM activity.
5. Create the enhancement point implementation in function module `RV_INVOICE_REFRESH`.
6. Implement ABAP logic to call the RFC function module in the SAP CRM system.

We'll go over each of these steps in the following sections.

4.4.1 Creating a New Function Group in the SAP CRM System

The new RFC function module should be held in a separate function group from all other custom function modules, so Christine's first task is to create a new function group. On the first screen of Transaction SE37, she selects Goto • Function Groups • Create Group.

In the following popup, she enters a function group name ("ZCRM_ACTIVITY_API") and a short text for the group. She then clicks on Save and specifies the development class and change request in the following popup.

4.4.2 Creating an RFC Function Module in the SAP CRM System

Christine then creates a new RFC function module in Transaction SE37. She enters "ZCRM_ACTIVITY_CREATE" as the name and clicks on the Create button (see Figure 4.15).

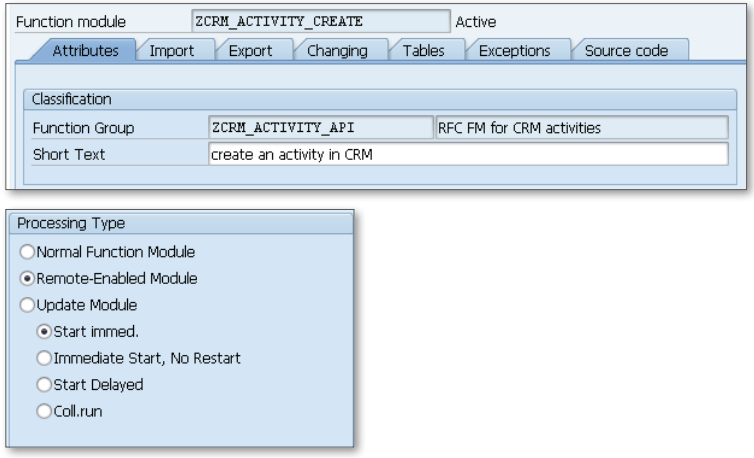


Figure 4.15 Create New Function Module ZCRM_ACTIVITY_CREATE

In the following popup window, she enters the function module name "ZCRM_ACTIVITY_CREATE" and the newly created function group "ZCRM_ACTIVITY_API". In addition, she also specifies a short text to describe this new function module.

She then goes to the Attributes tab and changes the function module's processing type from Normal Function Module to Remote-Enabled Module.

4.4.3 Specify Function Module Parameters

Next, Christine declares the function module parameters and exception. First, she selects the Import tab and adds the import parameters as shown in Figure 4.16.

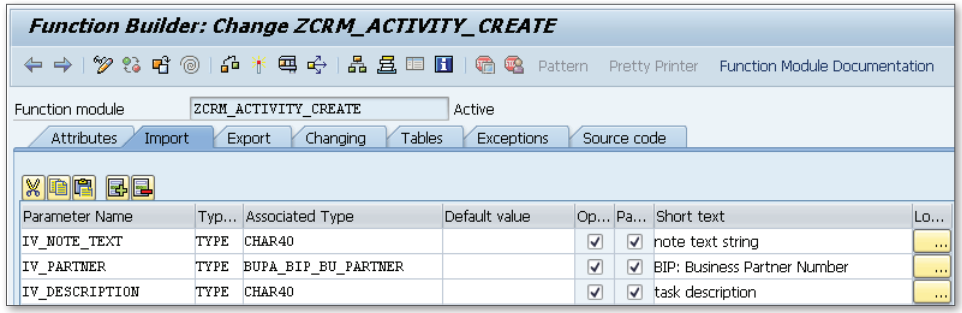


Figure 4.16 Import Tab of Function Module ZCRM_ACTIVITY_CREATE

The RFC module also needs a way of reporting back to the caller regarding whether a problem occurred. Therefore, Christine also defines an Error exception in the Exceptions tab and gives it a fitting description.

4.4.4 Implement ABAP Logic to Create an Activity in SAP CRM

When it comes to triggering business logic inside an SAP system, Sean has used Business Application Program Interfaces (BAPIs) in the past and wants to know why Christine created a separate RFC function module for this development.

Christine explains that she intends to make use of the Business Object Layer (BOL) programming interface within SAP CRM to make creating an activity a lot simpler than using classic BAPIs, which have a much broader and complex interface.

Using the Business Object Layer

BOL is a simple, easy-to-use application programming interface (API), designed specifically for SAP CRM. However, you can also use BOL programming for your own solutions, leveraging existing methods for querying the database or creating, changing, or deleting data. The easiest way for Christine to explain BOL

programming to Sean is by talking him through the code she has already created for this particular task.

BOL programming is based on the BOL core, which requires the instantiation of class `CL_CRM_BOL_CORE` as shown in Listing 4.1.

```
DATA:   lo_core        TYPE REF TO cl_crm_bol_core,
        lo_factory     TYPE REF TO cl_crm_bol_entity_factory,
        lo_entity      TYPE REF TO if_bol_bo_property_access,
        lo_order       TYPE REF TO cl_crm_bol_entity.

lo_core = cl_crm_bol_core=>get_instance( ).
lo_core->start_up( 'ALL' ).
lo_factory = lo_core->get_entity_factory( 'BTOrder' ).
lo_order = lo_factory->create( lt_params ).
CHECK lo_order IS BOUND.
```

Listing 4.1 Instantiating the BOL Core Object

Local class `lo_core` is instantiated using static method `get_instance()`. Class `lo_core` then "starts up" the component sets, which are basically business object groups holding different business entities. The entity that Christine is interested in is `BTOrder`, which is the first building block of her activity object. She then uses the factory `create()` method to instantiate the `lo_order` object, which is referenced to a BOL entity object (`CL_CRM_BOL_ENTITY`).

The `lo_order` object is the next stepping stone to build Christine's activity object, which is used to instantiate local class `lo_orderh`. This object is created as a child of `BTOrder` (see Listing 4.2).

```
* create the order header object
  lo_orderh = lo_order->get_related_entity( iv_relation_name =
'BTOrderHeader').
  IF lo_orderh IS NOT BOUND.
    TRY.
      lo_orderh = lo_order->create_related_entity(
              iv_relation_name = 'BTOrderHeader').
      CATCH cx_crm_genil_duplicate_rel cx_crm_genil_model_error.
    ENDTRY.

    lo_orderh->set_property(
```

```
  iv_attr_name = 'DESCRIPTION' iv_value = iv_description ).
ENDIF.
```

Listing 4.2 Creating BOL Relationships and Setting Attributes

BOL also allows Christine to set and get properties of object attributes. In the preceding example, the `set_property()` method sets the description of the SAP CRM activity, for example.

In this way, the BOL allows Christine to set all necessary attributes of a SAP CRM business object programmatically.

4.4.5 Create the Enhancement Point Implementation in Function Module RV_INVOICE_REFRESH

Christine now turns her attention to the code in the SAP ERP system that calls the RFC function module. First she has to create an enhancement implementation at the end of function module `RV_INVOICE_CREATE`. Christine selects the Enhancement button (Figure 4.17) and then right-clicks on the `ENHANCEMENT-POINT` to create an implementation by choosing Enhancement Operations • Create Implementation.

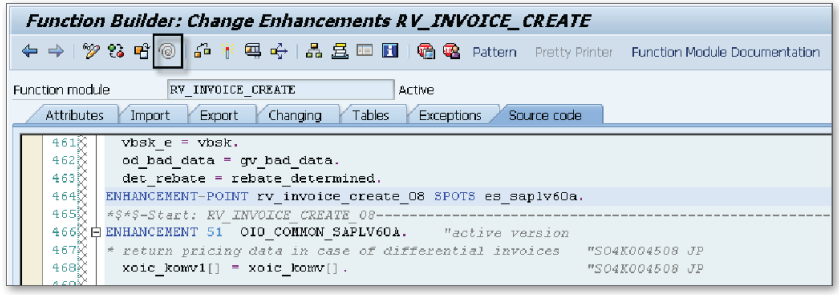


Figure 4.17 Enter Enhancement Mode by Clicking the Enhancement Button

In the following popup, Christine clicks on the New button to create a new enhancement implementation. She calls the implementation "ZENH_RV_INVOICE_CREATE", gives it a meaningful title, and confirms the screen.

She then selects the enhancement implementation she has just created and clicks on the Confirm (green tick) button.

This takes Christine back to the editor, showing a new, empty enhancement implementation (see Figure 4.18).

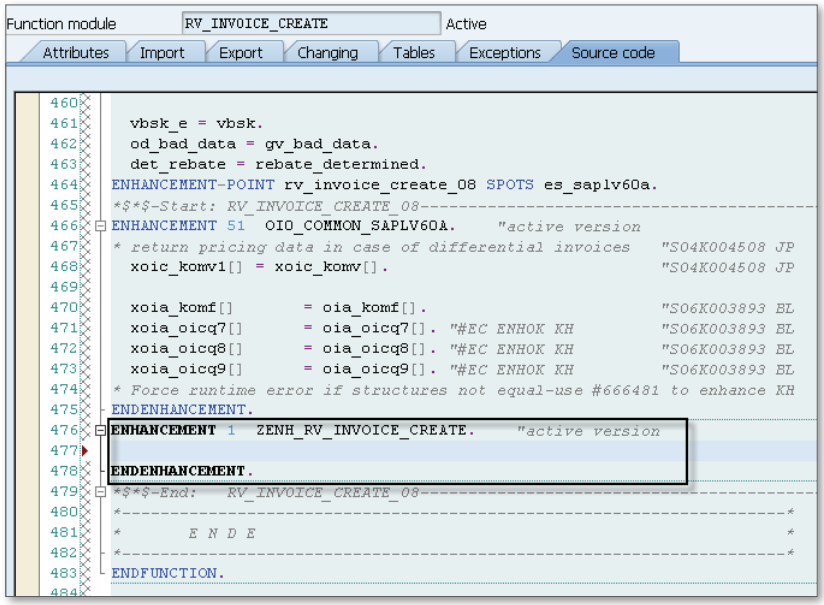


Figure 4.18 Enhancement ZENH_RV_INVOICE_CREATE in function module RV_INVOICE_CREATE

She adds the coding shown in Listing 4.3 to the newly created enhancement.

```
DATA: ls_xvbpa TYPE vbpa,
      ls_xvbfs TYPE vbfs.

LOOP AT xvbfs INTO ls_xvbfs.
  AT NEW vbeln.
*   only if success message was issued
    IF ls_xvbfs-msgty <> 'S'.
      CONTINUE.
    ELSE.
      LOOP AT xvbpa into ls_xvbpa WHERE vbeln = ls_xvbfs-vbeln
                    AND parvw = 'RG'.
        CALL FUNCTION 'ZCRM_ACTIVITY_CREATE'
          DESTINATION 'CDT500'
          EXPORTING
           iv_note_text   = 'Courtesy call after billing'
```

```
          iv_partner     = ls_xvbpa-kunnr
          iv_description = 'Customer Activity'
        EXCEPTIONS
          error        = 4.

      IF sy-subrc <> 0.
        MESSAGE i017(zcrm) WITH 'Could not create CRM activity'.
      ENDIF.
      EXIT.                  " only create 1 message/kunnr
    ENDLOOP.
  ENDIF.
ENDAT.
ENDLOOP.
```

Listing 4.3 Code Calling

This code loops around the billing error log message Table XVBFS. For each new billing document found, it checks whether billing was successful (message type = "S"). Only if a billing was executed successfully for this document will the code continue to find the corresponding billing payer (partner role RG) partner entry in Table XVBPA.

Christine then calls the RFC function module `ZCRM_ACTIVITY_CREATE` in destination system CDT500 and passes a note text, payer number, and an activity description to the function module. Any exceptions are reported back with an info message, which is more appropriate in this case because billing itself was executed successfully and an issue would have only occurred when creating the activity.

4.5 Summary

In this chapter, you learned how an enhancement can be used to reach out to another SAP system and create notifications. Most importantly, this chapter also highlighted the importance of knowing what you want to achieve with the enhancement and knowing when you want to execute your enhancement.

You probably noticed that Sean's search initially started with BAdIs, then led to customer exits, and eventually ended up with an explicit enhancement implementation. This occurred because none of his initial searches provided the best place within the billing process to place custom code.

You should now have an understanding of the iterative process of finding the right enhancement, which can involve time and determination.

BAdIs can easily be applied to inbound and outbound web services, and potentially minimize the amount of enhancement impact you're required to make.

5 Filtering Pricing Data within a Web Service

Since upgrading its systems to SAP NetWeaver 6.40 back in 2006, Byrell Corporation has worked toward a service-oriented architecture (SOA) and made use of enterprise services in SAP.

One of the first interfaces developed during a project in 2007 enabled Byrell's clients to develop their own frontend application for capturing sales orders with non-SAP user interfaces. By exposing sales order services this way, clients were able to create a simpler ordering application, for example, thus prompting users for fewer input fields when raising a sales document in the SAP backend. In addition, SOA enabled Byrell to increase the reuse of development components, increasing cost efficiency and promoting more agile development.

These sales order services exposed functions (called operations) allowed for the following:

1. Creation of sales orders
2. Change of existing sales orders
3. Reporting and display of sales order data

Within this chapter, Sean and Christine have to apply a change to the *Sales Order Create* enterprise service in order to limit some of the data that the service currently exposes to the outside world. To do this, they will use a BAdI that has been generated in the service interface.

You'll see how to manipulate outbound web service data using a BAdI. A web service can be an external interface providing data to the outside world. You'll also learn

some basics about service-oriented architecture (SOA), browse SAP ERP services, and then implement a BAdI to filter pricing data from the service signature.

Together with Sean and Christine, let's get started and learn a little about enterprise services and how to find and enhance them with BAdIs.

5.1 Background on SOA and Enterprise Services

Before going further into Christine and Sean's latest development request, it's important to cover a few basic facts and ideas around web services and how they're handled in SAP. Feel free to skip this section if you already know how to use enterprise services as a way to achieve SOA.

5.1.1 Service-Oriented Architectures

A central paradigm of SOA is the principle of separated, independent components that provide a functionality as a service. These services are combined and reused to create applications. Applications can be *service providers* or *service consumers*. Web services are one type of service provider and are probably the best suited to implement and achieve SOA because they provide and consume services independent of a programming language. Imagine web services as a language bridge between two individuals, enabling them to communicate, regardless of their origin and mother tongue.

5.1.2 Characteristics and Goals of SOA

In the following subsections, we'll discuss the means that SOA uses to achieve the goals of agility and reusability.

Split Functionality into Components

By designing software in a way that splits it up into components of functionality, you achieve a higher degree of exchangeability and reusability, which results in cost savings. Examples of this include a service that creates a sales order or a separate service that changes a sales order. You probably know this principle already from BAPIs in ABAP, which encapsulate parts of SAP backend functionality. In the

remainder of Section 5.1, you'll discover more details on the similarities and differences between web services and BAPIs.

Loose Coupling

Loose coupling allows reuse of services in a vast number of different consumer technologies and is achieved by the following:

- **Separating the interface and the implementation of the service (design by contract)**
  This gives both parties greater flexibility because with an agreed interface, the underlying implementation of how to provide and consume the data is left to the individual.
- **Using a technology-independent language description of the interface**
  This language is called *Web Services Description Language (WSDL).*
- **Providing the option to use services registries**
  Comparable to the yellow pages, services are published here to allow consumers to find suitable services. A platform-neutral standard called *Universal Description, Discover and Integration (UDDI)* supports this.

Syntactic (or Technical) Interoperability

The capability for the provider and consumer to communicate with each other when required must be ensured, despite all decoupling. Open standards, guaranteed quality, and stability have to ensure this.

Semantic Interoperability

This important aspect of SOA is actually a point that can't be fully resolved by standards and web services. It deals with the meaning, behavior, and effects of a service. Let's look at this problem a little further.

Example

Imagine that you're staying in a hotel in China and want to rent a car. You are certainly able to call the reception desk and explain your request. The technical interoperability between the telephone systems is excellent; you might as well make the phone call from your home (the reception corresponds to the services registry).

Let's assume that your dialogue partner speaks English; you still need to consider whether you both perceive "rent a car" in the same way. In China, it could mean that you rent a car that includes a chauffeur; that is, the semantic of the service is different from what you actually want. Even if "rent a car" means the same as in the West, you should be able to answer the question about the required car category, and so on; that is, you must understand the meaning of the input parameters of the service (see Figure 5.1).

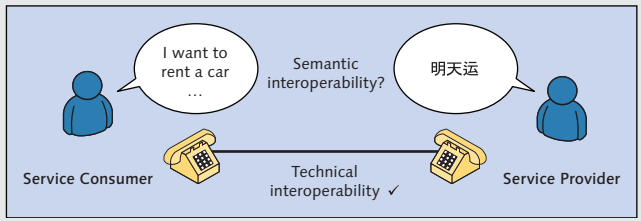


Figure 5.1 Difference between the Technical and Semantic Interoperability (Source: *Developing Enterprise Services for SAP*, SAP PRESS 2009)

5.1.3 Enterprise Services and Web Services

Enterprise services and web services are very often used ambiguously, so this section will clarify the differences between the two.

SAP defines an enterprise services as "a callable entity that provides the business logic and that is published by SAP in the Enterprise Services Repository (…)." Enterprise services describe real-world business processes with all data types, objects, and processes. They embody a higher level of standardization, meaning that there is less and less proprietary integration between components.

On the other hand, web services represent the underlying technical principle for the implementation of enterprise services. They are the technical realization of enterprise services.

Figure 5.2 shows the increased standardization and harmonization starting with RFCs and BAPIs, then web services, and finally enterprise services in terms of technical standardization and behavior.

After these rather theoretical considerations, let's have a look in Transaction SE80 and find enterprise services within an SAP system.

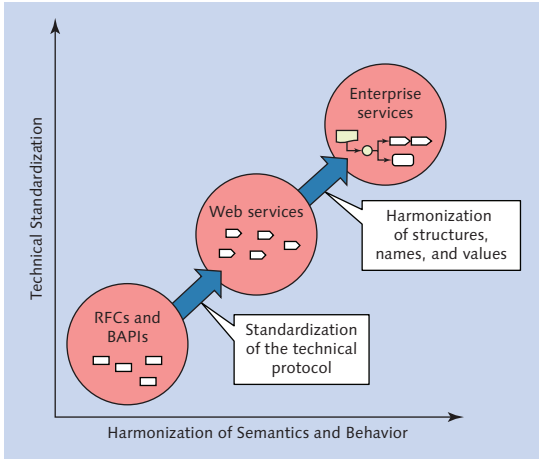


Figure 5.2 Ongoing Standardization and Harmonization for the Development of BAPIs to Enterprise Services (Source: *Developing Enterprise Services for SAP*, SAP PRESS 2009)

5.1.4 Finding Enterprise Services

You can use the Repository Infosystem (Transaction SE80) to search, view, create, and amend enterprise services. Within the Enterprise Services node, double-click on Service Definitions, enter "ECC_SalesOrder002QR" in the Service Definition field, and press the Execute (F8) button. The result set contains one single service definition for ECC_SalesOrder002QR, a service to search sales documents by ID (document number)—see Figure 5.3 for details.

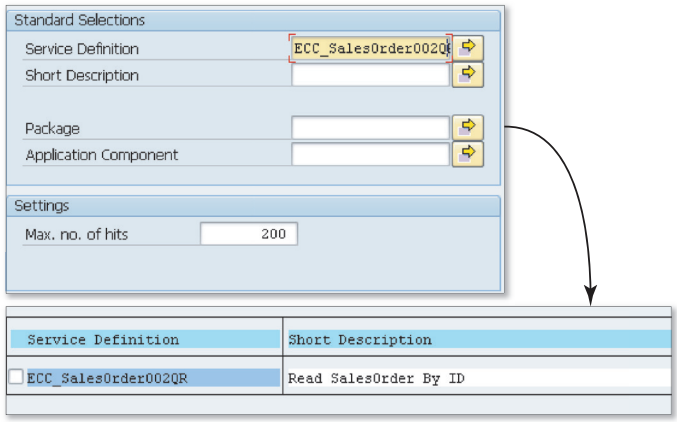


Figure 5.3 Navigating to the Correct Service Definition

If you double-click on the service definition, the detail screen shown in Figure 5.4 is displayed.

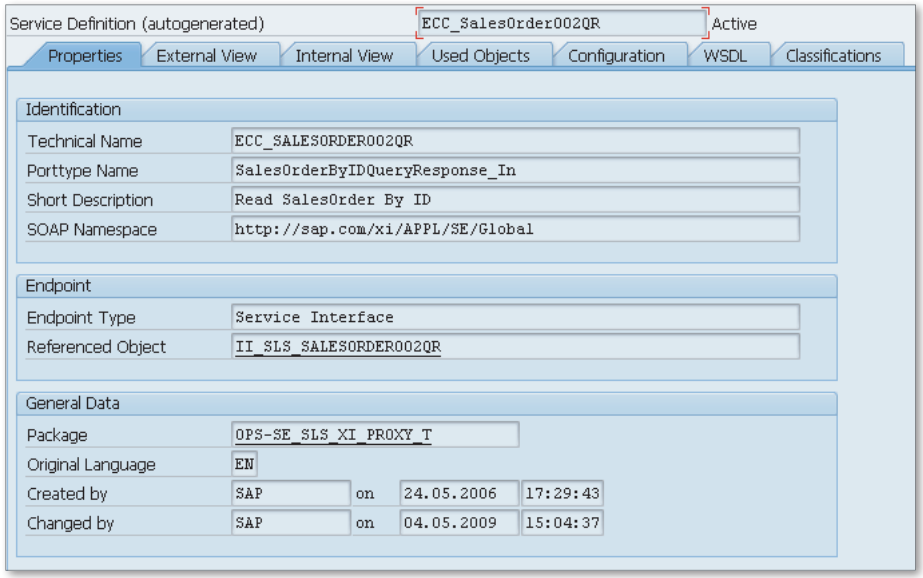


Figure 5.4 Details of Service Definition ECC_SalesOrder002QR

On closer inspection of the service definition, you probably recognize some of the details about enterprise services mentioned in Section 5.1.1, Section 5.1.2, and Section 5.1.3. In particular, the tabs WSDL (where the service definition is displayed in WSD language; see Figure 5.5) and Classifications (containing the service's registry entry details) will probably remind you of some of the underlying SOA characterizations. This shows how closely SAP tried to stick to the SOA script when it comes to the actual realization of services in the system.

The External View and Internal View tabs provide more information about the service interface and signature from an ABAP and data model perspective. In the Used Objects tab, you find a list of all the data types the services uses. The Configuration tab includes more information about security, access, and service settings (see Figure 5.6).

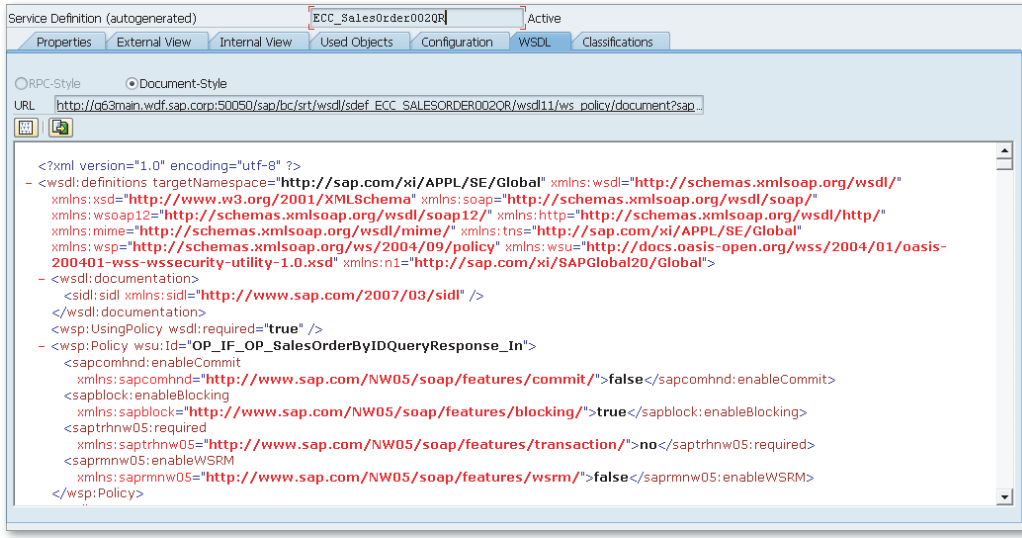


Figure 5.5 Service Definition ECC_SalesOrder002QR in WSDL Format

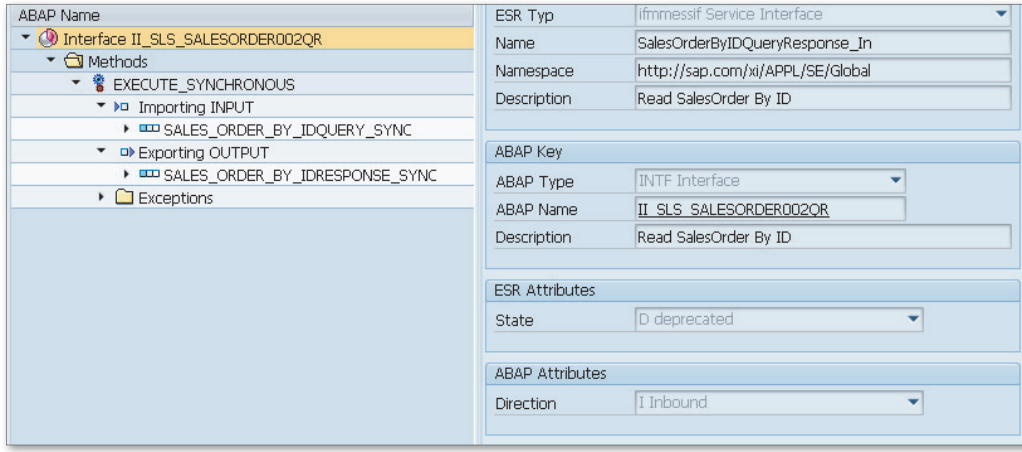


Figure 5.6 Internal View Tab with Signature Details for Enterprise Service ECC_SalesOrder002QR

Before moving on with the business scenario, let's have a look at the actual implementation of this service. By double-clicking on the Referenced Object II_SLS_SALESORDER002QR in the Endpoint screen section, you navigate to the actual service interface SalesOrderByIDQueryResponse_In. This service interface reveals the provider class implementation to us, which is called CL_SLS_SALESORDER002QR (see Figure 5.7).

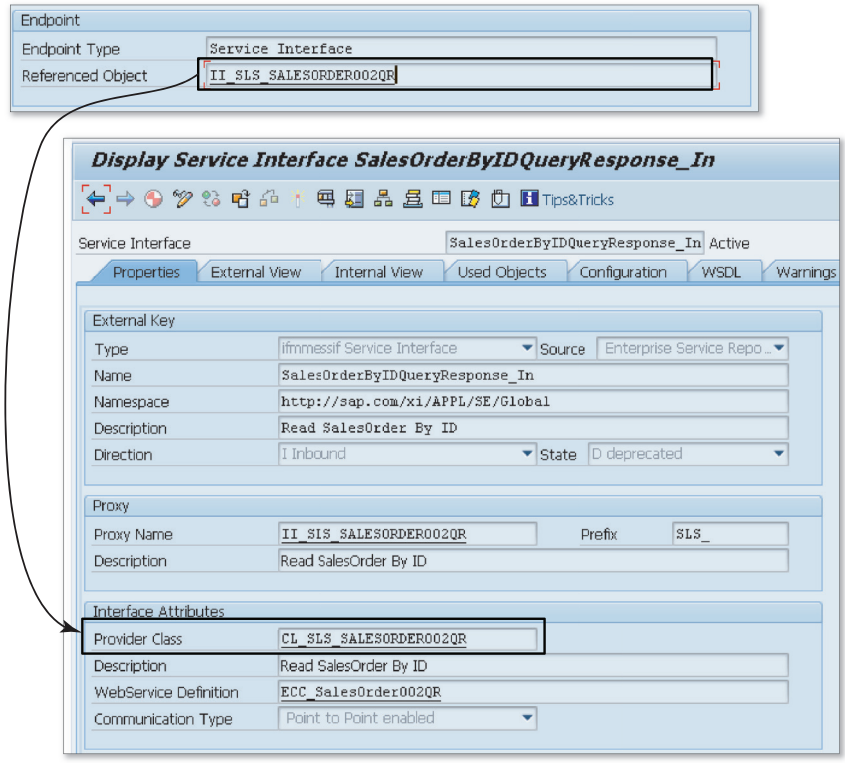


Figure 5.7 Service Interface SalesOrderByIDQueryResponse_In with Provider Class

By double-clicking on the ID of this provider class, we discover method II_SLS_SALESORDERCRTRC~EXECUTE_SYNCHRONOUS, which is executed whenever the service is called synchronously.

If you double-click and browse method EXECUTE_SYNCHRONOUS, you might also discover two BAdIs at the beginning and the end of the code. One BAdI is for changes to the incoming request, and the second one is for changes to the outgoing service response. This is standard procedure for SAP Sales and Distribution (SD) request services and could be something that you are familiar with from IDoc function modules (Figure 5.8). We will look closer at these BAdIs later in this chapter.

Method II_SLS_SALESORDER002QR~EXECUTE_SYNCHRONOUS Active

```
172
173  * Call BAdI SLS_SE_SALESORDER001QR for Outbound processing
174    TRY.
175        CALL BADI L_BADI->OUTBOUND_PROCESSING
176          EXPORTING
177            I_ERROR_FLAG = LV_ERROR_FLAG
178            I_ERROR_TAB  = LT_ERROR
179            I_HEADER     = GRS_HEADCOMV
180            I_PARTY_TAB  = LT_PARTY
181            I_ITEMC_TAB  = LT_ITEMC
182            I_ITEMT_TAB  = LT_ITEMT
183            I_SLINEC_TAB = LT_SLINEC
184          CHANGING
185            C_OUTPUT     = OUTPUT.
186      CATCH CX_PROXY_BADI_PROCESSING .
187    ENDTRY.
188
189
190
191  ENDMETHOD.
```

Figure 5.8 BAdI SLS_SE_SALESORDER001QR for Outbound Service Response

The service definition view is the starting point for Christine and Sean's latest development.

Further Information on Enterprise Services for SAP

The information we've provided on SOA and enterprise services can only highlight a few main points of this vast topic. We recommend reading *Developing Enterprise Services for SAP* by Thomas Pohl and Markus Peter (SAP PRESS 2010).

5.2 Business Scenario

Currently, Byrell's enterprise services for creating sales orders evaluates a request from a client application and raises the order in the backend SAP system. After the sales order has been processed, a service response message is sent back to the calling system using outbound processing. This SOAP response message is unfiltered and contains all parameters that were defined by SAP.

Figure 5.9 displays a simplified overview of some of the sales order enterprise services enabled in Byrell's SAP backend system. Client frontend applications interact with Byrell's system using SOAP message-based enterprise services. Note that all three client frontends have been developed using different development platforms. A first example system has been created using Adobe Flash, while another utilizes

PHP (Hypertext Preprocessor, an HTML-embedded scripting language) to send and receive SOAP messages. A third example client system uses SAP, which is created using Business Server Pages (BSP), for example. It is also important to point out that all three client frontend applications call different enterprise services in Byrell's backend system.

Figure 5.9 Overview of Sales Order Services Provided by Byrell's Backend System

Exposing sales order enterprise services to clients brought about an entirely new way to interact with clients for Byrell because data consumers were suddenly able to develop and call the applications they require by themselves.

In general, the business is more than happy with what the SOA project has achieved so far. However, a change request was raised by the sales team to limit the discount information that is currently given out.

Clients currently receive a response message when using a service to retrieve sales order data by document number (`ECC_SALESORDER010QR` or `SalesOrderERPByIDQuery_sync_V2`).

According to the sales department, this response message contains discount information in the XML response message (see Figure 5.10) that should not be given out to clients when sales order data is retrieved using the `ECC_SalesOrder002QR` enterprise service.

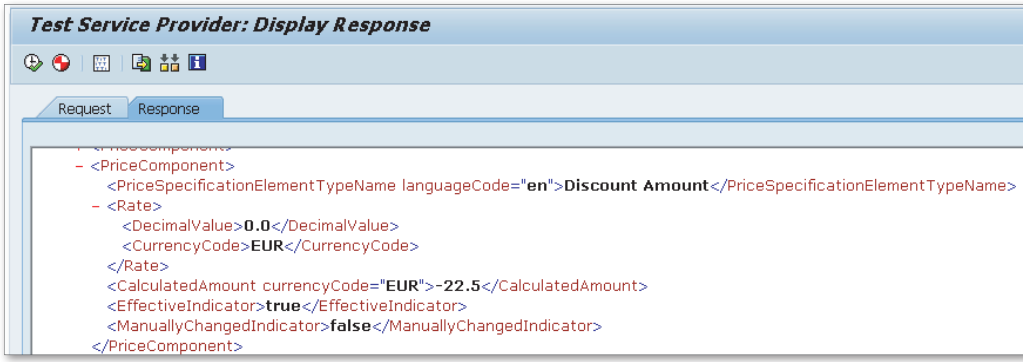


Figure 5.10 Response Message of a Web Service with Discount Data to Be Removed

Sean and Christine's main focus for this exercise will therefore be to ensure that this particular part of discount information is removed from the outbound XML message and not exposed to Byrell's clients.

5.3 Finding the Right Enhancement

Determining the correct enhancement is a little easier this time—we already mentioned in Section 5.1 that SAP enterprise services are delivered with inbound and outbound BAdIs to manipulate request and response messages. Christine thinks that the outbound message of the provider class for service `ECC_SALESORDER010QR` (`SalesOrderERPByIDQuery_sync_V2`) needs to be changed to exclude discount pricing data from the outbound messages. Using an outbound BAdI implementation for this is the ideal solution because it's entirely modification-free.

Sean was involved in the analysis around the SOA project in 2007 and therefore knows the development, but he wants to learn how to find the provider class for enterprise service `ECC_SALESORDER010QR` and create a BAdI implementation for it.

5.4 Implementing the Solution

To implement an outbound BAdI to filter out pricing data, follow the steps in the following sections.

5.4.1 Locate the Web Service using Transaction SE80

Christine shows Sean how to look up the relevant enterprise services for SAP ERP sales. She once again uses the Repository Infosystem (Transaction SE80) for this by choosing Repository Information System • Enterprise Services • Service Definitions.

Christine and Sean then look at available enterprise service definitions by placing the cursor in the Service field, searching for ECC_SALES*, and then clicking Execute (or pressing F8). ECC_SALESORDER010QR is used for the existing web service interface.

5.4.2 Locate a Provider Class for the Web Service

Christine then double-clicks on service definition ECC_SALESORDER010QR, which takes her through to the service definition detail screen (Figure 5.11).

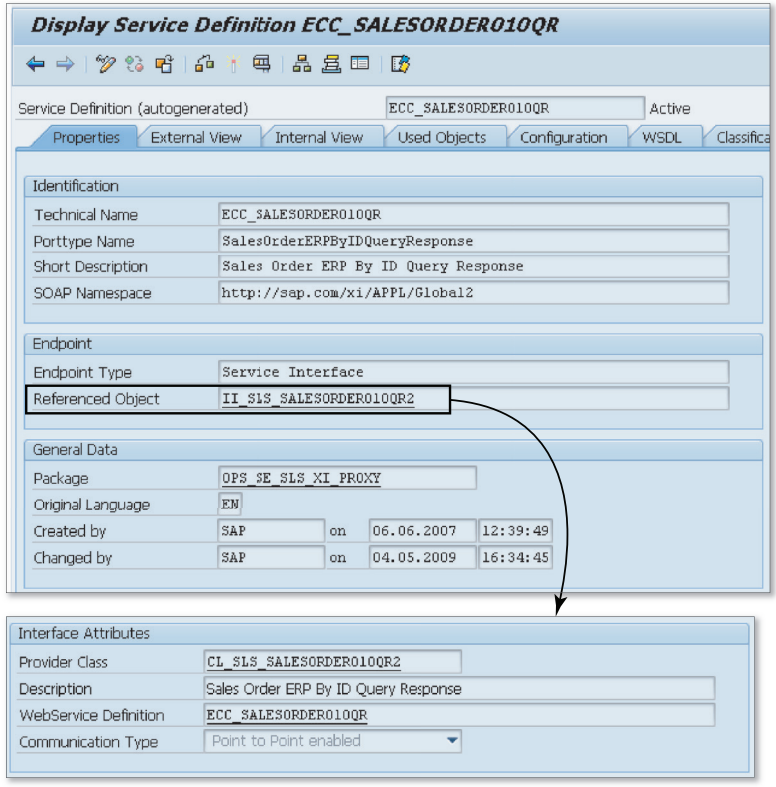


Figure 5.11 Navigating from Service Definition to Service Interface

On the following service interface screen, the provider class (an implementation of the service interface) is displayed in the subscreen entitled Interface Attributes. After double-clicking on provider class CL_SLS_SALESORDER010QR2, Christine and Sean take a closer look at the implementation of method EXECUTE_SYNCHRONOUS, which inherits from service interface II_SLS_SALESORDER010QR2 (see Figure 5.12).

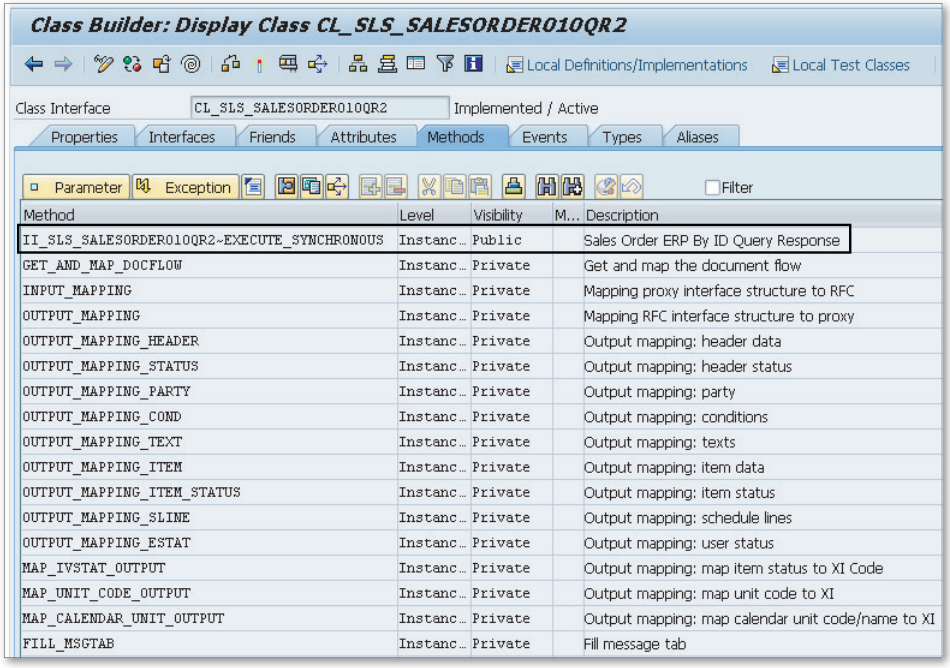


Figure 5.12 Method EXECUTE_SYNCHRONOUS of the Service Provider Class

Christine double-clicks on method EXECUTE_SYNCHRONOUS, which takes her to source code. As expected, the beginning and end of the method reveal inbound and outbound BAdIs. Figure 5.13 shows the code for the outbound BAdI, which Christine now creates a new implementation for.

```
Method    II_SLS_SALESORDER010QR2~EXECUTE_SYNCHRONOUS
584   * Call BAdI or Outbound processing
585     TRY.
586         CALL BADI l_badi->outbound_processing
587           EXPORTING
588             i_head_comv    = ls_head_comv
589             i_head_comr    = ls_head_comr
590             i_item_comv    = lt_item_comv
591             i_item_comr    = lt_item_comr
592             i_hvstat_comv = ls_hvstat_comv
593             i_hvstat_comr = ls_hvstat_comr
594             i_ivstat_comv = lt_ivstat_comv
595             i_ivstat_comr = lt_ivstat_comr
596             i_party_comv   = lt_party_comv
597             i_party_comr   = lt_party_comr
598             i_cond_comv    = lt_cond_comv
599             i_cond_comr    = lt_cond_comr
600             i_text_comv    = lt_text_comv
601             i_estat_comv   = lt_estat_comv
602             i_istat_comv   = lt_istat_comv
603             i_sline_comv   = lt_sline_comv
604             i_sline_comr   = lt_sline_comr
605             i_error_tab    = lt_error
606           CHANGING
607             c_output       = output.
```

Figure 5.13 BAdI SLS_APPL_SE_SALESORDER010QR2 for Outbound Data

5.4.3 Locate and Implement the Outbound BAdI

Christine double-clicks on the BAdI method OUTBOUND_PROCESSING and is taken to the overview of all the enhancement spots. Note that the editor automatically expands the BAdI definition chosen for her. A right-click on the Implementation node allows her to create an implementation.

After clicking Create Implementation, she is prompted with the familiar popup where she specifies the name and description for the enhancement spot implementation she is about to create. She enters "zenh_impl_salesorderqr010" as the name of the implementation and gives it a meaningful description. After confirming the popup screen and specifying a change request, the system takes Christine back to the implementation overview screen. The new implementation she just created has already been selected, and she continues by clicking on the checkmark button (see Figure 5.14).

Christine has now created an implementation of enhancement spot `SLS_SPOT_APPL_SE_SALESORDER`. The next step in the process is to create an implementation for BAdI `SLS_APPL_SE_ SALESORDER010QR2`.

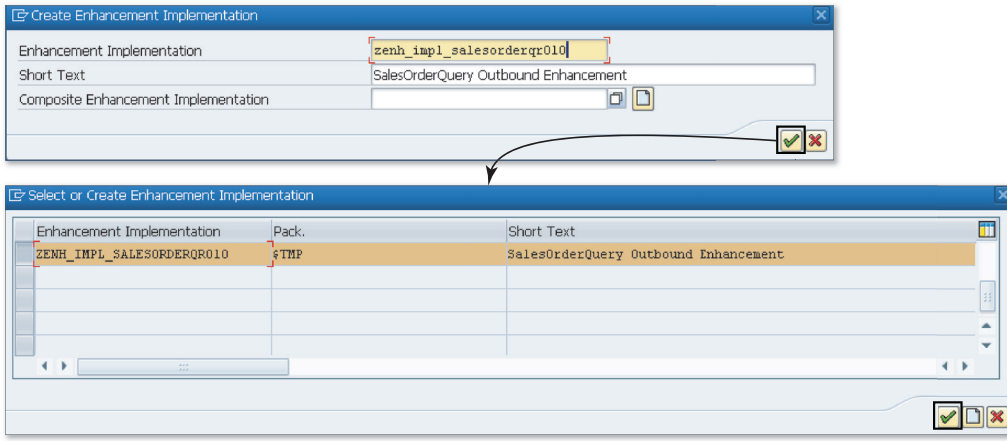


Figure 5.14 Popup to Specify the Implementation Name and Description

In the following popup, she enters "ZENH_IMPL_SALESORDER010QR_OUT" as the BAdI implementation name, gives it a meaningful description, and specifies new custom class ZCL_SALESORDER010QR_OUT. After confirming this screen and specifying a change request, Christine can view her new BAdI implementation (see Figure 5.15).

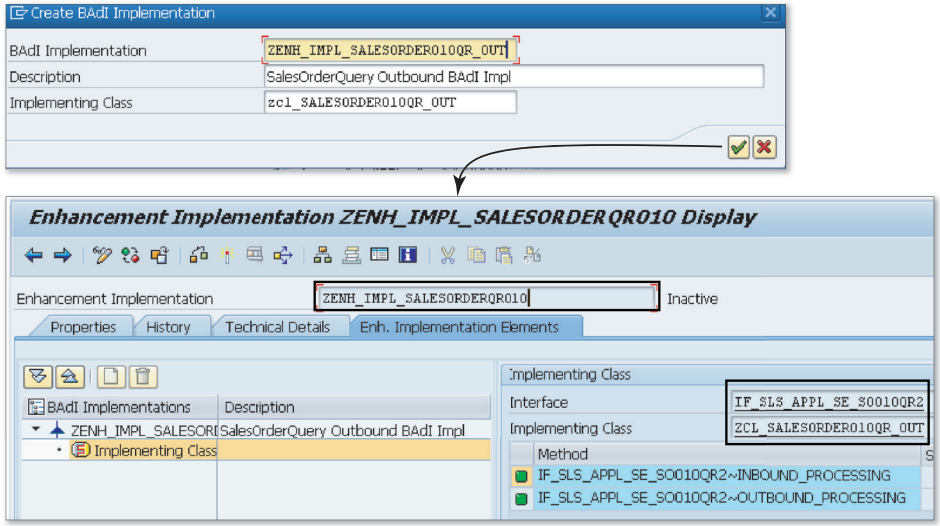


Figure 5.15 New BAdI Implementation Class ZCL_SALESORDER010QR_OUT

Now that Christine and Sean have done the groundwork (i.e., implementing the enhancement and the BAdI), they have to analyze which part of the BAdI signature contains the relevant discount information that needs to be filtered out.

5.4.4 Find the Correct Outbound Parameter in the BAdI Signature

One way to analyze the behavior and parameters of an often-used enhancement is by simply debugging it. In this case, it makes sense because the representation of the outbound parameters in the response of the web service is very different from the parameters of the outbound BAdI. By setting a breakpoint in the BAdI, Christine and Sean can examine the signature fields at runtime and establish which structures need to be excluded or refreshed to prevent exposure. We'll go through these instructions in the following subsections.

Activate the BAdI and Insert a BREAK Statement

Christine double-clicks the OUTBOUND_PROCESSING method of the implementing class ZCL_SALESORDER010QR_OUT and is prompted by the editor to create its implementation.

After clicking Yes, Christine is prompted with a new implementation of OUTBOUND_PROCESSING. She inserts a temporary `break` statement against her username to force the BAdI method into debug mode when it's executed and then activates the method (see Figure 5.16).

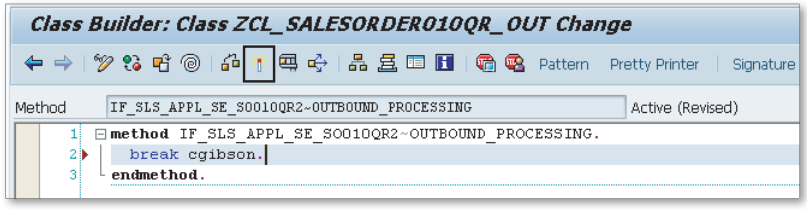


Figure 5.16 Temporary BREAK Statement for Further Analysis of the BAdI Signature

Test and Debug the BAdI Method

Christine now shows Sean how the provider class and its new BAdI outbound method can be tested by using existing sales order data in the system. Together with the breakpoint, this will help them to investigate the signature fields available

and decide which entries need to be removed from the output table so these will not be exposed by the service.

She goes back to enterprise service ECC_SALESORDER010QR, and then clicks on the TEST icon in the toolbar. On the following screen, there are further debugging options (e.g., for method and constructor debugging), but Christine simply confirms the screen with the EXECUTE button.

The editor now displays the request structure in XML format, which is an easy and quick way to test a service. By clicking on the XML EDITOR button, Christine changes into Edit mode and enters a sales order ID "3909" between the <ID> and </ID> tags, which is a sales order number that exists in the system (see Figure 5.17). She then executes the request by clicking the EXECUTE button.

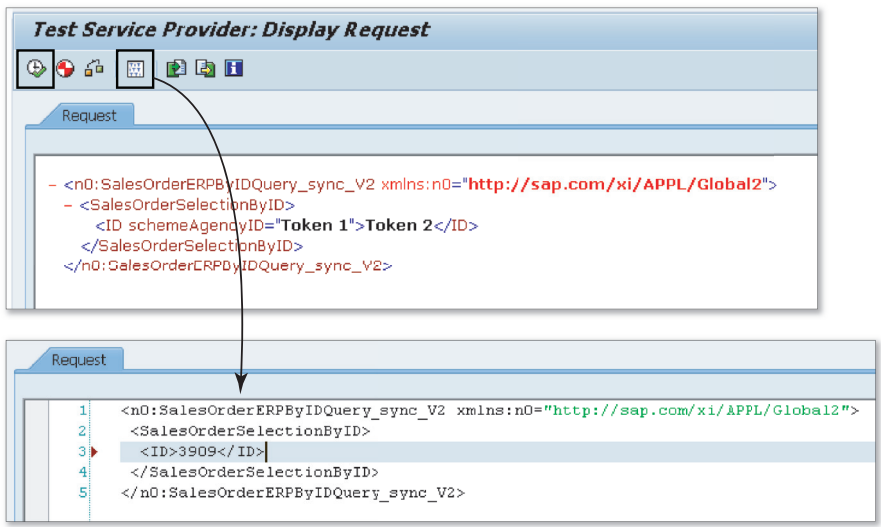


Figure 5.17 Testing the Web Service by Entering the Sales Order Number in XML

After a few seconds, both Christine and Sean are prompted with the debugger screen. Output structure C_OUTPUT contains another internal table called PRICE_COMPONENT, which holds the two lines that need to be filtered out (see Figure 5.18).

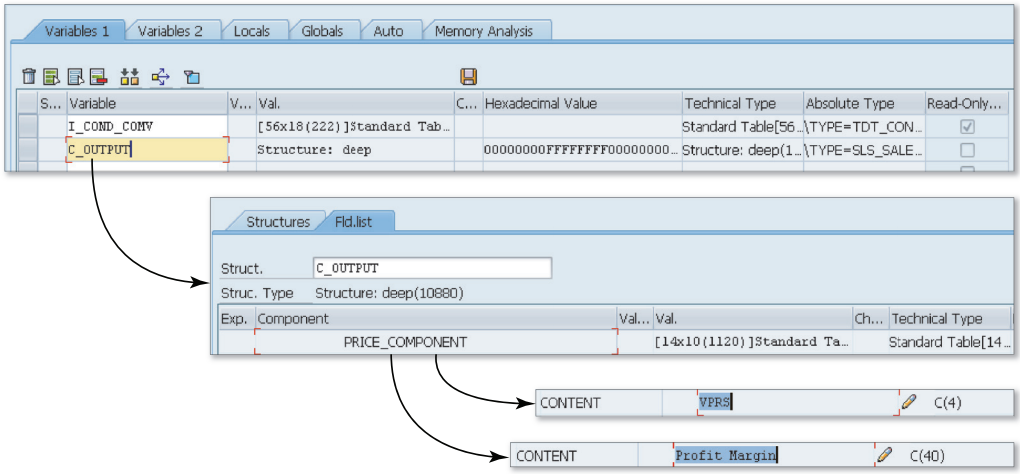


Figure 5.18 Navigation in the ABAP Debugger from the C_OUPUT Signature Element to Values to Be Filtered

Both the internal price (condition VPRS) and "Profit Margin" lines need to be eliminated from the internal table to prevent them from being exposed in the response XML of the web service. The coding to do this will be applied in the next section.

5.4.5 Implement ABAP Coding

The resulting ABAP code that Christine applies is relatively simple. All the coding has to do is run through Table PRICE_COMPONENT in structure `c_output` and delete every line that holds a value of `VPRS` in structure `PRICE_SPECIFICATION_ELEMENT_T1` or the phrase `Profit Margin` in structure `PRICE_SPECIFICATION_ELEMENT_T`.

Tips & Tricks

It's a good practice to create custom tables that hold specific values for filtering to avoid future coding changes. In addition, the Gross Margin field in this example is only filtered if the language is English, which is also better tackled using custom tables. For simplicity, however, no custom tables have been used here; instead, values are coded in the BAdI method.

The code in Listing 5.1 loops through all elements of output structure `price_component` within structure `c_output`. If content fields contain condition `VPRS` or description `Profit Margin`, the entry is deleted from the output table.

```
METHOD if_sls_appl_se_so010qr2~outbound_processing.
  DATA ls_price_component  TYPE sapplco_sales_order_erpby_id29.

  LOOP AT c_output-sales_order_erpby_idresponse-sales_order-price_
component
       INTO ls_price_component.

    IF ls_price_component-price_specification_element_t1-content =
'VPRS' OR
       ls_price_component-price_specification_element_t-content  =
'Profit Margin'.

      DELETE c_output-sales_order_erpby_idresponse-sales_order-price_
component
        USING KEY loop_key.

    ENDIF.

  ENDLOOP.

ENDMETHOD.
```

Listing 5.1 Method OUTBOUND_PROCESSING to Filter Pricing Elements

5.4.6 Testing the Enhanced Service

Christine and Sean run the same test steps as explained in Section "Test and Debug the BAdI Method" earlier in this chapter. As shown, the last entry for the XML response for `PriceComponent` now shows the cash discount instead of `VPRS` and `Profit Margin`, which is the intended result. Again, look at the results in the XML response (see Figure 5.19).

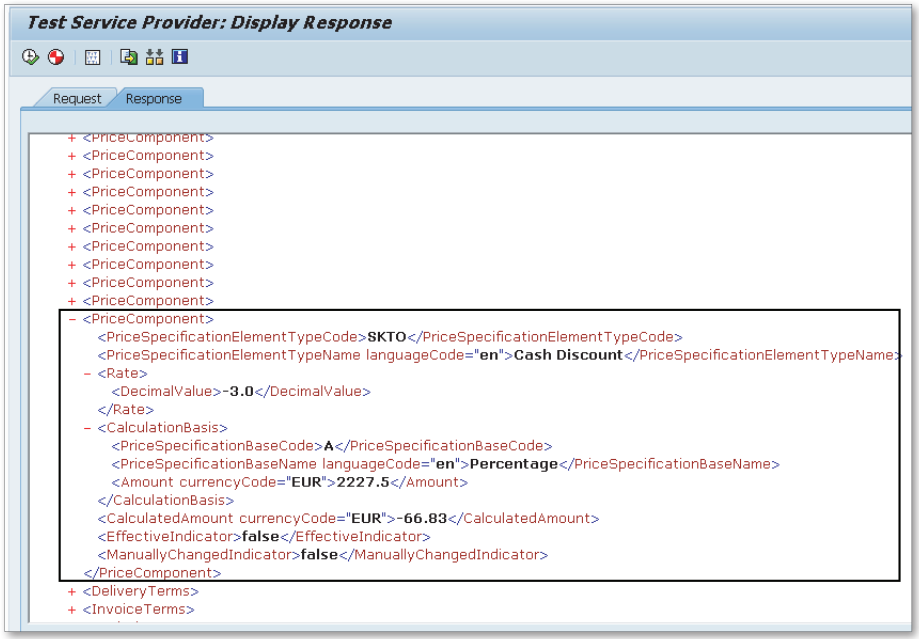


Figure 5.19 Amended XML Response—Now without Crucial Pricing Elements

5.5 Summary

On the whole, this chapter illustrates that the power of BAdIs can easily be applied to inbound and outbound web services. Obviously this is not limited to SD but applies to most enterprise services.

Locating the right BAdI to serve our purpose was a little easier this time, while the focus was on finding the right enterprise service and navigating the service signature and the parameters of the outbound BAdI module.

In the end, the actual ABAP coding was not very complex because the BAdI location and interface was granular enough to enable us to make a minimal change in the right place. An important fact to keep in mind when working on any enhancement is that it's crucial to make a change as minimal as possible in a location (exit, BAdI, enhancement) as defined as possible. Only when you keep this in mind can you ensure that the custom code works as intended without causing side effects.

In the next chapter, Christine and Sean will define custom fields to influence SD pricing.

Standard SD configuration can achieve very lightweight solutions, when paired with a minimal user exit implementation. In this chapter, Sean and Christine will use custom fields within SD pricing, which in turn influences price determination, to help Byrell manage their different clients.

6 Using a Custom Field in SD Pricing

In this chapter, you'll see how a user exit can be implemented to pass the value of a custom field to a communication structure for pricing in order to add a new type of charge to Byrell's delivery process. In practice, this can be very useful when trying to influence pricing using a non-standard decision method, like a bespoke cross-reference table, for example, which determines the calculation.

As most of this is new to Christine, Sean will show her how to use condition tables, access sequences, and pricing conditions to position on the correct condition to allow for the calculation of the new charges.

6.1 Business Scenario

Byrell has been accepting Internet customer orders on behalf of its clients by using a separate online sales order type (web order), which enabled users to identify orders raised from the web. Over time, Byrell has discovered that this process is impractical because it sets online order types apart from numerous standard order types and means all changes applied to standard orders have to be replicated for online order types. Therefore, bringing the order types back together would be more beneficial.

Byrell's systems cover several clients, all with different order types and requirements, which adds further complexity. For example, one client might only require a standard order type, which also applies to online orders. Another client might have a range of order types (domestic, international, parts or service orders) that all should be available as online orders, too.

Byrell also wants to allow a new online exclusive choice for a warehouse pickup/delivery option, which will require different pricing scenarios. For example, if a customer's online order is picked up at the warehouse or delivered as standard, the amount charged to the customer needs to be adjusted (see Table 6.1).

Charge Type	Description	Charge per Order
1	Web order with self-pickup	5.00 EUR
2	Web order with delivery	15.00 EUR

Table 6.1 Different Charges for Web Ordering

Moving forward, these two different web order charges in Table 6.1 won't be created as a separate order type, but instead will be used with the existing sales order types that each of Byrell's clients have already set up. Each of Byrell's customers should be able to individually amend these table entries per order or add more charges if necessary. In addition, charges may also vary for each sales organization, which mean the solution requires added flexibility.

The charge type field itself has already been added in the online store as part of a separate development, so Christine and Sean don't have to deal with this and can focus on the calculation of the charge.

6.2 Finding the Right Enhancement

Christine and Sean discuss this new development request and how to tackle the necessary changes. Christine's suggestion is to define a custom `Z*` table to hold the charges. Before a sales order is saved, a code enhancement would derive the matching web charge (if the order has a charge type set) and add it to the order pricing. This code enhancement would also check whether the condition has already been applied, in which case no action would need to be taken.

However, Sean has a different idea of how to tackle this development. He wants to leverage more of the classical SD configuration and development he is used to, making the overall process less dependent on code enhancement changes and more lightweight. He explains to Christine his plan for how this development could be tackled with only a single line of user enhancement code and suggests the following steps:

1. Add a new ZZWEB_ORDER field to Table VBAK.
2. Extend pricing communication structure KOMK.
3. Create a condition table using Transaction V/03 and including sales organization in the key combination to allow for different charges.
4. Program USEREXIT_PRICING_PREPARE_TKOMK.
5. Maintain the access sequence (Transaction SPRO).
6. Define condition types (Transaction SPRO).
7. Create condition records (Transaction VK11).

Tips & Tricks

In fact, coding USEREXIT_PRICING_PREPARE_TKOMK will only require a single line of code, which assigns the VBAK web order charge field to the TKOMK communication structure. All other steps are performed by standard functionality and configuration of conditions. From a support and upgrade perspective, this is preferable because coding enhancements are likely to require more effort when things go wrong.

Christine realizes that most of these steps—except the programming of the user exit—are new to her because they fall more into the area of SD consultants. Nevertheless, she is keen to find out more and asks Sean to talk her through these steps.

6.3 Implementing the Solution

Because Christine will be using Sean's solution for the enhancement, they first need to add a new custom field to sales order header Table VBAK to hold the charge type value, if applicable. In addition, the same field must be added to the communication structure to make it usable in pricing.

6.3.1 Creating a New Customer Field for Table VBAK

First, Sean accesses Transaction SE11 and creates a data element called ZEL_WEB_ORDER with a description of "web order flag", which will be used in the table append for standard sales document header Table VBAK (see Figure 6.1). He also enters field label "web order flag" in the Short Description field.

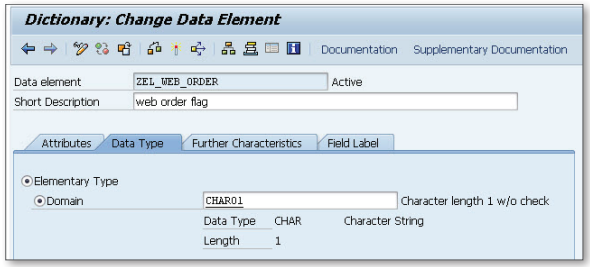


Figure 6.1 Creation of Data Element ZEL_WEB_ORDER

Next, Sean displays Table VBAK in Transaction SE11 in change mode by clicking on the Display/Change button. He then selects the Append Structure button from the menu bar. The next popup window displays all existing append structures for Table VBAK. This list contains all SAP and customer extensions that have been added to the standard table.

To add the web order field to Table VBAK, Sean creates a new append by clicking the Create button. A popup called Create Append Structure for VBAK appears, where he enters "zweb_order" as the name for the new table append in the Append Name field.

Figure 6.2 shows the individual component field ZZWEB_ORDER, which Sean adds to append ZWEB_ORDER. He uses his newly created data element ZEL_WEB_ORDER as the component type for his field component.

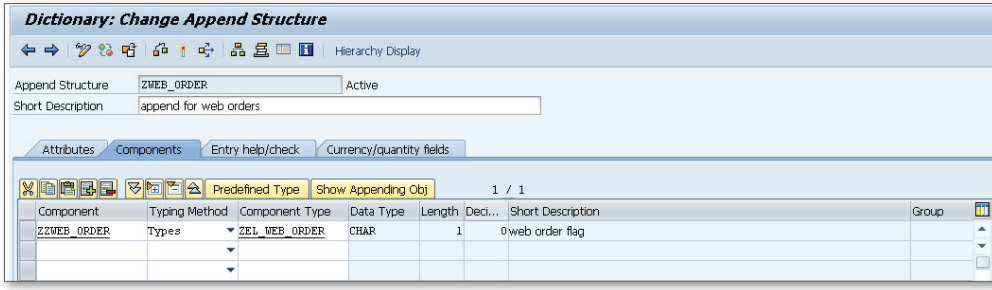


Figure 6.2 Append Structure ZWEB_ORDER

After the field is added to the append, Sean activates the structure by clicking on the Activate button. Table VBAK is shown with the newly added append (see Figure 6.3).

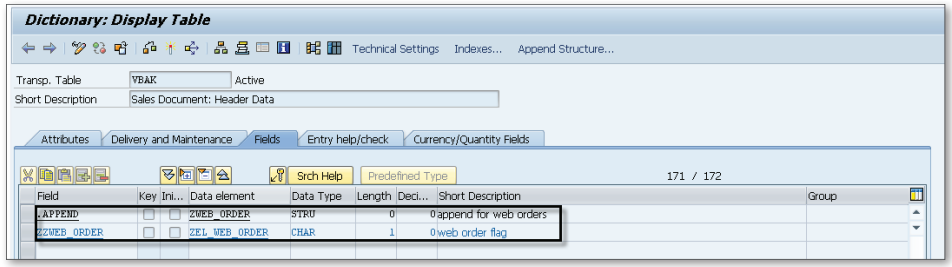


Figure 6.3 Table VBAK with New Append Structure ZWEB_ORDER

6.3.2 Adding a New Field to Pricing Communication Structure KOMK

Sean's next step is to extend the communication structure KOMK, which is used to pass VBAK fields to the pricing routines. Sean's suggested solution has to feed the web charge field, which he has just added to Table VBAK, to communication structure TKOMK. Underlying Table KOMK includes structure KOMKAZ, which is reserved for customer modifications. Adding a field to KOMKAZ requires a modification and therefore SAP registration. Sean uses *www.service.sap.com/sscr* to register the object and obtain the object key to open up structure KOMKAZ for customer modifications.

Tips & Tricks

Access to the preceding link requires specific authorizations. Please speak to your SAP administrator if you don't have access to the online SSCR suite to register the object.

Sean then changes the KOMKAZ structure by adding field ZZWEB_ORDER, referring to the same data element as used in Table VBAK (see Figure 6.4).

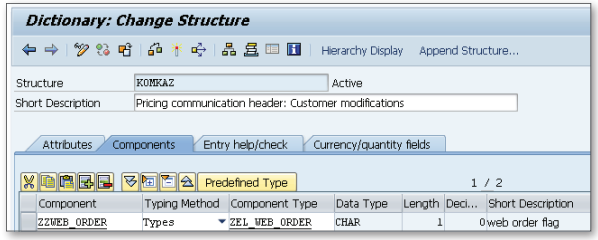


Figure 6.4 Structure KOMKAZ with the ZZWEB_ORDER Field

Finally, he activates KOMK by clicking on the Activate button.

6.3.3 Creating a New Condition Table

Sean then suggests generating a condition table for key combination sales organization/web order charge type, which will later hold the different charges, as requested in the business scenario.

He uses Transaction V/03 and enters a new table number to be created (in this case, he chooses 972) (see Figure 6.5).

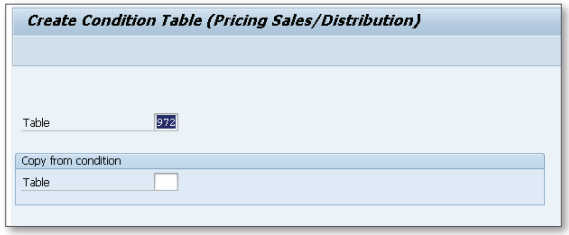


Figure 6.5 Creation of New Condition Table 972

Note

Remember to choose a number from 500 onward. The range 1 through 499 is reserved for SAP standard condition tables.

On the next screen, Sean selects two fields to include in his new table from the field catalogue displayed on the right:

- Sales Organization
- Web Order

He chooses the fields by double-clicking each of them, which automatically populates the left column (Selected fields; see Figure 6.6). Then he generates the condition table by clicking the Generate button.

6.3.4 Maintaining USEREXIT_PRICING_PREPARE_TKOMK

With the previous steps, Sean has added the new field into the standard VBAK table and the pricing communication structure. However, to populate the pricing communication structure from the sales order header field, a one-line code exit is required.

Figure 6.6 Condition Table Fields and the Generate Button

Christine briefly takes over at this point. She navigates to include Program `MV45AFZZ` and locates form `USEREXIT_PRICING_PREPARE_TKOMK`. Christine simply adds an implicit enhancement to the user exit, thus avoiding the creation of a code modification.

She then adds the following one line of code (see Figure 6.7), which moves the content of field `VBAK-ZZWEB_ORDER` to its equivalent in the `TKOM` communication structure. This is required so the web order situation can be evaluated during pricing.

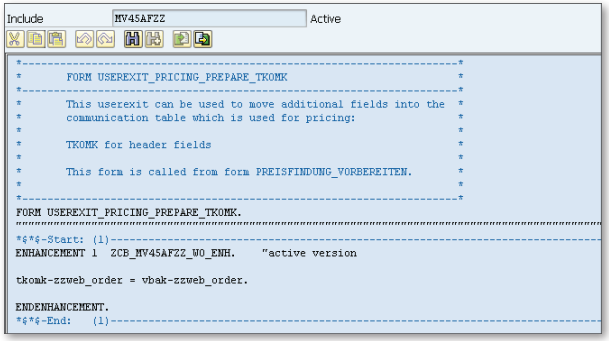


Figure 6.7 USEREXIT_PRICING_PREPARE_TKOMK

6.3.5 Maintaining the Access Sequence

Within SD configuration (Transaction SPRO), Sean then creates a new access sequence using the Sales Organization and web order fields from step 2 (refer to Section 6.3.2). Access sequences work like a filter by which the system determines whether a pricing condition is to be applied to an order, for example. Creating a new sequence for this development ensures that the new charge for web orders is picked up for sales orders that match the filter criteria.

Access sequences are defined in a node by choosing the menu options Sales and Distribution • Basic Functions • Pricing • Pricing Control • Define Access Sequences. In the following popup, Sean double-clicks Maintain Access Sequences and acknowledges that the table is held across SAP clients, so any changes he makes to pricing access sequences are effective across all clients in Byrell's SAP system.

He then clicks on the New Entries button in the menu bar to define a new access sequence. In the table on the right, Sean enters "Z072" in the Access Sequence column. As a description, he enters "Web Orders", confirms the new entry by pressing the Enter key, and then marks the line and double-clicks Accesses in the tree display on the left-hand side (see Figure 6.8).

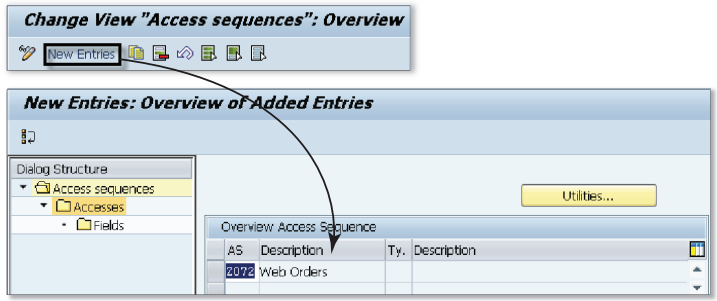


Figure 6.8 Creating an Entry for Access Sequence Z072

A new, empty table opens on the right-hand side, where new accesses can be defined. Sean enters a sequential step number of 10 and defines table 972 from step 3 (refer to Section 6.3.3), which should automatically derive the field entries. After confirming his entry by pressing Enter, he marks the entered line and double-clicks on Fields on the left-hand side to double-check the technical field details of his new access sequence. Then he saves it by clicking the Save button (see Figure 6.9).

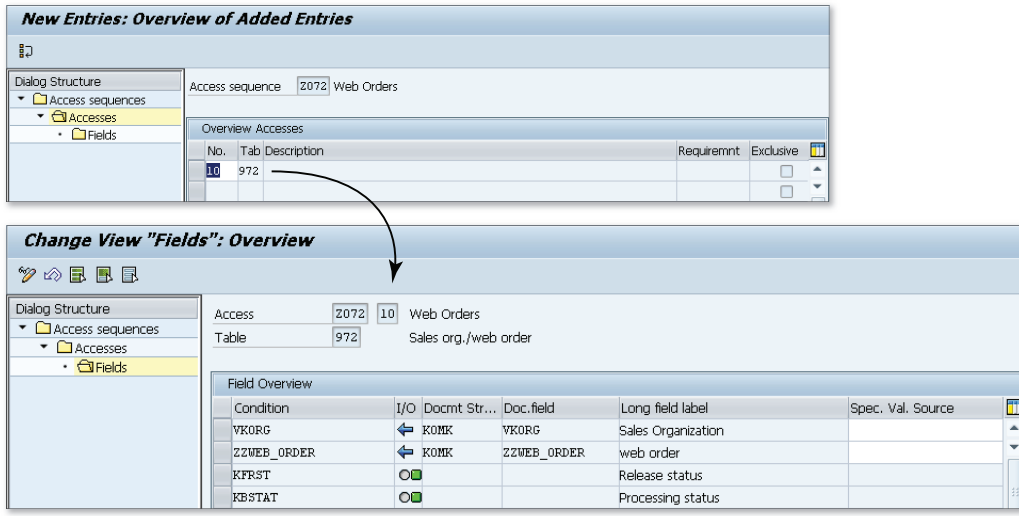


Figure 6.9 Defining Field Details for Access Sequence Z072

6.3.6 Defining a New Condition Type

For pricing to pick up the web order charge condition, a new condition type needs to be created and assigned to the access sequence that was created in step 4 (refer to Section 6.3.4).

Sean navigates back to the configuration nodes (refer to Figure 5.9), and this time he chooses Define Condition Types. He then double-clicks Maintain Condition Types in the Activity popup. He finds and marks condition PR00, and then clicks the Copy button from the menu bar. In the following screen, he specifies the ID ZP70, enters the description "Price" for the newly copied condition type, and assigns access sequence Z072 (created in step 5). Then he saves his work (see Figure 6.10).

6.3.7 Maintaining Condition Rates in Transaction VK11

Finally, Sean maintains the charge types using Transaction VK11. In the selection screen, he enters condition type "ZP70", which he created in the previous step, and then presses Enter. In the next popup, he simply confirms the key combination that is displayed because Sales Organization/web order is the only key combination that was specified in step 3.

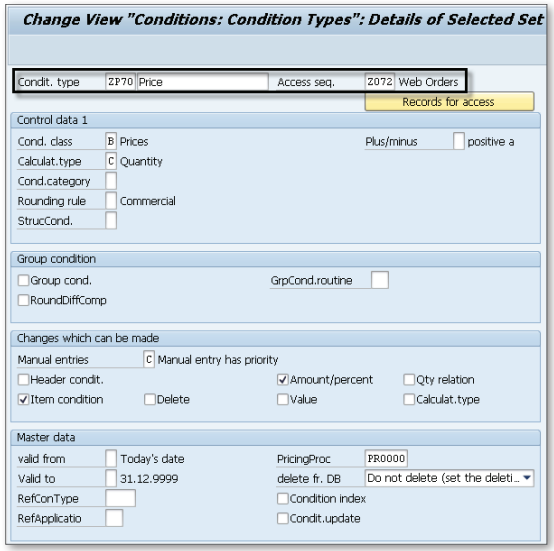


Figure 6.10 Defining Condition Type ZP70 as a Copy of Condition Type PR00

In the following screen, he defines the two charge types as specified earlier in Table 6.1. Note that Sean specified the SALES ORG. as a header detail and the WEB ORDER charge type (1 and 2) in the first column of the table display. When finished, he saves his work (see Figure 6.11).

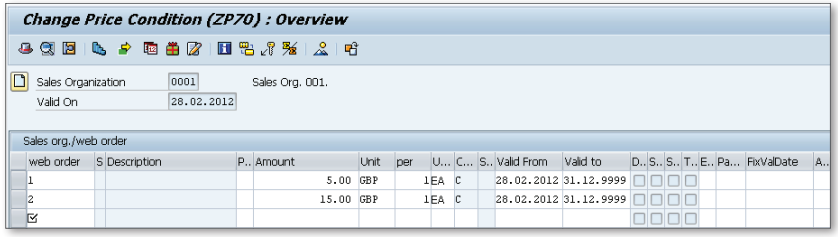


Figure 6.11 Maintaining Condition Rates Using Transaction VK11

6.4 Summary

This chapter showed that you can create powerful enhancements using very little custom coding. Sometimes it is possible to come up with smart solutions by leveraging standard SD configuration and simply adding a single line of code in a user exit.

From a business perspective, Byrell was looking for a more streamlined organization of its order types, because the separate web order type made maintenance of the SD sales order configuration more cumbersome than needed. This also made things too complex when applying changes to Byrell's different clients.

Sean and Christine delivered a solution that uses the web order charge type across all order types and clients, with the following results:

- Made the necessity of a web order type redundant
- Streamlined sales order types across Byrell's clients
- Delivered a new solution for delivery and pickup charges

These three aspects provide Byrell with a simpler solution, more flexibility, and additional charge functionality for its clients.

In the next chapter we'll evaluate a delivery document and introduce a check that sets a block at the document header level.

PART II
Enhancements in Delivery Processing

Occasionally, missing or poorly maintained master data can lead to problems. Standard SAP data validation might not provide sufficient depth to prevent all potential issues further downstream in the SD process. Additional validations, provided by using enhancements, can help in these circumstances.

7 Setting a Delivery Block on the Header Level

Missing or incorrect customer email address data can cause problems when the actual address is used to inform the customer about the status of a delivery, for example. During delivery processing, it might already be too late to check the existence of an email address in the master record, as the notification email is about to be dispatched. In such an instance, it is therefore necessary to check the email address field during delivery creation and warn the user in case it is not maintained. Unfortunately, SD standard functionality in SAP doesn't provide validations such as this, and only an enhancement can provide this type of validation.

In this chapter, we'll show you how an enhancement helps to prevent a customer service issue that's linked to missing master data. You'll learn how to set a header delivery block during creation or change of the document when certain conditions are met. In addition, you will also see how to implement a classic BAdI that has been migrated to an enhancement spot.

7.1 Business Scenario

Byrell has always prided itself on its responsiveness and proactive approach to customer service. Therefore, right from the initial implementation, the SAP ERP system leveraged standard routines during the delivery process to send out customer emails with delivery notifications attached. Byrell received a lot of praise from customers for this process, and it proved to be a simple, yet effective development.

However, some email dispatches fail regularly due to missing email address information on partner master records. Logs for email dispatches are checked daily, but often this is not enough to prevent confusion on the customer's behalf when a delivery notification arrives late. In addition, delivery process times have recently been improved, which has exacerbated the problem of missing partner email addresses data.

Both shipping and customer service departments have decided that they need to put better procedures in place to maintain master email information. In addition to this, both departments have also asked Sean to come up with a suggestion to resolve this problem in terms of delivery processing.

Sean's idea is to block a delivery from further processing unless an email address has been found, which can only be done using custom program logic. Technically, this requires setting a delivery block on the delivery document header if no email address information is held in the master data. This will still allow the user to save the document and rectify the problem afterward.

7.2 Finding the Right Enhancement

Sean discusses this development request with Christine and suggests the following steps:

1. At the end of delivery creation (Transaction VL01N), custom logic in an enhancement has to be triggered. (Sean and Christine will have to investigate where the best place for this logic is.)
2. The coding in the enhancement checks for a populated email address entry for the current customer.
3. If no email address is found, the code sets a delivery block in the delivery document header, which will flag the document to shipping department staff and prevent it from processing.
4. The coding issues a warning message if no email address is maintained and no delivery block exists already.

Based on his expertise and past experiences, Sean is happier to use the classical user exit `USEREXIT_SAVE_DOCUMENT_PREPARE` within include program `MV50AFZ1`.

Christine, however, who is more familiar with leveraging the Enhancement Framework, naturally prefers implementing BAdI LE_SHP_DELIVERY_PROC.

After discussing both approaches, they decide to use a BAdI implementation, since it is better to leverage newer enhancement methodologies. More importantly, Byrell's shipping department has alluded to plans to have different processing scenarios for this in the future. For example, some clients want a warning message displayed, while others prefer an error message. This could mean that multiple enhancement implementations and switches might be required in the future, which works in favor of a BAdI solution because it's easier and neater to keep the custom coding in separate places and implementations.

On the whole, Christine and Sean agree that it's hard to foresee what the business wants to do in this particular area in the near future. By going for a BAdI-based solution, they keep more options open to achieve an elegant and contemporary enhancement solution.

Which Is Best to Use: BAdI or Exit?

For further information on the pros and cons of BAdIs versus customer or user exits, refer to Chapter 1, where you'll find a comparison table, and Chapter 12 for some more in-depth views on this matter.

7.3 Implementing the Solution

Now that Christine and Sean have decided to use the BAdI route, the first step is to find the BAdI definition for LE_SHP_DELIVERY_PROC and then create a new implementation for it. We'll walk through each step in the following sections.

7.3.1 Locating the BAdI

Christine accesses Transaction SE80 to use the Repository Infosystem to find the BAdI. In the left-hand column, she opens up the Enhancements and then the Business Add-Ins folders. She then double-clicks on the Definitions node, which displays a selection screen on the right-hand side. She performs a search by entering package "LE_SHP_BADI" in the Package field (see Figure 7.1).

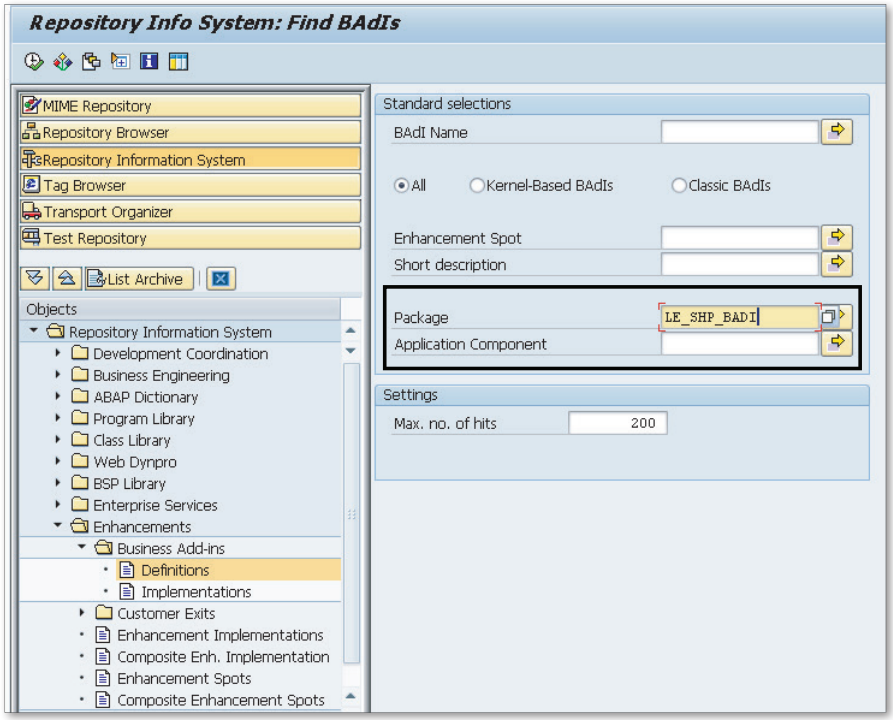


Figure 7.1 Search for a BAdI Definition Using Transaction SE80 Repository Infosystem

To her surprise, the infosystem brings back two results as shown in Figure 7.2.

Name of a BAdI Definition	Enhancement Spot	Description
LE_SHP_DELIVERY_PROC	LE_SHP_DELIVERY_PROC	
LE_SHP_DELIVERY_PROC		Enhancements in Delivery Processing

Figure 7.2 Search Results for a BAdI Definition LE_SHP_DELIVERY_PROC

Christine wonders why this is. She double-clicks on the second hit of her search results (the original, classic BAdI implementation), and she discovers that it has been migrated to an enhancement spot with the same name (the first hit of her search). Figure 7.3 shows where the classic BAdI is flagged as migrated.

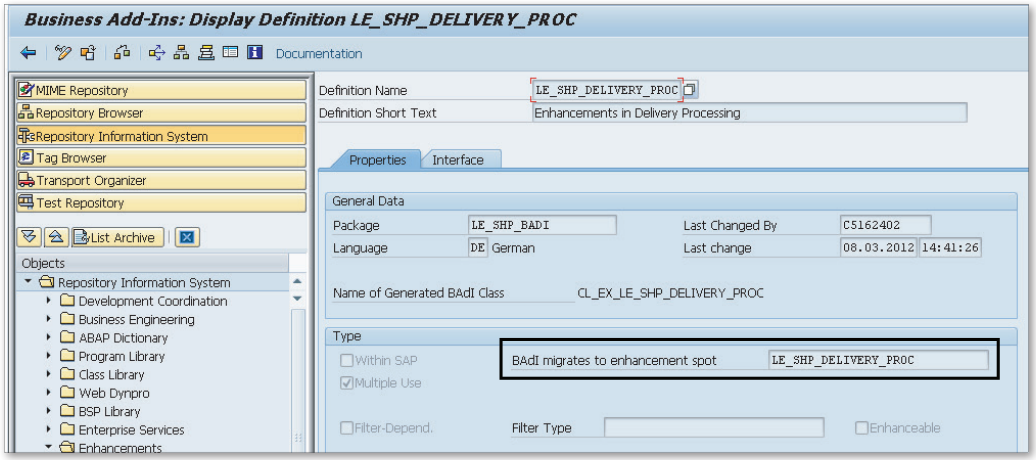


Figure 7.3 Migrated Classic BAdI LE_SHP_DELIVERY_PROC

Looking at enhancement spot `LE_SHP_DELIVERY_PROC`, Christine can see the counterpart of the migration of the original, classic BAdI (see Figure 7.4). She can also see that the enhancement spot is flagged for Multiple Use, which means several implementations of it can coexist. As you might remember from the introduction of this chapter, this feature could be important for future developments that the shipping department might have in mind.

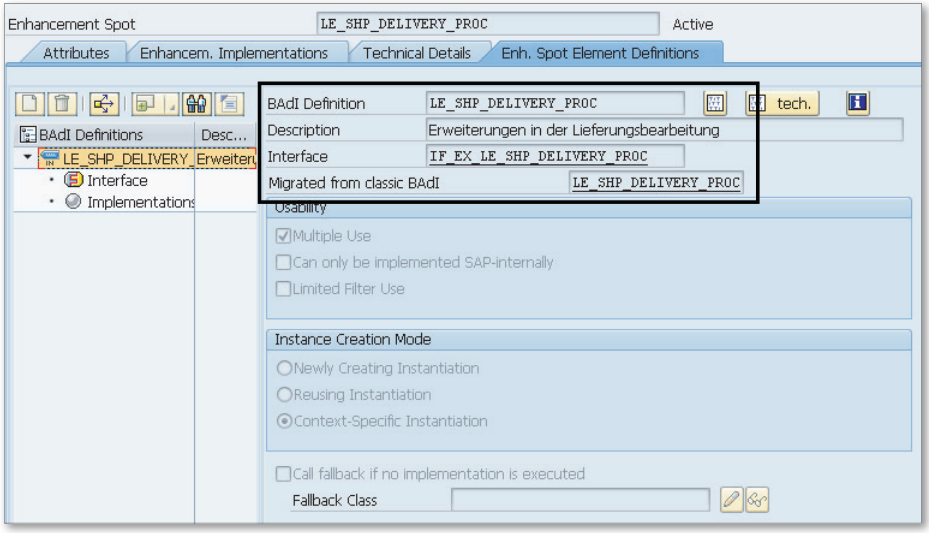


Figure 7.4 Enhancement Spot LE_SHP_DELIVERY_PROC, Containing Migrated Classic BAdI LE_SHP_DELIVERY_PROC

More Information

It's important to understand how classic BAdIs have been migrated over to the latest version of the Enhancement Framework. In the preceding example, you can see that by using an enhancement spot as a "wrapper," an older BAdI has been enabled to use newer features such as filtering. This is important to keep in mind when searching for the right enhancement solution.

Refer to Chapter 1, Section 1.3 for more information on classic BAdIs in the context of the Enhancement and Switch Framework.

By double-clicking on Implementation in the left-hand column, Christine can look through the list of already-existing implementations. She spots a lot of entries for this particular BAdI that have been delivered with the standard SD system (see Figure 7.5).

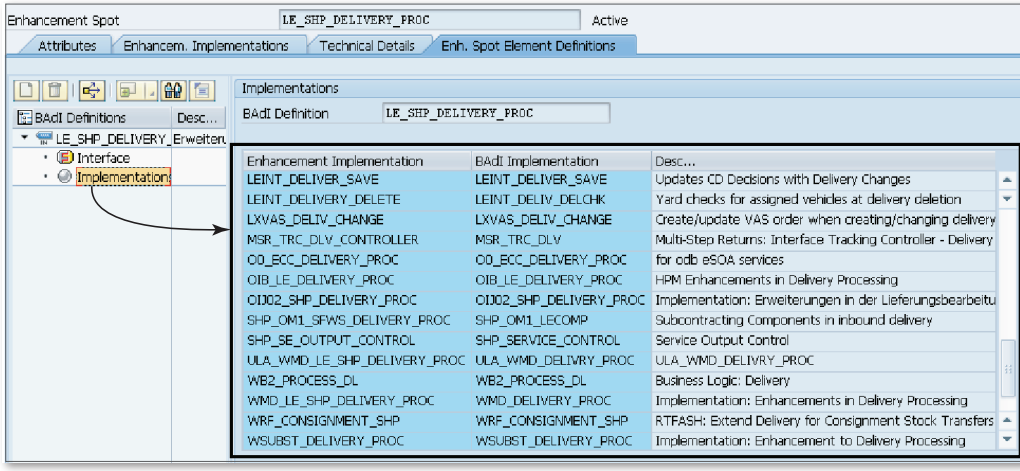


Figure 7.5 Standard BAdI Implementations for LE_SHP_DELIVERY_PROC

7.3.2 Creating a BAdI Implementation

Two steps are necessary for a new BAdI implementation:

1. **Create a new enhancement spot implementation**
   To implement a new BAdI with her own code, Christine first has to create a new enhancement spot implementation, which acts as a wrapper for this and

potentially other BAdI implementations. On the top toolbar, she clicks the Implement Enhancement Spot button.

2. **Create a new BAdI implementation for the new enhancement spot**
   The system then automatically guides Christine to the implementation of the BAdI itself (see Figure 7.6). Finally, the system also suggests creating the implementation from example classes, but because she wants to start fresh, Christine clicks the Create Empty Class button instead.

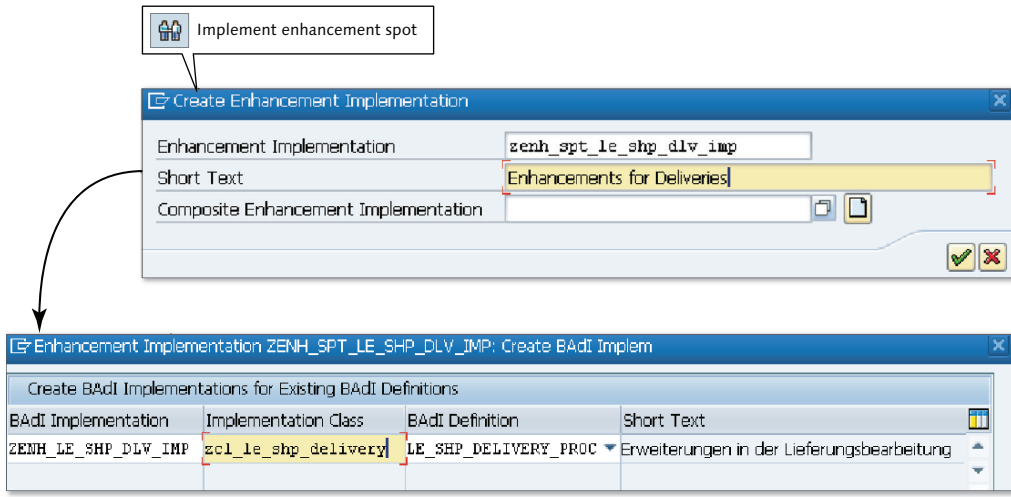


Figure 7.6 Creating Implementations for the Enhancement Spot and BAdI

The system then takes her back to the list display of all BAdI methods. For the task at hand, two methods specifically stand out: `delivery_final_check()` and `save_document_prepare()`.

Both methods are called before a delivery is saved, and using `delivery_final_check()` is perfectly fine because its signature contains all fields required for the email address check. However, Christine wouldn't be able to alter the delivery header block (parameter Table IT_XLIKP) in case no email address is maintained (see Figure 7.7). She therefore turns her attention to method `save_document_prepare()`, which allows changes to almost all delivery data before the document is saved.

Ty.	Parameter	Type spec.	Description
	IT_XLIKP	TYPE SHP_LIKP_T	Current Status of Delivery Headers
	IT_YLIKP	TYPE SHP_YLIKP_T OPTIONAL	Database Status: Delivery Header
	IT_XLIPS	TYPE SHP_LIPS_T OPTIONAL	Current Status of Delivery Items
	IT_YLIPS	TYPE SHP_LIPS_T OPTIONAL	Database Status: Delivery Items
	IT_XVBPA	TYPE SHP_VBPAVB_T OPTIONAL	Current Status of Delivery Partner
	IT_YVBPA	TYPE SHP_VBPAVB_T OPTIONAL	Database Status: Delivery Partner
	IT_XVBFA	TYPE SHP_VL10_VBFA_T OPTIONAL	Current Status of Document Flow
	IT_YVBFA	TYPE SHP_VL10_VBFA_T OPTIONAL	Database Status: Document Flow
	IT_XVBUK	TYPE SHP_VL10_VBUK_T OPTIONAL	Current Status of Header Status
	IT_YVBUK	TYPE SHP_VL10_VBUK_T OPTIONAL	Database Status: Header Status
	IT_XVBUP	TYPE SHP_VL10_VBUP_T OPTIONAL	Current Status of Item Status
	IT_YVBUP	TYPE SHP_VL10_VBUP_T OPTIONAL	Database Status: Item Status
	IT_XVBADR	TYPE SHP_SADRVB_T OPTIONAL	Administration Table for Address Information
	IS_V50AGL	TYPE V50AGL OPTIONAL	Global Delivery Control Flags
	IF_TRTYP	TYPE TRTYP OPTIONAL	Transaction Type
	IF_TCODE	TYPE LE_SHP_TCODE OPTIONAL	Functional Transaction Code in Delivery Processing
	CS_V50AGL_CUST	TYPE V50AGL_CUST OPTIONAL	Control Flags for Delivery Processing (Customer Part)
	CT_FINCHDEL	TYPE FINCHDEL_T	Inspection Results: Final Checks when Saving the Document
	value(FLT_VAL)	TYPE VKORG	Parameter FLT_VAL of method DELIVERY_FINAL_CHECK

Figure 7.7 Signature for Method delivery_final_check()—Crucial Input-Only Field

Christine double-clicks on the method `save_document_prepare()`, and a popup appears asking whether to create the implementation class method (see Figure 7.8). She clicks Yes and is then taken to the editor, where she can add the code for her BAdI (see Figure 7.8).

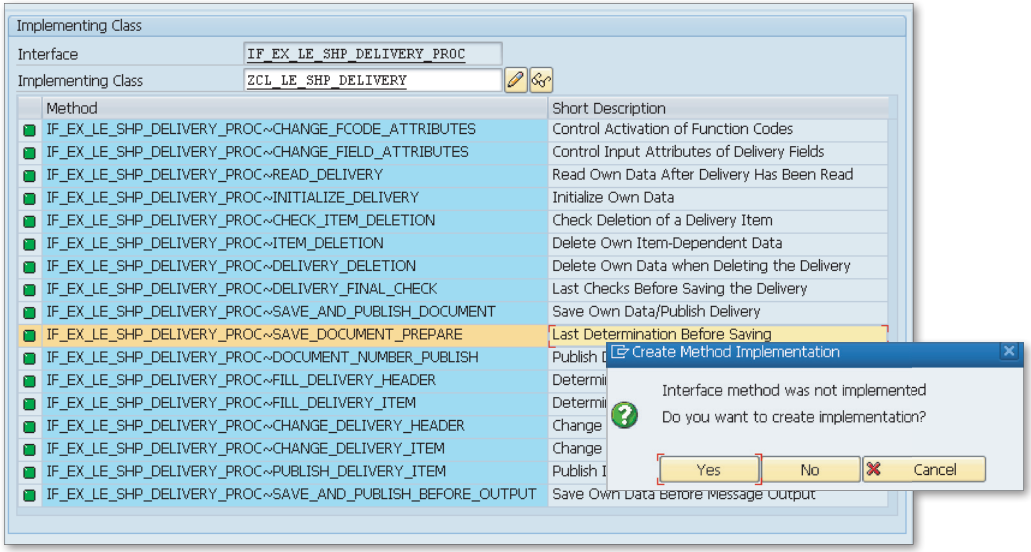


Figure 7.8 Double-clicking on the Method to Implement

7.3.3 Adding Code to the Implementation

Christine then adds the code to her implementation as shown in Listing 7.1.

```
METHOD if_ex_le_shp_delivery_proc~save_document_prepare.
  DATA: ls_xvbadr         TYPE sadrvb,
        ls_xlikp          TYPE likpvb,
        lv_modify_success TYPE abap_bool VALUE abap_false.

* step 1 - check if email address is populated
  LOOP AT ct_xvbadr INTO ls_xvbadr
       WHERE email_addr <> space.
    EXIT.
  ENDLOOP.

* step 2 - set delivery block and throw warning if not populated
  IF ls_xvbadr-email_addr = space.
    LOOP AT ct_xlikp INTO ls_xlikp WHERE lifsk = space.
      ls_xlikp-lifsk = '03'.
      MODIFY ct_xlikp FROM ls_xlikp TRANSPORTING lifsk.
      IF sy-subrc = 0.
        lv_modify_success = abap_true.
      ENDIF.
    ENDLOOP.

    IF lv_modify_success = abap_true.
      MESSAGE w001(zmsg_shp_del).  "Delivery block was set.
    ENDIF.
  ENDIF.
ENDMETHOD.
```

Listing 7.1 Coding for the save_document_prepare() Method

This example loops through the administration version of the address information table and scans the table for any entries without an email address.

If no email entry is found, then structure `ls_vbadr` is populated, and the current version of the delivery header table is looped through for all headers (usually one entry) without a delivery block against them.

The delivery block `LIFSK` is set to "03", and the field is appended to the current table entry. If the `MODIFY` was successful, the program sets flag `lv_modify_success` to true, which will determine later whether the system displays a warning message or not.

7.4 Summary

As on several occasions in this book, Christine and Sean had a choice between two different enhancement techniques (user exit or BAdI) in this chapter. They agreed to use the BAdI method because it kept more options open for potential future developments of the system, which might include more sophisticated BAdI filtering.

Technically, a solution using a user exit could have achieved the same outcome. However, it's important to think about the future when planning enhancements. This thought process should not only be used just in terms of upgrades, but also with regards to plans for improving and enhancing the system.

This chapter also highlighted how classic BAdIs have been migrated by SAP to more modern enhancement spots.

In the next chapter, you'll learn how use SD enhancements to create the foundation of a simple reporting system for key performance indicators (KPIs).

Enhancements can be used for more than just validations and messages; they can actually create small systems to avoid more costly processes.

8 Using a BAdI to Keep Track of Delivery KPIs

You can often use exits, BAdIs, and enhancements to manipulate data or perform data validations. This chapter introduces the idea of using a BAdI to extract delivery performance data into custom database tables. You'll see a scenario that demonstrates another, different use-case from what you saw in Chapter 2—how a BAdI can trigger different processes to help create a small key performance indicator system to store real-time data.

Let's get started by understanding Byrell's business scenario, and then follow along as Christine and Sean create a KPI data logging tool to monitor Byrell's deliveries.

8.1 Business Scenario

One of Christine's regular support tasks is to ensure that delivery KPI reporting runs smoothly. Because only some of Byrell's clients use business intelligence (BI) for transaction reporting, the central IT department came up with a two-pronged approach some time ago: a BI-based reporting solution in SAP and an ABAP-based delivery KPI program. Within this chapter, our focus will be on the ABAP-driven solution. We will discuss Byrell's current KPI solution, understand the current issues that exist within the solution, and see what Byrell would like the revamped solution to achieve.

8.1.1 Current KPI Solution

When the KPI solution was first designed, Byrell had fewer clients than it does now, with a lower number of delivery transactions. As numbers of clients and transactions have increased, the current solution requires too much support effort.

Christine helped to design the first KPI solution, and her weekly task for the past year has been to troubleshoot the overnight batch report job, which frequently fails due to its long run times. This problem has been exacerbated by more clients using the system around the clock, which makes batch scheduling harder because there are fewer time windows in which batch jobs can be run.

Currently, the system consists of a report that is scheduled nightly. This report performs the following tasks:

- Extracts all deliveries per the date/time range in the report variant.
- Retrieves all changes made to these deliveries
- Filters out change records that are relevant to the report.
- Sorts changes in date/time order.
- Displays these in a standard ABAP report, which is attached to the job.

The last point has always been a bone of contention because it involved giving end users access to Transaction SM37 (background job logs and schedules), which poses a risk. In addition, users have recently requested an ABAP List Viewer (ALV) report to allow for easier summarization, filtering, and download functionality because the scheduled job report cannot leverage ALV.

The KPI details extracted are as follows:

- Significant field changes (delivery date, quantity), which might indicate that a delivery has had its delivery priority changed in the planning schedule.
- Delivery creation date to calculate average delivery age as a KPI.
- Filter on specific delivery values (delivery blocks, for example), to flag deliveries with specific statuses and flag them in KPI reporting.

There are no plans to change the content of the KPI data extracted, which is currently based on hardcoded checks in the KPI extract batch program.

8.1.2 To-Be Process for KPI Reporting

While the shipping department asked the team to only look at ways to improve the existing batch solution, Christine has something more radical in mind. Her idea would also increase flexibility in terms of the data extracted and provide Byrell with a more cost-efficient maintenance approach. She therefore begins to draft a

short list of design suggestions for Sean and the shipping department. The following list details her suggestions for the key aspects of a new delivery KPI solution:

- Implement BAdIs in delivery processing to capture KPI events.
- Create a Z table to hold filters for individual fields to report on.
- Create a custom database table to store KPI report data.
- Create an ALV report to query these custom database tables.

Christine suggests using a BAdI to write KPI-relevant change data to a custom table whenever users change a delivery. The main advantage of this solution is that the data is captured at the point of creation, rather than retrospectively. This new solution will be more flexible because the fields that will be reported on will be held in a Z table, which will allow for easy adding or removing of filter fields when instructed so by Byrell's delivery planners. As already mentioned, KPI data itself will be held in custom database tables, which in turn will allow users to access reporting data at any time and in a more convenient way (ALV).

To explain her ideas better, Christine also creates a small solution overview of the to-be delivery KPI process (see Figure 8.1).

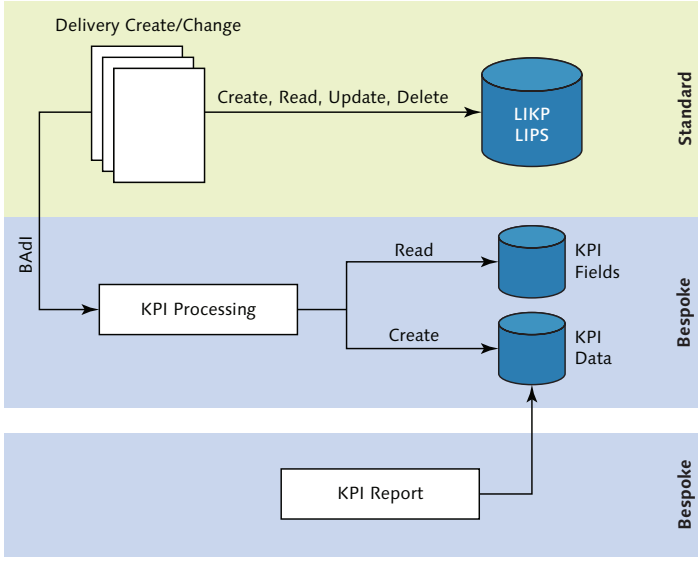


Figure 8.1 Christine's Suggested New Design for Capturing and Reporting KPIs

The STANDARD section of Figure 8.1 describes the delivery process in the standard system. Using standard transactions or BAdI interfaces, delivery data is created, read, updated, or deleted (CRUD) from standard tables such as LIKP (delivery header data) and LIPS (delivery item data). A BAdI in the middle level of the figure is called at the end of some of the delivery CRUD processes. This exit accesses two custom tables for KPI fields and KPI data. A third custom layer at the bottom of the figure contains a customer ALV report that accesses the KPI data table. This report can be run ad hoc and at any time by the users and is not directly dependent on how the data is captured or extracted. However, this chapter doesn't describe this ALV report any further but rather focuses on how the data is derived.

More Information on ALV Reports

For further details on ALV reports or to create your own ALV reports, refer to the following:

- Search for ALV reporting at *http://help.sap.com.*
- Check out the SDN Wiki pages on ALV at *http://wiki.sdn.sap.com/wiki/display/ABAP/ALV.*

Christine's suggestions are approved, and she is briefed by Sean to analyze and specify the solution in more detail.

8.2 Finding the Right Enhancement

Christine and Sean discuss which enhancement method will be the best—there are two options that allow the system to save additional data after creating or changing a delivery: user exit `USEREXIT_SAVE_DOCUMENT` (ABAP `MV50AFZ1`) and method `SAVE_AND_PUBLISH_DOCUMENT` of BAdI definition `LE_SHP_DELIVERY_PROC`.

Finding a User Exit and BAdI

Christine finds these two enhancement options by using the search function in the Repository Infosystem of Transaction SE80 (as already described in Chapter 7). To find user exits for delivery processing, use the search pattern shown in Figure 8.2. The result set of this search contains several ABAP includes, including `MV50AFZ1` for form exit `USEREXIT_SAVE_DOCUMENT`.

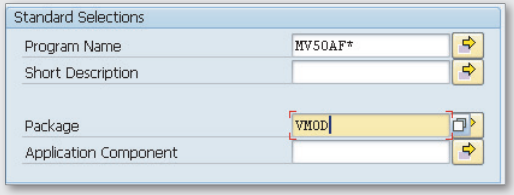


Figure 8.2 Search Pattern in Repository Infosystem to Find User Exits for Deliveries

Detailed steps for finding the BAdI definition `LE_SHP_DELIVERY_PROC` are provided in Chapter 7, where Christine put this BAdI to good use.

After a quick check in the system, Christine has established that `USEREXIT_SAVE_DOCUMENT` in ABAP `MV50AFZ1` is still in an untouched, standard condition.

She reflects once again on the actual purpose of her planned enhancement work. The main reason for the implementation is to evaluate delivery data, check for changes, and write data to a custom table if necessary. In other words, there is no requirement to *change* delivery-related data.

Moreover, in a scenario where a delivery was created, it will be necessary to have the new delivery number because this is crucial for logging KPI data to the custom table. Without the delivery number, the logged information will be of little value for KPI reporting. In addition, the delivery number is very likely to be featured as a key element of Christine's KPI table, which will be discussed in Section 8.3.

As a matter of fact, both `USEREXIT_SAVE_DOCUMENT` and BAdI method `SAVE_AND_PUBLISH_DOCUMENT` fulfill the preceding requirements.

Deciding between the BAdI and User Exit Solutions

On the whole, the differences between `USEREXIT_SAVE_DOCUMENT` and BAdI method `SAVE_AND_PUBLISH_DOCUMENT` are negligible for this specific scenario. However, keep in mind that the advantages of the Enhancement Framework (versions, filtering, no modification) are significant here, and it should always be the enhancement tool of choice if a BAdI implementation is considered.

Christine therefore decides to use the BAdI for this development because she wants to leverage new enhancement technology. Additionally, there's a chance that in the future, the KPI system might require filtering by criteria such as plant or sales organization, which could then be easily enabled using BAdI filters.

8.3 Implementing the Solution

Before Christine and Sean can start development, there is still some design work they both have to do for the KPI fields and data tables. As previously mentioned in Section 8.1, there will be two custom tables: one table to hold KPI field data and the other to store the actual delivery KPI data.

In the next sections, you'll follow along as Christine and Sean plan the layout of the KPI field and data table.

8.3.1 Custom KPI Field Table Layouts

The planned field layout is as follows (see Table 8.1 and Table 8.2). The `KPI_TABLE` and `KPI_FIELD` fields will hold the specific table and field information that's extracted, while the fields `LOG_OLD_VS_NEW`, `LOG_CREATE`, and `LOG_VALUE` are flags that determine criteria by which an entry is extracted into the KPI data table, exactly what the current KPI solution performs, but without the flexibility of the bespoke KPI field tables.

Field	Key	Type	Description
`MANDT`	√	`MANDT`	Client
`KPI_TABLE`	√	`ZKPI_EL_TABLE`	KPI field table
`KPI_FIELD`	√	`FIELDNAME`	KPI field
`LOG_OLD_VS_NEW`		`CHAR(1)`	Log entry created if current and `db` value are different
`LOG_CREATE`		`CHAR(1)`	Log entry when entry is created
`LOG_VALUE`		`CHAR(1)`	Log entry created when a specific value is reached

Table 8.1 Table Layout for KPI Field Table

The KPI data table shown in Table 8.2 stores the delivery document and item number. To differentiate each line, Sean suggests simply using a field holding an incremental number, which will be increased for every record.

Field	Key	Type	Description
MANDT	√	MANDT	Client
VBELN	√	VBELN_VL	Delivery number
POSNR	√	POSNR_VL	Delivery item number
GUID	√	OS_GUID	16-byte GUID number
KPI_DATE		DATE	KPI date
KPI_TIME		TIME	KPI time
KPI_TABLE		ZKPI_EL_TABLE	KPI field table
KPI_FIELD		FIELDNAME	KPI name
VALUE_OLD		CHAR(255)	Old field value
VALUE_NEW		CHAR(255)	New field value

Table 8.2 Table Layout for KPI Data Table

Christine has a better idea though. She convinces Sean that it would be easier to create a Globally Unique Identifier (GUID) field, which will be generated by the system. This GUID serves as a safeguard in case multiple entries are extracted at the same point in time and could cause a duplicate database key. Moreover, the table will hold the respective table and field name as well as old and new field values. Christine explains that she aims to use method `GET_NEW_GUID` of standard class `CL_RECA_GUID` to generate new, unique GUIDs.

Note on Class CL_RECA_GUID

Class `CL_RECA_GUID` is part of the SAP NetWeaver software component EA-APPL. If you intend to use this class within your system, ensure that this component is installed first. Alternatively, you can use function module `GUID_CREATE` to achieve the same goal.

8.3.2 Define Custom Table ZKPI_FIELDS

In a second step, Christine now uses Transaction SE11 to create the custom table for the KPI field and data. On the first screen, she enters "ZKPI_FIELDS" in the Database Table field and clicks the Create button.

Christine then goes to the Delivery and Maintenance tab and defines the new table as shown in Figure 8.3.

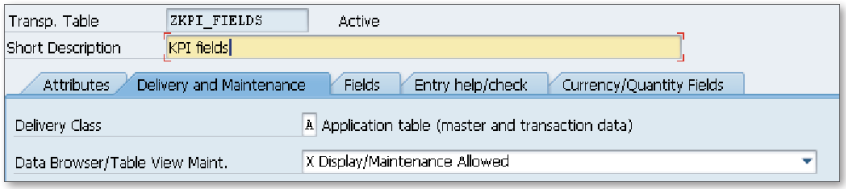


Figure 8.3 Display and Maintenance Tab for Table ZKPI_FIELDS

She then goes to the Fields tab and enters the six fields as defined earlier in Table 8.1 (see Figure 8.4).

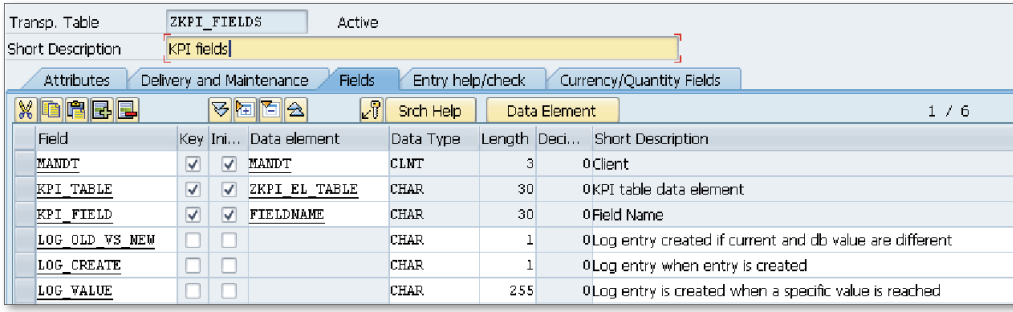


Figure 8.4 Field Tab Entries for Table ZKPI_FIELDS

Note

Together with a custom domain element, the data element ZKPI_EL_TABLE in Figure 8.4 ensures that users only have a limited choice of tables when choosing entries. This is achieved by defining a value range to data domain `ZKPI_DOM_TABNAME`, which is used for data element `ZKPI_EL_TABNAME`. You can view domain ZKPI_DOM_TABNAME by double-clicking on the domain entry in the Data Type tab of data element ZKPI_EL_TABNAME (see Figure 8.5).

Remember that this field holds the table name of the values that Christine wants to extract for the KPI data table. In other words, it should only contain a subset of the entries available in the signature of BAdI method `SAVE_AND_PUBLISH _DOCUMENT`.

See Figure 8.6 for an overview of how data element ZKPI_EL_TABLE is set up.

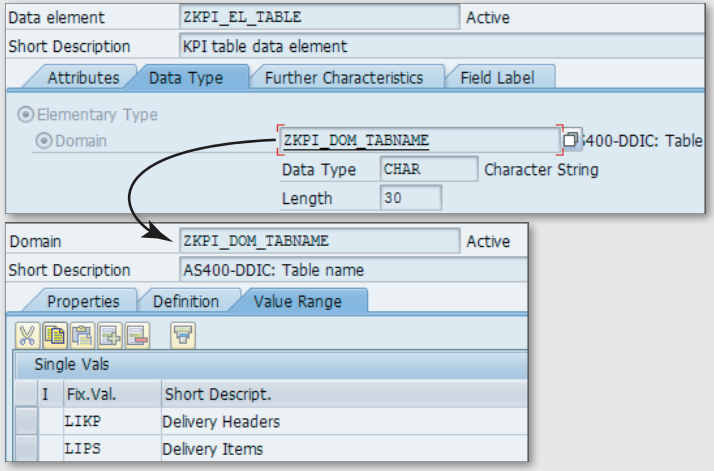


Figure 8.5 Data Element ZKPI_EL_TABLE and the Value Range Tab of Domain ZKPI_DOM_TABLE

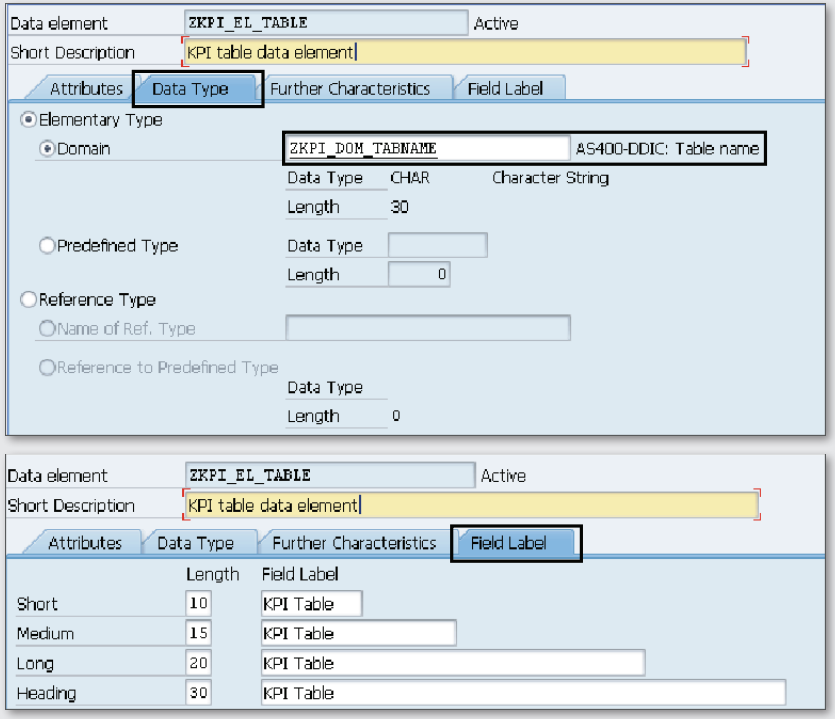


Figure 8.6 Overview of Data Element ZKPI_EL_TABLE

After Christine has declared all fields for Table ZKPI_FIELDS, she assigns an enhancement category by selecting Extras • Enhancement Category… from the top menu bar and then choosing the Can be enhanced (character-type or numeric) radio button on the popup screen.

Christine then activates the new ABAP Data Dictionary object by clicking the Activate ( ) button.

8.3.3 Generating a Table Maintenance for ZKPI_FIELDS

Before moving on to create the other custom table, ZKPI_DATA, Christine generates a table maintenance dialog program for users to be able to maintain `ZKPI_FIELDS` table entries in the future. She accesses the table maintenance generator by choosing Utilities • Table Maintenance Generator from the top menu bar. Alternatively, she can also use Transaction SE54 to call up the generator.

On the next screen, she assigns an authorization group ("&NC&" for no authorization assignment in this case because none was specified) and a new function group (ZKPI_FIELDS) for the maintenance view to be generated into. Because the maintenance program doesn't need any selection tables to navigate from, Christine selects one Step and gives it an overview screen of "100". She then clicks the Create button on the top left (see Figure 8.7).

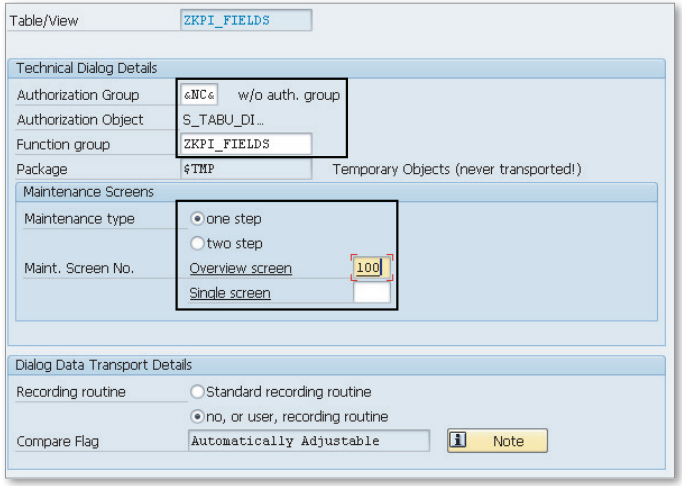


Figure 8.7 Generating a Table Maintenance Dialog

After the generation has finished, Christine can make changes to Table ZKPI_FIELDS from Transaction SM30 (Table Maintenance).

8.3.4 Define Custom Table ZKPI_DATA

Christine then creates her second custom table from Transaction SE11 (Data Dictionary), which will hold the actual KPI data. On the first screen, she enters "ZKPI_DATA" in the Database Table field and clicks the Create button.

Similar to the creation of Table ZKPI_FIELDS, Christine defines the table in the Delivery and Maintenance tab as shown in Figure 8.8.

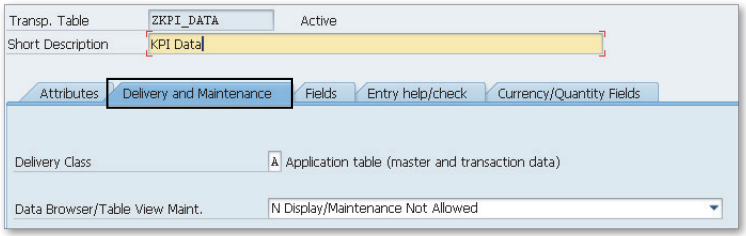


Figure 8.8 Display and Maintenance Tab for Table ZKPI_DATA

This time, she defines her table as Maintenance Not Allowed because she doesn't want users to be able to alter any KPI values. This obviously means that she also doesn't need to generate table maintenance for Table ZKPI_DATA.

She then goes to the Fields tab and enters the eight fields (see Figure 8.9) as defined earlier in Table 8.2.

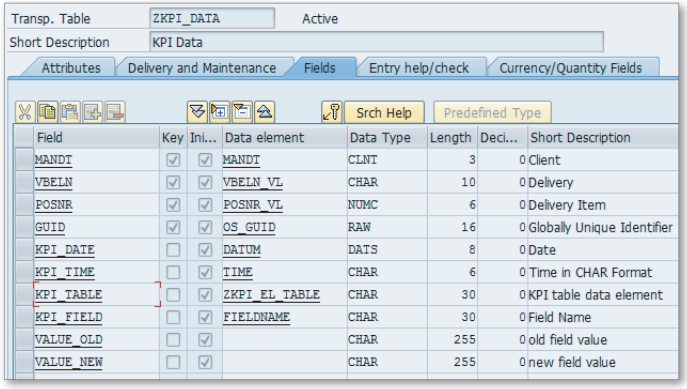


Figure 8.9 Field Tab Entries for Table ZKPI_DATA

Christine finishes off her Data Dictionary work by assigning an enhancement category (Extras • Enhancement Category...) and activating the table with the Activate ( ) button.

8.3.5 Implement ABAP Coding in Method SAVE_AND_PUBLISH_DOCUMENT

Christine already implemented BAdI definition LE_SHP_DELIVERY_PROC in Chapter 7, so she can move on to add her KPI coding to BAdI method SAVE_AND_PUBLISH_DOCUMENT.

Note: Reusing BAdI Implementations

Reusing an existing implementation of a BAdI can potentially cause other methods within the same implementation to fail if a change has been made to any other method, which causes a short dump. This can happen after a transport is applied that lacks some prerequisites (e.g., Data Dictionary objects) because of a wrong transport sequence. In such a case, the failing BAdI implementation can cause *all* other methods to fail until the problem is resolved.

While in practice it's perfectly fine to reuse an existing implementation, it might make sense to create a new implementation for different projects or work tasks. This decision can only be made on a case-by-case basis.

For the purpose of this exercise, it's fine to reuse the current implementation. However, please check implementations for productive use concerning whether they leverage other data dictionary objects—for example, across the BAdI methods from the same implementation—in which case, it's better practice to create a new implementation.

Christine navigates to the implementing class ZCL_LE_SHP_DLV (created in Chapter 7) and double-clicks on method SAVE_AND_PUBLISH_DOCUMENT (see Figure 8.10).

She then adds the ABAP code shown in Listing 8.1.

```
METHOD if_ex_le_shp_delivery_proc~save_and_publish_document.
  DATA: ls_kpi_fields        TYPE zkpi_fields,
        lt_kpi_fields        TYPE STANDARD TABLE OF zkpi_fields,
        ls_kpi_data          TYPE zkpi_data,
        lt_kpi_data          TYPE STANDARD TABLE OF zkpi_data,
        ls_xlikp             TYPE likpvb,
        ls_ylikp             TYPE likpvb,
        ls_xlips             TYPE lipsvb,
        ls_ylips             TYPE lipsvb,
```

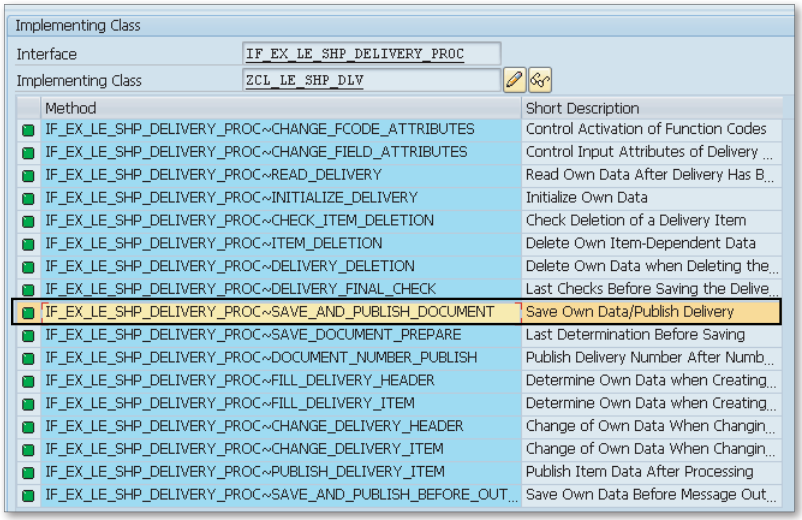


Figure 8.10 Navigating to BAdI Method SAVE_AND_PUBLISH_DOCUMENT

```
        lv_extract          TYPE abap_bool VALUE abap_false,
        lv_xstruc(50),       " name of structure for current data
        lv_xtable(50),       " name of table for current data
        lv_ystruc(50),       " name of structure for db data
        lv_ytable(50),       " name of table for db data
        lv_guid_16          TYPE guid_16.

  FIELD-SYMBOLS: <fs_oval_s> TYPE any,              " FS structure
                 <fs_oval_t> TYPE STANDARD TABLE,   " FS table
                 <fs_oval_f> TYPE any,              " FS field
                 <fs_nval_s> TYPE any,              " FS structure
                 <fs_nval_t> TYPE STANDARD TABLE,   " FS table
                 <fs_nval_f> TYPE any.              " FS field

*   Step 1 - derive KPI fields
    SELECT *
      FROM zkpi_fields
      INTO TABLE lt_kpi_fields.

*   Step 2 - establish change scenarios
    LOOP AT lt_kpi_fields INTO ls_kpi_fields.
      CONCATENATE 'ls_x'
```

```
                ls_kpi_fields-kpi_table
          INTO lv_xstruc.

    CONCATENATE 'ls_y'
                ls_kpi_fields-kpi_table
          INTO lv_ystruc.

    CONCATENATE 'it_x'
                ls_kpi_fields-kpi_table
          INTO lv_xtable.

    CONCATENATE 'it_y'
                ls_kpi_fields-kpi_table
          INTO lv_ytable.

    ASSIGN: (lv_xtable) TO <fs_nval_t>,
            (lv_xstruc) TO <fs_nval_s>,
            (lv_ytable) TO <fs_oval_t>,
            (lv_ystruc) TO <fs_oval_s>.

    LOOP AT <fs_nval_t> INTO <fs_nval_s>.
      IF ls_kpi_fields-kpi_table = 'LIKP'.     " delv headers
        READ TABLE it_ylikp INTO ls_ylikp
             WITH KEY vbeln = ls_xlikp-vbeln.
        IF sy-subrc <> 0.
*         is "log KPI on create" flag set & in Create mode?
          IF ls_kpi_fields-log_create = abap_true AND
                        sy-tcode+2(2) = '01'.
            lv_extract = abap_true.
          ENDIF.
        ENDIF.
      ELSEIF ls_kpi_fields-kpi_table = 'LIPS'. " delv items
        READ TABLE it_ylips INTO ls_ylips
             WITH KEY vbeln = ls_xlips-vbeln
                      posnr = ls_xlips-posnr.
        IF sy-subrc <> 0.
*         is "log KPI on create" flag set & in Create mode?
          IF ls_kpi_fields-log_create = abap_true AND
                        sy-tcode+2(2) = '01'.
```

```
            lv_extract = abap_true.
          ENDIF.
        ENDIF.
      ENDIF.

      ASSIGN: COMPONENT ls_kpi_fields-kpi_field OF STRUCTURE <fs_
oval_s> TO <fs_oval_f>,
              COMPONENT ls_kpi_fields-kpi_field OF STRUCTURE <fs_
nval_s> TO <fs_nval_f>.
*   "log KPI if database/current " flag set & values different?
      IF ls_kpi_fields-log_old_vs_new = abap_true AND
         <fs_oval_f> <> <fs_nval_f>               AND
         lines( <fs_oval_t> ) > 0.
        lv_extract = abap_true.

*     check "value reached" flag is set & value reached
      ELSEIF ls_kpi_fields-log_value <> space AND
             <fs_nval_f> = ls_kpi_fields-log_value.
        lv_extract = abap_true.
      ENDIF.

*     Step 3 - populate ls_kpi_data structure
      IF lv_extract = abap_true. " something to extract?
        IF ls_kpi_fields-kpi_table = 'LIKP'.
          ls_kpi_data-vbeln = ls_xlikp-vbeln.
          ls_kpi_data-posnr = 0.
        ELSEIF ls_kpi_fields-kpi_table =  'LIPS'.
          ls_kpi_data-vbeln = ls_xlips-vbeln.
          ls_kpi_data-posnr = ls_xlips-posnr.
        ENDIF.

*       get new GUID from CL_RECA_GUID class
        cl_reca_guid=>guid_create( IMPORTING ed_guid_16 = lv_guid_16
).
        ls_kpi_data-guid      = lv_guid_16.
        ls_kpi_data-kpi_date  = sy-datum.
        ls_kpi_data-kpi_time  = sy-uzeit.
        ls_kpi_data-kpi_table = ls_kpi_fields-kpi_table.
```

```
          ls_kpi_data-kpi_field = ls_kpi_fields-kpi_field.
          ls_kpi_data-value_old = <fs_oval_f>.
          ls_kpi_data-value_new = <fs_nval_f>.

          lv_extract = abap_false.

          APPEND ls_kpi_data TO lt_kpi_data.      " table for KPIs
        ENDIF.
      ENDLOOP.
    ENDLOOP.

*   Step 4 - extract KPI data to database table
    IF lines( lt_kpi_data ) > 0.
      MODIFY zkpi_data FROM TABLE lt_kpi_data.
      IF sy-subrc <> 0.
        MESSAGE w002(zmsg_shp_del). " message: Could not save KPI
      ENDIF.
    ENDIF.

ENDMETHOD.
```

Listing 8.1 KPI Coding for Method SAVE_AND_PUBLISH_DOCUMENT

The ABAP coding is separated into four key steps, which we discuss in the following subsections.

Step 1: Derive KPI Fields

Use a simple `SELECT ... INTO TABLE` to derive all entries from database Table ZKPI_FIELDS into internal Table LT_KPI_FIELDS.

Step 2: Establish Change Scenarios

There are three key KPI extract scenarios that Christine's solution allows for:

- Old and new values are different.
- A new delivery document is created.
- A specific field value has been reached.

You probably recognize these three scenarios from the layout overview of Table ZKPI_FIELDS (refer to Table 8.1). The coding at first uses `FIELD_SYMBOLS` (for tables, structures, and fields) to dynamically loop through either Table LIKP or Table LIPS

(which table isn't known at this stage because it's determined by the entries in Table ZKPI_FIELDS). Within the main loop around `FIELD-SYMBOL` table `<fs_nval_t>`, the code then differentiates between Table LIKP or Table LIPS because the table keys are different (`VBELN` versus `VBELN/POSNR`). Flag `LV_EXTRACT` is set for each record that should be appended to the extract database Table ZKPI_DATA later.

Step 3: Populate KPI Data Structure

Here, the field structure `LS_KPI_DATA` is populated from the currently processed delivery record. Static method `GUID_CREATE` from class `CL_RECA_GUID` is used to create a new GUID to ensure the KPI extract record is unique because the entries in the date/time fields aren't enough. Entries are stored in internal Table LT_KPI_DATA.

Step 4: Extract KPI Data to a Database Table

All entries in Table LT_KPI_DATA are appended to database Table ZKPI_DATA. A warning is given out if the `MODIFY` command was unsuccessful.

8.4 Summary

In this chapter, Christine and Sean demonstrated that enhancements can be used for more than just validations and messages. They created a small KPI extract system, which also helped to rectify a long-lasting support problem stemming from a lengthy overnight batch job that ploughed through change records.

You saw that a delivery process BAdI was chosen over a user exit, which was necessary to further filter key combinations such as plant/sales organization in the future. BAdIs take less effort than user exits, which are out-of-the-box fully integrated into SAP's Enhancement Framework. You can refer to Chapter 1 for further details on the Enhancement and Switch Framework.

Using this KPI example, you've also seen that sometimes it can be beneficial to extract data straight at the point of origin. In comparison, selecting change record entries at a later stage—using a batch job, for example—can be very time consuming and costly when system utilization increases and access is required around the clock.

In the next chapter, you'll learn how Christine and Sean enhance a standard SAP delivery monitor report by incorporating the KPI data from this chapter.

Enhancements can be an excellent way to enrich reports that are delivered by standard functionality with custom customer data.

9 Using a Customer Exit to Enhance the Outbound Delivery Monitor

After the new KPI data collection was introduced (refer back to Chapter 8), Byrell's distribution department started looking at different ways to use this data within their existing planning processes.

In Chapter 8, Christine facilitated a data extraction for delivery KPI reporting. Byrell's distribution department now wants to get more use out of this KPI data by incorporating it directly into their reports. In this chapter, you'll learn how Christine and Sean use the extracted KPIs to enhance a standard delivery monitor report, thereby combining two important sources of delivery data.

9.1 Business Scenario

One of the central tools used by Byrell's distribution planners is the standard outbound delivery monitor report (Transaction VL06O). This report has many uses; for example, it provides an overview of all deliveries according to picking dates, planned transport dates, or goods issue date.

Currently, distribution planners merge the standard delivery monitor report with the newly introduced KPI data using spreadsheets, but now the distribution department wants to create flexible and integrated reports. This improvement will eliminate the need to merge data in a spreadsheet. By doing this, Byrell's planners could benefit from the combined real-time information of the monitor report and the performance and history data of the KPI extract.

Sean attends a meeting with distribution planners at which the decision is made to change the delivery monitor to contain additional fields for the following:

- Number of KPI-related changes applied (for statistical purposes)
- Delivery age calculation to help with prioritization of backlog deliveries
- Flag to indicate whether the picking date or planned transport date has been changed within the past 24 hours (also to help with late changes to delivery priorities)

On closer inspection, Sean notices that only Byrell's first and last request actually requires the use of KPI data, while the second request (delivery age calculation) can actually be achieved with data already available in the report. It simply requires the addition of a new field to display the difference in days between two dates.

During the meeting with the delivery planners, Sean displays the delivery report, and the group debates options for enhancing it. Currently the report looks like the screen shown in Figure 9.1.

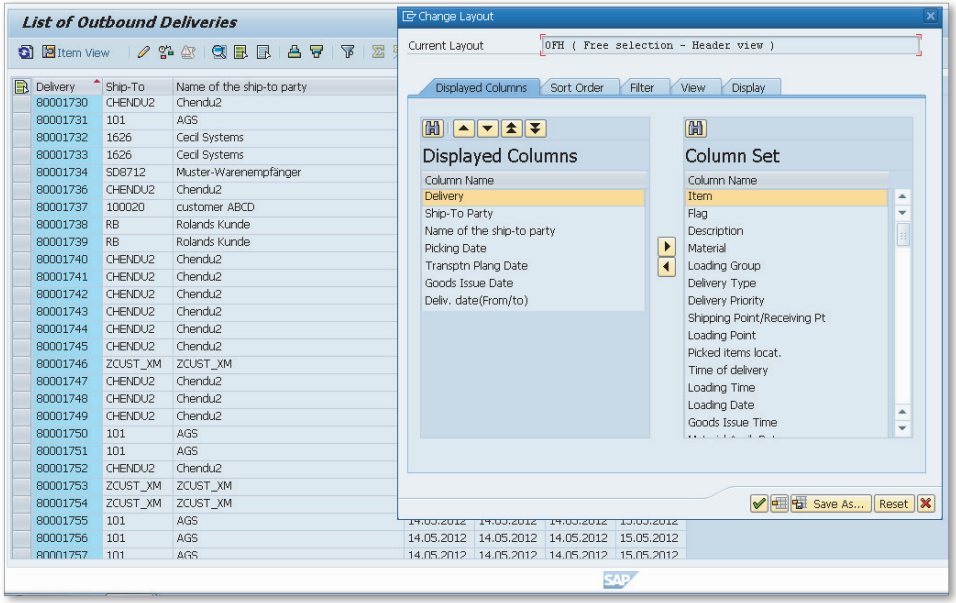


Figure 9.1 Outbound Delivery Monitor Report and Selector for Additional Fields

When it comes to enhancing this standard report, Sean knows that there are a couple of easy options:

1. Simply copy the report and amend a customer (Z*) version of it.
2. Use implicit enhancements to change the standard report.

Neither option really seems right to him, though. While he considers these solutions, Sean remembers that there is an SAP Note that deals with the enhancement of delivery monitor reports.

9.2 Finding the Right Enhancement

Sean logs on to the SAP Service Marketplace (*www.service.sap.com*) and uses the search string "enhance VL06O delivery monitor" in the SAP Notes section (see Figure 9.2).

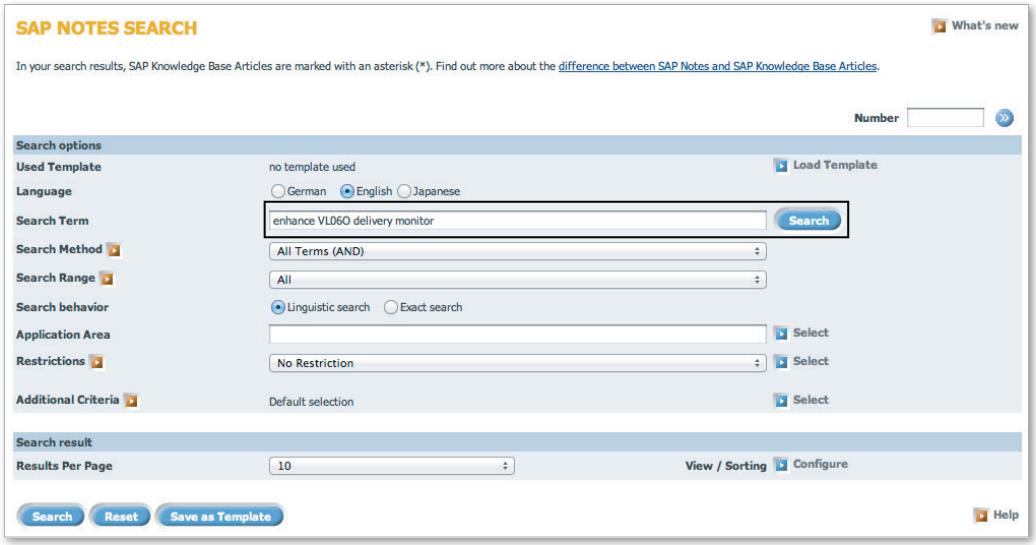


Figure 9.2 Search Screen for SAP Notes

Among the search results, Sean finds the one he was looking for—SAP Note 128150.

SAP Note 128150 – VL06: Designing Your Own Display Variants

At first, this note appears to be about the creation of report variants. However, upon closer inspection, the text also describes how additional fields (those not provided via standard delivery Tables LIKP, LIPS, and VBUP) can be included in the report and populated via customer exit `V50Q0001`. Additional fields are defined in append structure `LIPOVZ`, which is included in delivery monitor structure `LIPOV`. This structure acts as the ABAP List Viewer (ALV) line item structure for the delivery monitor.

For more information please refer to SAP Note 128150, which can be found on the SAP Service Marketplace (*www.service.sap.com*).

Tips & Tricks

SAP Notes are usually regarded as a resource for support solutions, but as you can see, sometimes a search on the SAP Service Marketplace can also reveal useful information regarding exits and enhancements.

9.3 Implementing the Solution

After reading through the note and refreshing his memory about how to enhance the delivery monitor using customer exit `V50Q0001`, Sean realizes that structure `LIPOVZ` plays a central role in this development. All new fields to be included in the report need to be added to this structure. In addition, coding has to be added to an exit routine where the additional KPI data is selected from the extract table and fed it into the new `LIPVZ` fields.

Based on this, Sean anticipates the following steps to be included in the functional specification that he will create for Christine:

1. Implement the customer exit routine.
2. Add new field entries (number of changes applied, delivery age, 24h change flag) to append structure `LIPOVZ`.
3. Add coding to derive the KPI data.
4. Add coding to calculate the delivery age.

Let's go through each of these steps in the following sections.

9.3.1 Implement the Customer Exit Routine

Christine's first task is to create the actual customer exit routine. To do this she uses Transaction CMOD, where she creates a new project for this particular exercise, which serves as a project envelope for the customer exit. She names the new project "ZDLV_MON" and clicks the Create button.

On the next screen, Christine has to specify a description for this new project (she enters "Customer Exit for Delivery Monitor Enhancement" in the Short text field). She clicks the Save button and then clicks the Enhancement Assignments button to navigate to the overview screen (see Figure 9.3).

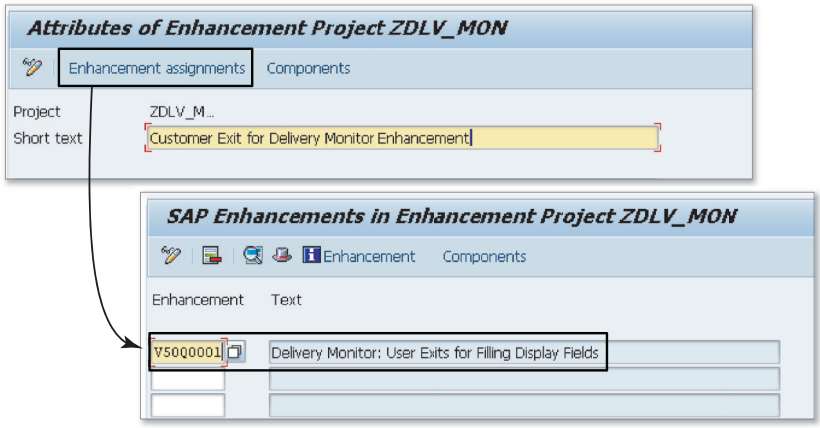


Figure 9.3 Specifying a Project Description and Navigating to Enhancements View

According to the note Sean found on the SAP Marketplace (128150), Enhancement `V50Q0001` should be used to include new fields in the delivery monitor. Christine enters the enhancement ID into the first column field and presses Enter to validate her entry. She then saves her work.

Now Christine clicks on the Components button. This is where she actually has to specify which implementation to use. In the case of this particular user exit, there is a choice between two different function module implementations. Per SAP Note 128150, Christine chooses the first function module (`EXIT_SAPLV50Q_001`) by double-clicking on it. This takes her straight to the function module editor (Transaction SE37) and displays the `INCLUDE` statement the user exit that was created for it by SAP developers (`INCLUDE ZXV50QU01`). She double-clicks on the `INCLUDE ZXV50QU01` statement and receives a popup, asking whether she wants to create the object (see Figure 9.4).

By clicking the Yes button (and specifying the development class and transport request details), she is taken to the editor, which displays a new and empty function group `Include` (see Figure 9.5). This is where she will add her coding later on in Sections 9.3.3 and 9.3.4. The coding can't be completed now because Christine hasn't added the new fields to the append structure `LIPOVZ` yet. She therefore saves her work by clicking the Save button.

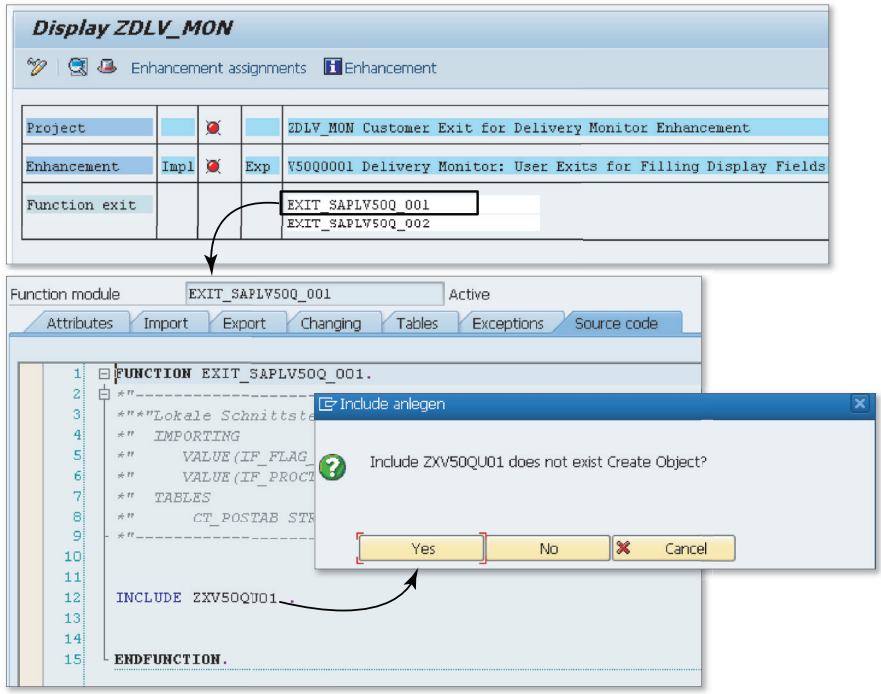


Figure 9.4 Creating a User Exit Function Module Implementation by Forward Navigation

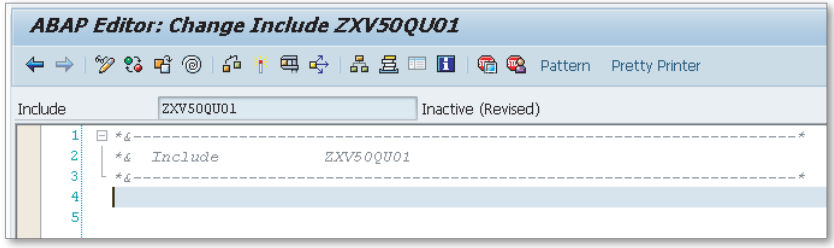


Figure 9.5 Empty Implementation of Function Group Include ZXV50QU01

9.3.2 Add New Field Entries to Append Structure LIPOVZ

Next, Christine adds the three new fields into the report line item structure (as mentioned in SAP Note 128150), which are described in the functional specification she received from Sean. She now has to define these fields as shown in Table 9.1.

Field Name	Type	Length	Description
DM_CHANGE_CTR	INT4	10	Number of changes applied
DM_DELV_AGE	INT4	10	Delivery age
DM_CHANGE_24	CHAR	1	Last 24h change flag

Table 9.1 Additional Fields for the Line Item Structure Append LIPOVZ

Christine will now create new data elements for these data fields and then assign the fields and data elements to structure LIPOVZ. Finally, she will perform a simple test to establish if structure LIPOVZ is picking up the new fields.

Creating New Data Elements

First, Christine defines new data elements for all three report fields, which will be used for the new fields in append LIPOVZ. She accesses Transaction SE11, enters "ZDEL_MON_CHNGES" on the initial selection screen, and clicks on the Create button to create a new data element.

She then assigns a short description for the new field ("Delivery Monitor – number of changes applied"). When it comes to defining the field type, Christine isn't sure how many changes the new report field will have to handle. To be on the safe side, she specifies it as INT4 with a length of 10.

Finally, she assigns field labels by clicking on the Field Label tab for the new data element (see Figure 9.6) and then activates her new data element by clicking the Activate button.

Note

Creating data elements DM_DELV_AGE (to display the delivery age) and DM_CHANGE_24 (as the 24h change flag) requires the same steps as were just used.

Assigning New Fields and Data Elements to Append Structure LIPOVZ

Christine now adds new fields into append structure LIPOVZ by using the new data elements she has just created. In Transaction SE11, she selects structure LIPOVZ and clicks on the Change button. Figure 9.7 shows the field names and data elements she adds to the append structure by typing the field names into the Component column.

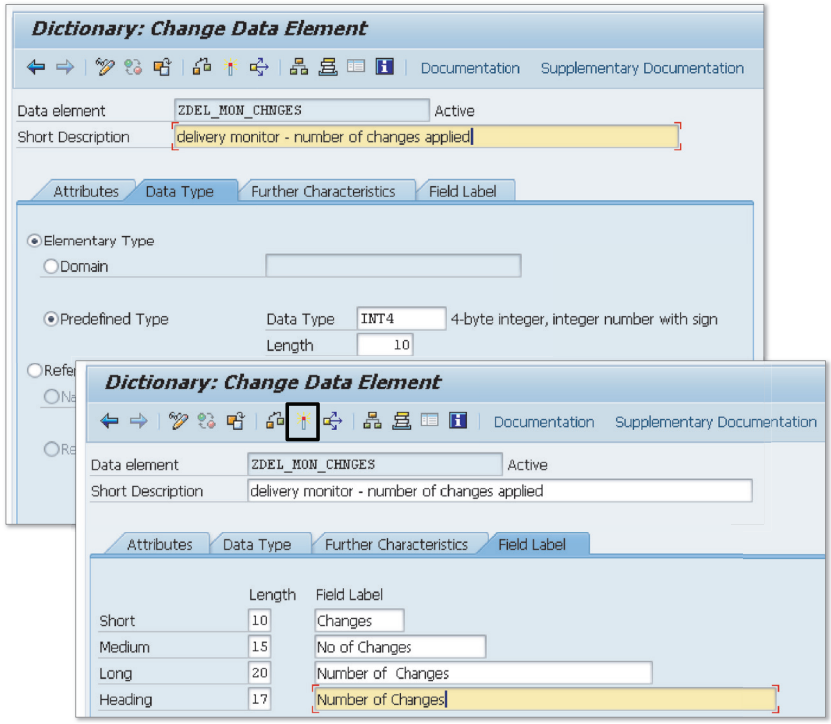


Figure 9.6 Typing and Field Label for Element ZDEL_MON_CHNGES

All three fields are typed with the data elements that she just created in the previous step (ZDEL_MON_CHNGES, ZDEL_MON_DELIVERY_AGE, and ZDEL_MON_DATE_CHANGE_FLAG).

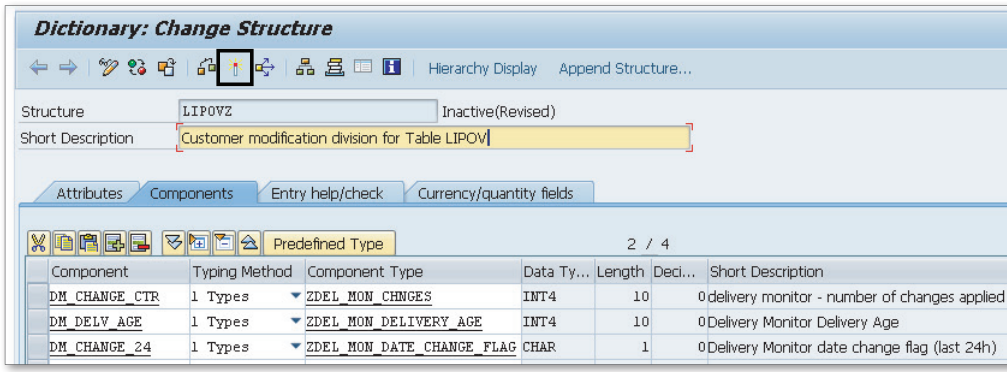


Figure 9.7 Append Structure LIPOVZ with the New Delivery Monitor Fields

After she has finished adding the three news fields, she clicks on the ACTIVATE button.

Testing the New Fields in the Delivery Monitor (VL06O)

Although Christine has not added any logic to populate her three new fields yet, she wants to test whether they appear in the report. She therefore starts Transaction VL06O and runs the report. In the ALV report screen, she selects the new fields from the ALV field selector popup (see Figure 9.8), which is displayed by clicking on the CHANGE LAYOUT button ( ).

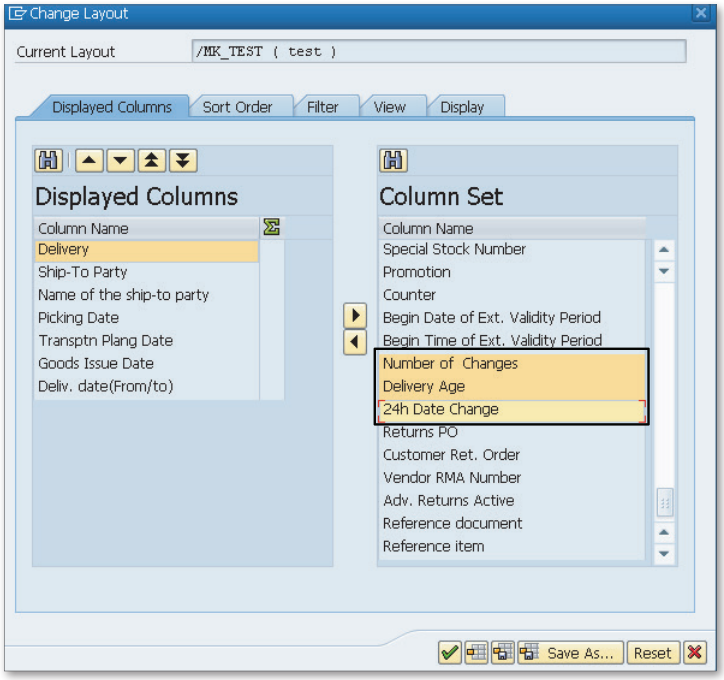
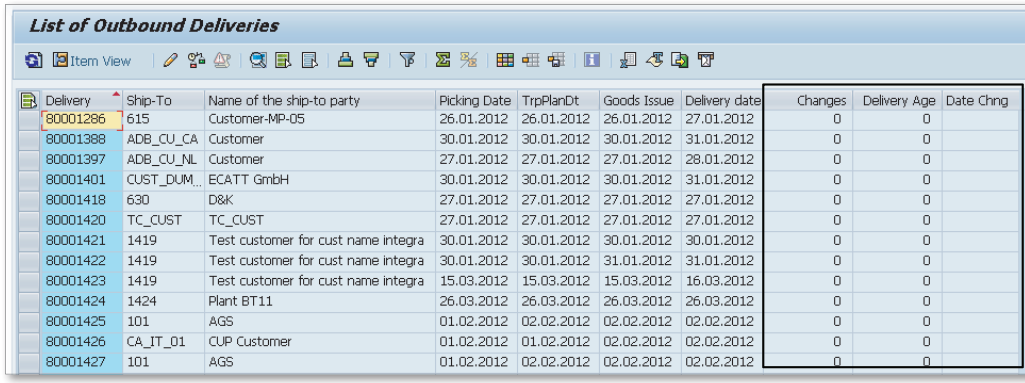


Figure 9.8 ALV Field Selector Displays Added New Fields

Once selected, the values are also displayed in the report (albeit without values, because no coding has been added to populate them yet). Nevertheless, it's a pass for Christine's first test of the enhanced delivery monitor (see Figure 9.9).



Delivery	Ship-To	Name of the ship-to party	Picking Date	TrpPlanDt	Goods Issue	Delivery date	Changes	Delivery Age	Date Chng
80001286	615	Customer-MP-05	26.01.2012	26.01.2012	26.01.2012	27.01.2012	0	0	
80001388	ADB_CU_CA	Customer	30.01.2012	30.01.2012	30.01.2012	31.01.2012	0	0	
80001397	ADB_CU_NL	Customer	27.01.2012	27.01.2012	27.01.2012	28.01.2012	0	0	
80001401	CUST_DUM...	ECATT GmbH	30.01.2012	30.01.2012	30.01.2012	31.01.2012	0	0	
80001418	630	D&K	27.01.2012	27.01.2012	27.01.2012	27.01.2012	0	0	
80001420	TC_CUST	TC_CUST	27.01.2012	27.01.2012	27.01.2012	27.01.2012	0	0	
80001421	1419	Test customer for cust name integra	30.01.2012	30.01.2012	30.01.2012	30.01.2012	0	0	
80001422	1419	Test customer for cust name integra	30.01.2012	30.01.2012	31.01.2012	31.01.2012	0	0	
80001423	1419	Test customer for cust name integra	15.03.2012	15.03.2012	15.03.2012	16.03.2012	0	0	
80001424	1424	Plant BT11	26.03.2012	26.03.2012	26.03.2012	26.03.2012	0	0	
80001425	101	AGS	01.02.2012	02.02.2012	02.02.2012	02.02.2012	0	0	
80001426	CA_IT_01	CUP Customer	01.02.2012	01.02.2012	02.02.2012	02.02.2012	0	0	
80001427	101	AGS	01.02.2012	02.02.2012	02.02.2012	02.02.2012	0	0	

Figure 9.9 Delivery Monitor Displays the Newly Defined Columns

9.3.3 Add Coding to Derive KPI Data and Calculate Delivery Age

Christine then goes back to Transaction SE37 where she displays function module `EXIT_SAPLV50Q_001` and double-clicks on `INCLUDE ZXV50QU01`. She then goes into change mode by clicking on the Display/Change button. To populate the new `LIPOV` structure fields with the correct values, Christine adds the coding shown in Listing 9.1.

```
*&---------------------------------------------------------------------*
*&  Include           ZXV50QU01
*&---------------------------------------------------------------------*
DATA: ls_postab      TYPE lipov,
      lv_change_made TYPE abap_bool VALUE abap_false,
      lv_erdat       TYPE sy-datum,
      lv_prev_day    TYPE sy-datum,
      lt_kpi_data    TYPE STANDARD TABLE OF zkpi_data,
      ls_kpi_data    TYPE zkpi_data.

LOOP AT ct_postab INTO ls_postab.
* Step 1 - Get KPI changes for particular delivery document
  SELECT *
    FROM zkpi_data
    INTO TABLE lt_kpi_data
   WHERE vbeln = ls_postab-vbeln.

  IF sy-subrc = 0 AND lines( lt_kpi_data ) > 0.
    ls_postab-dm_change_ctr = lines( lt_kpi_data ).
    lv_change_made = abap_true.
```

```
  ENDIF.

* Step 2 - check whether change flag to be set
  lv_prev_day = sy-datum - 1.     " date for yesterday
  LOOP AT lt_kpi_data INTO ls_kpi_data
       WHERE kpi_date >= lv_prev_day.
    CASE ls_kpi_data-kpi_date.
      WHEN sy-datum.
        IF ls_kpi_data-kpi_time <= sy-uzeit.
          ls_postab-dm_change_24 = abap_true.
          lv_change_made = abap_true.
        ENDIF.
      WHEN lv_prev_day.
        IF ls_kpi_data-kpi_time >= sy-uzeit.
          ls_postab-dm_change_24 = abap_true.
          lv_change_made = abap_true.
        ENDIF.

    ENDCASE.
  ENDLOOP.

* Step 3 - calculate delivery age
* only interested in documents that have been picked,
* and goods issued
  IF NOT ( ls_postab-koquk = 'C' AND  " conf. status
           ls_postab-kostk = 'C' AND  " picking status
           ls_postab-wbstk = 'C' ).   " GI status
    SELECT SINGLE erdat
      FROM likp
      INTO lv_erdat
     WHERE vbeln = ls_postab-vbeln.
    IF sy-subrc = 0.
      ls_postab-dm_delv_age = sy-datum - lv_erdat.
      lv_change_made = abap_true.
    ENDIF.
  ENDIF.

* Step 4 - modify ct_postab table
  IF lv_change_made = abap_true.
    MODIFY ct_postab FROM ls_postab.
  ENDIF.
```

```
  lv_change_made = abap_false.
  REFRESH lt_kpi_data.
ENDLOOP.
```

Listing 9.1 INCLUDE ZXV50QU01 to Evaluate and Populate Report Fields

Christine's coding loops around internal interface Table CT_POSTAB, which contains all report lines and has a `LIPOV` structure. Processing is separated into four steps as you can see in the code, with the first three steps populating each of the custom fields Christine added earlier.

Step 1

The process starts with a select on the KPI data table for the current delivery document. Items found are stored in internal Table LT_KPI_DATA. A `lines()` function is used to establish the number of lines in Table LT_KPI_DATA and store it in `LS_POSTAB-DM_CHANGE_CTR`, our first custom field. If the field value was set, then the `LV_CHANGE_MADE` flag is set, which will display the flag to indicate that the delivery was changed within the past 24 hours.

Step 2

The next step calculates yesterday's date by subtracting 1 from today's date. It then loops around all KPI data lines of yesterday (or today) and finds out whether the KPI-related change was made within the past 24 hours or not. If a change was made within the last 24 hours, then flag `DM_CHANGE_24` is set in structure `LS_POSTAB`. Again, flag `LV_CHANGE_MADE` is set.

Step 3

Christine's code calculates the age of the delivery (i.e., establishes how many days have passed from the creation of the delivery to today). However, per Sean's specification, the program also needs to filter out any documents that have already been completed, confirmed, and goods issued. For these documents, no delivery age needs to be calculated because they are regarded as completed by Byrell's delivery process. Christine's code checks three status fields (`koquk`, `kostk`, and `wbstk`) to ensure that only those documents are processed.

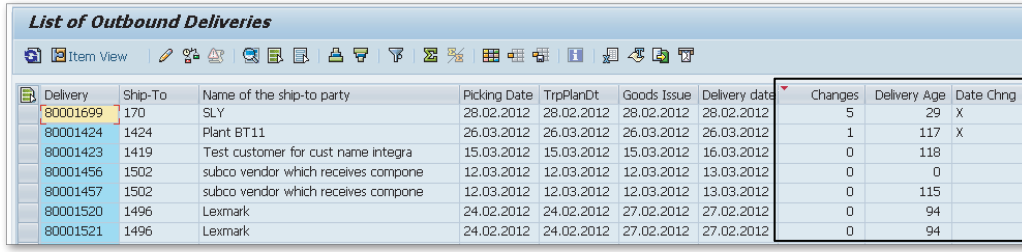
The code selects the creation date (field `ERDAT`) from delivery header Table LIKP, calculates the difference in days from today's date (field `SY-DATUM`), and stores it in field `DM_DELV_AGE` of structure `LS_POSTAB`. Once again, flag `LV_CHANGE_MADE` is set.

Step 4

The customer exit checks whether `LV_CHANGE_MADE` is set to "true." If this is the case, then the current `CT_POSTAB` is changed using structure `LS_POSTAB`.

9.3.4 Testing the Enhanced Delivery Monitor

Finally, Christine demonstrates the delivery monitor report to Sean, including the three new custom fields (see Figure 9.10).



Delivery	Ship-To	Name of the ship-to party	Picking Date	TrpPlanDt	Goods Issue	Delivery date	Changes	Delivery Age	Date Chng
80001699	170	SLY	28.02.2012	28.02.2012	28.02.2012	28.02.2012	5	29	X
80001424	1424	Plant BT11	26.03.2012	26.03.2012	26.03.2012	26.03.2012	1	117	X
80001423	1419	Test customer for cust name integra	15.03.2012	15.03.2012	15.03.2012	16.03.2012	0	118	
80001456	1502	subco vendor which receives compone	12.03.2012	12.03.2012	12.03.2012	13.03.2012	0	0	
80001457	1502	subco vendor which receives compone	12.03.2012	12.03.2012	12.03.2012	13.03.2012	0	115	
80001520	1496	Lexmark	24.02.2012	24.02.2012	27.02.2012	27.02.2012	0	94	
80001521	1496	Lexmark	24.02.2012	24.02.2012	27.02.2012	27.02.2012	0	94	

Figure 9.10 Test Run of the Enhanced Delivery Monitor Report

As introduced in Chapter 8, the flexible setup of the new KPI solution now enables Byrell to introduce new extraction fields and criteria more easily. In return this saves time and money, because it isn't necessary to change report or extract coding whenever a different field is to be reported on. In this chapter, Christine and Sean went one step further and introduced the KPI extract figures into a standard report by using a customer exit, thus eliminating the need for Byrell's planners to merge data into spreadsheets or other external documents. Planning is now made easier and quicker as all data is brought into one place, delivery monitor VL06O.

9.4 Summary

This chapter showed how a user exit can be used to enhance an SAP standard report. When it comes to important standard reports such as the delivery monitor, SAP has provided various options to use standard reports and yet still add custom fields and processing. Too often, developers simply copy the existing report and create a Z or Y version of it, thereby risking the loss of new and additional functionality whenever the system is upgraded.

Another novel approach in this chapter was the way Sean actually came across the user exit. SAP Notes represent another way of finding out about exits and shouldn't be overlooked. In your own enhancement research work, make it a habit to also check for any suitable SAP Notes—you might be surprised by what you find.

In the next chapter, you'll learn how to introduce VOFM routines as a nifty and lightweight way to perform an invoice split.

PART III
Enhancements in Billing

Specific industries or customers can necessitate custom changes to billing processes. In this chapter, you will look at a different enhancement method, which is more closely related to the IMG than what has been discussed so far.

10 Invoice Splitting Using a VOFM Routine

Most billing processes in Byrell's Sales and Distribution (SD) solution are delivery-based, meaning that a new invoice is created using data of one or many combined delivery documents. However, there are occasions when the billing process needs to force an invoice split to produce separate billing documents, such as in the Fast Moving Consumer Goods (FMCG) sector.

In these cases, using VOFM routine exits is essential: they're slightly different from other enhancements because they're driven by copy control configurations, and are therefore more closely embedded into the SD Implementation Guide (IMG). Please refer to Chapter 1, Section 1.2 for a general overview of customer routines (VOFM).

In this chapter, we'll introduce you to VOFM routines as a method to enhance SAP Sales and Distribution. You'll learn how a data transfer routine is used to create separate invoices for specific item categories, thus producing one invoice per customer for each item category.

10.1 Business Scenario

Byrell recently welcomed a new client, a brewery and big player within the FMCG sector. With a planned go-live for this new client within the next four months, configuration and development work are already underway. Among other client-specific tasks, the Byrell SAP team works with the client's key users to shape enhancements to SD billing.

A business requirement document about billing processes, which has been agreed upon with the brewery, contains the following short statement:

Daily billing due list processing and invoice split for domestic customers

Domestic invoices (billing type Z001) should be split for all customers by duty/non-duty items for standard food and beverage products, if the customer master indicator instructs to do so.

Sean has been given the task to work on billing processes. He already knows that some of the new client's products are liable to alcohol duty. The client therefore requires a solution that explicitly reports on such items to make tax declarations easier. For example, the to-be billing needs to ensure that end customers receive separate invoices for ordered products that are or are not liable to alcohol duty.

10.2 Finding the Right Enhancement

Sean looks at the requirements for this particular task and considers enhancement options.

10.2.1 Understanding the Invoice Split

First of all, it's important to understand how the invoice split is achieved. Within billing header Table VBRK, the centerpiece of invoicing split functionality is field `ZUKRI` (combination criteria in the billing document). This field is a special character field that can force invoice splits by adjusting the contents.

In a functional specification Sean prepares for Christine, he uses the following simple example graphic shown in Figure 10.1.

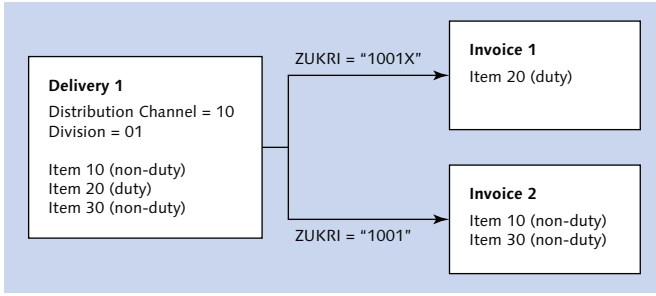


Figure 10.1 Delivery-Based Invoice Split for Duty and Non-Duty Materials

Delivery #1 on the left contains three items, of which two are non-duty items (items 10 and 30) and one duty item (item 20). Field VBRK-ZUKRI is key when it comes to this split because the system processes each item during billing and assigns different character strings for duty and non-duty items. In this case, the field will produce two separate invoices (Invoice #1 and Invoice #2). Figure 10.1 shows how the split is forced by concatenating an "X" character to the end of ZUKRI.

10.2.2 Finding Exits or Enhancements

Because Sean needs to be able to influence the value of VBRK-ZUKRI, he needs to look for exits or enhancements that give access to this field before the invoice is created. To do this, he uses the Repository Infosystem in Transaction SE80 to search for exits. Searching for ABAP programs that begin with RV60* reveals includes four user exits (see Figure 10.2).

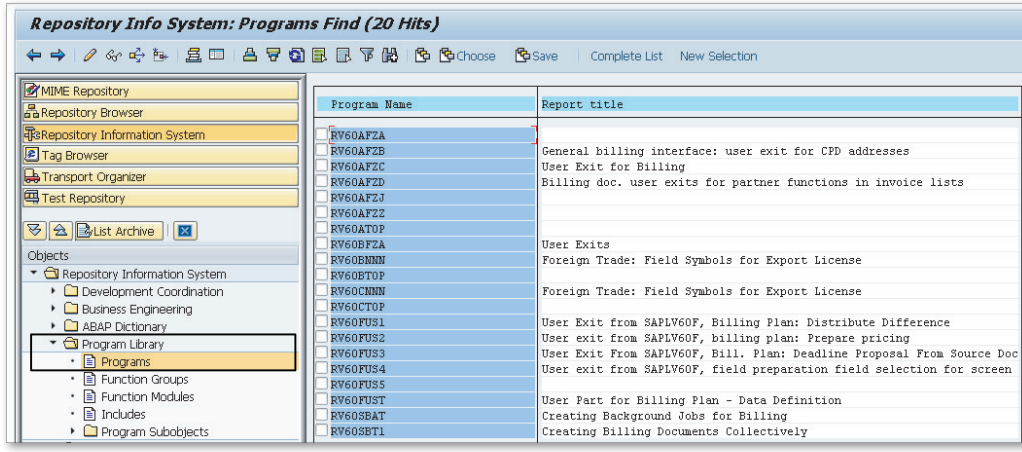


Figure 10.2 Search Result Set for ABAP in the RV60* Naming Range

Note

You have seen in previous chapters such as Chapter 4, Section 4.2, how the search for ABAP programs within the Repository Infosystem is executed. Please refer to this chapter to view more detailed steps.

All of the 20 found programs contain one or more user exits, so there are potentially a lot of user exits for Sean to search through. However, ABAP RV60AFZC stands out

because it contains USEREXIT_FILL_VBRK_VBRP. This could be just the ticket for Sean's requirements because this exit is called during the creation of the billing document, invoice header, and item fields (see Figure 10.3) to be amended.

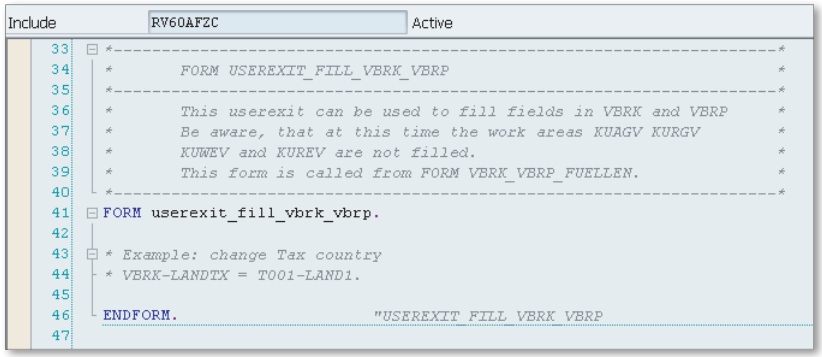


Figure 10.3 USEREXIT_FILL_VBRK_VBRP without Modification and Enhancement

When revisiting the initial requirements, Sean realizes that the enhancement should only cover billing type Z001. In the case of a user exit, this either means that the billing document type needs to be hard coded or maintained using a custom customer table. Hard coding is the least favorable option because every addition or change to document type filtering within the exit requires a program change. Speaking from experience, Sean knows that these kinds of exits can undergo changes, particularly when it comes to the document types. Therefore, using a custom customer table to deliver the applicable document types is the smarter solution.

However, he knows that he can use data transfer routines in Transaction VOFM to achieve the same goal and won't require hard coding or creating tables. This is because data transfer routines can be assigned to different billing types. Sean has a closer look at Transaction VOFM to locate the right data transfer routine.

10.2.3 Transfer Routines

Sean accesses Transaction VOFM and selects Data Transfer • Billing documents from the top menu bar. This takes him to the overview of Data Transports Billing Documents overview table. He double-clicks the first entry Inv.split (sample), which takes him to the coding of this sample routine (see Figure 10.4).

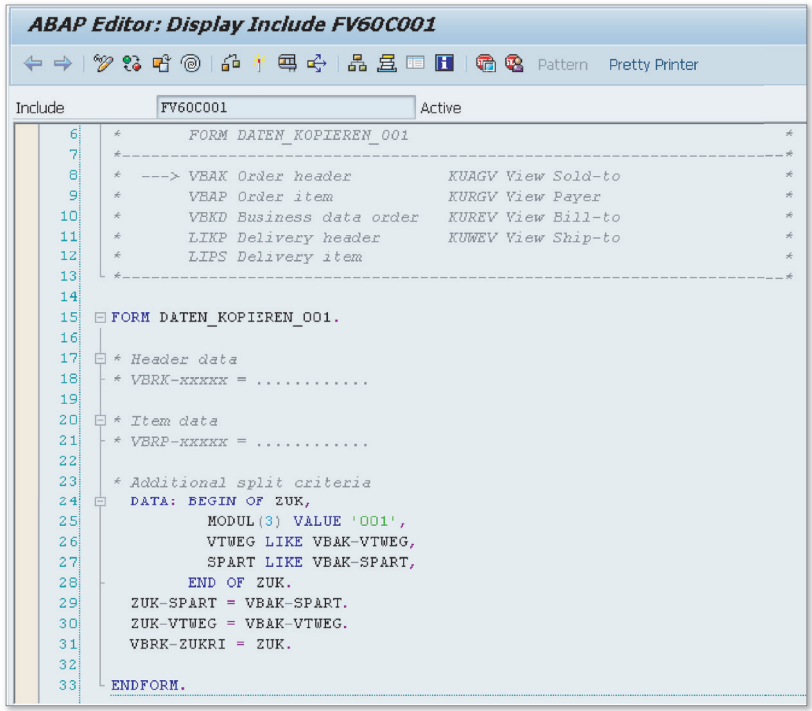


Figure 10.4 Coding for the Invoice Split Example Routine

Routine 001 is a sample invoice split routine that is delivered with all SAP SD systems and contains code that amends the 40-character `ZUKRI` field.

Routines for Copying, Data Transfer, Requirements, and Formulas

It's worthwhile emphasizing at this point why Sean decided to look for a data transfer routine and not a copy routine within Transaction VOFM. Consider the following items and factors:

- **Copy routines**
  These allow to you to copy a source document into a target document under specific conditions, for example, to copy one or multiple orders into one delivery document. This is not a requirement here.
- **Data transfer routines**
  These allow you to specify which field is to be transferred from the source to the target document. This obviously also allows for amendments on the field level such as for field `ZUKRI` ("combination criteria in billing document"), which is what Sean needs to do in the current example.

- **Requirements**
  These check for specific conditions before pricing conditions are called, for example. Requirements are found within pricing, messaging, and account determination. This is not needed in the current example.
- **Formulas**
  These are used within pricing and include calculations. This is also not required here.

For further information on customer-defined VOFM routines, please refer to Chapter 1, Section 1.2.

Sean decides that using a data transfer routine in Transaction VOFM is the best way to deal with this development request because the routine will be able to handle the invoice split with relatively little custom code. In addition, he can configure the routine only to be called for specific billing types, which is ideal.

10.3 Implementing the Solution

Sean now has to specify the steps he needs to take to facilitate Christine's development work. We'll go over each process in the following sections.

10.3.1 Copy the Existing VOFM Routine

Sean creates a new VOFM data transfer routine because rather than amending existing routines, it's better to copy an existing routine in this method. This keeps routines better organized and allows for an easier overview when a large number of VOFM routines are created. To do this, he follows these steps:

1. Executes Transaction VOFM, and selects Data Transfer • Billing Documents from the menu bar on the top.
2. Copying an existing entry works slightly different here when compared to other SAP standard screens. On the following screen, Sean enters "901" over the existing entry for Routine number 001, which lies within the allowed customer number range for routines of 600 to 999. He then gives his new routine a description of "invoice split" and presses Enter (see Figure 10.5).

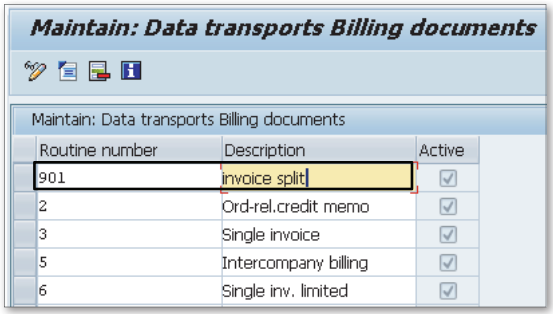


Figure 10.5 Copying an Existing Data Transfer Routine in Transaction VOFM

3. The system confirms that the routine was copied and can be amended with the popup message: "R3TR generated, changes possible." Sean then specifies a transport request for his new routine.
4. After the system takes him back to the list of routines, he scrolls down and finds his new custom routine in a deactivated state (see Figure 10.6).

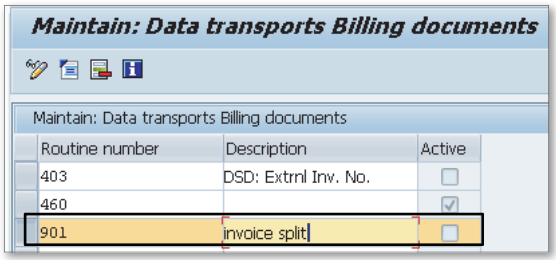


Figure 10.6 The Newly Created Routine 901, Before Activation

5. Sean now selects the line for routine 901 and chooses Edit • Activate from the top menu bar. He is then once again prompted for a transport request. On return to the table overview screen, he can now see that the routine has been activated because the Active box is ticked for routine 901.
6. Sean double-clicks on routine 901 and goes to the coding of the new routine. He will leave the actual coding to Christine, but he notices that the system has put a comment into the generated code to remind consultants that this routine was copied (see Figure 10.7).

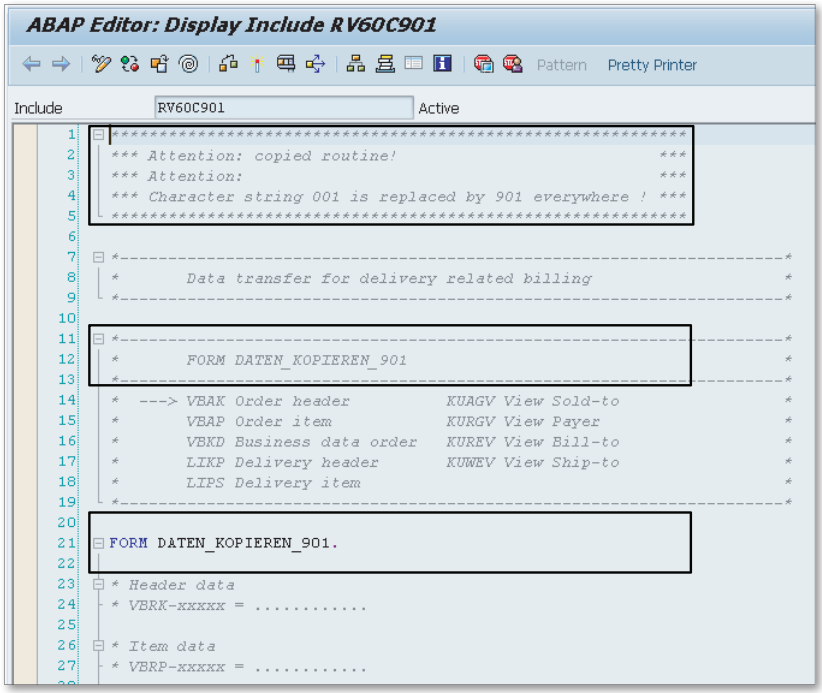


Figure 10.7 Coding of Custom Routine 901, Generated via Copy

10.3.2 Create an Implicit Enhancement within the Data Transfer Routine

Christine takes over at this point because the next two development steps are ABAP-related (and also because Sean asked her to show him again how to create an implicit enhancement, which is the next step).

In the past, Sean would have expected Christine to modify the newly generated routine because it's outside the customer name space (`RV60C901`). However, because the Enhancement Framework has emerged, this should now be done using an implicit enhancement.

Christine locates `INCLUDE` program `RV60CNNN` using the standard ABAP Editor in Transaction SE38 to navigate to ABAP `INCLUDE RV60C901` and then implement an implicit enhancement.

Why Select RV60CNNN First?

You might be wondering why Christine selects ABAP `RV60CNNN` (which actually has an `INCLUDE` statement for `RV60C901` in it) first. This is a way to "trick" the ABAP Editor to allow the display of implicit enhancement points. Implicit enhancement points need to be included in an Enhancement Framework program, but because of the way custom VOFM routines are generated, the editor can't see a link between the `INCLUDE` program and the higher level ABAP. In fact, if you try and select SHOW IMPLICIT ENHANCEMENT OPTIONS within `RV60C901`, you receive the message: "No implicit enhancement options can be displayed."

1. To "trick" the editor, Christine displays the higher level ABAP `INCLUDE RV60CNNN`. She navigates through to `RV60C901` by double-clicking on `INCLUDE RV60C901`, and then clicking on the ENHANCEMENT button (see Figure 10.8).

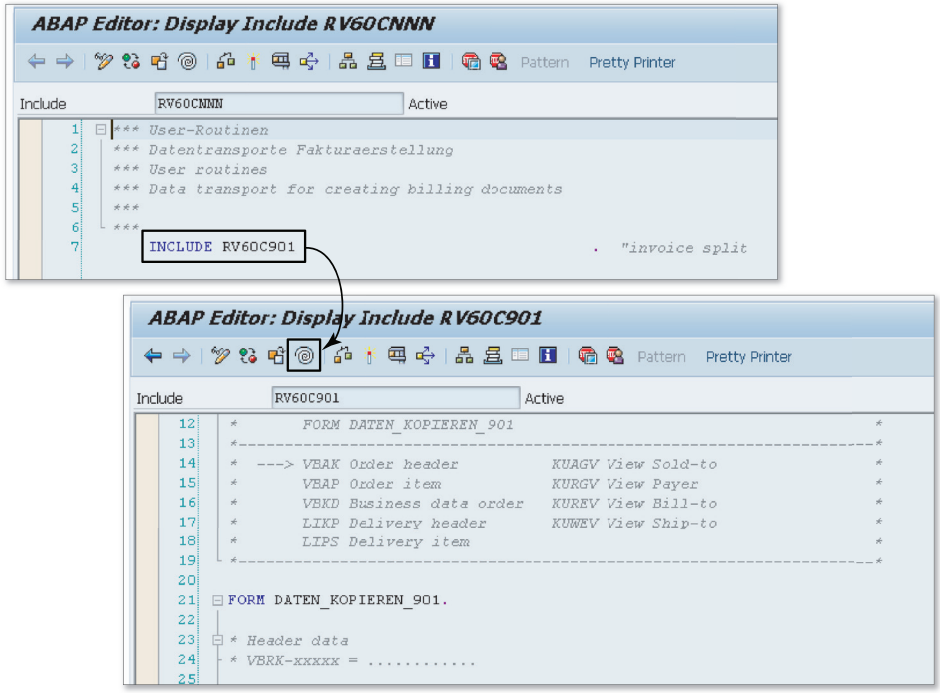


Figure 10.8 Navigate from RV60CNNN to RV60C901 and Choose Enhancement Mode

2. Christine then right-clicks into the code and selects Enhancement Operations • Show Implicit Enhancement Options. She then examines the various options to create implicit enhancements by looking at the highlighted code sections (see Figure 10.9).

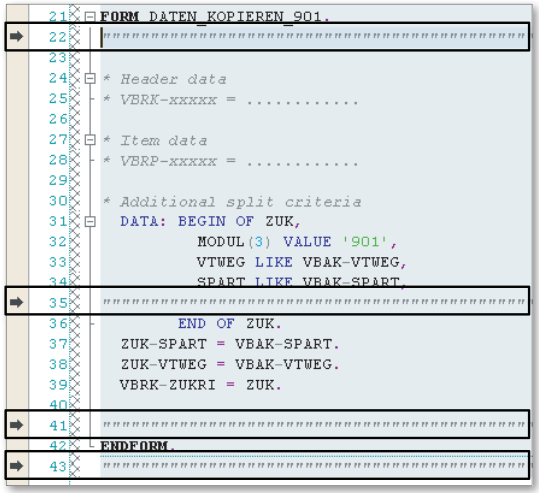


Figure 10.9 Implicit Enhancement Option within Form DATEN_KOPIEREN_901

Upon closer inspection, Christine determines that she needs to implement the second implicit enhancement point within the data structure declaration of `ZUK` as well as the third implicit enhancement point (before the `ENDFORM` statement), to add custom coding.

1. Christine creates an implementation for the second option by right-clicking on the dotted line and choosing Enhancement Operations • Create Implementation.
2. In the popup that appears, she clicks the Create button for a new implementation.
3. She gives the implementation a name of "zenh_impl_rv_inv_split" and a description of "Enhancement Implementation for Invoice Split," and then clicks the green checkmark icon.
4. The next screen prompts her for the development class and transport object, which she assigns. After this, she selects the newly created implementation and clicks the green checkmark button (see Figure 10.10).

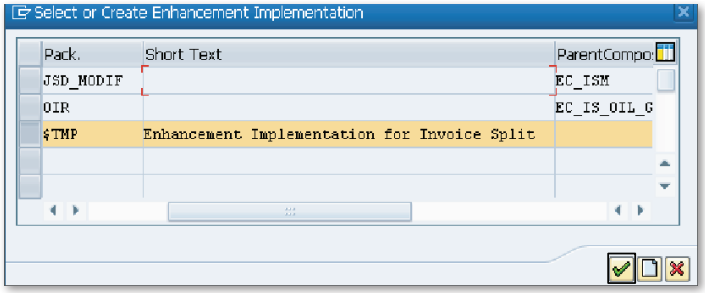


Figure 10.10 Selecting the New Implementation

5. She is then prompted with the new implementation within the code and saves her work.

After that, Christine repeats the same steps again for the third implementation option in `RV60C901`. She also doesn't forget to save her work. The custom routine now looks as shown in Figure 10.11.

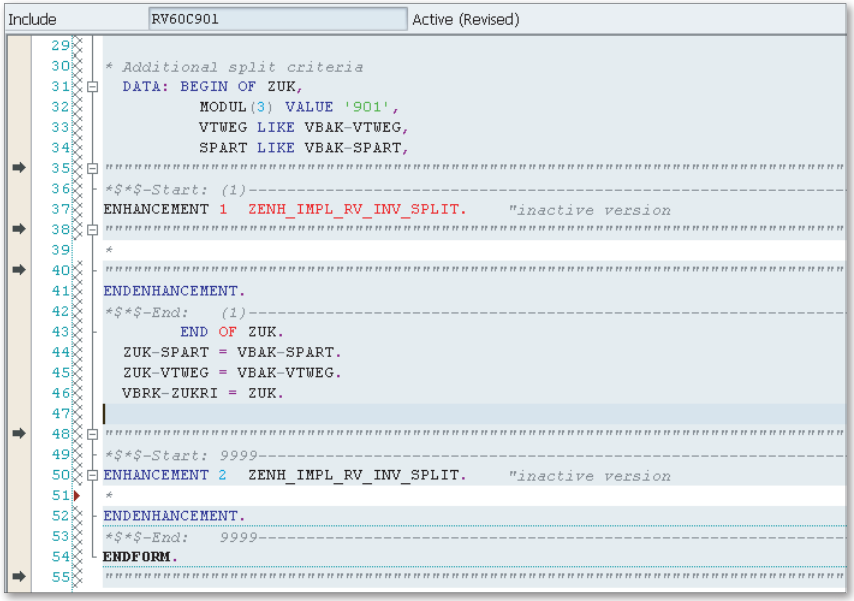


Figure 10.11 RV60C901 with Two Implicit Enhancement Implementations

10.3.3 Add Custom ABAP Coding

Christine is now ready to add her custom coding to these two implicit enhancement implementations:

1. First, she extends the ZUK declaration with the sales document category PSTYV by adding the line shown in Listing 10.1 to the declaration.

```
ENHANCEMENT 1  ZENH_IMPL_RV_INV_SPLIT.     "active version
      DATA PSTYV TYPE PSTYV.
ENDENHANCEMENT.
```

Listing 10.1 Sales Document Category Field PSTYV in Implicit Enhancement

   Field PSTYV is part of invoicing item Table VBRP and needs to be part of the ZUKRI structure to determine the invoice split.

2. Christine then adds the coding in Listing 10.2 to the second implicit enhancement implementation she created.

```
ENHANCEMENT 2  ZENH_IMPL_RV_INV_SPLIT.     "active version
  IF vbap-pstyv = 'ZDHN' OR vbap-pstyv = 'ZTAN'.
    zuk-pstyv  = vbap-pstyv.
    vbrk-zukri = zuk.
  ENDIF.
ENDENHANCEMENT.
```

Listing 10.2 Code to Ensure Split is Performed for Categories ZTAN and ZDHN

   This code assigns the sales document item category to the item category in Christine's local structure ZUK, which she declared earlier in Listing 10.1. By populating VBRK-ZUKRI for duty (ZTAN) and non-duty (ZDHN) items, the split will be done for these different item categories.

Hard Coding Values versus Customizing Tables

Christine hard coded item categories in the preceding example of the enhancement coding to simplify the example and focus on the invoice split.

If you anticipate that the assignment of item categories in your own solution is likely to change in the future, then it would be better to use a custom Z table to maintain these entries and check against this table within the enhancement.

10.3.4 Apply Routine to Copy Control

The final step in this procedure is for Sean to assign routine 901 to copy control, which specifies the values to be transferred from the delivery document to the billing document for the sales order cycle.

This is either done using Transaction VTFL or using the IMG menu path Sales and Distribution • Billing • Billing Documents • Maintain Copying Control for Billing Documents. Once Sean navigates to the transaction, he completes the following steps:

1. Sean double-clicks on the line Copying Control: Delivery Document to Billing Document in the Choose Activity screen because he wants to configure copying control for delivery-related billing. The next screen displays the copy control header information.
2. Sean wants to amend the configuration for the Outbound Delivery (LF) to Invoice (F1) and therefore selects the first line. He then double-clicks the Item node on the left of the screen (see Figure 10.12).

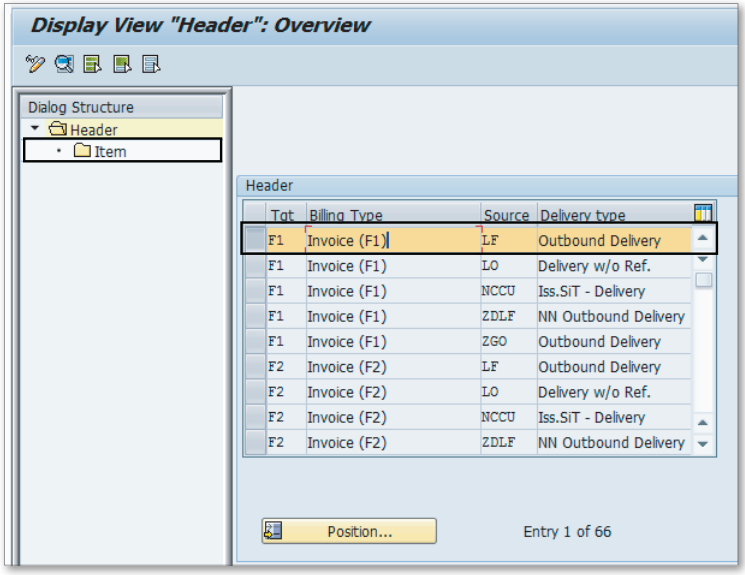


Figure 10.12 Copy Control for Outbound Delivery (LF) to Invoice (F1)

3. The next screen is the item overview, where Sean selects and double-clicks on the appropriate Item Category ZTAN.

4. In the next screen, Sean has to assign the new routine 901. He enters this in the DATA VBRK/VBRP field, where the data transfer routine is assigned as a code exit (see Figure 10.13).

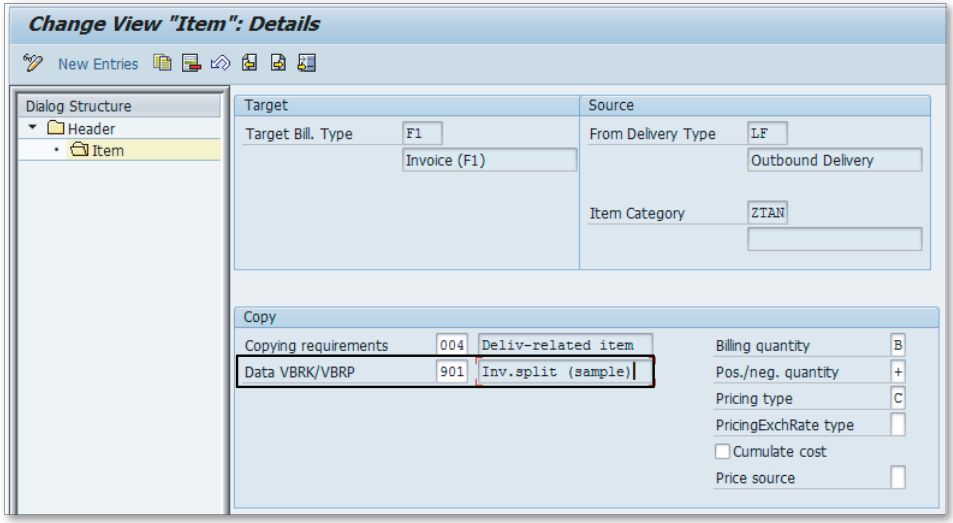


Figure 10.13 Assign Routine 901 as a Data Transfer Routine to Category ZTAN

5. Finally, Sean repeats the same procedure once more for item category ZDHN by going back to the item overview, selecting item category ZDHN, double-clicking on it, and entering the routine for this item category, too.

As you were able to see, there would have been the option to use user exits for this particular exercise, too, but the fact that VOFM routines offer an integration of document types and item categories trumped here. In general, however, it's perfectly fine to use a user exit here. You just have to keep in mind that any document types or item categories either require a hard-coded validation or custom Z tables to avoid future code changes.

10.4 Summary

Once again, implicit enhancements were used instead of modifications, which are fully integrated into the Enhancement Framework, and offer higher flexibility for developers. In addition, implicit enhancements provide an upgrade-safe way to enhance standard code and allow different versions within the same routine (as

shown in Listing 10.1 and Listing 10.2, where Christine created a declaration and a code enhancement).

Another interesting observation is that the coding of the exit routine is relatively short and trivial, which emphasizes the importance of the selection of the correct exit. This is the key to a lightweight and flexible enhancement solution: using as little custom code as necessary and leveraging as much standard functionality as possible.

In the next chapter you will learn how to enhance billing documents by changing numbering ranges and billing references using user exits.

A business document may need to include more than just the data captured within standard processes. There are a few simple ways to enhance a business user's information to make their job easier.

11 Changing Invoice Reference and Numbering Range during Document Creation

Business documents form part of real-world business processes; they are passed on between different parties and involved in different business steps. However, sometimes the standard Sales and Distribution system won't include all of the important business information on a screen or document, since this important information will vary between industry and company. SD billing processes in SAP contain a wide range of user exits (refer to Appendix A), which you can leverage for improvements such as this.

In this chapter, you'll follow along as Sean and Christine find two relatively small enhancements that will improve the readability of one of Byrell's documents, and thereby reduce human error. They'll illustrate how to tweak or enrich field contents slightly to make real-world processes just a little bit easier.

11.1 Business Scenario

Byrell's finance department wants to make use of Christine and Sean's enhancement talents to introduce small improvements to the billing process. Their ideas are focusing on improvements for the handling of billing documents. Basically, a quick glance at an invoice document should help experienced users easily categorize it. Here, areas of improvement are numbering ranges and invoice reference fields, which are important for the handling of a billing document.

During a short workshop with key users from the finance department, Sean is given two different tasks, which both focus on improving the handling of the document for its users.

The first task is to change the billing reference, which is a requirement to prefix the delivery type into the invoice Reference field VBRK-XBLNR. This field is found on the Header tab of Transaction VF01/02/03 (Billing Create/Change/Display; see Figure 11.1). Changing the value using an enhancement will enable users to easily identify and categorize a billing document by the beginning of its reference code because it contains the delivery type and number.

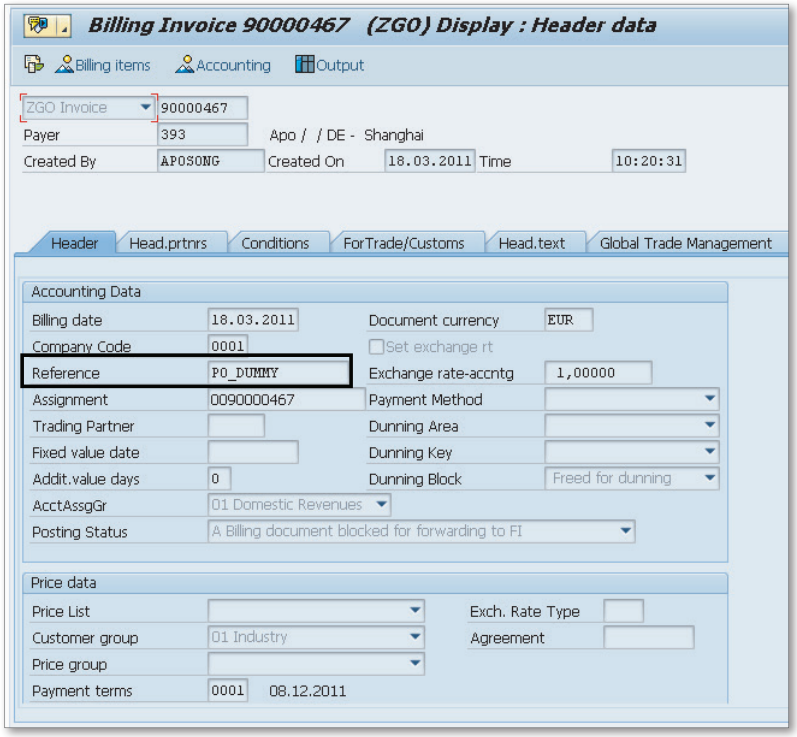


Figure 11.1 Invoice Reference Field XBLNR in the Billing Header

In the second part of this development exercise, the finance department wants to use different numbering ranges, determined by the sales district on the order. Again, this should improve the handling and readability of the document for its users. Especially when going through paperwork (still a reality in Byrell's finance

department), it will help staff separate billing documents easily using different numbering ranges.

The cross-reference table for sales district/numbering range combinations is shown in Table 11.1.

Number Range	Number Range ID	Sales District	Description
60000000- 61999999	Z1	EUROPE	Europe
62000000– 63999999	Z2	CHINA	China
64000000– 65999999	Z3	NTHA	North America
66000000- 67999999	Z4	STHA	South America

Table 11.1 Cross-Reference Table for Numbering Ranges and Sales Districts

After the workshop, Sean takes the following notes.

Workshop Notes

When you're creating a billing document, make sure that you amend the billing reference and numbering range in the following ways:

- **Change the billing reference depending on the delivery type**
  Use the delivery type and delivery number to populate the first 12 characters of the billing Reference field (`VBRK-XBLNR`).
- **Change the numbering range depending on the sales district**
  Use a custom table to indicate which numbering range is to be used for each sales district. A migration of existing invoices is not planned.

Sean has to define these as a prerequisite for the enhancement work because these numbering ranges (second bullet point in the Workshop Notes box) have not been created yet. Fortunately, the rough specification from the finance department also states that migrating existing invoices from the current numbering range to the new ones won't be necessary. This means that all historical invoices that used the old numbering range will be kept on the system as they are. In other words, the proposed new numbering range regime will only be applied to all new invoices from the day Christine and Sean's solution goes live.

Overall, both tasks seem fairly straightforward, but Christine and Sean have some investigating to do to come up with the best enhancement or exit for this scenario.

11.2 Finding the Right Enhancement

Once again, Christine and Sean start their search for exits and enhancements by using Transaction SE80, the Repository Infosystem. Under Program Library • Programs, they use the following search criteria in the selection screen on the right-hand side (see Figure 11.2). Sean knows that some of the billing-related user exits begin with "RV60*". In addition, they enter packages "VF" (R/3 invoice) and "VMOD" (SD customer modifications) as search criteria by clicking the Search Options button.

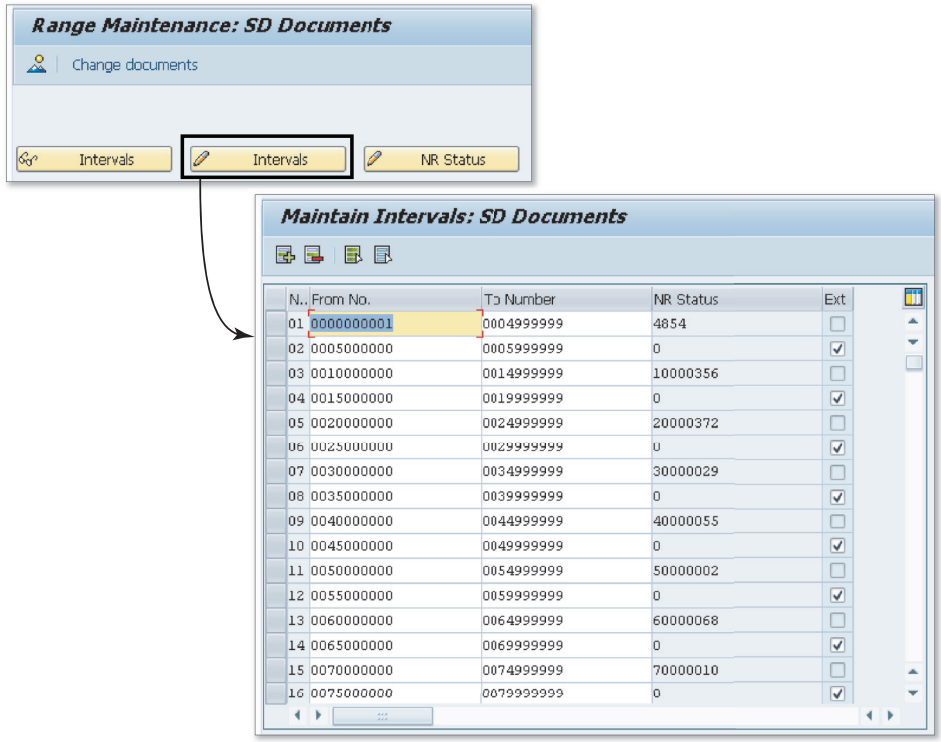


Figure 11.2 Search Criteria to Find a Billing Exit

The result set of their search looks like the screen shown in Figure 11.3.

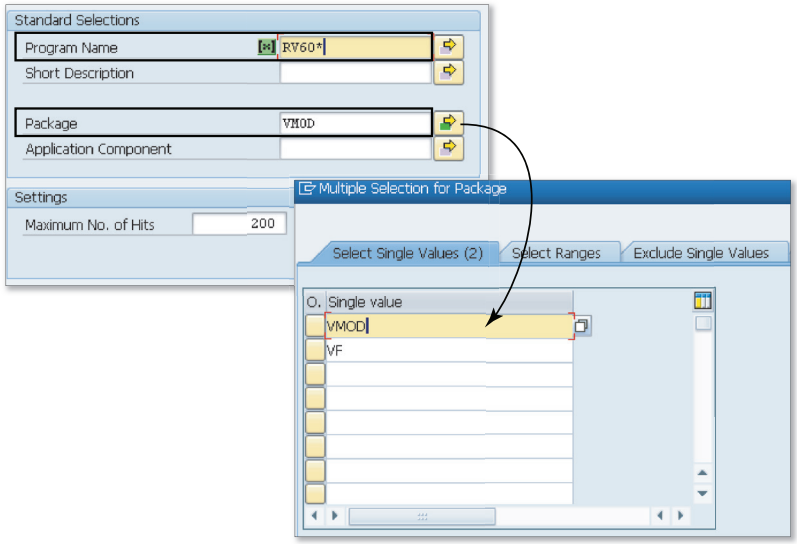


Figure 11.3 Result Set for Billing Exit Search

Upon closer inspection, it appears to Christine and Sean that ABAP `INCLUDE RV60AFZC` seems to fit the bill as far as changing header fields is concerned (see Figure 11.4). Form exit `USEREXIT_FILL_VBRK_VBRP` allows you to amend the contents of invoice field header and item fields, which will help fulfill the finance department's first request, changing the billing reference field `VBRK-XBLNR`.

Program Name	Report title
RV60AFZA	
RV60AFZB	General billing interface: user exit for CPD addresses
RV60AFZC	User Exit for Billing
RV60AFZD	Billing doc. user exits for partner functions in invoice lists
RV60AFZZ	
RV60BFZA	User Exits
RV60C901	
RV60FUS1	User Exit from SAPLV60F, Billing Plan: Distribute Difference
RV60FUS2	User exit from SAPLV60F, billing plan: Prepare pricing
RV60FUS3	User Exit From SAPLV60F, Bill. Plan: Deadline Proposal From Source Doc
RV60FUS4	User exit from SAPLV60F, field preparation field selection for screen
RV60FUS5	
RV60FUST	User Part for Billing Plan - Data Definition
RV60SBAT	Creating Background Jobs for Billing
RV60SBT1	Creating Billing Documents Collectively

Figure 11.4 ABAP INCLUDE RV60AFZC with Form Exit USEREXIT_FILL_VBRK_VBRP

Christine and Sean go back to the result list (refer back to Figure 11.3) of their search for billing user exits and look at another ABAP `INCLUDE RV60AFZZ`. This `INCLUDE`

contains yet another form exit called USEREXIT_NUMBER_RANGE, which—according to the comments in the header of the code—allows the user to determine the number range (see Figure 11.5). This exit will help with altering the number range, which is the second request by the Finance team.

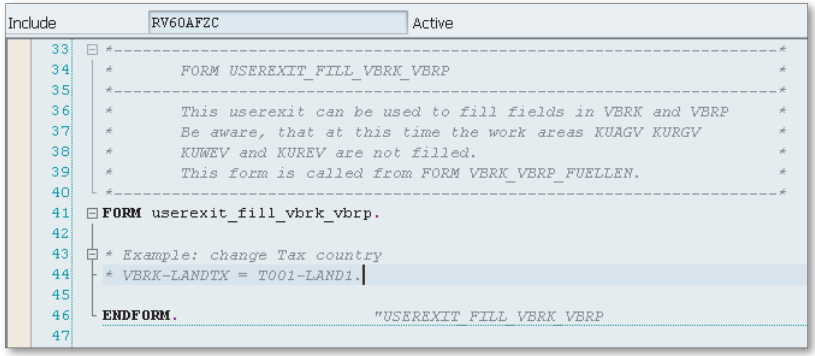


Figure 11.5 ABAP INCLUDE RV60AFZZ with Form Exit USEREXIT_NUMBER_RANGE

Tips & Tricks

Christine and Sean were fortunate enough to find suitable exits right off the bat, but sometimes it can be helpful to test an exit, BAdI, or enhancement before adding all of the coding.

By simply adding a hard-coded breakpoint against your user ID, you can evaluate whether the program in a particular process gives you access to the right data and lets you amend the required fields, structures, or tables.

Figure 11.6 shows you an example of this. Just make sure you assign it to a single user only, and remember to remove it after you've finished your investigation!

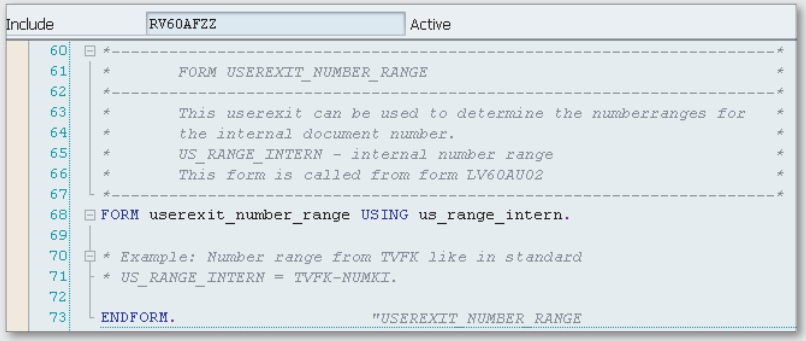


Figure 11.6 Using a Hard BREAK Point to Test Suitability of an Exit

11.3 Implementing the Solution

Christine and Sean have now found the right places for both enhancements. The next step is to plan their work tasks for the realization of the solution. The following steps are necessary to fulfill the finance department's requirements:

1. Define new numbering ranges for SD billing per Table 11.1.
2. Define a custom cross-reference table in the Data Dictionary to store sales district/number range combinations.
3. Implement an implicit enhancement in the ABAP `INCLUDE RV60AFZC`, form `USEREXIT_FILL_VBRK_VBRP`.
4. Add coding to establish the delivery type and alter the reference number in `VBRK-XBLNR`.
5. Implement an implicit enhancement in the ABAP `INCLUDE RV60AFZZ`, form `USEREXIT_NUMBER_RANGE`.
6. Add coding to amend the numbering range according to entries in the cross-reference table (sales district/number range ID).

We'll go over each of these steps in the following sections.

11.3.1 Define New Numbering Ranges

Sean starts by creating the new numbering ranges that have been requested by the finance team. Numbering ranges for SD billing are defined within the IMG under Sales and Distribution • Billing • Billing Documents • Define Number Range for Billing Documents.

On the following screen, Sean can view the existing number ranges (Figure 11.7).

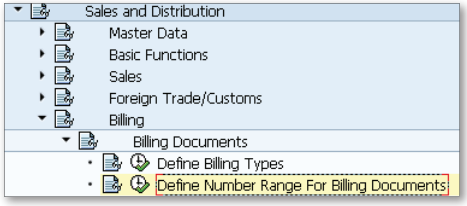


Figure 11.7 Maintenance Dialog for SD Billing Numbering Ranges

Sean clicks on the Intervals button in the Range Maintenance screen. On the following screen, he then enters the new numbering ranges, as listed earlier in Table 11.1 (see Figure 11.8).

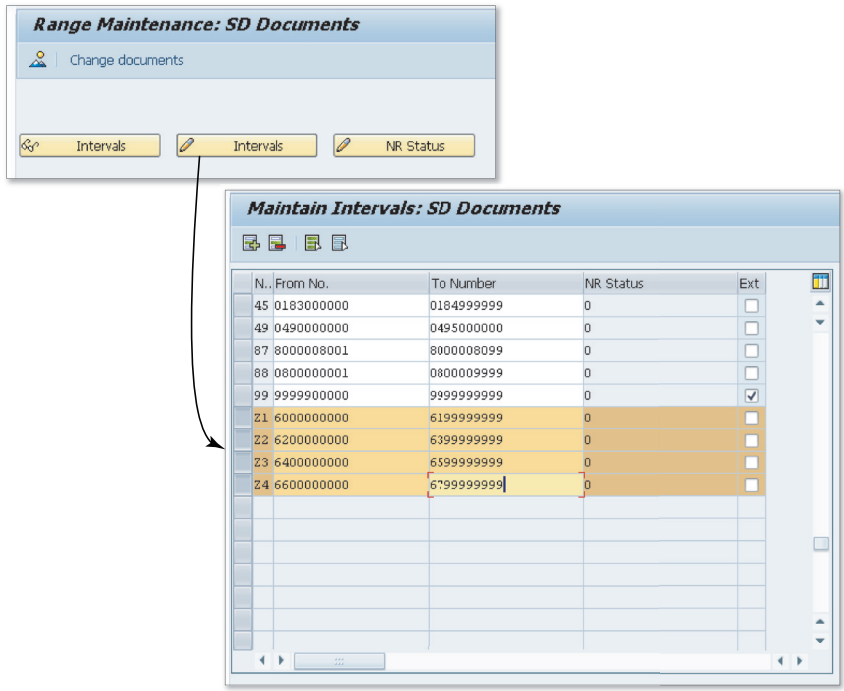


Figure 11.8 Change to Maintenance Mode and Enter New Numbering Ranges

He leaves the External Numbering option unchecked, which means that the issuing of numbers is managed by the system (as opposed to being determined by the user).

After Sean saves the new numbering ranges and assigns these to a customizing change request, he turns his attention to the cross-reference table.

11.3.2 Define the Cross-Reference Table

As requested by the finance department, the billing process should automatically choose different numbering ranges depending on the sales district information that's

stored on the billing header (VBRK-BZIRK). Sales district information is generally copied from the sales order during the creation process.

Sean now has to define a cross-reference table that holds the link between the sales district and numbering range ID. Christine and Sean agreed on the following simple database table shown in Table 11.2.

Field Name	Key	Type	Description
MANDT	X	MANDT	Client
SALES_DISTRICT	X	BZIRK	Sales District
NUMBERING_RANGE	X	NRNR	Numbering Range ID

Table 11.2 Field Layout for Table ZINV_NUM_RANGE

Having learned from Christine how to define tables in the Data Dictionary, Sean creates the development object in Transaction SE11 by adding fields MANDT, NUMBER_RANGE (data element NRNR), and SALES_DISTRICT (data element BZIRK). He then assigns Delivery Class "C" (customer customizing table) to make the table maintainable so new number ranges/sales districts can easily be assigned in the future (see Figure 11.9).

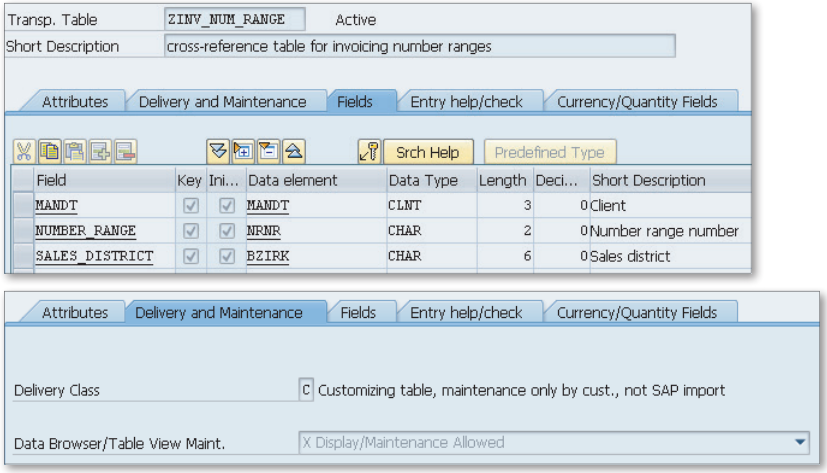


Figure 11.9 Field Layout for Cross-Reference Table ZINV_NUM_RANGE

Sean also remembers to create a maintenance view for this new table, which will allow key users to easily amend table entries (see Figure 11.10). He accesses this function by choosing Extras • Generate Maintenance View form the top menu. He then generates a one-step maintenance type (Overview screen "100"), without any authorization (Authorization Group "&NC&") and enters "ZINV_NUM_RANGE" in the Function group field, which will also be created for him by the system.

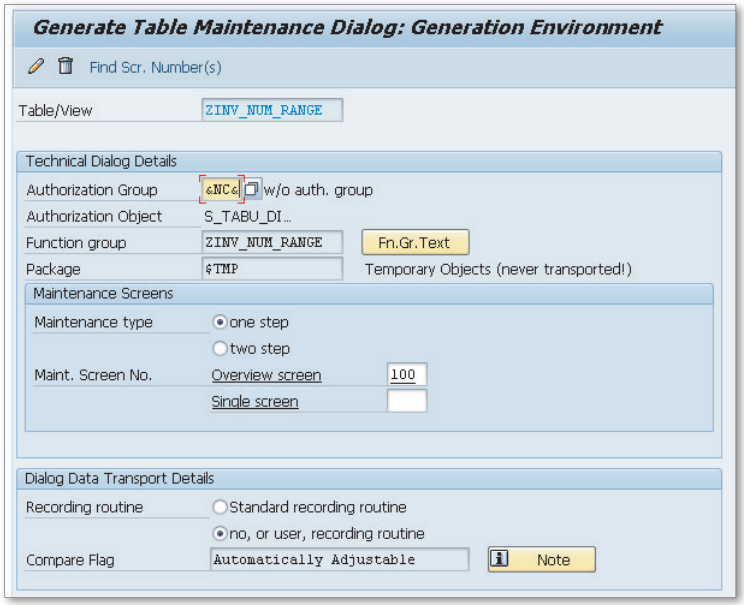


Figure 11.10 Generating Maintenance Screens for Table ZINV_NUM_RANGE

11.3.3 Implement Implicit Enhancements in RV60AFZC

Christine can now implement the first of two implicit enhancements. Firstly, she navigates to ABAP `INCLUDE RV60AFZC` to create an implicit enhancement within user exit form `USEREXIT_FILL_VBRK_VBRP`, which will later hold the coding to establish the delivery type.

Using Transaction SE80, Christine locates ABAP `INCLUDE RV60AFZC`, clicks on the Enhance button, right-clicks into the code, and selects Enhancement Operations • Show Implicit Enhancements Options from the context menu. She then

right-clicks on the quotes line after statement `FORM USEREXIT_FILL_VBRK_VBRP` and chooses Enhancement Operations • Create implementation (see Figure 11.11).

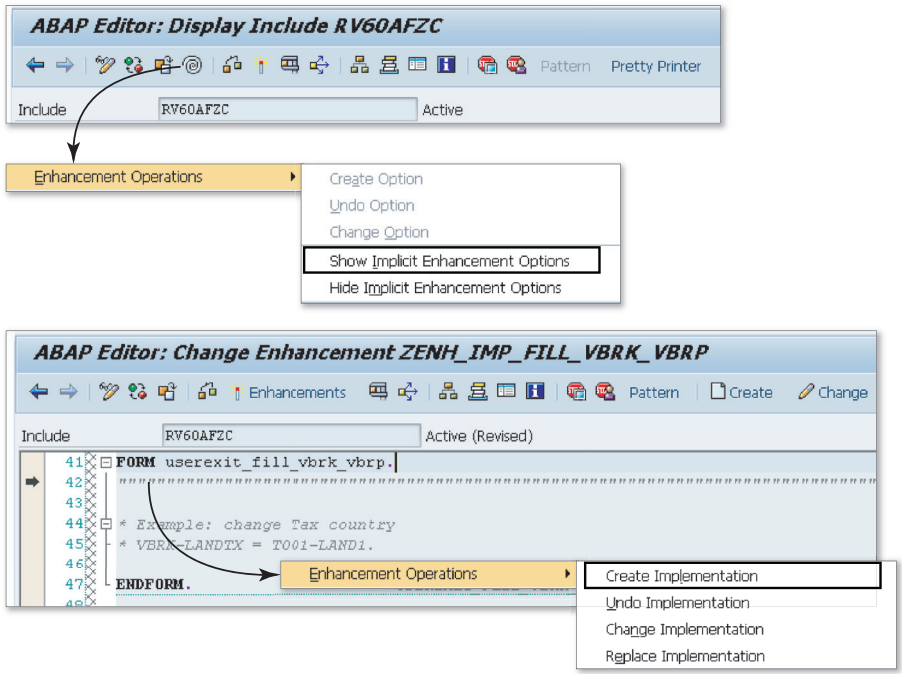


Figure 11.11 Implicit Enhancement for USEREXIT_FILL_VBRK_VBRP

The following popup window asks what kind of implicit enhancement should be created. Christine chooses the Code enhancement, which takes her to another popup screen, where she clicks on the Create button to create a new enhancement implementation.

Finally, she enters the name for her new enhancement implementation ("ZENH_IMP_FILL_VBRK_VBRP") and selects it from the list on the following screen. After confirming her selection, the system enters the ABAP Editor, where the implicit enhancement is now ready for input (see Figure 11.12).

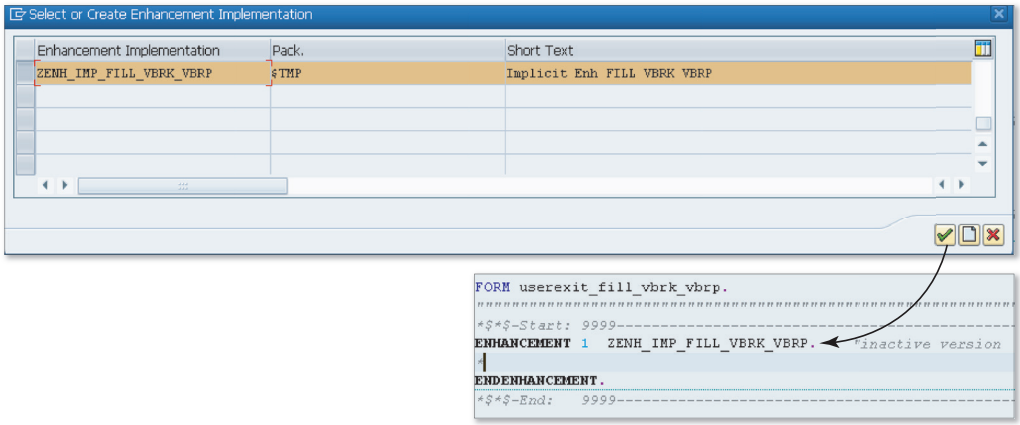


Figure 11.12 Selecting the Enhancement Implementation for Editing

11.3.4 Add Coding to Establish Delivery Type

Christine can now add coding to the implementation, which is shown in Listing 11.1.

```
DATA: ls_likp  TYPE likp,
      lv_delno TYPE vbeln_vl,
      lv_lfart TYPE lfart.

IF NOT vbak-vbeln IS INITIAL.
  SELECT SINGLE vbeln
    INTO lv_delno
    FROM vbfa
    WHERE vbelv   = vbak-vbeln AND
          vbtyp_v = 'C'        AND   " sales order
          vbtyp_n = 'J'.             " delivery

  IF sy-subrc = 0.
    SELECT SINGLE lfart
       FROM likp
       INTO lv_lfart
      WHERE vbeln = lv_delno.

      IF sy-subrc = 0.
         CONCATENATE lv_lfart lv_delno into vbrk-xblnr.
           CONDENSE vbrk-xblnr NO-GAPS.
           ENDIF.

  ELSE.
```

```
    vbrk-xblnr = 'n/a'.
  ENDIF.
ENDIF.
```

Listing 11.1 Coding for USEREXIT_FILL_VBRK_VBRP

The coding first checks whether the sales order document structure contains an order number. If the order number is populated, a `SELECT` statement queries the document flow Table VBFA for the subsequent delivery document number. This is done by searching for a `VBTYP_V` entry of `C` (for the preceding document, which is a sales order) and a `VBTYP_N` entry of `J` (for the subsequent document type of delivery). If an entry is found, then the delivery type and delivery number are concatenated into field `VBRK-XBLNR`. If the `SELECT` statement on Table VBFA is unsuccessful, an `n/a` (not applicable) string is inserted into the `XBLNR` field.

11.3.5 Implement an Implicit Enhancement in RV60AFZZ

Implementing an implicit enhancement into `USEREXIT_NUMBER_RANGE` of ABAP `INCLUDE RV60AFZZ` is done in exactly the same way as explained for `INCLUDE RV60AFZC` in Section 11.3.3; refer to this section for details. This implicit enhancement will hold the coding that will amend the numbering range.

11.3.6 Add Coding to Amend the Numbering Range

The coding for Christine's second user exit should ensure that upon creation of a new billing document, the number range cross-reference table is checked for the current sales district ID. See Listing 11.2 for the ABAP code.

```
DATA: lv_number_range TYPE nrnr.

  IF vbrk-bzirk IS NOT INITIAL.
    SELECT SINGLE number_range
      FROM ZINV_NUM_RANGE
      INTO lv_number_range
     WHERE sales_district = vbrk-bzirk.

    IF sy-subrc = 0.
      us_range_intern = lv_number_range.
    ENDIF.

  ENDIF.
```

Listing 11.2 Coding for USEREXIT_NUMBER_RANGE

Christine's coding first checks whether the sales district field in the current billing header field is populated or not. If it is populated, then the code attempts to fetch a corresponding entry from the custom customization Table ZINV_NUM_RANGE. If the database table holds an entry against this sales district, then field `LV_NUMBER_RANGE` is written to the global user exit field `US_RANGE_INTERN`, which amends the number range assignment.

11.4 Summary

This example showed you two effective ways of amending billing data using implicit enhancements within user exits. Real-world processes can be optimized by giving users more pointers and helpful data within a document. Here, altering the billing document reference number to include delivery type and delivery number can help users when dealing with the invoice to categorize the document with just one glance. The variation in number ranges also adds to the readability of the document, especially for more experienced users.

Once again, using an implicit enhancement rather than an old-style modification is the preferred option. As we pointed out in the introduction of this book, implicit enhancements offer significant advantages over modifications in terms of upgradability, versioning, and switching.

We're now finished with our business example.

The next chapter gives you an overview of the previous chapters and also provides important pointers on how to find the right enhancement technique.

PART IV
Conclusion

There is no one single best technique when it comes to enhancing SAP SD. This summary chapter recaps how we found enhancements in the previous chapters and then provides key criteria to keep in mind when you are creating or changing your own enhancements.

12 Finding the Right Enhancement Technique

In Chapters 2 through 11, we highlighted various ways to customize standard SAP programs. On the whole, these chapters covered the following enhancement techniques:

- User exits (also known as form exits)
- Customer exits
- BAdIs
- Implicit enhancements
- Explicit enhancements
- VOFM routines
- Modifications

Each chapter featured a different way in which Christine and Sean arrived at the best type of enhancement. Sometimes it was necessary to weigh enhancement options; on other occasions, there was only one solution for a particular situation. In order to achieve a more structured approach when looking for the enhancements and techniques, it's worthwhile to first have a look at how Christine and Sean found their solutions.

12.1 Techniques Used and How the Solution Was Found

Table 12.1 provides an overview list of the chapters, the enhancement technique used, and a short summary of how Christine and Sean arrived at their solution.

Chapter	Enhancement Technique Used	Comment
2	User exit (implicit enhancement)	ABAP include (`MV45AFZ*`) for sales order user exits was used.
3	User exit (screen exit, modification)	Screen exit was found by searching the SAP Service Marketplace (SAP Note 420546). A modification was applied, as no other enhancement technique is possible for screen flow changes.
4	Explicit enhancement	Starting in the Repository Infosystem (Transaction SE80), a correct `Enhancement Point` was found by navigating through the standard source code.
5	BAdI	In this SOA example, the BAdI was automatically generated into the provider class.
6	User exit (implicit enhancement)	Usage of the user exit was determined by the task itself.
7	BAdI	The Repository Infosystem was used to find user exits and BAdIs. Usage of both user exit and BAdI was possible. The decision to use the BAdI solution was made due to the higher flexibility of BAdIs.
8	BAdI	The Repository Infosystem was used to find user exits and BAdIs. Usage of both user exit and BAdI was possible. The decision to use the BAdI solution was made due to the higher flexibility of BAdIs.
9	Customer exit	The user exit was found by searching the SAP Service Marketplace (SAP Note 128150).
10	VOFM routine (implicit enhancement)	The Repository Infosystem was used to find user exits. A user exit option was considered at first, but a VOFM was preferred.
11	User exit (implicit enhancement)	The Repository Infosystem was used to find user exits or BAdIs.

Table 12.1 Overview of Enhancement Techniques Used in Each Chapter

12.2 Solution Search Categories

Comparing the enhancement scenarios in Table 12.1 and the method in which Christine and Sean arrived at a suitable solution, you can summarize the individual items into three categories, which highlight three important ways how to find a suitable enhancement:

- **Repository Infosystem search**
  A Repository Infosystem search using a specific development package, program name range, or customer exit ID led to the right enhancement (Chapters 2, 4, 7, 8, 10, and 11).
- **SAP Note search**
  A search for an SAP Note on the SAP Marketplace (*http://service.sap.com*) indicated the enhancement solution (Chapters 3 and 9).
- **Determined by functionality/technology**
  A chosen enhancement solution was determined by the underlying functionality or technology (Chapters 5 and 6).

For Christine and Sean, the Repository Infosystem was by far the best way to find enhancements, which is reflected by the number of mentions in the preceding chapters. Searches on the SAP Marketplace for Notes were featured twice, and in two other chapters, the solution was determined by the functionality (pricing, Chapter 5) or technology (web services, Chapter 6) used.

There are obviously other ways to find SAP enhancement solutions. In addition to the preceding solution search categories, you can also use the following methods:

- **SAP Community Network (*http://scn.sap.com*)**
  With its abundance of blogs, wikis, and forums, SCN has become a cornerstone of SAP community-based help and support.
- **SAP Help (*http://help.sap.com*)**
  SAP's own help documentation can also be a guide when looking for up-to-date information on enhancements or best practice.
- **Online web searches (Google, Bing, etc.)**
  General Internet research also deserves a mention at this point. There are various blogs, forums, and websites offering advice, overviews, and opinions in various areas of SAP, including enhancements.

Now that the key search methods for solutions are established, let's focus on some important key aspects when implementing new enhancements or changing existing ones.

12.3 Key Aspects When Implementing Enhancements

Enhancements do not take place in a vacuum. In most cases, the system you are planning to enhance already has pre-existing exits and modifications. The following subsections highlight some key aspects to keep in mind when applying your changes.

Use an Existing Enhancement Rather Than Adding a New Technique

For example, you may need to use an existing enhancement with a tried and tested user exit that still uses a modification. All that your new requirement aims to do is change and add to the logic of it, but you feel reluctant to use an old technique such as modifications.

In this case (despite reservations), it might still be better to simply extend the program logic of the old user exit, rather than adding the new logic into an additional BAdI, for example, because enhancement program logic would be kept in two different places, thus making it more difficult to support the solution. As a compromise, you could explore an implicit enhancement, which would keep the code in the same place but use a modern enhancement technique.

By the same token, should the timescales during a project allow for a redesign, it might be sensible to review a modification or user exit and move the enhancement into a BAdI or enhancement point, if applicable.

Keeping Future Requirements in Mind

When faced with the choice between two enhancement methods such as a user exit and a BAdI, keep in mind that it is always more prudent to go for a technique that is supported by the Enhancement Framework. The Enhancement Framework provides more flexibility, which also allows for filtering and versioning. This is particularly important when it comes to future requirements. If an assumption can be made that the enhancement is likely to change or be extended in the future, then a more flexible, modern enhancement technique should be used.

This is also the case when the user exit is implemented using implicit enhancements rather than using a modification. BAdIs are generally preferred over implicit enhancements. More details on the differences between user exits with implicit enhancements and BAdIs can be found in Chapter 2.

Modifications as a Means of Enhancing Standard Code

This might come as a slight disappointment, but there are still areas where modifications are necessary. Chapter 3 shows an example for this, where the screen flow could only be amended by a system modification because development objects such as this are not supported by the Enhancement Framework.

As a general rule, due to management features of the Enhancement Framework, implicit enhancements are preferred over system modifications. Due to the limitation of implicit enhancements in terms of positioning, however, sometimes only a modification can be put in the exact location. Implicit enhancements can only be inserted at the beginning of ABAP programs, subroutines, and methods—modifications are not limited by this.

After an upgrade, modifications and implicit/explicit enhancements are managed via Transactions SPAU and SPAU_ENH, which provide important tools to ensure that enhancements continue to work as intended.

At this point, it's important to point out again the relevance of enhancements and modifications when it comes to release upgrades of the SAP ERP application.

Always keep in mind that every single enhancement you add to a system can (and most likely will) create additional work during an upgrade. This is because the monitoring Transactions SPAU and SPAU_ENH will flag those standard programs and routines that were amended by the upgrade *and* contain customer enhancements.

Note

SAP upgrades and how to treat enhancements is a vast topic that is beyond the scope of this book. For more information on upgrades, refer to the following resources:

- SAP Service Marketplace for upgrade guides and release notes (*http://service.sap.com*)
- SAP Help on "Enhancement Information System" (part of the Enhancement Framework) (*http://help.sap.com*)

Documentation, Documentation, Documentation

Proper documentation of enhancements is crucially important. This information is vital when something isn't working as expected or the system is upgraded.

Five Questions to Ask before Creating a New Enhancement

In Chapter 1, we recommended that before creating a new enhancement, you should ask yourself the following questions before you create a new enhancement:

1. Could SAP standard functionality solve your problem?
2. Have you searched SAP Marketplace and checked for SAP Notes that might help you with your problem?
3. Can the same be achieved using configuration?
4. Is there already an existing customization object that the envisaged change could be incorporated in?
5. Have you discussed the problem with functional consultants and/or other team members?

Code enhancements should only be used when all other options have been exhausted. In addition, the last of the five questions is of particular significance because it deals with communication. Enhancement decisions should always be made by a group of people—technical and functional—who can judge and provide different angles to the solution.

12.4 There is No "Silver Bullet"

SAP enhancements are seen by many as an almost impenetrable area, sometimes even compared to "Black Magic" because there can be multiple ways to arrive at a solution. Looking across the SAP ERP portfolio, even more module-specific knowledge comes into play, adding to the complexity.

Nevertheless, this book provides you with some structure, guidelines, help, and inspiration when undertaking enhancements within the SAP SD component.

In the next and final chapter, we'll take a daring glimpse into the future of enhancements and the role they might play.

Developers and functional consultants need to understand the wider context of SAP ERP customizations and enhancements. Here, we'll explain their relevance and ongoing role in the future.

13 Future Outlook

To conclude this book, let's cast the net a little further and examine customizations in the eyes of more than just technical and functional consultants. As a matter of fact, enhancing any ERP software package—not just SAP ERP—is a heavily debated topic in the IT industry.

Generally speaking, "Customization is one of the most controversial topics surrounding ERP software," according to Eric Kimberling of US-based Panorama Consulting. For Kimberling, three main reasons have been identified for this:

1. There is an underlying assumption that customizations add risk and complexity to an ERP implementation, thus making the solution harder to upgrade in the future.
2. Customizations are seen to counteract any Best Practices that have influenced the development of an ERP software suite.
3. Customizations are regarded as a symptom of underlying, bigger problems within the project management of the implementation, for example. It can also be a pointer to a lack of adequate change management.

Critics of ERP customizations correctly point to these three reasons, demanding that consultants and project managers carefully weigh up any alterations to standard systems. More supportive experts point out the importance to reflect on the aspects of competitive advantage.

When computer systems first made their way into businesses, IT helped in rationalizing processes and thereby placed companies in a better market position. Software very often was handwritten and tailored to business needs. During the 1990s, with return on investment (ROI) becoming increasingly important in IT, more and more

standard software was introduced. As a consequence, an assumption was made that a one-size standard software can fit all.

As a consequence, a growing number of businesses wanted to reap the benefits of standard software offerings, such as SAP. Over time, however, these companies missed capabilities to change their systems easily to create or maintain competitive advantage through IT customizations. Among other reasons, this led to the introduction of more sophisticated enhancement techniques, such as the SAP Enhancement Framework, from the early 2000s onwards.

In the case of SAP SD, which forms part of the SAP ERP software suite, this book gives the reader a more detailed insight into how to develop "responsible enhancements": providing competitive advantage, yet remaining non-disruptive.

Responsible Enhancements and Their Importance

In cloud platform ERP systems of the future, the notion of "responsible enhancements" will become even more important, because enhancements can be shared (and potentially cause havoc) with other cloud tenants. Therefore, it will be vitally important to thoroughly consider each and every code enhancement.

Moreover, every implemented enhancement is likely to remain in a system for a very long time and will also drive time and effort when the system is to be upgraded. Therefore, it is important to enhance a system responsibly.

Systems Need the Capability and Agility to Adapt Easily

All of SAP's innovations and strategies are based on a stable core and interfaces, allowing for easy and agile enhancements. This book demonstrated how easy enhancements can be implemented. In the future, this will not change and is likely to become easier.

While SAP guarantees continued support and dedication for the tried and tested SAP ERP core system, the main focus for future innovation is going to be outside the core system. New products will be using this core technology and using channels and services to interact with the core. SAP NetWeaver Gateway is a recent example for this. Gateway enables customers to use open standards to easily bridge the worlds of SAP ERP and mobile technology.

Moreover, new trends such as cloud computing will change how and where ERP systems can be enhanced. For example, the SAP Business ByDesign cloud offering,

aimed at small and medium-sized enterprises (SMEs), doesn't currently allow for non-SAP core system enhancements. However it's possible to create user interface (UI) add-ons and enhancements outside the core system using a separate Software Development Kit (SDK). SAP argues that there are fewer requirements by SMEs to enhance their ERP software, because these customers prefer off-the-shelf solutions with established best practices. While this is true, it's unlikely that this approach will also be applicable to the large-scale cloud ERP systems of the future, especially if used by multiple tenants in a public cloud.

On the whole, customizations of large ERP systems are likely to remain, and even become more important as more and more corporations want to have the benefit of a standard system, while being able to "tweak" the system in certain areas to retain or increase competitive advantage.

Appendices

A Exits and BAdIs in Sales and Distribution

This appendix contains a list of all available exits for Sales and Distribution (SD) in SAP ECC 6.0. The tables cover those areas touched on in this book (sales, deliveries, invoicing) and more (transport documents, shipments).

Tables have been structured and categorized as follows:

- Exits are listed with a description, name main program or enhancement (for user exits also check development class `VMOD`). Function module names (customer exits) with their main programs in brackets.
- BAdIs hold a description, a BAdI definition, and a comment, if applicable, to point to further notes or information about the enhancement. As you can see from the BAdI listings, there are a fair amount of SAP internal BAdIs, which cannot be used by customers or partners.

A.1 Sales Orders

Enhancements for sales documents have been split into the following sections: standard processing, material determination, partner determination, pricing, product selection, text determination, variant configuration, availability check, product allocation, contract data processing, sales support, payment cards, and confirmations. Refer to Chapter 1 for further details on how to find exits and enhancements.

Description	Exit Name	Program/ Enhancement
Delete data in additional tables when a sales document is deleted	`USEREXIT_DELETE_DOCUMENT`	`MV45AFZZ`
Modify the attributes of screen fields. Processed for each field on the screen.	`USEREXIT_FIELD_MODIFICATION`	`MV45AFZZ`
Move some fields into the sales document header work area `VBAK`	`USEREXIT_MOVE_FIELD_TO_VBAK`	`MV45AFZZ`

Table A.1 Standard Processing (Exits)

Description	Exit Name	Program/ Enhancement
Move some fields into the sales document item work area VBAP	USEREXIT_MOVE_FIELD_TO_VBAP	MV45AFZZ
Move some fields into the sales document schedule line work area VBEP	USEREXIT_MOVE_FIELD_TO_VBEP	MV45AFZZ
Move some fields into the sales document business data work area VBKD	USEREXIT_MOVE_FIELD_TO_VBKD	MV45AFZZ
Determine the number ranges for the internal document number	USEREXIT_NUMBER_RANGE	MV45AFZZ
Read data in additional tables when the program reads a sales document	USEREXIT_READ_DOCUMENT	MV45AFZZ
Save data in additional tables when the document is saved	USEREXIT_SAVE_DOCUMENT	MV45AFZZ
Perform changes or checks before a document is saved	USEREXIT_SAVE_DOCUMENT_PREPARE	MV45AFZZ
Check when sales order item is deleted	USEREXIT_CHECK_XVBAP_FOR_DELET	MV45AFZB
Check when schedule line is deleted	USEREXIT_CHECK_XVBEP_FOR_DELET	MV45AFZB
Check the sales order header for completeness and consistency	USEREXIT_CHECK_VBAK	MV45AFZB
Check sales order items for completeness and consistency	USEREXIT_CHECK_VBAP	MV45AFZB
Check sales details for completeness and consistency	USEREXIT_CHECK_VBKD	MV45AFZB
Check schedule lines for completeness and consistency	USEREXIT_CHECK_VBEP	MV45AFZB
Check serial numbers for completeness and consistency	USEREXIT_CHECK_VBSN	MV45AFZB

Table A.1 Standard Processing (Exits) (Cont.)

Description	Exit Name	Program/ Enhancement
Check before deletion of serial number	USEREXIT_CHECK_XVBSN_FOR_DELET	MV45AFZB
Populate additional item data from main item	USEREXIT_FILL_VBAP_FROM_HVBAP	MV45AFZB
Find the source of the plant or item category	USEREXIT_SOURCE_DETERMINATION	MV45AFZB
Preassign sold-to party in sales documents	EXIT_SAPMV45A_002 (V45A0002)	SAPLXVVA
Copy packing proposal into outbound delivery orders	EXIT_SAPMV45A_005 (V45A0004)	SAPLXVVA
Update purchase order from sales order	EXIT_SAPFV45E_001 (V45E0001)	SAPLXVVA
Fill the interface structures for procurement	EXIT_SAPFV45E_002 (V45E0002)	SAPLXVVA
Scheduling agreement processing (customer exits)	EXIT_SAPFV45L_001 (V45L0001) EXIT_SAPFV45L_002 EXIT_SAPFV45L_003	SAPLX45L
Profitability segment in cross-company sales	EXIT_SAPFV45P_001 (V45P0001)	SAPLXVVA
Change sales document using configuration	EXIT_SAPFV45S_002 (V45S0001)	SAPLXVVA
Plan relevance for requirements from incomplete configuration	EXIT_SAPFV45S_003 (V45S0003) EXIT_SAPFV45S_004	SAPLXVVA
Display/change mode for evaluating parameter validity	EXIT_SAPFV45S_005 (V45S0004)	SAPLXVVA
Customer exits for SD text determination	EXIT_SAPLV45T_001 (V45T0001)	SAPLX45T
SD service management: forward contract data to item	EXIT_SAPLV45W_001 (V45W0001)	SAPLX45W

Table A.1 Standard Processing (Exits) (Cont.)

Description	BAdI Definition	Comment
Update planned values of sales orders	BADI_SD_UPDATE_PLVAL	Allows you to store conditions for updating plan values (RMCSS008) to Profitability Analysis and to Financial Accounting
Route determination	BADI_SD_ROUTE	Method ROUTE_DETERMINATION
General enhancements for sales order processing	BADI_SD_SALES	SAP internal in SAP ERP 6.0; see SAP Notes 648427, 1464620, 589746
Enhancements for sales order processing part 2	BADI_SD_SALES_BASIC	SAP internal; see SAP Note 589746
Activation of SAP Credit Management	UKM_R3_ACTIVATE	See SAP Note 1096883
Fill line items for liability update	UKM_FILL	See SAP Note 1466740
BADI for document flow modification	BADI_SD_DOCUMENTFLOW	SAP internal in SAP ERP 6.0; see SAP Notes 498711, 1583650
BAdI for connection—search engine in SD collective search help exits	BADI_SD_COM_SE	Only SAP internal
Enhancements for SD rebate processing	BADI_SD_REBATES	Released
BADI for order confirmation Adobe Print program	BADI_SD_SLS_PRINT01	Released
Tab page for own details—Header Detail Order	BADI_SD_TAB_CUST_H	Only SAP internal
Tab page for own data—Item Detail Order	BADI_SD_TAB_CUST_I	Only SAP internal

Table A.2 Standard Processing (BAdIs)

Description	Exit Name	Program
Additional fields for product substitution (header)	USEREXIT_MOVE_FIELD_TO_KOMKD	MV45AFZA
Additional fields for product substitution (item)	USEREXIT_MOVE_FIELD_TO_KOMPD	MV45AFZA
Additional fields for product listing or exclusion (header)	USEREXIT_MOVE_FIELD_TO_KOMKG	MV45AFZA
Additional fields for product listing or exclusion (item)	USEREXIT_MOVE_FIELD_TO_KOMPG	MV45AFZA

Table A.3 Material Determination

Description	Exit Name	Program/ Enhancement
Reference/duplicate decision making manual address	EXIT_SAPLV09A_001 (V09A0001)	SAPLXV09
EXIT in NO_KNVV for partner type	EXIT_SAPLV09A_002 (V09A0002)	SAPLXV09
User exit partner determination (entry mode XYZ)	EXIT_SAPLV09A_003 (V09A0003)	SAPLXV09
Partner determination (before entering determined partner)	EXIT_SAPLV09A_004 (V09A0004)	SAPLXV09
Populate communication structures for material determination	USEREXIT_MOVE_FIELD_TO_KOMKD USEREXIT_MOVE_FIELD_TO_KOMPD USEREXIT_MOVE_FIELD_TO_KOMKG USEREXIT_MOVE_FIELD_TO_KOMPG	MV50AFZK

Table A.4 Partner Determination

Description	Exit Name	Program
Move additional fields into the pricing communication table (header)	USEREXIT_PRICING_PREPARE_TKOMK	MV45AFZZ
Move additional fields into the pricing communication table (item)	USEREXIT_PRICING_PREPARE_TKOMP	MV45AFZZ
Modify the attributes of condition screen fields	USEREXIT_FIELD_MODIFICATION	SAPMV61A
Modify the subtotal attributes of condition screen fields	USEREXIT_FIELD_MODIFIC_KZWI	SAPMV61A
Modify the header total attributes of condition screen fields	USEREXIT_FIELD_MODIFIC_KOPF	SAPMV61A
Modify the blank lines condition screen fields	USEREXIT_FIELD_MODIFIC_LEER	SAPMV61A
Check pricing maintenance in the document	USEREXIT_PRICING_CHECK	SAPMV61A
Change the pricing rule	USEREXIT_CHANGE_PRICING_RULE	SAPLV61A
Initialize before the loop for the pricing procedure starts	USEREXIT_XKOMV_BEWERTEN_INIT	RV61AFZB
Transfer values into the pricing communication structure during pricing	USEREXIT_XKOMV_BEWERTEN_END	RV61AFZB
Change the dynamic part of the condition record	USEREXIT_XKOMV_ERGAENZEN	RV61AFZB
Change the ready-for-input fields of the manually entered condition record to the condition screen	USEREXIT_XKOMV_ERGAENZEN_MANU	RV61AFZB
Change the work fields of the condition line (with condition)	USEREXIT_XKOMV_FUELLEN	RV61AFZB
Change the work fields of the condition line (without condition)	USEREXIT_XKOMV_FUELLEN_O_KONP	RV61AFZB
Change the KONV fields for copied price components	USEREXIT_PRICING_COPY	RV61AFZB

Table A.5 Pricing (Exits)

Description	BAdI Definition	Method
Access in pricing (replacement)	SD_COND_ACCESS_A	FULL_ACCESS PRESTEP_ACCESS
Change price calculation	OID_SD_SALES_ITEM	IS-OIL/IS_MINE specific (see SAP Note 589746)

Table A.6 Pricing (BAdIs)

Description	Exit Name	Enhancement
Determine alternative materials for product selection	EXIT_SAPFV45S_001 (V45A0001)	SAPLXVVA

Table A.7 Product Selection

Description	Exit Name	Program
Influence text determination for header texts	USEREXIT_MOVE_FIELD_TO_TVCOM_H	MV45AFZB
Influence text determination for item texts	USEREXIT_MOVE_FIELD_TO_TVCOM_I	MV45AFZB

Table A.8 Text Determination

Description	Exit Name	Program
Option to include new fields for the variant configuration	USEREXIT_GET_FIELD_FROM_SDCOM	MV45AFZB
Format additional work areas for the variant configuration	USEREXIT_MOVE_WORKAREA_TO_SDWA	MV45AFZB

Table A.9 Variant Configuration

Description	Exit Name	Program/ Enhancement
Fill additional or different data into the header table of the availability check	USEREXIT_ADD_FIELD_ TO_HEADER	FV45VFZZ
Fill additional or different data into the schedule line table of the availability check	USEREXIT_ADD_FIELD_ TO_LINE	FV45VFZZ_ USEREXIT_ADD_ FIELD_TO
Custom logic for delivery group correlation	USEREXIT_DELIVERY_ GROUPS	FV45VFZY
Refresh of global data at end of transaction	USEREXIT_MVERF_INIT	FV45VFZY
Influence the availability check (input)	USEREXIT_ AVAILABILITY_IN	RV03VFZZ
Influence the availability check (output)	USEREXIT_ AVAILABILITY_OUT	RV03VFZZ
Initialization of global data	USEREXIT_DATA_REFRESH	RV03VFZZ
Plant selection for availability check	USEREXIT_PLANT_ SELECTION	RV03VFZZ
Reschedule schedule lines without a new ATP check	EXIT_SAPVSTRM_NO_ ATPCHK_001 (SDTRM001)	SAPLXVT1

Table A.10 Availability Check

Description	Exit Name	Program
Fill additional data into table account assignment COBL	USEREXIT_MOVE_FIELD_ TO_COBL	MV45AFZB
Move fields from Table COBL to Table VBAK	USEREXIT_COBL_ RECEIVE_VBAK	MV45AFZB
Move fields from Table COBL to Table VBAP	USEREXIT_COBL_ RECEIVE_VBAP	MV45AFZB
Move data to the communication table	USEREXIT_COBL_SEND_ ITEM	MV45AFZB

Table A.11 Product Allocation

Description	Exit Name	Program/Enhancement
SD service management: forward contract data to item	EXIT_SAPLV45W_001	V45W0001

Table A.12 Contract Data Processing

Description	Exit Name	Program/ Enhancement
SIS: Statistics update: sales activities/sales promotions	EXIT_SAPLMCV2_001	MCV20001
User exit f; supplying structure SADLSTLIS with append structure	EXIT_SAPLV43M_007	V43MLIS
Enhancements for linking to calendar	EXIT_SAPLV43K_001	V43K0001
Change sales activity data online (structure VBKAKOM_UPDATE)	EXIT_SAPMV43A_007	V43ADATA

Table A.13 Sales Support

Description	Exit Name	Program
Create several authorizations in the sales order at the same time	AUTHORIZATION_VALUE_SPLIT	MV45AFZH
Distribute open quantity onto selected batches	USEREXIT_BATCH_QUAN_ALLOCATION	MV50AFZ2
Control confirmation requirements on the item level	USEREXIT_LIPS-KOQUI_DETERMINE	MV50AFZ3
Restrict the delivery creation to some order items	KZKOR_DETERMINE	MV50AFZL
Determine delivery item status	USEREXIT_SET_STATUS_VBUP	LV50PFZA

Table A.14 Payment Cards, Confirmations, Sales Documents

Description	Exit Name	Program
Check of field VBLB-USR01	USEREXIT_CHECK_VBLB-USR01	MV45AFZC
Check of field VBLB-USR02	USEREXIT_CHECK_VBLB-USR02	MV45AFZC
Check of field VBLB-USR03	USEREXIT_CHECK_VBLB-USR03	MV45AFZC
Check of field VBLB-USR04	USEREXIT_CHECK_VBLB-USR04	MV45AFZC
Check of field VBLB-USR05	USEREXIT_CHECK_VBLB-USR05	MV45AFZC

Table A.15 Component Supply Processing

A.2 Deliveries

Most of delivery-related user exits are located within ABAP MV50AFZ1. BAdI LE_SHP_DELIVERY_PROC is the modern equivalent of this (as shown in Part 2 of this book). Please also note enhancements listed for IDoc processing, wave picking, delivery monitor, and outbound delivery orders.

Description	Exit Name	Program
Delete additional data in additional tables before a deleted delivery is saved	USEREXIT_DELETE_DOCUMENT	MV50AFZ1
Additional data in the delivery header	USEREXIT_MOVE_FIELD_TO_LIKP	MV50AFZ1
Additional data in the delivery item	USEREXIT_MOVE_FIELD_TO_LIPS	MV50AFZ1
Determine the number ranges for the internal document number	USEREXIT_NUMBER_RANGE	MV50AFZ1

Table A.16 Delivery Processing (User Exits)

Description	Exit Name	Program
Read data in additional tables when the delivery document is fetched	USEREXIT_READ_DOCUMENT	MV50AFZ1
Save additional data when the delivery document is saved	USEREXIT_SAVE_DOCUMENT	MV50AFZ1
Perform changes or checks before a delivery document is saved	USEREXIT_SAVE_DOCUMENT_PREPARE	MV50AFZ1
Refresh user-specific data when the processing of a delivery is finished	USEREXIT_REFRESH_DOCUMENT	MV50AFZ1
Distribute batch quantities within a delivery	USEREXIT_BATCH_QUAN_ALLOCATION	MV50AFZ2
Perform batch determination	USEREXIT_BATCH_DETERMINATION USEREXIT_LGORT_DETERMINATION	MV50AFZZ
Exit 2 for batch determination	USEREXIT_LIPS-KOQUI_DETERMINE	MV50AFZ3

Table A.16 Delivery Processing (User Exits) (Cont.)

Description	Function Module	Enhancement
User exit: foreign trade data proposal in MM and SD	▶ EXIT_SAPLV50E_001 ▶ EXIT_SAPLV50E_002 ▶ EXIT_SAPLV50E_003	V50EPROP
Delivery: Item status calculation	EXIT_SAPLV50P_001	V50PSTAT
Delivery monitor: user exits for filling display fields	▶ EXIT_SAPLV50Q_001 ▶ EXIT_SAPLV50Q_002	V50Q0001
Collective processing for delivery creation	EXIT_SAPLV50R_001	V50R0001
Collective processing for delivery creation	EXIT_SAPLV50R_VIEW_001	V50R0002

Table A.17 Delivery Processing (Exits)

Description	Function Module	Enhancement
Extension to delivery processing BAPI	▶ EXIT_SAPLV50R_CREA_001 ▶ EXIT_SAPLV50R_CREA_002 ▶ EXIT_SAPLV50R_CREA_003	V50R0003
User exits for delivery processing	▶ EXIT_SAPLV50S_001 ▶ EXIT_SAPLV50S_002	V50S0001
User exit for checking archivability of handling units	EXIT_SAPLV51R_001	V51R0001
Rough workload calculation in time per item	EXIT_SAPLV53C_001	V53C0001
W&S: RWE enhancement—shipping material type/time slot	EXIT_SAPLV53C_002	V53C0002
User exits for creating picking waves	EXIT_SAPMV53W_001	V53W0001
User exit for sorting deliveries for wave picks	EXIT_SAPMV53W_002	V53W0001
User exit: changing and adding to wave picks and deliveries	EXIT_SAPMV53W_003	V53W0001
Move some fields into the communication work area for product substitution	USEREXIT_MOVE_FIELD_TO_KOMKD	MV50AFZP

Table A.17 Delivery Processing (Exits) (Cont.)

Description	Method	Definition
Control activation of function codes	CHANGE_FCODE_ATTRIBUTES	LE_SHP_DELIVERY_PROC
Control input attributes of delivery fields	CHANGE_FIELD_ATTRIBUTES	LE_SHP_DELIVERY_PROC

Table A.18 Delivery Processing (BAdIs)

Description	Method	Definition
Read own data after delivery has been read	READ_DELIVERY	LE_SHP_DELIVERY_PROC
Initialize own data	INITIALIZE_DELIVERY	LE_SHP_DELIVERY_PROC
Check deletion of a delivery item	CHECK_ITEM_DELETION	LE_SHP_DELIVERY_PROC
Delete own item-dependent data	ITEM_DELETION	LE_SHP_DELIVERY_PROC
Delete own data when deleting the delivery	DELIVERY_DELETION	LE_SHP_DELIVERY_PROC
Make changes in header structure (TKOMK) and item structure (TKOMP)	CHANGE_INPUT_HEADER_AND_ITEMS	LE_SHP_PRICING
Process consignment: transfer of additional fields from general delivery interface	▶ MOVE_KOMDLGN_TO_LIKP ▶ MOVE_KOMDLGN_TO_LIPS	LE_SHP_GN_DLV_CREATE
Change item status of delivery items	CHANGE_ITEM_STATUS	LE_SHP_ITEM_STATUS
Perform subsequent delivery split	EXECUTE	LEDSP_SPLIT_EXECUTE
Make changes in header (IMKPF) and items (XIMSEG)	CHANGE_INPUT_HEADER_AND_ITEMS	LE_SHP_GOODSMOVEMENT
Activate the additional tab pages	ACTIVATE_TAB_PAGE	LE_SHP_TAB_CUST_HEAD
Transfer data: delivery data to subscreen	TRANSFER_DATA_TO_SUBSCREEN	LE_SHP_TAB_CUST_HEAD
Transfer data: data from subscreen back to delivery	TRANSFER_DATA_FROM_SUBSCREEN	LE_SHP_TAB_CUST_HEAD

Table A.18 Delivery Processing (BAdIs) (Cont.)

Description	Method	Definition
Transfer current function code to subscreen	PASS_FCODE_TO_SUBSCREEN	LE_SHP_TAB_CUST_HEAD
Activate the additional tab pages	ACTIVATE_TAB_PAGE	LE_SHP_TAB_CUST_ITEM
Transfer data: delivery data to subscreen	TRANSFER_DATA_TO_SUBSCREEN	LE_SHP_TAB_CUST_ITEM
Transfer data: data from subscreen back to delivery	TRANSFER_DATA_FROM_SUBSCREEN	LE_SHP_TAB_CUST_ITEM
Transfer current function code to subscreen	PASS_FCODE_TO_SUBSCREEN	LE_SHP_TAB_CUST_ITEM
Activate the additional tab pages	ACTIVATE_TAB_PAGE	LE_SHP_TAB_CUST_OVER
Transfer data: delivery data to subscreen	TRANSFER_DATA_TO_SUBSCREEN	LE_SHP_TAB_CUST_OVER
Transfer data: data from subscreen back to delivery	TRANSFER_DATA_FROM_SUBSCREEN	LE_SHP_TAB_CUST_OVER
Transfer current function code to subscreen	PASS_FCODE_TO_SUBSCREEN	LE_SHP_TAB_CUST_OVER
Manipulate IDoc in release receipt	IDOC_DATA_MODIFY	BADI_SD_EDI_DELSCHED

Table A.18 Delivery Processing (BAdIs) (Cont.)

Description	Function Module	Exit
Delivery (inbox): copy data	EXIT_SAPLV55K_001	V55K0001
Delivery (inbox): prepare processing	EXIT_SAPLV55K_002	V55K0002
Delivery (inbox): evaluate results	EXIT_SAPLV55K_003	V55K0003
Shipping notification (inbound): modification of IDoc control data	EXIT_SAPLV55K_004	V55K0004

Table A.19 Delivery (IDoc Exits)

Description	Function Module	Exit
Purchase order (inbound): modification of IDoc control data	EXIT_SAPLV55K_005	V55K0005
Shipping notification (inbox): copy data	EXIT_SAPLV55K_011	V55K0011
Shipping notification (Inbox): prepare processing	EXIT_SAPLV55K_012	V55K0012
Shipping notification (Inbox): evaluate results	EXIT_SAPLV55K_013	V55K0013

Table A.19 Delivery (IDoc Exits) (Cont.)

Description	Function Module	Exit
User exit for gate + material staging area determination (header)	EXIT_SAPLV02V_003	V02V0003
User exit for staging area determination (item)	EXIT_SAPLV02V_004	V02V0004
Delivery monitor: user exits for filling display fields	EXIT_SAPLV50Q_001	V50Q0001
User exit for output selection	EXIT_SAPLV50Q_002	V50Q0001

Table A.20 Delivery Monitor (Exits)

Description	Function Module	Program/ Enhancement
Control how the system copies packing proposals into the outbound delivery order	EXIT_SAPMV45A_005	SAPLXVVA
Restrict delivery creation to some order items from a delivery group	KZKOR_DETERMINE	MV50AFZL

Table A.21 Outbound Delivery Orders and Groups (Exits)

Description	Exit Name	Program/ Enhancement
User exits for creating picking waves	EXIT_SAPMV53W_001	V53W0001
User exit for sorting deliveries for wave picks	EXIT_SAPMV53W_002	V53W0001
User exit for changing and adding to wave picks and deliveries	EXIT_SAPMV53W_003	V53W0001
SD customer functions for resource-related billing	EXIT_SAPLV46H_002	V46H0001

Table A.22 Picking Waves (Exits)

A.3 Invoicing

For invoicing, the following tables have been created: billing interface, accounting, and billing plan. At the end of this section, you can also find a table with invoicing-related BAdIs.

Description	Exit Name	Program/ Enhancement
Change of the internal numbering range	USEREXIT_NUMBER_RANGE	RV60AFZZ
Routine for supplying new header field in billing	USEREXIT_ACCOUNT_ PREP_KOMKCV	RV60AFZZ
Routine for supplying new item field in billing	USEREXIT_ACCOUNT_ PREP_KOMPCV	RV60AFZZ
Change of billing date	USEREXIT_NUMBER_ RANGE_INV_DATE	RV60AFZZ
Providing the header and the item of the new billing document with deviating or additional data	USEREXIT_FILL_VBRK_ VBRP	RV60AFZZ

Table A.23 Billing Interface (Exits)

Description	Exit Name	Program/ Enhancement
Printing the item line of a billing document; can be supplemented or changed	USEREXIT_PRINT_ITEM	RV60AFZZ
Printing the header line of a billing document; can be supplemented or changed	USEREXIT_PRINT_HEAD	RV60AFZZ
Partner change when creating line item	▶ EXIT_SAPLV46H_001 (V46H0001) ▶ EXIT_SAPLV46H_002	SAPLXV46

Table A.23 Billing Interface (Exits) (Cont.)

Description	Exit Name	Program
Deviate address data for different partner functions	USEREXIT_AVBPAK_CPD	RV60AFZB
Copy new partner functions from the interface	USEREXIT_AVBPAK_ADD	RV60AFZA
Add a key field	USEREXIT_XVBAPF_KEY	RV60AFZA
Add a key field in the aggregated flow	USEREXIT_XVBAPF_KEY_CANC	RV60AFZA

Table A.24 Billing Interface (Exits)

Description	Exit Name	Enhancement
Header line in delivery to accounting	EXIT_SAPLV60B_001	SDVFX001
AR line in transfer to accounting	EXIT_SAPLV60B_002	SDVFX002
Cash clearing in transfer to accounting	EXIT_SAPLV60B_003	SDVFX003
G/L line in transfer to accounting	EXIT_SAPLV60B_004	SDVFX004

Table A.25 Accounting (Exits)

Description	Exit Name	Enhancement
Reserves in transfer to accounting	EXIT_SAPLV60B_005	SDVFX005
Tax line in transfer to accounting	EXIT_SAPLV60B_006	SDVFX006
Billing plan during transfer to accounting	EXIT_SAPLV60B_007	SDVFX007
Processing of transfer structures SD-FI	EXIT_SAPLV60B_008	SDVFX008
Item table for the customer lines	EXIT_SAPLV60B_010	SDVFX010
Communication structures for reconciliation account	EXIT_SAPLV60B_011	SDVFX011

Table A.25 Accounting (Exits) (Cont.)

Description	Exit Name	Program/ Enhancement
Billing Plan: Distribute Difference in Partial Billing	BILLING_SCHEDULE_DELTA	RV60FUS1
Populate fields in the billing plan (FPLT)	USEREXIT_MOVE_FIELD_TO_FPLT	RV60FUS1
Populate custom fields in the billing plan (FPLA)	USEREXIT_MOVE_FIELD_TO_FPLA	RV60FUS1
Move additional fields into the communication table TKOMX	USEREXIT_PRICING_PREPARE_TKOMX	RV60FUS2
Change the dates that have been copied from a date proposal	USEREXIT_DATE_PROPOSAL	RV60FUS3
SD billing plan differences to billing plan	EXIT_SAPLV60F_001	V60F0001
Customer functions for resource-related billing (creating item)	EXIT_SAPLV46H_001	V46H0001
Customer functions for resource-related billing (partner changes)	EXIT_SAPLV46H_002	V46H0001

Table A.26 Billing Plan (Exits)

Description	Exit Name	Enhancement
EDI supplier processing: self-billing (condition value tolerances in the self-billing)	EXIT_SAPLVED5_001	VED50001
EDI supplier processing: self-billing (messages credit memo procedures SBINV)	EXIT_SAPLVED5_002	VED50001
EDI supplier processing: self-billing (tolerance check credit memo procedure SBINV)	EXIT_SAPLVED5_003	VED50001
EDI supplier processing: self-billing (changing billing document data SBINV)	EXIT_SAPLVED5_004	VED50001
EDI supplier processing: self-billing (changes in workflow parameters)	EXIT_SAPLVED5_005	VED50001
EDI supplier processing: self-billing (copying data to screens for incoming EDI docs)	EXIT_SAPLVED5_006	VED50001

Table A.27 Accounting (Exits)

Description	BAdI Definition	Method
BAdI for transfer to Financial Accounting	BADI_SD_ACCOUNTING	▶ ACCOUNTING_INTERFACE ▶ ACCOUNTING_ITEM_LINE ▶ ACCOUNTING_HEAD_LINE
BAdI for billing document Adobe Printing program	BADI_SD_BIL_PRINT01	▶ INITIALIZE DATA ▶ GET_ITEM_DETAILS
Billing enhancements	BADI_SD_BILLING	Only SAP internal
Billing enhancements at item level	BADI_SD_BILLING_ITEM	Only SAP internal

Table A.28 Accounting (BAdIs)

Description	BAdI Definition	Method
BAdI for SAP Credit Management	BADI_SD_CM	▶ FSCM_GET_ACCOUNT_KEY ▶ FSCM_COMMITMENT_UPDATE_INVOICE
BAdI to populate country-specific fields in KOMP and KOMK	BADI_SD_COM_COUNTR	Only SAP internal
Redetermination of date values in billing plan	BADI_SD_DATE_UPDATE	DATE_TO_DETERMINATION
Grants management: set billing status	BADI_SD_GM	Released
Performance tuning for pricing in billing plan	BADI_SD_PRICING_TUN	PRICING_TUNING
SD customer functions for resource-related billing	BADI_SD_V46H0001	▶ EXIT_SAPLV46H_001 ▶ EXIT_SAPLV46H_002 ▶ EXIT_SAPLV46H_003
BAdI for revenue report	BADI_SD_VF48	Only SAP internal

Table A.28 Accounting (BAdIs) (Cont.)

A.4 Transport Documents

Transport documents have been separated into a table for exits and another table for BAdIs. Note the relatively high number of BAdIs that can be used.

Description	Exit Name	Enhancement
Status of shipments for a delivery	EXIT_SAPLV56L_001	V56L0001
Deactivates multiple transmission lock for delivery to TPS	EXIT_SAPLV56L_007	V56L0007
Shipment processing: determine location identification	EXIT_SAPLSTAG_002	V56LOCID

Table A.29 Transport Document (Exits)

Description	Exit Name	Enhancement
Collective processing of shipments (enhancement of field catalog)	EXIT_SAPLV56M_001	V56MVT04
Collective processing of shipments (assign deliveries to shipments)	EXIT_SAPLV56M_002	V56MVT04
Collective processing of shipments (filling generated shipment docs with data)	EXIT_SAPLV56M_003	V56MVT04
Shipment processing: leg determination	EXIT_SAPLV56S_001	V56SLDET
Filters delivery items for shipment	EXIT_SAPLV56T_001	V56TDLIF
Shipment processing: check whether changes were made	EXIT_SAPLV56U_001	V56UCHCH
Checks shipments are complete	EXIT_SAPLV56U_002	V56UCHCO
Delivery update on delivery routines (obsolete as of 4.6C)	EXIT_SAPLV56U_006	V56UDLUP
Shipment number allocation	EXIT_SAPLV56U_003	V56UNUMB
User-individual definition of transportation planning status	EXIT_SAPLV56U_007	V56USTAT
Updates new objects for transport	EXIT_SAPLV56U_005	V56USVDO
Preparation for updating new objects for transport	EXIT_SAPLV56U_004	V56USVDP
Shipment processing: check function code allowed	EXIT_SAPMV56A_001	V56AFCCH
Filtering shipping unit calculation	EXIT_SAPLV56A_003	V56AGTAR
Checks for archiving shipments (prefetch)	EXIT_SDVTTKWR_001	V56ARCHV
Checks for archiving shipments (checks per shipment)	EXIT_SDVTTKWR_002	V56ARCHV
Changes the number of lines for text input in shipment	EXIT_SAPMV56A_002	V56ATKTX

Table A.29 Transport Document (Exits) (Cont.)

Description	Exit Name	Enhancement
Transportation processing: field modification	EXIT_SAPLV56B_001	V56BMOD
Shipment processing: determine distance	EXIT_SAPLSTAG_001	V56DISTZ
Shipment processing: copy delivery data	EXIT_SAPLV56F_010	V56FCOPY
Shipment processing: activities for setting a status, time 1	EXIT_SAPLV56F_011	V56FSTAT
Shipment processing: activities for setting a status, time 2	EXIT_SAPLV56F_012	V56FSTATi
IDoc TPSDLS: modification of delivery header group	EXIT_SAPLV56I_001	V56I0001
IDoc TPSDLS: Modification of delivery item group	EXIT_SAPLV56I_002	V56I0002
IDoc TPSDLS: Modification of package data group	EXIT_SAPLV56I_003	V56I0003
IDoc TPSDLS: Modification of entire IDoc	EXIT_SAPLV56I_004	V56I0004
IDoc TPSDLS: Modification of delivery items relevant to shipment	EXIT_SAPLV56I_005	V56I0005
IDOC TPSDLS: User-defined determination for location substitution	EXIT_SAPLV56I_006	V56I0006
IDOC TPSSHT01: Modification of IDOC transportation segments	EXIT_SAPLV56I_010	V56I0010
IDOC TPSSHT01: Modification of transportation tables	EXIT_SAPLV56I_011	V56I0010
IDOC TPSSHT01: Save new tables	EXIT_SAPLV56I_012	V56I0010
IDoc control record modification in interface SD-TPS	EXIT_SAPLV56I_020	V56I0020
Freight costs RFC from APO	EXIT_SAPLLE_TRA_X_001	V56I0030

Table A.29 Transport Document (Exits) (Cont.)

Description	Exit Name	Enhancement
User exit for the structure of the planning overview	EXIT_SAPFV56I_001	V56IVIEW
Create outbound transportation output (EDI)	EXIT_SAPLV56K_001	V56K0001
Enhancement for calling shipment BAPIs	EXIT_SAPLV56K_BAPI_001	V56KBAPI

Table A.29 Transport Document (Exits) (Cont.)

Description	BAdI Definition	Method
Processes shipments during "At Save" context	BADI_LE_SHIPMENT	AT_SAVE
Processes shipments during "Before Update" context	BADI_LE_SHIPMENT	BEFORE_UPDATE
Processes shipments during "In Update" context	BADI_LE_SHIPMENT	IN_UPDATE
BADI for general checks and actions in handling unit management	BADI_HU_MAIN	multiple
Checks whether packing an item into a handling unit is allowed	BADI_HU_PACKING_ALWD	BADI_HU_PACKING_ALWD
Layer value management for BADIs	BADI_LAYER	multiple
Automatic packing (preparation)	BADI_HU_AUTOPACK	METHOD_PREPARE
Automatic packing (proposals)	BADI_HU_AUTOPACK	METHOD_PROPOSAL
Reduction of the quantity to be packed	BADI_HU_PACKING_QTY	BADI_HU_PACKING_QTY
BAdI for saving handling units	BADI_HU_SAVE	SAVE

Table A.30 Transport Documents and Handling Units (BAdIs)

A.5 Shipment Costs

Shipment costs have also been split into tables for exits and BAdIs. Most exits are outdated, and BAdIs should be used instead.

Description	Function Module	Enhancement
Shipment cost calculation: prepare price determination (KOMK)	EXIT_SAPLV54B_001	V54B0001
Shipment cost calculation: prepare price determination (KOMP)	EXIT_SAPLV54B_002	V54B0001
Shipment costs calculation: determine rate type and currency	EXIT_SAPLV54B_003	V54B0003
Shipment cost calculation: determine status	EXIT_SAPLV54B_004	V54B0004
Shipment costing: description(s) shipment cost item(s)	EXIT_SAPLV54C_001	V54C0001
Shipment costing: create shipment cost sub-items	EXIT_SAPLV54C_002	V54C0002
Shipment costs processing: determine invoicing party	EXIT_SAPLV54C_003	V54C0003
Shipment costs processing: determine loc. for tax invoice	EXIT_SAPLV54C_004	V54C0004
Shipment costing: determining the tax countries	EXIT_SAPLV54D_001	V54D0001
Determining the factors for apportionment of shipment costs	EXIT_SAPLV54K_001	V54KSFRC
Extended function codes for shipment cost information	EXIT_SAPLV54P_001	V54P0001
Extended function codes for shipment cost information	EXIT_SAPLV54P_002	V54P0001
Extended function codes for shipment cost information	EXIT_SAPLV54U_001	V54U0001

Table A.31 Shipment Costs (Exits)

Description	Function Module	Enhancement
Checks shipment costs for completion	EXIT_SAPLV54U_002	V54U0002
Specification of shipment cost number	EXIT_SAPLV54U_003	V54U0003
Formatting for update of new objects (shipment costs)	EXIT_SAPLV54U_004	V54U0004
Updating new objects in shipment cost processing	EXIT_SAPLV54U_005	V54U0005
Shipment purchase order—header data supply	EXIT_RV54POCR_006	V54U0006
Shipment purchase order—item data supply	EXIT_RV54POCR_007	V54U0007

Table A.31 Shipment Costs (Exits) (Cont.)

Description	BAdI Definition	Method
Specify the shipment cost type	BADI_SCD_CREATE	SET_SHIPMENT_COST_TYPE
Set header fields in a shipment cost document	BADI_SCD_CREATE	SET_HEADER_DATA
Perform account assignment determination before check	BADI_SCD_ACCTG	BEFORE_CHECK
Check whether the shipment cost document should be created	BADI_SCD_CREATE_CHCK	CHECK_DOCUMENT_CREATE
Check whether shipment cost item can be deleted	BADI_SCD_PROCESS_CHK	ITEM_DELETE_CHECK
Fill reference fields in service entry	BADI_SCD_TRANSFER	SET_REFERENCE_FIELDS
Check shipment costs document for completeness before saving	BADI_SCD_SAVE	CHECK_COMPLETE

Table A.32 Shipment Costs (BAdIs)

Description	BAdI Definition	Method
Prepare new objects when saving shipment cost documents	BADI_SCD_SAVE	AT_SAVE
Save new objects	BADI_SCD_SAVE	BEFORE_UPDATE
Determine determination rule	BADI_SCD_PO_SELECT	DETERMINATION_CHECK
Determine purchase order item(s) for standard check	BADI_SCD_PO_SELECT	PO_SELECT

Table A.32 Shipment Costs (BAdIs) (Cont.)

B Code Listing for Class ZCL_SURVEY

In this appendix, we've included the code listing for the constructor method of class ZCL_SURVEY (see Chapter 3).

```
METHOD constructor.

  DATA: lo_agent_survey TYPE REF TO zca_survey_pers,
        lo_survey TYPE REF TO zcl_survey_pers,
        lo_query_manager_survey_t TYPE REF TO if_os_query_manager,
        lo_query_survey_t     TYPE REF TO cl_os_query,
        lo_filter_survey_t    TYPE REF TO if_os_query_filter_expr,
        lt_survey_t_ref       TYPE STANDARD TABLE OF osref,
        ls_survey_t_ref       TYPE osref,
        ls_survey_t           TYPE zcust_satisf_t.

  DATA: lo_agent_survey_text  TYPE REF TO zca_survey_text_pers,
        lo_survey_text        TYPE REF TO zcl_survey_text_pers,
        lo_query_manager_survey_tt TYPE REF TO if_os_query_manager,
        lo_query_survey_tt    TYPE REF TO cl_os_query,
        lo_filter_survey_tt   TYPE REF TO if_os_query_filter_expr,
        lt_survey_tt_ref      TYPE STANDARD TABLE OF osref,
        ls_survey_tt_ref      TYPE osref,
        ls_survey_tt          TYPE zcust_satisf_tt.

  DATA: lv_pattern            TYPE string.

* Step 1 - get survey data table entries
  lo_agent_survey = zca_survey_pers=>agent.

  lo_query_manager_survey_t  =  cl_os_system=>get_query_manager( ).
  lo_query_survey_t ?= lo_query_manager_survey_t->create_query( ).
  lv_pattern = iv_kunnr.
  lo_filter_survey_t = lo_query_survey_t->if_os_query_expr_
factory~create_like_expr(
      i_attr   = 'KUNNR'
      i_pattern = lv_pattern ).
  lo_query_survey_t->if_os_query~set_filter_expr( lo_filter_survey_t ).
```

```
  TRY.
    lt_survey_t_ref =
    lo_agent_survey->if_os_ca_persistency~get_persistent_by_query(
        i_query = lo_query_survey_t ).

  ENDTRY.

* convert references
  LOOP AT lt_survey_t_ref INTO ls_survey_t_ref.
    lo_survey ?= ls_survey_t_ref.
    ls_survey_t-kunnr        = lo_survey->kunnr.
    ls_survey_t-date_last    = lo_survey->date_last.
    ls_survey_t-time_last    = lo_survey->time_last.
    ls_survey_t-text_key     = lo_survey->text_key.
    ls_survey_t-zcust_satisf = lo_survey->zcust_satisf.
    APPEND ls_survey_t TO gt_survey.
  ENDLOOP.

* Step 2 - now retrieve linked survey texts
  lo_agent_survey_text = zca_survey_text_pers=>agent.
  LOOP AT gt_survey INTO ls_survey_t.

    lo_query_manager_survey_tt  =  cl_os_system=>get_query_manager( ).
    lo_query_survey_tt ?= lo_query_manager_survey_tt->create_query( ).
    lv_pattern = ls_survey_t-text_key.
    lo_filter_survey_tt = lo_query_survey_tt->if_os_query_expr_
factory~create_like_expr(
        i_attr   = 'TEXT_KEY'
        i_pattern = lv_pattern ).
    lo_query_survey_tt->if_os_query~set_filter_expr( lo_filter_survey_
tt ).

    TRY.
      lt_survey_tt_ref =
      lo_agent_survey_text->if_os_ca_persistency~get_persistent_by_
query(
          i_query = lo_query_survey_tt ).

    ENDTRY.

* convert text line references
    LOOP AT lt_survey_tt_ref INTO ls_survey_tt_ref.
      lo_survey_text ?= ls_survey_tt_ref.
```

```
      ls_survey_tt-text_key = lo_survey_text->text_key.
      ls_survey_tt-line     = lo_survey_text->line.
      ls_survey_tt-text     = lo_survey_text->text.
      APPEND ls_survey_tt TO gt_text.
    ENDLOOP.

    FREE: lo_query_manager_survey_tt,
          lo_query_survey_tt,
          lo_filter_survey_tt.

  ENDLOOP.

ENDMETHOD.
```

C The Author

Michael Koch was born in Aachen, Germany. His career in IT started in 1990 as a COBOL and RPG programmer. He soon started a double business degree course at Fachhochschule Aachen, which led to his first adventures in ABAP coding. Michael completed his course at Coventry University, England, where he graduated in 1998.

Since then, he has worked on national and multinational SAP projects in both consulting and end-user roles. During his work in the SAP arena, he covered various industries including automotive, FMCG, entertainment, insurance, telecoms, and high tech.

In 2002, Michael founded his own freelance business, Pixelbase (*http://pixelbase.co.uk*), which provides independent SAP development consulting. Pixelbase's focus is SAP NetWeaver web development, rich Internet applications, application architecture and design, customizations, upgrades, and interfaces. Pixelbase takes pride in its work and aims to give clients value for money by offering independent, real-world advice.

Michael now lives in England with his family. He has been a member of the SAP Mentor initiative since 2009.

Index

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

X

Z

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: laura.korslund@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/H3255.

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/H3255. 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.

About Us and Our Program

The website http://www.sap-press.com provides detailed and first-hand information on our current publishing program. Here, you can also easily order all of our books and e-books. For information on Galileo Press Inc. and for additional contact options please refer to our company website: http://www.galileo-press.com.

Legal Notes

This section contains the detailed and legally binding usage conditions for this e-book.

Copyright Note

Your Rights as a User

Digital Watermark

Trademarks

Limitation of Liability