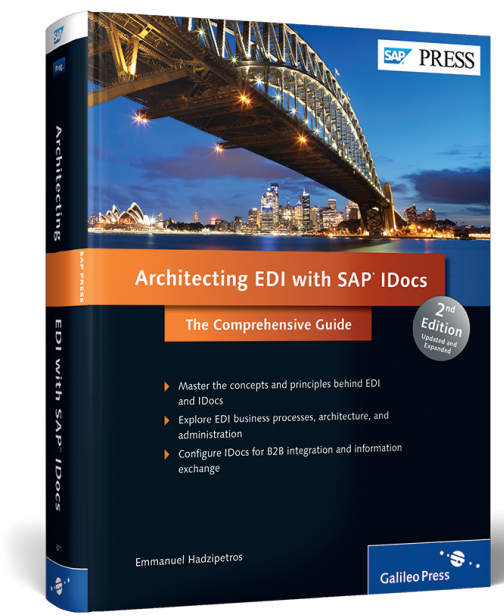
Emmanuel Hadzipetros

Architecting EDI with SAP® IDocs

The Comprehensive Guide



Galileo Press

Bonn • Boston

Contents at a Glance

Contents

ACT III Realizing the Dream—Building Acme's SAP EDI System

Prologue

Let's return to Hollywood and once again take the studio tour of Acme Pictures and revisit its legendary founder, Darryl Q. Fernhausen.

Since the first edition of this book was published, we've had time to reconsider Acme's implementation of an Electronic Data Interchange (EDI) system in an SAP environment. While the first edition covered a lot of ground, it left out even more. And, frankly, we made some mistakes, not all of which were typos.

This second edition is our attempt to plug some of these gaps. While much of the original remains, we've tried to put more emphasis on the business context and have added new interfaces from the purchasing cycle.

We have consolidated and expanded our discussions of message control and added a number of new custom tools and utilities. We have changed our IDoc format from ASCII to XML and introduced some basic concepts of working with, and extending, XML schema.

All in all, we hope that the result is a tighter book that covers more subjects in a clearer manner. But this book is still about SAP and EDI. And the basics haven't changed.

EDI is still the most widely used form of electronic commerce in the world today. It is highly unlikely that this will change anytime soon. EDI has been around for decades. It is reliable, proven, stable, and supported by long-established standards developed and maintained by such global bodies as the United Nations, the International Organization for Standardization (ISO), GS1, and the American National Standards Institute (ANSI).

If money talks, EDI has the eloquence of a Shakespearean actor. It supports trillions of dollars in transactions every year in a wide variety of industries. Many companies will not buy goods or services from suppliers that cannot exchange standard business documents through EDI.

In the United States, Wal-Mart, the world's largest retailer, is at the heart of a gigantic global EDI ecosystem with thousands of suppliers that rivals most governments

in its use of information and communications technology. If you want to sell to Wal-Mart, you can do it only through EDI.

The importance of EDI as an enterprise integration application for thousands of SAP customers is beyond dispute. EDI, and the large-volume batch processing of business transactions that it enables, is a key element of the SAP environment in thousands of locations around the world. Where EDI is present, most of the transactional data that flow between SAP and external trading partners are carried by EDI.

With more than 70 percent of the global business software market, SAP is the business system of record that a majority of EDI consultants, developers, and production support teams work with every day.

Together, SAP and EDI are the heart, bones, arteries, and brains of modern business and government organizations.

So why are we touring an imaginary Hollywood studio? Look beyond the glitz and the glitter and it is just another business. Acme Pictures is a good model for the challenges faced by many businesses when they implement SAP and EDI together.

Acme Pictures sells movies on DVD, a consumer product sold through retail, not all that different from other products that wind up on store shelves around the world. We could just as easily be talking about video games, pharmaceuticals, carpet, shoes, software, beverages, snack foods, or the book that I hope you are now enjoying.

For Acme Pictures, it's a simple equation: The volume of business it does with large retail chains across North America, and the huge number of documents that this business generates, can only be managed through EDI.

The other thing, of course, is that Hollywood is a lot of fun. But so is SAP and EDI, even if the details can get a little dry at times. I'll never forget the advice of a project manager during kick-off for an SAP implementation at a steel mill in Ohio: "Work hard and have lots of fun!"

The fun is in the creative process of designing and building a system that fits the client's business. It's discovering how far you can push the limits of your knowledge to provide your client with a useful system that will support its business for years to come.

The Book and Its Audience

Audiences are the lifeblood of Hollywood. Books, too. And while this book will never be made into a movie, it may provide some useful information to SAP and EDI consultants, developers, managers, and anyone else implementing, supporting, or considering EDI in an SAP environment.

This book is the culmination of my 20-year odyssey as an SAP consultant and developer with a perennial fascination for data flows and integration. It represents ideas about integration architecture considered throughout my SAP career, which includes project work in four countries, three continents, and such industries as beverages, electrical utilities, steel, electronics, textiles, pharmaceuticals, and entertainment.

We will take the studio tour of an SAP EDI implementation project, lovingly referred to as Plan Q from Outer Space, or just plain Plan Q—in honor of our imaginary studio's most famous film.

This project-based approach attempts to deal in a holistic manner with the entire SAP EDI ecosystem at Acme Pictures, beginning with an overview of the business it supports, just enough to discuss technical solutions. The real-world business of a Hollywood studio is far more complex and nuanced than we could describe in these pages.

This book is not an SAP or EDI programming guide. It does assume an ability to follow program logic and visualize end-to-end systems. We rely on standards and standard functionality wherever possible.

Where potential solutions to common problems are presented, we will step through the process flow and logic. But we will not write the code for you. Consider our logic as starting points for your own creative exploration. And forgive me in advance if some of our ideas don't add up for you.

Our real hope is that this book helps you formulate questions that you may not have considered. After all, every business, no matter how big or how small, is as unique as the people who run it.

Structure of the Book

The book loosely follows the phases of our imaginary SAP project with detours for background information about IDocs and EDI.

In deference to our Hollywood theme, the book is organized into four acts, which are meant to build your knowledge of Acme's business, its systems, and SAP IDoc and EDI development. A summary of each chapter follows.

Act I—Hollywood, DVDs, and the After Life of Movies

- **Chapter 1—Hollywood's B-Movie Queen Does SAP and EDI**
  This chapter introduces Acme Pictures, its visionary founder, Darryl Q. Fernhausen, and its unique approach to the movie business. We also touch on SAP EDI development strategy.
- **Chapter 2—The Blueprint: Discovery and Documentation**
  In this chapter we document Acme's DVD business and the legacy systems that support it. It also provides an overview of key customer and vendor processes, including purchasing and order-to-cash.
- **Chapter 3—Designing the New SAP EDI Architecture**
  We present the vision for the new system. We'll introduce to-be systems and interfaces, and the Resource Integration Manager (RIM). We also examine the key business processing cycles enabled by EDI.

Act II—Taming Chaos with Standards: EDI in an SAP Environment

- **Chapter 4—EDI: The Ugly Stepsister of E-Commerce**
  This chapter covers EDI for SAP professionals, including a brief tour of its fascinating history and introduce the major EDI standards, with special emphasis on EDIFACT and ANSI X12.
- **Chapter 5—Real-World Business Process Integration with EDI**
  Relationships and integrating processes between trading partners are at the heart of EDI. In this chapter we look at the role of Acme's EDI RIM, with its adapters and services, and its connections to SAP.
- **Chapter 6—EDI Architecture in SAP: IDoc Basics**
  In this chapter we discuss Intermediate Documents (IDocs)—the intelligent messages defined by the Data Dictionary and the underlying logic that determines how they are used.
- **Chapter 7—Configuring IDocs in SAP for EDI Exchange**
  From partner profiles to message control and mapping tables, inbound and outbound IDoc configuration and processing flows in SAP are the focus of this chapter.

- **Chapter 8—Custom IDocs and IDoc Extensions**
  An introduction to IDoc development in SAP. We go over development tools and process flows and build, code, and configure one custom and one extended IDoc.

Act III—Realizing the Dream: Building Acme's SAP EDI System

- **Chapter 9—Generating the PO for Replication Services**
  This chapter defines the function and technical setup for the outbound purchase order for replication services to Acme's contract manufacturer, including configuration of message control to output an ORDERS IDoc.
- **Chapter 10—The Inbound Goods Receipt**
  In this chapter we discuss the functionality and configuration of inbound inventory adjustments and goods receipt through an EDI X12 867 transaction.
- **Chapter 11—Processing the Inbound Supplier Invoice**
  We review posting requirements for the inbound supplier invoice for contract manufacturing services, including invoice verification and configuration.
- **Chapter 12—The Inbound Customer Purchase Order**
  This chapter examines the inbound X12 850 to ORDERS IDoc customer purchase order, including logic for SDQ processing and code to block posting of duplicate POs to SAP Sales Orders.
- **Chapter 13—Building the Outbound Order Confirmation**
  Covers the generation of the outbound ORDRSP confirmation from an SAP sales order. We'll build an extended IDoc and look at logic for a custom program to bundle multiple sales orders from the same SDQ PO into a single X12 855 interchange.
- **Chapter 14—Sending a Shipping Order to the Supplier**
  We discuss the outbound SHPORD to X12 830 shipping order to the supplier, including message control configuration enabling output of IDocs from the SAP delivery document.
- **Chapter 15—The Inbound Shipping Confirmation**
  This chapter focuses on the X12 856 shipping confirmation from the vendor, which updates pick quantity and posts goods issue in the delivery document in Acme's SAP system after the order ships.
- **Chapter 16—The Advanced Shipping Notice to the Customer**
  In this chapter we focus on the advanced ship notice (ASN), which tells the

customer what to expect in its shipment. We will emphasize the critical business requirement for accuracy and timeliness and discuss conditions for creating the DESADV IDoc.

- **Chapter 17—Generating the Outbound Customer Invoice**
  Here we cover the INVOIC to X12 810 customer invoice, generated from the SAP billing document. We detail output requirements for the IDoc and step through a custom ALV grid program for changing the PO number in the IDoc.
- **Chapter 18—Processing the Inbound Payment Advice**
  The focus of this chapter is the inbound payment advice, which records details of a customer payment on all invoices, including debits and credits. We also discuss common issues with very large X12 820 files.

Act IV—Finishing Touches

- **Chapter 19—Extending the Interface: Custom IDoc Tools**
  Fun with ABAP, ALE, and XML as we look at custom utilities that take advantage of standard SAP functionality.
- **Chapter 20—Testing the EDI System in SAP**
  Acme's testing strategy is the focus of this chapter. We examine the composition and role of the test team and outline the key test phases.
- **Chapter 21—Troubleshooting and Recovery**
  Defining success and failure in Acme's SAP EDI architecture. We'll look at standard monitoring tools and consider situations that appear successful but could lead to errors in later stages of the EDI cycle.
- **Epilogue**
  With the project complete, the integration team relaxes at a famous Hollywood watering hole to toast the successful release of an Acme *film noir* classic and the success of the new SAP EDI system.

Acknowledgments

If no man is an island, as the English poet and preacher John Donne once observed, the same is especially true for authors, who often labor in solitude and obscurity. During the long, lonely hours spent writing a book, it is easy to forget the many people who enrich our lives and our work every day.

The quest to define, begin, complete, and rewrite this book was a labor of love that extended over many years. It would have been a mission impossible without the

people—family, friends, and colleagues—who sometimes endured yawn-inducing dissertations of SAP integration issues.

First and foremost, this is dedicated to my son and two daughters and especially to the grandchildren that I've been blessed with since publication of the first edition. The future belongs to you.

To all my family and friends, wherever in or out of this crazy, beautiful world you may find yourselves: thank you for being you.

A good programmer never stops learning. A good consultant learns so that he can pass on his knowledge to benefit his client. A passion to learn and to acquire and pass on new skills is the key to success and to having fun in this business. Learning means working with other people—listening, studying, discussing, playing, and poking around systems.

I've been lucky in the colleagues that I've worked with over the years. I've learned so much from so many people that it's impossible to remember them all. So thanks to everybody. You know who you are. It's been a privilege working with you.

I also want to thank the folks at SAP PRESS for their support and continuing belief in this project.

Finally, I want to express a heartfelt thank you to the countless numbers of extraordinary working people in Hollywood who labor quietly every day in the offices and back lots of the studios beyond the glare of the klieg lights.

When the director shouts "Lights…cameras…action!" they turn on the lights, run the cameras, build the sets, feed the crews, park the star wagons, clean up, and run the IT systems that keep the business humming.

They are, always have been, and always will be the real stars of Hollywood.

"I couldn't build my way out of a paper bag," Darryl Q would tell the carpenters who built his cheesy sets. Acme's legendary founder knew that the success of his films depended on his workers' ability to build. Just as the success of the new SAP EDI system, and the business that relies on it, rests on the ability of the team to build and extend IDocs. So let's delve into this fascinating topic and go over the tools and techniques we use to craft our own custom IDocs.

8 Custom IDocs and IDoc Extensions

Now we're getting to the interesting part: creating, coding, and configuring custom IDocs and extending standard IDocs.

Our custom IDoc will post an X12 846 inventory report to a custom table. Acme will use the data to write custom reports to support inventory balancing.

We'll build our extended IDoc from basic type ORDERS05 for an outbound supplier purchase order using message type ORDERS. We'll add a BOM segment just beneath the E1EDP01 item parent to send the bill of materials associated with a purchase order to Acme's third-party manufacturer, Disc Services International, in the outbound 850 supplier purchase order.

Let's begin by taking a quick tour of the tools that we'll use to develop and configure our IDoc interfaces.

8.1 IDoc Development and Configuration Tools

Our starting point for the development of custom IDocs and extensions is the EDI area menu. You can access it with Transaction WEDI from the SAP Easy Access main menu. WEDI is an area menu and cannot be accessed from other transaction screens.

The key IDoc development tools that we'll be using for our customization are clustered in the Development folder, illustrated in Figure 8.1.

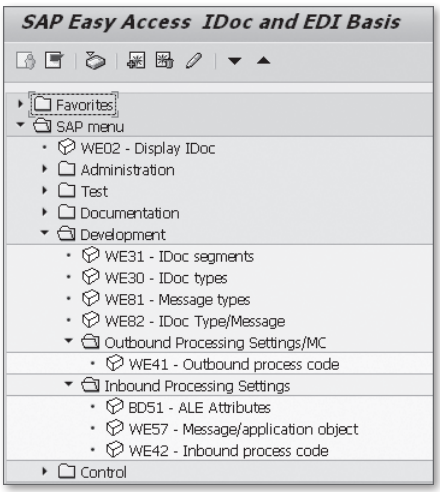


Figure 8.1 IDoc Development Tools in the EDI Area Menu

There are other development tools that we'll be using in addition to those in the EDI area menu. We'll look each tool in the order of use in our development workflow.

8.1.1 Transaction SE11—Data Dictionary

Get to the Data Dictionary with Transaction SE11 through the Repository Information System (Transaction SE90) in the ABAP Dictionary folder, or through SAP menu Tools • ABAP Workbench • Development • ABAP Dictionary.

We've touched on the role of the Data Dictionary in defining IDoc syntax and architecture (see Chapter 6, Section 6.2, IDoc Architecture and the Data Dictionary). We'll use the Data Dictionary to create the following objects:

- Domains: Tables DD01L and DD01T.
- Data elements: Tables DD04L and DD04T. Domains and data elements will be used for custom fields in custom segments.
- Structures and transparent tables to store master and transactional data for use in custom programs: Tables DD02L and DD02T.

We'll also use the Data Browser to look at data stored in transparent tables and to do informal extracts for analysis. The Data Browser can be reached with Transaction SE16.

8.1.2 Transaction WE31—Segment Editor

We'll use the segment editor to build and edit segments for custom and extended IDoc basic types. The segments are saved as structures in the Data Dictionary in tables DD02L and DD02T.

As shown in Figure 8.2, the segment editor keeps track of all external names and versions of the segment, along with its string lengths, the number of fields it contains, and the SAP release number current when it was last changed.

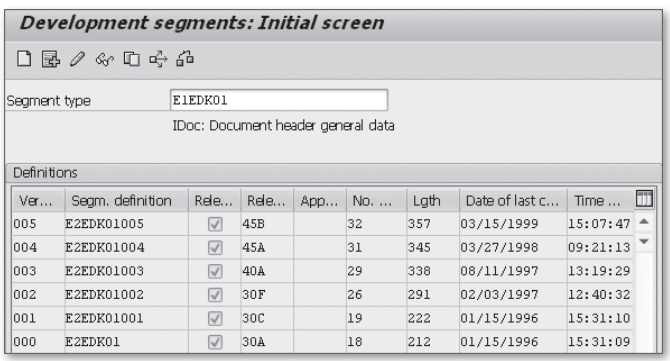


Figure 8.2 Segment Versions in the Initial Screen of the Segment Editor

Double-click any segment definition row to get to its fields, as illustrated in Figure 8.3 for segment type E1EDK01.

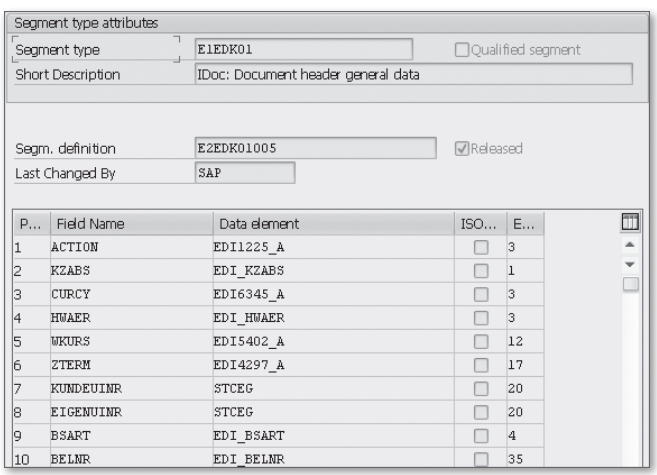


Figure 8.3 Field List in Segment Editor

Fields are added to this screen during the creation of segments, and data elements are assigned to the fields. The data element is linked to a domain. Double-clicking on the data element name opens the DISPLAY DATA ELEMENT screen. Double-clicking on the domain name opens the DISPLAY DOMAIN screen. If there are qualifiers associated with the domain in a value range, you can click on the VALUE RANGE tab to see them.

The segment is activated after it has been released in the initial screen by selecting menu option EDIT • SET RELEASE. The segment must be released before it can be used in a basic type.

To get to the segment editor, open the DEVELOPMENT folder in the WEDI area menu, and double-click IDOC SEGMENTS, or use Transaction WE31, SAP menu TOOLS • ALE • ALE DEVELOPMENT • IDOC • IDOC TYPE DEVELOPMENT • SEGMENTS.

8.1.3 Transaction WE30—IDoc Type Editor

We use the type editor to assemble segments into custom or extended IDoc basic types. It's also a great way to display the structure and segment attributes of standard IDoc basic types such as ORDERS05.

Double-click on any segment name to open the ATTRIBUTE DISPLAY dialog. It records key parameter values for the segment that controls its place within the IDoc basic type, including the following:

- Segment type name
- Mandatory segment flag
- Minimum and maximum number of occurrences
- Parent segment number
- Hierarchy level

Click the SEGMENT EDITOR button to open the FIELD DISPLAY screen of the segment editor.

The custom or extended IDoc is released after it's been assembled in the initial screen of the type editor with menu option EDIT • SET RELEASE. It must be activated before it can be used in an interface or transported to other SAP clients.

You can get to the type editor by opening the DEVELOPMENT folder in the WEDI area menu, and double-clicking IDOC TYPES, or by using either Transaction WE30

or SAP menu Tools • ALE • ALE Development • IDoc • IDoc Type Development • IDoc Types.

IDoc basic and extended types are stored in table IDOCSYN.

8.1.4 Transaction WE81—Logical Messages

Create logical message types when building custom IDoc in the message type editor. To get there, double-click Logical Messages in the Development folder in the WEDI area menu, use Transaction WE81, or follow menu path Tools • ALE • ALE Development • IDoc • IDoc Type Development • Logical Messages.

Logical message types are stored in tables EDMSG and EDIMSGT.

8.1.5 Transaction WE82—Message to Basic Type Link

This links the message type to the IDoc basic type, providing structure to the logical message. Multiple message types can be linked to one basic type. Transaction WE82 is also used to link IDoc extensions to messages and basic types. Message, basic, and extended types are linked in table EDIMSG.

The relevant menu paths are IDoc Type/Message in the Development folder of the WEDI area menu or Tools • ALE • ALE Development • IDoc • IDoc Type Development • IDoc Type for Message.

8.1.6 Transaction SE37—Function Editor: Function Groups

All function modules are created within a function group. Create function groups in the Program Library folder in the Repository Information System (Transaction SE80). Click the Edit object button and navigate to the Function group tab, as illustrated in Figure 8.4.

You can also use Transaction SE37, SAP menu path Tools • ABAP Workbench • Development • Function Builder, or menu path GoTo • Function Groups • Create Group.

Function groups are programs that logically group related function modules into a common package with global data types, declarations, constants, and so on. The naming convention for the function pool program generated for the function group is always *SAPL<FUNCGRP>*, where FUNCGRP is the name of the function group.

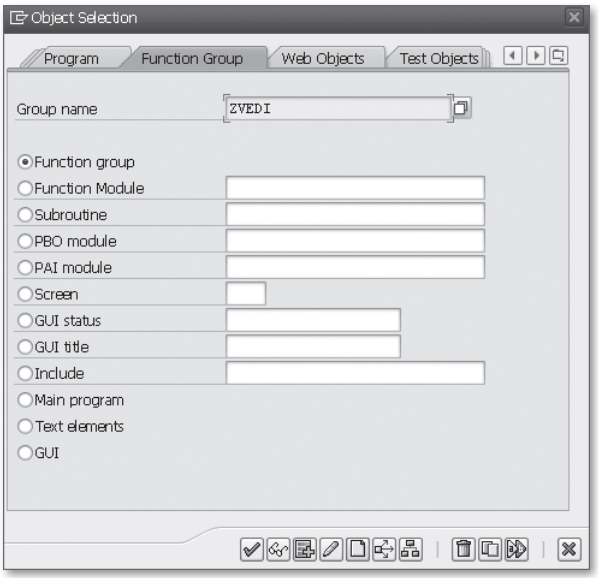


Figure 8.4 Creating a Function Group in the Repository Info System

For example, function group `EINM` (Figure 8.5), which includes function `IDOC_OUTPUT_ORDERS`, has the program name `SAPLEINM`.

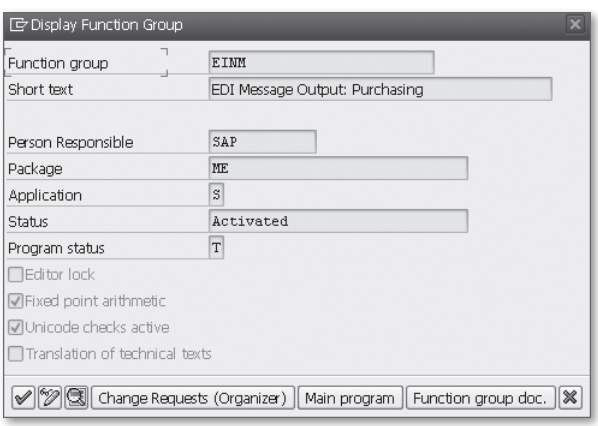


Figure 8.5 Attributes for Function Group EINM

The main function group program always contains the following includes:

- `L<FUNCGRP>TOP`
  Global data declarations for the function group.

- L<FUNCGRP>UXX
  Stores includes with the function modules.
- L<FUNCGRP>F0X
  Form subroutines called by the functions. The X character can be a number from 0 to N or a letter to distinguish multiple include programs.

There could also be other includes that do not follow this naming convention with form routines that perform special functions.

We'll create one function group per function module for Acme's custom IDoc development.

8.1.7 Transaction SE37—Function Editor: Function Modules

This is where the code hits the road. Most of our programming in the IDoc interface will be in the Function Builder.

Function modules are created using the same transactions and menu paths as function groups.

Functions are encapsulated programs that perform one function. They have a standard interface with import and export parameters and can transfer internal tables for processing at runtime. They also return error codes that can be trapped for error message processing.

The great bulk of the work of the IDoc interface is done with functions, even with the trend toward object-oriented programming in SAP. Functions are so pervasive throughout the interface that it's highly unlikely they'll be replaced any time soon, although method calls are increasingly being used and exits, in particular, are mirrored by BAdIs.

For example, functions are used to build partner profiles within WE20. This is a three-step process implemented by four function calls that populate the partner profile tables:

1. Build the general view:
   - EDI_AGREE_PARTNER_INSERT: Table EDPP1
2. Build outbound parameters and message control:
   - EDI_AGREE_OUT_MESSTYPE_INSERT: Table EDP13

   - EDI_AGREE_OUT_IDOC_INSERT: Table EDP12
3. Build inbound parameters:
   - EDI_AGREE_IN_MESSTYPE_INSERT: Table EDP21

Functions are also used as customer exits for IDocs and other standard SAP applications. Customer exits allow the user to extend the functionality of standard IDoc functions to accommodate unique business requirements.

IDocs are mostly processed through function modules. Workflow tasks can be used to process some IDocs but we won't be doing this in our Acme implementation except for the standard task that processes the STATUS message type. Our main concern here is with the IDOC_INPUT and OUTPUT functions, which follow the naming conventions:

- IDOC_INPUT_<MESSAGE>
  The application programming interface (API) for inbound functions is described in the upcoming subsection "API for Inbound IDoc Processing Functions." An example is function IDOC_INPUT_ORDERS.
- IDOC_OUTPUT_<MESSAGE>
  The API for outbound functions with message control is described in the upcoming subsection "API for Outbound IDoc Processing Functions." An example is function IDOC_OUTPUT_INVOIC.
- MASTER_IDOC_CREATE_<MESSAGE>
  These are standalone IDocs that are not generated by message control. An example is the BOMMAT message that sends bills of material master data with function MASTER_IDOC_CREATE_BOMMAT.

The Function Builder is divided into the seven tabs seen in Figure 8.6:

1. Attributes: Administrative data for the function:
   - Function group, program names, descriptions, and package
   - Processing type flag, including remote-enabled (RFC) function
2. Import: Structured strings based on Data Dictionary types used to bring data into the function for processing at runtime.
3. Export: Structured strings based on Data Dictionary types used to return data from the function after processing at runtime.

4. Changing: Tables or structured strings to carry data that will be changed by the function by runtime processing. Rarely used.
5. Tables: Internal tables based on Data Dictionary tables or structures that will carry table data for processing at runtime.
6. Exceptions: String descriptions of error conditions that can be raised at runtime. Exceptions end processing at the point they are raised and return control to the calling program, which can then use them to trigger error message or other processing.
7. Source Code: Default view where the function's ABAP code is written. The function name at the top of the window is followed by a commented block documenting import and export parameters, tables, and exceptions, as illustrated in Figure 8.6.

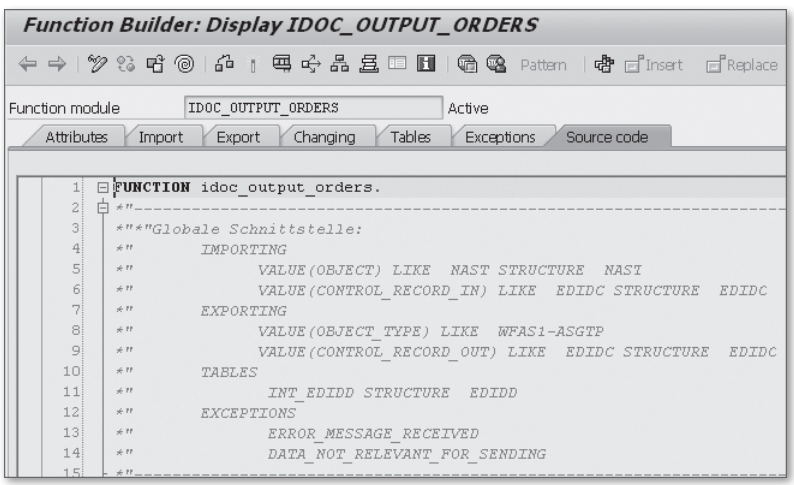


Figure 8.6 Function Builder Source Code Screen

API for Outbound IDoc Processing Functions

The API for outbound IDoc processing functions is standard for all outbound processing actions that use message control. The outbound function is identified and called by form `EDI_PROCESSING` in program `RSNASTED`, but can also be used in custom code. Check the Function Builder for the structures of each parameter.

- Importing: Parameters passed from `RSNASTED`.
  - `OBJECT`: The NAST table record containing the document key and output type for the business document that will be read to build the IDoc.

- CONTROL_RECORD_IN: The control record built by RSNASTED from the partner profile read before the IDoc function is called.

- Exporting: Parameters passed back to RSNASTED.
  - OBJECT_TYPE: Business object name for the business document. Links the IDoc to the business document.
  - CONTROL_RECORD_OUT: Fully populated control record built by the IDoc processing function returned to RSNASTED.
- Tables: Internal tables that pass data arrays from the calling function or back to it from the IDoc processing function.
  - INT_EDIDD: Output. Returns the data records for the IDoc from the processing function.
- Exceptions: Parameters that raise errors in the IDoc processing function. Stop execution of the function at the point the error is trapped and return control to the calling program.
  - ERROR_MESSAGE_RECEIVED: Identifies errors called during IDoc processing.
  - DATA_NOT_RELEVANT_FOR_SENDING: Added to customer functions and method calls in outbound IDoc processing.

API for Inbound IDoc Processing Functions

The API for inbound IDoc processing functions is standard for all inbound processing actions. Inbound functions are called by the standard IDoc interface function IDOC_INPUT, but you can also call them from custom code. Check the Function Builder for the structure of each parameter.

- Importing: Parameters passed from the calling function.
  - INPUT_METHOD: Used only for call transaction posting. Default is blank for background mode. Other modes are A for All screens in foreground, and E for Display Error screen only.
  - MASS_PROCESSING: For workflow processing. Default is blank.
- Exporting: Parameters passed back to the calling function.
  - WORKFLOW_RESULT: Workflow error handling. Triggers tasks that pass success or error messages to the SAP workplace inbox or other workflow targets.

- APPLICATION_VARIABLE: Advanced workflow programming. Default value is space.
- IN_UPDATE_TASK: Triggers a follow-up task to handle database commit. Default value is space for no update task. "X" delays posting until an explicit commit is called.
- CALL_TRANSACTION_DONE: Set to "X" if the status record isn't updated in the code of the IDoc processing function. In this case, it is updated within the calling function's processing flow.
- DOCUMENT_NUMBER: This is an example of how the standard can vary. IDOC_INPUT_ORDERS uses this parameter to return the sales order number after posting, but not all IDoc functions do this.

- Tables: Internal tables that pass data arrays from the calling function or back to it from the IDoc processing function.
  - IDOC_CONTRL: Input. Passes control record data to the processing function.
  - IDOC_DATA: Input. Passes data records to the IDoc processing function for posting to the document.
  - IDOC_STATUS: Output. Returns status records indicating success or failure in posting for each IDoc passed to the function. Linked to IDOC_CNTRL and IDOC_DATA through the IDoc number.
  - RETURN_VARIABLES: Output. Returns additional posting results for each IDoc that are used in workflow processing.
  - SERIALIZATION_INFO: Output. Used by the IDoc interface to sort a batch of IDocs in a particular order.

8.1.8 Transaction SMOD—SAP Enhancements

Enhancements collect one or more customer exits that are called from strategic points in the code of the IDoc processing function. The customer exits are grouped together in components within each enhancement.

Click on the Components button to access the exits. All customer exits that are associated with the component are listed in the Change Project screen.

For example, enhancement SIDOC001 contains one component: customer exit function EXIT_SAPLEDI1_001, used to process the control record of the IDoc before

it's created on the IDoc database during inbound and outbound processing (see Figure 8.7). It gives the customer the opportunity to add data to the control record that aren't provided by standard processing.

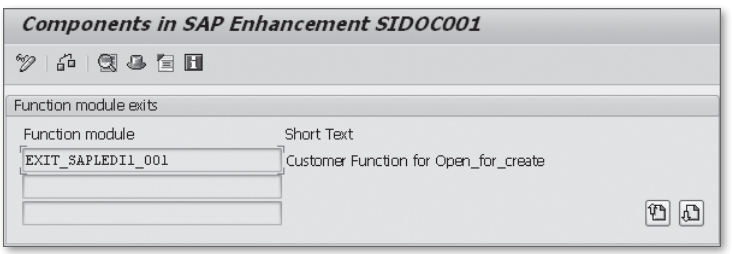


Figure 8.7 Components for Enhancement SIDOC001

This list of functions is our entry point to code the customer exit. Double-click the exit name to open the Function Builder Source code window. Note the include statement and program name. We write the code in the include program. But we don't do this through the enhancement object. We first assign the enhancement to a modification project.

Choose Enhancements • Customer Exits • Enhancements in the Repository Information System, use Transaction SMOD, or follow menu path Tools • ABAP Workbench • Utilities • Enhancements • Definition.

Enhancements are stored in tables MODSAP and MODSAPT.

8.1.9 Transaction CMOD—Project Management for SAP Enhancements

We'll use CMOD to create and manage modification to code customer exits for IDoc functions. One or more enhancements can be assigned to each project. For the sake of simplicity, we'll assign only one enhancement per project.

To get to modification projects, use Transaction CMOD, go to Enhancements • Customer Exits • Projects in the Repository Information System, or follow menu path Tools • ABAP Workbench • Utilities • Enhancements • Project Management.

Modification projects are stored in tables MODACT and MODTEXT.

8.1.10 Transaction WE57—Link Function to Message and Basic Type

This is a key piece of configuration for setting up partner profiles and processing inbound IDocs. It links a processing function, workflow, or task to basic types, extended types, and logical messages.

Different function modules can be assigned to different messages linked to the same basic type. We can add a message code and/or message function and associate different processing functions to the same message and basic type. The message code and function are optional but, if used, become a mandatory part of the partner profile read key.

This gives us tremendous control over how IDocs can be processed under different scenarios and use cases.

The standard link between logical message ORDERS, basic type ORDERS05, and function `IDOC_ORDERS_INPUT` posts a sales order.

We can add a new link for the same message and basic type to a custom function by creating a message code that will be used, for example, to route IDoc data to a custom table for follow-up reporting. The possibilities are limited only by our imagination and our development budget.

These links drive our partner profile. If we were to link ORDERS and ORDERS05 to custom function module `ZSD_INPUT_ORDREPORT` and message code SD0, we would create a partner profile with the following parameters:

- Customer number
- Partner type KU
- Partner function SP
- Message type ORDERS
- Message code SD0
- Custom process code linked to the custom function

Although an optional data element, the message code becomes part of the mandatory key for the partner profile if it is used. It must be present in the control segment field MESCOD to trigger this custom processing.

To get to the link editor, use Transaction WE57, select Development • Inbound Processing Settings • Message/Application object from the WEDI area menu,

or follow SAP menu path Tools • ALE • ALE Development • IDOC • Inbound Processing • Function Module • Assign IDoc Type and Message Type.

Table EDIFCT stores these links. Table TOJTB adds a link to the corresponding object in the Business Object Repository (BOR).

8.1.11 Transaction BD51—Define IDoc Attributes

Attributes determine how inbound IDoc functions are processed from the following options:

- **0**
  Mass processing for functions that use direct input to the database to post to a document or other data object.
- **1**
  Individual input for functions that use call transaction to post.
- **2**
  Individual input with a call transaction that locks the IDoc.
- **Dialog allowed**
  Allows screen display during call transactions when IDocs are processed in foreground mode.

To define IDoc attributes, choose Development • Inbound Processing Settings • ALE Attributes in the WEDI area menu, use Transaction BD51, or follow menu path Tools • ALE • ALE Development • IDoc • Inbound Processing • Function Module • Maintain Attributes.

Attributes for functions are stored in table TBD51.

8.1.12 Transaction WE42—Inbound Process Code

The inbound process code links one IDoc processing function to one or more message types. The process code is required for the inbound partner profile.

Standard process code REMA, for example, is linked to function module `IDOC_INPUT_REMADV` and to logical messages CREADV, DEBADV, and REMADV.

If a message code and/or message function has been added to the link between the logical message, basic type, and processing function in WE57, then the inbound process code must also contain a record with the link between the message type and the message code and/or message function. This is illustrated in Figure 8.8.

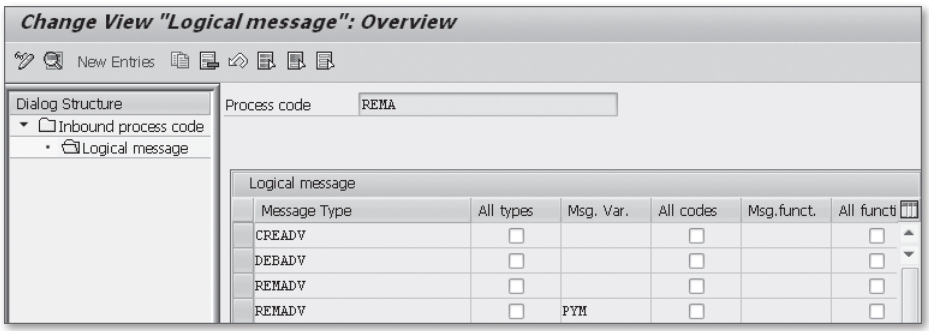


Figure 8.8 Multiple Use Cases Driven by the Process Code

This allows the same function and message to be used in different use cases. You would need to write custom code in a user exit to take advantage of this flexibility.

Use Transaction WE42 to configure the inbound process code or follow menu paths Development • Inbound Processing Settings • Inbound process code in the WEDI area menu or Tools • ALE • ALE Development • IDOC • Inbound Processing • Define process code in the SAP menu.

The inbound process code is stored in table EDE2T. Its link to the message type key is stored in TMSG2 and to the function module in TEDE2.

8.1.13 Transaction WE41—Outbound Process Code

The outbound process code links one IDoc processing function to one or more message types. Like the inbound, the outbound process code supports multiple use cases through the addition of message codes and/or message functions to the logical message. The configuration screens for the outbound process code are identical to WE41.

The outbound process code is used in the message control screen of the partner profile. It triggers the linked function that reads data from an SAP business document to build and write an IDoc to the database.

Use Transaction WE41 to configure the outbound process code or go to Development • Outbound Processing Settings/MC • Outbound process code in the WEDI area menu or Tools • ALE • ALE Development • IDOC • Outbound Processing • Define process code in the SAP menu.

The outbound process code is stored in table EDE1T. Its link to the message type key is in TMSG1 and to the message and function module in TEDE1.

8.2 Building a Custom IDoc: Inbound Inventory Report

Now it's time to have a little fun. We're going to build a custom IDoc that maps to an inbound X12 846 inventory report from Acme's vendor Disk Services International (DSI). Before we begin, we'll outline the workflow for creating a custom IDoc from scratch.

8.2.1 Custom IDoc Development Workflow

Figure 8.9 outlines the three steps for building a custom IDoc: develop the IDoc, code the IDoc function, and configure the interface.

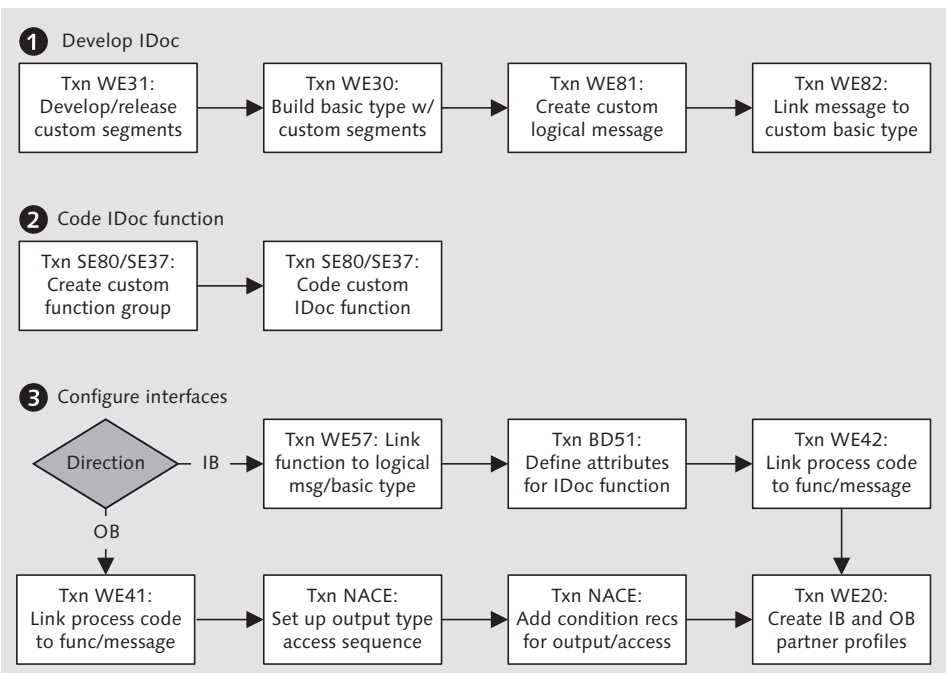


Figure 8.9 Custom IDoc Development Workflow

We'll get into the gritty details as we build our 846 interface. For now, we'll provide a high-level checklist of development tasks:

1. If required, first create domains and data elements in the data dictionary with Transaction SE11.
2. Create segments in the segment editor with Transaction WE31 using standard or custom data elements to create fields.
3. Assemble custom and standard segments into a basic type in the IDoc type editor using Transaction WE30.
4. Create a custom logical message with Transaction WE81.
5. Link the message to the custom basic type with Transaction WE82.
6. Create a new function group for the IDoc processing function with Transaction SE37.
7. Create and code a function module to process the custom IDoc within the new function group with Transaction SE37.
   - Inbound functions post the IDoc to a document or record.
   - Outbound functions extract data to build and distribute an IDoc.
8. For an inbound IDoc, do the following configuration:
   - Link the custom IDoc function to the logical message and basic type with Transaction WE57.
   - Define attributes for the function with Transaction BD51.
   - Create a process code linking the logical message to the custom IDoc function with Transaction WE42.
   - Create an inbound partner profile for the customer, message, and process code with Transaction WE20.
9. For an outbound IDoc, do the following configuration:
   - Create a process code linking the logical message to the custom IDoc function with Transaction WE41.
   - Set up message control: output type, access sequence, and condition record for the output and access with Transaction NACE.
   - Create an outbound partner profile for the customer, message, XML file port, basic type, and message control, including the process code with Transaction WE20.

8.2.2 Building the IDoc Interface

The inbound X12 846 inventory report from DSI carries end-of-day inventory summaries by material for finished movies on DVD and components such as packaging, blank disks, labeling, inserts, stickers, and so on.

Our IDoc will in3sert 846 data into a custom table (ZEDINVRPT) that will be used for daily inventory balancing reports. Table 8.1 list its fields; these are labeled as *M* for mandatory and *O* for optional.

Field	Data Element	Description	Req.
MANDT	MANDT	SAP client	M
INVRPTNO	CHAR10	Unique ID for inventory report rec	M
MATNR	CHAR18	Material number	M
WERKS	CHAR4	Plant/warehouse	M
LGORT	CHAR4	Storage location	O
MENGE	QUAN13	Inventory quantity	M
MEINS	UNIT3	Unit of measure	M
CREDAT	DATS8	Create date	O
CRETIM	TIMS6	Create time	O
UPDAT	DATS8	Date record last changed	O
UPTIM	TIMS6	Time record last changed	O

Table 8.1 Structure of Custom Table ZEDINVRPT

We'll name our message ZINVRPT, which follows the SAP convention of using EDIFACT message names for similar objects. We'll name our basic type ZINVRPT01 and it will have the two segments described in Table 8.2.

Segment	Description	Usage	Repeat
ZIVRPH	Header-level data	M	1
ZIVRPD	Item-level detail	M	N

Table 8.2 Structure of IDoc Basic Type ZINVRPT01

Creating Custom Table ZEDINVRPT

First we create the custom table ZEDINVRPT. The IDoc will post its data to this table and it is a mandatory part of the development cycle.

1. Run Transaction SE11 and enter table name ZEDINVRPT into the Database table field.
2. Click Create to open the Dictionary: Change Table screen in the Delivery and Maintenance tab. Enter a table description in the Short Description field and select Delivery Class A for application data (master and transaction data).
3. In the Fields tab, enter the values shown in Figure 8.10.

Transp. Table: ZEDINVRPT New
Short Description: 846 EDI inventory report interface data

Attributes | Delivery and Maintenance | Fields | Entry help/check | Currency/Quantity Fields

Srch Help | Predefined Type

Field	Key	Initial Values	Data element	Data Type	Len...	Dec...	Short Description
MANDT	☑	☐	MANDT	CLNT	3	0	Client
INVRPTNO	☑	☐		CHAR	10	0	Inventory report ID number
MATNR	☑	☐	MATNR	CHAR	18	0	Material Number
WERKS	☑	☐	WERKS_D	CHAR	4	0	Plant
LGORT	☐	☐	LGORT_D	CHAR	4	0	Storage Location
MENGE	☐	☐	MENGE_D	QUAN	13	3	Quantity
MEINS	☐	☐	MEINS	UNIT	3	0	Base Unit of Measure
CREDAT	☐	☐		DATS	8	0	Create date
CRETIM	☐	☐		TIMS	6	0	Create time
UPDAT	☐	☐		DATS	8	0	Date record last changed
UPTIM	☐	☐		TIMS	6	0	Time record last changed

Figure 8.10 Field Structure of Inventory Report Table ZEDINVRPT

4. In the Currency/Quantity Fields tab a link must be made between the quantity field MENGE and a corresponding unit of measure field in an existing table. Since this is an inventory report, we'll use MSEG-MEINS.
5. Save the table and assign it to a transport. Click Technical Settings and set Data Class to APPLI (Transaction Data, Transparent Tables) and Size Category to 0 (Data records expected: 0 to 4,300).
6. Save and click Revised <-> Activate.
7. Return to the Fields tab and click Activate (Ctrl+F3) in the toolbar above the table name field.

Our custom is ready to accept data.

Creating the Segments

We'll begin by creating the segments. Table 8.3 lists the structure of the header segment ZIVRPH.

Pos	Field	Data Element	Description
01	CREDAT	EDI_CCRDAT	Date IDoc created
02	CRETIM	EDI_CCRTIM	Time IDoc created

Table 8.3 Field Structure of Segment ZIVRPH

Table 8.4 lists the field structure of details segment ZIVRPD.

Pos	Field	Data Element	Description
01	MATNR	MATNR	SAP material number
02	WERKS	WERKS_D	Plant/warehouse
03	LGORT	LGORT_D	Storage location
04	MENGE	MENGE_D	Inventory quantity
05	MEINS	MEINS	Unit of measure

Table 8.4 Field Structure of Segment ZIVRPD

We'll first create ZIVRPH in the segment editor (see Figure 7.13). Go to Transaction WE31, enter "ZIVRPH" into the Segment type field, and click Create.

1. The Create segment definition screen opens. Enter a description in the Short Description field:
2. Create the following fields:
   - Enter "CREDAT" in Field Name and "EDI_CCRDAT" in Data element.
   - Enter "CRETIM" in Field Name and "EDI_CCRTIM" in Data element.
3. Click Save. Assign the segment to a package and a change request. It should like Figure 8.11.

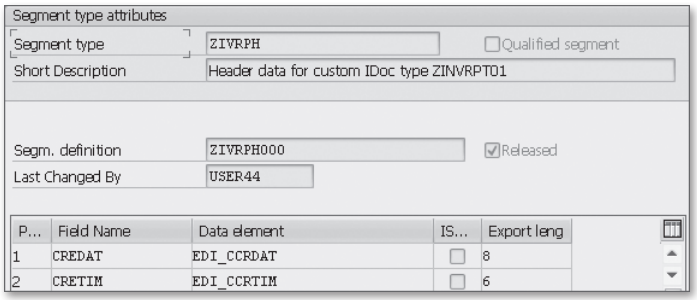


Figure 8.11 Fields Added to the Custom Segment

4. Press F3 to back out to the segment editor's opening screen. Release the segment by selecting menu option Edit • Set Release. Once released, the initial screen will look like Figure 8.12.

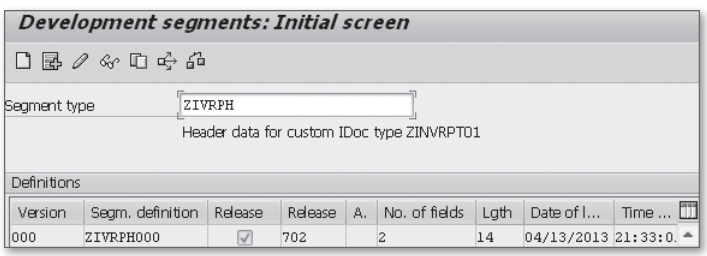


Figure 8.12 Segment ZIVRPH After It's Been Released

5. Follow the same steps to create and release details segment ZIVRPD using the field and data element names in Table 8.4.

Building an IDoc Basic Type

We'll create our custom IDoc basic type with Transaction WE30.

1. Enter "ZINVRPT01" into the Obj. name field, select Basic type, and click Create. The Create Basic Type dialog opens after the system informs you that the name is longer than eight characters. The following are radio buttons under New basic IDoc type:
   - Create new builds a new custom IDoc type.
   - Create as copy copies an existing IDoc type that we can change.

- Create successor creates a new release version of an existing custom IDoc type.

2. Select Create new, and click OK to open the IDoc type editor. We'll assemble our basic type from segments in this screen. To add segments, put the cursor on the IDoc type root name and click Create segment, press Shift+F6, or follow the menu path Edit • Create Segment.
3. The Maintain Attributes dialog opens, as in Figure 8.13. To add header segment ZIVPRH, do the following:
   - Enter "ZIVRPH" in the Segm.type field.
   - Select the Mandatory seg. checkbox.
   - Enter "1" in Minimum number and "1" in Maximum number.

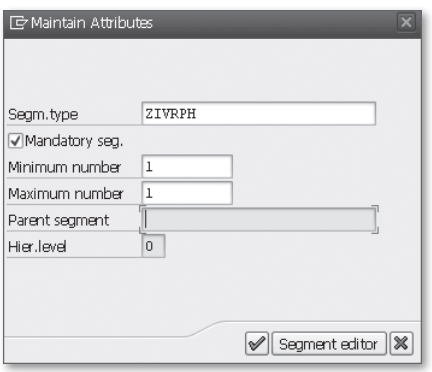


Figure 8.13 Adding a New Segment to the IDoc Type

4. Click OK. ZIVRPH is inserted as a child of the basic type root. Select ZIVRPH and click Create segment to add the details segment.
5. The Segment Hierarchy dialog opens. Select Add segment type as child.
6. Click OK to open the Maintain Attributes dialog. Do the following for the next segment:
   - Enter "ZIVRPD" in the Segm.type field.
   - Select the Mandatory seg. checkbox.
   - Enter "1" in Minimum number and "999999" in Maximum number.
7. ZIVRPD is added as a child segment to ZIVRPH, as shown in the basic type editor in Figure 8.14.

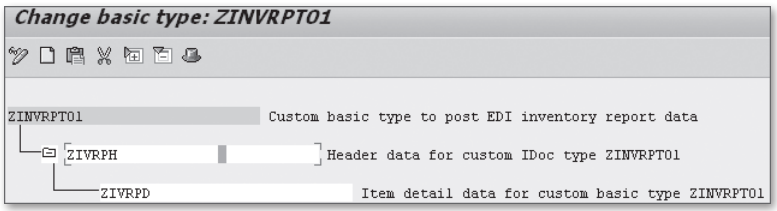


Figure 8.14 Our New IDoc Basic Type ZINVRPT01

8. Double-click ZIVRPD to view the MAINTAIN ATTRIBUTES dialog. The parent segment is ZIVRPH, and the hierarchy level is 2.
9. Save the basic type, and assign it to a package and a change request. Back out of the edit window. Release the IDoc basic type by selecting menu option EDIT • SET RELEASE.

Create a Custom Message Type

Next we create the logical message. Use Transaction WE81, and click DISPLAY CHANGE (menu option TABLE VIEW • DISPLAY -> CHANGE). Click NEW ENTRIES (menu option EDIT • NEW ENTRIES) to open the OVERVIEW OF ADDED ENTRIES screen.

Enter "ZINVRPT" in MESSAGE TYPE and a description of the message in the SHORT TEXT field, as shown in Figure 8.15. Save the message type and assign it to a customizing request.

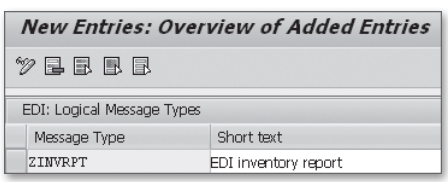


Figure 8.15 Create a Custom Message Type for the IDoc

Link Message to Basic Type

Now we link the logical message to the basic type with Transaction WE82. Click Display Change (menu option TABLE VIEW • DISPLAY -> CHANGE) and then click New Entries (menu option EDIT • NEW ENTRIES) to open the OVERVIEW OF ADDED ENTRIES screen.

Enter the following values into the table control, as illustrated in Figure 8.16:

- Enter "ZINVRPT" in the Message type field.
- Enter "ZINVRPT01" in the Basic type field.
- Enter version "702" (your current SAP system release) in the Release field.

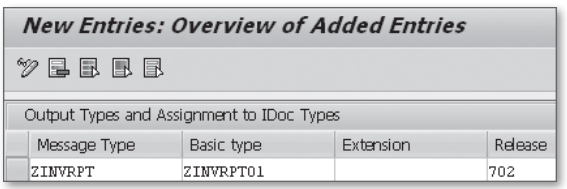


Figure 8.16 Linking the Message Type to the IDoc Basic Type

Save the entry and assign it to a customizing request.

Create the Function Group

We'll create the function group in the object navigator.

1. Run Transaction SE80 and click Edit Object at the top of the Object Navigator window.
2. Select the Function group tab, enter "ZEDINVRP" in the Group name field, and click Create.
3. The Create Function Group dialog opens. Enter a short description in the Short text field, and click Save.
4. Assign the function group to a package and a change request.

Function group `ZEDINVRP` is now ready for coding. It should look like Figure 8.17 in the object navigator.

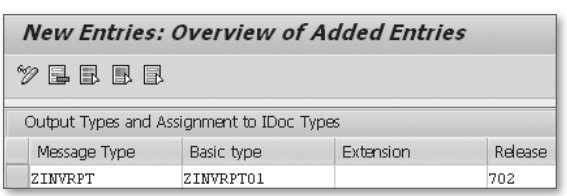


Figure 8.17 Function Group ZEDINVRP is Primed for Coding

Coding the Inbound IDoc Processing Function

We're still in the object navigator:

1. Click Edit Object, select Function module in the Function Group tab, and enter the function name "ZIDOC_INPUT_ZINVRPT".
2. Click Create. The Create Function Module dialog opens.
3. The Import parameters screen of the Function Builder opens. We'll use standard import parameters that will be passed to the function by the IDoc interface at runtime.
4. Enter the following values illustrated in Figure 8.18.

Hint

Copy and paste standard values for all parameter tabs from any standard inbound processing IDoc function.

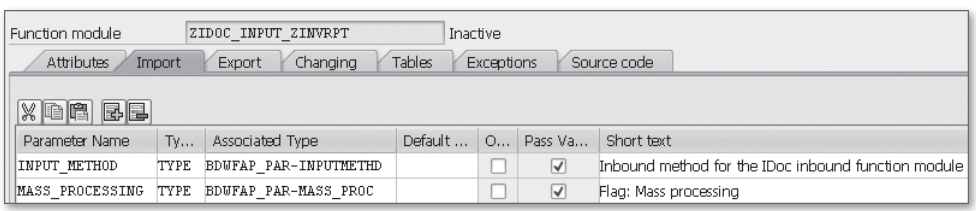


Figure 8.18 Standard Import Parameters for Inbound IDoc Functions

5. Export parameters return workflow and other information to the IDoc interface at runtime. You can also return document numbers and other application data for reporting.

   Click the Export tab and enter the values illustrated in Figure 8.19.

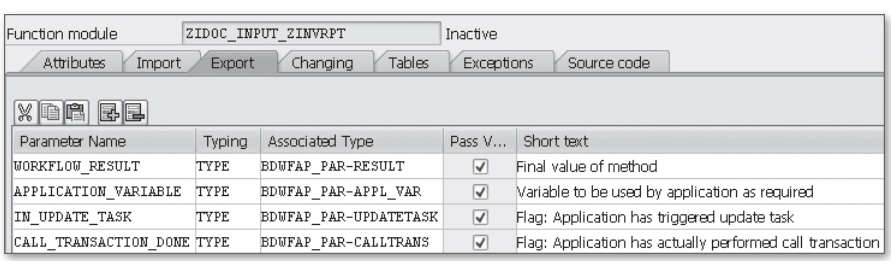


Figure 8.19 Standard Export Parameters for Inbound IDoc Functions

6. Click the Tables tab and enter the values for the internal tables listed in Figure 8.20.

   The tables will pass data in and out of the function. We're most interested in the IDoc control, data, and status records. We won't add any exceptions for this example.

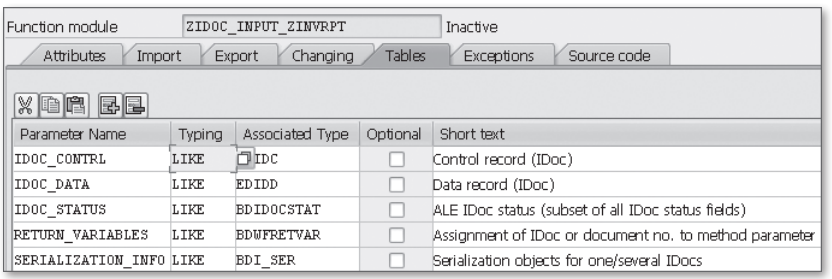


Figure 8.20 Standard Table Parameters for Inbound IDoc Functions

7. Click the Source code tab and enter the function's code as it appears in Listing 8.1.

```
*"----------------------------------------------------------
FUNCTION ZIDOC_INPUT_ZINVRPT.
*"----------------------------------------------------------
*"*"Local Interface:
*"  IMPORTING
*"     VALUE(INPUT_METHOD) TYPE  BDWFAP_PAR-INPUTMETHD
*"     VALUE(MASS_PROCESSING) TYPE  BDWFAP_PAR-MASS_PROC
*"  EXPORTING
*"     VALUE(WORKFLOW_RESULT) TYPE  BDWFAP_PAR-RESULT
*"     VALUE(APPLICATION_VARIABLE) TYPE  BDWFAP_PAR-APPL_VAR
*"     VALUE(IN_UPDATE_TASK) TYPE  BDWFAP_PAR-UPDATETASK
*"     VALUE(CALL_TRANSACTION_DONE) TYPE BDWFAP_PAR-CALLTRANS
*"  TABLES
*"     IDOC_CONTRL STRUCTURE  EDIDC
*"     IDOC_DATA STRUCTURE  EDIDD
*"     IDOC_STATUS STRUCTURE  BDIDOCSTAT
*"     RETURN_VARIABLES STRUCTURE  BDWFRETVAR
*"     SERIALIZATION_INFO STRUCTURE  BDI_SER
*"----------------------------------------------------------
*Data declarations
data: izedinvrpt type standard table of
         zedinvrpt with header line.
```

```
data: gs_zivrph type zivrph,
      gs_zivrpd type zivrpd.
data: gs_last_no like zedinvrpt-invrptno.
*Get last used ID number
clear gs_last_no.
select max( invrptno ) into gs_last_no
    from zedinvrpt.
gs_last_no = gs_last_no + 1.
*process IDoc records and format insert for izedinvrpt
loop at idoc_contrl.
  refresh izedinvrpt. clear izedinvrpt.
  loop at idoc_data where docnum = idoc_contrl-docnum.
    case idoc_data-segnam.
*process header record
      when 'zivrph'.
        gs_zivrph = idoc_data-sdata.
*process detail record
      when 'zivrpd'.
        gs_zivrpd = idoc_data-sdata.
        izedinvrpt-mandt = sy-mandt.
        izedinvrpt-invrptno = gs_last_no.
        izedinvrpt-matnr = gs_zivrpd-matnr.
        izedinvrpt-werks = gs_zivrpd-werks.
        izedinvrpt-lgort = gs_zivrpd-lgort.
        izedinvrpt-menge = gs_zivrpd-menge.
        izedinvrpt-meins = gs_zivrpd-meins.
        izedinvrpt-credat = gs_zivrph-credat.
        izedinvrpt-cretim = gs_zivrph-cretim.
        izedinvrpt-upddat = sy-datum.
        izedinvrpt-updtim = sy-uzeit.
        append izedinvrpt.
        gs_last_no = gs_last_no + 1.
    endcase.
  endloop.
*insert IDoc records to table zedinvrpt
  modify zedinvrpt from table izedinvrpt.
  if sy-subrc = 0.
*success message to status record
    clear idoc_status.
    idoc_status-docnum = idoc_contrl-docnum.
    idoc_status-msgty  = 's'.
    idoc_status-msgid  = 'ZEDI01'.
    idoc_status-msgno  = '001'.
```

```
      idoc_status-status = '53'.
      append idoc_status.
    else.
*verify record count and total before updating.
      clear idoc_status.
      idoc_status-docnum = idoc_contrl-docnum.
      idoc_status-msgty  = 'e'.
      idoc_status-msgid  = 'ZEDI01'.
      idoc_status-msgno  = '002'.
      idoc_status-status = '51'.
      append idoc_status.
    endif.
  endloop.

  ENDFUNCTION.
```

Listing 8.1 Source Code for Inventory Report IDoc Function

This code is stripped down to its essentials. The control and data records are imported into the function at runtime through the `IDOC_CNTRL` and `IDOC_DATA` internal tables. The program logic follows:

1. Gets the last used invoice report ID from `ZEDINVRPT-INVRPTNO` and increments it for use by the incoming records.
2. Loops through `IDOC_CNTRL`.
3. Loops through `IDOC_DATA` at the current IDoc number.
4. Evaluates the `SEGNAM` field for the current segment name.
5. Moves the `SDATA` field in the data record to a string structured by the segment type.
6. Moves data from the structured string to an internal table structured by our target database table ZEDINVRPT.
7. When the loops on `IDOC_DATA` and `IDOC_CNTRL` are done and internal table `IZED-INVRPT` is fully populated with all IDoc data, then database table ZEDINVRPT is updated from the internal table.
8. If the update succeeds, the status record is updated with status 53, and a success message pulled from custom message class `ZEDI01`.
9. If the update fails, the status record is updated with status 51, and an error message pulled from custom message class `ZEDI01`.

10. Status and message values are passed to internal table `IDOC_STATUS` and returned to the calling IDoc interface function `IDOC_INPUT`, which passes them to routines that update the status records in the database.

All inbound IDoc processing functions work in essentially the same way, with varying levels of complexity for data processing and checks depending on transactional requirements. If you understand this processing approach, you'll understand a lot about the IDoc interface.

Now that the code is written, we'll step through the configuration we need to plug all of our custom objects into the standard IDoc interface.

Link the Function to Message and Basic Type

First up is to link the custom function to our logical message and basic type.

1. Go to Transaction WE57, and click Display -> Change (menu path Table View • Display -> Change)
2. Click New Entries (or press F5) to open the Details of Added Entries screen and enter the following values into it:
   - Enter "ZIDOC_INPUT_ZINVRPT" into the Function Module field.
   - Select Function module from the Function Type dropdown.
   - Enter "ZINVRPT01" in the Basic type field.
   - Enter "ZINVRPT" in the Message Type field.
   - Select Inbound from Direction dropdown.

Click Save and assign the changes to a customizing request. The screen should look like Figure 8.21 after you're done.

Set Attributes for the Function

To set attributes for the function, follow these steps:

1. Go to Transaction BD51 and click New Entries.
2. Enter "ZIDOC_INPUT_ZINVRPT" in the Function module (inbound) field and "0" in Input t. column for direct input. The function directly inserts IDoc data into the custom table.

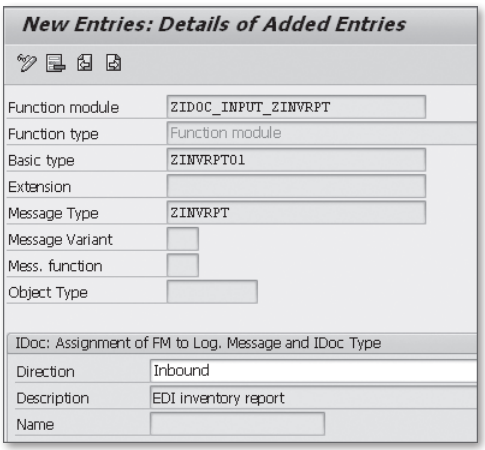


Figure 8.21 Linking the Function to the IDoc Basic and Message Types

3. Save and assign the attributes to a transport. The screen should look like Figure 8.22.

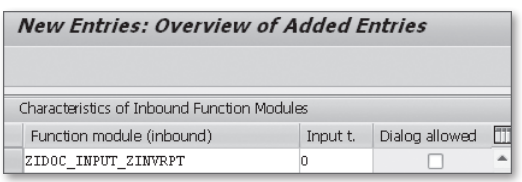


Figure 8.22 Function Attributes for Direct Input

Create a Custom Process Code

The process code ties all the processing pieces together.

1. Go to Transaction WE42, and switch to change mode.
2. Click New Entries to open the Details of Added Entries screen, and do the following:
   - Enter "ZINRP" in the Process code field.
   - Enter a text description in the Description field. Begin the description with the message name.
   - Set Processing type as the function module.
3. Save the entry and assign it to a transport.

4. A more detailed view of the Added Entries screen opens. Select the function "ZIDOC_INPUT_ZINVRPT" from the Function Module dropdown list.
5. Save and add to a transport. After the function has been assigned, back out via F3 to the added entries screen. The function name will be linked to the process code. The screen should look like Figure 8.23.

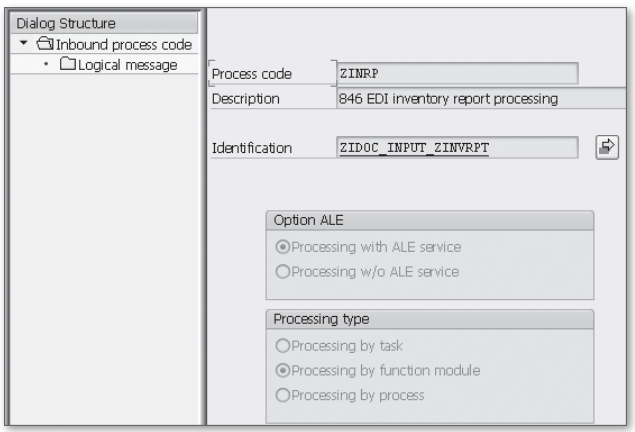


Figure 8.23 The Process Code and Function Module Linked

6. Double-click the Logical message folder in the Dialog Structure navigation display to open the Logical message overview screen. Click New Entries and enter "ZINVRPT" into the Message type field, as shown in Figure 8.24.

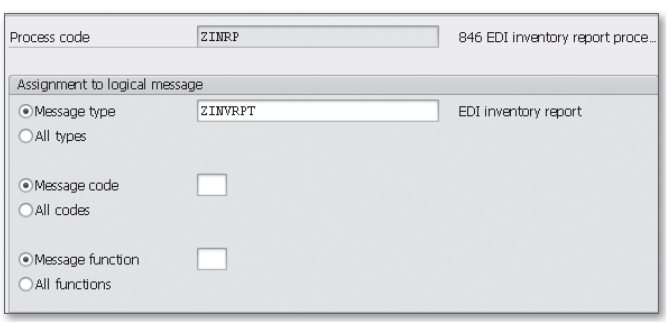


Figure 8.24 Link the Logical Message to the Process Code

Don't forget to save. This completes the link between the process code, the logical message, and the custom function module. All these objects are now ready to be used in a partner profile.

Define the Partner Profile

All that's left is the inbound partner profile. The inventory report is coming from the supplier Disc Services International, entered in Acme's SAP vendor master as DISK01.

So we'll create a partner profile with partner type LI for vendor.

1. Run Transaction WE20, select the Partner Type LI folder, and create a vendor partner profile header for DSI with partner number DISK01.
2. Save the general view and create inbound parameters to add message type ZINVRPT. Enter the following values into the Inbound parameters screen:
   - Enter "VN" in the Partner Role field.
   - Enter "ZINVRPT" in the Message type field.
   - Enter "ZINRP" in the Process code field.
   - Set Trigger Immediately as the processing mode. This is a simple table insert that comes in once a day and doesn't involve a lot of processing, so we won't schedule a background job.
3. Save the partner profile. The finished product should look like the inbound partner profile in Figure 8.25.

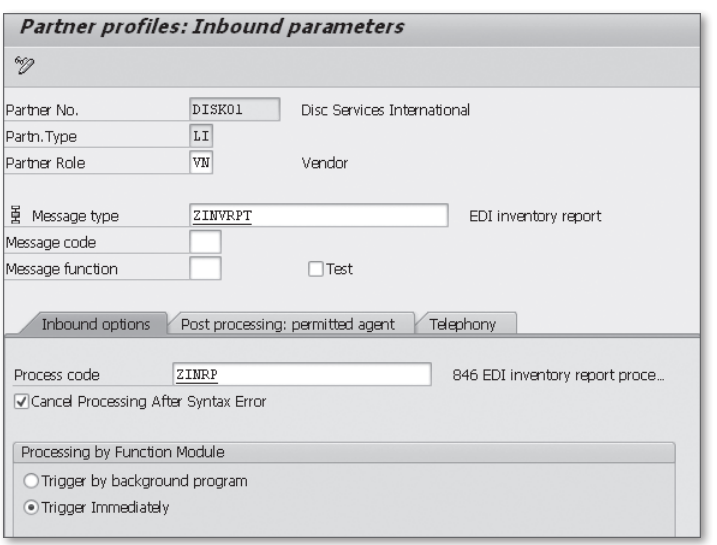


Figure 8.25 Inbound Parameters for DSI Message Type ZINVRP

This interface is now ready to roll—after it goes through the full testing cycle, of course.

8.3 Extending an IDoc: Outbound PO with BOMs

Next we'll try our hand at adding a custom segment to a standard IDoc. Our scenario is pretty simple. DSI needs to receive the bill of materials when Acme sends them a purchase order. They need to know what components Acme expects them to use in manufacturing.

DSI may have the components in inventory or they may need to order from a third-party supplier. The purchase order BOM is key to their manufacturing process.

A custom segment, Z1EDP01, will be created to hold the components just below E1EDP01, the parent for all line item segments, in an extension of IDoc basic type ORDERS05 that we will name ZORDRS01.

The code will be written in a CMOD modification project using enhancement SDEDI001 in component `EXIT_SAPLVEDC_002`.

8.3.1 IDoc Outbound Development Workflow

Extended IDocs are built on existing standard basic types. They are created to send or receive data that are not accommodated in a standard IDoc. Figure 8.26 outlines the workflow for extending an IDoc, which is broken into three main steps: extend the IDoc, code the IDoc enhancement, and configure the interfaces.

To create an extended IDoc type, you insert one or more segments into a standard basic type. The insertion point should make sense in terms of the data the extended segment contains and its semantic context within the IDoc.

We also need custom code to populate and process the extended segments. The code is usually written in a user exit, but sometimes you need a custom IDoc function. For example, message type MBGMCR doesn't have any customer exits, so it would need a custom version of its processing function if it were extended or if you needed to do any non-standard processing.

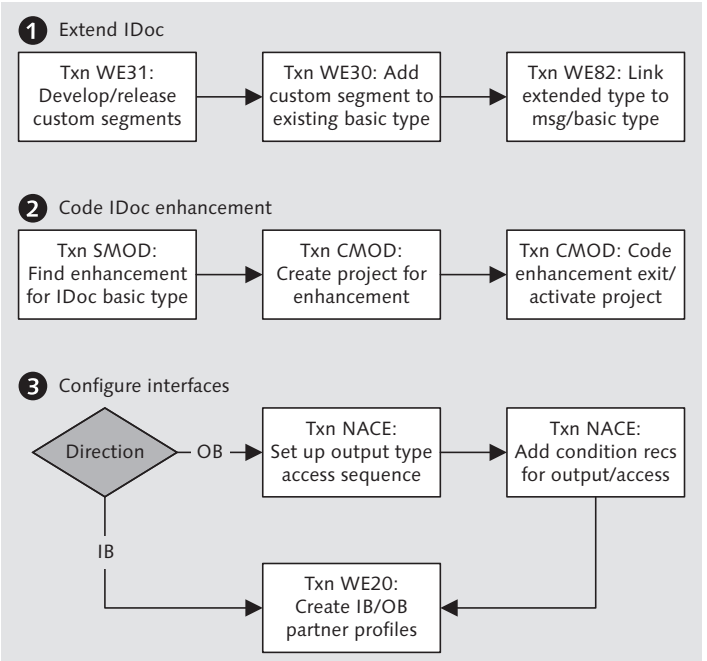


Figure 8.26 Basic Workflow for Extending IDocs

The process for creating and configuring a custom function for an extended IDoc type is the same as for a custom IDoc with the exception that we don't need to create a new message type.

To extend an IDoc by coding a customer exit, follow these steps:

1. Create custom segments in the segment editor with Transaction WE31.
2. Copy an existing standard basic type into an extended type in the IDoc type editor with Transaction WE30.
3. Insert custom segments into the extended type with Transaction WE30.
4. Link the extended type to the logical message and basic type with Transaction WE82.
5. Identify an enhancement with user exits for the extended type with Transaction SMOD.
6. Create a modification project to write and manage the custom code for the exit with Transaction CMOD.

7. Code the customer exit enhancement component in the CMOD project.
8. For an inbound extended IDoc, create inbound partner profile for the customer, message, and process code with Transaction WE20.
9. For an outbound extended IDoc, there are two additional steps:
   - Set up message control (output type, access sequence, and condition record for the output and access with Transaction NACE).
   - Create outbound partner profile for the customer, message, XML file port, basic and extended types, and message control with Transaction WE20.

Identifying Customer Exits

Let's root around the Data Dictionary and consider a simple backend way to identify user exits that we can use in modification projects. First we need to look at the standard frontend approach.

Modification projects organize our user exit work. We can add more than one enhancement to a project allowing it to encompass an entire business process. But at Acme, we'll only create one project per enhancement.

The following process is the typical frontend approach to finding exits:

1. Identify the function for the IDoc that will be extended.
2. Go to the Function Builder in Transaction SE37, enter the function name in the Function module field, and click Display.
3. Click Attributes to get the package name: VED for all SD EDI development objects and ME for purchasing documents, including EDI and ALE exits.
4. Go to Transaction SMOD, select the input help dropdown (or press F4) and click Information System to open the Repository Info System: Find Exits dialog.
5. Enter the package name in the Package field of the search help dialog and click Execute.
6. The Repository Info System: Find Exits dialog opens, listing all enhancements in the package.

   For SD package VED, nine hits are returned; for purchasing package ME, 36 hits are returned in ERP 6.0, as shown in Figure 8.27.

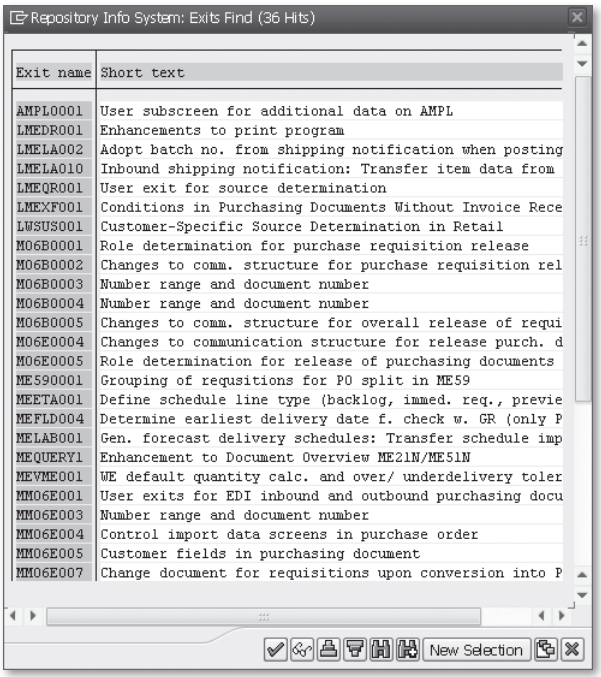


Figure 8.27 SMOD Search Help Results for Package ME

Descriptions of enhancements don't always reveal their purpose or identify their message types. For example, if you search package ME for user exits in PO message ORDERS, you'd have to carefully scan the list of 36 enhancements to learn that MM06E001 is the one you need.

We can use the Data Dictionary backend to refine our search. It takes a couple of steps however. First replicate the SMOD search help results by following these steps:

1. Go to the Data Browser with Transaction SE16.
2. Enter "TADIR" into the TABLE NAME field and press Enter. TADIR contains all development repository objects in the system.
3. Enter the following values into the selection screen for TADIR.
   - Enter "R3TR" in the PGMID field.
   - Enter "SMOD" in the OBJECT field.
   - Enter "ME" in the DEVCLASS field.
4. Click EXECUTE or press F8.

A list of every enhancement stored in package VED is returned in the field OBJNAME. So what do we need to know to make our job of identifying exits easier?

Focusing on Enhancements

Each enhancement contains all the user exits available to any IDoc function in components that hold a function module with the naming convention `EXIT_<PROGRAM>_00X` where:

- `PROGRAM` is the name of the function pool program that contains the IDoc processing function module.
- `00X` is the number of the function within the exit function group.

For example, enhancement SIDOC001, which is used to change the control segment just before an IDoc is written to the database, has only one component: function `EXIT_SAPLEDI1_001`.

Enhancement MM06E001, with purchasing document exits, has 20 components; they are all called during various processing stages of more than one message type, including outbound purchase orders, inbound purchase order change and acknowledgments, and inbound ship notifications.

For processing data in the outbound PO, the key components include the following:

1. `EXIT_SAPLEINM_001`: Control record changes
2. `EXIT_SAPLEINM_002`: Data record changes while IDoc is being built
3. `EXIT_SAPLEINM_011`: Final data changes after the IDoc has been built

Note that `EXIT_SAPLEINM_011` is also called for outbound message types ORDCHG (PO change), REQOTE (request for quotation), BLAORD (purchasing contracts), and BLAOCH (purchasing contract change).

The exit names are in function `IDOC_OUTPUT_ORDERS`. You can find them by going into the code of the function in Transaction SE37 and searching the string `CUSTOMER-FUNCTION` using the binocular FIND icon at the top of the FUNCTION BUILDER screen. Select IN MAIN PROGRAM.

Every call to a customer function in every message type processed by the main program is listed. The call syntax is `CALL CUSTOMER-FUNCTION '00X'` where `00X` is

the number of the exit being called. We're interested in `CUSTOMER-FUNCTION '002'` for `EXIT_SAPLEINM_011`.

Click the instance of the customer function call in the Global Search in Programs window, and the system will go to the call point in the code. The customer function is called after each segment has been appended to `INT_EDIDD`, the internal table used to build the IDoc.

The `'002'` in single quotes following `CALL CUSTOMER-FUNCTION` is the function name. Double-click `'002'` within the single quotes and the system navigates to the exit function. The source code for the function is listed in Listing 8.2.

```
*"----------------------------------------------------------
FUNCTION EXIT_SAPLEINM_002.
*"----------------------------------------------------------
*"*"Global Interface:
*"  IMPORTING
*"     VALUE(XEKKO) LIKE EKKO STRUCTURE  EKKO
*"     VALUE(XLFA1) LIKE LFA1 STRUCTURE  LFA1
*"     VALUE(XLFB1) LIKE LFB1 STRUCTURE  LFB1
*"     VALUE(DOBJECT) LIKE NAST STRUCTURE  NAST OPTIONAL
*"  TABLES
*"     INT_EDIDD STRUCTURE EDIDD
*"     XEKPO STRUCTURE UEKPO OPTIONAL
*"     XEKET STRUCTURE UEKET OPTIONAL
*"     DEKEK_X STRUCTURE EKEK_X OPTIONAL
*"     DEKEH STRUCTURE IEKEH OPTIONAL
*"     DSADR STRUCTURE SADR OPTIONAL
*"     DVBAK STRUCTURE MMVBAK OPTIONAL
*"     DVBAP STRUCTURE MMVBAP OPTIONAL
*"     DVBKD STRUCTURE MMVBKD OPTIONAL
*"  CHANGING
*"     VALUE(ISC_ENHANCEMENT) TYPE ISC_EXIT_SAPLEINM_002 OPTIONAL
*"  EXCEPTIONS
*"     ERROR_MESSAGE_RECEIVED
*"     DATA_NOT_RELEVANT_FOR_SENDING
*"----------------------------------------------------------

  INCLUDE ZXM06U02.

ENDFUNCTION.
```

Listing 8.2 Call to Enhancement Component for Outbound PO

The IMPORTING parameters of the exit are identical to the EXPORTING parameters of the customer function, which is a shell that calls the exit.

Note include program ZXM06U02, which is where the exit code goes. The program doesn't exist until the system creates it when we double-click its name. We can write our code here through the IDoc function, but it is a best practice to code and manage exits in a modification project.

Shortcut to Identifying Enhancements

Let's go back to the Attributes screen of IDOC_OUTPUT_ORDERS in Transaction SE37. There are values here that can help us identify our enhancement in the Data Dictionary. Copy the program name SAPLEINM. We can use it to identify which enhancement we need through a table read.

1. Go back to the Data Browser with Transaction SE16.
2. Enter "MODSAP" in the Table Name field, and press Enter. MODSAP stores SAP enhancements and their components.
3. Enter the program name bracketed by asterisks (*SAPLEINM*) in the Member field. Or, better yet, get the exit name from the IDoc processing function (EXIT_SAPLEINM_002). Click Execute.

   The enhancement MM06E001 is in the Name field.

Between the IDoc processing function and table MODSAP, you can quickly identify the enhancement you need to add to your modification project.

Exit Function Groups

The names of exit function groups begin with an X, which can narrow a search for exit functions in Transaction SE37. An open-ended search for function groups that begin with X in ECC 6.0 returns 5,899 exits for all applications.

Function group XM06 contains all 61 user exits for package ME, which contains all development objects for purchasing.

One of the joys of working with SAP is that a little educated poking around goes a long way. And it's fun, too!

It's time to build our extended IDoc. We'll begin with the custom segment.

8.3.2 Create Segment Z1EDP01

The structure of custom segment Z1EDP01 is described in Table 8.5.

Pos	Field	Data Element	Description
01	MATNR	MATNR	SAP component material number
02	MAKTX	MAKTX	Component description

Table 8.5 Structure of BOM Segment Z1EDP01

Build the custom segment in the segment editor with Transaction WE31. Enter "Z1EDP01" in the NAME field and click CREATE to open the CREATE SEGMENT DEFINITION screen.

Enter a description in the SHORT DESCRIPTION field. Add the field and data elements from Table 8.5 into the FIELD NAME and DATA ELEMENT fields in the table control.

Click SAVE and assign the segment to a change request. Press F3 to back out to the segment editor's initial screen. Release the segment by selecting menu option Edit • SET RELEASE.

8.3.3 Build Extension ZORDRS01

Next we extend the standard basic type:

1. Go to the IDoc type editor with Transaction WE30. Enter the name of the IDoc Extension ("ZORDRS01") in the OBJ. NAME field, select EXTENSION, and click CREATE. The CREATE EXTENSION DIALOG opens, as shown in Figure 8.28.
2. Select CREATE NEW and enter "ORDERS05" in the LINKED BASIC TYPE field. Click OK to open the IDoc type editor.

   The structure of basic type ORDERS05 is displayed in the editor with the root name ZORDRS01.
3. Expand the E1EDP01 item group by clicking on the folder icon next to the segment name.
4. To add the custom segment, put the cursor on the E1EDP01 segment name and click CREATE SEGMENT, or press Shift+F6, or choose menu option EDIT • CREATE SEGMENT. A pop-up informs you that the custom segment will be added as a child to E1EDP01. Click OK.

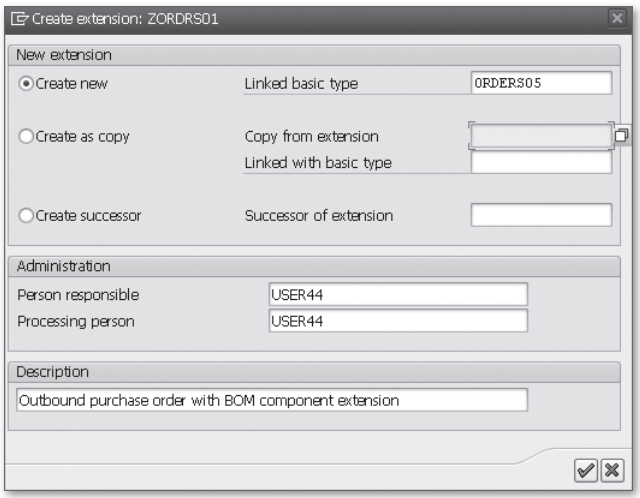


Figure 8.28 Select a Linked IDoc Basic Type to Create an Extension

5. The Maintain Attributes dialog opens. Enter the following values to add the Z1EDP01 segment:
   - Enter "Z1EDP01" in the Segm.type field.
   - Enter "1" in the Minimum number field.
   - Enter "999" in the Maximum number field.
6. Click OK. Z1EDP01 appears as a child of E1EDP01 (Figure 8.29). Extended type ZORDRS01 now has its custom segment.

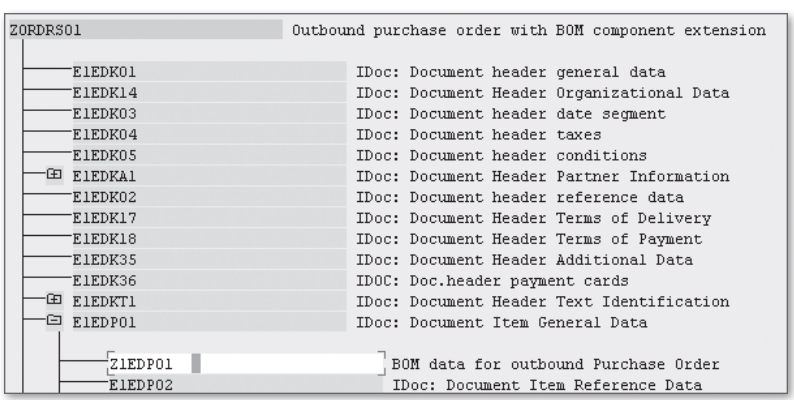


Figure 8.29 Z1EDP01 Is a Child of E1EDP01

7. Save the extension, assign it to a change request, and back out of the edit window.
8. Release the extended type in the initial screen by selecting menu option Edit • Set Release.

8.3.4 Link Message to Basic and Extended Types

To link the message and basic type to the extended type, follow these steps:

1. Go to Transaction WE82, and click Change • Display, press Ctrl+F4, or follow menu path Table View • Display -> Change.
2. Click New Entries (or press F5) to open the Overview of Added Entries screen. Enter the following values in the table control (see Figure 8.30):
   - Enter "ORDERS" into the Message type field.
   - Enter "ORDERS05" into the Basic type field.
   - Enter "ZORDRS01" into the Extension field.
   - Enter version "702" in the Release field.

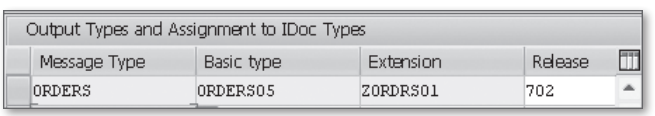


Figure 8.30 Link between the Extension and the Message and Basic Types

3. Save the entry and assign it to a transport request.

8.3.5 Create the Modification Project

Next we'll create the modification project with Transaction CMOD, as shown in Figure 7.35.

1. Enter the project name "ZEDIMPO1" in the Project field and click Create. The project attributes screen opens. Enter a description in the Short text field (see Figure 8.31).

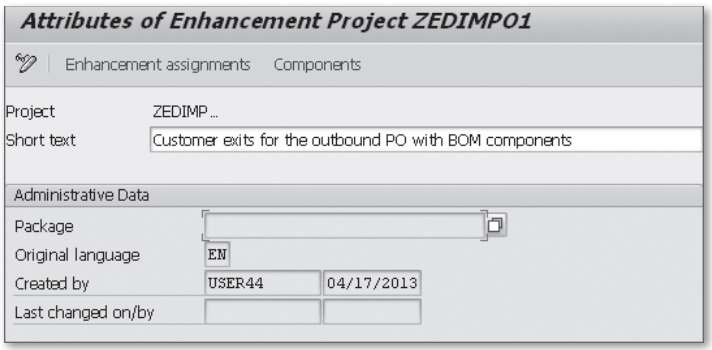


Figure 8.31 Modification Project ZEDIMP01 Attributes

2. Save the project and assign it to a change request.
3. Click the ENHANCEMENT ASSIGNMENTS button. Enter "MM06E001" into the ENHANCEMENT field (see Figure 8.32). Save.

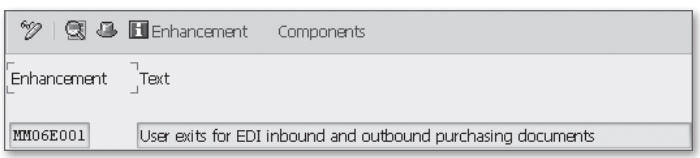


Figure 8.32 Our User Exit Is in Enhancement MM06E001

4. Click COMPONENTS to load a list of all exit functions assigned to the enhancements. A partial list is displayed in Figure 8.33.

   The checkmark next to the enhancement means that there is already active code in the exit.

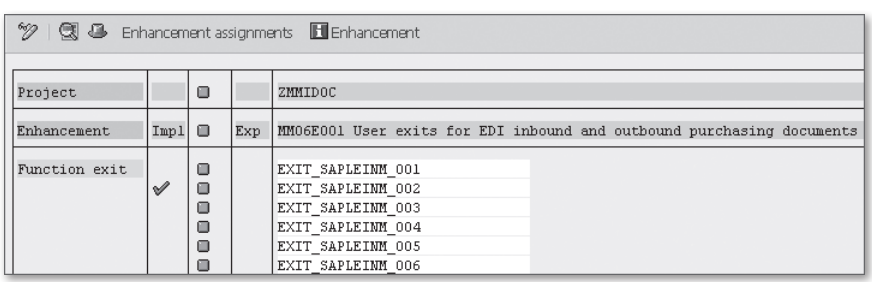


Figure 8.33 Exit Functions Are Components of Enhancements

5. We'll write the code in component `EXIT_SAPLEINM_002`. Double-click the exit to open the source code window of the Function Builder.

   Note the import and table parameters of the exit function in Figure 8.34. We'll read our inputs from the application tables and append our new segment to `INT_EDIDD` when we write our code.

   `INT_EDIDD` is used to assemble the IDoc segments in the order that they will appear in the finished IDoc.

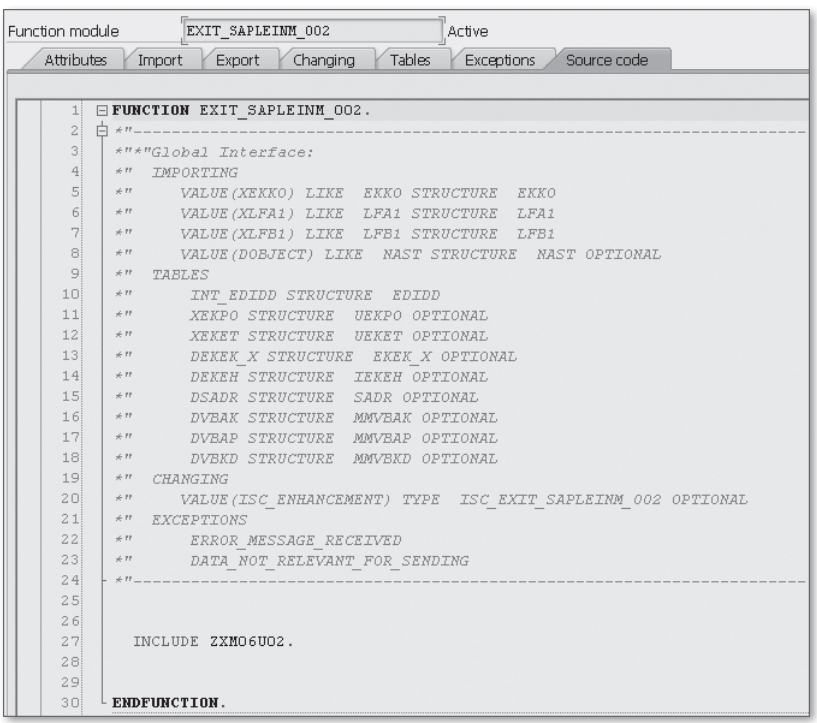


Figure 8.34 Exit Parameters in the Source Code of the Customer Exit

We first need to create the include program `ZXM06U02`, if it doesn't already exist. When you double-click the include name, SAP returns the following information message:

Program names ZX… are reserved for includes of exit function groups.

Press Enter to get past the message and open the CREATE OBJECT dialog informing you of the following:

Include ZSM06U02 does not exist. Create object?

Click Yes and assign the include program to a change request. When the ABAP editor loads, we're ready to code.

8.3.6 Coding the Exit

The code for our exit will be called after each instance of segment E1EDP01 is processed and added to the IDoc.

The exit will read the BOM for the purchase order line item material from table MDSB, which stores the material number for the item components against the PO and line item numbers. We can then get the description for each component from table MAKT.

The material number and description for each component will be passed to a separate instance of our custom segment Z1EDP01 and appended to internal table `INT_EDIDD`, which is used to build the IDoc.

The data that we need to identify the BOMs for the finished DVDs in the PO are in the table parameters of `EXIT_SAPLEINM_002`.

The IDoc is built from data in a number of structures and internal tables, including XEKKO (purchase order header) and XEKPO (line item details), which will give us the PO number and line item number to read MDSB. The IDoc build for each segment—and the customer function call for each segment—is in form `FUELLEN_IDOC_INTTAB`.

All line item segments are built within a loop on XEKPO. E1EDP01, the parent segment at the item level, is the first appended to the IDoc within the XEKPO loop. Our code will be called by the exit after E1EDP01 is written to internal table `INT_EDIDD`.

The same exit is called for every segment after it has been written to the IDoc, regardless of level. So we need to check that the current segment name is E1EDP01. This is in field SEGNAM in internal table `INT_EDIDD`.

In addition, we're only doing this for DSI. So we also need to check that the PO is for vendor DISK01. This value will be available to the exit in the import parameter `LFA1-LIFNR`.

This, and the custom segment name, is all we need to get the BOM and append it to the IDoc as a child of E1EDP01. The code is shown in Listing 8.3. Once again,

remember that the code is stripped down to its essentials and is really only a starting point for this solution.

```
*&---------------------------------------------------------*
*&  Include       ZXM06U02                                 *
*&---------------------------------------------------------*
** Type declarations                                       *
** Component material numbers from MDSB
TYPES: BEGIN OF t_bomrec,
        matnr type matnr,
      END OF t_bomrec.
** Internal table declaration                             *
** BOM data for custom segment data:
it_pobom type table of t_bomrec.
** Data Declarations                                      *
** Work areas
data: wa_pobom like line of it_pobom,
      ls_z1edp01 type Z1EDP01,
      ls_maktx type maktx.
** Begin processing                                       *
** Restrict processing to segment E1EDP01 and vendor DSI
if xlfa1-lifnr = 'DISK01' and int_edidd-segnam = 'E1EDP01'.
** Get all BOM components and descriptions for finished good
** in current purchase order item.
  select matnr into corresponding fields
        of table it_pobom from mdsb
        where ebeln = xekpo-ebeln
          and ebelp = xekpo-ebelp.
** Get material description by looping on it_pobom and
** reading MAKT by material number
  loop at it_pobom into wa_pobom.
    clear ls_maktx.
    select single maktx into ls_maktx
      from makt where matnr = wa_pobom-matnr.
    if sy-subrc = 0.
** Populate custom segment string
      ls_z1edp01-matnr = wa_pobom-matnr.
      ls_z1edp01-maktx = ls_maktx.
    endif.
** Pass and append data to INT_EDIDD
    int_edidd-segnam = 'Z1EDP01'.
    int_edidd-sdata = ls_z1edp01.
```

```
    append int_edidd.
  endloop.
endif.
```

Listing 8.3 Exit Code Passes BOM Components to Segment Z1EDP01

8.3.7 Customize Message Control

Now we move on to outbound configuration: message control and a partner profile. We'll need a custom output type so that we can enter unique values into the message control screen of our outbound partner profile.

Message type ORDERS will be used in more than one outbound interface for DSI and will require more than one outbound partner profile. We need a unique output type for our BOM ORDERS interface that we will not use in any other partner profile.

Each partner profile must be unique, as we have seen from our discussions of partner profile and message control keys in tables EDP13 and EDP12 in Chapter 7, Section 7.2, Outbound Configuration Generates IDocs.

To create our custom output type ZNEU, we'll copy standard type NEU in application EF (purchase order). We'll only use this custom type to generate an outbound ORDERS IDoc from a PO with the BOM extension.

1. Call Transaction NACE.
2. Select application EF (purchase order) and click Output types to open the Output Types: Overview in display mode.
3. Select the menu path Table View • Display • Change (or press Ctrl+F1) and then select standard output NEU. Click Copy as or press F6.
4. Change the name of the output type to "ZNEU" and enter a description for the extended PO with BOM. Make sure the Access to conditions and Multiple issuing checkboxes are both set, as in Figure 8.35.
5. Access sequence 0001 (document type, purchasing organization, and vendor) gives us enough granularity to restrict who gets a PO with BOMs from which purchase order.

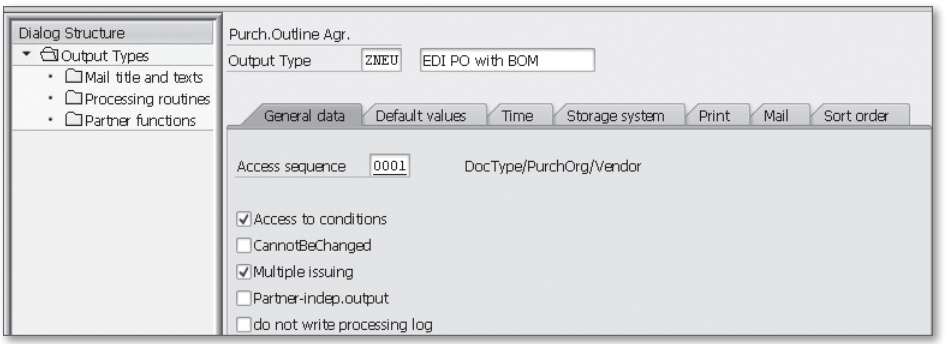


Figure 8.35 CustomOutput Type ZNEU with Access Sequence 0001

6. We'll use a custom PO document type (ZNB) copied for standard PO document type NB to make sure that only some POs are sent to DSI with an extended BOM segment. We'll use ZNB to create purchase orders that will be sent to DSI with the BOM in the extended IDoc type.

Creating a Custom Purchasing Document Type

Not all purchase order IDocs will be sent with a custom BOM segment. We need to distinguish between orders that will generate a BOM segment and those that won't. To do this, we'll create a custom purchase order document type and use this in our condition record.

Functional consultants and business users define custom document types after thorough analysis of how these documents will be used by the business and what data they need to contain in their different use cases.

It's not really the job of the EDI team, but it doesn't hurt to know.

Purchasing document types are configured in the IMG (Transaction SPRO). We'll take the easy road and copy the existing standard PO document type to a custom code.

1. In the IMG, follow menu path SAP Customizing Implementation Guide • Materials Management • Purchasing • Purchase Order • Define Document Types.
2. In the table control of the Document Types Purchase Order Change screen, find and select document type NB (standard PO).
3. Click the Copy As button (or press F6).
4. Change the document type code to ZNB and add a short description.
5. Press Enter. The Specify object to be copied dialog opens with the message that the document type has dependent entries.
6. Click Copy all to get an exact copy of the standard document type.
7. Save and assign the custom document type to a transport request.

Our custom purchasing document type can now be used to create purchase orders for whoever needs an EDI PO with the BOM attached.

7. We will select access number 10 in access sequence 0001, which points to condition table 25 (B025) and communications structure KOMKBEA with key fields:
   - BSART: Purchasing document type
   - EKORG: Purchasing organization
   - LIFNR: Vendor
8. Press Enter. The Specify object to be copied dialog opens with the observation that the output type has dependent entries. Click Copy all. Another dialog opens with the number of dependent entries.
9. The system returns us to the Output types: Overview screen in change view. Select output ZNEU, and double-click the Processing routines folder to open the Processing routines: Overview screen.
10. If there is no program name for medium EDI, add one. Click New entries to open the Details of Added Entries screen.
11. Enter the following values into the added entries details screen:
    - Enter "EDI" in the Transm.Medium field.
    - Enter "RSNASTED" in the Program (Processing 1) field; this is the standard SAP output program. You can also use a custom ABAP output program. Use `RSNASTED` as a model.
    - Enter "EDI_PROCESSING" in the Form Routine field. This form identifies and calls the function that will build the IDoc, writes it to the database, sends it to the outbound IDoc processing function, and updates its status record.
12. Press Enter and double-click the Partner functions folder to open the Partner Functions: Overview screen.
13. Make sure that there's an entry for medium EDI and partner type VN for vendor. If not, click New entries and select EDI in the Medium field and VN in the Funct field.
14. Save and assign the new output to a change request. You may need to cycle through several objects to complete the assignment to the request.

Assign ZNEU to a Procedure

1. Back out to the initial output control screen in Transaction NACE. Select application EF and click the PROCEDURES button.
2. There's only one procedure in application EF, and that is RMBEF1. We'll need to assign our custom output type to it.
3. Select the procedure and double-click the CONTROL folder in the navigation pane.
4. Select output type NEU and click the COPY AS button (or press F6). The entry is copied into the CHANGE VIEW CONTROL: OVERVIEW screen.
5. Change the step number to "15" and the output type name to "ZNEU", but leave everything else the same.
6. Press Enter to return to the overview screen. Save the entry and assign it to a change request. The entry should look like Figure 8.36.

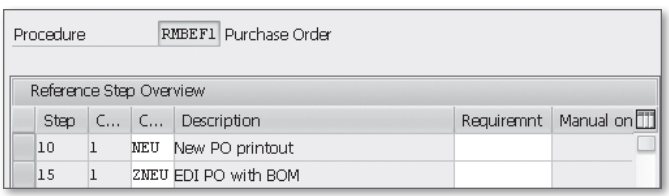


Figure 8.36 Assigning ZNEU to the Purchasing Procedure

Create Condition Records

Condition records drive generation of the IDoc for Acme's vendor, DSI. In this case, the IDoc will only be generated for PO document type. Follow these steps:

1. In Transaction NACE, select application EF and click CONDITION RECORDS.
2. Select OUTPUT TYPE "ZNEU". Click the CONDITIONS RECORDS button. The KEY COMBINATION dialog opens. Select the second key combination, as shown in Figure 8.37.

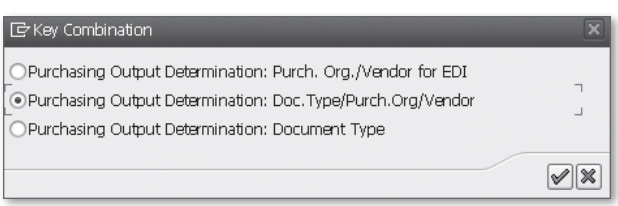


Figure 8.37 Select the Access Sequence in the Key Combination Dialog

3. Press Enter or click the green checkmark. The selection screen for the condition record opens. Enter the following values and click Execute to open the condition record table control (see Figure 8.38):
   - Enter "ZNB" for EDI PO with BOM enhancement in the Purchase Doc. Type field.
   - Enter "3000" (sample) in the Purch. Organization field.
4. In the condition records table control, enter the following:
   - "DISK01" for Acme's vendor Disc Services International
   - "VN" for vendor as the function
   - "6" for EDI as the medium
   - "4" for immediate when the document is saved as the dispatch time
   - "EN" as the language

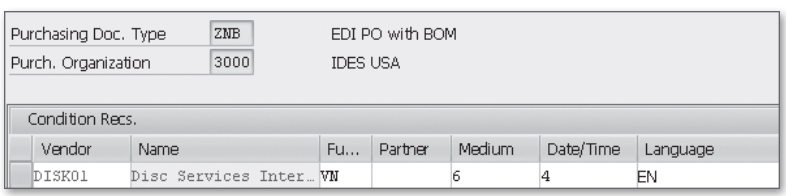


Figure 8.38 Condition Record for Output Type ZNEU in Application EF

Note that the full access key for the condition record is present in the header and table control area of the entry.

8.3.8 Build Outbound Partner Profile

All that's left is to set up a unique outbound partner profile for DSI using our logical message, extended type, and custom message control.

Message code BOM will create the unique key in table EDP12 for our partner profile. The message code links to the logical message and process code. The standard process code for the PO is ME10, which links message type ORDERS to IDoc function `IDOC_OUTPUT_ORDERS`. And this is where we coded our customer exit to populate the BOM extension.

We have two choices: either create a custom process code or extend the standard to include message code BOM. We'll extend the standard ME10.

Extending the Process Code

We can extend process code ME10 with Transaction WE41 or by following WEDI area menu Development • Outbound Processing Settings/MC • Outbound process code.

1. Select process code ME10 and double-click the Logical message folder in the navigation pane to open the Logical Message Details screen.
2. Click New Entries (or press F5) and add the following values to the added entries screen:
   - Message type: ORDERS
   - Message code: BOM

   We could select the All codes radio button under Message code and the system would pass any value found in the message code field, but being specific gives us better control over our partner profile keys.
3. Save and assign the entry to a change request. The screen should look like Figure 8.39.

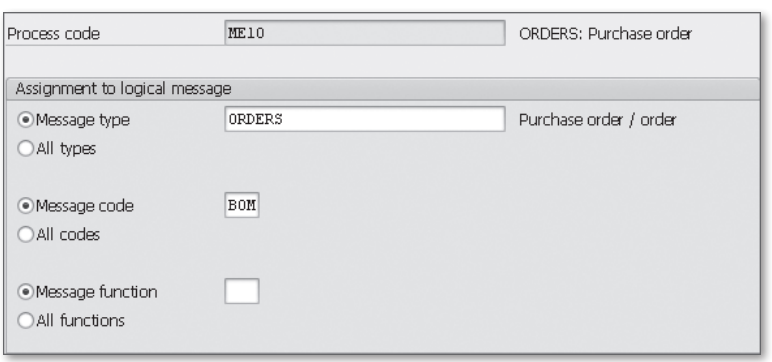


Figure 8.39 Linking Logical Message ORDERS to Message Code BOM

4. There should now be two entries for message ORDERS in the Logical Message table control in the Logical Message Overview screen, as in Figure 8.40.

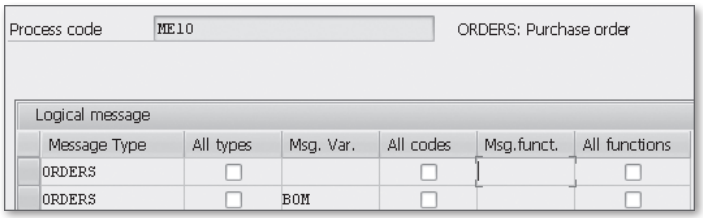


Figure 8.40 Process Code Extended with Multiple Instances of ORDERS

Creating the Partner Profile

Go to Transaction WE20 and create the header level partner profile for DSI in the Partner Type LI folder in the navigation pane, if one doesn't already exist.

1. Click Create outbound parameter beneath the outbound parameters table control and enter the following values that are shown in Figure 8.41:
   - Partner Role: "VN"
   - Message Type: "ORDERS"
   - Message code: "BOM"

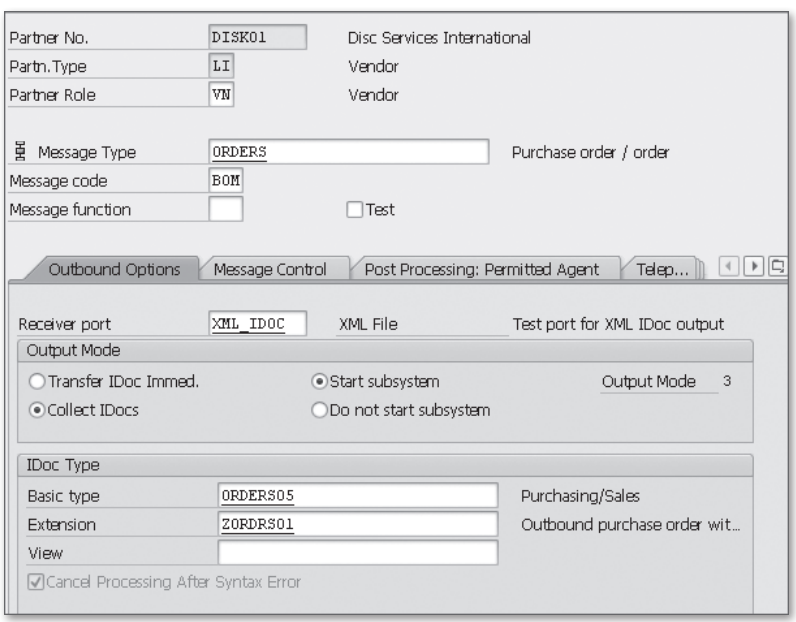


Figure 8.41 Outbound Parameters for the BOM PO IDoc

- Receiver port: "XML_IDOC"
- Output mode: The radio buttons for Collect IDocs and Start subsystem
- Basic type: "ORDERS05"
- Extension: "ZORDRS01"

2. Click on Message Control and add two entries with the following values that are shown in Figure 8.42:
   - Application: "EF"
   - Message type: "ZNEU"
   - Process code: "ME10"
   - Change message: One checkbox entry null and one checked

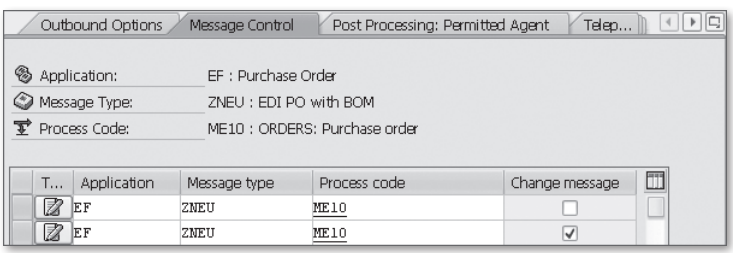


Figure 8.42 Message Control Values for the BOM ORDERS

3. We'll also need EDI information for the IDoc control record. All of Acme's IDocs will include this data. Select the EDI Standard tab, and enter the following values:
   - EDI Standard: "X" for X12
   - Message type: "850" for outbound X12 850
   - Version: "005010"
4. Save the partner profile.

This extended IDoc is now ready to test. Feel free to play with it, tweak it, extend it, and so on. This is only the starting point.

8.4 Summary

Now we've had a brief introduction to building custom and extended IDocs. We've toured the key IDoc development and configuration tools and have seen how they can be used to build a simple custom inbound and an extended outbound IDoc. We've also touched on enhancements and modification projects and have seen how simple it can be to find a specific user exit and enhancement using the Data Dictionary.

But there's so much more to know, so little time, and so few pages to learn it all, which is a major problem in any implementation project. We need to learn enough quickly enough to do the work that needs to be done. So our next step is to put our brief introduction to this material to work. Our legs may still be wobbly, but we'll take the plunge and begin building Acme's new SAP EDI system, at least the order-to-cash cycle of interfaces, beginning with the inbound customer purchase order.

Besides, the great Darryl Q. Fernhausen often began a new picture knowing even less about the plot, the actors, or the writers. So let's open the curtain on Act III and prepare to take the stage.

Index

A

B

C

D

E

F

G

H

I

J

L

M

O

P

R

S

T

U

V

W

X