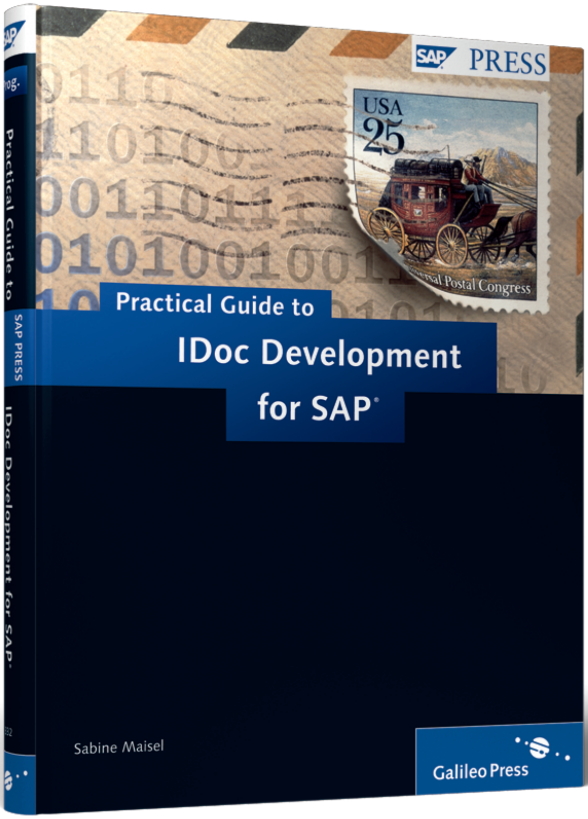
Sabine Maisel

Practical Guide to IDoc Development for SAP®



www.sap-press.com

Galileo Press®

Bonn • Boston

Contents at a Glance

1 Introduction 15

2 Generating IDocs 29

3 Test Tools 51

4 Changes to IDocs 67

5 Confirmations 187

6 Serializing IDocs 193

7 Administration 211

8 IDocs in Conjunction with SAP NetWeaver Process Integration 243

Contents

Preface 11

1 Introduction 15

1.1 What Is ALE? 15
1.2 BAPIs and IDocs — An Introduction 18
1.2.1 BAPIs 18
1.2.2 IDocs 19
1.3 Differentiation of ALE and EDI 25
1.4 Summary 27

2 Generating IDocs 29

2.1 Standard Methods for the IDoc Generation 29
2.1.1 Shared Master Data Tool 29
2.1.2 Message Control 43
2.1.3 Special Functions 46
2.2 Use of Logical Systems in the Message Control 49
2.3 Summary 50

3 Test Tools 51

3.1 Individual IDocs 52
3.2 Testing Processing of Multiple IDocs 57
3.2.1 Message Control 57
3.2.2 Sending Ready-for-Dispatch IDocs 58
3.2.3 Files 59
3.3 Processing Status Files 61
3.4 Summary 66

4 Changes to IDocs 67

4.1 Customizing 67
4.1.1 Filtering Using Filter Objects 67

4.1.2 Custom Filter Objects 73
4.1.3 Filtering Segments 76
4.1.4 Reducing IDocs Through Views 77
4.1.5 Rules 80
4.1.6 Version Conversion 84
4.1.7 Special Conversions in SAP ERP Financials 85
4.2 Adapting Existing IDoc Types 89
4.2.1 Different Exit Types on the Basis of the Material Master Example 90
4.2.2 General Exits 119
4.2.3 Custom Segments 122
4.2.4 Special Requirements for Master Data 134
4.3 Custom Function Modules for File Generation in File Ports 138
4.4 Custom IDocs 141
4.4.1 Creating Custom IDoc Types and Message Types 143
4.4.2 Generating an IDoc 153
4.4.3 Updating an IDoc 161
4.4.4 Generating IDoc Function Modules 172
4.4.5 Error Workflow for Custom IDocs 175
4.4.6 Useful Function Modules 184
4.5 Summary 186

5 Confirmations 187

5.1 "ALEAUD" IDocs 187
5.2 "STATUS" IDocs 190
5.3 Summary 192

6 Serializing IDocs 193

6.1 Serialization Using Groups 193
6.2 Serialization Using Timestamps 198
6.3 Serialization Using Business Objects 200
6.4 Serialization Using qRFC 205
6.5 Summary 209

7 Administration 211

7.1 IDoc Links 211
7.2 Regular Jobs 216
7.3 Transaction Code Overview 217
7.4 Archiving 224
7.5 Status Conversion 230
7.6 Summary 242

8 IDocs in Conjunction with SAP NetWeaver Process Integration 243

8.1 Conversion of Logical Systems to Business Systems 243
8.2 Conversion of IDoc Partner Roles 244
8.3 Header Mapping 245
8.4 Handling the Control Record in SAP NetWeaver PI 246
8.5 Updating IDocs Directly in SAP NetWeaver PI 248
8.6 Summary 248

The Author 249
Index 251

Sabine Maisel

Practical Guide to IDoc Development for SAP®

Galileo Press

Bonn • Boston

Preface

IDocs represent a standard interface to SAP systems. They are always asynchronous, so the system should be designed for changing and creating data in the database. The error handling, which can be carried out with a time delay in asynchronous communication, is always handled where the error occurs — in contrast to the normal RFC communication where the error is always notified to the sender.

Although more recent SAP releases also have other non-SAP-proprietary communication options (such as SOAP, HTTP, and proxies), the importance of IDocs is undiminished: This is because of the high number in which they are available, as well as the multitude of software solutions that work with SAP and already support this format. IDocs as standard interfaces to SAP systems are used both for EDI scenarios between different enterprises and for ALE scenarios within enterprises.

Like all standard interfaces, the IDocs refer to the part of the SAP system that is delivered by SAP. In most businesses, however, it's necessary to break adaptations that were made in the business part of the SAP system down to the interfaces. This task is carried out by ABAP developers. This book is particularly intended for them, and the tasks relating to these adaptations are discussed in detail. The system administrators usually handle the communication settings so these settings are only mentioned if they are directly related to the development work.

Content of This Book

This book describes things in the sequence in which they appear. It starts with the creation of IDocs and concludes with the regular tasks. The adaptation of IDocs to customer requirements starts with the lowest effort, that is, the options of Customizing, and proceeds step by step to the task that requires the highest effort: the complete custom programming of IDocs.

Chapter 1, *Introduction*, differentiates ALE and EDI and describes the basic principles of IDocs. Chapter 2, *Generating IDocs*, outlines the different options for generating IDocs. Chapter 3, *Test Tools*, discusses how you can test IDocs for the data exchange even without the communication partner.

After the generation of standard IDocs has been fully outlined, this chapter details the customer adaptations. Here, IDocs are connected to the different enhancement technologies of SAP. Some options of IDoc manipulation already arise in Customizing, while others require custom developments or the enhancement of the standard IDoc function modules. There are also special development objects that are only used in combination with IDocs. So Chapter 4, *Changes to IDocs*, describes particularly with regard to IDocs how you must handle enhancements and what you need to consider. The focus is on the enhancement technologies relevant for IDocs. Those enhancements that are used for all IDoc types are discussed in detail. This chapter also describes the specific features in the context of enhancements or custom-developed IDocs, such as the workflow connection.

Chapter 5, *Confirmations*, then discusses the live operation and shows how you can learn from your communication partners what happened to the IDoc despite the asynchronous procedure. Chapter 6, *Serializing IDocs*, outlines the different options to keep a specific sequence for processing IDocs, as well as the availability of these options. Chapter 7, *Administration*, describes necessary regular tasks in detail. Finally, in Chapter 8, *IDocs in Conjunction with SAP NetWeaver Process Integration*, those who work with SAP NetWeaver PI (formerly SAP NetWeaver XI) learn how they can implement tasks that are only required in the context of IDocs, specifically in the communication with SAP NetWeaver PI.

Acknowledgments

Above all, I would like to thank my husband Manfred. This book consumed a lot of our time, and without his support and all the work he did, I wouldn't have been able to write it.

I would also like to thank Maike Lübbers, Stefan Proksch, and Meg Dunkerley of Galileo Press — without their encouragement and support, I might have been able to finish this book, but it would definitely have contained many errors.

Sabine Maisel

This chapter introduces the interface types used by SAP in ALE scenarios. You'll learn about the characteristics of the different techniques and why different areas need different interface types. You'll also review terminology used throughout the book.

1 Introduction

You're certainly familiar with the topic of electronic data exchange between different enterprises. But even within enterprises, individual process steps can't be carried out on the same machine (or in the same database) using the same software. SAP meets this requirement by providing standard interfaces for locations where this kind of separation is common. *Standard interfaces* apply specific norms to actual data transfer to provide you with a certain release security.

1.1 What Is ALE?

Structuring communication

An example of how you can structure the communication according to process steps is the separation of human resource management and accounting, which you can find in many companies. All data generated in HR — on business trips, for example — that also needs to be transferred to accounting can be posted in the SAP system both locally or remotely. Because of this, it is at the sole discretion of a company whether to use one or several systems. SAP refers to this separation of processes within an enterprise as *Application Link Enabling* (ALE); processes that contain the relevant interfaces are called *ALE business processes* or *ALE scenarios*.

Interfaces

An interface generally allows you to send data from a certain point within the process (outbound interface) or to receive data at a specific point (inbound interface). A prerequisite for the useful employment of an interface is that both a technical and semantic description of that interface are available.

Technical description

The *technical description* specifies which data is supplied or expected in which format and in which sequence. This description is required to unpack the data.

Semantic description

The *semantic description* contains all of the required business information concerning the use of the interface, for instance, whether the data is read-only or can be edited. This description is required to interpret the incoming data.

Technical and Semantic Descriptions of an Interface

Consider an interface that we all use very often in our daily life: transferring sentences by telephone. Here, the semantic description corresponds to the knowledge that the last sentence is usually a farewell; the technical description, on the other hand, corresponds to the knowledge of how to structure a farewell note in English.

Remote Function Call

Of course, in addition to the descriptions, the physical transfer of data is necessary as well via a protocol. SAP supports its own RFC protocol (*Remote Function Call*) in all releases, whereas newer releases also support HTTP (*Hypertext Transfer Protocol*) and SOAP (which originally stood for *Simple Object Access Protocol*). Because ALE business processes represent a slightly older technology, they use only standard interfaces based on RFC and don't support the new proxy interfaces yet, which are based on SOAP.

Protocol

Protocol of an Interface

Let's get back to our telephone analogy: The phone line basically allows you to transfer data via fax or language. However, before you do that, you must agree with your partner on which of the two options you want to use. This corresponds to selecting the protocol, that is, the transfer type.

When designing an interface, you have the following options:

Synchronous communication

- **Synchronous interfaces**
  Synchronous interfaces consist of a *request* and a *response*. During this process, the connection between the systems is kept alive. Conse-

quently, synchronous interfaces immediately provide a result. This means that you — the user of this type of interface — can wait for the response. The advantage of a synchronous interface is that you receive a direct confirmation as to whether your order was processed or not. It's your responsibility to trigger the execution of the task. The disadvantage of this type of interface is that you can't continue to work until the other system processes the request or when the other system isn't available. In addition to that, long runtimes can require you to repeat the same operation if you think an error has occurred.

Example: Synchronous Interface

You want to buy a book about ALE and IDocs and call your local bookshop to obtain information on available books and the respective prices. It depends on the response you get whether you'll actually make the purchase or not. If a technical failure terminates the phone call prematurely, you'll call back only if you haven't received any response yet.

Asynchronous communication

- **Asynchronous interfaces**
  Asynchronous interfaces send the required data to the other system involved and don't receive an immediate response. This means that only a request is sent. As a prerequisite, no response is needed for this process. Consequently, you can continue working irrespective of the other system. However, because you won't receive an immediate result, in an asynchronous communication process, it must be ensured without your interference that the request arrives in the other system — and only once. The fact that an action is carried out only once is referred to as *transaction security*.

Example: Transaction Security

You want to buy my book and send a fax containing the purchase order data to the bookshop. The bookseller doesn't respond to your fax. However, the transmission report of your fax machine tells you that the bookshop has received the purchase order exactly once.

Interfaces in ALE scenarios

ALE scenarios can consist of any number of synchronous and asynchronous interfaces. SAP uses synchronous interfaces every time data from the database is supposed to be displayed; asynchronous interfaces, in turn, are used every time you want to create or change data in the database. In the latter case, the transaction security is particularly important to maintain database consistency.

1.2 BAPIs and IDocs — An Introduction

BAPIs

The medium for synchronous interfaces in ALE scenarios is the *Business Application Programming Interface* (BAPI). The term *Application Programming Interface* (API) is generally used in the IT environment. SAP extended it by the addition "Business" to indicate that the process not only involves a purely technical data transfer, but a step within a business process. In this context, the individual steps that make up the process can occur in different systems.

IDocs

Asynchronous interfaces in ALE scenarios are implemented using *Intermediate Documents* (IDocs). In addition to the business documents that are created in the different systems involved, asynchronous processes also generate a data document that ensures transaction security. Because this document isn't part of the actual business process but is rather an intermediate document, it has been assigned the corresponding name.

The following sections provide a brief introduction to the topic of BAPIs and IDocs. You'll learn how these different methods work and why two different types of interface exist within ALE scenarios.

1.2.1 BAPIs

A BAPI is a remote-enabled function module, that is, a module that can be executed by another program. In this context, it isn't important whether that program calls the function module locally or from within a different system. BAPIs differ from regular *RFC-enabled function modules* (RFM) in the following aspects:

BAPI requirements

- As a method, a BAPI is part of a business object type from the *Business Object Repository* (BOR). For all relevant objects in the SAP system, SAP provides a corresponding element in the BOR. In the BOR, you can find information regarding which tables pertain to an object and what you can do with them. It's also the basis for the *business workflow*. So, it makes much sense to also store in the BOR information about which process steps in ALE scenarios you can carry out with the respective objects.

- BAPIs are released, have a frozen signature, and thus provide release security.
- They typically use an update technique for postings instead of posting directly. Moreover, they don't contain any `Commit` or `Rollback` statements. (The few, mostly very old exceptions are specifically documented by SAP.)
- The actual posting process is triggered by the special `BAPI_TRANSACTION_COMMIT` BAPI or is undone via `BAPI_TRANSACTION_ROLLBACK`.
- BAPIs usually don't display the results because it's unlikely that the caller has a GUI (*graphical user interface*) available. BAPIs that were specifically developed for an SAP/SAP communication are an exception to that. However, these BAPIs aren't relevant for interfaces to external systems.

Calling BAPIs via RFC

Like all remote-enabled function modules, BAPIs can generally be called using a *synchronous RFC* (sRFC) or a *transactional RFC* (tRFC). The transactional call is asynchronous and makes sure that the call is executed exactly once. However, note that both the synchronous and the transactional call notify only the calling system in case of potential errors, even if that system can't remove the error. That's why BAPIs are generally used only for synchronous calls in ALE scenarios. In asynchronous cases, errors should preferably be reported directly to the receiver, provided these errors are business-related. BAPIs aren't intended for this purpose. The following section describes how the separation by error types is implemented by means of IDocs.

1.2.2 IDocs

SAP developed IDocs specifically to enable the asynchronous message exchange between multiple SAP and/or non-SAP systems. The advantages over a normal, asynchronously (transactionally) called RFM include an improved error-handling process as well as a specific monitoring layer made available to IDocs. Because error processing via IDocs affects the structure of IDoc interfaces, we'll look at it first in the following sections.

Error Handling with IDocs

You can generally distinguish between two different types of error:

Error types

- **Technical errors**
  Technical errors obviate the communication with the partner system. They occur, for example, if the partner system isn't available, the database in that system can't accept any additional requests, or the specified password is incorrect.
- **Business-related errors**
  Business-related errors occur if the partner system generally responds to requests but can't process your specific request. Missing authorizations, missing Customizing, or the like can cause this type of error.

In the context of RFCs, this distinction isn't made. In a RFC, the user directly receives an error message and must make sure that the request will be repeated at a later point in time. Because calls containing errors can't be repeated automatically and don't have to be logged, the distinction between error types wouldn't make sense here.

In tRFCs, each call that contains an error is written to a table in the sending system and can be restarted using a specific transaction. In this case, it does make sense to distinguish between the types of error. In that case, technical errors would be logged in the sending system, and the call would be repeated, if necessary, while business-related errors would be logged and processed in the receiving system. This option to distinguish between different types of errors is a specific feature of IDocs and isn't provided by RFCs their self. In addition, you can also change the contents of IDocs retroactively in case of errors, which isn't possible either with standard RFCs.

Errors in sending and receiving systems

To distinguish between technical errors and business-related errors, the processing of the respective processes must be divided into two steps as well. The first step is responsible for purely transferring the request to the database of the receiving system. This step doesn't depend on the type of data that is transferred. For example, it doesn't make any difference whether you want to create materials or cancel a sales order. To carry out the transfer in the same manner for all kinds of data, you use a neutral format: the IDoc format.

The second step is carried out locally in the receiving system and involves posting the relevant document. This step can vary depending on the business object to be processed.

Technical Description of the IDoc Format

IDoc as a neutral format

An IDoc consists of a *header record*, any number of lines of application data (*application records*), and any number of *status records* per IDoc.

Header record/ control record

- **Header**
  The header contains general information about which data is supposed to be transferred, who is the sender, and who is the receiver. This means that the receiver can learn from the header record which data has been received and — based on this information — decide how to process this data.

Data records

- **Data records**
  The data records contain business-related information. To make sure the technical format is independent of the business object and can also be understood by non-SAP systems, the content of each data record is stored as a string of 1000 characters. This character string is preceded by a control area containing information about how to interpret the 1000 characters.

Status records

- **Status records**
  Status records contain information about the previous statuses of the IDoc, such as "successfully created" or "successfully posted." Status records aren't transferred; that is, both the sender and receiver keep their own status records.

From a technical point of view, these three types of record have the same structure. Consequently, the structure doesn't provide any clue as to which kind of business-related information you're dealing with.

Figure 1.1 contains a theoretical illustration of how the different types of records can be used. Note that you can use an unlimited number of data records and status records. This is described in greater detail in the following section.

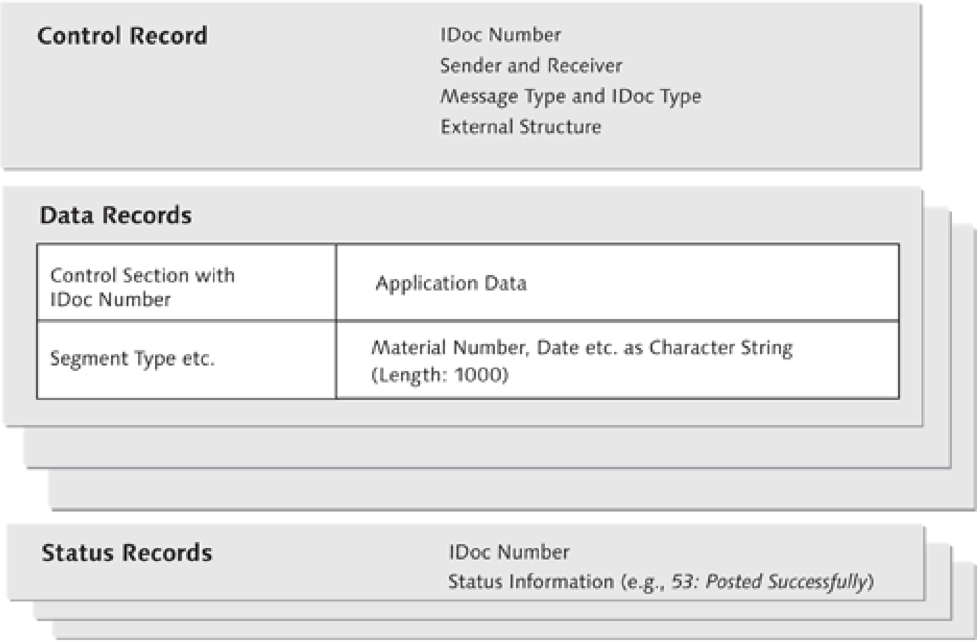

Control Record
IDoc Number
Sender and Receiver
Message Type and IDoc Type
External Structure
Data Records
Control Section with IDoc Number
Application Data
Segment Type etc.
Material Number, Date etc. as Character String (Length: 1000)
Status Records
IDoc Number
Status Information (e.g., *53: Posted Successfully*)

Figure 1.1 Structure of an IDoc

To process the IDoc, the receiver needs *meta information*, that is, information concerning the manner in which the data is supposed to be processed. The meta information that describes an IDoc consists of three parts:

Message type

- **Message type**
  The message type provides a semantic description of the data to be processed. For example, message type `MATMAS` indicates that material master data is supposed to be exchanged. The message type is also responsible for the way a message will be processed. When you create or change a sales order, for example, the actual information is identical, but the processing is different, which is why the creation of the order is assigned the `ORDERS` message type, while message type `ORDCHG` is assigned to the change activity. The names of message types are based on the names of the UN/EDIFACT standard (*United Nations Electronic Data Interchange for Administration, Commerce and Transport*), but apart from the names, there are no other similarities between the IDoc and EDIFACT formats.

IDoc type

- **IDoc type**
  The IDoc type represents the technical description of the data. It tells you which fields of which business objects are supposed to be filled

with which values. This means that the IDoc type describes how to interpret the 1000 characters in the data record.

Segment type

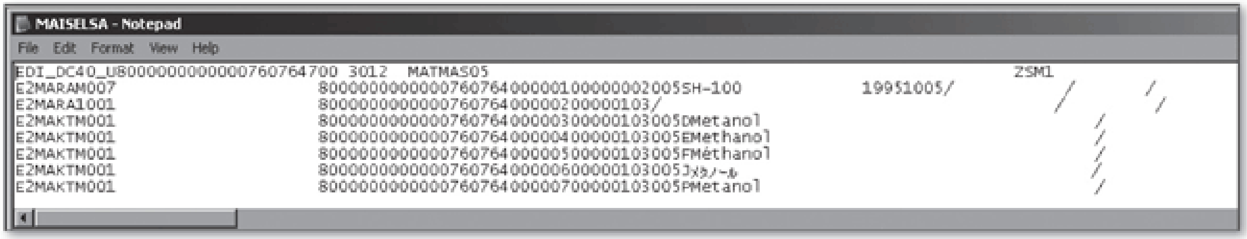
- **Segment type**
  For each possible variant of these 1000 characters, there is one segment type. For example, segment type E1MARAM indicates that material master data from Table MARA is going to follow. In each SAP system, information is stored as to which IDoc type can have which segments, how often, and in which sequence.

You can export this data to external systems to avoid having to maintain all structures manually. SAP supports the transfer of data via transport IDocs, C files, DTD (*Document Type Description*), or HTML (*Hypertext Markup Language*). If you want to use transport IDocs, you must ensure that your partner can generally receive IDocs from you, which means that your SAP system must know the partner as a logical system.

The IDoc, which exists as a set of data records, is then transferred from the sender to the receiver. The most commonly used type of transfer is the tRFC. The transfer is carried out in such a way that the sending system calls the function module IDOC_INBOUND_ASYNCHRONOUS in the receiving system. This function module receives the data and posts the header and data records to the associated database tables EDIDC and EDID4, and the status of the IDoc to EDIDS. After that has been done, the technical part is finished. If problems occur during this process, the sender will be notified and must resend the data at a later point in time.

IDoc as file

Other types of transfer, such as *file* (flat file), XML (file in XML format), or *HTTP* can also be used because of the separation of data transfer and data processing. Figure 1.2 shows the first part of a small material master IDoc as a file, opened in Notepad.



```
EDI_DC40_U8000000000000760764700 3012  MATMAS05                                              ZSM1
E2MARAM007                      80000000000007607640000010000000200 5SH-100          19951005/          /        /
E2MARA1001                      800000000000076076400000200000103/                                      /        /
E2MAKTM001                      80000000000007607640000030000010300 5DMetanol                              /
E2MAKTM001                      80000000000007607640000040000010300 5EMethanol                             /
E2MAKTM001                      80000000000007607640000050000010300 5FMéthanol                             /
E2MAKTM001                      80000000000007607640000060000010300 5Jメタノール                            /
E2MAKTM001                      80000000000007607640000070000010300 5PMetanol                              /
```

Figure 1.2 IDoc in Notepad Editor

You can specify the type of transfer to be used for transferring IDocs to the partner system via the *outbound partner profile* (Transaction WE20). After the IDoc has arrived in the receiving system, the business-related posting process begins. Typically, this is also carried out by means of a function module.

The *inbound partner profile* (also Transaction WE20) specifies the process code to be used. For each possible combination of sender and message type, you can define which function module will be used. SAP provides all necessary function modules for all ALE scenarios available. Note that you can customize these function modules or even replace them with your own. Chapter 4, Changes to IDocs, provides more detailed information on this.

IDoc tracking

If an error occurs during the business-related posting, the error will be documented in the subsequent status record of the IDoc, and you can restart the posting process later. SAP provides a separate transaction for posing an IDoc after a processing error: Transaction BD87. This transaction can resend IDocs that contained errors in the sending system during the first attempt and can re-process erroneous IDocs in the receiving system, if necessary. After the reprocess, everything is finished. The posting of the IDoc creates the business Document, or changes it. This is only done once. Once it is successfully created or changed, the Business Document is not touched again. If both systems are SAP systems, you can even use the *IDoc tracking* function from within the sending system to obtain information on the current status of the respective IDoc in the receiving system. You can start IDoc tracking from within the IDoc monitor (Transaction BD87). This function uses the function module `IDOC_DATE_TIME_GET` via sRFC to read the IDoc number and its status in the receiving system.

BAPIs and IDocs in ALE Scenarios

Due to the transaction security enabled by the updating of erroneous connections and because of target-oriented error-handling procedures, the ALE integration scenarios provided by SAP always use BAPIs for synchronous, requesting access, and IDocs for asynchronous, changing access. If it's possible to transfer data in both ways, that is, synchronously and asynchronously (e.g., in a first, synchronous test run, and then the actual posting asynchronously), both a BAPI and a matching IDoc are needed. The IDoc therefore can be created via Transaction BDBG, which is also available to customers.

1.3 Differentiation of ALE and EDI

The concept of ALE comprises communication within an enterprise. However, within an enterprise, different processes are carried out in physically different IT systems, which means that individual process steps can involve different systems. As you've already learned, these steps can be carried out in a synchronous process via BAPIs as well as in an asynchronous process involving IDocs.

Logical system

In this context, the communication itself is carried out between the logical systems (LSs). Each LS represents a combination of hardware and software installed on that system. In an SAP system, a client corresponds to a LS. The LS is assigned to the client and provides information as to whether a document was created on that SAP client or whether it was adopted by means of consolidation processes (the majority of documents contain the original system in a specific field in the database). The assignment of the LS is part of the post-installation procedures. A system must know its own system name as well as the names of all LSs it communicates with.

Uniqueness of logical systems

The LSs must be unique across the entire enterprise (actually across the entire range of communication). Moreover, the LSs for non-SAP systems with which a communication is established must also be known. You can assign the LS for your own client as well as make systems known with which messages will be exchanged via ALE-Customizing, which can be reached using Transaction SALE. Figure 1.3 shows the Customizing functions, Define Logical System and Assign Logical System to Client. Transaction BD54 enables you to assign names to LSs. In addition to the name for the LS, you should also enter a brief description.

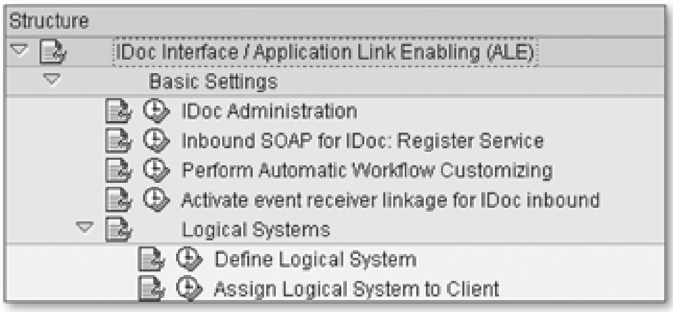
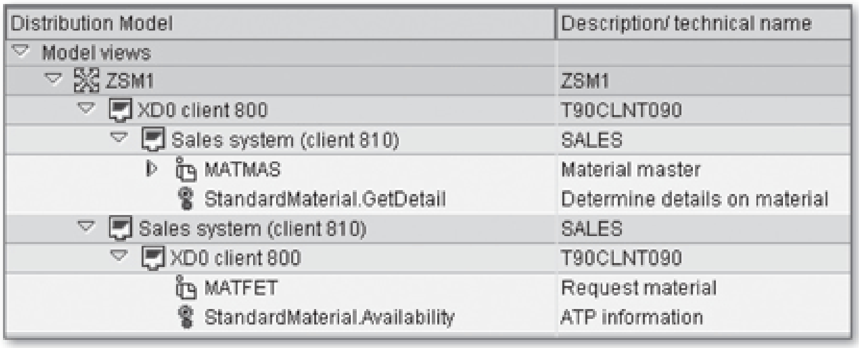

Structure
IDoc Interface / Application Link Enabling (ALE)
Basic Settings
IDoc Administration
Inbound SOAP for IDoc: Register Service
Perform Automatic Workflow Customizing
Activate event receiver linkage for IDoc inbound
Logical Systems
Define Logical System
Assign Logical System to Client

Figure 1.3 Setting Up Logical Systems

Customer distribution model

The *customer distribution model* (Transaction BD64) allows you to define which BAPIs and IDocs will be exchanged between which systems. In this context, each system can function as the sender or receiver. In our example (see Figure 1.4), the LS `T90CLNT090` sends the IDoc `MATMAS` to the LS `SALES`, while the IDoc `MATFET` is sent in the opposite direction. In addition, two BAPIs are exchanged. The `Material.GetDetail` BAPI is sent from `T90CLNT090` to `SALES`, while the `Material.Availibility` BAPI is sent in the opposite direction.



Distribution Model	Description/ technical name
▽ Model views	
▽ ZSM1	ZSM1
▽ XD0 client 800	T90CLNT090
▽ Sales system (client 810)	SALES
▷ MATMAS	Material master
StandardMaterial.GetDetail	Determine details on material
▽ Sales system (client 810)	SALES
▽ XD0 client 800	T90CLNT090
MATFET	Request material
StandardMaterial.Availability	ATP information

Figure 1.4 Distribution Model with Example for IDoc and BAPI

EDI

In contrast to ALE as a means of enterprise-internal communication, EDI (*Electronic Data Interchange*) comprises the transfer of data between different enterprises, for example, between customers and suppliers. Each company has its own process flows. However, at certain points within a process, the partner must be informed. To prevent the partner from having to enter specific information manually into the system, the relevant information must be transmitted to the partner electronically.

EDI is always an asynchronous process. With regard to SAP systems, this means that IDocs are also used for EDI communication. However, note that EDI communication is always carried out without the distribution model and LSs. Within the system, the relevant senders and receivers of messages are also referred to as *partners*; customers and suppliers are one example here. Note that they must be maintained as customers and suppliers in the SAP system. No new, IDoc-specific master data is needed in this context.

EDI standards

Because different companies use different IT systems, specific EDI standards exist to facilitate data transfers. In European countries, the EDI-

FACT standard is primarily used, whereas in the United States, ANSI ASC X12 (*American National Standards Institute Accredited Standards Committee X12*) is the predominant standard. EDIFACT assigns names to messages, while ANSI uses numbers. For example, according to the EDIFACT standard, the message `ORDERS` represents a purchase order; according to the ANSI standard, the same message is called `850`. With regard to message types, SAP has based its naming conventions on the EDIFACT standard; however, the structure of the messages differs substantially from that standard. (Chapter 2, Generating IDocs, describes the structure of IDocs in greater detail.) In SAP systems, IDocs are used for asynchronous data transfers both in EDI and ALE processes, which is why the procedures described in the course of this book can always be applied to both ways of communication.

The following chapters will focus entirely on using IDocs. As far as the use of BAPIs is concerned, it's similar to using function modules. Consequently, developers who are familiar with function modules can carry out all development tasks related to BAPIs without a problem. IDocs, on the other hand, contain some specific features that aren't usually described in developer manuals. These specific features are described in this book.

1.4 Summary

In this chapter, we discussed the reasons for using interfaces in general, as well as the types of interfaces provided from SAP. You learned that Customizing and programming with IDocs is very important.

In the next chapter, we'll take a deeper look at IDocs, by starting at the very beginning – the different possibilities to create IDocs for sending them to our LSs or partners.

In the sending system, it's necessary to generate IDocs so that they can be sent to the receiving system. How this is done depends on the type of data and the application. This chapter describes the different generation options and their use.

2 Generating IDocs

SAP has provided tools for the generation of IDocs in all locations where they are used in ALE scenarios or classic EDI, and they can usually be activated using the Customizing settings. However, there are different methods of IDoc generation depending on the type of data and the location where the IDoc will be generated. This chapter presents the most common methods of IDoc generation. Usually the IDoc administrator — not the developer — implements the settings required for creating the partner profiles, for instance, so they are outlined only briefly, and more importance is attached to the functional process flow.

2.1 Standard Methods for the IDoc Generation

Initially, you must distinguish between the generation of master data and the generation of transaction data because there are different requirements on the generation process or the generation frequency depending on the type of data. A special tool is available to generate master data, called the *Shared Master Data Tool* (SMD). Transaction data IDocs are generated via the already-existing message control. There are also some special functions for IDocs that are directly generated in a process.

2.1.1 Shared Master Data Tool

Master data in the IDoc

The SMD is a special tool for sending master data via IDoc. Master data is characterized by a relatively long retention period in the system during which the data is changed rarely. Master data usually consists of multiple views that are not used all the time. You can omit views, including those that contain mandatory fields, because the check of whether all mandatory fields are populated is only carried out if the view is actually used. This enables you to select from the wealth of information that is offered for a specific object and use exactly the data that is actually required within your enterprise.

To distribute data using IDocs, an automated process is desirable that responds to the creation and modification of master data without requiring further user interventions. Also, empty views are not supposed to be transferred.

Automation and control via views

The SMD takes these requirements into account. The technical implementation of automation and control via views entails that already-existing procedures can be used for both functions. For automation, you revert to recording changes, which is implemented by default; for control via views, you use the option (which originated from batch input processing) to control irrelevant fields using a NO_DATA character. Then, the IDocs are regularly generated via background jobs.

Additionally, for almost all objects, you also have the option to explicitly generate IDocs or to request IDocs. You can use this option if waiting for periodically scheduled jobs is impossible.

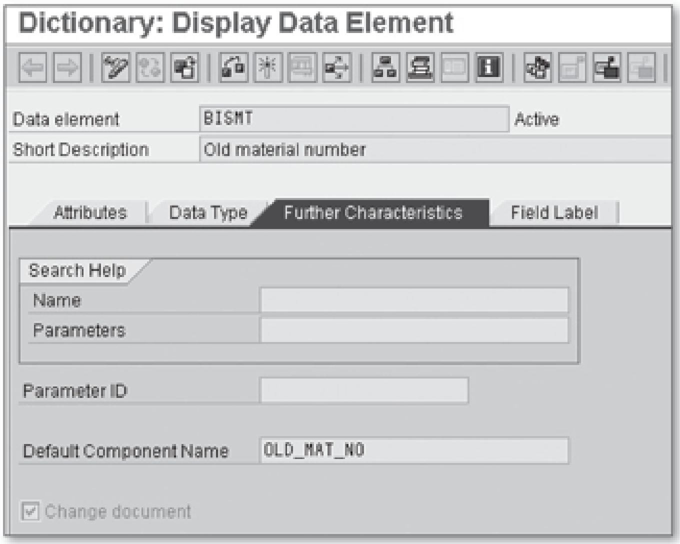
Recording Changes

Change pointer

For consistency reasons, changes to the master data are updated independently of the use of ALE in SAP systems. For each individual data element of the tables concerned, SAP has stipulated whether a change should be logged or not. Figure 2.1 shows the CHANGE DOCUMENT FLAG, which is activated in this case as an example of BISMT (*Old Material Number*) from the MARA TABLE.

Writing the history in the application

For updating the changes, you always call the CHANGEDOCUMENT_OPEN function module, which prepares the writing of the change history. Then, all changes to be written are collected, and the process is concluded using the CHANGEDOCUMENT_CLOSE function module. Because IDocs are supposed to be generated wherever changes are updated by default, the CHANGEDOCUMENT_CLOSE function module has an ALE share in addition to its standard function. This enables you to generate *change pointers* for ALE for the desired message types. In all Unicode-enabled releases, this is done using the CHANGE_POINTERS_CREATE_LONG function module; in old releases, this is done using the CHANGE_POINTERS_CREATE function module.



Dictionary: Display Data Element

Data element: BISMT Active
Short Description: Old material number

Search Help
Name:
Parameters:

Parameter ID:

Default Component Name: OLD_MAT_NO

☑ Change document

Figure 2.1 Characteristics of Data Elements

Generating change pointers directly

You also have the CHANGE_POINTERS_CREATE_DIRECT function module as a second option to generate change pointers. This function module is called by applications that are not connected to the previously described change management for documents.

In both cases, change pointers are only written if you use the SMD for performance reasons. In the ALE Customizing, you can specify whether this is the case and for which master data you require change pointers. There is a separate Transaction code SALE for the ALE Customizing that takes you directly to the correct position in the menu tree. Figure 2.2 shows the menu path in Customizing in which you make the necessary settings.

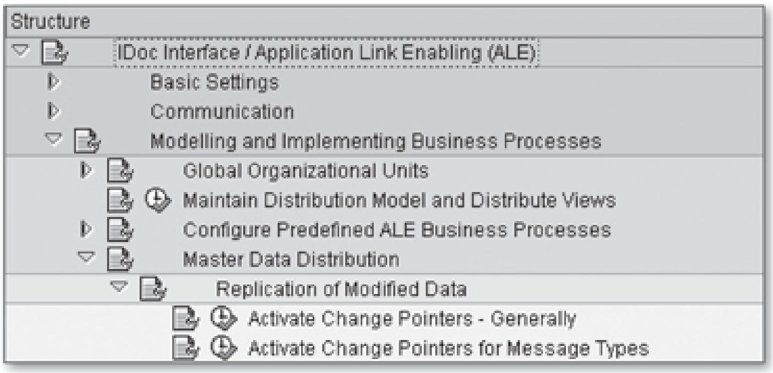
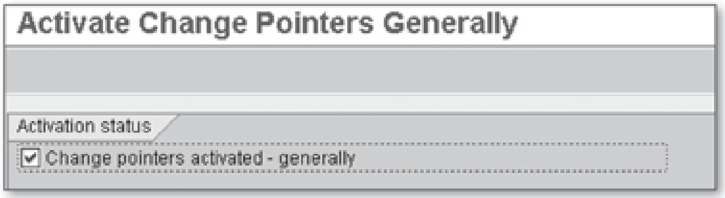

Structure
IDoc Interface / Application Link Enabling (ALE)
Basic Settings
Communication
Modelling and Implementing Business Processes
Global Organizational Units
Maintain Distribution Model and Distribute Views
Configure Predefined ALE Business Processes
Master Data Distribution
Replication of Modified Data
Activate Change Pointers - Generally
Activate Change Pointers for Message Types

Figure 2.2 Activating Change Pointers for the SMD

Activating change pointers

Initially, you activate the generation of change pointers generally. As a result, the share of the `CHANGEDOCUMENT_CLOSE` function module that has not been used up to now and that is responsible for the SMD is run through. This must be set once only for all master data. Figure 2.3 shows the appropriate functionality.



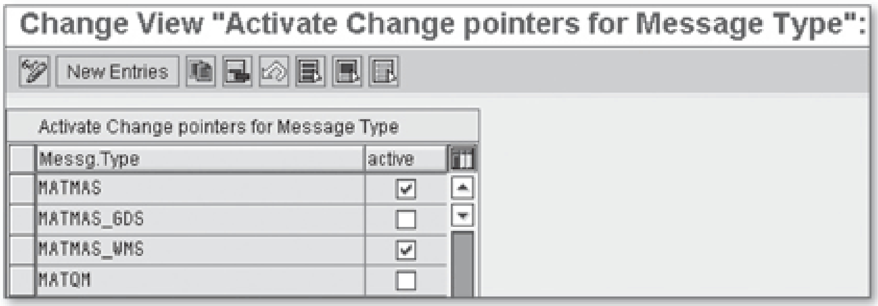
☑ Change pointers activated - generally

Figure 2.3 Activate Change Pointers Generally

Activating change pointers for object

If the change pointers are activated generally, you can specify in a second step for which message type you require the generation of change pointers. This is done in the second menu subitem (Activate Change Pointers for Message Types) and has been implemented for the message types, `MATMAS` and `MATMAS_WMS` (see Figure 2.4). No change pointers are written for `MATCOR` and `MATMAS_GDS`, which have not been activated.

In this context, it's important that a change pointer isn't written for each changed field because there are fields whose values are not significant for the downstream system. By means of Transaction BD52, SAP provides fields for each message type connected to the SMD, which are relevant for changes from SAP's point of view. For the material master, the DMAKT-SPRAS field is important, for example. In the transaction, you enter the fields that are used within your enterprise. If you've made changes to the corresponding master data tables via the SAP enhancement concept and use customer-specific fields, you can also set the customer-specific fields here. Refer to Chapter 4, Changes to IDocs, to learn how to provide these fields.

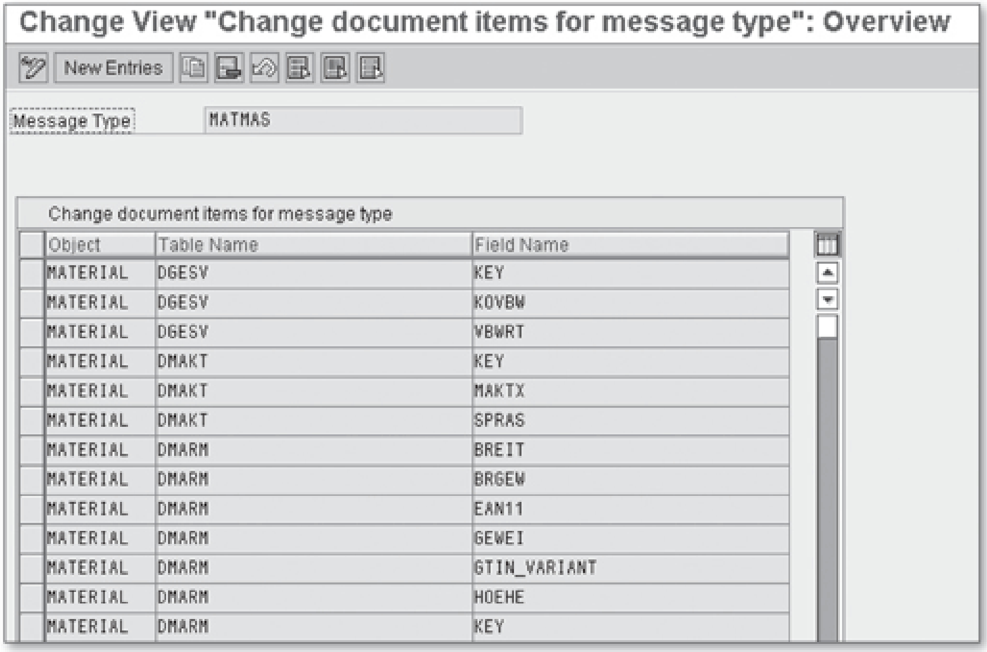


Change View "Activate Change pointers for Message Type":

Activate Change pointers for Message Type

Messg.Type	active
MATMAS	☑
MATMAS_GDS	□
MATMAS_WMS	☑
MATQM	□

Figure 2.4 Activate Change Pointers for Message Type



Change View "Change document items for message type": Overview

Message Type: MATMAS

Change document items for message type

Object	Table Name	Field Name
MATERIAL	DGESV	KEY
MATERIAL	DGESV	KOVBW
MATERIAL	DGESV	VBWRT
MATERIAL	DMAKT	KEY
MATERIAL	DMAKT	MAKTX
MATERIAL	DMAKT	SPRAS
MATERIAL	DMARM	BREIT
MATERIAL	DMARM	BRGEW
MATERIAL	DMARM	EAN11
MATERIAL	DMARM	GEWEI
MATERIAL	DMARM	GTIN_VARIANT
MATERIAL	DMARM	HOEHE
MATERIAL	DMARM	KEY

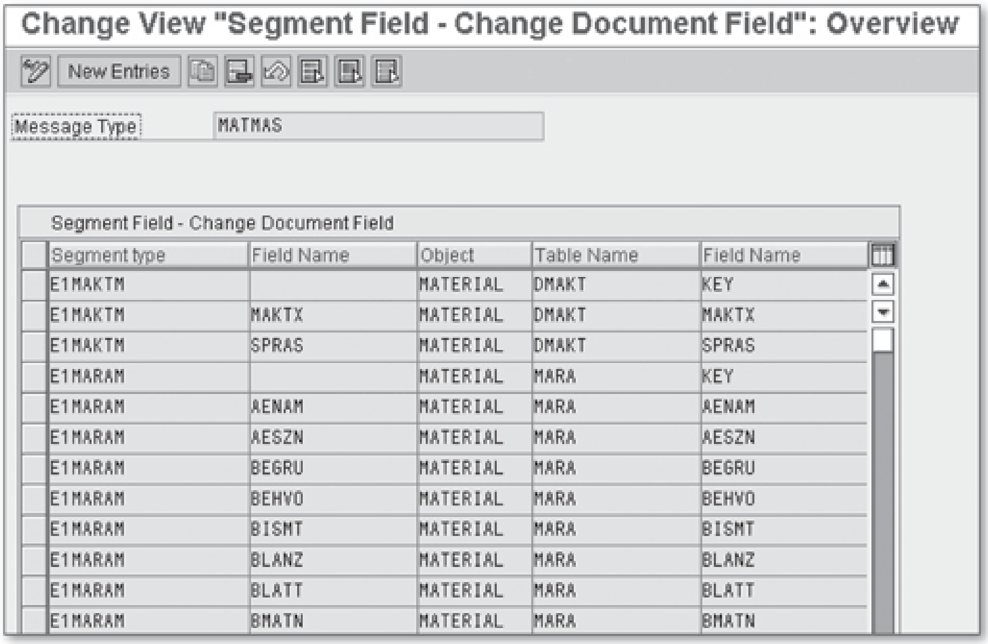
Figure 2.5 Change-Relevant Fields in Transaction BD52

Change-relevant fields

Figure 2.5 shows a section of the fields for the MATMAS message type, which are provided by SAP as change-relevant. The message type is directly referenced to fields and tables of the material master except for the KEY field. This field isn't part of the respective table but assumes a very important, additional control role: It ensures that the creation of a table entry can be sent via IDoc. If the KEY field is specified in Transaction BD52, a change pointer is written during the creation of the corresponding object, for example, during the initial creation of the material for the MARA-KEY dummy field or during the creation of a text in a new language for the MAKT-KEY dummy field. Imagine that the key value of the table concerned is changed from "empty" to the new value. As a result, all fields of this table are transferred.

Assigning change-relevant fields

Additionally, for each of the change-relevant fields, you must specify to which field in which segment of the IDoc type it belongs. This is done in Transaction BD66, which is shown in Figure 2.6. The DMAKT-SPRAS sample field from Transaction BD52 belongs to the `E1MAKTM` IDoc segment and to the field that is also called "SPRAS." For your own fields, you must specify this using the New Entries button.



Change View "Segment Field - Change Document Field": Overview

Message Type: MATMAS

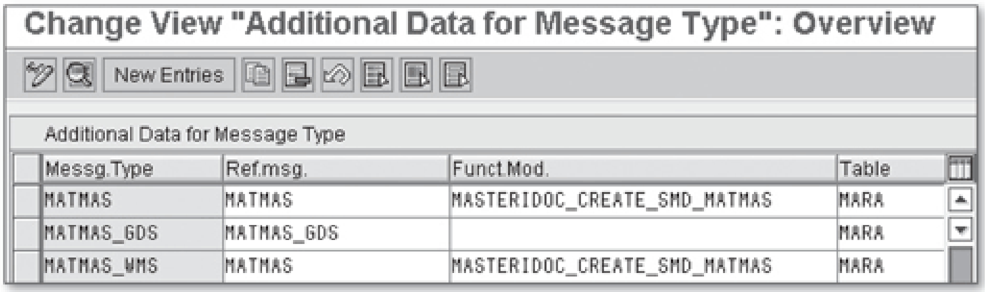
Segment Field - Change Document Field

Segment type	Field Name	Object	Table Name	Field Name
E1MAKTM		MATERIAL	DMAKT	KEY
E1MAKTM	MAKTX	MATERIAL	DMAKT	MAKTX
E1MAKTM	SPRAS	MATERIAL	DMAKT	SPRAS
E1MARAM		MATERIAL	MARA	KEY
E1MARAM	AENAM	MATERIAL	MARA	AENAM
E1MARAM	AESZN	MATERIAL	MARA	AESZN
E1MARAM	BEGRU	MATERIAL	MARA	BEGRU
E1MARAM	BEHVO	MATERIAL	MARA	BEHVO
E1MARAM	BISMT	MATERIAL	MARA	BISMT
E1MARAM	BLANZ	MATERIAL	MARA	BLANZ
E1MARAM	BLATT	MATERIAL	MARA	BLATT
E1MARAM	BMATN	MATERIAL	MARA	BMATN

Figure 2.6 Assignment of IDoc Fields to Change-Relevant Fields

Evaluating change pointers

The change pointers are then evaluated. It just depends on the object concerned which function module is used here. When you call Transaction BD60, you can view these function modules and replace them with your own function modules if you want to make so many changes to the standard functionality that you don't want to enhance or modify the SAP original. Figure 2.7 shows another example of the material master data. There, the function module that uses change pointers to generate IDocs is called MASTERIDOC_CREATE_SMD_MATMAS.


Change View "Additional Data for Message Type": Overview

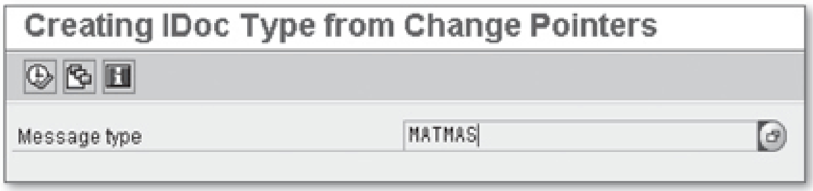
Additional Data for Message Type

Messg.Type	Ref.msg.	Funct.Mod.	Table
MATMAS	MATMAS	MASTERIDOC_CREATE_SMD_MATMAS	MARA
MATMAS_GDS	MATMAS_GDS		MARA
MATMAS_WMS	MATMAS	MASTERIDOC_CREATE_SMD_MATMAS	MARA

Figure 2.7 Function Modules to Evaluate Change Pointers

"RBDMIDOC" background job

The RBDMIDOC report, which must be scheduled at regular intervals, must then use these function modules to generate IDocs from the change pointers and update which change pointers have been processed. As a transfer value, you can specify for which message type you want to implement the evaluation. For this purpose, you specify the appropriate type (here: MATMAS) in the Message Type field in the initial screen of the RBDMIDOC report. This is illustrated in Figure 2.8.


Creating IDoc Type from Change Pointers

Message type: MATMAS

Figure 2.8 Initial Screen of the "RBDMIDOC" Report

Changes in the SMD

For sending IDocs using the SMD and for change pointers, the application and the ALE communication layer interact closely. Figure 2.9 schematically shows the process flow of IDoc generation. The entire process of writing change pointers takes place in the application; the evaluation of the change pointer and the generation of IDocs are carried out by the ALE communication layer.

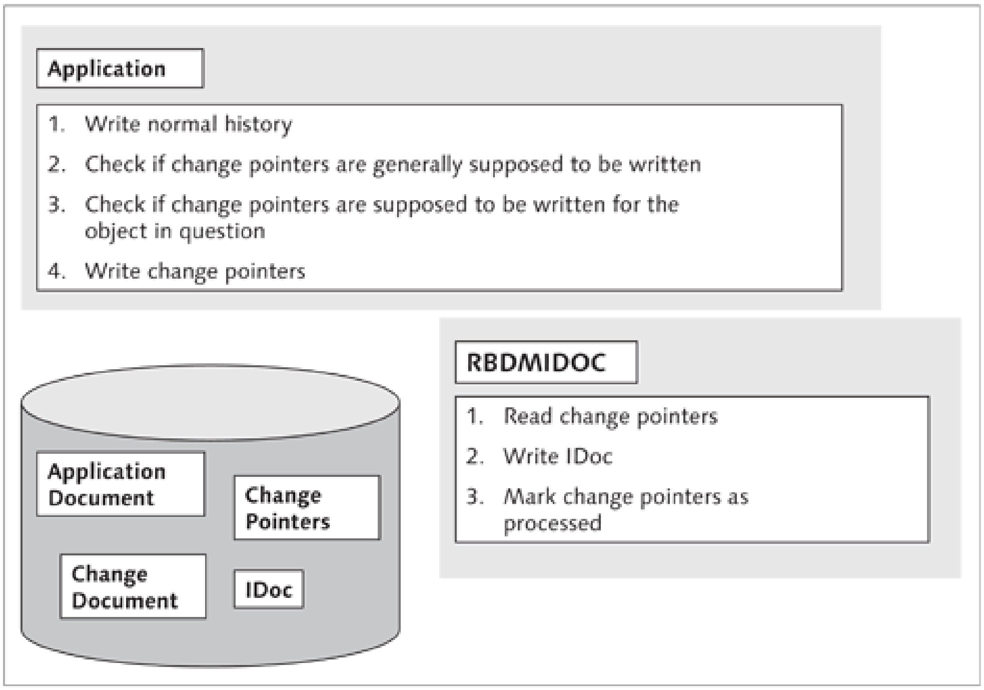

Application
1. Write normal history
2. Check if change pointers are generally supposed to be written
3. Check if change pointers are supposed to be written for the object in question
4. Write change pointers
RBDMIDOC
1. Read change pointers
2. Write IDoc
3. Mark change pointers as processed
Application Document
Change Pointers
Change Document
IDoc

Figure 2.9 IDoc Generation Using the SMD

Note that when you send IDocs via change pointers, you only send those views in which changes have actually been made. If a new view has been created, all its fields are transferred; if a view has been changed, only the changed fields are transferred to increase performance. Chapter 4, Changes to IDocs, uses the example of the material master to describe how you can change this default behavior using a minor modification.

You can find the change pointers in the `BDCP` table and the corresponding status records in the `BDCPS` table. As of SAP NetWeaver Application Server 6.20 (SAP NetWeaver AS), processing with higher performance is possible via a shared table called `BDCP2`. However, this new procedure isn't supported for all message types. Whether it's applicable for your message type is indicated in the detail view of Transaction BD60. As of Release 7.1, only the new processing using Table `BDCP2` is available for all message types.

Distribution lock

For the distribution via the SMD, you must consider some specifics for some master data. You can set a distribution lock for the material master to generally prevent the sending of a material. For this purpose, you must make a short detour. In the design data of the material in the MARA table, there is a cross-plant material status (MARA-MSTAE field). This status refers to an existing entry in the T141 table. Here, you can assign additional properties for each status value. If the DLOCK field provided here is selected, the distribution lock is set.

For contracts (BLAORD message type), only released contracts are transferred using the SMD.

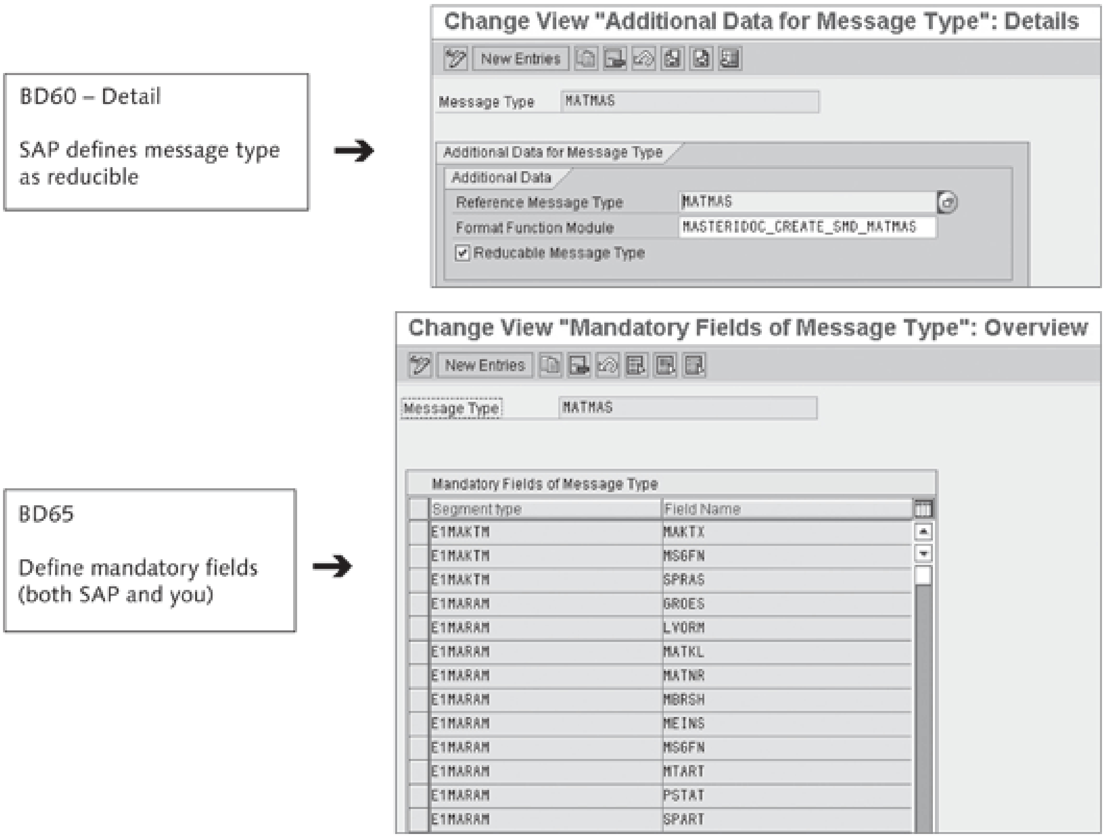
Reducing Messages

The second requirement on the distribution of master data arises from the distribution of all master data to individual views and the option to define, in Customizing, which fields of a view are actually supposed to be used. This is scalable for the IDoc transfer by using the reduced message types.

Reduced message type

A reduced message type always relates to an existing message type but transfers less data. The reduction isn't possible for all message types; so the developer of the message type must explicitly define it as reducible. All views and fields that must be transferred as a minimum are predefined here; all other views and fields can be selected additionally if required. Transaction BD60 in the detail view for a message type is the transaction for defining a message type as reducible. By calling Transaction BD65, you define the mandatory fields.

For each delivered message type, SAP specifies whether it's reducible. Because this involves functions in the generation and update module for the corresponding IDoc, customers can't simply set the message type to reducible retroactively. The mandatory fields that SAP provides with Transaction BD65 correspond to the Customizing that SAP provides for transactions that are used to maintain master data, for instance, Transaction MAT1 for maintaining the material master. If you make changes to the mandatory fields in the application's Customizing, you also adapt them in Transaction BD65 so that custom-developed mandatory fields are also mandatory for reducing in the IDoc. Figure 2.10 shows a section of the data for the MATMAS message type.



BD60 – Detail

SAP defines message type as reducible

Change View "Additional Data for Message Type": Details

Message Type: MATMAS

Additional Data for Message Type

Additional Data

Reference Message Type: MATMAS

Format Function Module: MASTERIDOC_CREATE_SMD_MATMAS

☑ Reducable Message Type

BD65

Define mandatory fields (both SAP and you)

Change View "Mandatory Fields of Message Type": Overview

Message Type: MATMAS

Mandatory Fields of Message Type

Segment type	Field Name
E1MAKTM	MAKTX
E1MAKTM	MSGFN
E1MAKTM	SPRAS
E1MARAM	GROES
E1MARAM	LVORM
E1MARAM	MATKL
E1MARAM	MATNR
E1MARAM	MBRSH
E1MARAM	MEINS
E1MARAM	MSGFN
E1MARAM	MTART
E1MARAM	PSTAT
E1MARAM	SPART

Figure 2.10 Basic Maintenance for Reducible Message Types

Reduction in Customizing

You then create your own reduced message types in Customizing of Transaction SALE. Under the Create Reduced Message Type menu item or via Transaction BD53 you can find the initial screen as shown in Figure 2.11.

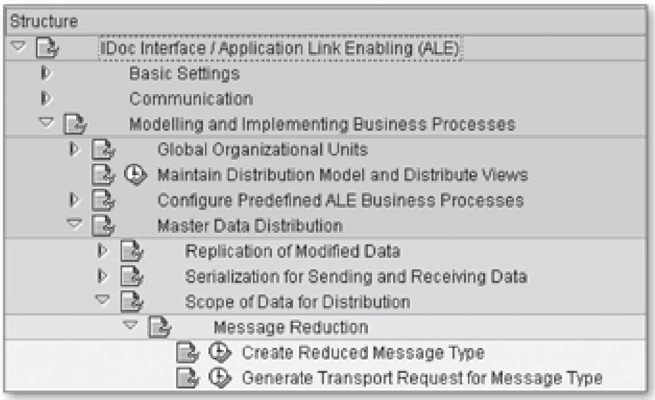
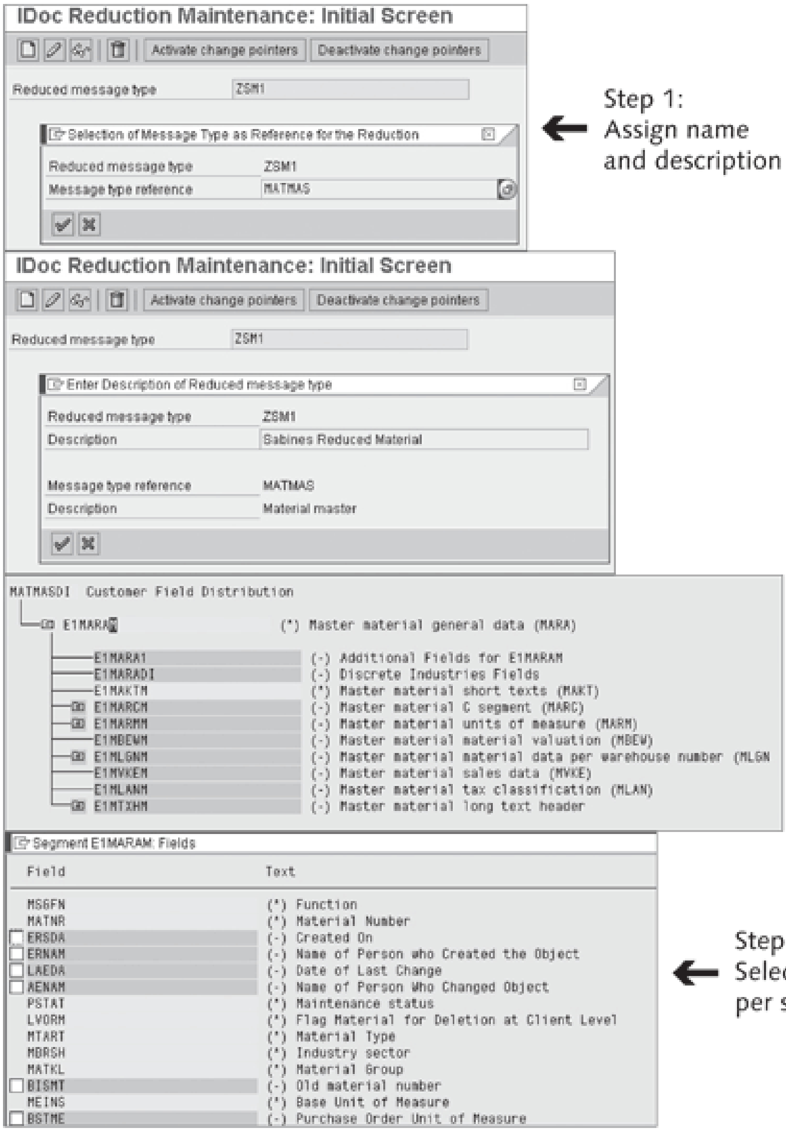

Structure
IDoc Interface / Application Link Enabling (ALE)
Basic Settings
Communication
Modelling and Implementing Business Processes
Global Organizational Units
Maintain Distribution Model and Distribute Views
Configure Predefined ALE Business Processes
Master Data Distribution
Replication of Modified Data
Serialization for Sending and Receiving Data
Scope of Data for Distribution
Message Reduction
Create Reduced Message Type
Generate Transport Request for Message Type

Figure 2.11 Create Reduced Message Type

Reduction steps

When you specify the name of the new reduced message type (see Figure 2.12), you must consider the naming rules for your own objects (the name must start with Y or Z or your own namespace).



IDoc Reduction Maintenance: Initial Screen
Activate change pointers
Deactivate change pointers

Reduced message type: ZSM1

Selection of Message Type as Reference for the Reduction

Reduced message type: ZSM1
Message type reference: MATMAS

Step 1:
Assign name
and description

IDoc Reduction Maintenance: Initial Screen
Activate change pointers
Deactivate change pointers

Reduced message type: ZSM1

Enter Description of Reduced message type

Reduced message type: ZSM1
Description: Sabines Reduced Material

Message type reference: MATMAS
Description: Material master

```
MATMASDI  Customer Field Distribution
└─ E1MARAM                        (*) Master material general data (MARA)
   ├── E1MARA1                    (-) Additional Fields for E1MARAM
   ├── E1MARADI                   (-) Discrete Industries Fields
   ├── E1MAKTM                    (*) Master material short texts (MAKT)
   ├── E1MARCM                    (-) Master material C segment (MARC)
   ├── E1MARMM                    (-) Master material units of measure (MARM)
   ├── E1MBEWM                    (-) Master material material valuation (MBEW)
   ├── E1MLGNM                    (-) Master material material data per warehouse number (MLGN
   ├── E1MVKEM                    (-) Master material sales data (MVKE)
   ├── E1MLANM                    (-) Master material tax classification (MLAN)
   └── E1MTXHM                    (-) Master material long text header
```

Step 2:
Select relevant
segments

Segment E1MARAM: Fields

Field	Text
MSGFN	(*) Function
MATNR	(*) Material Number
□ ERSDA	(-) Created On
□ ERNAM	(-) Name of Person who Created the Object
□ LAEDA	(-) Date of Last Change
□ AENAM	(-) Name of Person Who Changed Object
PSTAT	(*) Maintenance status
LVORM	(*) Flag Material for Deletion at Client Level
MTART	(*) Material Type
MBRSH	(*) Industry sector
MATKL	(*) Material Group
□ BISMT	(-) Old material number
MEINS	(*) Base Unit of Measure
□ BSTME	(-) Purchase Order Unit of Measure

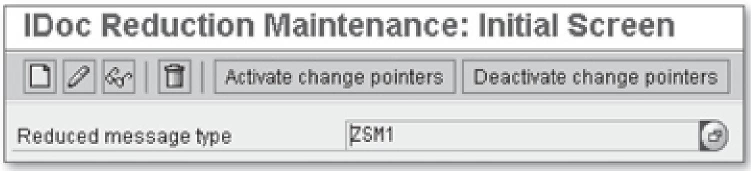
Step 3:
Select relevant fields
per segment

Figure 2.12 Creating a Reduced Message Type

Segments and fields that are green in the SAP system and displayed with an * behind the name are mandatory and can't be reduced. Fields and segments that are red or marked with – are optional and not selected; segments or fields that are white or marked with + are optional and selected in the respective reduced message type. You now specify which segments you want to have in addition to the mandatory segments by selecting the segment and clicking Select. As soon as you've activated a segment, you can select the fields within the segment you want to have in addition to the mandatory fields, and then click Select again.

Change pointers for reduced message types

If you want to generate change pointers for custom reduced message types, you can activate the generation in Transaction BD53 using the Activate Change Pointers button. Of course, you can also activate the change pointers for the reduced message type in Customizing of Transaction SALE. Here, you must make sure that you not only set the flags for generating change pointers but also copy all standard field assignments and mandatory field assignments, which is automatically done in Transaction BD53. In Transaction BD53, you can also disable the writing of change pointers using the Deactivate Change Pointers button (see Figure 2.13).


IDoc Reduction Maintenance: Initial Screen
Activate change pointers
Deactivate change pointers

Reduced message type: ZSM1

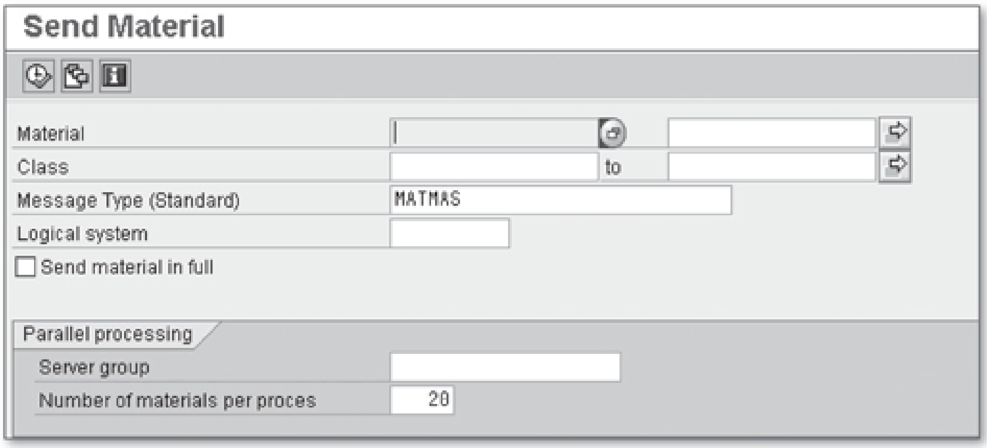
Figure 2.13 Activate the Change Pointer for Reduced Message Types

Directly Generating or Requesting Master Data

If you want to publish the creation or change of material master data without waiting for IDocs that are generated from change pointers, you can use Transaction BD10. In Table 7.5 of Chapter 7, Section 7.3, Transaction Code Overview, you can also find the transactions that belong to other master data.

Sending master data

Because master data usually offers the option of reduction, these transactions “expect” you to enter the message type you want to use for sending and the logical target systems you want to send to. Additionally, you can select the objects you want to generate IDocs for. However, this is only possible via the material numbers or the class membership of the object (see Figure 2.14).



Send Material

Material			
Class		to	
Message Type (Standard)	MATMAS		
Logical system			

□ Send material in full

Parallel processing

Server group	
Number of materials per proces	20

Figure 2.14 Targeted Sending of Material Master IDocs

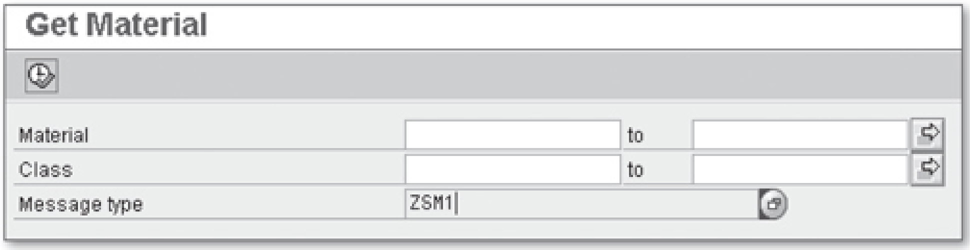
Sending the material

If you activated the Send Material in Full flag and also set the distribution of classification IDocs to the same partner, the system generates the classification IDoc that belongs to the material additionally to the material IDoc itself. The specifications for the parallel processing help you increase the performance if you send a high quantity of data, for instance, for initial data load. If you leave the Logical System field empty, the data is sent to all partners that are available in Transaction BD64. If a selection is made, Transaction BD64 checks whether the selected logical system is permitted as a receiver of the material master IDocs. If there is a positive result, the system sends the IDoc.

If you are the receiver of master data IDocs and know that the sending system has changed or has newly created data, you can request a corresponding master data IDoc. The name of the appropriate message type starts like the master data IDoc but uses another abbreviation at the end, that is, `FET` (for "fetch") instead of `MAS`. For example, the name is `MATMAS` for the message type of the material master IDocs and `MATFET` for the fetch IDoc.

Fetch IDoc

These *fetch IDocs* must be maintained as usual in the distribution model (see Figure 1.4 in Chapter 1, Section 1.3, Differentiation of ALE and EDI) — just in the other direction; here, the partner that receives the master data IDoc sends the fetch IDoc. Fetch IDocs always transfer the object key for which master data IDocs are requested and the message type that is supposed to be used to send the data. You can use Transaction BD11 to "get" material masters (see Figure 2.15).



Get Material

Material ______ to ______

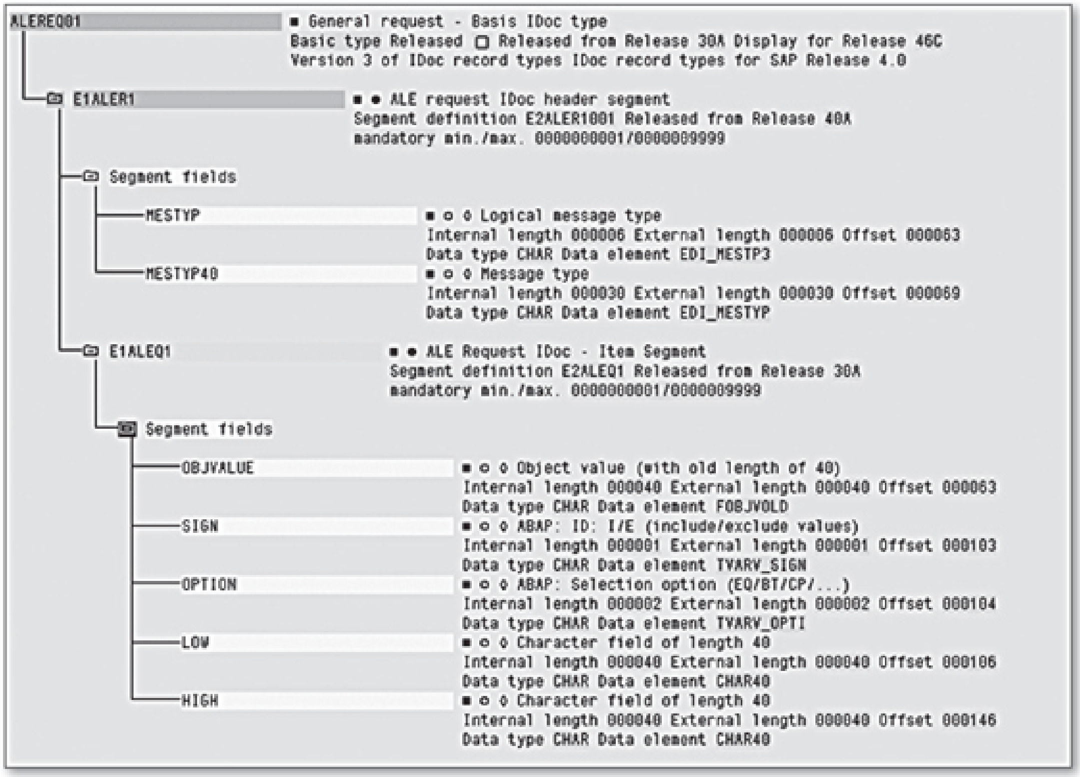
Class ______ to ______

Message type ZSM1

Figure 2.15 Requesting the Material Master IDoc

"ALEREQ01" IDoc type

The same ALEREQ01 IDoc type is assigned to all fetch message types. It contains the segments shown in Figure 2.16.

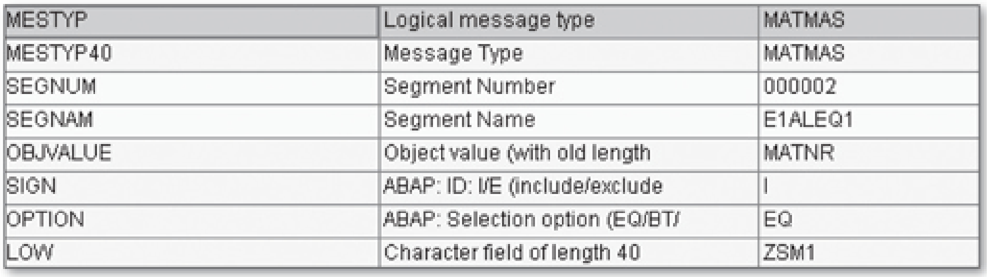


```
ALEREQ01                ■ General request - Basis IDoc type
                        Basic type Released □ Released from Release 30A Display for Release 46C
                        Version 3 of IDoc record types IDoc record types for SAP Release 4.0
  E1ALER1                  ■ ● ALE request IDoc header segment
                           Segment definition E2ALER1001 Released from Release 40A
                           mandatory min./max. 0000000001/0000009999
    Segment fields
      MESTYP                     ■ ○ ◊ Logical message type
                                 Internal length 000006 External length 000006 Offset 000063
                                 Data type CHAR Data element EDI_MESTP3
      MESTYP40                   ■ ○ ◊ Message type
                                 Internal length 000030 External length 000030 Offset 000069
                                 Data type CHAR Data element EDI_MESTYP
    E1ALEQ1                    ■ ● ALE Request IDoc - Item Segment
                               Segment definition E2ALEQ1 Released from Release 30A
                               mandatory min./max. 0000000001/0000009999
      Segment fields
        OBJVALUE                   ■ ○ ◊ Object value (with old length of 40)
                                   Internal length 000040 External length 000040 Offset 000063
                                   Data type CHAR Data element FOBJVOLD
        SIGN                       ■ ○ ◊ ABAP: ID: I/E (include/exclude values)
                                   Internal length 000001 External length 000001 Offset 000103
                                   Data type CHAR Data element TVARV_SIGN
        OPTION                     ■ ○ ◊ ABAP: Selection option (EQ/BT/CP/...)
                                   Internal length 000002 External length 000002 Offset 000104
                                   Data type CHAR Data element TVARV_OPTI
        LOW                        ■ ○ ◊ Character field of length 40
                                   Internal length 000040 External length 000040 Offset 000106
                                   Data type CHAR Data element CHAR40
        HIGH                       ■ ○ ◊ Character field of length 40
                                   Internal length 000040 External length 000040 Offset 000146
                                   Data type CHAR Data element CHAR40
```

Figure 2.16 "ALEREQ01" IDoc Type

Example of "MATFET" IDoc

The actually sent fetch message contains information on the message type that is expected as the response, the short and the long name (before or after Release 4.0), and the keys of elements that are supposed to be sent as a response. Figure 2.17 shows a `MATFET` IDoc that request the `ZSM1` material. For the `MATMAS` IDoc, both the long and the short name is "MATMAS" because this is a very old message type.



MESTYP	Logical message type	MATMAS
MESTYP40	Message Type	MATMAS
SEGNUM	Segment Number	000002
SEGNAM	Segment Name	E1ALEQ1
OBJVALUE	Object value (with old length	MATNR
SIGN	ABAP: ID: I/E (include/exclude	I
OPTION	ABAP: Selection option (EQ/BT/	EQ
LOW	Character field of length 40	ZSM1

Figure 2.17 MATFET IDoc

Because master data is usually exchanged between different systems of the same enterprise, logical systems are used as partners like they are used in ALE.

2.1.2 Message Control

Message control

The message control is a standard function of SAP, which triggers a data transfer for all transaction data that is supposed to be received by other enterprises, too. This can be done using a printer, fax, or an IDoc. For processing using IDocs, you can use *transmission medium 6* for processing using partner functions, and you can use *transmission medium A* for processing using logical systems. All the required settings for the message control are in Transaction NACE.

You use the *condition technique* to specify when you generate which messages in which way. The key for the conditions is composed of the application you're in (e.g., "EF" for purchasing), the output type you want to generate (e.g., "NEU" for a purchase order), and the partner role everything will be sent to (e.g., partner "vendor" in its role as goods vendor).

"EDI_PROCESSING" function

In *message determination*, the actual message is generated using the `RSNAST00` report. The `EDI_PROCESSING` function that is used here transfers the message as an EDI message using the IDoc; it belongs to transmission medium 6. Depending on the system setting, you can call the `RSNAST00` report directly upon saving the document or at regular intervals as a batch job.

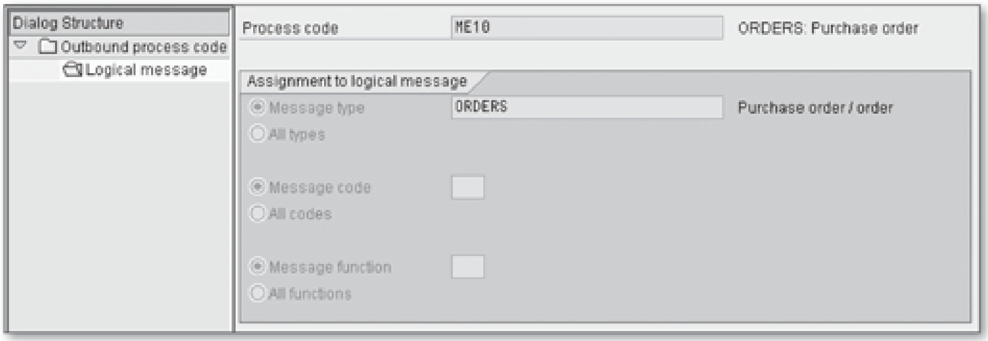
In *message control*, an EDI communication is usually assumed so that you work with partners and not with logical systems.

Outbound partner profile

The information on how an IDoc is sent to the receiver (e.g., via RFC or file) and whether an EDI subsystem is supposed to be used if necessary, is set in the *outbound partner profile* in Transaction WE20 both for the communication with partners and for the communication with logical systems.

Process code

Here, you also specify which IDoc type is supposed to be used. If you work with the message control, in the outbound partner profile under the Message Control tab, you specify which process code (and which underlying function module) will be used to populate the IDoc data. You can find the valid process codes for the respective output type in Transaction WE41 (Figure 2.18).



Process code ME10 ORDERS: Purchase order

Assignment to logical message

● Message type ORDERS Purchase order / order
○ All types

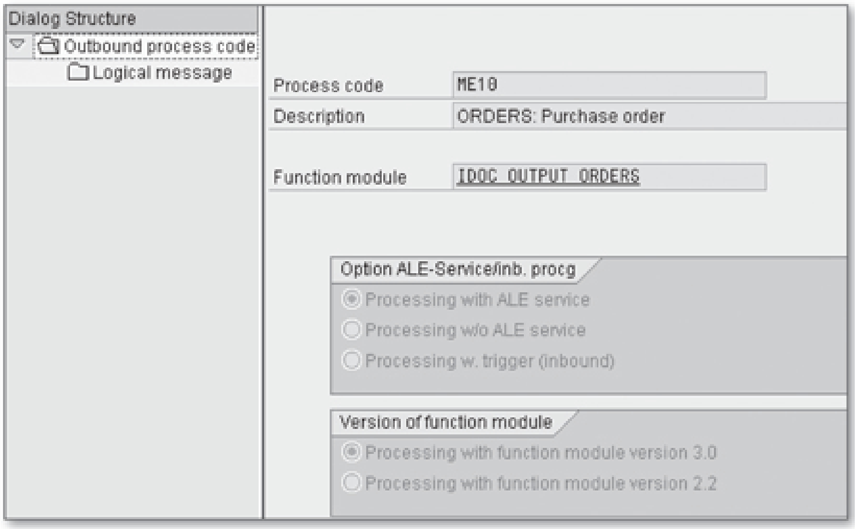
● Message code
○ All codes

● Message function
○ All functions

Figure 2.18 Assignment of the Process Code to the Message Type

Message codes and message functions

You can also specify the optional *message codes* and *message functions*, which you know from the partner profiles, to be able to use different process codes for updating the IDocs. Message codes and functions are freely selectable, and you don't need to adhere to naming rules. However, this also means that no input help is provided in Transaction WE20, so you must ensure the correct spelling of the names yourself. In the details of the sample process code for the `ORDERS` generation, which is shown in Figure 2.19, you can view the link to the associated function module.



Dialog Structure
- Outbound process code
  - Logical message

Process code: ME10
Description: ORDERS: Purchase order
Function module: IDOC_OUTPUT_ORDERS

Option ALE-Service/Inb. procg
- ● Processing with ALE service
- ○ Processing w/o ALE service
- ○ Processing w. trigger (inbound)

Version of function module
- ● Processing with function module version 3.0
- ○ Processing with function module version 2.2

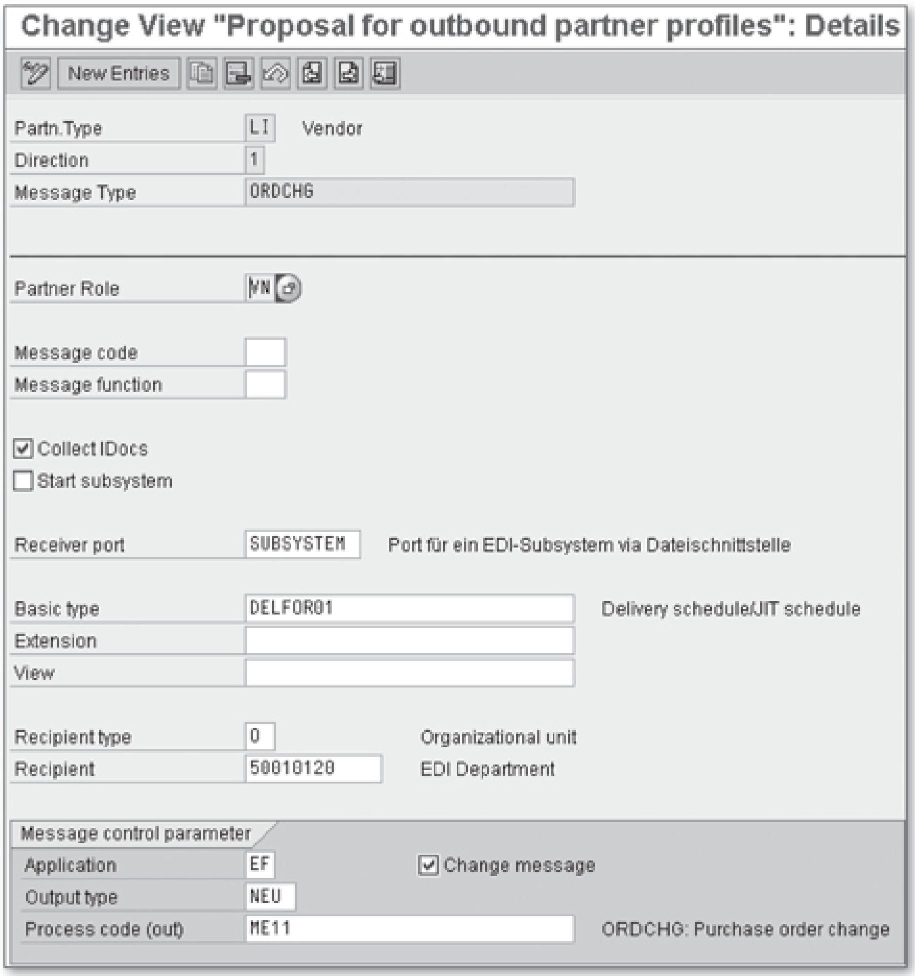
Figure 2.19 Assignment of the Process Code to the Function Module

ALE services

Under Option ALE-Service/Inb. Procg, you can select whether the ALE services are supposed to be used or not. The ALE services represent options of IDoc manipulation using filters and rules. Chapter 4, Section 4.1, Customizing, discusses the ALE services topic in more detail.

Default values

Because you often require a lot of partner profiles — especially if you work with partners — you have the option to create templates to save some time. The partner profile from Transaction WE20 can be created from this template, so you don't need to create it manually. Transaction WE24 is the transaction for the template in the outbound processing (see Figure 2.20). Transaction WE27 is the corresponding transaction for the inbound processing.



Change View "Proposal for outbound partner profiles": Details

Partn.Type: LI Vendor
Direction: 1
Message Type: ORDCHG

Partner Role: VN

Message code:
Message function:

☑ Collect IDocs
□ Start subsystem

Receiver port: SUBSYSTEM Port für ein EDI-Subsystem via Dateischnittstelle

Basic type: DELFOR01 Delivery schedule/JIT schedule
Extension:
View:

Recipient type: O Organizational unit
Recipient: 50010120 EDI Department

Message control parameter
Application: EF ☑ Change message
Output type: NEU
Process code (out): ME11 ORDCHG: Purchase order change

Figure 2.20 Default Values for Outbound Partner Profiles

2.1.3 Special Functions

Warehouse management example

In some cases, a specific business process can be implemented completely locally on an SAP system or distributed across multiple SAP or non-SAP systems. IDocs are only generated when processes are distributed across multiple systems, and the generation can be activated via the Customizing functions of the respective application. Warehouse management is an example of this direct IDoc generation. The default setting of warehouse management assumes that your warehouse is managed by your SAP system. If this isn't the case, you can activate the connection of your warehouse system via ALE in Customizing. This connection causes a `WMTORD` IDoc to be generated directly when you create a warehouse transport request to notify the external warehouse of what is supposed to be transported. However, these special cases can't be described in general but instead must be named and set up in cooperation with the individual end-user. Because all transactions that are required for such a special case are module-specific, they are not discussed further here, but just be aware that such special cases exist.

IDoc – BAPI interface

Within the scope of ALE scenarios, there can also be cases in which sample postings are implemented synchronously via BAPI, and the actual postings are implemented asynchronously via IDoc. Then, the BAPI is created on the development side, and the appropriate IDoc is generated using Transaction BDBG. You can also use this transaction if SAP doesn't provide an IDoc for a BAPI and you still require it for a distribution scenario that isn't provided for by SAP. Again in this case, you must observe the naming rules for customer objects.

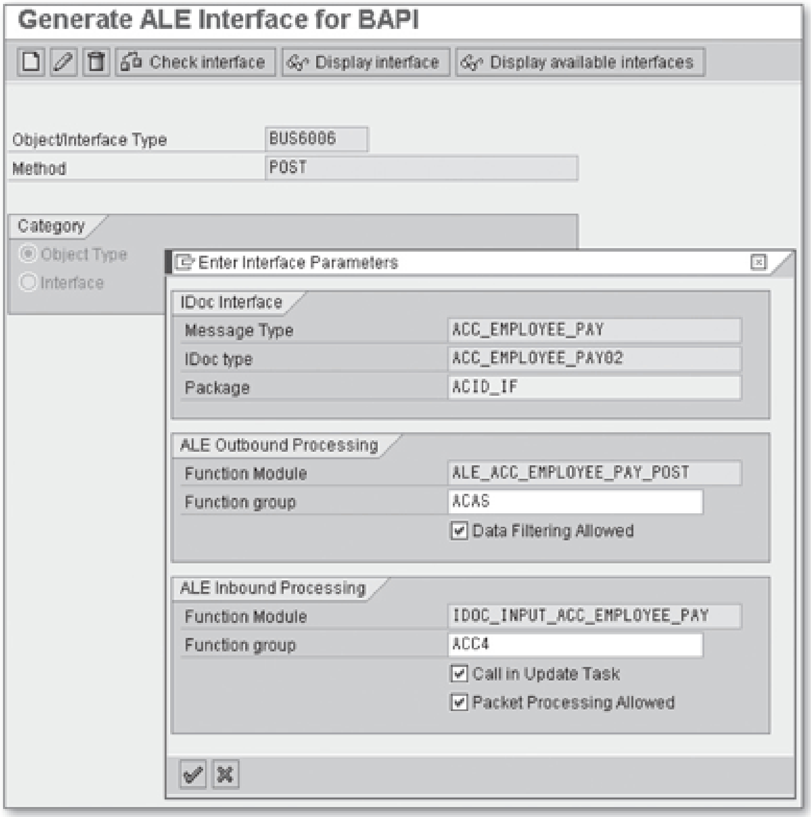
Figure 2.21 shows an IDoc generated by SAP in the SAP namespace. In the IDoc Interface tab, you can view the names for the message type and the IDoc type; in the ALE Outbound Processing tab, you can view the function group in which the IDoc modules are located as well as the name of the module that generates the IDoc; and in the ALE Inbound Processing tab, you can view the name of the module that extracts the IDoc in the receiving system and triggers the update.

BAPI processing

The process of generating an IDoc from a BAPI is as follows:

1. The sending system wants to call the BAPI and checks whether this is supposed to be carried out locally or remotely.
2. If the call is remote and is supposed to be carried out transactionally, the function module that is generated in Transaction BDBG is called in the sending system. This function module transfers the transfer parameters of the BAPI to the IDoc format.
3. After you've made the settings in the customer distribution model and in Transaction WE20, this generated IDoc is transferred to the receiving system.
4. In the receiving system, the `BAPI_IDOC_INPUT1` function module is called using the `BAPI` process code or `BAPI_IDOC_INPUTP` using the `BAPP` process code; this depends on whether one or more data records are received simultaneously. These function modules call the inbound function module that is generated in Transaction BDBG, which extracts

the IDoc and uses the transferred data to call the original BAPI that carries out the actual update.



Generate ALE Interface for BAPI

Object/Interface Type	BUS6006
Method	POST

Category
● Object Type
○ Interface

Enter Interface Parameters

IDoc Interface

Message Type	ACC_EMPLOYEE_PAY
IDoc type	ACC_EMPLOYEE_PAY02
Package	ACID_IF

ALE Outbound Processing

Function Module	ALE_ACC_EMPLOYEE_PAY_POST
Function group	ACAS

☑ Data Filtering Allowed

ALE Inbound Processing

Function Module	IDOC_INPUT_ACC_EMPLOYEE_PAY
Function group	ACC4

☑ Call in Update Task
☑ Packet Processing Allowed

Figure 2.21 IDoc Interface to a BAPI

Asynchronous BAPIs

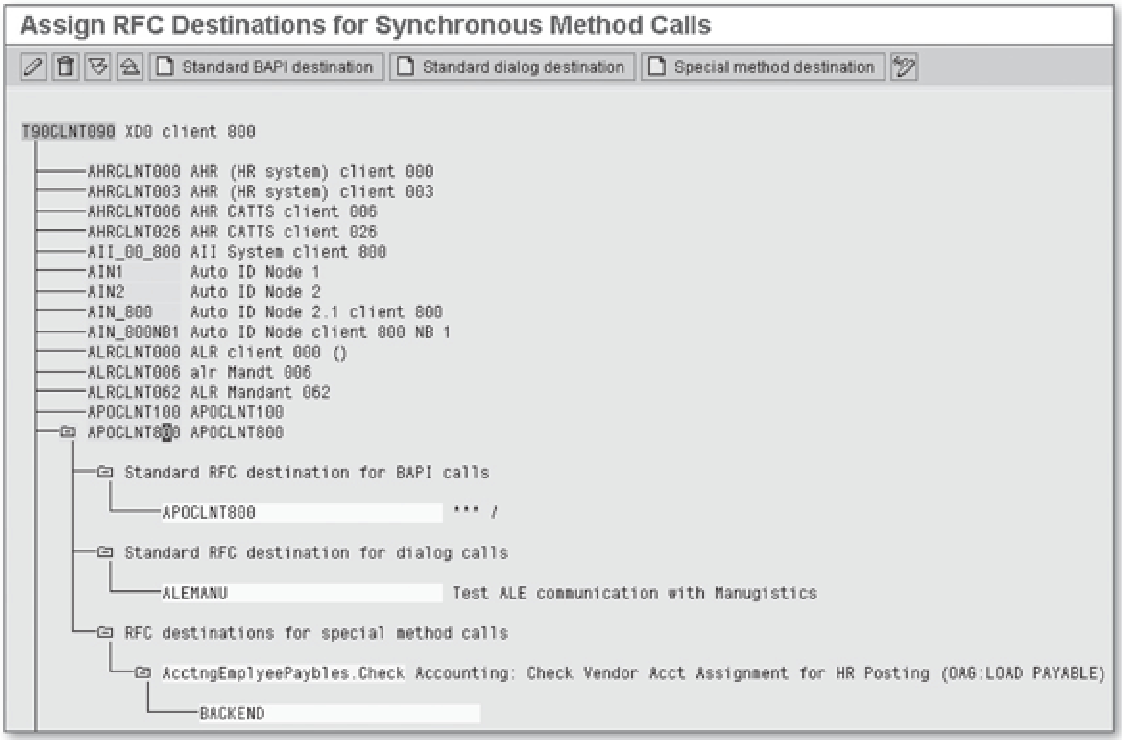
Because both cases represent an ALE scenario, you maintain the customer distribution model for BAPIs and IDocs. For the BAPIs, you enter the methods that must be processed both synchronously and asynchronously via IDocs. Additionally, you require partner profiles for the transactional case as usual. If you have these partner profiles generated from the distribution model, the system automatically knows for which BAPIs there are message types and thus for which it requires a partner profile.

Determining the target system for synchronous BAPIs

Use Transaction BD97 to maintain the destination for the synchronous BAPI call in a remote system. This can be done both generally for all method calls and only for special BAPIs and for dialog calls. You need the dialog calls if you want to work with the IDoc tracing in Transaction BD87. The distinction with regard to dialogs is made for security reasons.

Because a dialog user must be used in the RFC destination, this user should have only a few authorizations.

Figure 2.22 shows an example for each of the three cases.


Assign RFC Destinations for Synchronous Method Calls
Standard BAPI destination
Standard dialog destination
Special method destination

```
T90CLNT090 XD0 client 800
 |
 |-------AHRCLNT000 AHR (HR system) client 000
 |-------AHRCLNT003 AHR (HR system) client 003
 |-------AHRCLNT006 AHR CATTS client 006
 |-------AHRCLNT026 AHR CATTS client 026
 |-------AII_00_800 AII System client 800
 |-------AIN1       Auto ID Node 1
 |-------AIN2       Auto ID Node 2
 |-------AIN_800    Auto ID Node 2.1 client 800
 |-------AIN_800NB1 Auto ID Node client 800 NB 1
 |-------ALRCLNT000 ALR client 000 ()
 |-------ALRCLNT006 alr Mandt 006
 |-------ALRCLNT062 ALR Mandant 062
 |-------APOCLNT100 APOCLNT100
 |----□  APOCLNT800 APOCLNT800
 |    |
 |    |--□ Standard RFC destination for BAPI calls
 |    |  |
 |    |  |------APOCLNT800                 *** /
 |    |
 |    |--□ Standard RFC destination for dialog calls
 |    |  |
 |    |  |------ALEMANU                    Test ALE communication with Manugistics
 |    |
 |    |--□ RFC destinations for special method calls
 |       |
 |       |--□ AcctngEmplyeePaybles.Check Accounting: Check Vendor Acct Assignment for HR Posting (OAG:LOAD PAYABLE)
 |          |
 |          |------BACKEND
```

Figure 2.22 Customizing for Synchronous BAPI Calls in ALE Scenarios

The standard destination for BAPI calls for the APOCLNT800 logical system is APOCLNT800, the destination for the dialog calls is called ALEMANU, and the BACKEND destination is used for the AcctngEmplyeePaybles.Check method only.

2.2 Use of Logical Systems in the Message Control

"ALE_ PROCESSING" subroutine

Transaction data may be exchanged within ALE scenarios. For example, this could be the scenario of central sales/decentralized shipping. Here, purchase orders, deliveries, and invoices are exchanged between plants of the same enterprise. For this case, in message control, you simply use the customer or vendor as the partner. Instead of "6" for the EDI processing, however, you enter "A" for the ALE processing as the transmission medium for the found message. As a result, the system no longer uses the `EDI_PROCESSING` subroutine and instead uses the `ALE_PROCESSING` subroutine.

Message control and ALE

You then maintain a customer distribution model as usual and use a logical system as the sender and a logical system as the receiver of the message. After the message control within the application, the transmission medium A reads the customer distribution model and replaces the partner found in the message control with the logical system found in the model. For this purpose, you need the assignment of output types to message types. They are available in the process codes in the outbound processing in Transaction WE41 and in the settings that you made for the message control in the outbound partner profile in Transaction WE20.

Uniqueness of the assignment

The data from Transaction WE20 is evaluated in the `ALE_PROCESSING` subroutine. Because the subsequent receiver isn't yet determined in this first step, the system searches using the "Logical System" partner type and the output type only. If more than one entry is found in Transaction WE20, for example, `NEU - ORDERS`, which would be the standard, and `NEU - ZSMORD` for a self-programmed message type, `ALE_PROCESSING` cancels with an error. You establish the required uniqueness by using your own output types for your own message types.

The search via the "Logical System" partner type and the partner type results in the benefit of a reduced number of partner profiles. Then you can just have a partner profile for the logical system instead of a partner profile for each vendor or customer.

2.3 Summary

In this chapter, we've taken a closer look at the creation of IDocs. You've learned which methods for creation exist and how they are related to the different needs of different processes or data types. The decision as to which one should be used is made by SAP.

Now let's take it a step further regarding our work with IDocs. The next chapter will teach you how to test the creation and the posting of IDocs.

"Anything that can go wrong will go wrong." (Murphy's Law)
To ensure that as little as possible goes wrong in your production system, SAP provides a variety of test tools, each of which are described in this chapter.

3 Test Tools

Testing without a partner

A business process that spans several systems can't be fully tested and accepted unless all of the partners test the process. However, basic settings usually undergo a preliminary test, without any involvement by your partner. During the definition phase, it may also be necessary for you to determine exactly which fields are contained in an IDoc, and of these fields, which are needed for a particular scenario. For this basic test, SAP has delivered a test environment in which systems can generate and post IDocs without having a suitable SAP system group. These IDocs can then be used as sample IDocs or templates for the partner.

Inbound and outbound processing

You can manually create IDocs and post them to your own SAP system, as well as make IDocs available for outbound processing. All of the test tools are stored under the area menu in Transaction WEDI. Figure 3.1 shows the seven test transactions provided there.

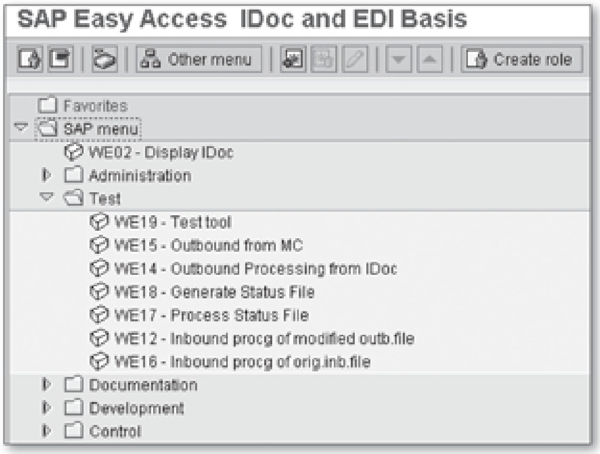

SAP Easy Access IDoc and EDI Basis
Other menu
Create role
Favorites
SAP menu
WE02 - Display IDoc
Administration
Test
WE19 - Test tool
WE15 - Outbound from MC
WE14 - Outbound Processing from IDoc
WE18 - Generate Status File
WE17 - Process Status File
WE12 - Inbound procg of modified outb.file
WE16 - Inbound procg of orig.inb.file
Documentation
Development
Control

Figure 3.1 Test Transactions in the Area Menu of Transaction WEDI

Part of these transactions is also available in the menus of those applications that frequently work with IDocs. However, this chapter will primarily focus on using individual transactions, rather than showing them in a specific module. Note that most values entered for test purposes are case-sensitive.

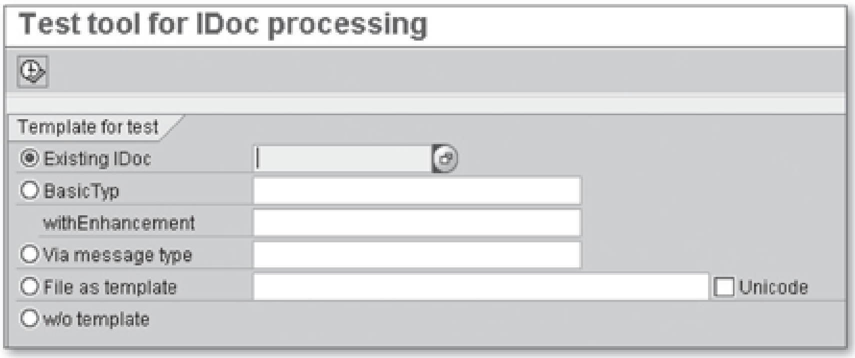
3.1 Individual IDocs

Transaction WE19

Transaction WE19 is the test tool for individual IDocs. You can choose whether you want to use a finished IDoc as a template or whether you want to create a completely new IDoc based on a message type or IDoc type. You can also use an existing file. In this case, however, you must use a flat file because XML files aren't supported.

IDoc template

Transaction WE19 is particularly helpful if it's not yet clear which IDoc fields must be filled and which values must be specified. You can run through several variants until you've determined the field information relevant for your set of tasks. Figure 3.2 provides an overview of the initial screen as well as the selection options available in this transaction.



Test tool for IDoc processing

Template for test

● Existing IDoc

○ BasicTyp

withEnhancement

○ Via message type

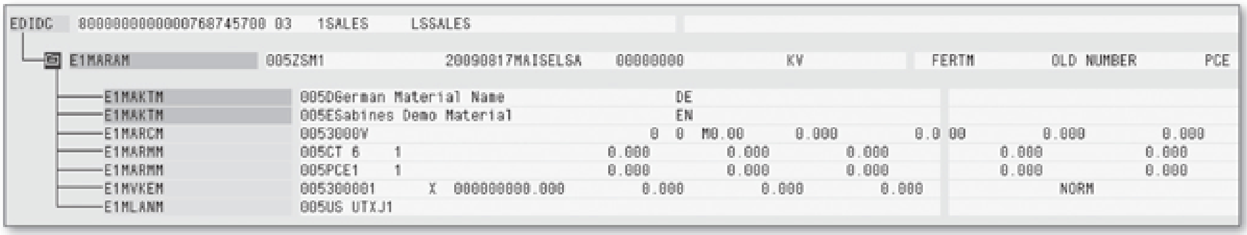
○ File as template □ Unicode

○ w/o template

Figure 3.2 Test Tool for Individual IDocs

Manual test IDoc

After you've called an IDoc, you can structure it so that it fulfills your test requirements. Each segment can be edited separately. While you're in edit mode, you also can clearly see which field you're working in without having to count the fields. You can also add more segments. If you're using the template option, the system checks some (unfortunately, not all) of the required properties of the IDoc type or message type that you've selected. All of the variants are checked to determine whether mandatory segments are missing or whether the segment in use exists in the segment definition in the SAP system. Figure 3.3 shows the segments of an example IDoc in Transaction WE19.

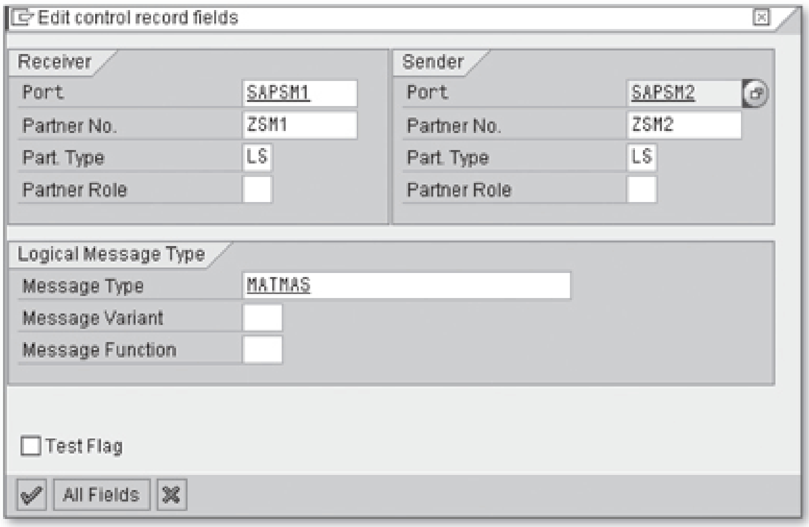


```
EDIDC   0000000000000768745700 03   1SALES     LSSALES
   E1MARAM          005ZSM1            20090817MAISELSA    00000000          KV        FERTM        OLD NUMBER      PCE
      E1MAKTM       005DGerman Material Name                          DE
      E1MAKTM       005ESabines Demo Material                         EN
      E1MARCM       0053000V                                      0  0  M0.00       0.000         0.0 00         0.000         0.000
      E1MARMM       005CT 6     1                            0.000           0.000         0.000           0.000         0.000
      E1MARMM       005PCE1     1                            0.000           0.000         0.000           0.000         0.000
      E1MVKEM       005300001        X  000000000.000          0.000           0.000         0.000              NORM
      E1MLANM       005US UTXJ1
```

Figure 3.3 View of an IDoc in Transaction WE19

Editing the control record

You can now edit both the control record and the data records and, for example, send an IDoc to a receiver other than the original receiver. As you can see in Figure 3.4, only the most important fields in the control record are displayed initially. However, if you also want to view or edit the other fields, simply choose All Fields.



Edit control record fields

Receiver

Port	SAPSM1
Partner No.	ZSM1
Part. Type	LS
Partner Role	

Sender

Port	SAPSM2
Partner No.	ZSM2
Part. Type	LS
Partner Role	

Logical Message Type

Message Type	MATMAS
Message Variant	
Message Function	

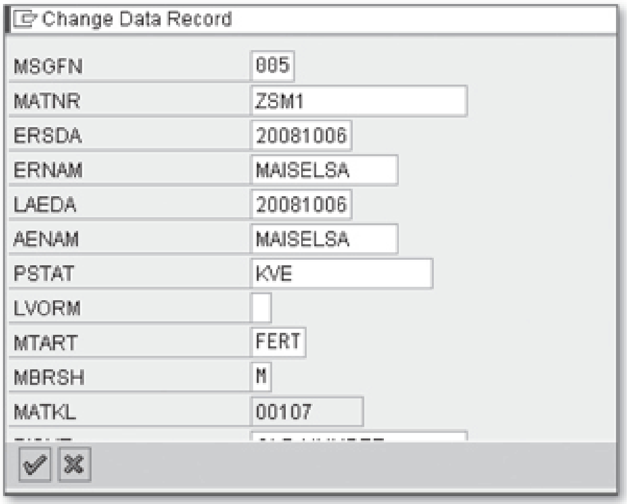
□ Test Flag

Figure 3.4 Editing the Control Record

Test flag

The *test flag* plays a special role if you actually send test IDocs to other systems. For every single partner profile within inbound processing in SAP systems, you can determine whether the Test Flag will apply to real or test IDocs. If you set the Test Flag, you can receive the IDoc, save it to the database, and view its contents in Transaction BD87, but you can't post it. Other systems that exchange IDocs or EDI messages usually have similar built-in security mechanisms for test purposes.

Each time you double-click a data record, you get a list of all possible fields for this segment. You also can scroll down through the entire data record (see Figure 3.5). If you're not working with a template, you enter the field contents that you require here. If you're working with a template, the values of the template IDoc are prefilled, and you can change them as required.

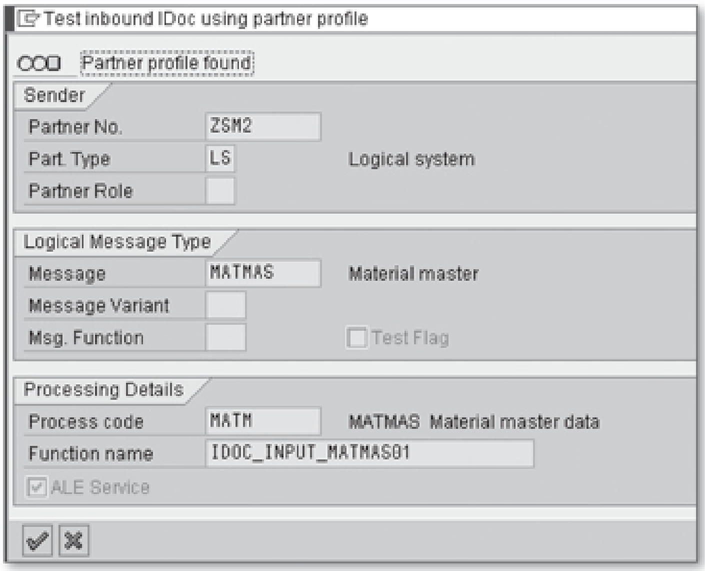

Change Data Record

MSGFN	005
MATNR	ZSM1
ERSDA	20081006
ERNAM	MAISELSA
LAEDA	20081006
AENAM	MAISELSA
PSTAT	KVE
LVORM	
MTART	FERT
MBRSH	M
MATKL	00107

Figure 3.5 Editing a Data Record

Processing the test IDoc

Now that you've created your complete IDoc, it can be processed. In other words, it can be tested with the relevant standard settings for inbound and outbound processing. For inbound processing, additional test options are available, either directly by entering a function module or indirectly by using a file (we'll describe this in greater detail later). Figure 3.6 shows *standard inbound processing*. All of the data is obtained from a partner profile, and it can't be changed. If no partner profile is found, you can create an IDoc, but you can't post it.



Test inbound IDoc using partner profile

Partner profile found

Sender

Partner No.	ZSM2	
Part. Type	LS	Logical system
Partner Role		

Logical Message Type

Message	MATMAS	Material master
Message Variant		
Msg. Function		□ Test Flag

Processing Details

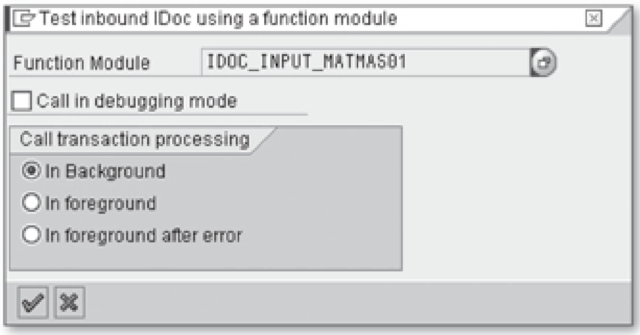
Process code	MATM	MATMAS Material master data
Function name	IDOC_INPUT_MATMAS01	

☑ ALE Service

Figure 3.6 Testing Standard Inbound Processing

Test using a function module

When you use the Test Using a Function Module option, it's possible to directly specify a function module for IDoc processing (see Figure 3.7). The test IDoc is then processed with this function module, irrespective of whether or not a suitable partner profile exists. This enables testing to be completed prior to the final settings, which are sometimes implemented by others. You can also test your own developments here and navigate directly to debugging, if necessary. If there are several process codes and several function modules for the same message type, you can also run through all of them here. You can then use the results to determine the most relevant process code for your purposes.



Test inbound IDoc using a function module

Function Module IDOC_INPUT_MATMAS01

□ Call in debugging mode

Call transaction processing

● In Background

○ In foreground

○ In foreground after error

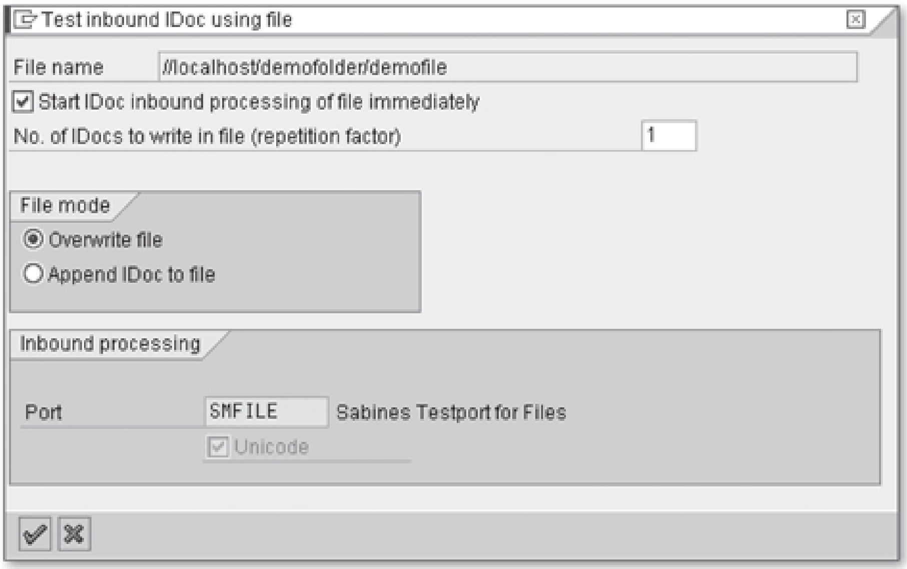
Figure 3.7 Extended Test in Inbound Processing

Test IDoc as a file

The two scenarios described previously — a test using the standard settings or a test using a function module — work directly in the SAP system. Frequently, however, you also have to import partner files into your business processes. To prevent errors from occurring, your partner will most likely appreciate a *sample file* whose structure can be used during live implementation. By using a file to test inbound processing, you can write such a file with several IDocs and then process it directly.

Testing inbound processing

Writing a file with a repeat factor in append mode can also be used as the basis for mass tests for performance analysis purposes. Consequently, IDocs are generated from your data as often as permitted by the repeat factor and then placed in a large shared file. To use this function, you must enter a valid file port for which you can activate or deactivate Unicode (see Figure 3.8). However, you must specify the file path yourself because it can't be obtained from the port information.


Test inbound IDoc using file

File name //localhost/demofolder/demofile

☑ Start IDoc inbound processing of file immediately

No. of IDocs to write in file (repetition factor) 1

File mode

● Overwrite file

○ Append IDoc to file

Inbound processing

Port SMFILE Sabines Testport for Files

☑ Unicode

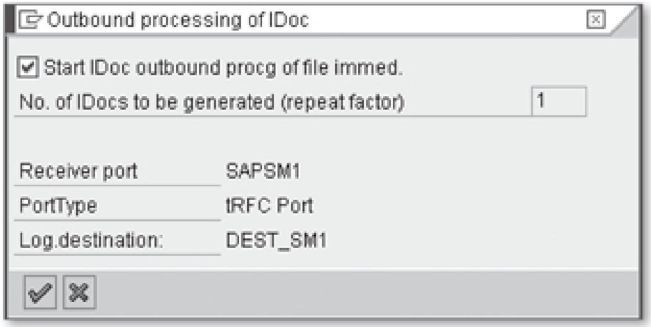
Figure 3.8 Testing Inbound Processing Using a File

Status values of the test IDoc

No matter which of these three test methods you use to create an inbound IDoc, the IDoc is stored with an additional status record that always indicates that this IDoc was not really created but rather created using test Transaction WE19. However, the real changes are written to the database, so you must carefully consider whether you want to permit the use of this test transaction in a production system. Figure 3.14 in Section 3.3, Processing Status Files, shows the status values of an IDoc created using test Transaction WE19. In outbound processing, the test status value is *42: IDoc was created by test transaction*.

Testing outbound processing

When testing outbound processing, you can also send several IDocs at once. However, you don't have the previously mentioned additional options of debugging or working without suitable partner profiles here. You can only use standard outbound processing. As you can see in Figure 3.9, all of the information you require (apart from the repeat factor) is obtained from the partner profiles. These settings are then used to send an IDoc to a partner.



Outbound processing of IDoc

☑ Start IDoc outbound procg of file immed.

No. of IDocs to be generated (repeat factor)	1
Receiver port	SAPSM1
PortType	tRFC Port
Log.destination:	DEST_SM1

Figure 3.9 Testing Standard Outbound Processing

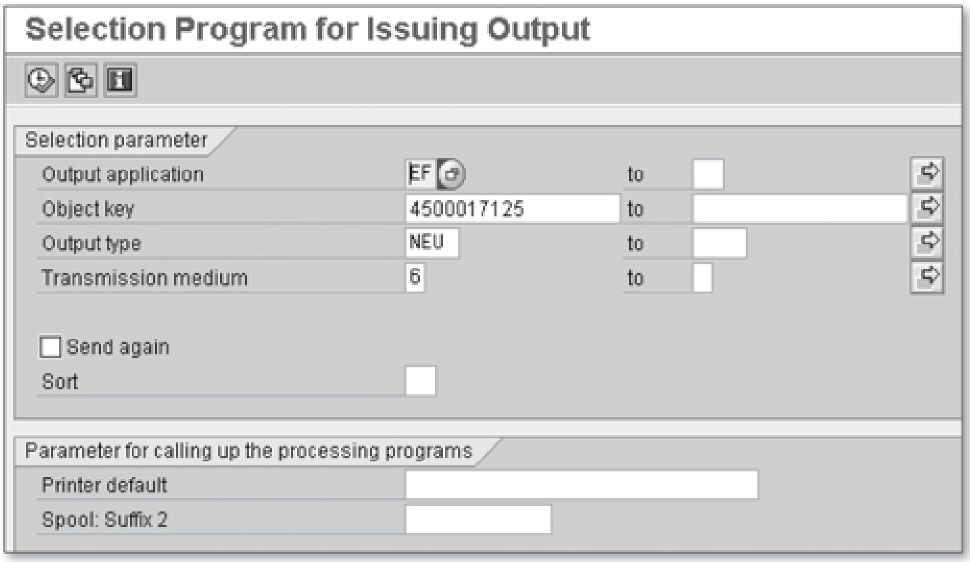
3.2 Testing Processing of Multiple IDocs

Only one IDoc was created when we used the test methods described in the previous section. However, you also have the option of creating multiple IDocs at once. We'll now describe this option in detail.

3.2.1 Message Control

Test transaction WE15 and report "RSNAST00"

If message control is used to generate IDocs, the application frequently configures the parameter settings there. The actual IDocs are first generated using a scheduled report (and not as soon as the document is posted). Frequently, this process can't be adapted for test purposes, and, as a result, you must manually start the relevant report each time you want to test an IDoc. To make life a little easier for you, Transaction WE15 is available for generating IDocs from message control records. This transaction directly references the report `RSNAST00`, and it's listed in your EDI overview menu WEDI. Figure 3.10 shows a purchasing example that uses the standard message type delivered by SAP.



Selection Program for Issuing Output

Selection parameter

Output application	EF	to	
Object key	4500017125	to	
Output type	NEU	to	
Transmission medium	6	to	

☐ Send again

Sort

Parameter for calling up the processing programs

Printer default

Spool: Suffix 2

Figure 3.10 Test Transaction for Message Control

The main difference between this test transaction and all other test transactions is that an IDoc that was actually generated by an application is sent earlier than planned, instead of waiting for the system to execute the report `RSNAST00` by default. A new IDoc isn't generated (the original job scheduled later doesn't generate this IDoc again). In status monitoring, you can't see how this IDoc was generated because it simply starts as normal with status *01: IDoc generated*.

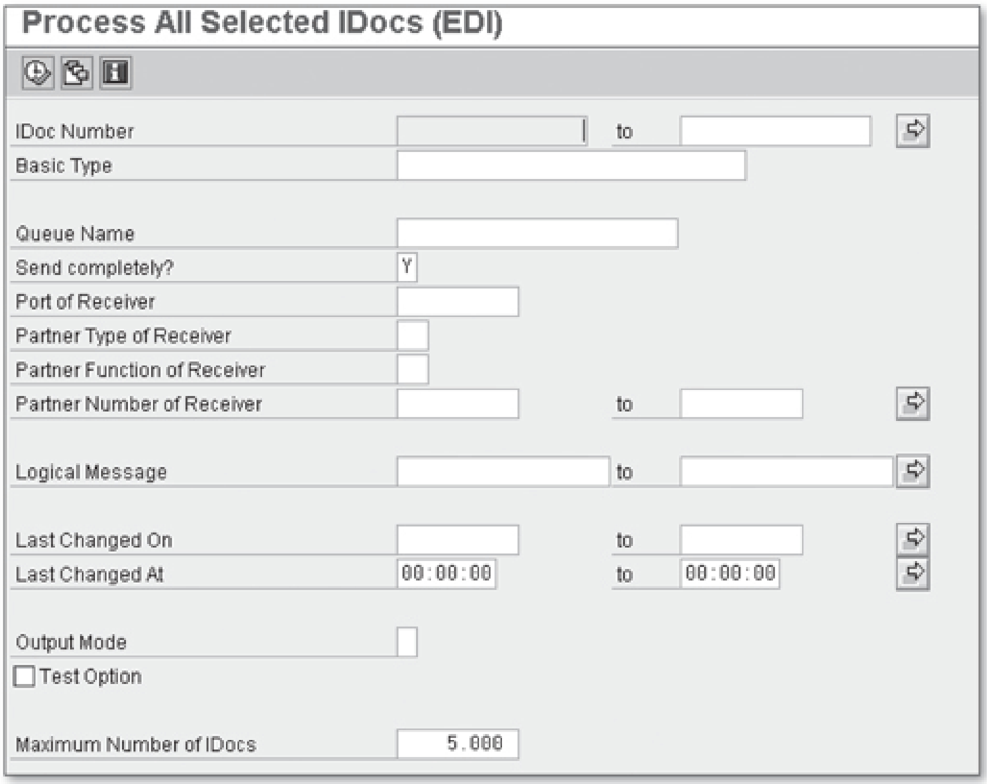
3.2.2 Sending Ready-for-Dispatch IDocs

Advanced dispatch of finished IDocs

Transaction WE14 is the only transaction that can be used to anticipate a job that is otherwise scheduled on a regular basis. Here, ready-for-dispatch IDocs (status *30*) are dispatched before the regular "date," if you don't want to wait until then. However, you can also set the test flag here, so that your partner knows that he should not process the IDoc. Figure 3.11 shows how you can use Transaction WE14 to specify which IDocs you want to create.

Standard report "RSEOUT00"

Similarly, this transaction doesn't generate a new IDoc. Consequently, in Transaction BD87 (IDoc Monitor), you can't identify a difference to the IDocs that are dispatched regularly via a background job because Transaction WE14 also provides special access to the standard report RSEOUT00.



Process All Selected IDocs (EDI)

IDoc Number		to	
Basic Type			
Queue Name			
Send completely?	Y		
Port of Receiver			
Partner Type of Receiver			
Partner Function of Receiver			
Partner Number of Receiver		to	
Logical Message		to	
Last Changed On		to	
Last Changed At	00:00:00	to	00:00:00
Output Mode			
☐ Test Option			
Maximum Number of IDocs	5.000		

Figure 3.11 Selection of IDocs to Be Sent

3.2.3 Files

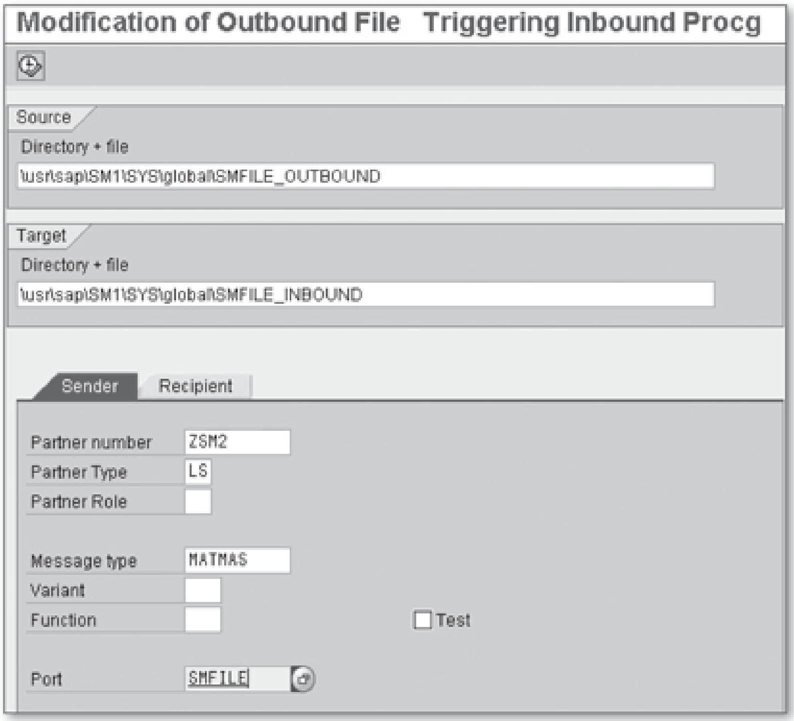
In addition to testing individual IDocs, you can also process complete files with IDoc data. You can create these files or they can be transferred from a partner. Here, we'll assume that all of the data in the partner file is correct and simply needs to be imported. This is done using Transaction WE16. Once again, you only have to enter the file name, file path, and port, as well as set the Unicode flag on the input screen.

Editing the control record

In the case of files that you create yourself, you can either create a file in an editor, which is a very laborious task, or you can use a test tool in outbound processing to create a file that will contain the data you require. In the latter scenario, however, your logical system is specified as the sender (and not the receiver) in the control record. Therefore, the control record can't be used for your purposes. SAP provides Transaction WE12 here, which you can use to obtain the business data from the file specified as well as manually enter the control record data. The resulting file is then temporarily stored again.

Changing the sender

You can, for example, make this file available to a partner as a template. Figure 3.12 shows part one of the required information on the Sender tab page.



Modification of Outbound File Triggering Inbound Procg

Source

Directory + file: \usr\sap\SM1\SYS\global\SMFILE_OUTBOUND

Target

Directory + file: \usr\sap\SM1\SYS\global\SMFILE_INBOUND

Partner number: ZSM2

Partner Type: LS

Partner Role:

Message type: MATMAS

Variant:

Function:

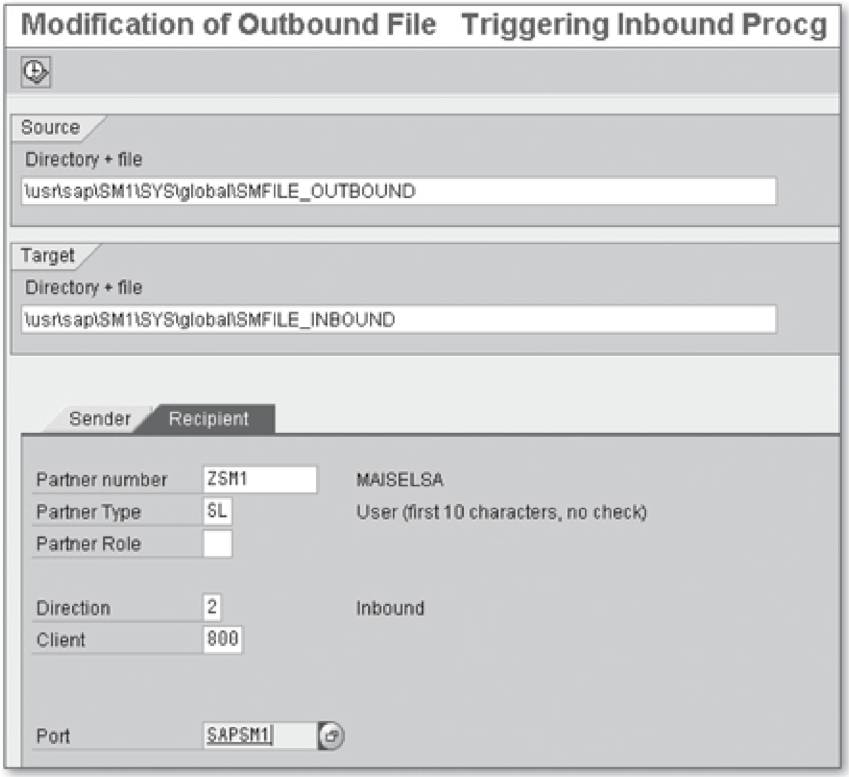
□ Test

Port: SMFILE

Figure 3.12 Transfer Data in Sending System

Changing the receiver

On the Recipient tab page, you enter part two of the information required to override the control record (see Figure 3.13). IDocs generated using Transaction WE12 also obtain the value *74: Inbound IDoc from test transaction* as their first status, so that it's always clear that they are for test purposes only.



Modification of Outbound File Triggering Inbound Procg

Source

Directory + file: \usr\sap\SM1\SYS\global\SMFILE_OUTBOUND

Target

Directory + file: \usr\sap\SM1\SYS\global\SMFILE_INBOUND

Partner number	ZSM1	MAISELSA
Partner Type	SL	User (first 10 characters, no check)
Partner Role		
Direction	2	Inbound
Client	800	
Port	SAPSM1	

Figure 3.13 Transfer Data in Receiving System

You now know all of the options available for generating IDocs from test transactions. In the next section, we'll turn our attention to status files, which also have some good test options.

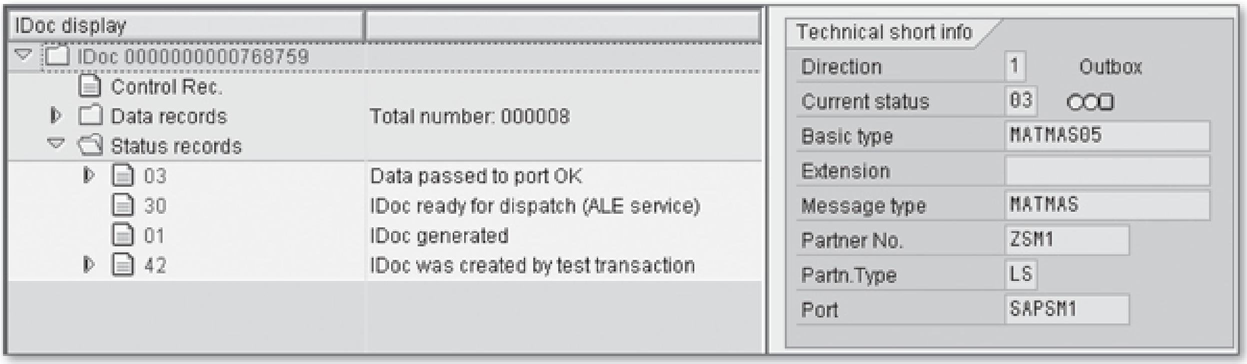
3.3 Processing Status Files

IDoc processing is always an asynchronous process, which means that, if no further action is taken, you don't receive any information about whether or not the partner has successfully processed your IDoc. However, because this information is frequently required, there are various ways in which your partner can make this information available to you.

"STATUS" and "SYSTAT01"

One of these options, which is frequently used by EDI subsystems, involves resending an IDoc with message type `STATUS` and IDoc type `SYSTAT01` along with an IDoc status file. Normally, the sender doesn't know if the receiver was able to successfully process its IDoc. However, the receiver can use a status file or status IDoc to make this information available to the sender. The status IDoc will be described in greater detail in Chapter 5, Section 5.2, STATUS IDocs. Testing concerns the status file only.

Figure 3.14 shows the status values of an IDoc that has already been dispatched. This IDoc will be our starting point for testing status files. The status values *01: IDoc generated*, *30: IDoc ready for dispatch (ALE service)*, and *03: Data passed to port OK* are the status values that an IDoc must receive at least once so that it can be dispatched successfully. In addition, status *42* specifies that the test transaction has successfully created the IDoc. Because IDocs can also be viewed by auditing authorities, if required, it must be clear that an IDoc is either "real" or has been created for test purposes. SAP uses status value *42* to take this situation into account.



IDoc display	
IDoc 0000000000768759	
Control Rec.	
Data records	Total number: 000008
Status records	
03	Data passed to port OK
30	IDoc ready for dispatch (ALE service)
01	IDoc generated
42	IDoc was created by test transaction

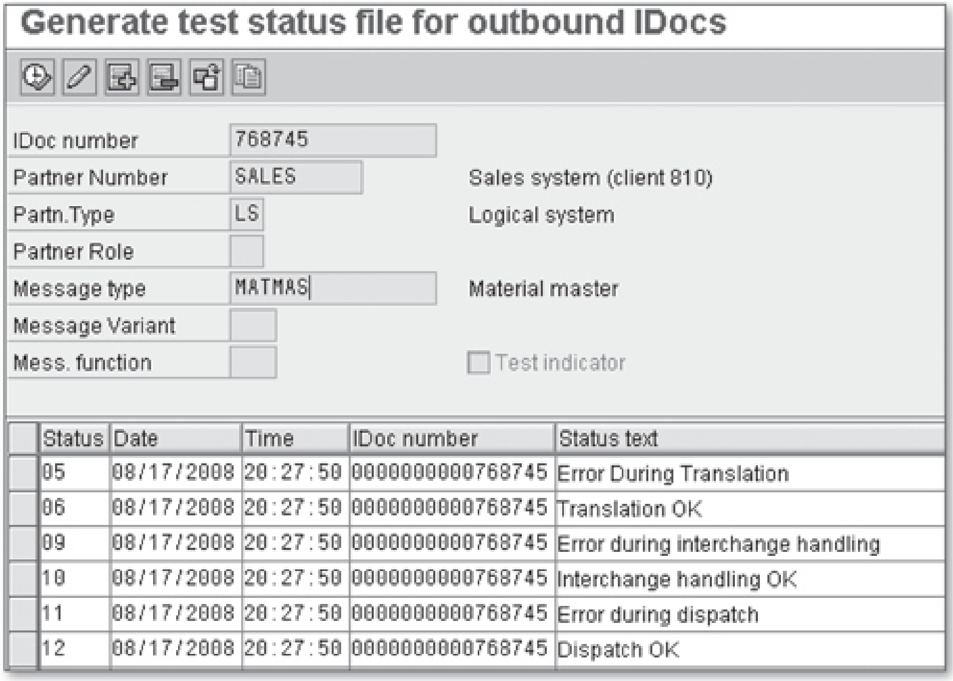
Technical short info

Direction	1	Outbox
Current status	03	
Basic type	MATMAS05	
Extension		
Message type	MATMAS	
Partner No.	ZSM1	
Partn.Type	LS	
Port	SAPSM1	

Figure 3.14 Status Values of an IDoc Created Using Transaction WE19

Creating a status file

When testing a status file, the first step is to create a status file in Transaction WE18. The system proposes certain default values that are used when creating a status file, for example, *05: Error During Translation* (see Figure 3.15). You can edit these status values and add new values, if required.



Generate test status file for outbound IDocs

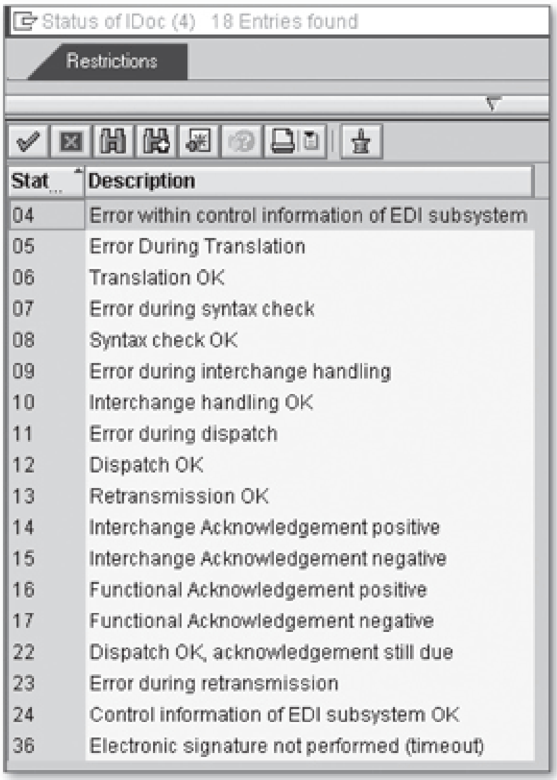
IDoc number	768745	
Partner Number	SALES	Sales system (client 810)
Partn.Type	LS	Logical system
Partner Role		
Message type	MATMAS	Material master
Message Variant		
Mess. function		□ Test indicator

Status	Date	Time	IDoc number	Status text
05	08/17/2008	20:27:50	0000000000768745	Error During Translation
06	08/17/2008	20:27:50	0000000000768745	Translation OK
09	08/17/2008	20:27:50	0000000000768745	Error during interchange handling
10	08/17/2008	20:27:50	0000000000768745	Interchange handling OK
11	08/17/2008	20:27:50	0000000000768745	Error during dispatch
12	08/17/2008	20:27:50	0000000000768745	Dispatch OK

Figure 3.15 Creating a Status File

Possible status values

The status values themselves are made available via the input help, which contains all of the status values that are possible here from SAP's perspective, namely all values that belong to processing layer S = external system/EDI subsystem. A list of possible status values is provided in Transaction WE47, Status Maintenance (see Figure 3.16). Unfortunately, the descriptive short text can't be transferred at present. There is also no separate help, which means that you must manually enter the short text in test Transaction WE18.



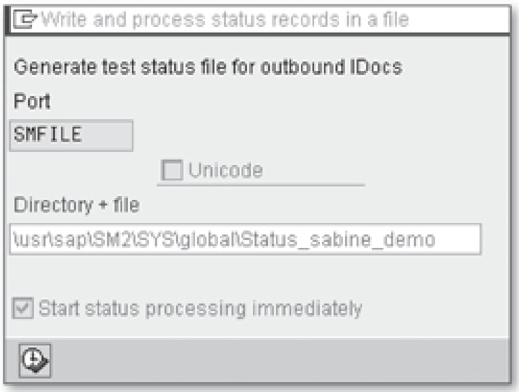
Status of IDoc (4) 18 Entries found

Stat...	Description
04	Error within control information of EDI subsystem
05	Error During Translation
06	Translation OK
07	Error during syntax check
08	Syntax check OK
09	Error during interchange handling
10	Interchange handling OK
11	Error during dispatch
12	Dispatch OK
13	Retransmission OK
14	Interchange Acknowledgement positive
15	Interchange Acknowledgement negative
16	Functional Acknowledgement positive
17	Functional Acknowledgement negative
22	Dispatch OK, acknowledgement still due
23	Error during retransmission
24	Control information of EDI subsystem OK
36	Electronic signature not performed (timeout)

Figure 3.16 Possible Status Values for the Status File

Status file according to your specifications

In the next step (see Figure 3.17), you use the Directory + file field to specify where you want to save the new file, and you use the Start Status Processing Immediately flag to specify whether you want the file to be processed immediately. The system then creates a status file according to your specifications. Your partner can now use this file as a template for creating such files.



Write and process status records in a file

Generate test status file for outbound IDocs

Port

SMFILE

☐ Unicode

Directory + file

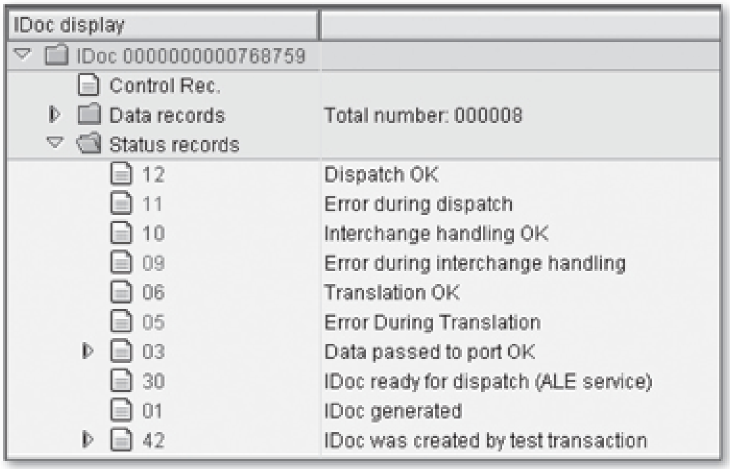
\usr\sap\SM2\SYS\global\Status_sabine_demo

☑ Start status processing immediately

Figure 3.17 Input Values for Saving the Status File

Importing the status file

You can now use Transaction WE17 to import a status file (that you've created yourself or transferred from a partner) into your system. The path and port specifications correspond to those specified in the transaction shown previously (WE19). For test purposes, it doesn't matter if you've created the file yourself or if the file has been transferred from a partner. In addition to your previous status values, the IDoc also gets the new status values from the status file. Figure 3.18 shows the outcome when you use the data entered in Transaction WE18.



IDoc display	
▽ IDoc 0000000000768759	
Control Rec.	
▷ Data records	Total number: 000008
▽ Status records	
12	Dispatch OK
11	Error during dispatch
10	Interchange handling OK
09	Error during interchange handling
06	Translation OK
05	Error During Translation
▷ 03	Data passed to port OK
30	IDoc ready for dispatch (ALE service)
01	IDoc generated
▷ 42	IDoc was created by test transaction

Figure 3.18 Status Values of IDoc After Importing the Status File

Response to status values

As you can see, the status values in Figure 3.18 are success and failure status values. Note that, in the case of error messages, you can trigger error workflows (error Transaction EDIS) again, and the IDoc may get a status that makes it ready-for-dispatch again. The last original status (03: Data passed to port OK) is a status value that doesn't allow the IDoc to be dispatched again. The option of using the new status value to reschedule an IDoc is often the reason why such status values are exchanged with an EDI subsystem or another partner. As far as the SAP system is concerned, there are no problems once an IDoc is dispatched successfully. Consequently, there is no need to dispatch the IDoc again. If the receiving system experiences serious errors and needs the IDoc again, it must inform the sending system. Without the use of status IDocs or status files, it would not be possible to subsequently assign a new status to IDocs that have been successfully processed in the SAP system.

3.4 Summary

In this chapter, you learned about test tools. Now, you should be able to test the IDoc connections that you would like to use with your partners without having to involve the partners. Depending on the direction of the IDoc (inbound or outbound), you can figure out the test tool that fits your needs best. You also learned how status handling can be tested as needed. By knowing that the standard process works as necessary, we can move onto the adoption of special needs to the SAP Standard IDocs, in Chapter 4.

So far, this book has described the use of SAP-provided IDocs. In this chapter, you'll learn how you can customize these IDocs with options ranging from slight Customizing adjustments to complex custom developments integrated into exits.

4 Changes to IDocs

Changes to the standard version

The SAP-provided standard IDoc types (which are also called basic types) consider fields that are included in the SAP standard and relevant for the corresponding scenario. This may not be sufficient for the process you want to provide, particularly if customer-specific enhancements have been implemented in the application transactions, which are corresponding to the IDoc types. Consequently, there are various options for changing IDocs that apply to both the data volume and the data content. You can even completely suppress the generation of IDocs under certain circumstances. The changes can be made via Customizing as well as by using the entire range of enhancement technologies provided by SAP, which are introduced in this chapter. Let's start with the Customizing, which requires the least interference.

4.1 Customizing

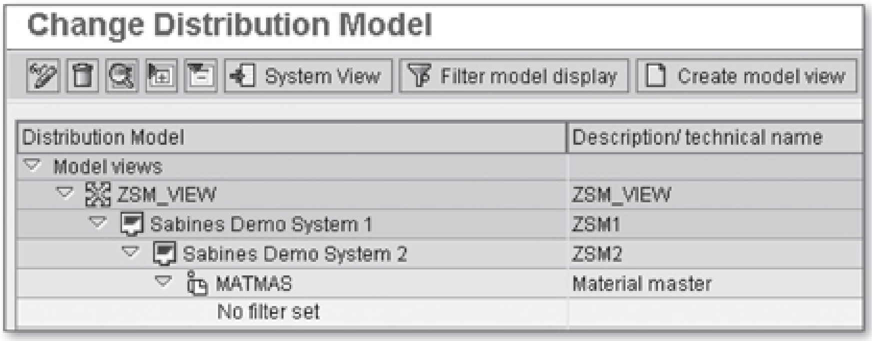
Customizing provides several options for suppressing parts of IDocs or entire IDocs and processing content of IDocs using rules. Customizing through filtering and conversion is used most often and can be implemented relatively easily, which is why it's introduced first.

4.1.1 Filtering Using Filter Objects

Filter objects

You can configure filtering with *filter objects* in Transaction BD64 in the customer distribution model. If SAP provides these, you can select filter objects and assign values to them for any object that is available in a customer distribution model view. Each filter object corresponds to a field.

You can also combine filter objects. In this case, AND as well as OR links of multiple fields are feasible. The distribution model directly indicates if a message type contains filter objects (see Figure 4.1).



Change Distribution Model

Distribution Model	Description/ technical name
▽ Model views	
▽ ZSM_VIEW	ZSM_VIEW
▽ Sabines Demo System 1	ZSM1
▽ Sabines Demo System 2	ZSM2
▽ MATMAS	Material master
No filter set	

Figure 4.1 Distribution Model Without Filtering

Filter groups

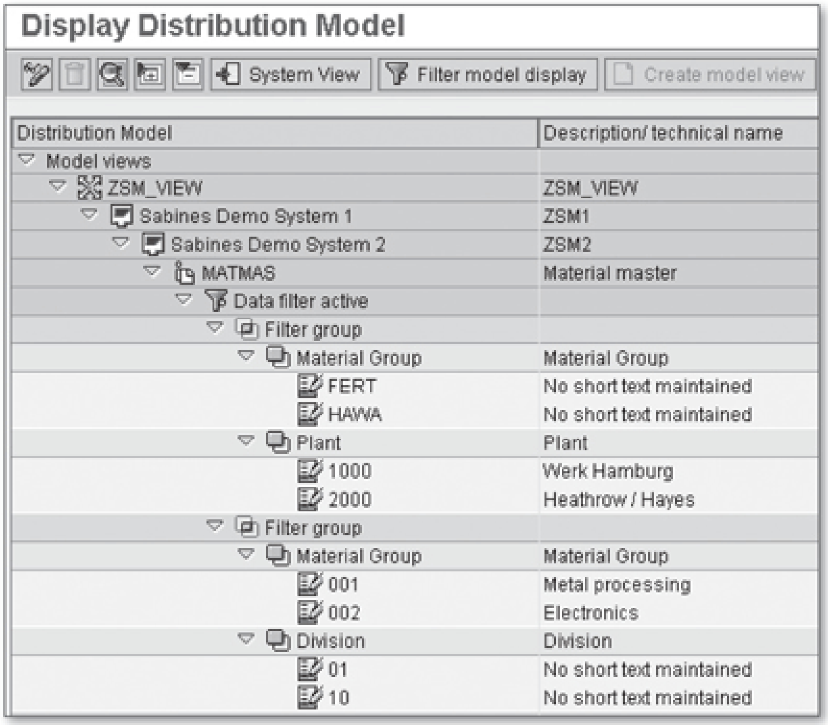
If filter objects are available, you can create *filter groups* for the respective message type. Your settings always refer to a combination of sender and receiver, so you can decide for each sending or receiving system if filtering will be implemented or not. Figure 4.2 shows an example with two filter groups, where each filter group includes two filter objects (Material Group and Plant as well as Material Group and Division) that contain two value instances each. The link is designed so that the conditions of one *or* the other filter group must be met. If conditions for multiple filter objects are specified within the filter group, all conditions must be met. Within a filter object, one condition must be met.

Positive filtering

If a filter object belongs to an optional segment of an IDoc, the system generates this segment if the filter value is met; the system doesn't generate the segment if the filter value isn't met. In the example from Figure 4.2 with plants 1000, 2000, 3000, and 4000 assigned to a material, the system would generate two `E1MARCM` segments for the first filter group, that is, the segments with plants 1000 and 2000, and would suppress the other two segments.

Suppressing IDocs using filters

If a filter object belongs to a field in a mandatory segment, the system suppresses the entire IDoc if it can't generate the segment. In our example, the system would generate IDocs for materials with the `FERT` and `HAWA` material types; for the `HALB` material type, it would suppress the IDoc completely. For a material (independent of the material type and plant) that belongs to the 002 or 001 material group and 01 or 10 division, the system would generate a complete IDoc according to the rules of the second filter group.

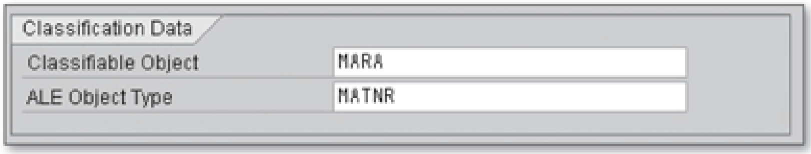


Distribution Model	Description/ technical name
Model views	
ZSM_VIEW	ZSM_VIEW
Sabines Demo System 1	ZSM1
Sabines Demo System 2	ZSM2
MATMAS	Material master
Data filter active	
Filter group	
Material Group	Material Group
FERT	No short text maintained
HAWA	No short text maintained
Plant	Plant
1000	Werk Hamburg
2000	Heathrow / Hayes
Filter group	
Material Group	Material Group
001	Metal processing
002	Electronics
Division	Division
01	No short text maintained
10	No short text maintained

Figure 4.2 Distribution Model with Filtering Using Standard Filter Objects

Filter object "class"

For message types that belong to an object type that allows for classification, you can also filter by class membership. Whether this is possible is described in Transaction BD60, which you already know from Chapter 2, Section 2.1.1, Shared Master Data Tool. Figure 4.3 displays a part of Transaction BD60 showing the settings for the `MATMAS` material master message type. These settings are implemented by the developer of the message type because this is the only person who knows if an appropriate classification object is available.

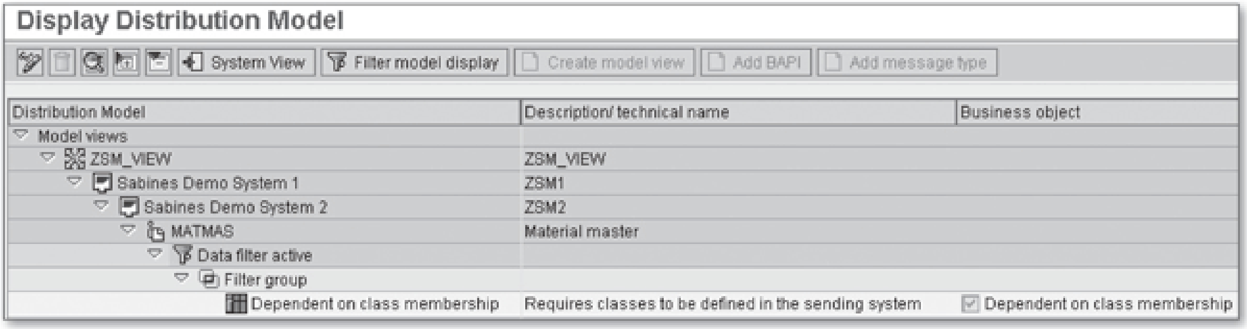


Classification Data

Classifiable Object	MARA
ALE Object Type	MATNR

Figure 4.3 Transaction BD60

If filtering by classification is possible, this is indicated in the filter group as a specific filter object, namely, Dependent on Class Membership (see Figure 4.4). However, the corresponding filter mechanism is only activated here. For actually using the filtering, further steps need to be performed.



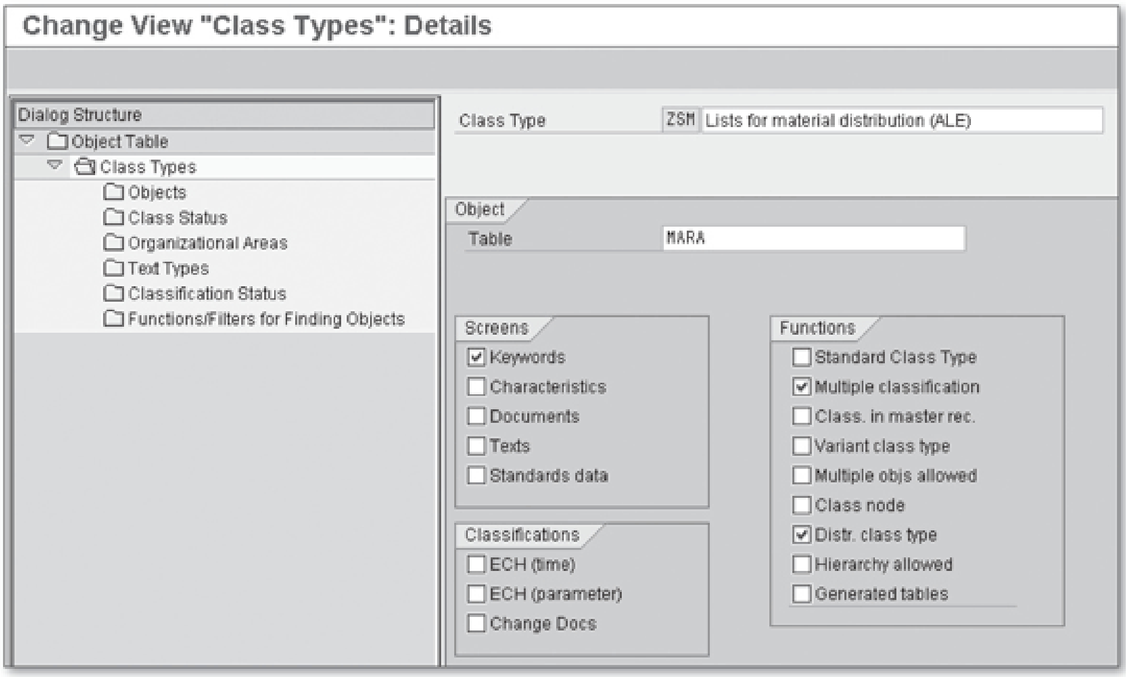
Distribution Model	Description/ technical name	Business object
Model views		
ZSM_VIEW	ZSM_VIEW	
Sabines Demo System 1	ZSM1	
Sabines Demo System 2	ZSM2	
MATMAS	Material master	
Data filter active		
Filter group		
Dependent on class membership	Requires classes to be defined in the sending system	☑ Dependent on class membership

Figure 4.4 Filtering by Class Membership

In general, you can maintain the customer distribution model in both the sending and the receiving system. All settings that are necessary for filtering, however, must be made in the sending system because this system is not supposed to send unwanted IDocs at all.

Distribution class type

First of all, you must create an appropriate class type. You can do this using Transaction O1CL. You have to assign the class type to the corresponding table in Transaction BD60. In our material master example, this would be Table `MARA`. Furthermore, you must identify the class type as a distribution class type using the Distribution Class Type checkbox in the Functions tab. There must be only one distribution class type for each classifiable object, which shouldn't be the standard class type. The system always proposes the standard class type if a class is used for this object. However, this distribution class type is less often used than the class types for finding objects by classification, so, you shouldn't define it as the standard class type. You can maintain the remaining fields according to your common enterprise regulations; these fields aren't relevant for the distribution, and some of them can't be selected anyway. Figure 4.5 shows an example of the input.



Change View "Class Types": Details

Dialog Structure
- Object Table
  - Class Types
    - Objects
    - Class Status
    - Organizational Areas
    - Text Types
    - Classification Status
    - Functions/Filters for Finding Objects

Class Type: ZSM Lists for material distribution (ALE)

Object

Table: MARA

Screens
- ☑ Keywords
- □ Characteristics
- □ Documents
- □ Texts
- □ Standards data

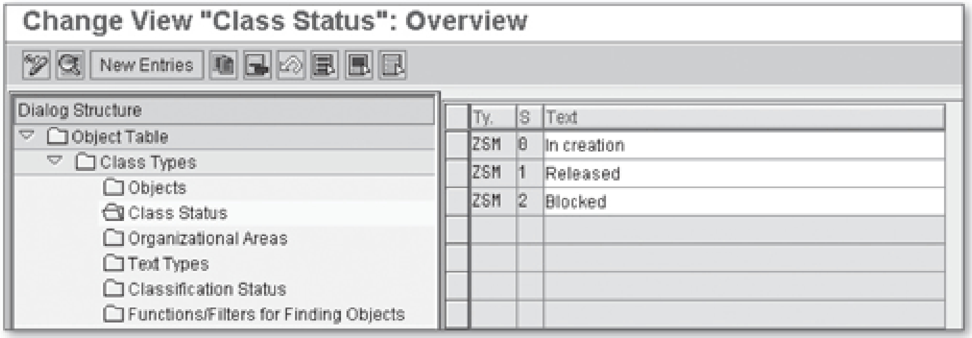
Classifications
- □ ECH (time)
- □ ECH (parameter)
- □ Change Docs

Functions
- □ Standard Class Type
- ☑ Multiple classification
- □ Class. in master rec.
- □ Variant class type
- □ Multiple objs allowed
- □ Class node
- ☑ Distr. class type
- □ Hierarchy allowed
- □ Generated tables

Figure 4.5 Properties of the Distribution Class Type

Status values of classes

For each class type, you can specify different status values, which the classes of this class type can adopt (see Figure 4.6). For the mere distribution, the 1: Released status is required. Whether you need to assign further status depends on your enterprise regulations. The 1: Released status is necessary to use a class as a criterion for distribution. Figure 4.6 displays two additional status values that are frequently used: 0: In creation and 2: Blocked. Because both don't allow for assignments of objects, you don't use them for distribution classes. SAP provides them for the material master distribution class type, so they are displayed here as well.



Change View "Class Status": Overview

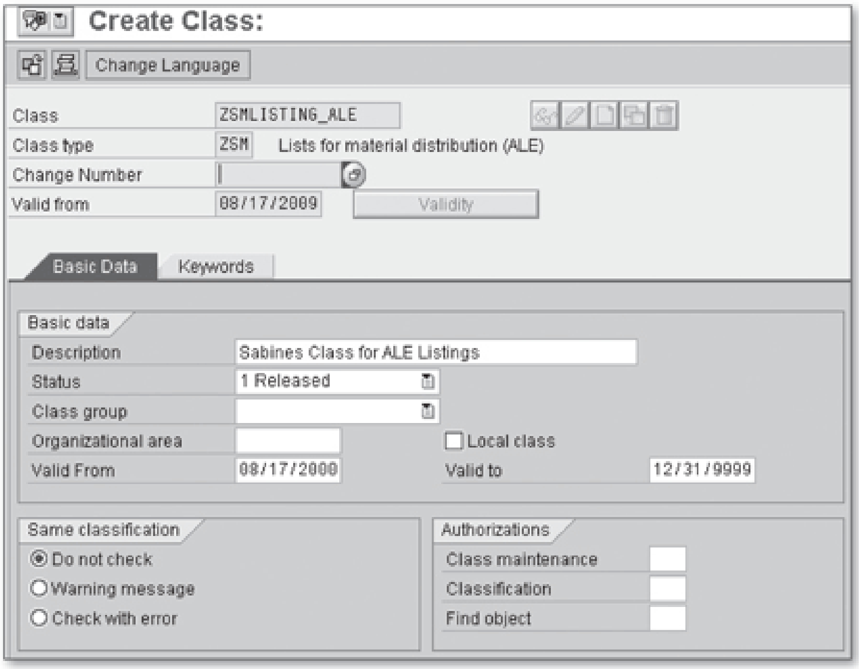
Dialog Structure
- Object Table
  - Class Types
    - Objects
    - Class Status
    - Organizational Areas
    - Text Types
    - Classification Status
    - Functions/Filters for Finding Objects

Ty.	S	Text
ZSM	0	In creation
ZSM	1	Released
ZSM	2	Blocked

Figure 4.6 Status Values of the Distribution Class Type

Same classification

Due to the previous work, you can now create a class in a released status that belongs to the distribution class type using Transaction CL02. You must select the Do Not Check option for Same Classification (see Figure 4.7). The check actually ensures that two objects with the same values assigned to their characteristics can't be assigned to the same class. However, because distribution classes don't involve characteristic value assignments, the second assigned object already entails multiple classification (two objects with exactly the same properties), so a check isn't wanted here.


Create Class:

Class	ZSMLISTING_ALE
Class type	ZSM Lists for material distribution (ALE)
Change Number	
Valid from	08/17/2009

Basic data

Description	Sabines Class for ALE Listings		
Status	1 Released		
Class group			
Organizational area		□ Local class	
Valid From	08/17/2008	Valid to	12/31/9999

Same classification

● Do not check
○ Warning message
○ Check with error

Authorizations

Class maintenance	
Classification	
Find object	

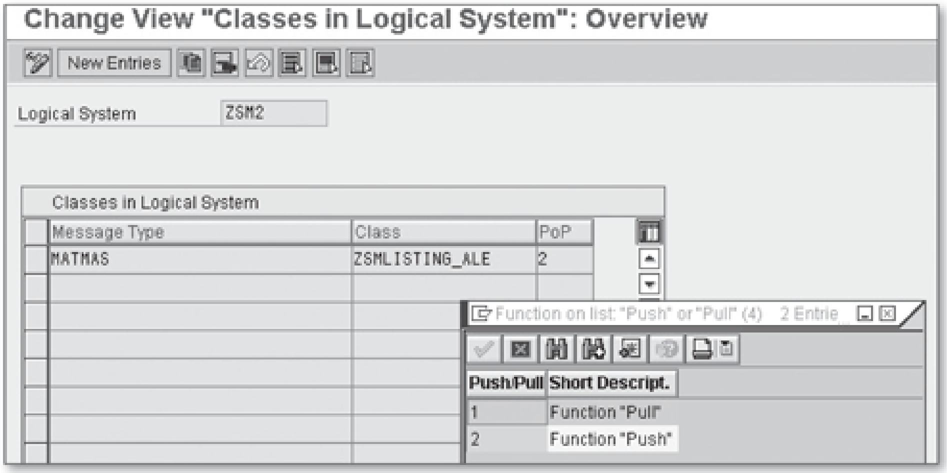
Figure 4.7 Creating a Distribution Class

Assigning to the partner

You can now assign such a distribution class to a partner using Transaction BD68 (see Figure 4.8). Define the receiving system as the Logical System. In the PoP field, always enter "2" for push because you want to send objects. Now, it's no longer possible to send objects to the partners that haven't been assigned to the corresponding class in Transaction CL20N. The IDoc for an object that hasn't been assigned to a class (e.g., material) is suppressed completely and will never be displayed in Transaction BD87, IDoc monitoring.

Assigning multiple classes

You can also assign multiple classes to the same receiving system. No settings have to be made on the receiver side in this case. The pull functionality ("1") isn't used for ALE communication.



Change View "Classes in Logical System": Overview

Logical System: ZSM2

Classes in Logical System

Message Type	Class	PoP
MATMAS	ZSMLISTING_ALE	2

Function on list: "Push" or "Pull" (4)   2 Entrie

Push/Pull	Short Descript.
1	Function "Pull"
2	Function "Push"

Figure 4.8 Assigning the Distribution Class to the Partner System

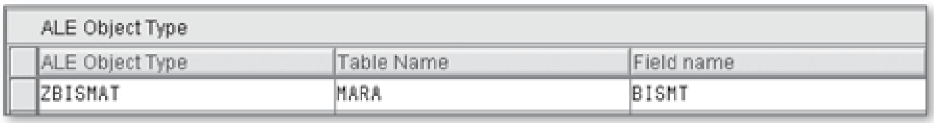
This filtering enables you to send the data to the partner system. Depending on the system's function, this data is relevant there. For example, if you have a special distribution center, you probably only want to send trading goods (HAWA material type) to the corresponding warehouse management system; if you work with a central master data system, and every plant uses its own SAP system, each system should be provided with the material data for its plant only.

4.1.2 Custom Filter Objects

The filtering options described so far are delivered by SAP. If you have additional requirements, you can also create custom filter objects. This can be done not only for standard SAP fields and segments but also for fields in custom tables, for table appends, or for custom segments. Filtering by filter objects is an ALE service.

Creating filter objects

You implement the necessary settings in Transaction BD95. First, assign a name to the filter object. This name must meet the common customer namespace rules, that is, begin with Z or Y or your own customer namespace. In our example, the name is ZBISMAT (see Figure 4.9). Assign the table (MARA) and the field in the database (BISMT) to this name.

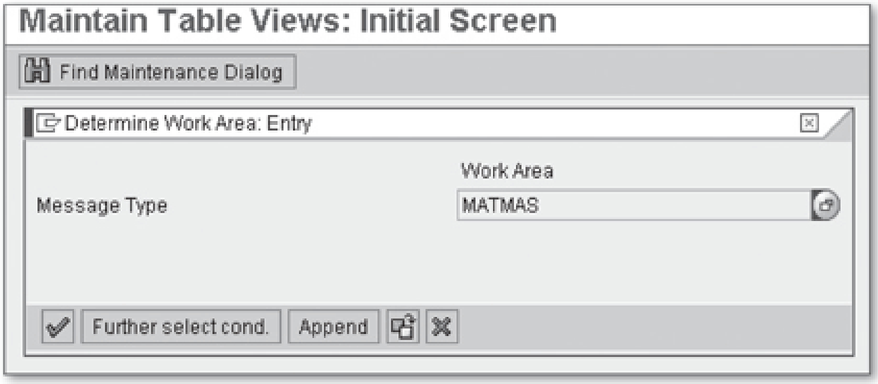


ALE Object Type

ALE Object Type	Table Name	Field name
ZBISMAT	MARA	BISMT

Figure 4.9 Creating a Custom Filter Object

To implement that your filter object is actually used in the ALE services, you must define in which segments of which message types it can be used. For this purpose, first select a message type as the work area in Transaction BD59. Figure 4.10 shows this for the material master example, that is, for the `MATMAS` message type as the work area.



Maintain Table Views: Initial Screen

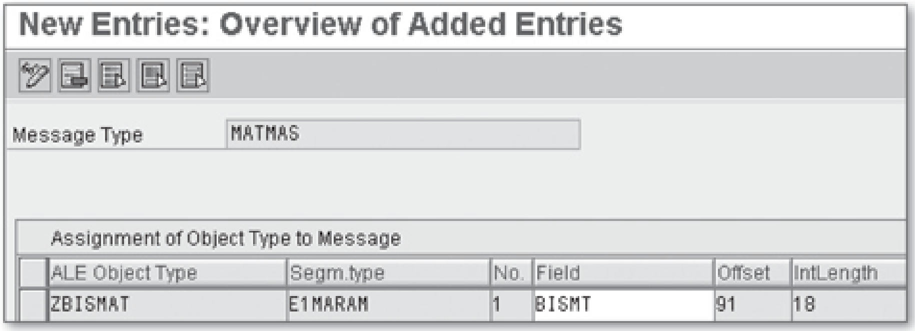
Determine Work Area: Entry

Work Area

Message Type MATMAS

Figure 4.10 Assigning the Custom Filter Object to the Message Type — Step 1

Then, select the segment of the message type in which you want to carry out the filtering (see Figure 4.11). In our example, this is the `E1MARAM` segment. You can now address all fields of this segment to generate your personal filter object.



New Entries: Overview of Added Entries

Message Type MATMAS

Assignment of Object Type to Message

ALE Object Type	Segm.type	No.	Field	Offset	IntLength
ZBISMAT	E1MARAM	1	BISMT	91	18

Figure 4.11 Assigning the Custom Filter Object to the Message Type — Step 2

Assigning the filter object to the IDoc field

When assigning the filter object, the system automatically determines where in the data record within the segment the respective field is located. This leads to the Offset; in our example, the value is "91." If you've created the segment that contains the filter object yourself, and if you modify it so that the position of the corresponding field changes, the system doesn't copy the new offset to your filter object automatically. In this case, you have to update the new information yourself.

Customizing the offset

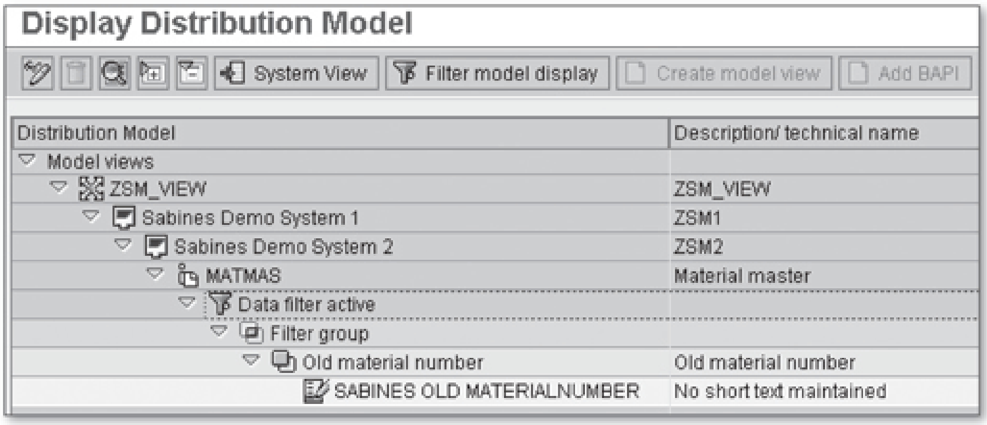
To do this, first determine the new offset value, for example, using Transaction WE60, the IDoc documentation. Then, select your filter object, and choose Edit • Change Field Content in the menu. Three setting options are available:

- Field Name
- Internal Length
- Table Position

First, select Table Position, and select the previously determined offset value. If the length of the filter fields has changed, too, you can adapt the corresponding values in the same way via Internal Length. If the field name has changed, the system finds the new value automatically.

Filtering in the customer distribution model

After you've implemented these settings, the system provides your created filter object in the customer distribution model which can be seen in Figure 4.12. You can use it like a standard SAP filter object. Of course, you can also use the same filter object for multiple message types.



Distribution Model	Description/ technical name
▽ Model views	
▽ ZSM_VIEW	ZSM_VIEW
▽ Sabines Demo System 1	ZSM1
▽ Sabines Demo System 2	ZSM2
▽ MATMAS	Material master
▽ Data filter active	
▽ Filter group	
▽ Old material number	Old material number
SABINES OLD MATERIALNUMBER	No short text maintained

Figure 4.12 Customer Distribution Model with Filtering Using Custom Filter Objects

It makes sense to provide an input help for the field (the SAP system outputs a warning if you refer to a field without a check table for a filter object).

Suppressing an IDoc

Customer segment for suppressing IDocs

If a filter condition for a field isn't met in a mandatory segment, the system doesn't generate the entire IDoc. You can implement this behavior also for fields in optional segments by creating a customer segment as a mandatory segment, using the same field in this customer segment, and generating a specific filter object for it. For example, if you want the system to send a material master IDoc to a particular receiver only if a specific plant is available, you can implement this by specifying the plant and then filtering by plant in a customer segment that is defined as a mandatory segment and directly belongs to the `E1MARAM` segment type.

The SAP standard for the "plant" filter object, in turn, involves suppressing inappropriate plant segments but still sending the `E1MARAM` segment type and `E1MAKTM` segments. However, this doesn't always meet the customer requirements. Section 4.2.3, Custom Segments, describes how you can define custom segments.

Overview of Various IDoc Filters

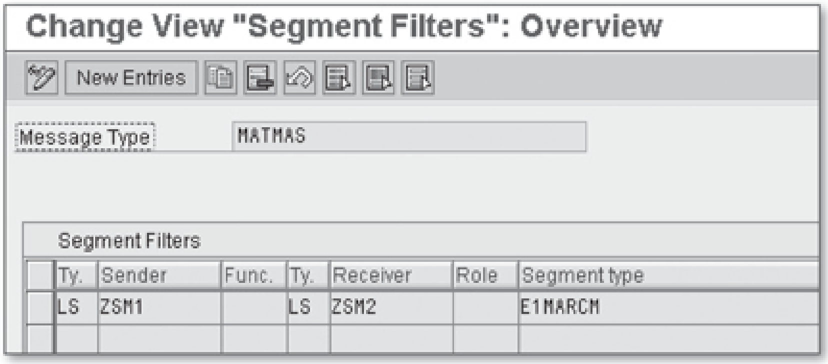
When filtering an IDoc using filter objects, you can — depending on the content of the IDocs — send only specific segments or even entire IDocs. The filtering is positive here: You define which elements can pass. When reducing IDocs, you can also completely omit entire segments and suppress individual fields in the remaining segments. However, you can usually reduce IDocs for master data only. If you don't want to send segments for non-master data for performance reasons, you can suppress them using segment filtering or views. Both approaches are explained in the following section.

4.1.3 Filtering Segments

When segments are used for filtering, the system completely suppresses the generation of a specific segment for a particular combination of sender, receiver, and message type. The content of the segment isn't relevant here. For example, if you don't want to send any plant data of a material to the downstream system, you must configure this via segment filtering.

Filtering segments in the Customizing

The filtering takes place in Customizing. In Transaction BD56, you define the combination that you want to suppress (see Figure 4.13). For this purpose, enter "LS" for logical system in the two Type fields, specify the logical system name of your sending system in the Sender field, and specify the logical system name of the receiving system in the Receiver field. In the case of EDI communication, the specification in the Type field is "KU" for customer or "LI" for vendor. In this case, you must also maintain the partner role whose value can be "AG" for sold-to party, for example. The Segment Type field indicates the name of the segment you want to suppress. In contrast to reducing IDocs and using filter objects in the customer distribution model, which always takes place in the sending system, the filtering can also be implemented in the receiving system. That means the IDoc is transferred completely but not updated completely. For performance reasons, you should only configure filtering by segments in the receiving system if the sending system is an external system that can't suppress segments itself. The filtering of segments is also an ALE service.



Change View "Segment Filters": Overview

Message Type: MATMAS

Segment Filters

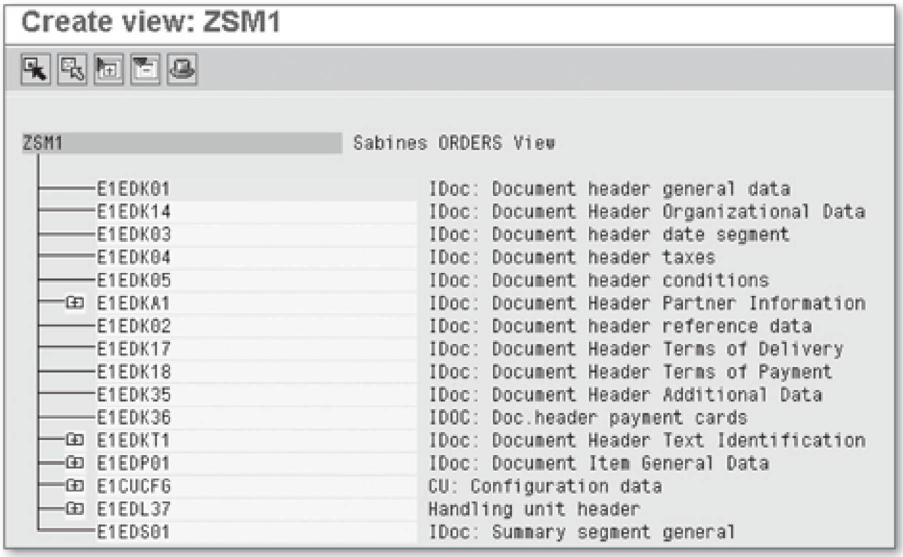
Ty.	Sender	Func.	Ty.	Receiver	Role	Segment type
LS	ZSM1		LS	ZSM2		E1MARCM

Figure 4.13 Filtering a Segment

4.1.4 Reducing IDocs Through Views

IDoc views

Creating and using views is possible for all IDoc types, but doesn't provide as many functions as reduced IDocs. Here, you can select at segment level which objects you want to send; however, you can no longer suppress specific fields within a segment as provided for the reduction. The transaction that enables you to define the views is Transaction WE32. The example deliberately combines a non-reducible message type, that is, ORDERS, with the ORDERS05 IDoc type. For IDocs that allow for reduction, this is usually the more powerful method for customizing IDocs. Figure 4.14 shows the initial screen of this transaction.



Create view: ZSM1

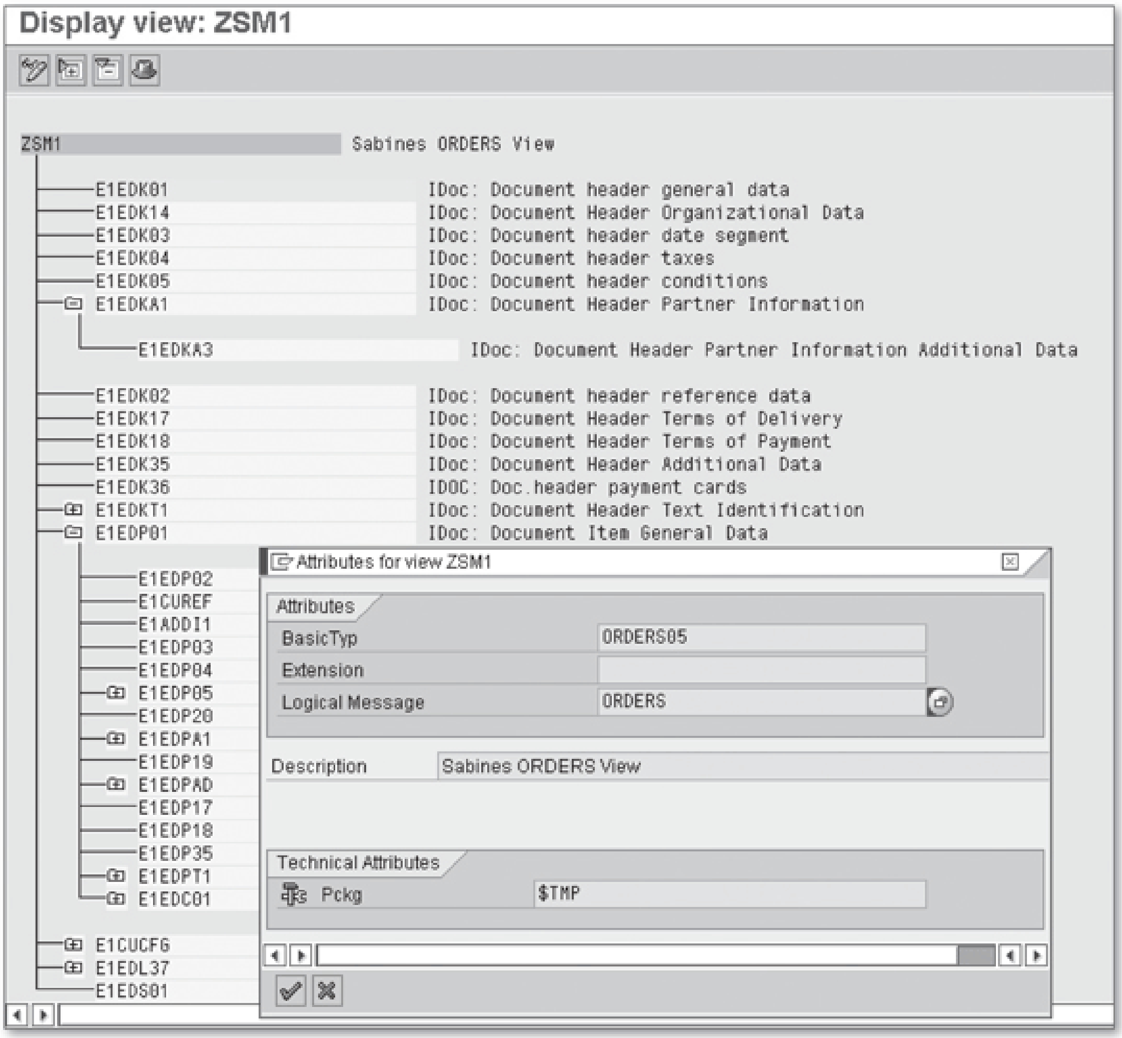
```
ZSM1                      Sabines ORDERS View

├──E1EDK01                IDoc: Document header general data
├──E1EDK14                IDoc: Document Header Organizational Data
├──E1EDK03                IDoc: Document header date segment
├──E1EDK04                IDoc: Document header taxes
├──E1EDK05                IDoc: Document header conditions
├─⊞ E1EDKA1               IDoc: Document Header Partner Information
├──E1EDK02                IDoc: Document header reference data
├──E1EDK17                IDoc: Document Header Terms of Delivery
├──E1EDK18                IDoc: Document Header Terms of Payment
├──E1EDK35                IDoc: Document Header Additional Data
├──E1EDK36                IDOC: Doc.header payment cards
├─⊞ E1EDKT1               IDoc: Document Header Text Identification
├─⊞ E1EDP01               IDoc: Document Item General Data
├─⊞ E1CUCF6               CU: Configuration data
├─⊞ E1EDL37               Handling unit header
└──E1EDS01                IDoc: Summary segment general
```

Figure 4.14 Initial Screen for Views

Mandatory segments in views

This method also involves segments that are necessary from the SAP perspective and need to be part of the view. The top-level or *root segment* (here: E1EDK01) is indispensable. You already know the reason for this from the filtering and reduction processes: If you skip the top-level item, the entire IDoc disappears. You can now define which segments you want to add to the segment that is required as a minimum. Figure 4.15 displays a complete view.

This figure also shows the attributes of the view, its name, for example. When you've defined a segment as relevant for your view using the menu that opens when you right-click on the segment, you can expand it and select the segments listed underneath that you want to include in your view.



Display view: ZSM1

```
ZSM1                    Sabines ORDERS View

 ├──E1EDK01            IDoc: Document header general data
 ├──E1EDK14            IDoc: Document Header Organizational Data
 ├──E1EDK03            IDoc: Document header date segment
 ├──E1EDK04            IDoc: Document header taxes
 ├──E1EDK05            IDoc: Document header conditions
 ├─⊟ E1EDKA1           IDoc: Document Header Partner Information
 │  └──E1EDKA3            IDoc: Document Header Partner Information Additional Data
 ├──E1EDK02            IDoc: Document header reference data
 ├──E1EDK17            IDoc: Document Header Terms of Delivery
 ├──E1EDK18            IDoc: Document Header Terms of Payment
 ├──E1EDK35            IDoc: Document Header Additional Data
 ├──E1EDK36            IDOC: Doc.header payment cards
 ├─⊞ E1EDKT1           IDoc: Document Header Text Identification
 ├─⊟ E1EDP01           IDoc: Document Item General Data
 │  ├──E1EDP02
 │  ├──E1CUREF
 │  ├──E1ADDI1
 │  ├──E1EDP03
 │  ├──E1EDP04
 │  ├─⊞ E1EDP05
 │  ├──E1EDP20
 │  ├─⊞ E1EDPA1
 │  ├──E1EDP19
 │  ├─⊞ E1EDPAD
 │  ├──E1EDP17
 │  ├──E1EDP18
 │  ├──E1EDP35
 │  ├─⊞ E1EDPT1
 │  └─⊞ E1EDC01
 ├─⊞ E1CUCFG
 ├─⊞ E1EDL37
 └──E1EDS01
```

Attributes for view ZSM1

Attributes

BasicTyp	ORDERS05
Extension	
Logical Message	ORDERS

Description: Sabines ORDERS View

Technical Attributes

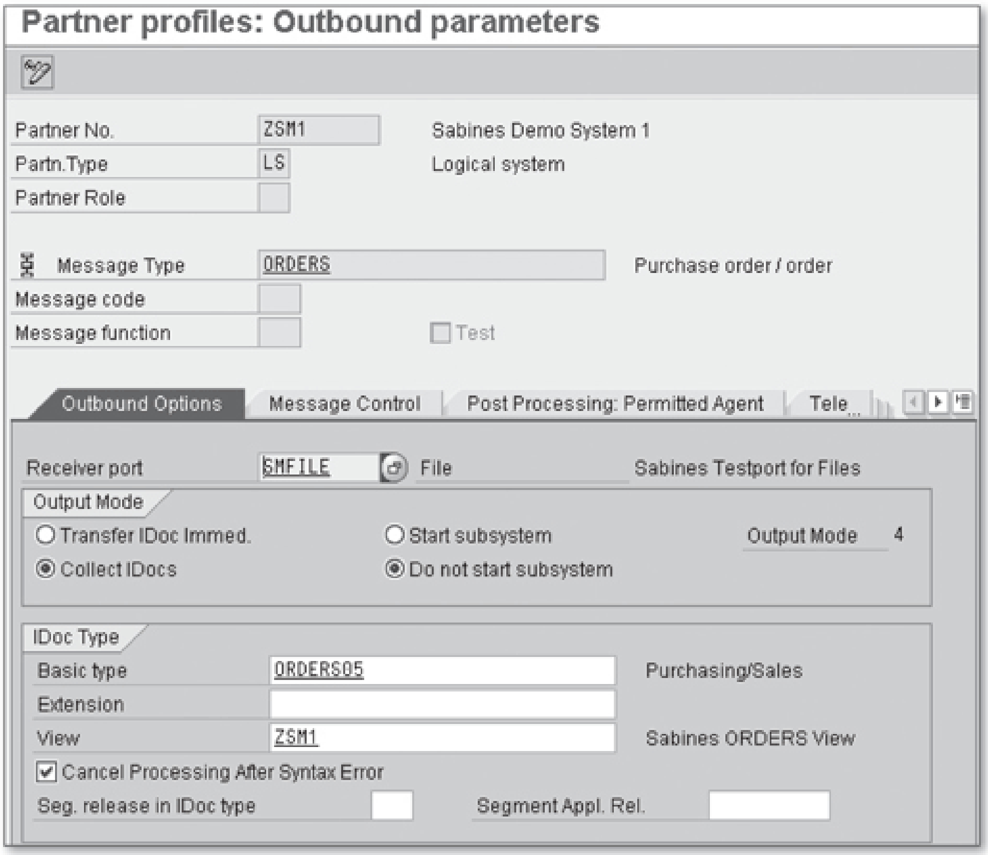
Pckg	$TMP

Figure 4.15 Example of a Custom-Defined View

In contrast to reduced IDocs where you generate a specific message type, you continue to use standard messages here. Consequently, you must also specify in the partner profiles in Transaction WE20 if you want to work with a view and which view you want to use (see Figure 4.16).

Views in the partner profile

As usual, our example uses the `ORDERS` message type for the partner profile. In the details (Outbound Options • IDoc Type) in the IDoc Type area, you then specify `ZSM1` in the View field in addition to `ORDERS05` in the Basic Type field.



Partner profiles: Outbound parameters

Partner No.	ZSM1	Sabines Demo System 1
Partn.Type	LS	Logical system
Partner Role		
Message Type	ORDERS	Purchase order / order
Message code		
Message function		□ Test

Receiver port SMFILE File Sabines Testport for Files

Output Mode

○ Transfer IDoc Immed. ○ Start subsystem Output Mode 4

● Collect IDocs ● Do not start subsystem

IDoc Type

Basic type	ORDERS05	Purchasing/Sales
Extension		
View	ZSM1	Sabines ORDERS View

☑ Cancel Processing After Syntax Error

Seg. release in IDoc type Segment Appl. Rel.

Figure 4.16 Using the View in the Partner Profile

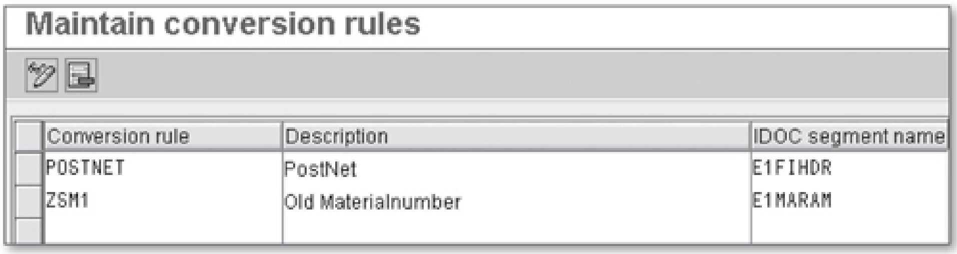
Filtering by views isn't an ALE service, so it can also be used for IDoc modules for which ALE services are deactivated.

4.1.5 Rules

All previous changes to the IDocs aim at reducing the scope of the data that is supposed to be transferred. This is usually done for performance reasons and to reduce the data volume on the database if the partner doesn't need the data.

Changing the content

Often, however, you also need to change the actual content of an IDoc to enable the respective communication partner to process it. Typical examples are different field lengths or abbreviations, such as adding/deleting leading zeros or using different company codes or warehouse numbers. You can handle this without development work using a rule if simple conversion logic is used. Figure 4.17 shows how you create the rule in Transaction BD62 by assigning a name to it first.



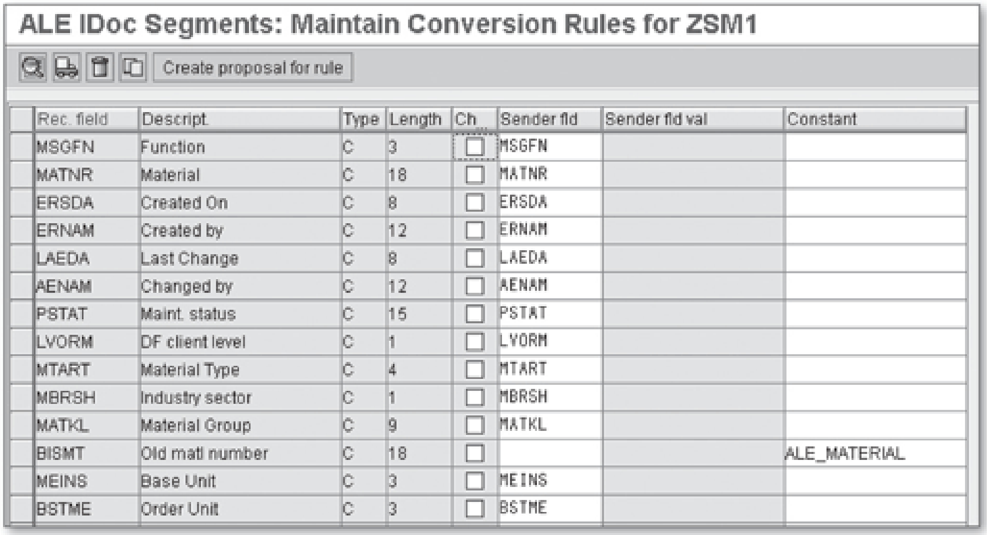
Maintain conversion rules

Conversion rule	Description	IDOC segment name
POSTNET	PostNet	E1FIHDR
ZSM1	Old Materialnumber	E1MARAM

Figure 4.17 Naming the Rule

Creating a rule

When assigning a name to the rule, you also define to which segment the rule applies, so the system lists all fields of this segment if you maintain the actual rule in Transaction BD79. The Create Proposal for Rule function enables you to predefine the `MOVE` rule for all fields, which describes the process of simply copying the field content. `MOVE` is the default setting anyway. Figure 4.18 shows the initial screen of the rule maintenance.

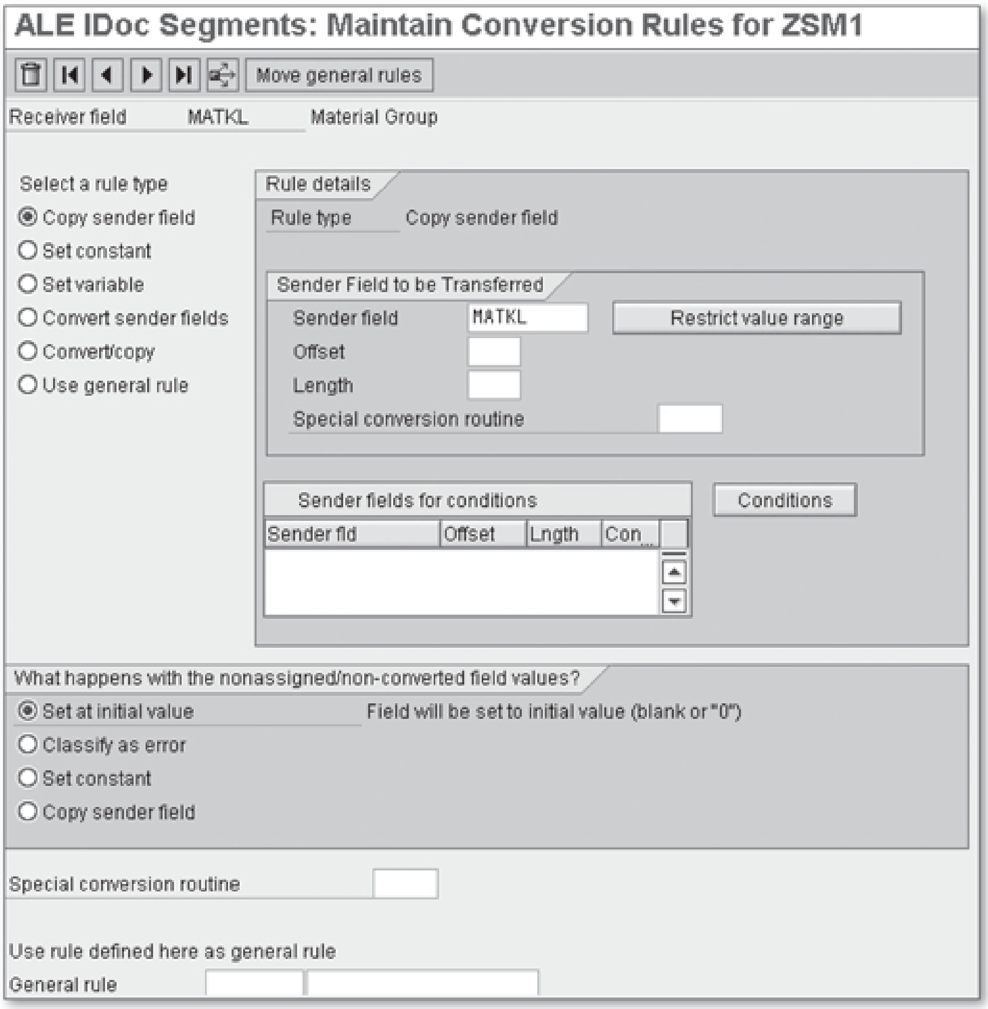


ALE IDoc Segments: Maintain Conversion Rules for ZSM1

Rec. field	Descript.	Type	Length	Ch	Sender fld	Sender fld val	Constant
MSGFN	Function	C	3	□	MSGFN		
MATNR	Material	C	18	□	MATNR		
ERSDA	Created On	C	8	□	ERSDA		
ERNAM	Created by	C	12	□	ERNAM		
LAEDA	Last Change	C	8	□	LAEDA		
AENAM	Changed by	C	12	□	AENAM		
PSTAT	Maint. status	C	15	□	PSTAT		
LVORM	DF client level	C	1	□	LVORM		
MTART	Material Type	C	4	□	MTART		
MBRSH	Industry sector	C	1	□	MBRSH		
MATKL	Material Group	C	9	□	MATKL		
BISMT	Old matl number	C	18	□			ALE_MATERIAL
MEINS	Base Unit	C	3	□	MEINS		
BSTME	Order Unit	C	3	□	BSTME		

Figure 4.18 Rule Maintenance — Overview

For each field, you can define how it's determined from the source field. Figure 4.19 displays the maintenance menu using the MATKL field as an example.



ALE IDoc Segments: Maintain Conversion Rules for ZSM1

Receiver field MATKL Material Group

Select a rule type

- ● Copy sender field
- ○ Set constant
- ○ Set variable
- ○ Convert sender fields
- ○ Convert/copy
- ○ Use general rule

Rule details

Rule type Copy sender field

Sender Field to be Transferred

Sender field MATKL

Offset

Length

Special conversion routine

Sender fields for conditions

Sender fld	Offset	Lngth	Con...

What happens with the nonassigned/non-converted field values?

- ● Set at initial value — Field will be set to initial value (blank or "0")
- ○ Classify as error
- ○ Set constant
- ○ Copy sender field

Special conversion routine

Use rule defined here as general rule

General rule

Figure 4.19 Rule Maintenance — Data for a Field

Rule types

The following rule types can be selected:

- **Copy Sender Field**
  The Copy Sender Field rule type is the `MOVE` copy rule and thus corresponds to the default setting. In the details menu in the lower transaction area, however, you can deploy only a part of the sender field using the offset and length.

- **Set Constant**
  If you set a constant, the system always overwrites an existing value with the same new value.

- **Set Variable**
  Here, you can use a predefined variable as the target value. If the IDoc is copied from a file in the initial screen, you can specify this variable separately for every transfer of IDocs. If you don't work with files, you can use the `KKCD0001` SAP enhancement to set the variables.
- **Convert Sender Fields**
  To convert fields, specify a new value with regard to the initial value of the field. Here, individual field values or intervals can be copied to a target value. You can define, for example, that the value "10" becomes value "100" or that every value from "20" to "40" becomes value "200".
- **Convert/Copy**
  This is a combination of the two rule types, Copy Sender Field and Convert Sender Field, which were already described. You can define conditions for the copy and for the conversion. Previous releases didn't include this rule type. You had to use a conversion rule in which the new value was identical to the old value in some fields.
- **Use General Rule**
  The *general rule* allows for specifying recurring rules only once and referring to this specification for other fields. You can define every rule in the menu as a general rule by assigning a unique name to it. For other fields, the Use General Rule rule type is applied to use the previously defined general rule.

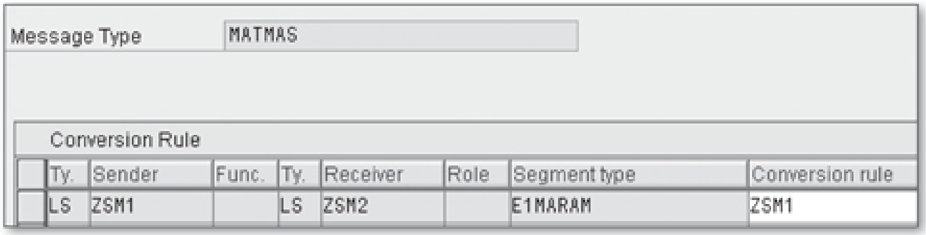
Conversion exits

In addition to the actual rule, there is recurring information, which is displayed in the bottom area of the dynpro in Figure 4.19. This includes the Special Conversion Routine, which refers to the *conversion exit*. Conversion exits format data for the screen. They usually suppress leading zeros. A conversion exit always contains the name of the conversion routine, for example, ALPHA. For fields that can consist of both numbers and text, the `ALPHA` exit maps numbers as right aligned and text as left aligned. Each exit also has two function modules whose names always follow the `CONVERSION_EXIT_<NAME>_<FUNCTION>` pattern, for example, `CONVERSION_EXIT_ALPHA_OUTPUT` for the mapping on a dynpro in our `ALPHA` sample exit.

In addition, for all rules, you can define the behavior if fields can't be assigned. You can either copy a constant or a sender field or alternatively trigger an error.

Assigning the rule

You decide whether a rule is actually used via Transaction BD55, which enables you to assign the rule to a partner. Figure 4.20 shows how you define for each combination of sender and receiver if a rule is used and which rule is used. The use of conversion rules is also an ALE service.



Message Type: MATMAS

Conversion Rule

Ty.	Sender	Func.	Ty.	Receiver	Role	Segment type	Conversion rule
LS	ZSM1		LS	ZSM2		E1MARAM	ZSM1

Figure 4.20 Assigning the Rule to the Message Type and Partner

Restrictions for SAP ERP HCM

You can't use conversion rules in SAP ERP HCM. This depends on the common processing method for infotypes here. Because the actual values can't be addressed directly but only via infotypes for security reasons, the mechanisms used in the conversion rules don't take effect because the proper infotypes aren't known at execution.

4.1.6 Version Conversion

Conversion to older segment versions

IDocs are also supposed to exchange information between two systems with different release versions. To adapt the IDocs accordingly, you can use the *version conversion* option (see also Section 4.2.2, General Exits). For this purpose, you can enter an IDoc type of an older release in the partner outbound processing, which removes all segments that didn't exist in the respective release. You can also specify an older release version within a *segment version* and consequently ensure that the system generates the individual segments based on this release. By default, no specification is made in the segment version; so the system generates the most recent release version of the individual segments for older IDoc types. The version conversion is also an ALE service.

Changes and ALE Services

In the context of the IDoc function module Customizing, you learned that you can define for every function module whether ALE services are available or not. From the previously described change options, the following are ALE services:

- Filtering by filter objects (outbound only)
- Segment filtering
- Field conversion using rules
- Version conversion

You configure the version conversion in the partner profile, and all other options are set via Customizing. Filtering by filter objects is possible in the sending system (outbound) only; all other options are also available on the receiver side (inbound). Your settings are only used, however, if the corresponding function module supports ALE services (which fortunately holds true for most of them).

4.1.7 Special Conversions in SAP ERP Financials

Global organizational units

Financial Accounting includes company codes and business areas, which cover specific controlling functions. It's particularly critical that data is delivered to the appropriate area when being transferred. Often, the various systems use the same organizational units (e.g., the SAP-provided company code, `0001`) with a different meaning. For the resulting conversions that are frequently required, SAP has developed a specific approach: *cross-system company codes* or *cross-system business areas*. You can access these settings via the ALE Customizing only but not via any transaction code. Figure 4.21 shows this using Transaction SALE in SAP ERP.

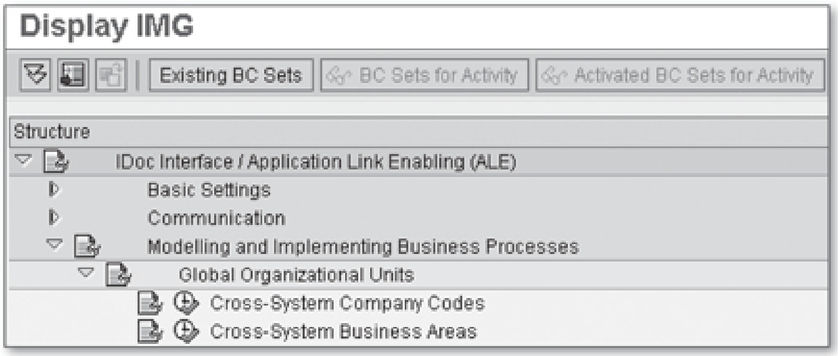
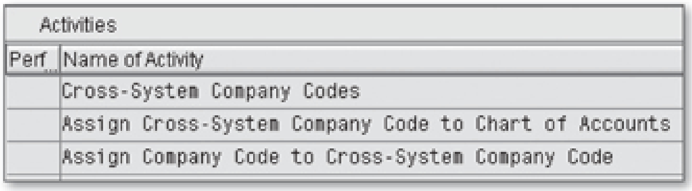

Display IMG
Existing BC Sets
BC Sets for Activity
Activated BC Sets for Activity
Structure
IDoc Interface / Application Link Enabling (ALE)
Basic Settings
Communication
Modelling and Implementing Business Processes
Global Organizational Units
Cross-System Company Codes
Cross-System Business Areas

Figure 4.21 Menu Path for Global Organizational Units

The IDoc Interface/Application Link Enabling (ALE) • Modeling and Implementing Business Processes • Global Organizational Units path enables you to configure cross-system company codes and cross-system business areas. Figure 4.22 displays the steps that you need to perform for creating a cross-system company code: creating cross-system unique company codes, assigning a cross-system company code to the chart of accounts, and assigning the local company code to the cross-system company code.



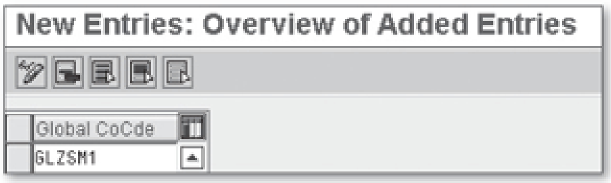
Activities

Perf	Name of Activity
	Cross-System Company Codes
	Assign Cross-System Company Code to Chart of Accounts
	Assign Company Code to Cross-System Company Code

Figure 4.22 Settings for Cross-System Company Codes

Cross-system company code

Each cross-system company code is assigned with a unique name. It can contain six characters and can be longer than the normal company code (four characters) so that adding the prefix "GL" for "global" makes sense here. There are no rules for names of global company codes, however. Figure 4.23 shows an example for the structure of the name of a global company code. It consists of the prefix "GL," the letter "Z," the code "SM," and the digit "1." The financial department of your enterprise is responsible for publishing the required information on charts of accounts, company codes, and global company codes that may have to be used. You should not make this decision yourself because it can affect operations that are relevant to the financial statement.



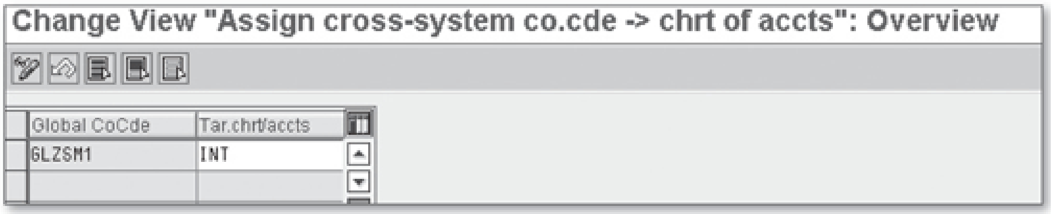
New Entries: Overview of Added Entries

Global CoCde
GLZSM1

Figure 4.23 Creating a Global Company Code

Chart of account assignment

You must assign a chart of account to every global company code. This chart of account is then used when the IDocs are updated. The charts of accounts in the sending and receiving systems must correspond with regard to their accounts. In the example, the target chart of account, INT, has been assigned to the global company code, GLZSM1 (see Figure 4.24).



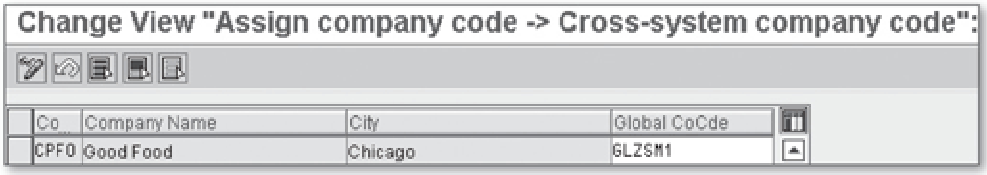
Change View "Assign cross-system co.cde -> chrt of accts": Overview

Global CoCde	Tar.chrt/accts
GLZSM1	INT

Figure 4.24 Assigning the Chart of Accounts to the Cross-System Company Code

Assigning the global company code

Now, you assign this global company code to one of the local company codes in your client. The sending system then generates the global company code everywhere in the IDoc where the assigned local company code would be specified; the receiving system, in turn, replaces each global company code in the IDoc with the assigned local company code. Because this replacement must work on both sides, each global object must be assigned to exactly one local object only. In the example in Figure 4.25, the global company code, GLZSM1, would consequently be assigned to the CPFO company code of Company Good Food.



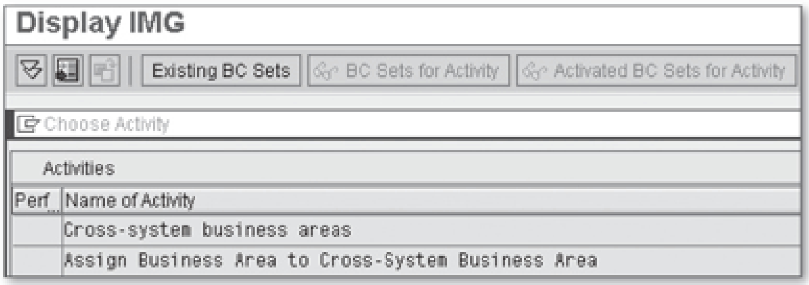
Change View "Assign company code -> Cross-system company code":

Co	Company Name	City	Global CoCde
CPFO	Good Food	Chicago	GLZSM1

Figure 4.25 Assigning the Cross-System Company Code to the Company Code

Cross-system business areas

In a similar way, you can follow the IDoc Interface/Application Link Enabling (ALE) • Modeling and Implementing Business Processes • Global Organizational Units menu path in Transaction SALE to configure cross-system business areas. Figure 4.26 shows the two steps that you need to perform for this purpose. As with the company code, the first step is to specify the name of the cross-system business area. Then you must assign it to the business area in the client.



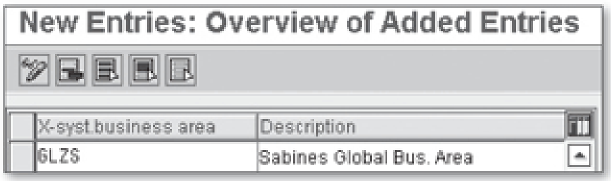
Activities

Perf.	Name of Activity
	Cross-system business areas
	Assign Business Area to Cross-System Business Area

Figure 4.26 Configuring Cross-System Business Areas

Creating the cross-system business area

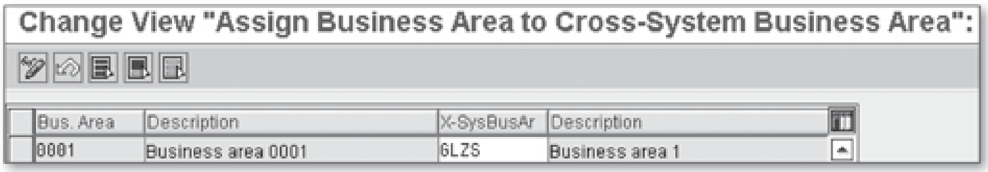
In contrast to the cross-system company code, the name of the cross-system business area consists only of four characters (here: `GLZS`, see Figure 4.27). In this case, too, only the cross-communication uniqueness is critical, and no naming rules apply.



X-syst.business area	Description
GLZS	Sabines Global Bus. Area

Figure 4.27 Creating a Cross-System Business Area

Afterward, you must assign the cross-system business area to a local business area. In our example, the cross-system business area, `GLZS`, is assigned to the local business area, `0001` (see Figure 4.28). Here, too, every global object must be assigned to exactly one local object.


Change View "Assign Business Area to Cross-System Business Area":

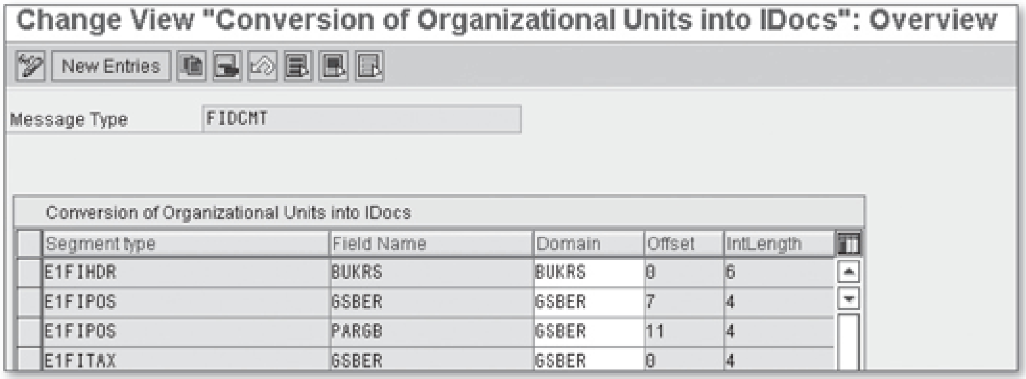
Bus. Area	Description	X-SysBusAr	Description
0001	Business area 0001	GLZS	Business area 1

Figure 4.28 Assigning the Cross-System Business Area to the Local Business Area

The settings described previously form the prerequisite for converting global organizational units. The actual execution, however, depends on the communication partners and on the message types used. For example, you can only use the local company code if you send data to your own warehouse, and you can only use the global company code if you send data to a FI (Financial Accounting) consolidation system.

Figure 4.29 shows how Transaction BD58 converts the company code and the business area in all involved segments for the FIDCMT message type. For this process, it's critical that you specify the field in Field Name that you want to convert because an IDoc can contain several appropriate fields. The FIDCMT IDoc transfers line items for the general ledger in SAP ERP Financials.

Activating for each segment and field


Change View "Conversion of Organizational Units into IDocs": Overview

Message Type FIDCMT

Conversion of Organizational Units into IDocs

Segment type	Field Name	Domain	Offset	IntLength
E1FIHDR	BUKRS	BUKRS	0	6
E1FIPOS	GSBER	GSBER	7	4
E1FIPOS	PARGB	GSBER	11	4
E1FITAX	GSBER	GSBER	0	4

Figure 4.29 Activating the Global Conversion for Each Message Type

4.2 Adapting Existing IDoc Types

Customers who have implemented their own developments within the SAP system often also want to use these modifications for the IDocs that belong to the modified objects. As within the modules, SAP meets this requirement by providing an enhancement concept. In addition to adapting IDocs to the common enhancement technologies, you must also adapt them to your own developments by specifying which additional fields and segments are required.

Adapting through programming

The following section first introduces the different types of exits provided by SAP and then uses the material master as an example because it contains a lot of exits. You'll also learn more about the specific exits that apply to all IDocs. The section concludes by describing IDoc-independent enhancements of IDoc types, which is interesting because implementing changes to the send or update process for IDocs usually also involves a requirement for specific fields.

4.2.1 Different Exit Types on the Basis of the Material Master Example

SAP distinguishes between different types of system adaptation. Customizing and personalization are customer-specific or user-specific adjustments. For example, you define in Customizing how many and which plants you use in your SAP system as well as their properties. The personalization adjustment comprises specifications on the default printer and the like. Such system adjustments don't involve programming work.

Exits

However, you can also change the SAP code, which enables you to change settings directly in the original SAP program. These *modifications* can lead to major problems in the case of release changes, so they are frowned upon and must be made known to SAP via the Online Service System. Many enterprises don't allow for modifications at all for these reasons.

Properties of exits

To avoid the modification problem, SAP provides exits in places that are supposed to be used with changes compared to the standard. For such an exit, SAP is responsible for actually providing an exit in a standard SAP program by defining whether an enhancement in the program exists and where it's called. SAP also determines which data is provided for the customer within the exit and whether the customer is allowed to change and write back certain data.

The second step is implemented on the customer side. Here, the exits that were delivered empty can be filled with your own code. SAP provides different types of exits:

Exit types

- **Customer exits**
  Customer exits have been provided since Release 3.0. They can be used only once, are used cross-client, and are delivered by SAP in an enhancement and implemented and activated by customers in an enhancement project. After a change of release, they have to be reactivated.
- **Business transaction events**
  Business transaction events (BTE) have been delivered since Release 4.0. Originally, they were just used in FI; now, many other applications also contain BTEs in the standard SAP programs. You can use

BTEs client-specifically. A distinction is made between two types of BTEs:

  - **Publish&Subscribe interfaces**
    Publish&Subscribe or P/S interfaces are supposed to inform components other than those in which your process is currently executed that a particular process has been carried out in the SAP system and transfer data to these components. A return transport of data isn't provided for. P/S interfaces can therefore have several active implementations at the same time.
  - **Process interfaces**
    Process interfaces replace standard processes. Data is transferred to and returned from the process module that is executed instead of the standard. Process interfaces can only have one active implementation.

- **BAdIs (Business Add-Ins)**
  BAdIs have been provided since Release 4.6 in the common variant; they have also been delivered in a new version bound to *enhancements* since SAP NetWeaver 7.0. BAdIs are object-oriented enhancements. SAP provides a BAdI interface, and the customer generates the corresponding implementation. The SAP developer who delivers the BAdI can decide if it can be used multiple times and if it will be filter-dependent.
- **Enhancements**
  Enhancements are the new concept that has been recently added to SAP NetWeaver 7.0. You can work with explicit enhancements, which are provided by SAP through *enhancement spots*; however, implicit enhancements are also available, which can be implemented at specific locations of every program that doesn't directly belong to the SAP Basis without any interference of the SAP developer.

Development environment

Transaction SE84 (see Figure 4.30) provides an overview of existing exits and potentially existing information.

When implementing any of these exit technologies, you must consider specific aspects because the general program flow mustn't be changed by the exit.

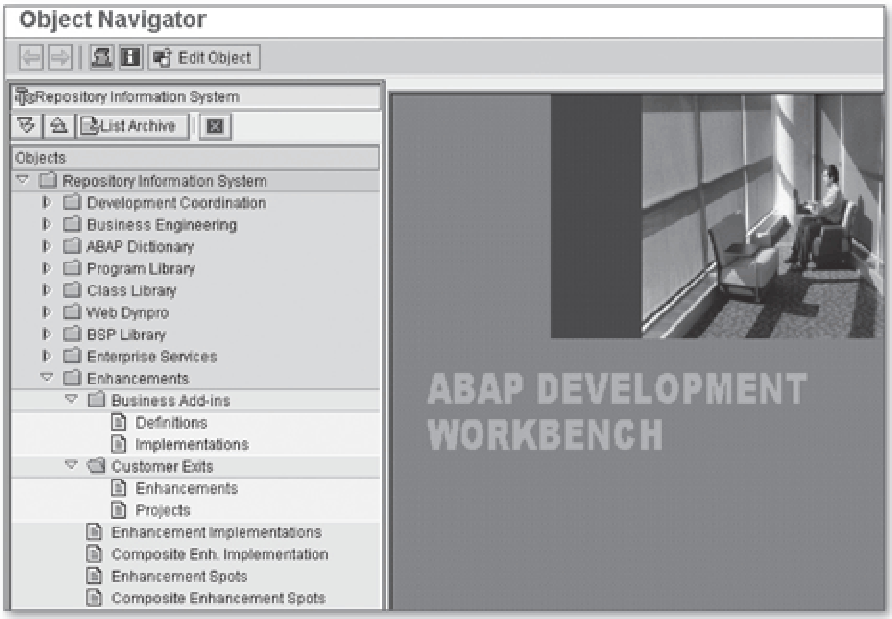


Figure 4.30 Exits in the Repository Information System

Programming Guidelines for Exits

Anything that affects the flow of the original program isn't allowed in exits. In other words, you mustn't create events, subroutines (`FORM`), or modules (`MODULE`) here. Instead, you must create them in separate includes. Data declarations that have been agreed upon with the `DATA` statement apply locally.

Consistency

For consistency reasons, you must also ensure that the customer data is only updated or reset if the same is done with the SAP data. Statements, such as `COMMIT` or `ROLLBACK`, are therefore not permitted. Postings can only be implemented with update technologies and are then processed by the `COMMIT` or `ROLLBACK` operation of the SAP program.

Considering the details, the different enhancement technologies this chapter introduces require different approaches for generating your own code, so these technologies are described in greater detail using the material master as an example. The descriptions only refer to the exit types that you can actually use in IDocs. Where screen and menu exits are possible, no description is provided because IDocs never work with menus or dynpros.

This section also doesn't include code. Code rather depends on whether you want to send or receive IDocs and not on the type of the exit. The code is discussed separately from the described properties of the various exits in Sections 4.2.2, General Exits, and 4.2.3, Custom Segments.

Customer Exits

SAP creates customer exits in Transaction SMOD; that is, you can use this transaction to obtain information on which customer exits are available. Here, several exits can be combined in one enhancement. In the IDoc area, the exits for generating IDocs in outbound processing and for updating the same message types in inbound processing often belong to the same enhancement.

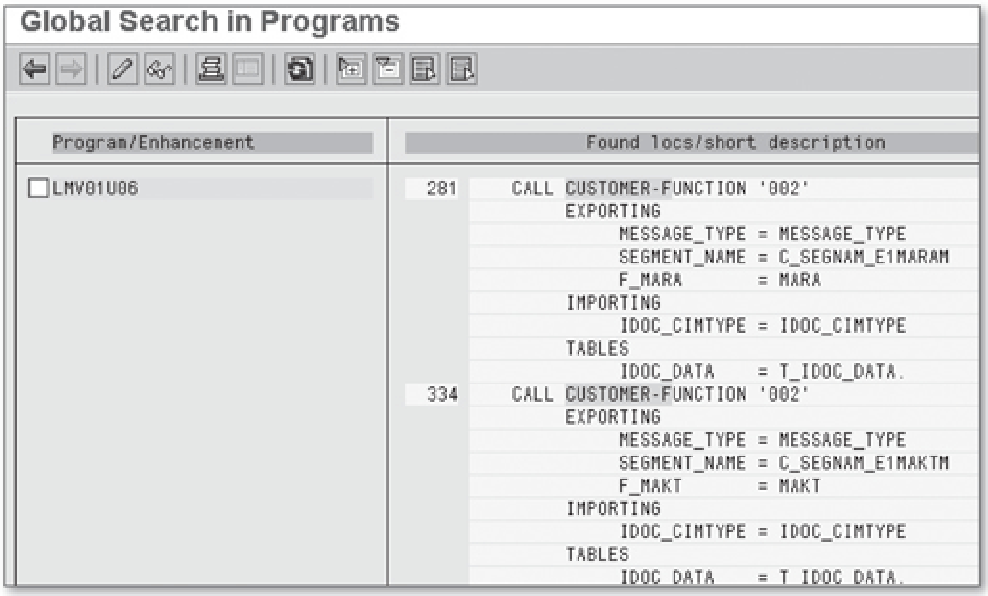
"CALL CUSTOMER-FUNCTION" command

The SAP program calls a customer exit using the `CALL CUSTOMER-FUNCTION 'nnn'` command. The "nnn" addition is any three-digit number that is unique within the enhancement. So if you want to know whether a program provides a customer exit, you can search for `CUSTOMER-FUNCTION`. This indicates which exits are called, which signature the used exit function modules have, and where in the program they are called.

Material master example

If you browse the already-known function module, `MASTERIDOC_CREATE_MATMAS` (Figure 4.31), you see that an exit with number "002" is available and that it has been called several times, namely, each time a new segment has been generated. You can also see which data has been transferred. Information is provided on the message type, the name of the currently generated segment, and all previously generated IDoc data. You can change the user data and return the reference for your enhancement of the IDoc, `CIMTYP`. Section 4.2.3, Custom Segments, describes this in more detail.

When you know the calling program and the exit ID, you also know the name of the exit module, which follows the `EXIT_<PROGRAMNAME>_nnn` pattern. In our example, the exit module is called `EXIT_SAPLMV01_002`. Double-clicking on the three-digit number (`002`) of the name in the search result directly navigates you to the function module (see Figure 4.37 later in this chapter).



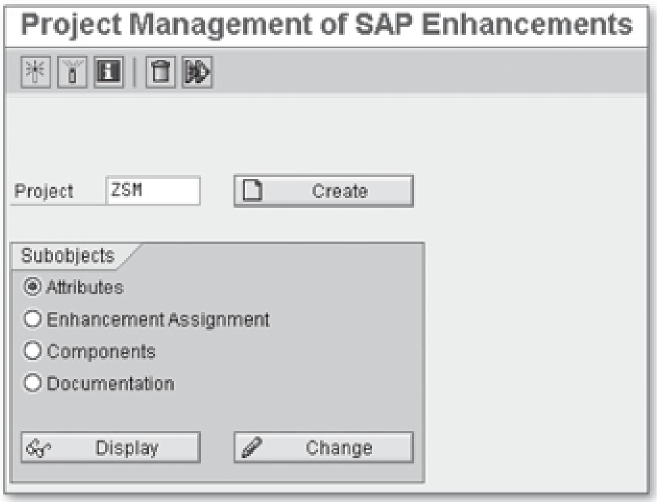
□ LMV01U06

```
281    CALL CUSTOMER-FUNCTION '002'
         EXPORTING
              MESSAGE_TYPE = MESSAGE_TYPE
              SEGMENT_NAME = C_SEGNAM_E1MARAM
              F_MARA       = MARA
         IMPORTING
              IDOC_CIMTYPE = IDOC_CIMTYPE
         TABLES
              IDOC_DATA    = T_IDOC_DATA.
334    CALL CUSTOMER-FUNCTION '002'
         EXPORTING
              MESSAGE_TYPE = MESSAGE_TYPE
              SEGMENT_NAME = C_SEGNAM_E1MAKTM
              F_MAKT       = MAKT
         IMPORTING
              IDOC_CIMTYPE = IDOC_CIMTYPE
         TABLES
              IDOC_DATA    = T_IDOC_DATA.
```

Figure 4.31 Function Module Exits When Generating MATMAS IDocs

Customer enhancement project

When you know which exit you want to implement, you can create your own project using Transaction CMOD (see Figure 4.32). Use the match icon at the left of the toolbar to activate the project at the end of your development work. Only then will your exit be actually executed. First, assign a name to your project. Because SAP doesn't deliver projects, you can theoretically use the entire namespace.



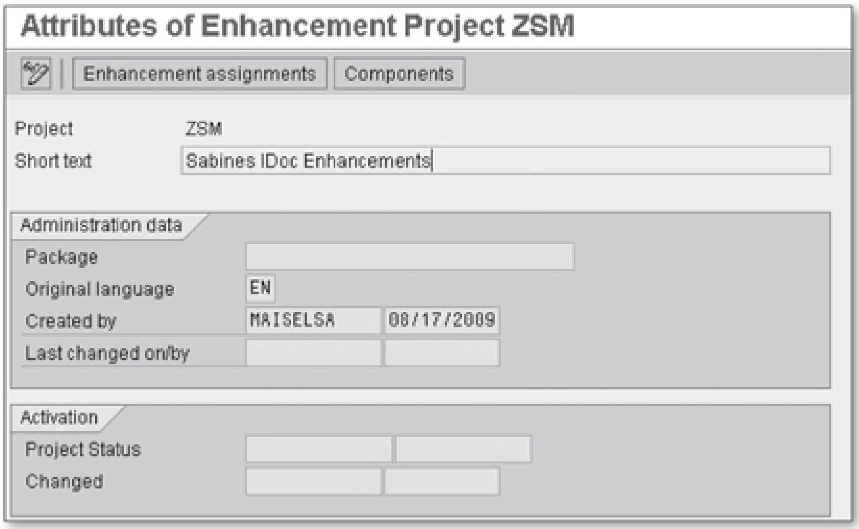
Project ZSM

Subobjects

● Attributes
○ Enhancement Assignment
○ Components
○ Documentation

Figure 4.32 Creating a Customer Project

After clicking the Create button, you can define the attributes of your project. The following sample customer project (see Figure 4.33) is a very common development object. As such, it includes all general data of development objects, for example, specifications on the person who created the object and when it was created, as well as a package. You manage the object via a transport request, which you can use to transport it to the live system.



Attributes of Enhancement Project ZSM

Project ZSM
Short text Sabines IDoc Enhancements
Package
Original language EN
Created by MAISELSA 08/17/2009
Last changed on/by
Project Status
Changed

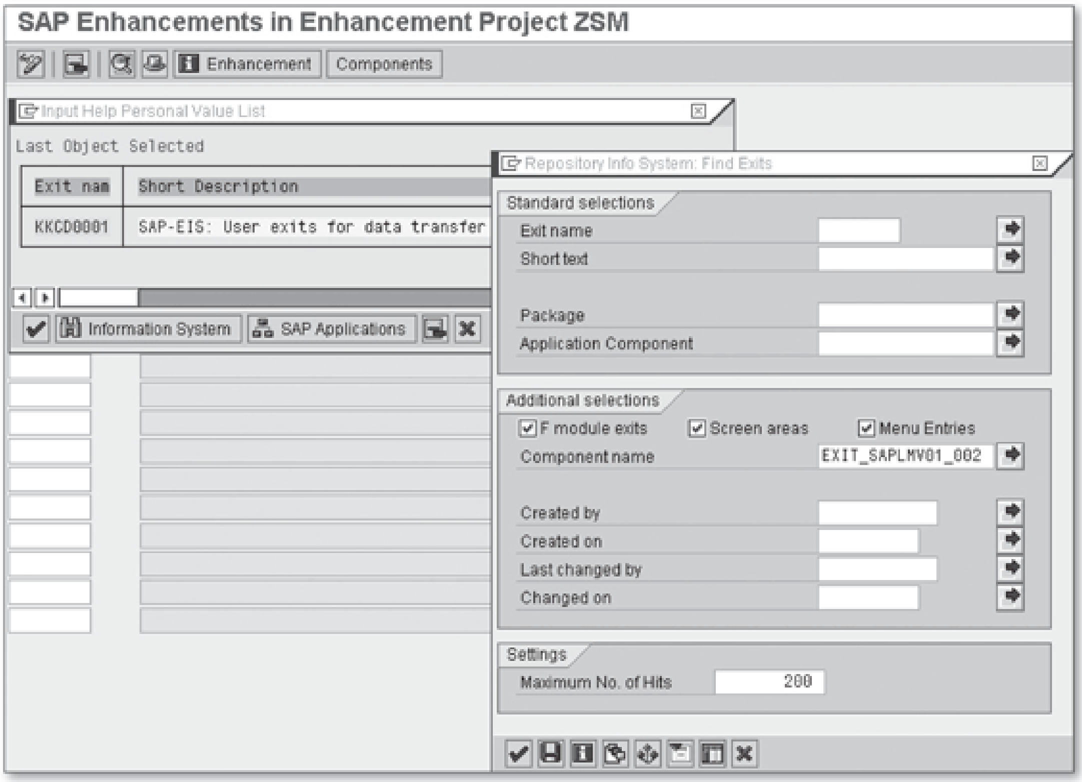
Figure 4.33 Attributes of the Customer Project

Assigning the enhancement

Now you can assign one or several enhancements to your customer project using the Enhancement Assignments button. Every enhancement can only belong to one project. Finding the enhancement that you want to implement to change the material master IDoc is easier here because you can enter the name of the previously identified exit module to determine the enhancement it belongs to. As shown in Figure 4.34, this isn't provided by SAP by default so you need to select the corresponding options in the Additional Selections tab first.

Enhancement for material master IDoc

The enhancement found in the example is called `MGV00001`, and you can now assign it to your customer enhancement project by specifying it in Enhancement (Figure 4.35).



SAP Enhancements in Enhancement Project ZSM

Enhancement
Components
Input Help Personal Value List

Last Object Selected

Exit nam	Short Description
KKCD0001	SAP-EIS: User exits for data transfer

Information System
SAP Applications
Repository Info System: Find Exits

Standard selections

Exit name

Short text

Package

Application Component

Additional selections

☑ F module exits ☑ Screen areas ☑ Menu Entries

Component name EXIT_SAPLMV01_002

Created by

Created on

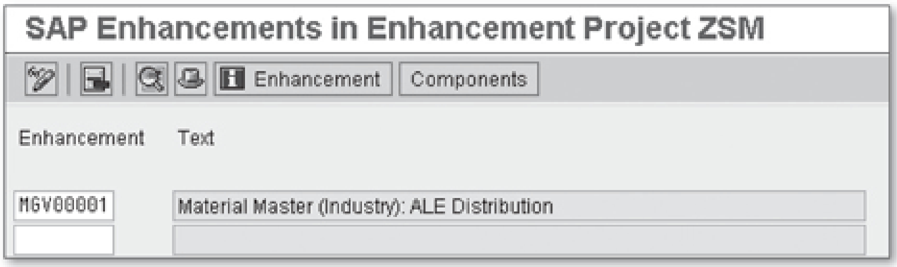
Last changed by

Changed on

Settings

Maximum No. of Hits 200

Figure 4.34 Finding the Enhancement



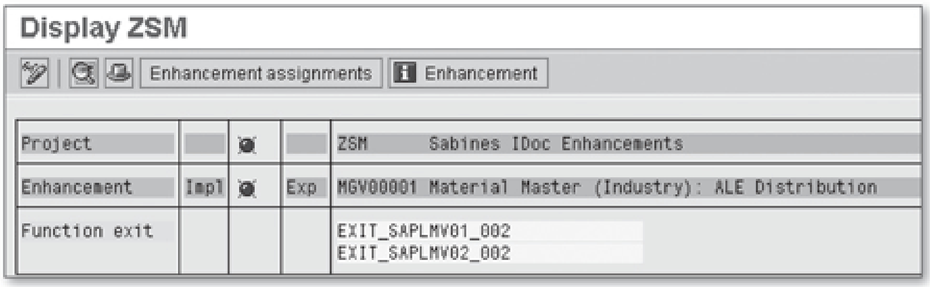
SAP Enhancements in Enhancement Project ZSM

Enhancement	Text
MGV00001	Material Master (Industry): ALE Distribution

Figure 4.35 Assigning the Enhancement

MGV00001 is an enhancement that already contains two function exits. You can also implement only one of them if you don't need the other. The crossed-out circles in Figure 4.36 indicate that the enhancement hasn't been activated yet. The empty column in front of the two exits shows that they also haven't been implemented yet.

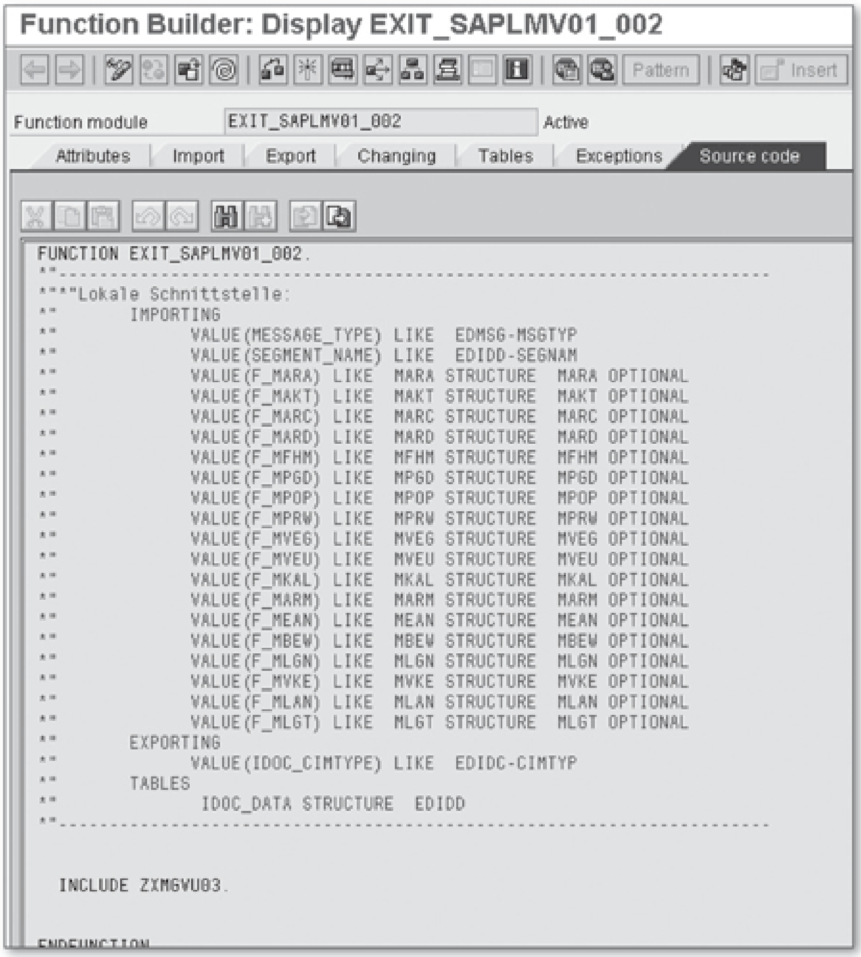
The EXIT_SAPLMV01_002 exit now comprises the complete signature, as you can see in Figure 4.37. Nearly all import parameters are optional, which is why only very few were displayed during the call.



Display ZSM

Project				ZSM Sabines IDoc Enhancements
Enhancement	Impl		Exp	MGV00001 Material Master (Industry): ALE Distribution
Function exit				EXIT_SAPLMV01_002 EXIT_SAPLMV02_002

Figure 4.36 Components of the Enhancement



Function Builder: Display EXIT_SAPLMV01_002

Function module EXIT_SAPLMV01_002 Active

```
FUNCTION EXIT_SAPLMV01_002.
*"----------------------------------------------------------------------
*"*"Lokale Schnittstelle:
*"       IMPORTING
*"             VALUE(MESSAGE_TYPE) LIKE  EDMSG-MSGTYP
*"             VALUE(SEGMENT_NAME) LIKE  EDIDD-SEGNAM
*"             VALUE(F_MARA) LIKE  MARA STRUCTURE  MARA OPTIONAL
*"             VALUE(F_MAKT) LIKE  MAKT STRUCTURE  MAKT OPTIONAL
*"             VALUE(F_MARC) LIKE  MARC STRUCTURE  MARC OPTIONAL
*"             VALUE(F_MARD) LIKE  MARD STRUCTURE  MARD OPTIONAL
*"             VALUE(F_MFHM) LIKE  MFHM STRUCTURE  MFHM OPTIONAL
*"             VALUE(F_MPGD) LIKE  MPGD STRUCTURE  MPGD OPTIONAL
*"             VALUE(F_MPOP) LIKE  MPOP STRUCTURE  MPOP OPTIONAL
*"             VALUE(F_MPRW) LIKE  MPRW STRUCTURE  MPRW OPTIONAL
*"             VALUE(F_MVEG) LIKE  MVEG STRUCTURE  MVEG OPTIONAL
*"             VALUE(F_MVEU) LIKE  MVEU STRUCTURE  MVEU OPTIONAL
*"             VALUE(F_MKAL) LIKE  MKAL STRUCTURE  MKAL OPTIONAL
*"             VALUE(F_MARM) LIKE  MARM STRUCTURE  MARM OPTIONAL
*"             VALUE(F_MEAN) LIKE  MEAN STRUCTURE  MEAN OPTIONAL
*"             VALUE(F_MBEW) LIKE  MBEW STRUCTURE  MBEW OPTIONAL
*"             VALUE(F_MLGN) LIKE  MLGN STRUCTURE  MLGN OPTIONAL
*"             VALUE(F_MVKE) LIKE  MVKE STRUCTURE  MVKE OPTIONAL
*"             VALUE(F_MLAN) LIKE  MLAN STRUCTURE  MLAN OPTIONAL
*"             VALUE(F_MLGT) LIKE  MLGT STRUCTURE  MLGT OPTIONAL
*"       EXPORTING
*"             VALUE(IDOC_CIMTYPE) LIKE  EDIDC-CIMTYP
*"       TABLES
*"              IDOC_DATA STRUCTURE  EDIDD
*"----------------------------------------------------------------------


  INCLUDE ZXMGVU03.


ENDFUNCTION
```

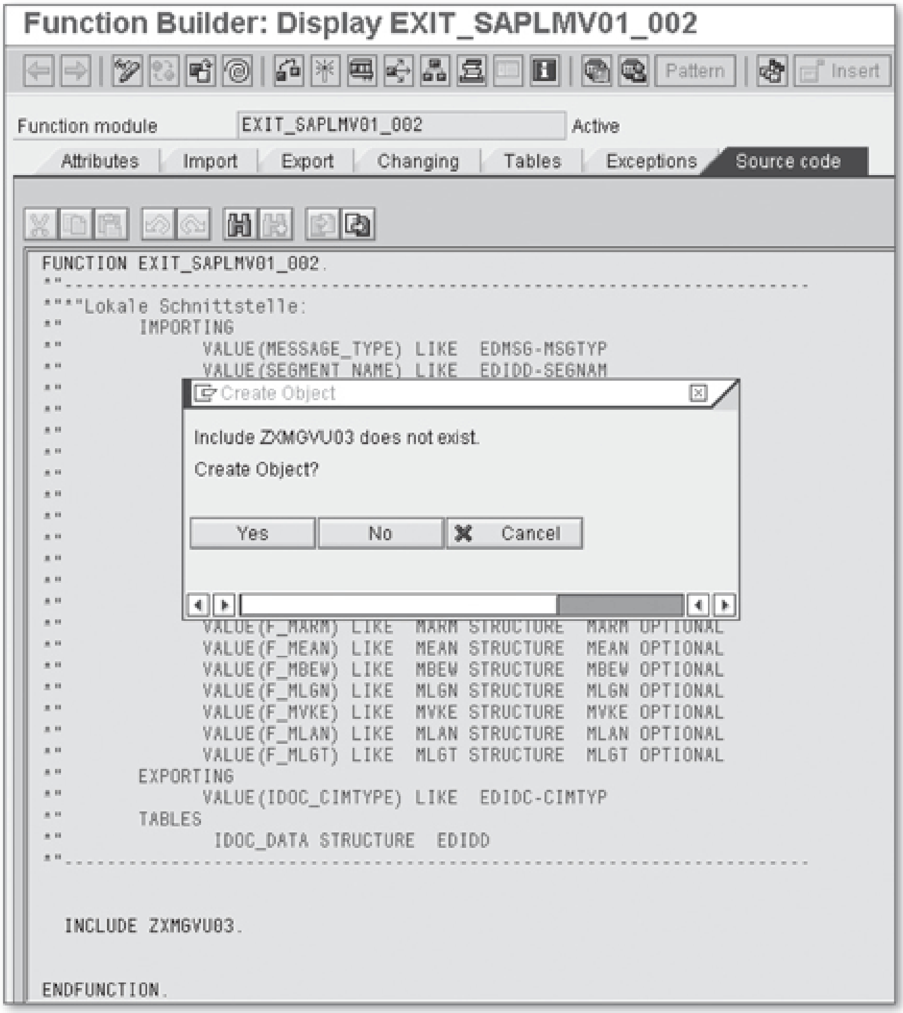
Figure 4.37 Content of the SAP Exits

Customer includes in the exit

The exit belongs to the SAP system and is located in the SAP namespace. In addition, the content of the exit module consists of only one include (ZXMGVU03). This include, in turn, is located in the customer namespace and doesn't exist in the SAP system until the exit has been implemented.

Creating the include

If you double-click on the include name, you can create the include via forward navigation, as shown in the Create Object dialog in Figure 4.38.



Function module EXIT_SAPLMV01_002 Active

```
FUNCTION EXIT_SAPLMV01_002.
*"----------------------------------------------------------------------
*"*"Lokale Schnittstelle:
*"       IMPORTING
*"             VALUE(MESSAGE_TYPE) LIKE  EDMSG-MSGTYP
*"             VALUE(SEGMENT_NAME) LIKE  EDIDD-SEGNAM
*"
*"
*"
*"
*"
*"
*"
*"
*"
*"
*"
*"             VALUE(F_MARM) LIKE  MARM STRUCTURE  MARM OPTIONAL
*"             VALUE(F_MEAN) LIKE  MEAN STRUCTURE  MEAN OPTIONAL
*"             VALUE(F_MBEW) LIKE  MBEW STRUCTURE  MBEW OPTIONAL
*"             VALUE(F_MLGN) LIKE  MLGN STRUCTURE  MLGN OPTIONAL
*"             VALUE(F_MVKE) LIKE  MVKE STRUCTURE  MVKE OPTIONAL
*"             VALUE(F_MLAN) LIKE  MLAN STRUCTURE  MLAN OPTIONAL
*"             VALUE(F_MLGT) LIKE  MLGT STRUCTURE  MLGT OPTIONAL
*"       EXPORTING
*"             VALUE(IDOC_CIMTYPE) LIKE  EDIDC-CIMTYP
*"       TABLES
*"              IDOC_DATA STRUCTURE  EDIDD
*"----------------------------------------------------------------------


  INCLUDE ZXMGVU03.


ENDFUNCTION.
```

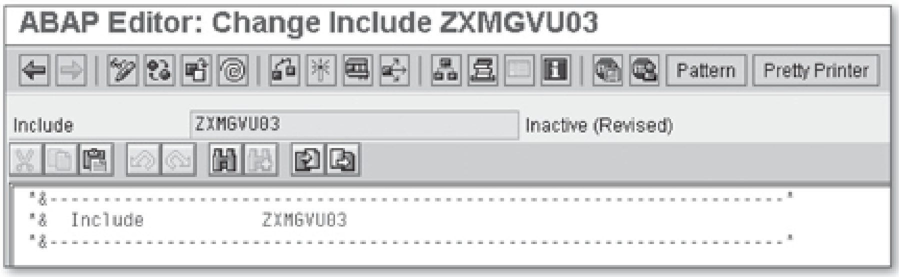
Create Object

Include ZXMGVU03 does not exist.

Create Object?

Figure 4.38 Creating an Include in the Customer Namespace

The system then provides the empty include (see Figure 4.39) in which you can insert your own code. Make sure you save your entries already when editing the include. If you activate the include without having saved your entries and an error occurs, you may have to enter the code all over again.


ABAP Editor: Change Include ZXMGVU03

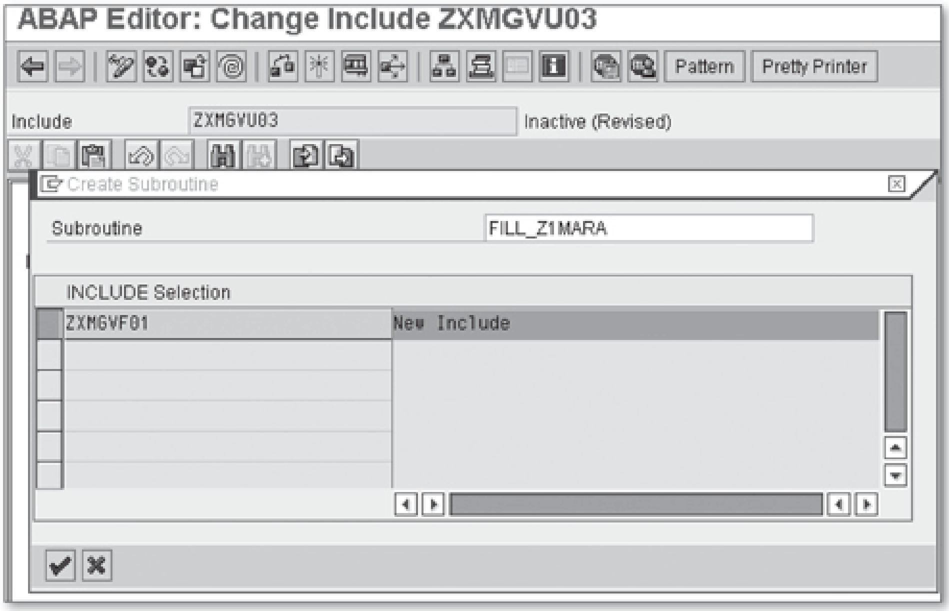
Include ZXMGVU03 Inactive (Revised)

```
*&---------------------------------------------------------------------*
*&  Include           ZXMGVU03
*&---------------------------------------------------------------------*
```

Figure 4.39 Include to Be Filled

Grouping code

You may want to group your code for greater clarity. This can be done by creating additional includes. The includes then begin with "Z"; that is, they are located in the customer namespace but still belong to the SAP exit function group that always begins with "X." SAP proposes `ZXMGVF01` for its name (see Figure 4.40).


ABAP Editor: Change Include ZXMGVU03

Include ZXMGVU03 Inactive (Revised)

Create Subroutine

Subroutine FILL_Z1MARA

INCLUDE Selection

ZXMGVF01	New Include

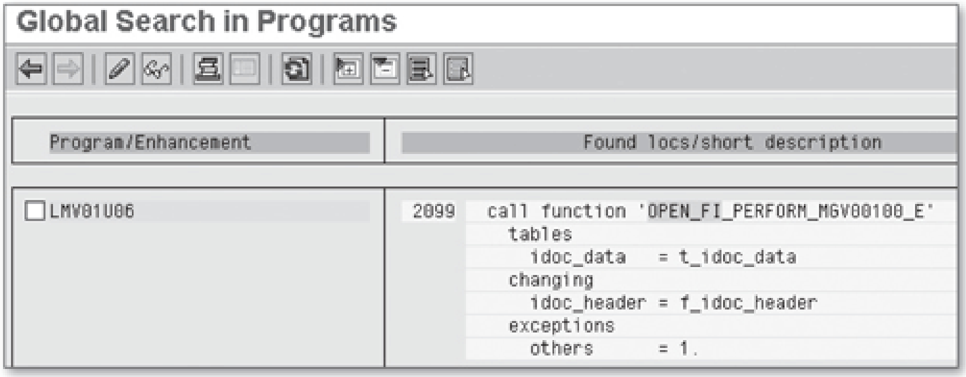
Figure 4.40 Customer Include

Business Transaction Events

Business transaction events (BTEs) are also handled via function modules but in a different way than customer exits. For BTEs, SAP calls in the delivered coding a function module using the `CALL FUNCTION 'OPEN_FI_PERFORM_XXXXXXXX_E'` command; here, `XXXXXXXX` is the *event ID*.

Determining BTEs

You can search for OPEN_FI* in the desired code to find the existing BTEs (see Figure 4.41). The specifications in the right field indicate if a BTE exists, where it's called, and which signature it provides. The BTE found here has ID MGV00100, is called only once, and provides all IDoc data together with the control record.



Global Search in Programs

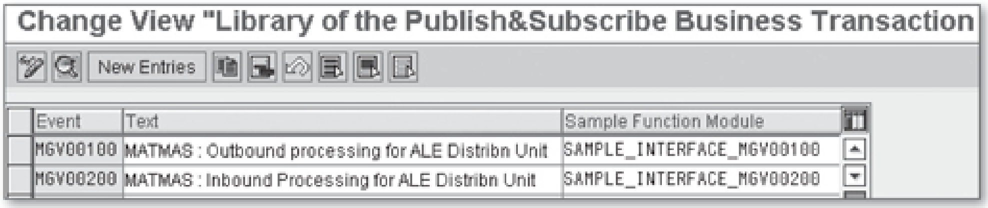
Program/Enhancement	Found locs/short description
□ LMV01U06	2099 call function 'OPEN_FI_PERFORM_MGV00100_E'

```
2099  call function 'OPEN_FI_PERFORM_MGV00100_E'
        tables
          idoc_data   = t_idoc_data
        changing
          idoc_header = f_idoc_header
        exceptions
          others      = 1.
```

Figure 4.41 Finding BTEs

Sample function module

In Transaction BF01, you can find the sample function modules that belong to P/S interfaces. Transaction BF05 specifies the sample function modules that belong to process interfaces. The sample function module that belongs to the MGV00100 BTE is the SAMPLE_INTERFACE_MGV00100 function module (see Figure 4.42). Always check both BTE types if you don't know which type the developer used.



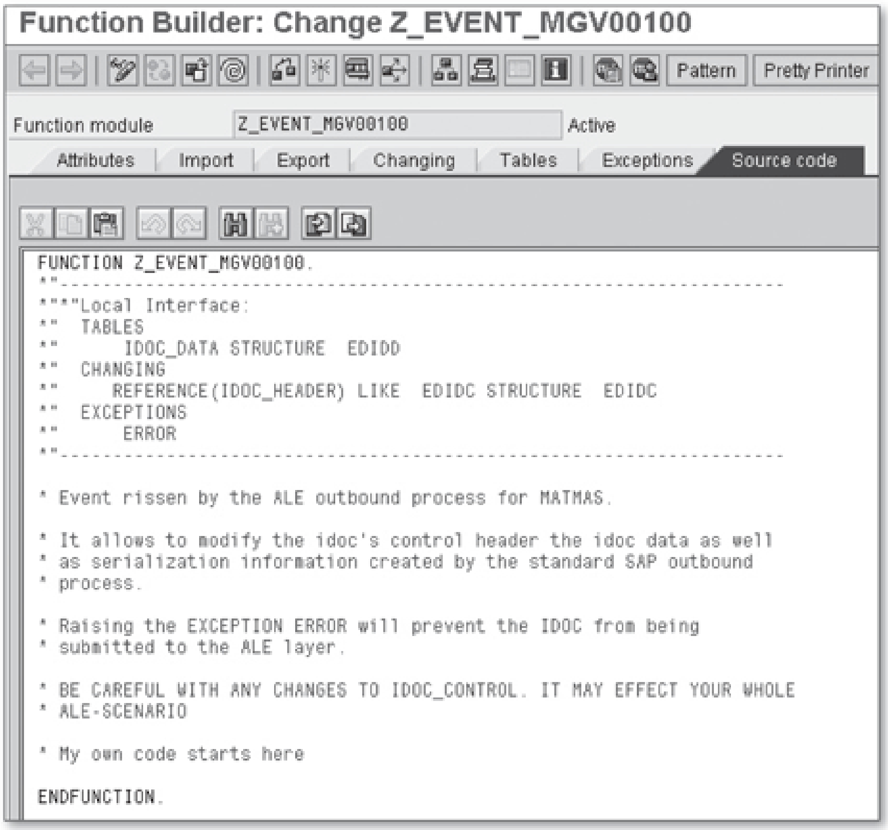
Change View "Library of the Publish&Subscribe Business Transaction

Event	Text	Sample Function Module
MGV00100	MATMAS : Outbound processing for ALE Distribn Unit	SAMPLE_INTERFACE_MGV00100
MGV00200	MATMAS : Inbound Processing for ALE Distribn Unit	SAMPLE_INTERFACE_MGV00200

Figure 4.42 Event with Sample Function Module in Transaction BF01

Creating your own module for BTEs

The sample function module contains the signature exactly in a format expected by the OPEN_FI_PERFORM ABAP statement of the calling program so that you can use it as a template. Copy this function module to your own namespace, and insert the desired code (indicated by * My own code starts here in Figure 4.43).



Function Builder: Change Z_EVENT_MGV00100

Function module Z_EVENT_MGV00100 Active

```
FUNCTION Z_EVENT_MGV00100.
*"----------------------------------------------------------------------
*""Local Interface:
*"  TABLES
*"      IDOC_DATA STRUCTURE  EDIDD
*"  CHANGING
*"     REFERENCE(IDOC_HEADER) LIKE  EDIDC STRUCTURE  EDIDC
*"  EXCEPTIONS
*"      ERROR
*"----------------------------------------------------------------------

* Event rissen by the ALE outbound process for MATMAS.

* It allows to modify the idoc's control header the idoc data as well
* as serialization information created by the standard SAP outbound
* process.

* Raising the EXCEPTION ERROR will prevent the IDOC from being
* submitted to the ALE layer.

* BE CAREFUL WITH ANY CHANGES TO IDOC_CONTROL. IT MAY EFFECT YOUR WHOLE
* ALE-SCENARIO

* My own code starts here

ENDFUNCTION.
```

Figure 4.43 Copy of the Sample Module

Customer product

To actually execute the corresponding function module, you have to activate it. However, this is done in a completely different way than for customer exits. The transaction for managing BTEs is Transaction FIBF. First of all, you have to create a customer management object by following the SETTINGS • PRODUCTS • ... OF A CUSTOMER menu path (see Figure 4.44).

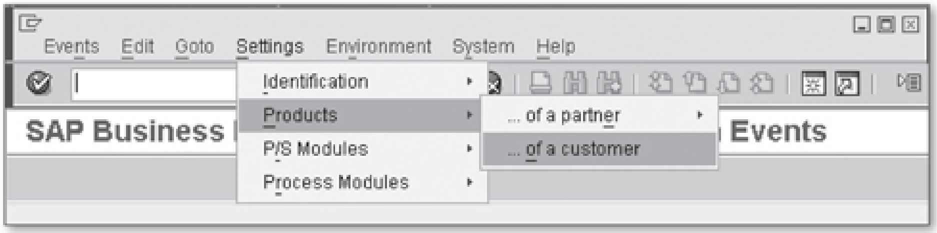
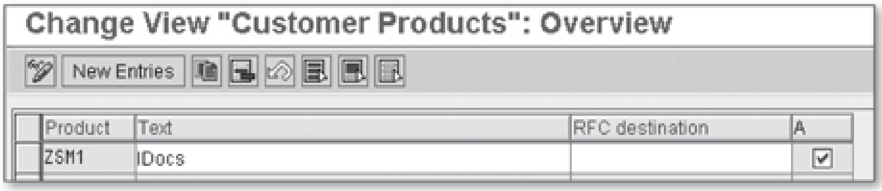


Figure 4.44 Menu Path for the Customer Product

The entries for the product mainly consist of the name (see Figure 4.45). Because SAP delivers products for particular tasks, and SAP partners use these BTEs in some products, you should be careful when assigning the names.

Activating the customer product for each client

The last field, with "A" as the heading, plays a particularly critical role because it's the activation column. The corresponding checkbox indicates whether the product in the current client is active (BTEs can be activated client-independently) so that the respective BTEs can be executed.



Change View "Customer Products": Overview

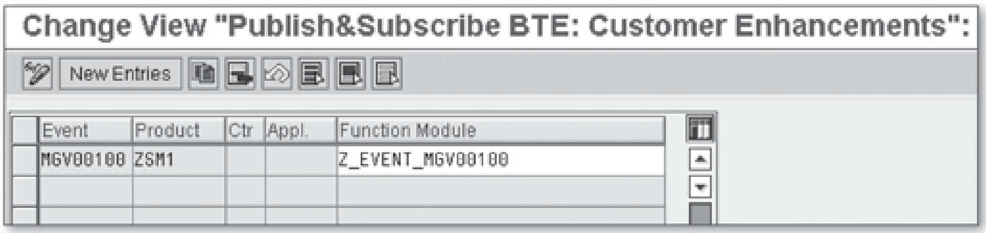
Product	Text	RFC destination	A
ZSM1	IDocs		☑

Figure 4.45 Creating a Customer Product

The RFC Destination field remains empty if you remain in the same system that executes the calling program. If you enter a valid destination from Transaction SM59 in this field, the events that are assigned to this product are executed on the system that belongs to this destination.

Assigning an event to the product

You assign your events to the product (combined according to whether all or none of these events are active) via the SETTINGS • P/S MODULES or PROCESS MODULES menu path. Figure 4.46 displays all required specifications.



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

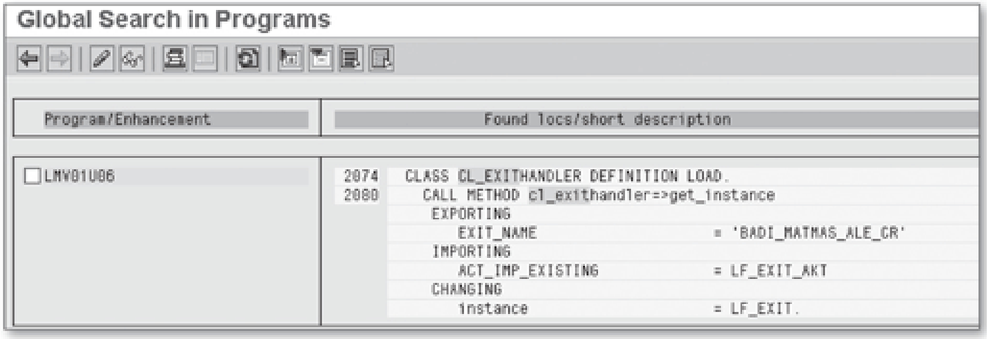
Event	Product	Ctr	Appl.	Function Module
MGV00100	ZSM1			Z_EVENT_MGV00100

Figure 4.46 Assigning Events to Products

Depending on the environment, you can use the entries made for country (Ctr) and application (Appl.) as the filter criteria; for IDocs, however, these restrictions don't have any effect, so the fields remain empty.

Classic BAdIs

Classic BAdIs are handled via a handler class. The name of this class is CL_EXITHANDLER, and its class method, GET_INSTANCE, is called once for each BAdI. If you want to know in a program if you can use a BAdI, search for "CL_EXITHANDLER." You can find the requested BAdI in the EXIT_NAME export variable. A corresponding instance variable called LF_EXIT is specified here (see Figure 4.47).



```
Program/Enhancement     Found locs/short description

□ LMV01U06     2074   CLASS CL_EXITHANDLER DEFINITION LOAD.
               2080     CALL METHOD cl_exithandler=>get_instance
                          EXPORTING
                            EXIT_NAME                = 'BADI_MATMAS_ALE_CR'
                          IMPORTING
                            ACT_IMP_EXISTING         = LF_EXIT_AKT
                          CHANGING
                            instance                 = LF_EXIT.
```

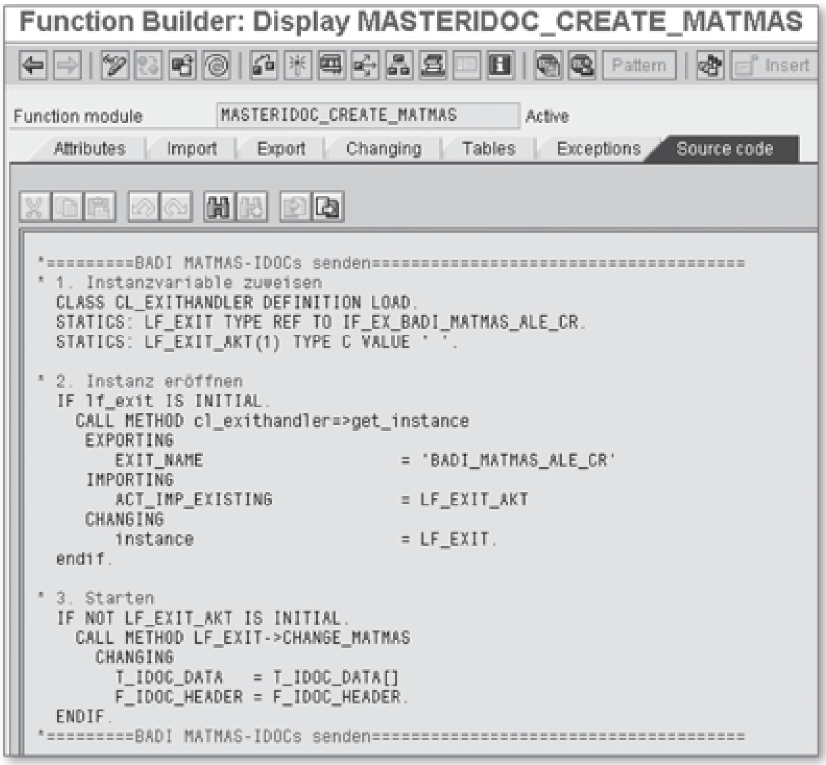
Figure 4.47 Identifying a Classic BAdI

Calling the BAdI

The name of the instance variable now enables you to determine which methods of the BAdI are called and where they are called. Here as well, SAP defines which data is transferred, which means the CHANGE_MATMAS method is called once, transfers all IDoc data and the control record, and both can be changed. Figure 4.48 shows the code provided by SAP for calling the classic BAdI.

Creating a BAdI implementation

BAdIs have to be delivered by SAP, so SAP creates the BAdI and the methods with your signature and calls them in the SAP program. Consequently, you only have to define what is supposed to be done in the BAdI. The BAdI is created in Transaction SE18. You can use Transaction SE19 to create an implementation for the BAdI where you define what you want to do with the provided data within the method. Figure 4.49 displays the initial screen of the transaction in SAP NetWeaver 7.0. In older releases, the New BAdI option was missing; everything else stays the same.

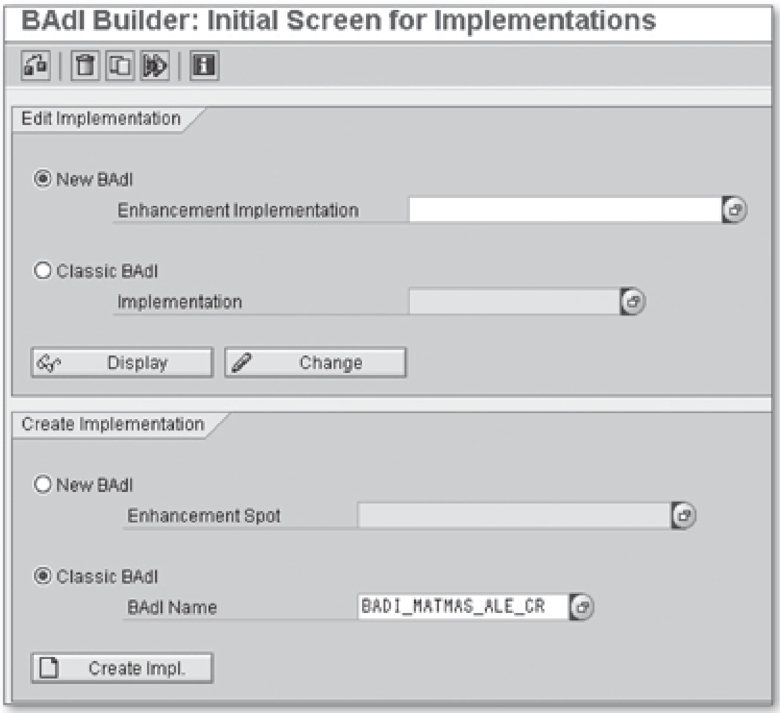


Function Builder: Display MASTERIDOC_CREATE_MATMAS

Function module MASTERIDOC_CREATE_MATMAS Active

```
*=========BADI MATMAS-IDOCs senden=====================================
* 1. Instanzvariable zuweisen
  CLASS CL_EXITHANDLER DEFINITION LOAD.
  STATICS: LF_EXIT TYPE REF TO IF_EX_BADI_MATMAS_ALE_CR.
  STATICS: LF_EXIT_AKT(1) TYPE C VALUE ' '.

* 2. Instanz eröffnen
  IF lf_exit IS INITIAL.
    CALL METHOD cl_exithandler=>get_instance
     EXPORTING
        EXIT_NAME                    = 'BADI_MATMAS_ALE_CR'
     IMPORTING
        ACT_IMP_EXISTING             = LF_EXIT_AKT
     CHANGING
        instance                     = LF_EXIT.
  endif.

* 3. Starten
  IF NOT LF_EXIT_AKT IS INITIAL.
    CALL METHOD LF_EXIT->CHANGE_MATMAS
      CHANGING
        T_IDOC_DATA   = T_IDOC_DATA[]
        F_IDOC_HEADER = F_IDOC_HEADER.
  ENDIF.
*=========BADI MATMAS-IDOCs senden=====================================
```

Figure 4.48 Calling the Classic BAdI



BAdI Builder: Initial Screen for Implementations

Edit Implementation

● New BAdI

Enhancement Implementation

○ Classic BAdI

Implementation

Create Implementation

○ New BAdI

Enhancement Spot

● Classic BAdI

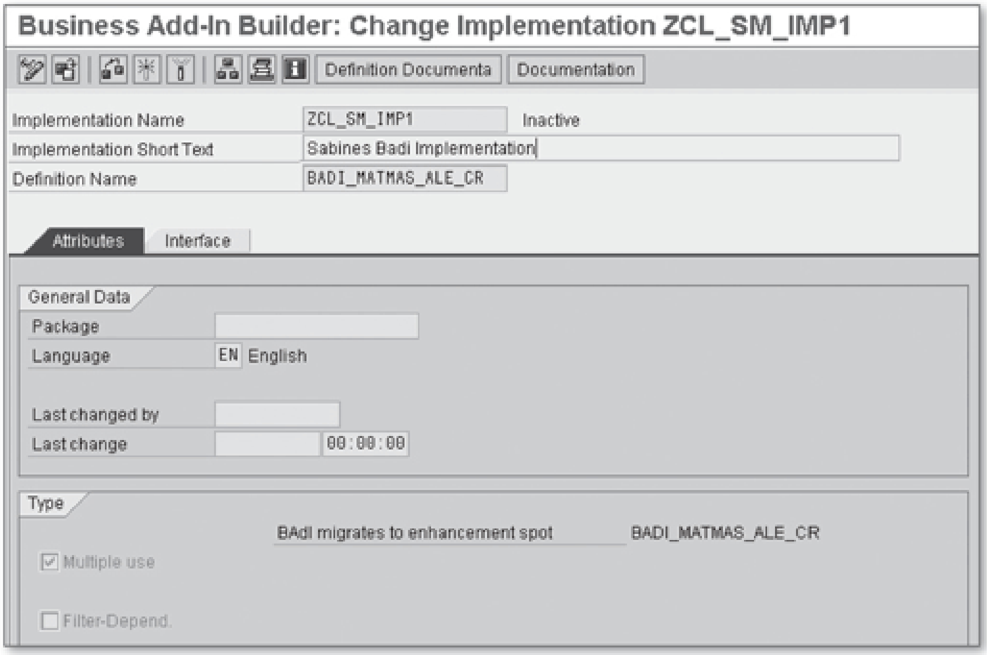
BAdI Name BADI_MATMAS_ALE_CR

Figure 4.49 Initial Screen for Creating an Implementation

Specify your classic BAdI in the Create Implementation tab, and click on the Create Implementation button to navigate to the next input screen. First, you need a name for your implementation (see Figure 4.50). The usual naming rules (Z, Y, or customer namespace as a prefix) apply. In addition, you should name the implementation like a class, for example, `ZCL_SM_IMP1`, because it's also displayed in the class library in Transaction SE24. Technically, it's just the implementing class of an interface with SAP creating the interface and you creating the implementation.

Attributes of the implementation

Here, it's also indicated whether the BAdI is or isn't Filter-Dependent or intended for Multiple use (see Figure 4.50, ATTRIBUTES • TYPE). The classic BAdI shown here (`BADI_MATMAS_ALE_CR`) has been migrated into a new BAdI in the meantime so that it would no longer be used in a new release. As an example, it's as good as any other, so we'll stick to the material master for the sake of clarity. Use the match icon (which is lit this time) to activate the BAdI at the end of your development work.



Business Add-In Builder: Change Implementation ZCL_SM_IMP1

Implementation Name	ZCL_SM_IMP1	Inactive
Implementation Short Text	Sabines Badi Implementation	
Definition Name	BADI_MATMAS_ALE_CR	

General Data

Package		
Language	EN	English
Last changed by		
Last change		00:00:00

Type

BAdI migrates to enhancement spot BADI_MATMAS_ALE_CR

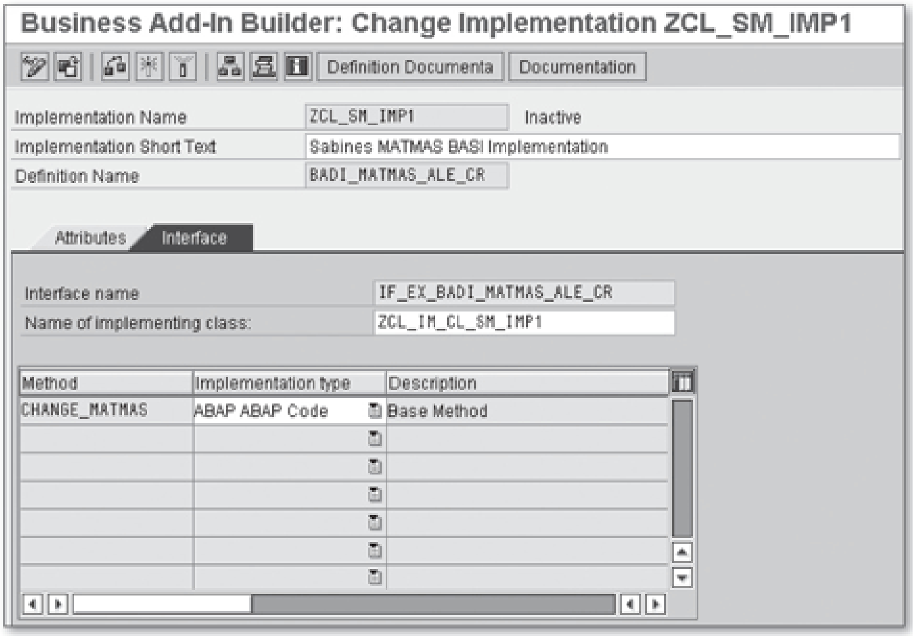
☑ Multiple use

□ Filter-Depend.

Figure 4.50 Creating the Implementation — Details

Methods of the BAdI

The Interfaces tab (see Figure 4.51) provides information on the methods of the BAdI. Double-clicking on a method takes you to the location where you can create your own code (see Figure 4.52).



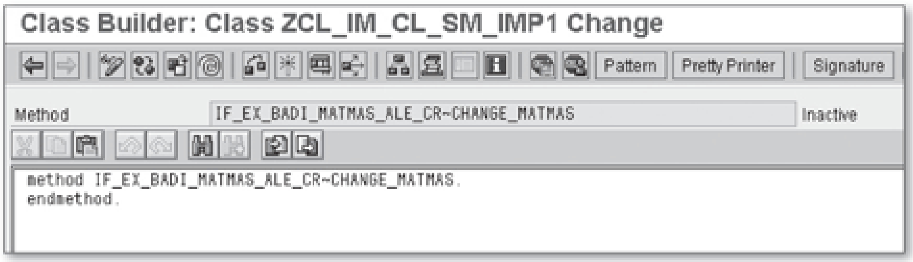
Business Add-In Builder: Change Implementation ZCL_SM_IMP1

Implementation Name	ZCL_SM_IMP1	Inactive
Implementation Short Text	Sabines MATMAS BASI Implementation	
Definition Name	BADI_MATMAS_ALE_CR	

Interface name	IF_EX_BADI_MATMAS_ALE_CR
Name of implementing class:	ZCL_IM_CL_SM_IMP1

Method	Implementation type	Description
CHANGE_MATMAS	ABAP ABAP Code	Base Method

Figure 4.51 Method View of the Implementation



Class Builder: Class ZCL_IM_CL_SM_IMP1 Change

Method	IF_EX_BADI_MATMAS_ALE_CR~CHANGE_MATMAS	Inactive

```
method IF_EX_BADI_MATMAS_ALE_CR~CHANGE_MATMAS.
endmethod.
```

Figure 4.52 Developing a Custom Method

Note that you must enable your code here. To actually execute the exit, you have to activate the implementation in the main screen of Transaction SE19.

The BAdI is cross-client; that is, your changes take effect in the entire SAP system.

Explicit Enhancements as of SAP ERP 6.0 with Basis 7.0

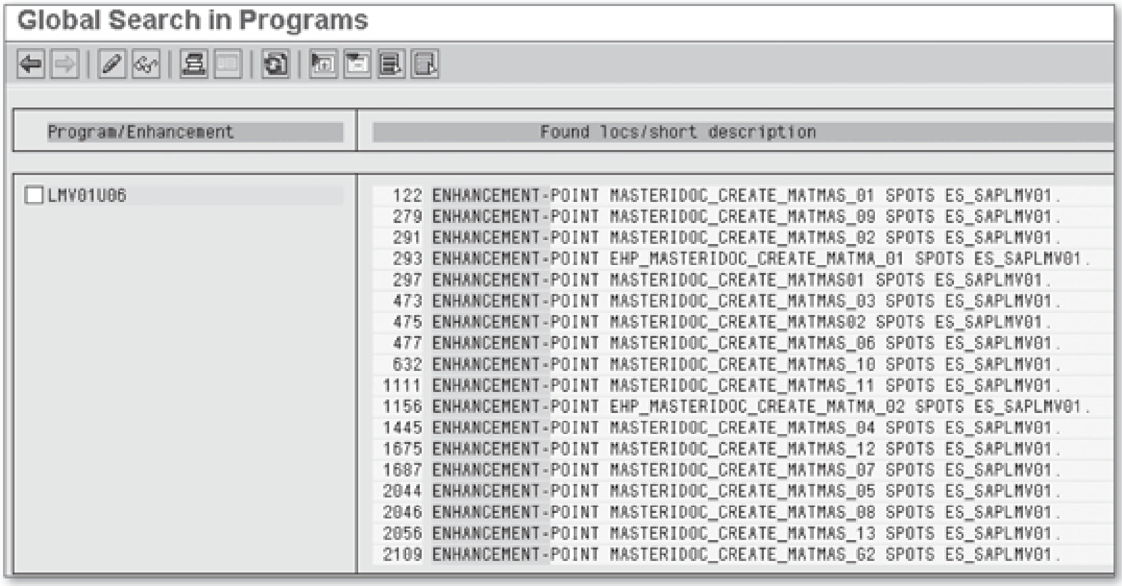
Since SAP NetWeaver Release 7.0, you can use a new enhancement technology. Let's have a look at the *explicit enhancements* first (implicit enhancements are described later in the section titled Implicit Enhancements as of SAP ERP Central Component 6.0 with Basis 7.0). SAP provides these enhancements at specific locations in the code so that you can use them.

Enhancement spot

Explicit enhancements are combined in *enhancement spots*. *Enhancement points* and *enhancement sections* are also available. There is no default code for enhancement points. You simply insert your desired additions where the enhancement point is located. For enhancement sections, SAP delivers a code that is executed as long as you don't create an implementation; however, you can replace it with your own implementation.

Searching for enhancements

You can easily find explicit enhancements by searching for the term "enhancement-." The hyphen is important; otherwise, the search results would also contain implementations because SAP often already delivers implementations, for example, for industry solutions. Figure 4.53 displays the search result for the enhancements of the MASTERIDOC_CREATE_MATMAS function module.



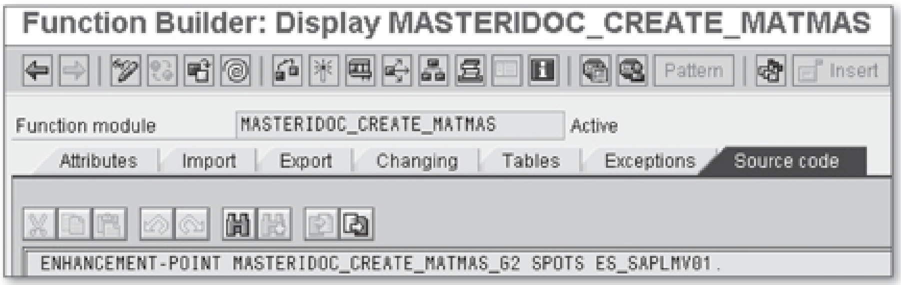
Global Search in Programs

```
Program/Enhancement     Found locs/short description

□ LMV01U06               122 ENHANCEMENT-POINT MASTERIDOC_CREATE_MATMAS_01 SPOTS ES_SAPLMV01.
                         279 ENHANCEMENT-POINT MASTERIDOC_CREATE_MATMAS_09 SPOTS ES_SAPLMV01.
                         291 ENHANCEMENT-POINT MASTERIDOC_CREATE_MATMAS_02 SPOTS ES_SAPLMV01.
                         293 ENHANCEMENT-POINT EHP_MASTERIDOC_CREATE_MATMA_01 SPOTS ES_SAPLMV01.
                         297 ENHANCEMENT-POINT MASTERIDOC_CREATE_MATMAS01 SPOTS ES_SAPLMV01.
                         473 ENHANCEMENT-POINT MASTERIDOC_CREATE_MATMAS_03 SPOTS ES_SAPLMV01.
                         475 ENHANCEMENT-POINT MASTERIDOC_CREATE_MATMAS02 SPOTS ES_SAPLMV01.
                         477 ENHANCEMENT-POINT MASTERIDOC_CREATE_MATMAS_06 SPOTS ES_SAPLMV01.
                         632 ENHANCEMENT-POINT MASTERIDOC_CREATE_MATMAS_10 SPOTS ES_SAPLMV01.
                        1111 ENHANCEMENT-POINT MASTERIDOC_CREATE_MATMAS_11 SPOTS ES_SAPLMV01.
                        1156 ENHANCEMENT-POINT EHP_MASTERIDOC_CREATE_MATMA_02 SPOTS ES_SAPLMV01.
                        1445 ENHANCEMENT-POINT MASTERIDOC_CREATE_MATMAS_04 SPOTS ES_SAPLMV01.
                        1675 ENHANCEMENT-POINT MASTERIDOC_CREATE_MATMAS_12 SPOTS ES_SAPLMV01.
                        1687 ENHANCEMENT-POINT MASTERIDOC_CREATE_MATMAS_07 SPOTS ES_SAPLMV01.
                        2044 ENHANCEMENT-POINT MASTERIDOC_CREATE_MATMAS_05 SPOTS ES_SAPLMV01.
                        2046 ENHANCEMENT-POINT MASTERIDOC_CREATE_MATMAS_08 SPOTS ES_SAPLMV01.
                        2056 ENHANCEMENT-POINT MASTERIDOC_CREATE_MATMAS_13 SPOTS ES_SAPLMV01.
                        2109 ENHANCEMENT-POINT MASTERIDOC_CREATE_MATMAS_62 SPOTS ES_SAPLMV01.
```

Figure 4.53 Finding Explicit Enhancements

Calling the BAdI

The code itself indicates the calling point and the name of the corresponding enhancement point (see Figure 4.54). Enhancement points and enhancement sections don't have a signature for the transfer of data. That means you can access all variables within the enhancement that are available at this location in the code of the calling program. However, this also means that it can be complicated to determine the names of the respective variables and which data is available.



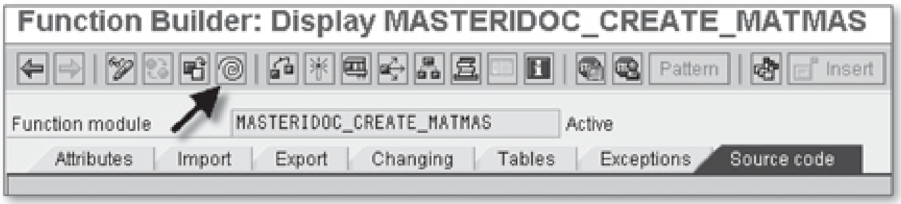
Function module MASTERIDOC_CREATE_MATMAS Active

```
ENHANCEMENT-POINT MASTERIDOC_CREATE_MATMAS_G2 SPOTS ES_SAPLMV01 .
```

Figure 4.54 Enhancement in the Source Code

Enhancements: activating the editing function

If you now want to implement an explicit enhancement, you can't use the standard editing function. You have to inform the system that you don't want to edit the original but instead want to edit your own implementation. Figure 4.55 shows the icon that you can use for this purpose.



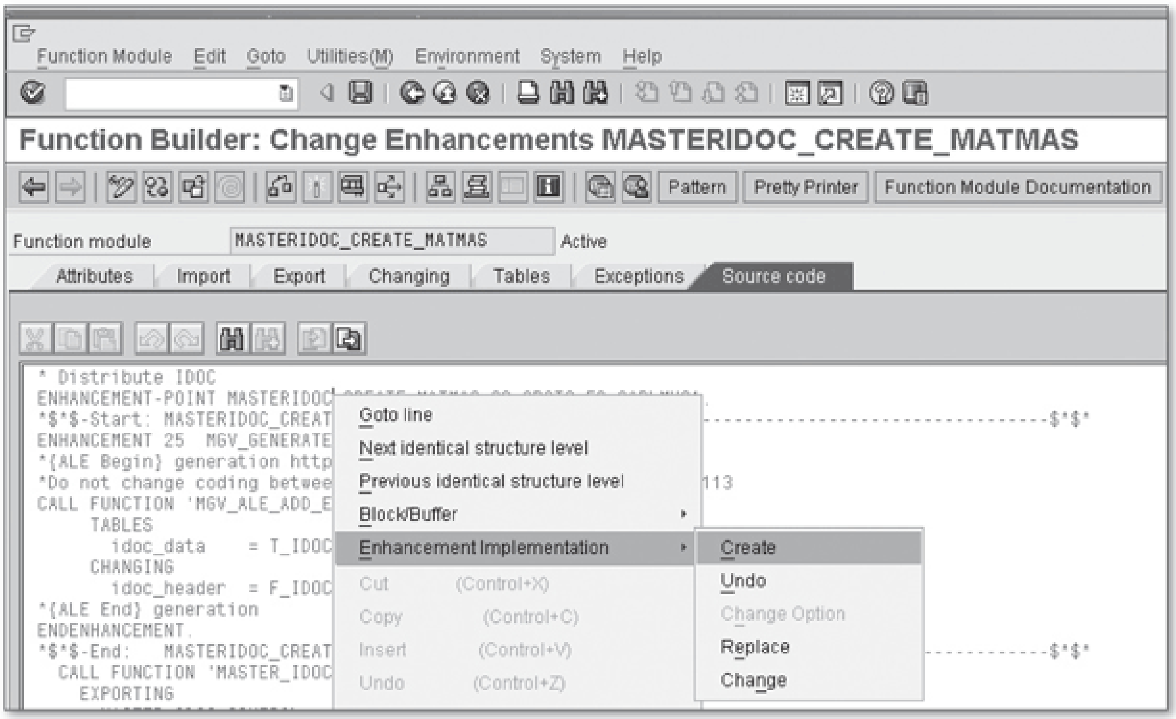
Function module MASTERIDOC_CREATE_MATMAS Active

Figure 4.55 Activating the Enhancement Editing Function

Enhancement functions

After enabling the editing of enhancements, you can set the mouse pointer to "enhancement" in the code and open the context menu by right-clicking on it. Now, you can create, change, or undo implementations using the Enhancement Implementation menu item (see Figure 4.56).

Follow the ENHANCEMENT IMPLEMENTATION • CREATE menu path. Because SAP also implements enhancement spots, it's possible that the SAP namespace already displays implementations (Figure 4.57). For the names of the implementations, the rule that they must begin with "Z," "Y," or the customer namespace applies again.

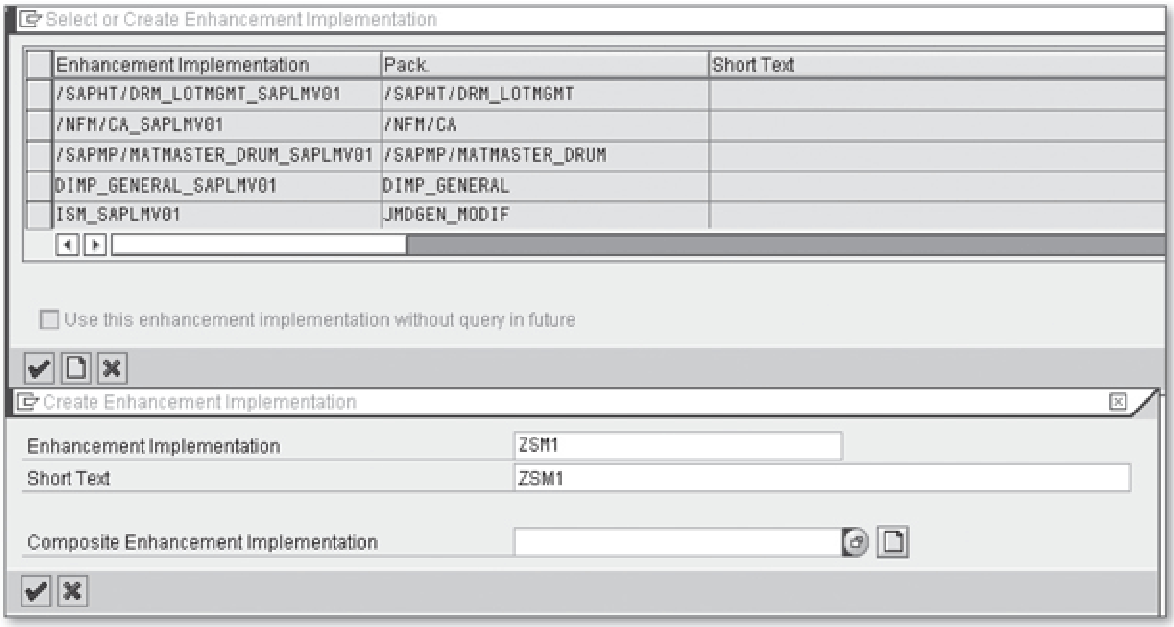


Function Builder: Change Enhancements MASTERIDOC_CREATE_MATMAS

Function module MASTERIDOC_CREATE_MATMAS Active

```
* Distribute IDOC
ENHANCEMENT-POINT MASTERIDOC
*$*$-Start: MASTERIDOC_CREAT                                   ---------------------------$*$*
ENHANCEMENT 25  MGV_GENERATE
*{ALE Begin} generation http
*Do not change coding betwee                                   113
CALL FUNCTION 'MGV_ALE_ADD_E
     TABLES
       idoc_data    = T_IDOC
     CHANGING
       idoc_header  = F_IDOC
*{ALE End} generation
ENDENHANCEMENT.
*$*$-End:   MASTERIDOC_CREAT                                   ------------$*$*
  CALL FUNCTION 'MASTER_IDOC
    EXPORTING
```

Figure 4.56 Enhancement Implementation (Context Menu)



Select or Create Enhancement Implementation

Enhancement Implementation	Pack.	Short Text
/SAPHT/DRM_LOTMGMT_SAPLMV01	/SAPHT/DRM_LOTMGMT	
/NFM/CA_SAPLMV01	/NFM/CA	
/SAPMP/MATMASTER_DRUM_SAPLMV01	/SAPMP/MATMASTER_DRUM	
DIMP_GENERAL_SAPLMV01	DIMP_GENERAL	
ISM_SAPLMV01	JMDGEN_MODIF	

□ Use this enhancement implementation without query in future

Create Enhancement Implementation

Enhancement Implementation: ZSM1

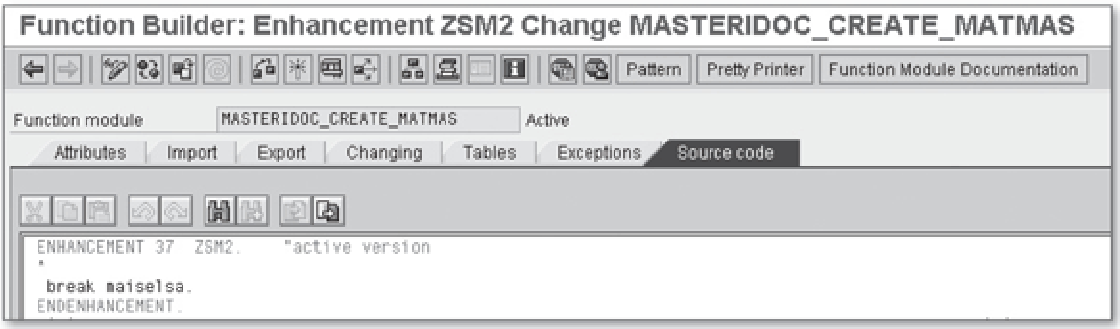
Short Text: ZSM1

Composite Enhancement Implementation:

Figure 4.57 Creating an Implementation

BAdI implementation

Enter the name (here: "ZSM1") in the Enhancement Implementation field as well as a description in the Short Text field, and confirm your input. After the implementation has been created, you can enter code within a defined area (Figure 4.58).


Function Builder: Enhancement ZSM2 Change MASTERIDOC_CREATE_MATMAS
Pattern
Pretty Printer
Function Module Documentation
Attributes
Import
Export
Changing
Tables
Exceptions
Source code

Function module: MASTERIDOC_CREATE_MATMAS Active

```
ENHANCEMENT 37  ZSM2.    "active version
*
 break maiselsa.
ENDENHANCEMENT.
```

Figure 4.58 Editing an Enhancement Implementation

Implementation code

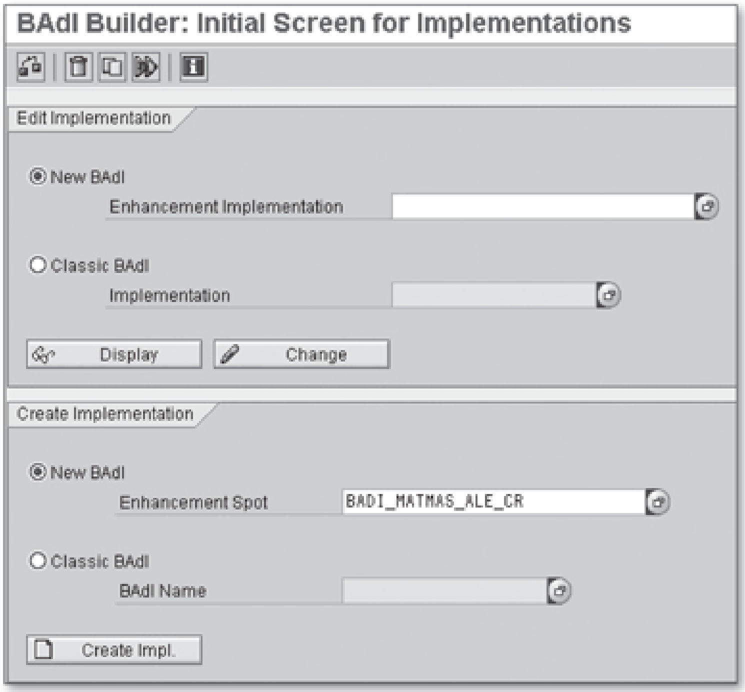
Bear in mind that the activation icon shown here only activates your part of the code. The *breakpoint*, which was entered as the only command in the example, helps you determine which variables are available, but it should always be removed or replaced by the desired program behavior as soon as possible.

New BAdIs as of SAP ERP 6.0 with Basis 7.0

The benefit of new BAdIs, which have been provided since SAP ERP 6.0, is that they have a higher performance than classic BAdIs and that they are connected to enhancement spots. Their structure, however, is somewhat different from the structure of classic BAdIs. There are two options for using new BAdIs. The first option was illustrated in the material master example: A new BAdI was made from migrating a classic BAdI. In this case, the classic BAdI calls the new BAdI dynamically. You can no longer find the BAdI name by browsing the source code. The classic BAdI, however, indicates the name of the new BAdI next to BAdI Migrates to Enhancement Spot (see Figure 4.50). Considering this new name, navigate to Transaction SE19 to initiate the actual implementation.

Implementing new BAdIs

The second option is that the new BAdI has actually been created. You can find it by searching for the `GET*BADI` command. The new BAdI is also implemented in Transaction SE19, but the next screens are considerably different from the screens of the implementation of classic BAdIs. Consequently, an example of this is also provided next. It's critical that you specify the name of the enhancement spot, and not the BAdI name for new BAdIs (Figure 4.59).


BAdI Builder: Initial Screen for Implementations

Edit Implementation

● New BAdI
Enhancement Implementation

○ Classic BAdI
Implementation

Create Implementation

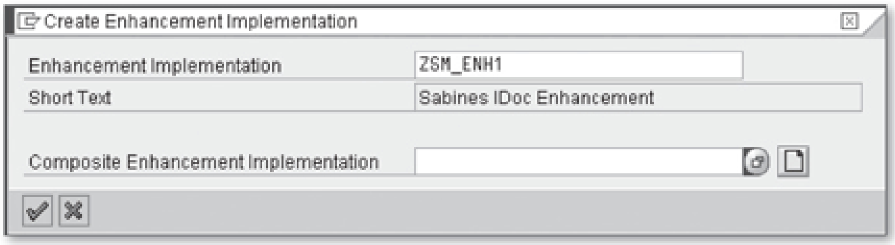
● New BAdI
Enhancement Spot: BADI_MATMAS_ALE_CR

○ Classic BAdI
BAdI Name

Figure 4.59 Initial Screen of the BAdI Builder (Transaction SE19)

Name of the implementation

Next, enter the name of the enhancement implementation in Transaction SE19. This name enables you to always easily find the enhancement in Transaction SE80, that is, the development environment. Figure 4.60 shows that composite enhancement implementations are also possible if you want to combine individual objects. You can then transport or enable these composite enhancements together in SAP's switch framework. You would combine items that are located in different SAP enhancements but belong to one customer development. They are solely classification items.


Create Enhancement Implementation

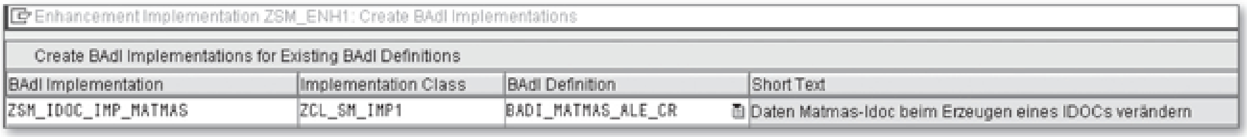
Enhancement Implementation: ZSM_ENH1
Short Text: Sabines IDoc Enhancement

Composite Enhancement Implementation

Figure 4.60 Assigning a Name to the Implementation

Implementing class

For new BAdIs, too, you must first assign a name to the enhancement implementation in Transaction SE19. Using this name, you can also modify your enhancements in this transaction at a later stage (here: ZSM_IDOC_IMP_MATMAS). You also need the specification for the implementation class (ZCL_SM_IMP1), which should correspond to the class name rules if possible (see Figure 4.61).



Enhancement Implementation ZSM_ENH1: Create BAdI Implementations

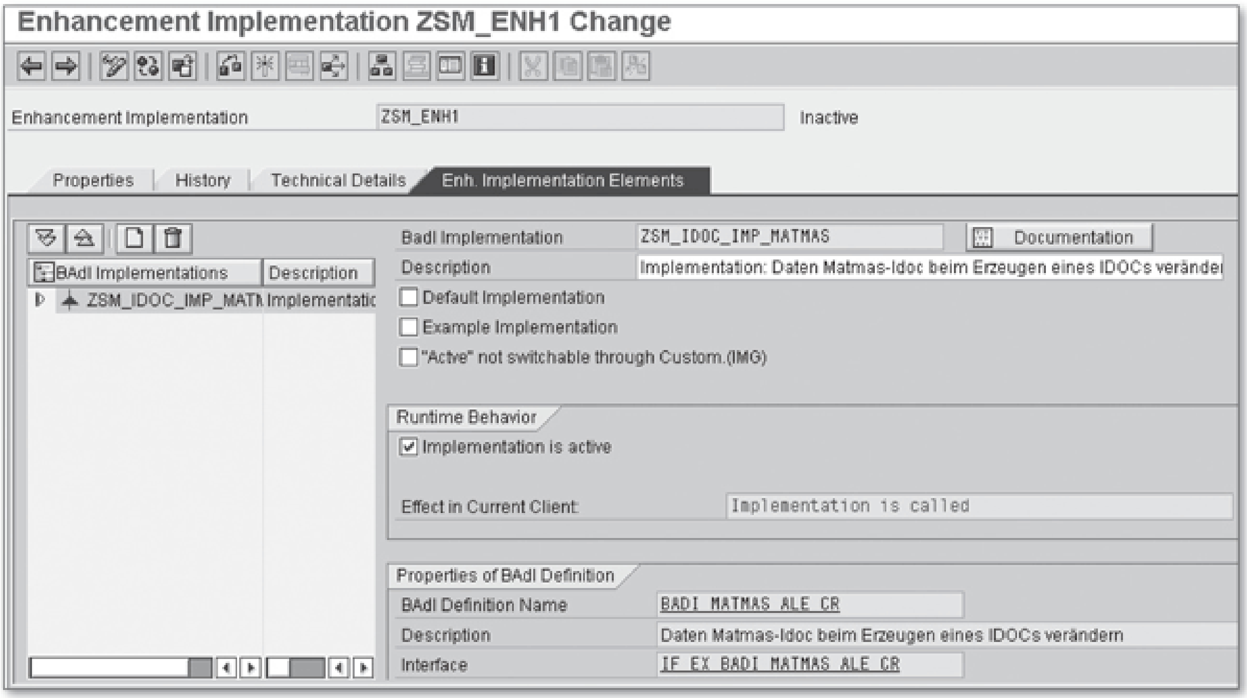
Create BAdI Implementations for Existing BAdI Definitions

BAdI Implementation	Implementation Class	BAdI Definition	Short Text
ZSM_IDOC_IMP_MATMAS	ZCL_SM_IMP1	BADI_MATMAS_ALE_CR	Daten Matmas-Idoc beim Erzeugen eines IDOCs verändern

Figure 4.61 Name of the Implementing Class

Activating the implementation

The next screen now displays the properties of your new implementation (see Figure 4.62). If everything is done, you can enable it using the Implementation Is Active checkbox. New BAdIs also allow you to first complete the development work and then ensure that the code is executed.



Enhancement Implementation ZSM_ENH1 Change

Enhancement Implementation: ZSM_ENH1 Inactive

Badl Implementation: ZSM_IDOC_IMP_MATMAS

Description: Implementation: Daten Matmas-Idoc beim Erzeugen eines IDOCs verände

☐ Default Implementation

☐ Example Implementation

☐ "Actve" not switchable through Custom.(IMG)

Runtime Behavior

☑ Implementation is active

Effect in Current Client: Implementation is called

Properties of BAdI Definition

BAdI Definition Name: BADI_MATMAS_ALE_CR

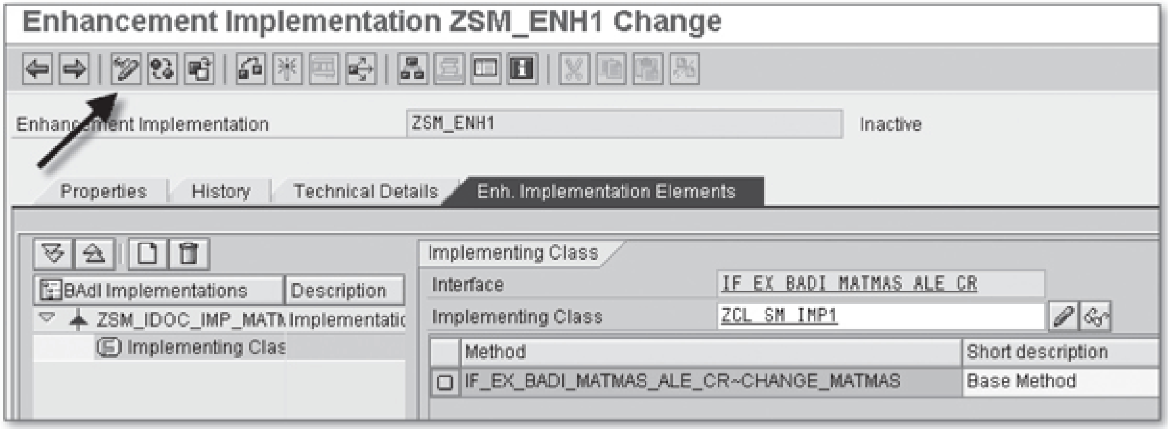
Description: Daten Matmas-Idoc beim Erzeugen eines IDOCs verändern

Interface: IF_EX_BADI_MATMAS_ALE_CR

Figure 4.62 Properties of the Implementation

Maintaining the implementing class

To view the implementing class, double-click on Implementing Class (on the left in Figure 4.63). On the right, the system then displays the implementing method that is copied by the BAdI interface (here: IF_EX_BADI_MATMAS_ALE_CR~CHANGE_MATMAS). Clicking the change icon (pen) takes you to the view of your class.


Enhancement Implementation ZSM_ENH1 Change

Enhancement Implementation: ZSM_ENH1 Inactive

BAdI Implementations	Description
ZSM_IDOC_IMP_MATM	Implementatic
Implementing Clas	

Implementing Class

Interface: IF_EX_BADI_MATMAS_ALE_CR

Implementing Class: ZCL_SM_IMP1

Method	Short description
IF_EX_BADI_MATMAS_ALE_CR~CHANGE_MATMAS	Base Method

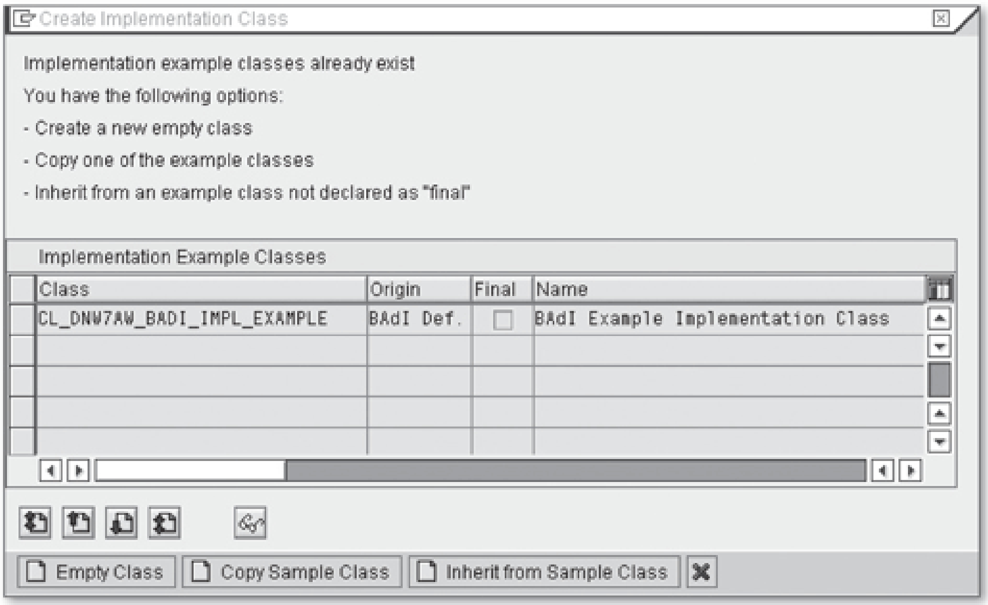
Figure 4.63 Navigation to the Implementing Class

Sample class for new BAdIs

New BAdIs enable the developer of the BAdI to provide a sample class. This way, he can easily indicate the intended usage options for his BAdI. This function is optional; that is, it isn't provided by all BAdIs. To illustrate this function, the example in Figure 4.64 displays the corresponding screenshot of a different BAdI. The figure shows the respective query as well as the available options. You can basically decide if you want to use an empty class, use the sample class as a template, or have a class inherit from the sample class. The benefit of having your BAdI inherit from the sample class is that you're automatically provided with potential changes in recent releases.

Sample method for new BAdIs

Whether you query the sample method or not — the next step always takes you to the class maintenance (see Figure 4.65) from where you can navigate to the source code of your method by double-clicking on the method in the Methods tab.



Create Implementation Class

Implementation example classes already exist

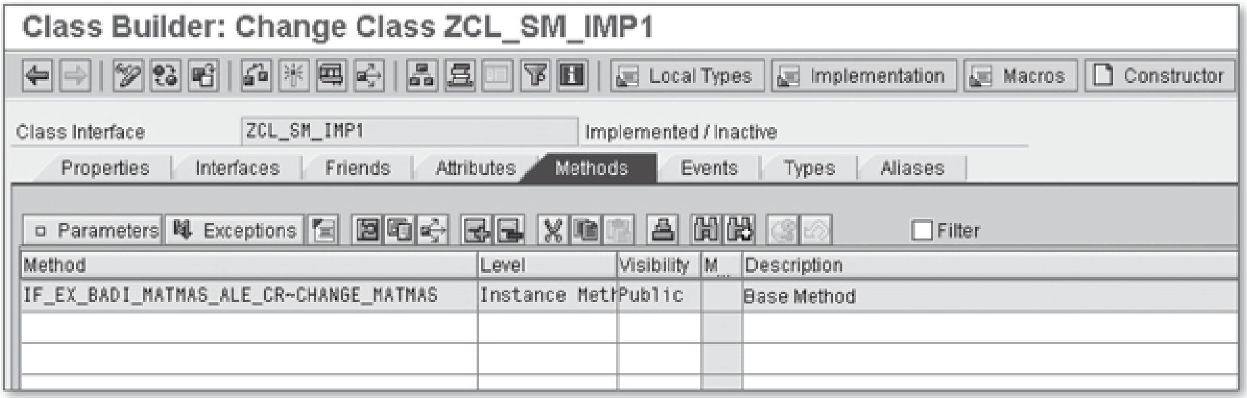
You have the following options:

- Create a new empty class
- Copy one of the example classes
- Inherit from an example class not declared as "final"

Implementation Example Classes

Class	Origin	Final	Name
CL_DNW7AW_BADI_IMPL_EXAMPLE	BAdI Def.	□	BAdI Example Implementation Class

Figure 4.64 Optional Sample Method



Class Builder: Change Class ZCL_SM_IMP1

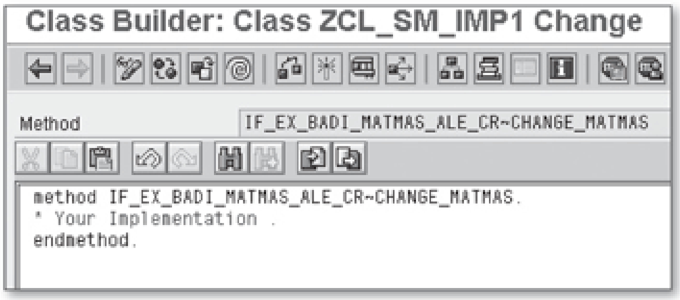
Class Interface ZCL_SM_IMP1 Implemented / Inactive

Method	Level	Visibility	M..	Description
IF_EX_BADI_MATMAS_ALE_CR~CHANGE_MATMAS	Instance Meth	Public		Base Method

Figure 4.65 Navigation to the Implementing Class

Class and method maintenance

Figure 4.66 displays a field in which you can enter your source code, that is, the complete functions, for example, to populate the custom segment (in our example, only a comment, `* Your Implementation`, was inserted for the sake of clarity).



Method IF_EX_BADI_MATMAS_ALE_CR~CHANGE_MATMAS

```
method IF_EX_BADI_MATMAS_ALE_CR~CHANGE_MATMAS.
* Your Implementation .
endmethod.
```

Figure 4.66 Implementing the BAdI Method

Implicit Enhancements as of SAP ERP 6.0 with Basis 7.0

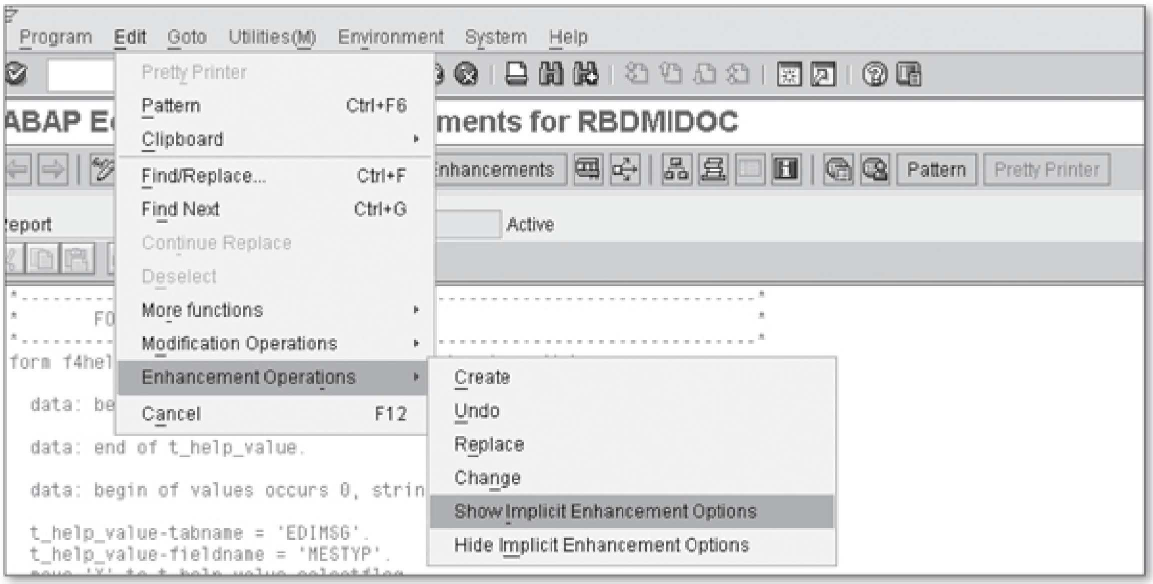
In addition to the enhancements described so far, the new enhancement options also include *implicit enhancements*. These are enhancements that aren't predefined by the SAP developer. Instead, they are available at specific locations in all programs that aren't part of the actual SAP Basis. For function modules and includes as they are provided in IDoc processing, these are the following locations:

- At the end of an include (reports are considered includes as well)
- In the signature of function modules
- At the end of structure type definitions (before `end of`)
- At the beginning and end of forms

There are some more locations, particularly in the context of classes, but they aren't relevant for IDoc processing.

Displaying implicit enhancements

Implicit enhancements are available in every program and are hidden at first. Only if you inform the system that you want to use implicit enhancements does the system display the enhancements so that you can edit them. First, click the enhancement icon ◎. In the screen that now opens (see Figure 4.67), follow the Edit • Enhancement Operations • Show Implicit Enhancement Options menu path to display the implicit enhancements.



```
*---------          ----------------------------*
*       FO                                      *
*---------          ----------------------------*
form f4hel
  data: be
  data: end of t_help_value.
  data: begin of values occurs 0, strin
  t_help_value-tabname = 'EDIMSG'.
  t_help_value-fieldname = 'MESTYP'.
```

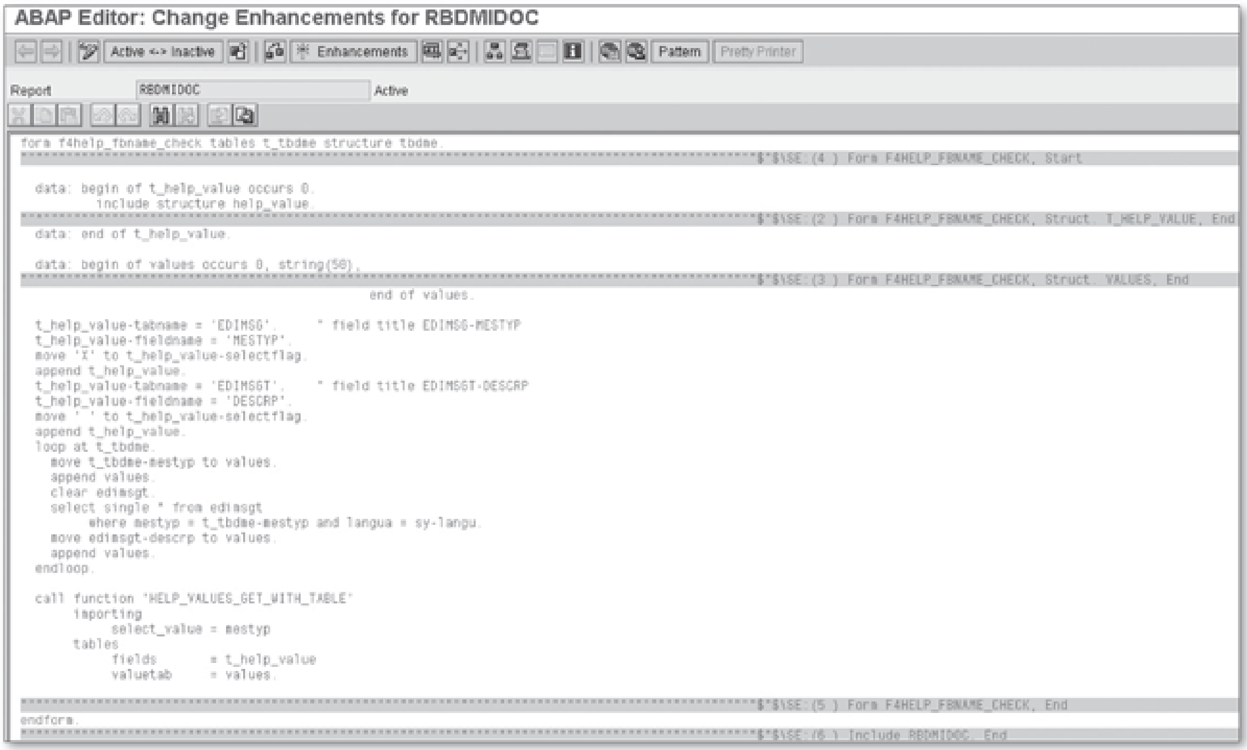
Figure 4.67 Displaying Implicit Enhancements

Implicit enhancements in the "RBDMIDOC" report

The program text then displays additional lines that consist of quotation marks and names. These are the implicit enhancements. Because they are available in all programs, you need to generate their names. Their names always consist of a sequential number and a description that indicates the location to which they belong (e.g., at the end of a form). Figure 4.68 displays implicit enhancement both at the beginning and end of a form and in a data statement as well as at the end of the include (the RBDMIDOC report here). This is indicated by quotation marks, which run across the entire line.

Implementing an implicit enhancement

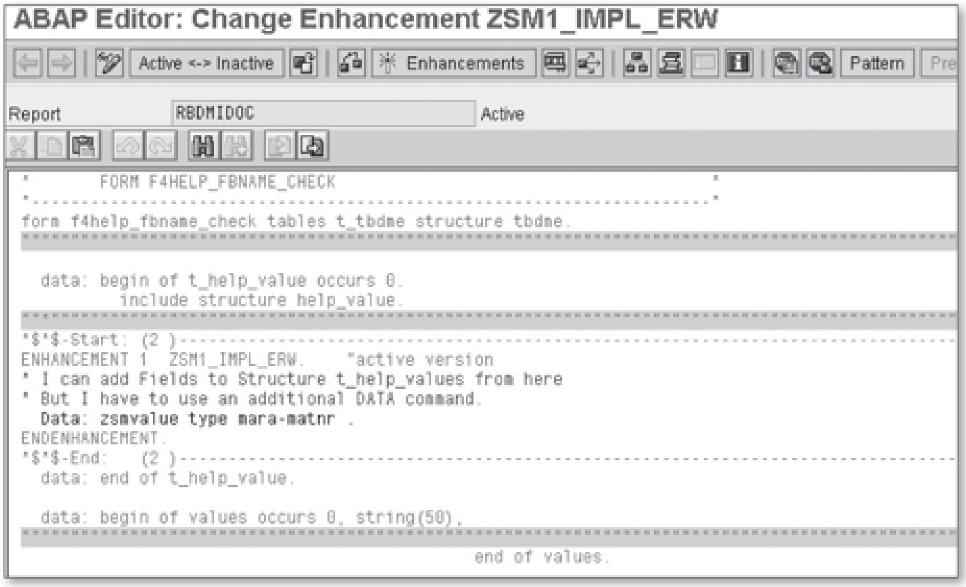
An implicit enhancement is implemented in the same way as an explicit enhancement. An implementation that has already been carried out remains visible (see Figure 4.69) even if the display of implicit enhancements is disabled again, for example, after Transaction SE38 has been recalled.



Report RBDMIDOC Active

```
form f4help_fbname_check tables t_tbdme structure tbdme.
********************************************************************************$"$\SE:(4 ) Form F4HELP_FBNAME_CHECK, Start

  data: begin of t_help_value occurs 0.
          include structure help_value.
********************************************************************************$"$\SE:(2 ) Form F4HELP_FBNAME_CHECK, Struct. T_HELP_VALUE, End
  data: end of t_help_value.

  data: begin of values occurs 0, string(50),
********************************************************************************$"$\SE:(3 ) Form F4HELP_FBNAME_CHECK, Struct. VALUES, End
                                               end of values.

  t_help_value-tabname = 'EDIMSG'.      " field title EDIMSG-MESTYP
  t_help_value-fieldname = 'MESTYP'.
  move 'X' to t_help_value-selectflag.
  append t_help_value.
  t_help_value-tabname = 'EDIMSGT'.     " field title EDIMSGT-DESCRP
  t_help_value-fieldname = 'DESCRP'.
  move ' ' to t_help_value-selectflag.
  append t_help_value.
  loop at t_tbdme.
    move t_tbdme-mestyp to values.
    append values.
    clear edimsgt.
    select single * from edimsgt
         where mestyp = t_tbdme-mestyp and langua = sy-langu.
    move edimsgt-descrp to values.
    append values.
  endloop.

  call function 'HELP_VALUES_GET_WITH_TABLE'
       importing
            select_value = mestyp
       tables
            fields       = t_help_value
            valuetab     = values.

********************************************************************************$"$\SE:(5 ) Form F4HELP_FBNAME_CHECK, End
endform.
********************************************************************************$"$\SE:(6 ) Include RBDMIDOC, End
```

Figure 4.68 Implicit Enhancements in the "RBDMIDOC" Report



Report RBDMIDOC Active

```
*       FORM F4HELP_FBNAME_CHECK                                      *
*---------------------------------------------------------------------*
form f4help_fbname_check tables t_tbdme structure tbdme.
*********************************************************************************

  data: begin of t_help_value occurs 0.
          include structure help_value.
*********************************************************************************
"$"$-Start: (2 )-----------------------------------------------------------------
ENHANCEMENT 1  ZSM1_IMPL_ERW.    "active version
* I can add Fields to Structure t_help_values from here
* But I have to use an additional DATA command.
  Data: zsmvalue type mara-matnr .
ENDENHANCEMENT.
"$"$-End:   (2 )-----------------------------------------------------------------
  data: end of t_help_value.

  data: begin of values occurs 0, string(50),
*********************************************************************************
                                               end of values.
```

Figure 4.69 Implemented Implicit Enhancement

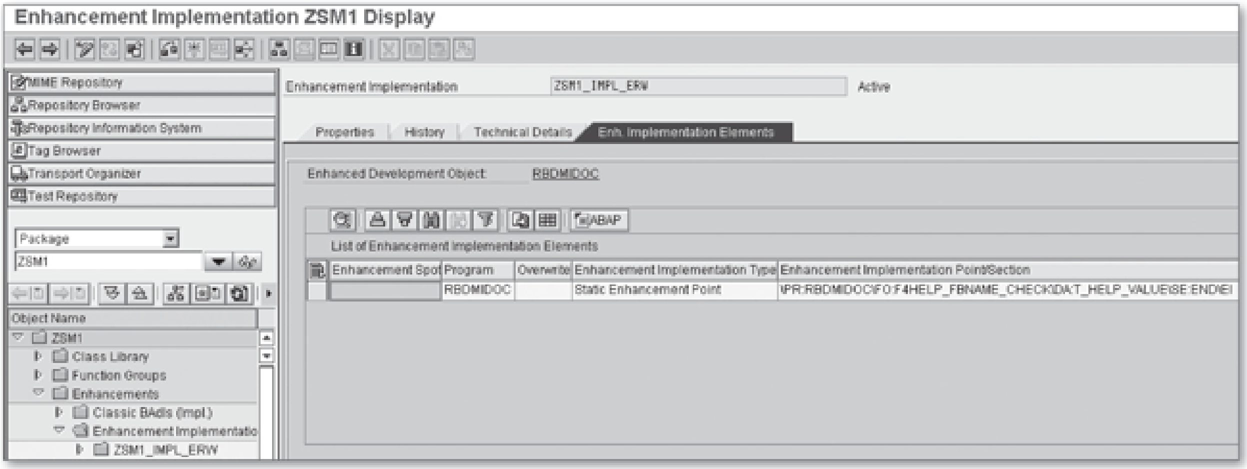
Special aspects of structures

The sample implementation of an implicit enhancement was deliberately selected for a structure definition here. For implicit enhancements, the original SAP code and your enhancement code are available as specific programming objects and must also be valid and can be activated as specific programming objects. The runtime environment doesn't insert them at the appropriate location until the main program has been executed. Because of this, the implementation requires a second `Data` statement.

Enhancement in the runtime environment

You can view all enhancements that belong to the new enhancement technology in Transaction SE80. Figure 4.70 displays our implicit enhancement, `ZSM1_IMPL_ERW`, with reference to the report name or the include name in the Program column and the location of the enhancements in the code (in the Enhancement Point/Section column).

Static enhancements refer to code for the data declaration; all other enhancements are *dynamic*.



Enhancement Implementation ZSM1 Display

Package: ZSM1

Enhancement Implementation: ZSM1_IMPL_ERW Active

Enhanced Development Object: RBDMIDOC

List of Enhancement Implementation Elements

Enhancement Spot	Program	Overwrite	Enhancement Implementation Type	Enhancement Implementation Point/Section
	RBDMIDOC		Static Enhancement Point	\PR:RBDMIDOC\FO:F4HELP_FBNAME_CHECK\DA:T_HELP_VALUE\SE:END\EI

Figure 4.70 Enhancement in Transaction SE80

Special aspects of ALE services

Enhancements and ALE Services

Irrespective of the technology you implement your enhancements with, note that this is always done during the generation of the IDoc in the outbound processing. In general, all ALE services aren't executed until the IDoc has been entirely generated, including your enhancements, so potential filtering processes or changes aren't implemented until your enhancements have been created. The IDoc may still contain various plant segments during the runtime of your exit even if the system — due to the object filtering — generates only one plant segment on the database at the end of the IDoc creation. Consider this when implementing your exits.

4.2.2 General Exits

Some operations are handled in the same way for most of the IDocs. This also applies to the version conversion, which can be configured in Transaction WE20. For master data IDocs, these operations also include the writing of change pointers. The next section introduces the enhancement objects that SAP provides for this purpose.

Version Conversion

"ALE00001" SAP enhancement

Depending on the release of the SAP system, there can be a specific version for each segment. For the communication with previous releases, it may be necessary to generate the segments with a release version that you can freely define in Transaction WE20 (see also Section 4.1.6, Version Conversion). This can be done without any development work. In addition, however, SAP provides an SAP enhancement in Transaction SMOD to enable customers to implement their own functionality.

The EXIT_SAPLBD11_001 exit in the ALE00001 enhancement is continuously executed even if no version conversion is specified in Transaction WE20. It's one of the few exits that also deliver header and status data of the IDoc and can be changed if required. The customer include in which you insert your code is called ZXSBDU01 and is an exit in an SAP enhancement. The sample code shows how you can determine the customer ID in the SAP system from the ILN (International Location Number) in the IDoc (see Listing 4.1).

In general, this exit can be executed in the inbound and outbound processing, which means you have to define the direction of the communication first. In the next step, you can make the required changes (to the control record or the IDoc data). Finally, you must change the following fields:

Return changes

- idoc_control_out-idoctp
- idoc_control_out-upddat
- idoc_control_out-updtim

The program that calls the exit checks whether these field have been changed. Only then does the system copy the other, potentially changed values from the idoc_control_out structure. It's sufficient here to address the fields; that is, IDOCTP enables you to copy the initial value, which is usually desired.

Sample code

```
*&--------------------------------------------------------*
*&  Include           ZXSBDU01
*&--------------------------------------------------------*
* Check whether IDoc is received, otherwise do nothing!
check idoc_control_in-direct eq ‘2’.
* If yes, check which IDoc and change
**************************************************
* Local variables
**************************************************
data: wa_kunnr type kna1-kunnr .
DATA: wa_idoc_data type edidd .
DATA: wa_e1edka1 type e1edka1 .
DATA: wa_e1edk14 type e1edk14 .
data: wa_vkorg type tvko-vkorg .
DATA: XBBBNR LIKE KNA1-BBBNR .
DATA: XBBSNR LIKE KNA1-BBSNR .
DATA: XBUBKZ LIKE KNA1-BUBKZ .
Check message type
case idoc_control_in-stdmes .
  when ‘ORDERS’ .
**************************************************
* Fill vendor number and customer number
**************************************************
Fill values
    loop at idoc_data into wa_idoc_data
      where segnam eq ‘E1EDKA1’ .
      wa_E1EDKA1 = wa_idoc_data-SDATA .
      if wa_e1edka1-partn is initial . “ Only if it is missing
        XBBBNR = wa_E1EDKA1-ILNNR+0(7) .
        XBBSNR = wa_E1EDKA1-ILNNR+7(5) .
        XBUBKZ = wa_E1EDKA1-ILNNR+12(1) .
        CHECK: NOT XBBBNR IS INITIAL .
        CHECK: XBBBNR NE SPACE .
        SELECT kunnr FROM KNA1 into wa_kunnr
        WHERE BBBNR EQ XBBBNR
                 AND   BBSNR EQ XBBSNR
                 AND   UBKZ  EQ XBUBKZ .
          EXIT .
        ENDSELECT .
        IF SY-SUBRC = 0 .
          wa_e1edka1-partn = wa_kunnr .
```

```
**************************************************
* Return data to IDoc
**************************************************
Return to IDoc
          wa_idoc_data-sdata = wa_e1edka1 .
          modify idoc_data from wa_idoc_data .
        endif .
      endif .
    endloop .
  when ‘ANYOTHERTYPE’ .
* Whatever you want to do with this message type.
  when others .
endcase .
**************************************************
* Change control record so that the changes
 are implemented after the exit;
 this applies to all changed segments
**************************************************
move idoc_control_in to idoc_control_out .
idoc_control_out-IDOCTP = idoc_control_in-IDOCTP .
idoc_control_out-upddat = sy-datum .
idoc_control_out-updtim = sy-uzeit .
```

Listing 4.1 Sample Code for the “ALE00001” Enhancement

BAdI for the Generation of Change Pointers

As described in Chapter 2, Section 2.1.1, Shared Master Data Tool, change pointers are generated in the CHANGE_POINTERS_CREATE_LONG or CHANGE_POINTERS_CREATE_DIRECT function module. SAP also provides an exit — a classic BAdI — for the generation of change pointers. The exit has only been available as of SAP NetWeaver 6.20.

“BDCP_BEFORE_WRITE” BAdI

The BDCP_BEFORE_WRITE BAdI is a filter-dependent BAdI with the message type as the filter. The prerequisites for using this exit are that change pointers are generally enabled, that they are enabled for the corresponding message type, and that the fields that are supposed to be relevant to the changes are maintained for the message type. In the standard version, these settings enable the SAP system to create all required change pointers. The BAdI is then called before it creates the change pointers on the database and allows for more restrictions on the generation of change pointers than the Customizing settings. The BAdI contains sample code for two cases:

- The system will write change pointers for particular materials only.
- The system will write change pointers for particular users only.

If you want to use this BAdI, you can copy the sample code and adapt it to your requirements. Here, you must take into account some specific aspects.

"OBJECTID" field

First of all, the sample code provided by SAP assumes that the OBJECTID field contains the client at the beginning 3 digits. This isn't the case. So please don't remove the three additional fields for the client in your implementation if you want to filter by object IDs.

"CPIDENT" field

In addition, the CPIDENT field doesn't contain the number that it will have in the BDCP and BDCPS tables at a later stage. Instead, it only contains a temporary number until the system assigns the actual number during the update.

Inserting data

In the exit help, SAP states that you can also insert change pointers; the sample code, in turn, says that you can't use the exit for this purpose. Technically, however, you can insert them, but you must ensure that the temporary CPIDENT field of your internal table contains sequential numbers without duplicate values; otherwise, the update is canceled. You should be careful when inserting change pointers. If you want to debug your exit, update debugging usually needs to be enabled. Most master data posts its changes using the update technology on the database, so writing change pointers also takes place during update processes.

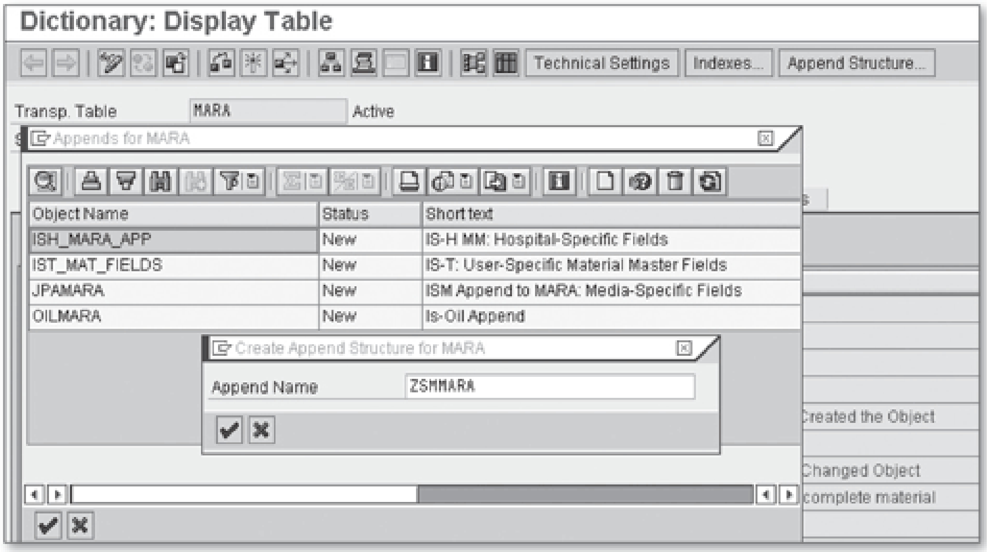
4.2.3 Custom Segments

Both when enhancing existing IDoc types and when creating custom IDoc types, a prerequisite is the generation of segments that contain the application data. Such a segment can include up to 1,000 alphanumeric characters. All information on the generation of custom segments is discussed again in Section 4.4.1, Creating Custom IDocs.

Creating Segments

Customer segments

Before you can use custom segments, you usually have to modify the business data. Normally, you can expect that this has already been developed by your colleagues from the application department. The following example, however, also includes the data in the material master that will be used in the enhancement. Figure 4.71 shows how you can append additional customer-specific fields to tables (here: MARA) using the Append Structure button (on the top right in the figure) in Transaction SE11. For this purpose, assign an append name first, which must be within the customer namespace (here: ZSMMARA).


Dictionary: Display Table

Transp. Table MARA Active

Appends for MARA

Object Name	Status	Short text
ISH_MARA_APP	New	IS-H MM: Hospital-Specific Fields
IST_MAT_FIELDS	New	IS-T: User-Specific Material Master Fields
JPAMARA	New	ISM Append to MARA: Media-Specific Fields
OILMARA	New	Is-Oil Append

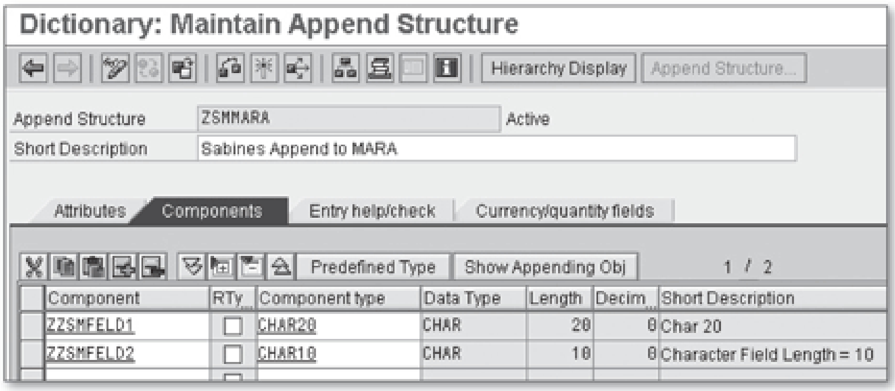
Create Append Structure for MARA

Append Name ZSMMARA

Figure 4.71 Creating an Append to the "MARA" Table

In the next step, you can — as shown in Figure 4.72 — specify in the Components tab which new fields you want to use. The names of these fields should also correspond to the customer naming rules because programs treat them as fields that are directly included in the MARA table and because you can never be sure which fields SAP will add.

Segment fields


Dictionary: Maintain Append Structure

Append Structure ZSMMARA Active
Short Description Sabines Append to MARA

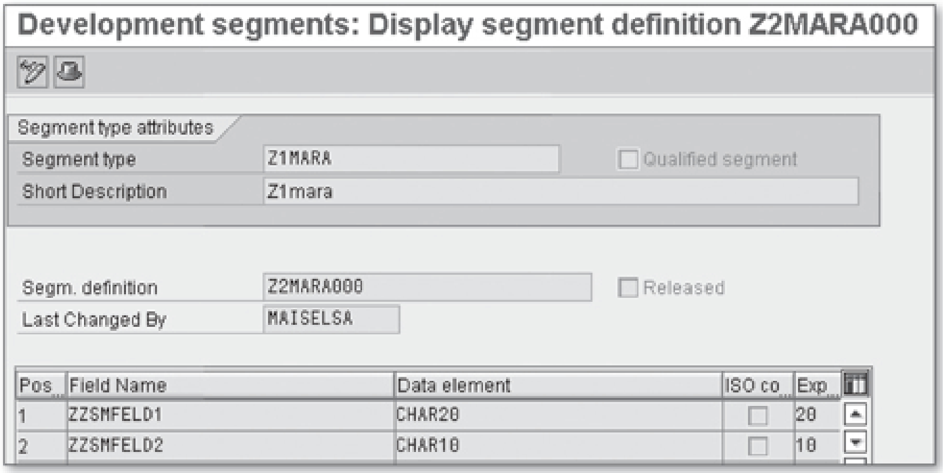
Component	RTy	Component type	Data Type	Length	Decim	Short Description
ZZSMFELD1	□	CHAR20	CHAR	20	0	Char 20
ZZSMFELD2	□	CHAR10	CHAR	10	0	Character Field Length = 10

Figure 4.72 Custom Fields

Custom segment types

Now you carry out the first IDoc-relevant step for enhancing IDoc types, which is creating a custom segment type. You create segment types in Transaction WE31. They can have names with up to 27 characters beginning with "Z" or "Y" or /<NAMESPACE>/ if they are customer segments. This is followed by 1; the remaining part of the name can be freely defined by you (in our example, the name is Z1MARA).

SAP segments often have a name that directly refers to the database table from which the fields are used. But this isn't mandatory. Figure 4.73 displays our sample segment, which belongs to the append discussed earlier for the MARA table.



Development segments: Display segment definition Z2MARA000

Segment type attributes

Segment type	Z1MARA	□ Qualified segment
Short Description	Z1mara	
Segm. definition	Z2MARA000	□ Released
Last Changed By	MAISELSA	

Pos	Field Name	Data element	ISO co	Exp
1	ZZSMFELD1	CHAR20	□	20
2	ZZSMFELD2	CHAR10	□	10

Figure 4.73 Creating a Custom Segment Type

When creating the segment, the system determines the lengths of the fields from the database. If you modify the field lengths at a later stage, you must manually change them in the segment.

Segment type in Transaction SE11

You can view the segment type in Transaction SE11, the *Data Dictionary* (DDIC), where it's defined as a structure. The segment type contains the information in a SAP-specific format in which the database also stores the fields.

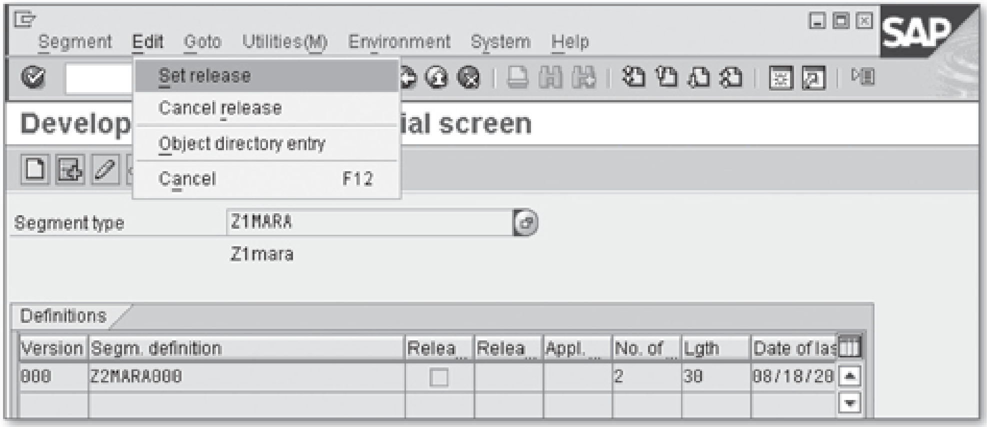
The system also creates a segment definition for every segment type with the same name parts as in the original name except for the 1, which is replaced by a 2 here (e.g., Z2MARA). This segment definition can be available in various versions, which is indicated by a three-digit number at the end of the name. The segment definition of our example is the first version, so it's assigned the "000" addition: `Z2MARA000`. The segment definition indicates the external format, that is, the format that is actually exchanged with the partner and that differs from the internal format of the SAP database for some fields.

Neutral format

To deliver a neutral format, you must provide all information as a text. The SAP database stores numbers without decimal separators; the decimal separator, however, is always transferred as a point. A possible minus sign is transferred in the last field. Compared to the internal presentation, these numbers always contain two additional places. Dates are usually transferred as follows: four digits for the year, two digits for the month, and two digits for the day.

Release

You must release the segment before you can use it. For this purpose, follow the Edit • Set release menu path (see Figure 4.74).



Segment type: Z1MARA
Z1mara

Version	Segm. definition	Relea...	Relea...	Appl....	No. of...	Lgth	Date of las
000	Z2MARA000	□			2	30	08/18/20

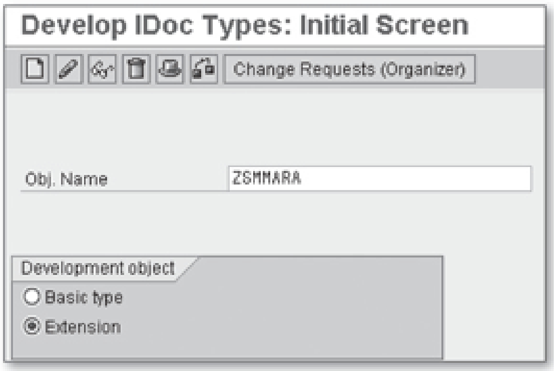
Figure 4.74 Releasing a Segment

After the release, every change results in a new version of the segment. In the Version Conversion ALE service (see Section 4.2.2, General Exits), you can ensure that the system uses the appropriate older version of the segment definition if your partner uses an older release.

Creating an Enhancement Type

Creating an enhancement type

After you create all required segments, you can create the actual enhancement type. It also follows the common customer naming rules and can contain up to 30 characters. Transaction WE30 is the corresponding transaction. It's critical that you select the Extension checkbox (Figure 4.75) in the initial screen because the default setting is Basic Type, which is only required for completely new IDoc types.


Develop IDoc Types: Initial Screen

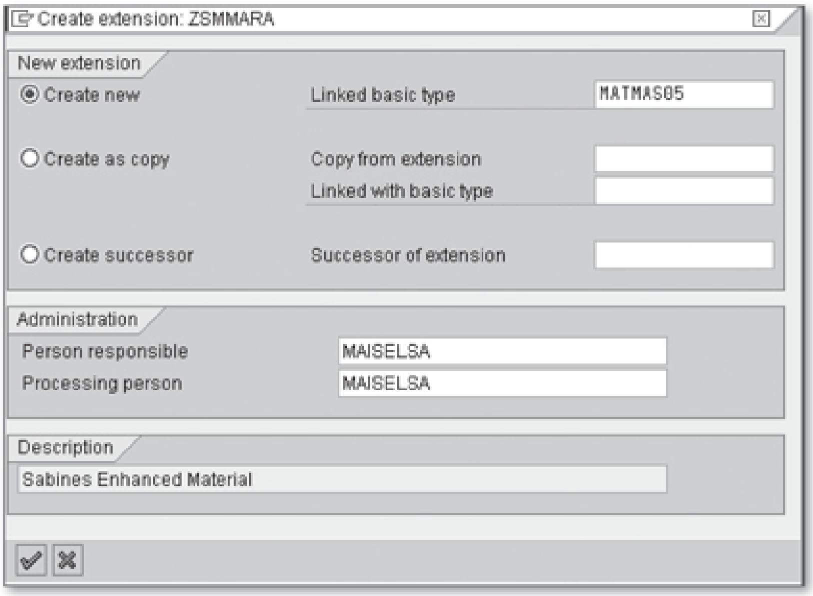
Obj. Name: ZSMMARA

Development object
○ Basic type
● Extension

Figure 4.75 Creating the Enhancement Type

Properties of the enhancement type

In the next step (see Figure 4.76), define which basic type you want your enhancement type to refer to; in our example, it's the `MATMAS05` basic type. You can also copy and further process existing enhancements or create an enhancement as the follow-up object of an existing enhancement.


Create extension: ZSMMARA

New extension
● Create new — Linked basic type: MATMAS05
○ Create as copy — Copy from extension: — Linked with basic type:
○ Create successor — Successor of extension:

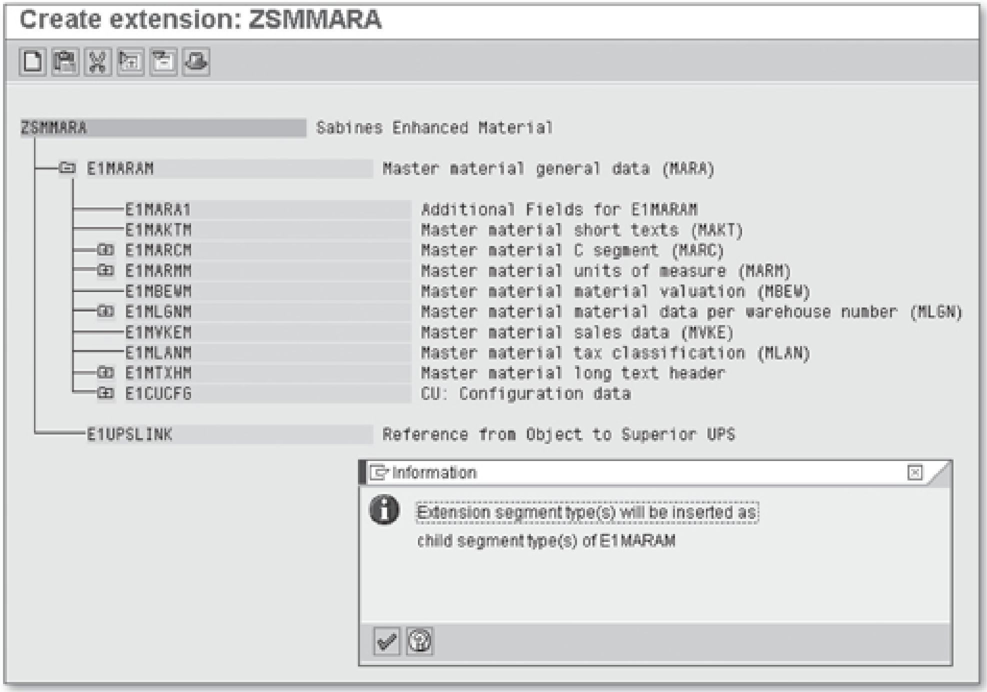
Administration
Person responsible: MAISELSA
Processing person: MAISELSA

Description
Sabines Enhanced Material

Figure 4.76 Necessary Specifications

Enhancement of the "MATMAS" basic type

The next screen displays the basic type that is defined as a template (see Figure 4.77). You can choose the segment for which you want to create your child segments by selecting the segment and clicking on the Create icon. The system then outputs a message that indicates that custom segments are only permitted as child segment types for enhancements.



Create extension: ZSMMARA

```
ZSMMARA                 Sabines Enhanced Material

  E1MARAM               Master material general data (MARA)

      E1MARA1           Additional Fields for E1MARAM
      E1MAKTM           Master material short texts (MAKT)
      E1MARCM           Master material C segment (MARC)
      E1MARMM           Master material units of measure (MARM)
      E1MBEWM           Master material material valuation (MBEW)
      E1MLGNM           Master material material data per warehouse number (MLGN)
      E1MVKEM           Master material sales data (MVKE)
      E1MLANM           Master material tax classification (MLAN)
      E1MTXHM           Master material long text header
      E1CUCFG           CU: Configuration data

  E1UPSLINK             Reference from Object to Superior UPS
```

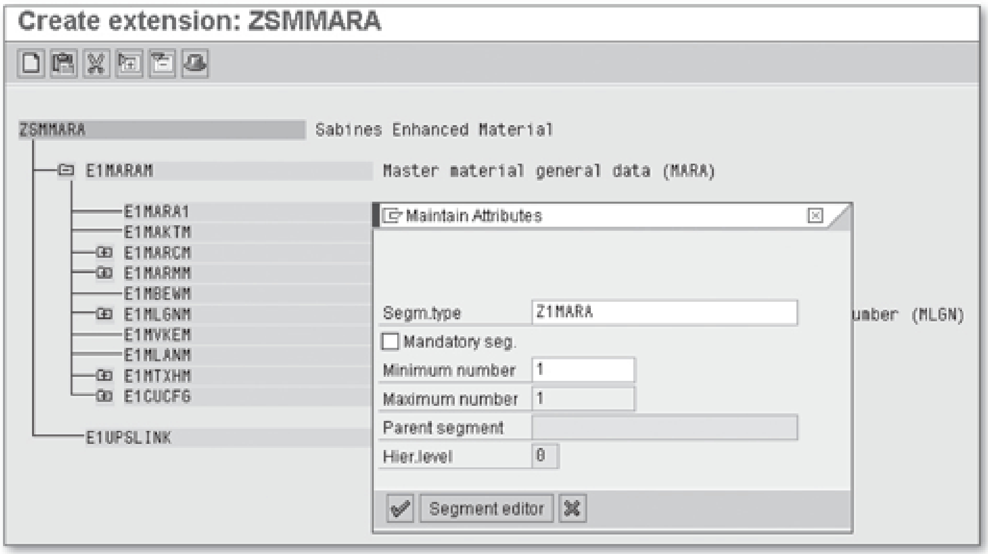
Information

Extension segment type(s) will be inserted as child segment type(s) of E1MARAM

Figure 4.77 Inserting a Custom Segment

Occurrence of the segments

Next, you must specify which segment type you want to use (see Figure 4.78). Furthermore, you need to define the occurrence of the segment within the IDoc; the minimum number must always be "1." "0" would be considered "blank" here, which justifies this uncommon entry. The segment type only becomes a Mandatory segment if you select the corresponding flag. In our example, the segment is optional and can occur only once. (The append entry in the ZSMMARA table can be provided only once for each material; accordingly, the Z1MARA segment can occur only once for each material as well.)



Create extension: ZSMMARA

```
ZSMMARA                      Sabines Enhanced Material
  E1MARAM                    Master material general data (MARA)
    E1MARA1
    E1MAKTM
    E1MARCM
    E1MARMM
    E1MBEWM
    E1MLGNM                                                   umber (MLGN)
    E1MVKEM
    E1MLANM
    E1MTXHM
    E1CUCFG
  E1UPSLINK
```

Maintain Attributes

Segm.type: Z1MARA

☐ Mandatory seg.

Minimum number: 1

Maximum number: 1

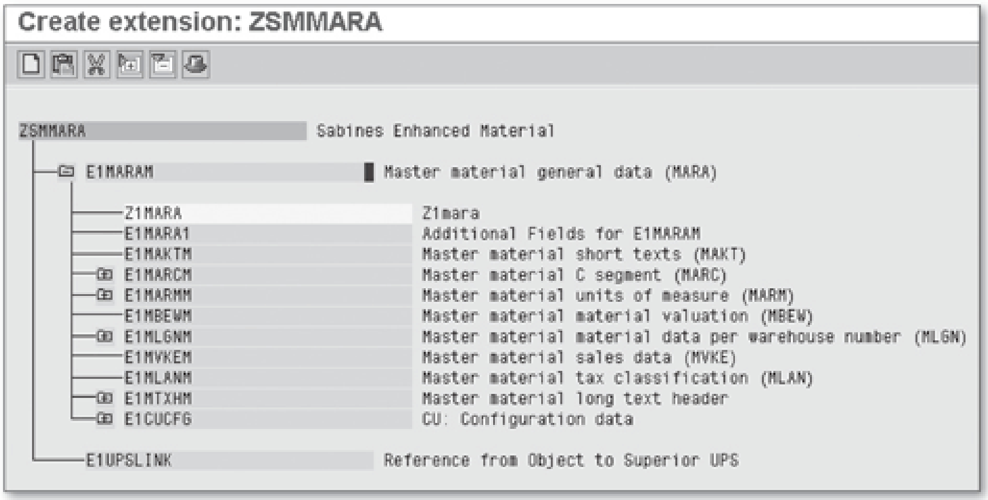
Parent segment:

Hier.level: 0

Figure 4.78 Specifications for a Custom Segment

Enhancement for "Z1MARA"

The system indicates the custom enhancement segments in white and the initial segments in blue. This enables you to directly identify which segments have been added. Figure 4.79 displays the enhancement result, that is, the added (white) segment, Z1MARA.



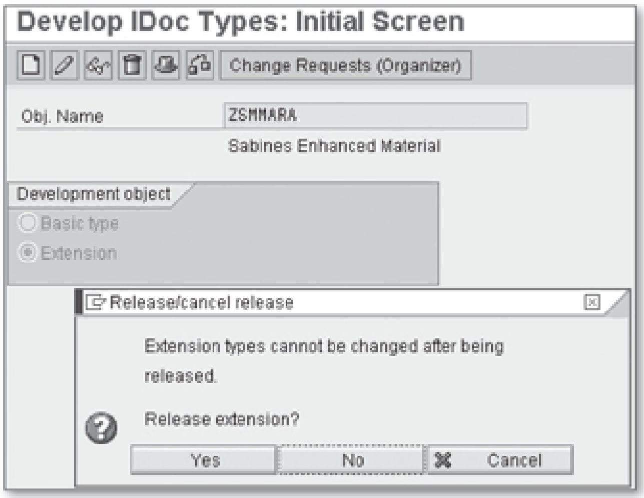
Create extension: ZSMMARA

```
ZSMMARA                      Sabines Enhanced Material
  E1MARAM                    Master material general data (MARA)
    Z1MARA                   Z1mara
    E1MARA1                  Additional Fields for E1MARAM
    E1MAKTM                  Master material short texts (MAKT)
    E1MARCM                  Master material C segment (MARC)
    E1MARMM                  Master material units of measure (MARM)
    E1MBEWM                  Master material material valuation (MBEW)
    E1MLGNM                  Master material material data per warehouse number (MLGN)
    E1MVKEM                  Master material sales data (MVKE)
    E1MLANM                  Master material tax classification (MLAN)
    E1MTXHM                  Master material long text header
    E1CUCFG                  CU: Configuration data
  E1UPSLINK                  Reference from Object to Superior UPS
```

Figure 4.79 Complete Enhancement

Releasing an enhancement

You must also release the enhancement. For this purpose, follow the EDIT • SET RELEASE menu path (see Figure 4.80). You can no longer modify your enhancement after the release.



Develop IDoc Types: Initial Screen

Obj. Name: ZSMMARA

Sabines Enhanced Material

Development object

○ Basic type

● Extension

Release/cancel release

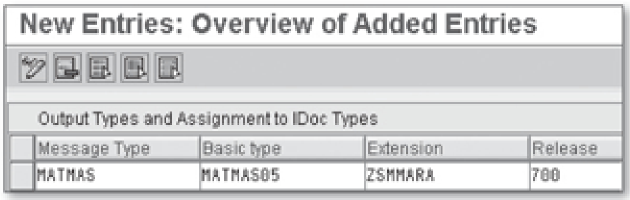
Extension types cannot be changed after being released.

Release extension?

Figure 4.80 Releasing the Enhancement

Assignment to message type and IDoc type

Now, you have to assign the released enhancement to the message type and the IDoc type because you can only use it with this combination. Figure 4.81 shows how this is done for the sample enhancement, `ZSMMARA`, in Transaction WE82.



New Entries: Overview of Added Entries

Output Types and Assignment to IDoc Types

Message Type	Basic type	Extension	Release
MATMAS	MATMAS05	ZSMMARA	700

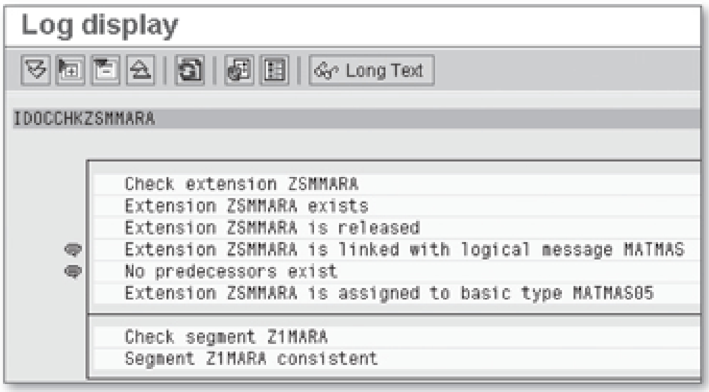
Figure 4.81 Assigning the Enhancement to the Message Type

Checking the enhancement

If you're not sure whether you've considered all necessary aspects, you can check your enhancement. Such check functions are also provided for segment types, but it makes more sense to carry out the check after everything has been implemented so that the child elements are also checked. To check the enhancement, use the Enhancement • Check menu or the scales icon, which you can see in Figure 4.80, shown earlier.

Figure 4.82 shows what is checked and how this is displayed in the case of success. If an error occurred, you can correct it using this log.

After having completed the enhancement, you need to fill the segments and the name of the enhancement in the control record if you send the IDoc and read the respective segments if you update the IDoc.



```
IDOCCHKZSMMARA

Check extension ZSMMARA
Extension ZSMMARA exists
Extension ZSMMARA is released
Extension ZSMMARA is linked with logical message MATMAS
No predecessors exist
Extension ZSMMARA is assigned to basic type MATMAS05

Check segment Z1MARA
Segment Z1MARA consistent
```

Figure 4.82 Checking the Enhancement

Filling Segments

You now select an appropriate exit in the outbound function module of the IDoc and implement your changes (Listing 4.2). The function module from the enhancements serves as an example here. The procedure is identical for all enhancement technologies, but the names of the transfer variables in the sample code correspond to those of the `EXIT_SAPLMV01_002` exit.

Filling the segments with an appropriate exit

```
*-----------------------------------------------------*
*   INCLUDE ZXMGVU03
*-----------------------------------------------------*
* Fill enterprise-internal segments for MATMAS,
case message_type .
```

Checking for message or IDoc types

```
* Change if this is really the required
* message.
  when ‘MATMAS’ .
* only then implement the inserts.
    idoc_cimtype = ‘ZSMMARA’ .
    case segment_name .
* Check which segment it is. If it
* is one of those to be enhanced, a corresponding
* customer segment is generated and appended.
```

Generating and appending the segment

```
      when ‘E1MARAM’ .
        perform fill_z1mara
        tables idoc_data .
      when others .
        exit .
```

```
    endcase .
  when others .
    exit .
endcase .
```

Listing 4.2 Enhancing the "EXIT_SAPLMV01_002"

Example of the called form

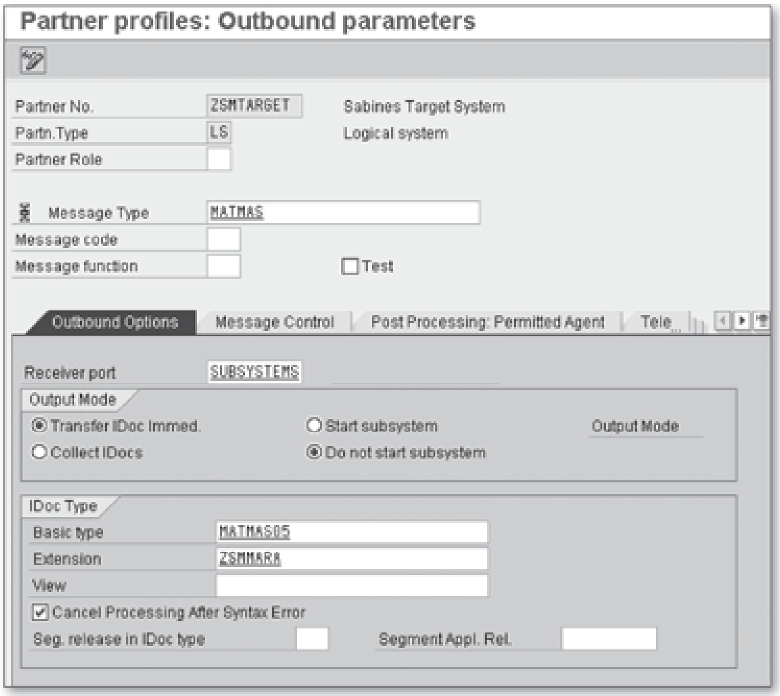
In Listing 4.3, you can see the `fill_z1mara` form, which is called in the exit mentioned previously. Of course a division into subroutines in the code isn't necessary but makes it easier to read programs.

```
*-----------------------------------------------------------*
***INCLUDE ZXMGVF01 .
*----------------------------------------------------------*
*&      Form  fill_z1mara
*&----------------------------------------------------------*
FORM fill_z1mara  tables idoc_data structure edidd .
  tables: e1maram, z1mara .
  data: matnr type mara-matnr .
  read table idoc_data with key segnam = 'E1MARAM' .
* read possible, because the segment occurs only once.
  e1maram = idoc_data-sdata .
  matnr = e1maram-matnr .
  select single ZZSMFELD1 ZZSMFELD2 from Mara
         into corresponding fields of z1mara
         where matnr eq e1maram-matnr .
  if sy-subrc eq 0 .
    idoc_data-segnam = 'Z1MARA' .
    idoc_data-sdata = z1mara .
    append idoc_data .
  endif .
ENDFORM .                    " fill_z1mara
```

Listing 4.3 Code for the "fill_z1mara" Form

Partner profile with enhancement

You have to let the system know if you work with a customer-specific enhancement; otherwise, the system identifies custom segments as incorrect. For this purpose, navigate to the Outbound Options tab and the IDoc Type subtab in the partner profile of the sending system (see Figure 4.83). Here, the IDoc is designed with the corresponding control record, and the sending system checks for all segments whether they are allowed in this combination of IDoc type and enhancement.



Partner profiles: Outbound parameters

Partner No.	ZSMTARGET	Sabines Target System
Partn.Type	LS	Logical system
Partner Role		
Message Type	MATMAS	
Message code		
Message function		□ Test

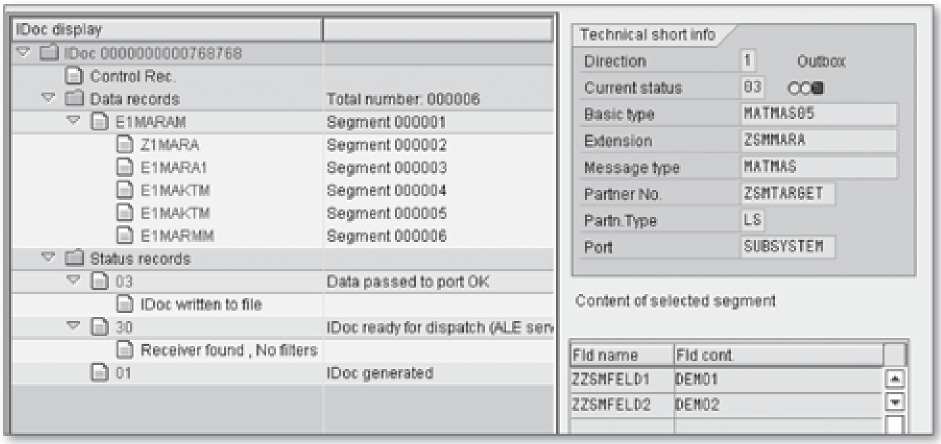
Receiver port: SUBSYSTEMS

Output Mode
- ● Transfer IDoc Immed.
- ○ Collect IDocs
- ○ Start subsystem
- ● Do not start subsystem
- Output Mode

IDoc Type
- Basic type: MATMAS05
- Extension: ZSMMARA
- View:
- ☑ Cancel Processing After Syntax Error
- Seg. release in IDoc type:
- Segment Appl. Rel.:

Figure 4.83 Partner Profile with Enhancement

The IDoc additionally contains the name of the enhancement in the control record; in our example, the enhancement is ZSMMARA in the Extension field. Finally, you can generate the enhanced IDoc that will be sent (see Figure 4.84).



IDoc display	
IDoc 0000000000768768	
Control Rec.	
Data records	Total number: 000006
E1MARAM	Segment 000001
Z1MARA	Segment 000002
E1MARA1	Segment 000003
E1MAKTM	Segment 000004
E1MAKTM	Segment 000005
E1MARMM	Segment 000006
Status records	
03	Data passed to port OK
IDoc written to file	
30	IDoc ready for dispatch (ALE serv
Receiver found , No filters	
01	IDoc generated

Technical short info

Direction	1	Outbox
Current status	03	○○●
Basic type	MATMAS05	
Extension	ZSMMARA	
Message type	MATMAS	
Partner No.	ZSMTARGET	
Partn.Type	LS	
Port	SUBSYSTEM	

Content of selected segment

Fld name	Fld cont.
ZZSMFELD1	DEMO1
ZZSMFELD2	DEMO2

Figure 4.84 Enhanced IDoc to Be Sent

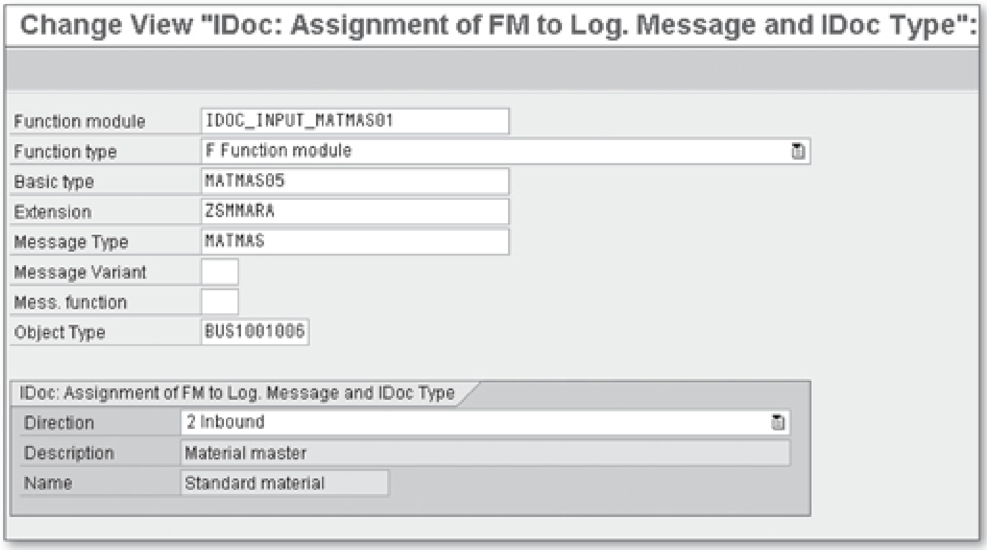
You've now completed all necessary tasks on the sender side. The next section deals with updating an enhanced IDoc.

Control record for the enhancement

Posting Segments

Assigning an inbound function module

To use an enhancement in inbound processing, you must first assign the corresponding inbound function module that consists of message type, IDoc type, and enhancement. You do this in Transaction WE57 (see Figure 4.85). Don't get confused by the name of the IDOC_INPUT_MATMAS01 function module; it can process all material master IDocs (the name was assigned for historical reason because it was developed when only the MATMAS01 IDoc type was available).



Change View "IDoc: Assignment of FM to Log. Message and IDoc Type":

Function module	IDOC_INPUT_MATMAS01
Function type	F Function module
Basic type	MATMAS05
Extension	ZSMMARA
Message Type	MATMAS
Message Variant	
Mess. function	
Object Type	BUS1001006

IDoc: Assignment of FM to Log. Message and IDoc Type

Direction	2 Inbound
Description	Material master
Name	Standard material

Figure 4.85 Assigning the Inbound Function Module for the Enhancement

The programming work consists of two parts. First, you must ensure in the application that the corresponding update modules also transfer the customer-specific fields to the database. Usually, you're not responsible for this part of the development work, and it's too module-specific to be explained in detail here. The second part of the programming work is to ensure that the data that has been additionally transferred in the IDoc is read and transferred to the transfer parameters of the update module, which must be done by you.

Using data from the IDoc

Again, the material master is used as an example here. Listing 4.4 shows the sample code for the fields from the customer-specific segment, Z1MARA.

```
data: wa_z1mara type z1mara .
if message_type eq 'MATMAS' .
  if f_cust_segment-segnam eq 'Z1MARA' .
    wa_z1mara = f_cust_segment-sdata .
    move-corresponding wa_z1mara to f_mara_ueb .
  endif .
endif .
```

Listing 4.4 Transferring the Data from the "Z1MARA" Customer Segment

The transfer structure is delivered by SAP so that it already includes potential appends to the `MARA` table automatically. This is usually the case. If not, you can also add an append to the transfer structure that contains the same fields as your append to the data table.

When do ALE services run?

Combination of ALE Services and Exits

SAP generally provides the option to implement changes to IDocs via the settings in Customizing. The manipulations, however, are implemented after your IDoc has been generated. Your exits, in contrast, are executed while the IDoc is generated. This means that possible filtering processes or rules may not be deployed when your exit is executed. You should consider this for the implementation. For example, if you only want to process a plant segment for a specific partner, you must assume that your exit works while all plant segments are still included. Consequently, you must make sure that your changes are made to the correct plant segment.

4.2.4 Special Requirements for Master Data

You already know that the SMD provides special functions for master data. Sometimes, however, the available special functions aren't wanted. The following sections describe how you can modify them.

Sending All Data After Changes

When using Shared Master Data Tools and the corresponding change pointers, SAP assumes that IDocs should be kept as small as possible for performance reasons. This also usually corresponds to reality. But sometimes you collaborate with partners who can only process complete data records. To still be able to use change pointers, you can implement minor modifications for material masters.

"Send complete" modification

The setting that only the complete material is supposed to be sent is already available for the material master, for example, if the cross-plant status of a value "with distribution lock" has been changed to a value "without distribution lock." For this purpose, the MASTERIDOC_CREATE_SMD_MATMAS function module manages an internal table, a_t_complex_matnr, with the MANDT, MATNR, and MSGFN fields. If you now append your materials that are to be sent completely as well and set the MSGFN field to "005" for "message replaces previous messages," the system sends the complete material. The corresponding sample code could be structured as shown in Listing 4.5.

Getting the change pointer

Appending to SAP table

```
* This change has the effect that specific messages
* are always sent completely.
* The self-developed reduced message type
* is the key for the decision here.
data: MESTYPE type BDCPS-MESTYPE .
LOOP AT A_T_CHGPTRS .
select single MESTYPE from BDCPS
       into MESTYPE where CPIDENT eq A_T_CHGPTRS-CPIDENT .
  if MESTYPE eq ‘ZSMMAT’ .
    T_MARAKEY-MANDT = SY-MANDT .
    T_MARAKEY-MATNR = A_T_CHGPTRS-CDOBJID .
    COLLECT T_MARAKEY .
    A_T_COMPLEX_MATNR-MANDT = SY-MANDT .
    A_T_COMPLEX_MATNR-MATNR = A_T_CHGPTRS-CDOBJID .
    A_T_COMPLEX_MATNR-MSGFN = C_MSGFN_R .
    COLLECT A_T_COMPLEX_MATNR .
    clear MESTYPE .
   ENDIF .
ENDLOOP .
```

Listing 4.5 Sending the Complete Material After Modifications

This code must be added before the system further processes the data. Because the module may change, you should search for the following comment:

```
DESCRIBE TABLE t_marakey LINES hlines .
  CHECK hlines GT 0 .
```

Location of the modification

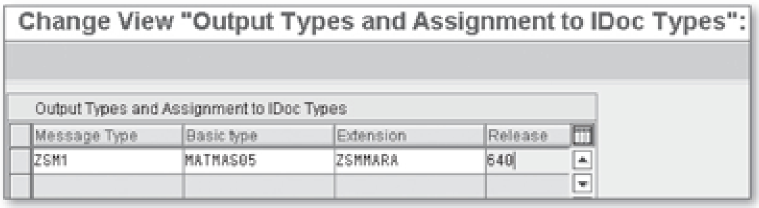
Then, insert the code directly before this in your function module. Next, instead of the function module provided by SAP, assign your own function module in Transaction BD60. Be sure to identify the changes in the original module for each release change and reproduce them in your function module if required. Alternatively, you can copy the SAP module after the release change and integrate your modifications. Unfortunately, an appropriate exit isn't yet available at the location where the modification needs to be implemented.

Reduced IDocs in Combination with Custom Segments

If you've added custom segments to the master data IDocs, you may want to work with reductions — for reasons of performance or because your partner doesn't want to receive particular data. Because your enhancement is an IDoc type but your reduction a message type, you can't simply use one of them as the basis for the other one. Instead, you have to generate a reduced IDoc that doesn't know your customer-specific segments and in which you can delete everything from the standard version that you don't want to send. In addition, you must also generate an enhancement type that contains your additional segments without knowing the reduction.

Enhancing and reducing

When creating the reduced message type, Transaction WE82 automatically generates an entry in which the basic type is assigned to the newly created reduced message type. This is done automatically when you save the reduced message type so that you can use it immediately. To now use the combination from your reduction and your enhancement, copy the basic type, and insert the enhancement in the copied entry. Figure 4.86 shows this for the ZSM1 reduction (see Chapter 2, Section 2.1.1, Shared Master Data Tool) and the ZSMMARA enhancement from this section.



Change View "Output Types and Assignment to IDoc Types":

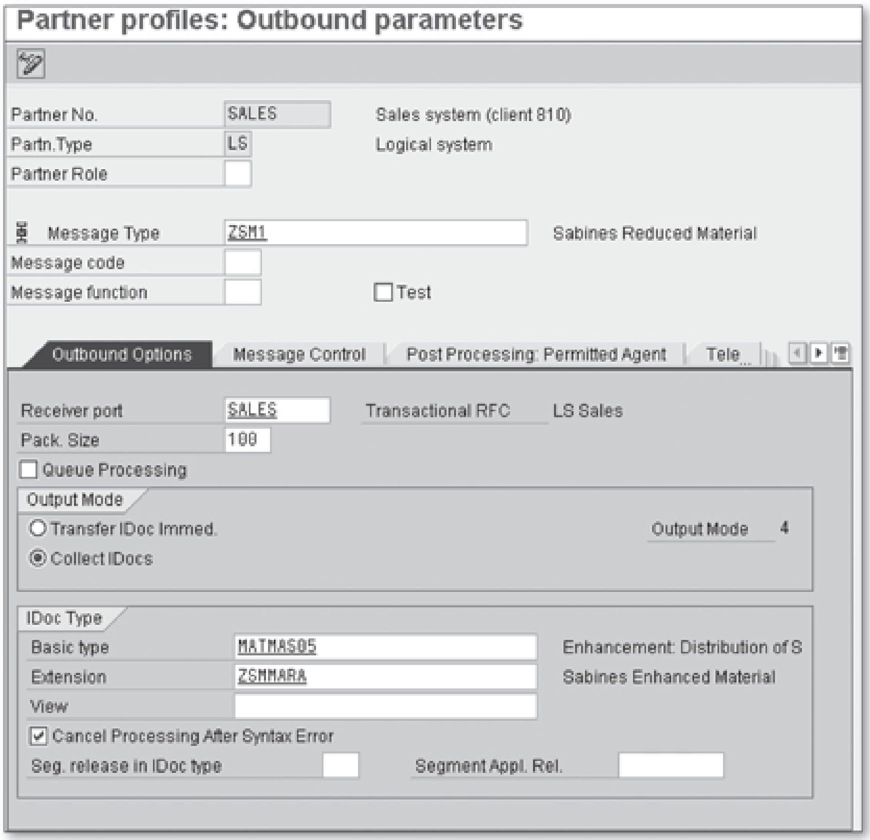
Output Types and Assignment to IDoc Types

Message Type	Basic type	Extension	Release
ZSM1	MATMAS05	ZSMMARA	640

Figure 4.86 Assigning the Enhancement to the Reduced Message Type

Appropriate partner profile

In the partner profile, the enhancement is combined with the basic type, and the standard module carries out the reduction while your exit fills the customer-specific fields. Figure 4.87 displays the appropriate partner profile in the outbound processing for the reduced message type, ZSM1, with "MATMAS05" specified in the Basic Type field and "ZSMMARA" in the Extension field. The inbound processing must take what it gets; however, enhancements and reductions must also be known there.



Partner profiles: Outbound parameters

Partner No.	SALES	Sales system (client 810)
Partn.Type	LS	Logical system
Partner Role		
Message Type	ZSM1	Sabines Reduced Material
Message code		
Message function		□ Test

Receiver port	SALES	Transactional RFC	LS Sales
Pack. Size	100		

□ Queue Processing

Output Mode

○ Transfer IDoc Immed.

● Collect IDocs

Output Mode 4

IDoc Type

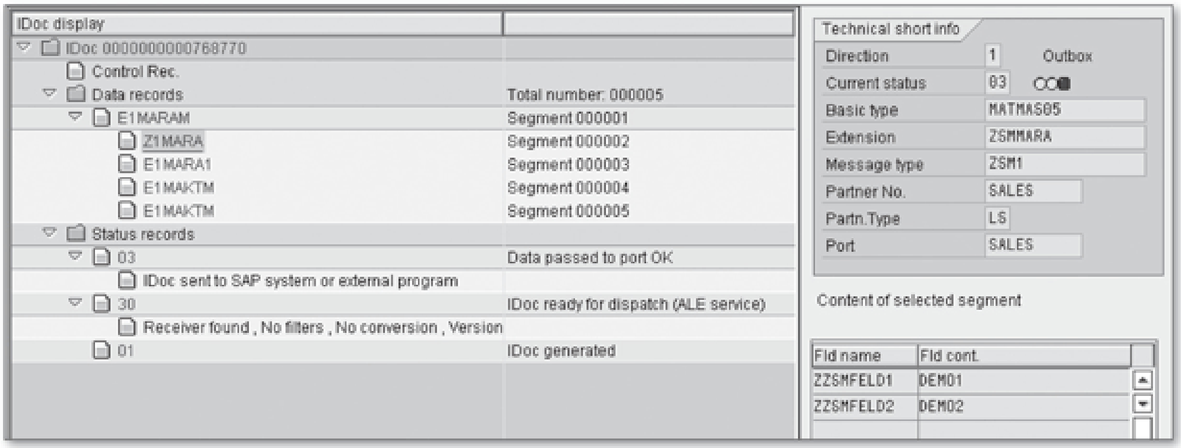
Basic type	MATMAS05	Enhancement: Distribution of S
Extension	ZSMMARA	Sabines Enhanced Material
View		

☑ Cancel Processing After Syntax Error

Seg. release in IDoc type | Segment Appl. Rel.

Figure 4.87 Partner Profile with Reduction and Enhancement

To test these settings, generate your IDoc (see Figure 4.88).



IDoc display	
IDoc 0000000000768770	
Control Rec.	
Data records	Total number: 000005
E1MARAM	Segment 000001
Z1MARA	Segment 000002
E1MARA1	Segment 000003
E1MAKTM	Segment 000004
E1MAKTM	Segment 000005
Status records	
03	Data passed to port OK
IDoc sent to SAP system or external program	
30	IDoc ready for dispatch (ALE service)
Receiver found , No filters , No conversion , Version	
01	IDoc generated

Technical short info

Direction	1	Outbox
Current status	03	○○●
Basic type	MATMAS05	
Extension	ZSMMARA	
Message type	ZSM1	
Partner No.	SALES	
Partn.Type	LS	
Port	SALES	

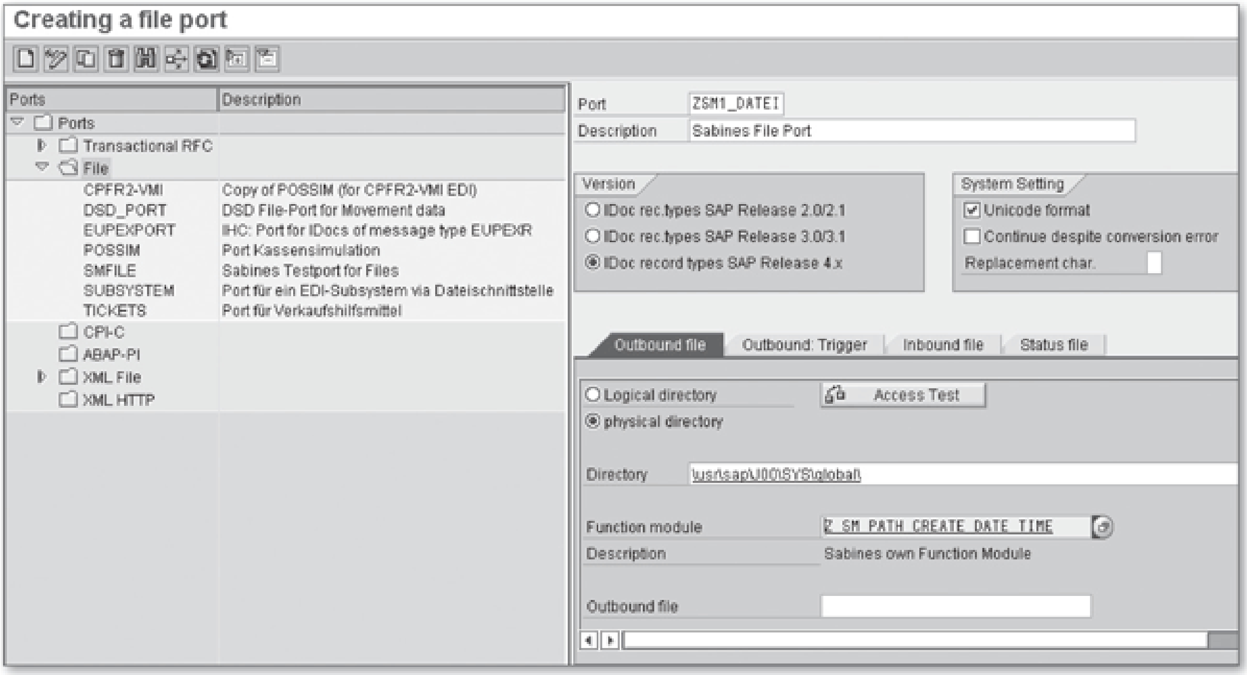
Content of selected segment

Fld name	Fld cont.
ZZSMFELD1	DEM01
ZZSMFELD2	DEM02

Figure 4.88 Sample IDoc

4.3 Custom Function Modules for File Generation in File Ports

For file ports, you need to specify to which file the corresponding IDoc is supposed to be written if this port is used. You do this in Transaction WE21, the *port management* (see Figure 4.89). Here, you can always write the IDoc to a definite file by specifying the name of the corresponding file in the Outbound File field in the tab with the same name.



Creating a file port

Ports	Description
Ports	
Transactional RFC	
File	
CPFR2-VMI	Copy of POSSIM (for CPFR2-VMI EDI)
DSD_PORT	DSD File-Port for Movement data
EUPEXPORT	IHC: Port for IDocs of message type EUPEXR
POSSIM	Port Kassensimulation
SMFILE	Sabines Testport for Files
SUBSYSTEM	Port für ein EDI-Subsystem via Dateischnittstelle
TICKETS	Port für Verkaufshilfsmittel
CPI-C	
ABAP-PI	
XML File	
XML HTTP	

Port: ZSM1_DATEI

Description: Sabines File Port

Version

○ IDoc rec.types SAP Release 2.0/2.1

○ IDoc rec.types SAP Release 3.0/3.1

● IDoc record types SAP Release 4.x

System Setting

☑ Unicode format

□ Continue despite conversion error

Replacement char.

○ Logical directory

● physical directory

Directory: \usr\sap\U00\SYS\global\

Function module: Z_SM_PATH_CREATE_DATE_TIME

Description: Sabines own Function Module

Outbound file:

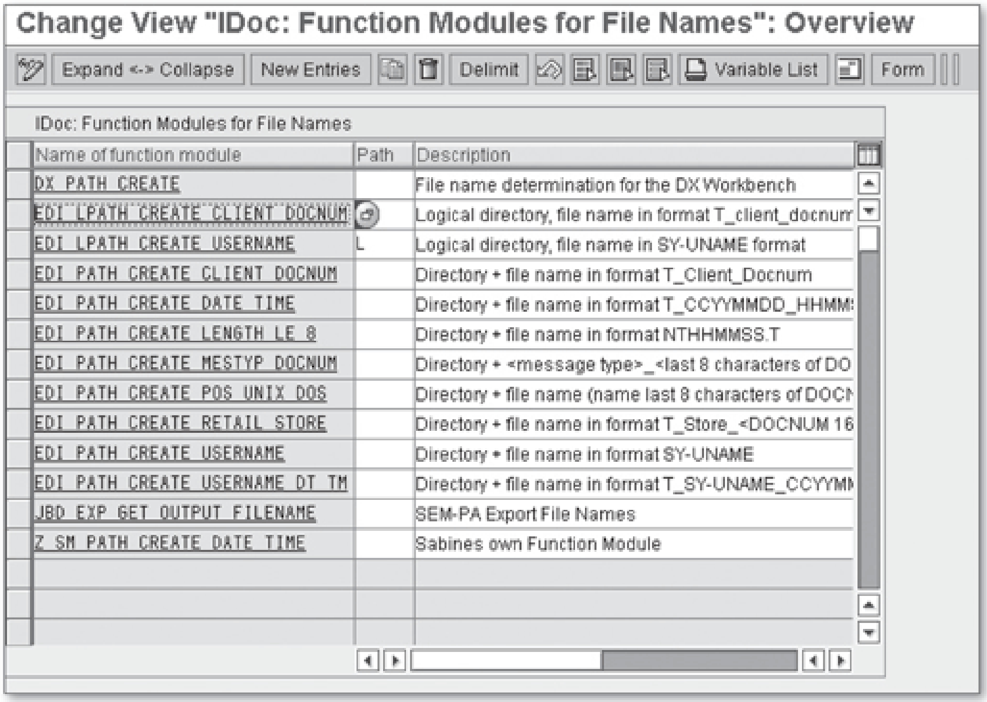
Figure 4.89 File Port with Custom Function Module

Module for file names in file ports

More frequently, however, file names are supposed to be assigned dynamically. In these cases, use a function module that comprises the name of the file from the IDoc data, for example, from the IDoc number, the current time, or similar information. This way, you avoid that the system overwrites an existing file, and you can also find the data more easily. Figure 4.89 shows the use of an additional function module that is created within the customer namespace (`Z_SM_PATH_CREATE_DATE_TIME`).

Maintenance transaction for file function modules

It isn't sufficient to simply implement the required function module. To have the system display it in the selection list in the file port, you must additionally register it for the file ports and let the system know whether it uses logical or physical paths. In Transaction WE55, you implement this assignment by adding a new entry (see Figure 4.90). Function modules marked with "L" use logical paths; all other modules use physical paths.


Change View "IDoc: Function Modules for File Names": Overview

IDoc: Function Modules for File Names

Name of function module	Path	Description
DX_PATH_CREATE		File name determination for the DX Workbench
EDI_LPATH_CREATE_CLIENT_DOCNUM		Logical directory, file name in format T_client_docnum
EDI_LPATH_CREATE_USERNAME	L	Logical directory, file name in SY-UNAME format
EDI_PATH_CREATE_CLIENT_DOCNUM		Directory + file name in format T_Client_Docnum
EDI_PATH_CREATE_DATE_TIME		Directory + file name in format T_CCYYMMDD_HHMM
EDI_PATH_CREATE_LENGTH_LE_8		Directory + file name in format NTHHMMSS.T
EDI_PATH_CREATE_MESTYP_DOCNUM		Directory + <message type>_<last 8 characters of DO
EDI_PATH_CREATE_POS_UNIX_DOS		Directory + file name (name last 8 characters of DOC
EDI_PATH_CREATE_RETAIL_STORE		Directory + file name in format T_Store_<DOCNUM 16
EDI_PATH_CREATE_USERNAME		Directory + file name in format SY-UNAME
EDI_PATH_CREATE_USERNAME_DT_TM		Directory + file name in format T_SY-UNAME_CCYYM
JBD_EXP_GET_OUTPUT_FILENAME		SEM-PA Export File Names
Z_SM_PATH_CREATE_DATE_TIME		Sabines own Function Module

Figure 4.90 Function Modules for File Names

The signature of a function module that works with logical paths looks like the one shown in Listing 4.6.

Signature for logical paths

```
*"----------------------------------------------------------
*"*"Local interface:
*"        IMPORTING
*"              VALUE(DATATYPE)  LIKE  EDIPO-ACTRIG
*"              VALUE(DIRECTORY) LIKE  EDIPO-OUTPUTDIR
*"              VALUE(FILENAME)  LIKE  EDIPO-OUTPUTFILE
*"              VALUE(CONTROL)   LIKE  EDIDC STRUCTURE  EDIDC
*"        EXPORTING
*"              VALUE(PATHNAME)  LIKE  EDI_PATH-PTHNAM
*"        EXCEPTIONS
*"              LOGICAL_PATH_ERROR
*"---------------------------------------------------------
```

Listing 4.6 Signature of a Function Module Using Logical Paths

Transaction FILE

Furthermore, you must also determine the physical path, which has been assigned in Customizing in Transaction FILE. For this purpose, SAP provides a specific function module. The following sample code (Listing 4.7) indicates its use.

```
DATA: LOGICAL_PATH LIKE  FILEPATH-PATHINTERN.
  LOGICAL_PATH = DIRECTORY.
CALL FUNCTION ‘FILE_GET_NAME_USING_PATH’
     EXPORTING
*         CLIENT                     = SY-MANDT
          LOGICAL_PATH               = LOGICAL_PATH
*         OPERATING_SYSTEM           = SY-OPSYS
*         PARAMETER_1                = ‘ ‘
*         PARAMETER_2                = ‘ ‘
*         USE_BUFFER                 = ‘ ‘
          FILE_NAME                  = SY-UNAME
*         USE_PRESENTATION_SERVER    = ‘ ‘
     IMPORTING
          FILE_NAME_WITH_PATH        = PATHNAME
     EXCEPTIONS
          PATH_NOT_FOUND             = 1
          MISSING_PARAMETER          = 2
          OPERATING_SYSTEM_NOT_FOUND = 3
          FILE_SYSTEM_NOT_FOUND      = 4
          OTHERS                     = 5 .
```

Listing 4.7 SAP Function Module That Determines the Physical File Name for a Logical File Name

Signature for physical paths

If you generate a function module that works with physical paths right from the beginning, you must use the following signature (see Listing 4.8).

```
*"----------------------------------------------------------
*"*"Local interface:
*"       IMPORTING
*"             VALUE(DATATYPE)  LIKE  EDIPO-ACTRIG
*"             VALUE(DIRECTORY) LIKE  EDIPO-OUTPUTDIR
*"             VALUE(FILENAME)  LIKE  EDIPO-OUTPUTFILE
*"             VALUE(CONTROL)   LIKE  EDIDC STRUCTURE  EDIDC
*"       EXPORTING
*"             VALUE(PATHNAME)  LIKE  EDI_PATH-PTHNAM
*"----------------------------------------------------------
```

Listing 4.8 Signature of a Function Module Using Physical File Paths

4.4 Custom IDocs

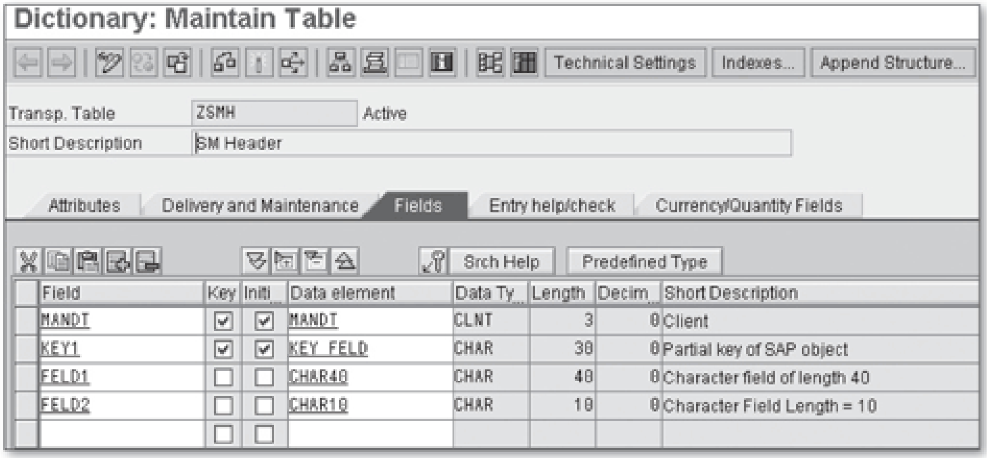
The prerequisite for using custom IDocs are custom tables from which data is retrieved for sending the IDocs and stored in inbound processing. As an example, the descriptions in the following sections use a simple case with only a few fields, which nevertheless covers all aspects that are relevant to IDocs.

Data for the custom IDoc

This example uses a *header table* and an *item table*. The item table contains several identical fields that the IDoc processes using a qualifying segment. Due to the difficult reusability, SAP doesn't recommend the use of qualifiers. In the standard version, however, quite a lot of qualifiers still exist so that you may not be able to avoid using them.

Header table

Figure 4.91 displays the structure of a header table. An interface project usually already includes these tables so that you can use them in your IDoc. To illustrate the relationship between IDoc and data record, the application tables were created as well and not considered as existing.



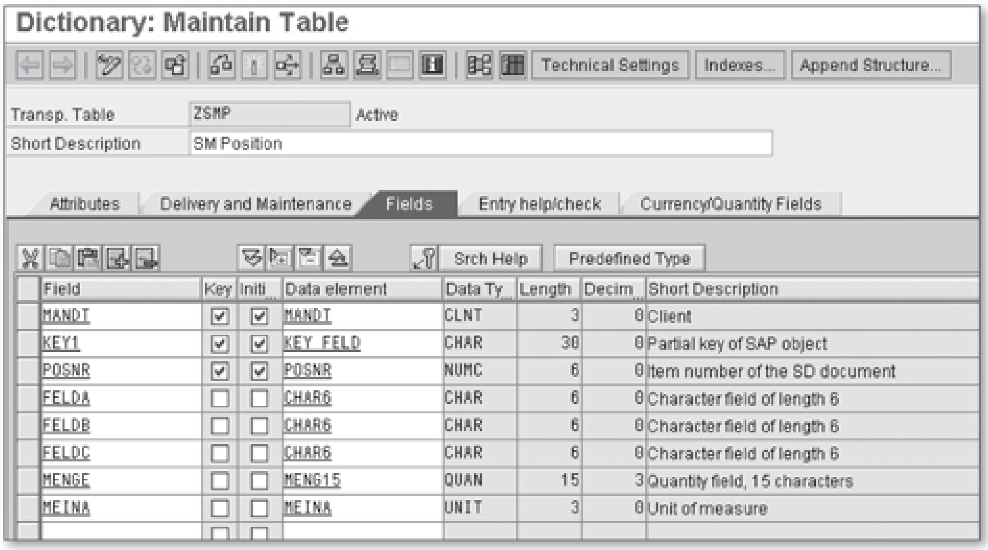
Transp. Table: ZSMH Active
Short Description: SM Header

Field	Key	Initi...	Data element	Data Ty...	Length	Decim...	Short Description
MANDT	☑	☑	MANDT	CLNT	3	0	Client
KEY1	☑	☑	KEY_FELD	CHAR	30	0	Partial key of SAP object
FELD1	☐	☐	CHAR40	CHAR	40	0	Character field of length 40
FELD2	☐	☐	CHAR10	CHAR	10	0	Character Field Length = 10
	☐	☐					

Figure 4.91 Sample Header Table

Item table

The item table that belongs to the header table contains an item number as an additional key field, three similar fields for the qualification, and a quantity field with a unit of measurement for a special implementation for some units of measurement. Figure 4.92 displays the definition of the item table in Transaction SE11.



Dictionary: Maintain Table

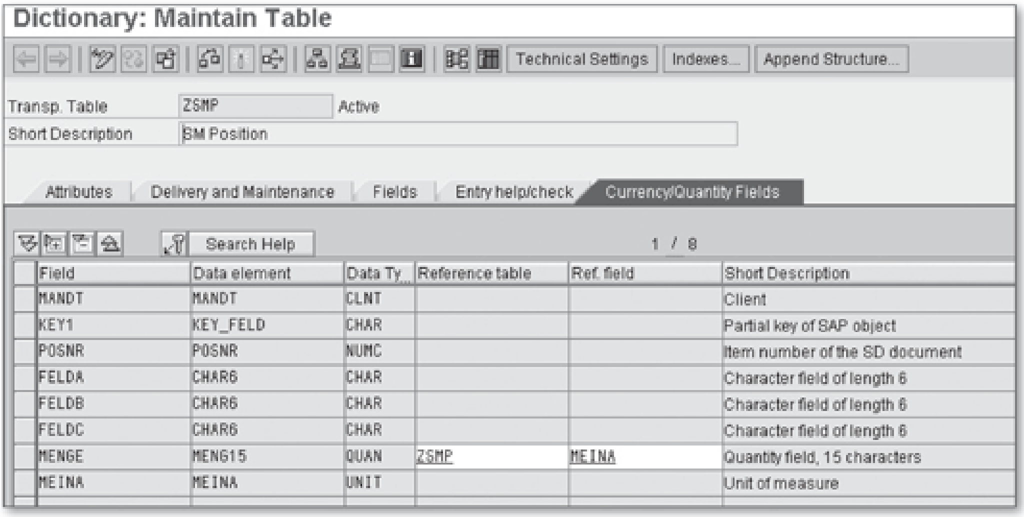
Transp. Table ZSMP Active
Short Description SM Position

Field	Key	Initi...	Data element	Data Ty...	Length	Decim...	Short Description
MANDT	☑	☑	MANDT	CLNT	3	0	Client
KEY1	☑	☑	KEY_FELD	CHAR	30	0	Partial key of SAP object
POSNR	☑	☑	POSNR	NUMC	6	0	Item number of the SD document
FELDA	□	□	CHAR6	CHAR	6	0	Character field of length 6
FELDB	□	□	CHAR6	CHAR	6	0	Character field of length 6
FELDC	□	□	CHAR6	CHAR	6	0	Character field of length 6
MENGE	□	□	MENG15	QUAN	15	3	Quantity field, 15 characters
MEINA	□	□	MEINA	UNIT	3	0	Unit of measure

Figure 4.92 Sample Item Table

Quantity fields in the Data Dictionary

For the Menge field, you need to specify which reference unit of measure is supposed to be used. Figure 4.93 displays it in the Reference Table and Reference Field columns.



Dictionary: Maintain Table

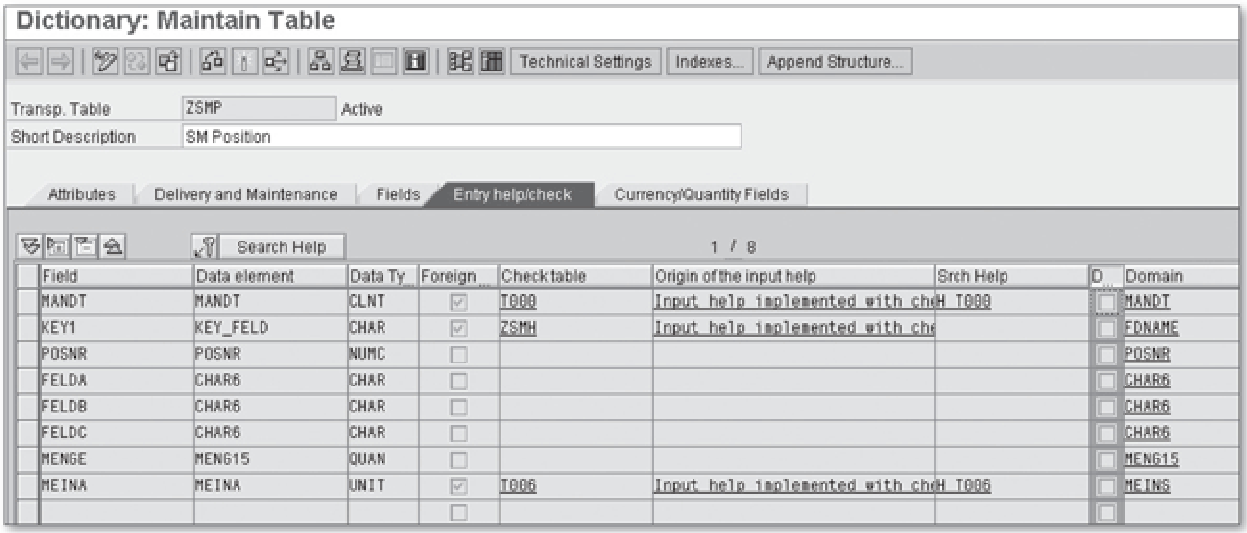
Transp. Table ZSMP Active
Short Description SM Position

Field	Data element	Data Ty...	Reference table	Ref. field	Short Description
MANDT	MANDT	CLNT			Client
KEY1	KEY_FELD	CHAR			Partial key of SAP object
POSNR	POSNR	NUMC			Item number of the SD document
FELDA	CHAR6	CHAR			Character field of length 6
FELDB	CHAR6	CHAR			Character field of length 6
FELDC	CHAR6	CHAR			Character field of length 6
MENGE	MENG15	QUAN	ZSMP	MEINA	Quantity field, 15 characters
MEINA	MEINA	UNIT			Unit of measure

Figure 4.93 Reference Unit of Measure in the Item Table

Input help

You should provide an input help or check table wherever possible (see Figure 4.94). In the ZSMP sample table, the input helps are enabled for the KEY1 field, which should refer to the header table, and for the unit of measure field (MEINA), which is copied from Customizing. In addition, SAP delivers a check table for the client (MANDT field). This table is used here.



Dictionary: Maintain Table

Transp. Table ZSMP Active
Short Description SM Position

Field	Data element	Data Ty...	Foreign ...	Check table	Origin of the input help	Srch Help	D...	Domain
MANDT	MANDT	CLNT	☑	T000	Input help implemented with che	H_T000	□	MANDT
KEY1	KEY_FELD	CHAR	☑	ZSMH	Input help implemented with che		□	FDNAME
POSNR	POSNR	NUMC	□				□	POSNR
FELDA	CHAR6	CHAR	□				□	CHAR6
FELDB	CHAR6	CHAR	□				□	CHAR6
FELDC	CHAR6	CHAR	□				□	CHAR6
MENGE	MENG15	QUAN	□				□	MENG15
MEINA	MEINA	UNIT	☑	T006	Input help implemented with che	H_T006	□	MEINS
			□				□	

Figure 4.94 Entries for the Input Check

4.4.1 Creating Custom IDoc Types and Message Types

To use the ZSMH and ZSMP sample tables in an IDoc, you must create the corresponding custom segments. Section 4.2.3, Custom Segments, already introduced the creation of segments in the context of enhancements for SAP IDocs. At this point, however, some general remarks concerning segments need to be made, which refer more to design guidelines than handling.

Segment size

In general, a segment can contain up to 1,000 characters. For performance reasons, you should transfer as few segments as possible because each segment entails additional control information. So it's useful to generate segments that are as long as possible. But you have to perform a balancing act here: The external length may be greater than the length on the database; consequently, you can't fully use the 1,000 characters for the internal length. If the table to which the segment belongs is larger, you may want to add fields to the same segment definition at a later stage. This should also be considered when defining the length of the initial segment.

Reusability

You should also try to generate reusable segments for your own interest (to reduce your work). For example, if you want to have different tables with address data, you only generate one segment for them. In this segment, you can fill whatever fields are needed in a specific case. The fields should be long enough so that you can also use them for the longest entries from the tables. Of course, you can also reuse SAP segments. So if you know a suitable SAP segment, first check if it meets your requirements before creating a custom segment.

If you already know that you want to implement communication via EDI, also have a look at the corresponding EDI message. If this message for the EDI field that corresponds to the SAP field allows for a greater field length, use the EDI field length in your segment and not the shorter SAP field length. This also facilitates later conversions. It can also be useful to check which fields are used together in the EDI standard when combining fields in a segment definition. If you then actually combine these fields in a segment, this also facilitates the conversion to the EDI standard. To meet the European standard, EDIFACT, refer to the information provided on *http://www.edifactory.de*.

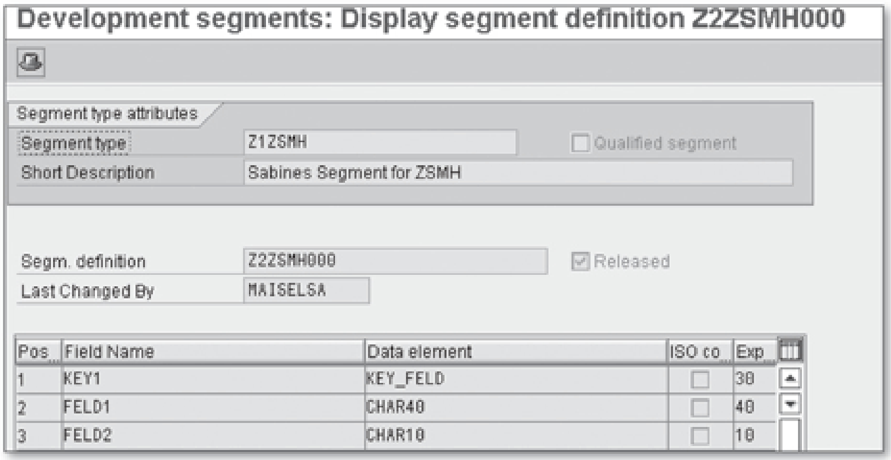
Segment for the header

For the `ZSMH` and `ZSMP` tables, three segments were created as examples: one for the header, one for the three similar fields, and one for the remaining fields of the item table. All segments are very short because they refer to sample data and not to real data.

Figure 4.95 displays the header segment. The Segment for the Position elements only contains the item number. Because IDocs have a hierarchical structure, you don't have to repeat the KEY1 key field of the header table. This would only place an unnecessary load on the system.

Segment for the item table

As you can see in Figure 4.96, the checkbox for the ISO code is selected for the unit of measure (MEINA). You still have to program this implementation, as the checkbox serves for information only. A character data element with a characteristic that has two characters more (`CHAR17`) has been selected for the MENGE field, which has 15 characters in the `ZSMP` table. As you recall, this addition is necessary to transfer the decimal point and a possibly existing minus sign to the partner in the external format.



Development segments: Display segment definition Z2ZSMH000

Segment type attributes

Segment type: Z1ZSMH   □ Qualified segment

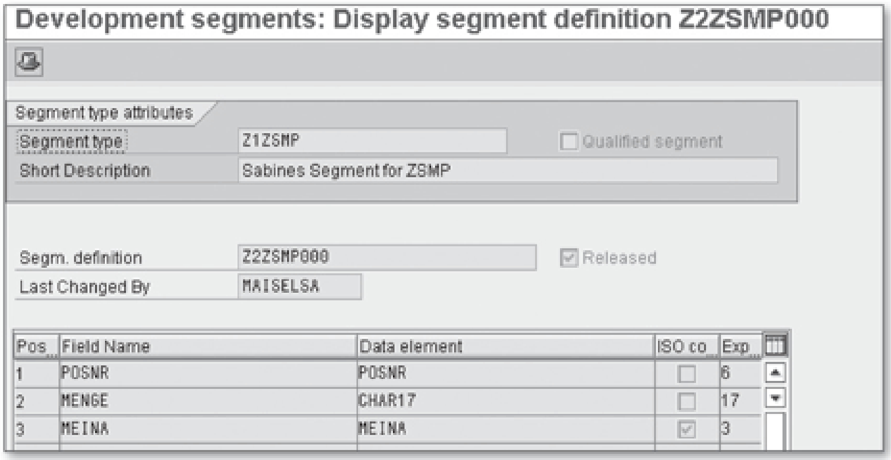
Short Description: Sabines Segment for ZSMH

Segm. definition: Z2ZSMH000   ☑ Released

Last Changed By: MAISELSA

Pos	Field Name	Data element	ISO co	Exp
1	KEY1	KEY_FELD	□	30
2	FELD1	CHAR40	□	40
3	FELD2	CHAR10	□	10

Figure 4.95 Segment for the Header Table



Development segments: Display segment definition Z2ZSMP000

Segment type attributes

Segment type: Z1ZSMP   □ Qualified segment

Short Description: Sabines Segment for ZSMP

Segm. definition: Z2ZSMP000   ☑ Released

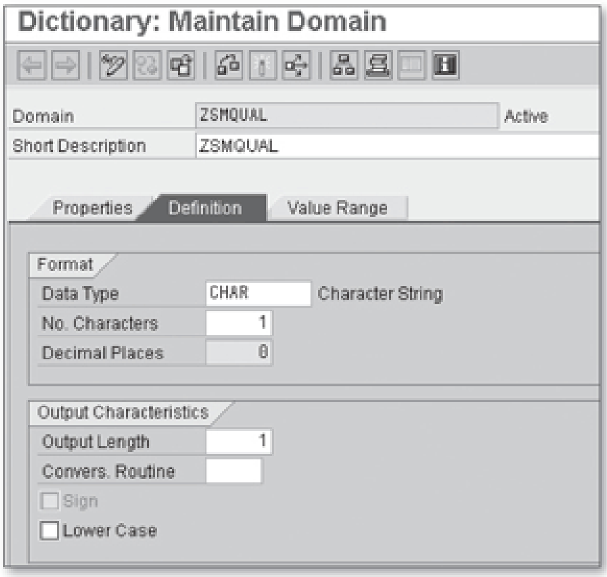
Last Changed By: MAISELSA

Pos	Field Name	Data element	ISO co	Exp
1	POSNR	POSNR	□	6
2	MENGE	CHAR17	□	17
3	MEINA	MEINA	☑	3

Figure 4.96 Segment for the Item Table

Qualifier in the DDIC

Finally there is the third segment, which will be called Z1ZSMQ. A *qualifying segment* is supposed to be used here. You can transfer various fields to a qualifying segment. A controlling field, called the *qualifier*, then indicates which field is meant. You use these fields if you don't know how many of the fields that are identical with regard to the properties will be transferred. In our example, the field can occur zero to three times, and you decide depending on the qualifier into which of the three fields, FELDA, FELDB, or FELDC, the respective value is supposed to be entered. You already know this procedure from partner roles in Sales and Distribution (SD) or Materials Management (MM) in SAP. The prerequisite here, however, is a specific domain for the qualifier, which has already been created as you can see in Figure 4.97.



Dictionary: Maintain Domain

Domain: ZSMQUAL Active
Short Description: ZSMQUAL

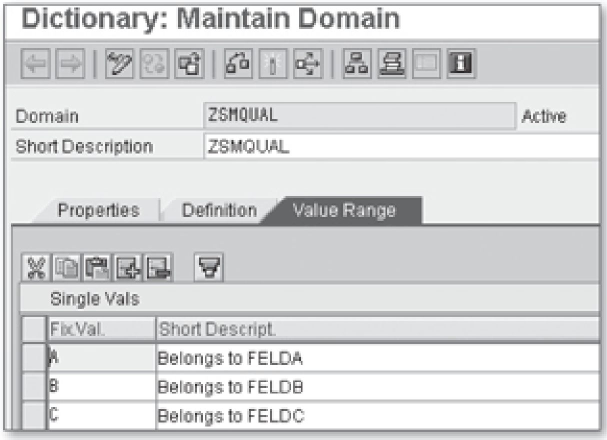
Format
Data Type: CHAR Character String
No. Characters: 1
Decimal Places: 0

Output Characteristics
Output Length: 1
Convers. Routine:
□ Sign
□ Lower Case

Figure 4.97 Domain for the Qualifier

Domain for the qualifier

You then define the possible input values in the Value Range tab of the domain. In our example, this is "A" or "B" or "C" (see Figure 4.98). If you use only one value range in the domain, the IDoc documentation displays which selection options are available for this field.



Dictionary: Maintain Domain

Domain: ZSMQUAL Active
Short Description: ZSMQUAL

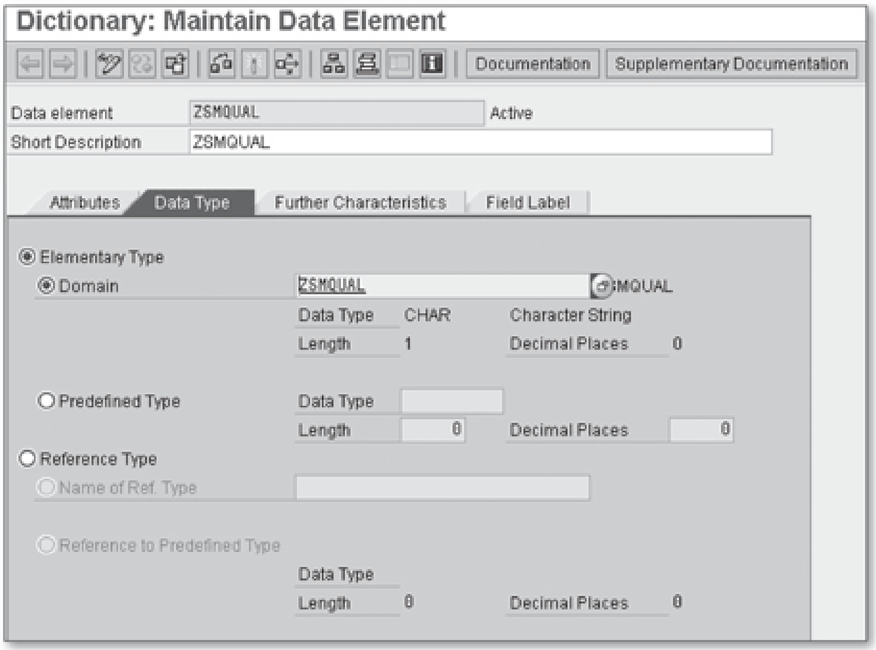
Single Vals

FixVal.	Short Descript.
A	Belongs to FELDA
B	Belongs to FELDB
C	Belongs to FELDC

Figure 4.98 List of the Input Values

Data element for the qualifier

You then use this domain to create the appropriate data element in the next step in Transaction SE11. For this purpose, assign a meaningful name within the customer namespace, and refer to the newly created domain. Figure 4.99 illustrates this using the ZSMQUAL data element as an example.



Dictionary: Maintain Data Element

Data element ZSMQUAL Active
Short Description ZSMQUAL

● Elementary Type
● Domain ZSMQUAL MQUAL
Data Type CHAR Character String
Length 1 Decimal Places 0

○ Predefined Type Data Type
Length 0 Decimal Places 0

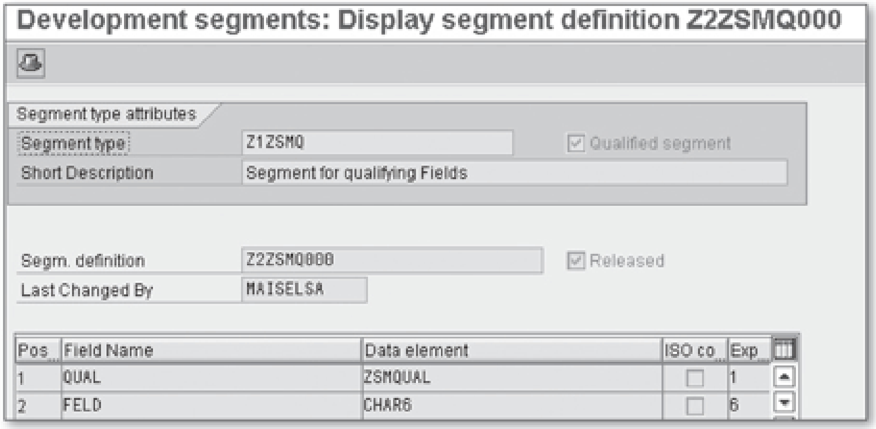
○ Reference Type
○ Name of Ref. Type

○ Reference to Predefined Type
Data Type
Length 0 Decimal Places 0

Figure 4.99 Data Element for the Qualifier

Qualifying segment

The (admittedly rather small) segment in our example now merely contains a field for the qualifier and an additional field. This field has the properties of the field that the table contains three times in an identical format, FELDA, FELDB, or FELDC. Figure 4.100 displays the third, qualifying segment of our example as Z1ZSMQ.


Development segments: Display segment definition Z2ZSMQ000

Segment type attributes
Segment type Z1ZSMQ ☑ Qualified segment
Short Description Segment for qualifying Fields

Segm. definition Z2ZSMQ000 ☑ Released
Last Changed By MAISELSA

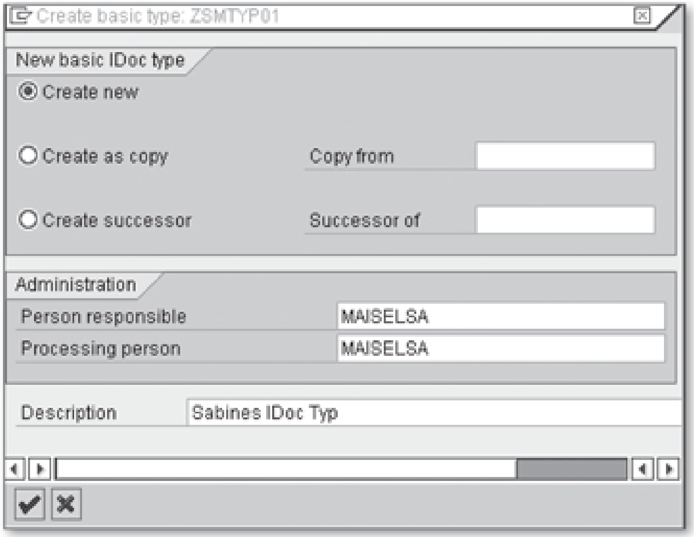
Pos	Field Name	Data element	ISO co	Exp
1	QUAL	ZSMQUAL	□	1
2	FELD	CHAR6	□	6

Figure 4.100 Qualifying Segment

Custom IDoc basic type

Don't forget to release all segments as described in Section 4.2.3, Custom Segments. After that, the system builds the IDoc type from the segments.

This is again implemented in Transaction WE30; this time, however, you create a specific basic type. The already known naming rules ("Z," "Y," or /<NAMESPACE>/ at the beginning of the name) apply here as well. Figure 4.101 displays the screen with the general specifications for the IDoc type.



Create basic type: ZSMTYP01

New basic IDoc type

● Create new

○ Create as copy — Copy from

○ Create successor — Successor of

Administration

Person responsible: MAISELSA

Processing person: MAISELSA

Description: Sabines IDoc Typ

Figure 4.101 Creating a Custom IDoc Type

Specifications for the root segment

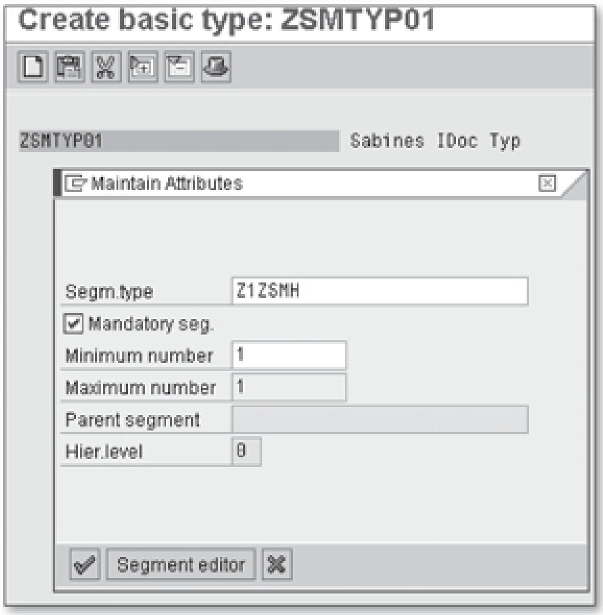
First, enter the main or root segment in the Segment Type field (in our example, this is the segment for the header table, Z1ZSMH). It can occur only once but has to occur, which is why the Mandatory Seg. checkbox is selected (see Figure 4.102).

Inserting segments

Afterward, create the item segment (here: Z1ZSMP), which can occur any number of times (see Figure 4.103). Even though it's optional, you must set the minimum number of occurrence for this segment to "1." Because the Mandatory Seg. checkbox isn't selected, the segment can also be missing. Before you navigate to the attribute maintenance, the system asks you at which level you want to add the segment.

Additional segment

In our example, the qualifying segment, Z1ZSMQ, was also added at the child level because it will be generated from the same table record at a later stage (see Figure 4.104). You can now build your entire IDoc this way.



Create basic type: ZSMTYP01

ZSMTYP01 Sabines IDoc Typ

Maintain Attributes

Segm.type: Z1ZSMH

☑ Mandatory seg.

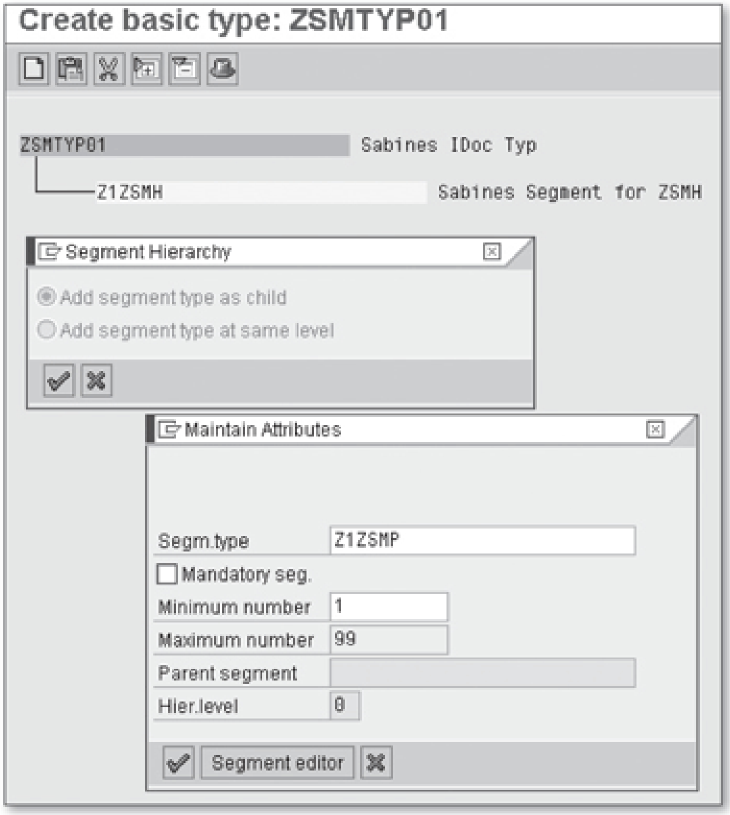
Minimum number: 1

Maximum number: 1

Parent segment:

Hier.level: 0

Figure 4.102 Defining the Root Segment



Create basic type: ZSMTYP01

ZSMTYP01 Sabines IDoc Typ

Z1ZSMH Sabines Segment for ZSMH

Segment Hierarchy

● Add segment type as child

○ Add segment type at same level

Maintain Attributes

Segm.type: Z1ZSMP

□ Mandatory seg.

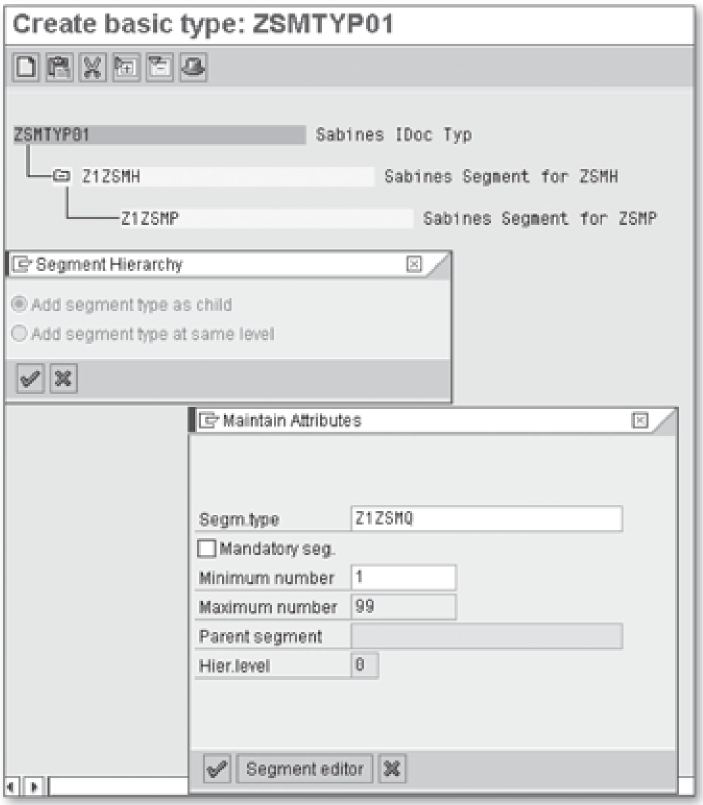
Minimum number: 1

Maximum number: 99

Parent segment:

Hier.level: 0

Figure 4.103 Specifications for the Item Segment



Create basic type: ZSMTYP01

ZSMTYP01 Sabines IDoc Typ
Z1ZSMH Sabines Segment for ZSMH
Z1ZSMP Sabines Segment for ZSMP

● Add segment type as child
○ Add segment type at same level

Segm.type Z1ZSMQ
□ Mandatory seg.
Minimum number 1
Maximum number 99
Parent segment
Hier.level 0

Figure 4.104 Specifications for the Qualifying Segment

Documentation in Transaction WE60

Transaction WE60 enables you to have the system display the documentation for your IDoc type. The system automatically generates the documentation from the data that you specified. The structure description (see Figure 4.104) then might look as shown in Table 4.1.

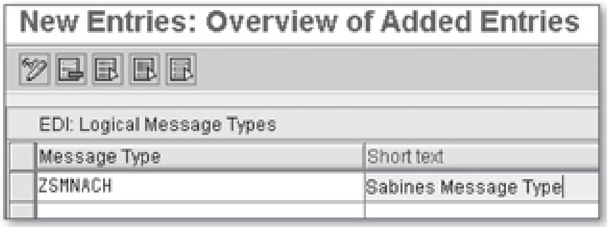
Basic Type	ZSMTYP01
ZSMTYP01	Sabines IDoc Type
Z1ZSMH	Sabines Segment for ZSMH Status: mandatory, minimum number: 1, maximum number: 1
Z1ZSMP	Sabines Segment for ZSMP Status: optional, minimum number: 1, maximum number: 99
Z1ZSMQ	Segment for Qualifying Fields Status: mandatory, minimum number: 1, maximum number: 3

Table 4.1 ZSMTYP01 Basic Type

The remaining part of the documentation isn't described here because you can obtain the corresponding information from the preceding figures.

Custom message type

When the IDoc type is complete, you have to create the message type. This is done in Transaction WE81 (the message type is called "logical message" here). Figure 4.105 shows the `ZSMNACH` sample message type.



New Entries: Overview of Added Entries

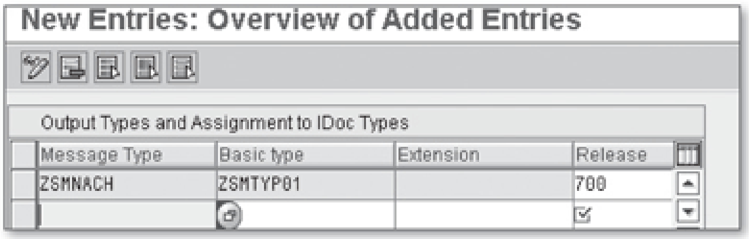
EDI: Logical Message Types

Message Type	Short text
ZSMNACH	Sabines Message Type

Figure 4.105 Creating the Message Type

Message type — IDoc type

In Transaction WE82, the message type and the IDoc type that has been created as the basic type are assigned to each other. Figure 4.106 displays this assignment for the previously created objects, `ZSMNACH` and `ZSMTYP01`.



New Entries: Overview of Added Entries

Output Types and Assignment to IDoc Types

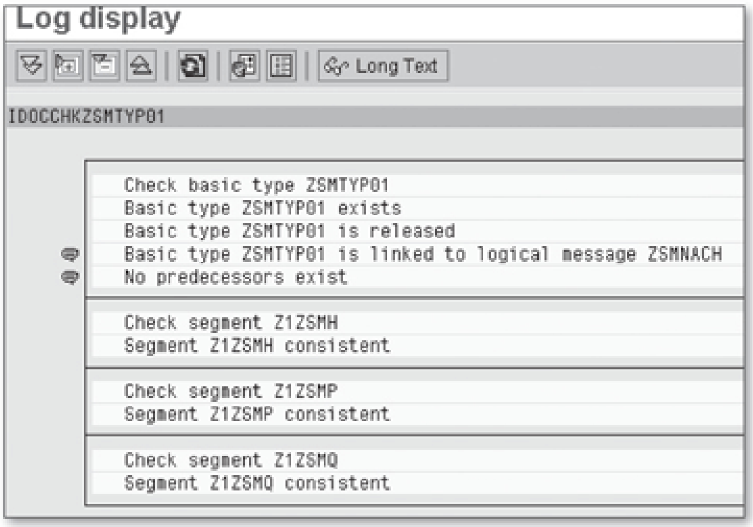
Message Type	Basic type	Extension	Release
ZSMNACH	ZSMTYP01		700

Figure 4.106 Assigning the IDoc Type to the Message Type

Check log

After you've completed everything, you can again check your results. As you can see in Figure 4.107, everything is okay for our sample IDoc.

If you need additional functions, such as reductions or links to business object types, you can configure them in Transaction BD60. This transaction is optional because a "normal" IDoc doesn't need these settings. However, you already know them from other SMD functionalities.

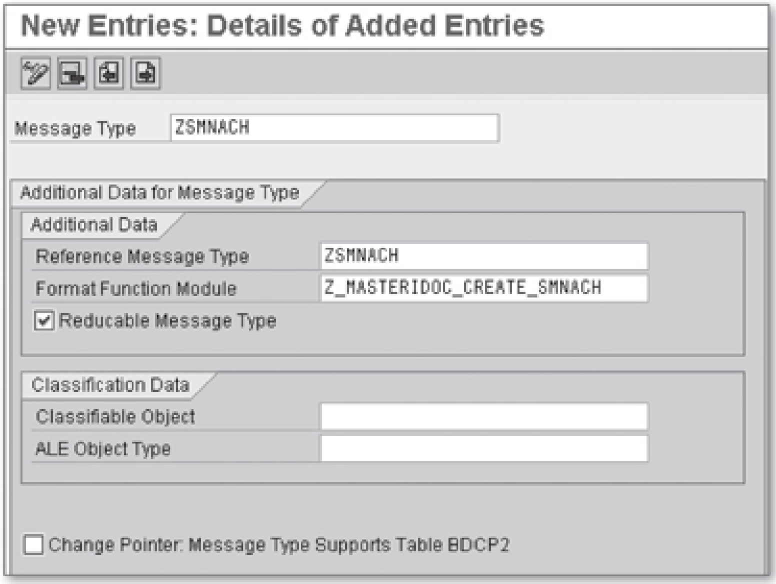


```
IDOCCHKZSMTYP01

Check basic type ZSMTYP01
Basic type ZSMTYP01 exists
Basic type ZSMTYP01 is released
Basic type ZSMTYP01 is linked to logical message ZSMNACH
No predecessors exist

Check segment Z1ZSMH
Segment Z1ZSMH consistent

Check segment Z1ZSMP
Segment Z1ZSMP consistent

Check segment Z1ZSMQ
Segment Z1ZSMQ consistent
```

Figure 4.107 Check Log for Success

Properties of the message type

Figure 4.108 shows possible Reference Message Type, Format Function Module, and Reducable Message Type properties for our sample message type. Object types and classification data haven't been created because this has to be done by the developers of the application. The object types serve to create links, and the classification data is required if you want to send master data via distribution class types.


New Entries: Details of Added Entries
Additional Data for Message Type
Additional Data
Classification Data

Message Type: ZSMNACH

Reference Message Type: ZSMNACH
Format Function Module: Z_MASTERIDOC_CREATE_SMNACH
☑ Reducable Message Type

Classifiable Object:
ALE Object Type:

☐ Change Pointer: Message Type Supports Table BDCP2

Figure 4.108 SMD Settings for the "ZSMNACH" Message Type

You can now begin generating and processing the appropriate IDocs for this message and IDoc type.

4.4.2 Generating an IDoc

When creating an IDoc, you should observe some rules in advance:

Programming guidelines

- The program should only generate segments that contain fields that aren't initial.
- The segments may only contain alphanumeric characters because the format is supposed to be generally comprehensible.
- For all languages, currencies, and units of measurement, use the corresponding ISO values as the standard version does. SAP provides function modules for conversions, which you can also use.
- As common for alphanumeric content, all fields should be filled left-aligned. You can use the `CONDENSE` command for this purpose.
- For decimal numbers, the system always uses a point as the decimal separator; the possibly existing minus sign is indicated at the end. Thousands separators are never used.
- A floating point number is displayed with a point as the decimal separator and without a thousands separator. A possible existing minus sign is indicated at the beginning this time; the exponent is always at the end.
- Date fields are displayed in the same way as on the database, that is, in the yyyyMMdd format.
- Times are also displayed in the same way as on the database, that is, in the HHmmss format.

IDoc generation

There are three options for generating an IDoc: You can generate a direct transaction or a direct send report, use message control, or work with change pointers. The respectively required Customizing for the necessary transaction codes and assignments of the function modules is described in Chapter 2, Section 2.1, Standard Methods for the IDoc Generation. This example therefore works with a function module. In general, you can use this function module, which we'll call `Z_MASTERIDOC_CREATE_ZSMNACH`, for all options mentioned earlier for the generation of IDocs. It's called in a report (see Listing 4.9).

Function module

This function module covers the main program aspects. It transfers a variable that you can use to determine the data that is supposed to be sent. In this case, it's the OBJKEY variable. To keep it simple, the variable refers to the ZSMH table; if you use larger tables, you should create a structure that only consists of the key fields. This is followed by the variables for the sender (sender partner ID SNDPRN, sender partner type SNDPRT, and sender partner function SNDPFC) and the receiver (receiver partner ID RCVPRN and receiver partner type RCVPRT). The system returns the number of the generated IDocs so that you're provided with an internal table for the control records.

Generating an IDoc: signature

Basically, a function module like this one can generate multiple IDocs, but our example is limited to one. The signature of the function module is followed by data declarations and the initialization process of all structures.

```
FUNCTION z_masteridoc_create_zsmnach.
*"----------------------------------------------------------
*"*"Local interface:
*"  IMPORTING
*"     VALUE(OBJKEY) TYPE  ZSMH
*"     VALUE(RCVPFC) TYPE  BDALEDC-RCVPFC
                                       DEFAULT SPACE
*"     VALUE(RCVPRN) TYPE  BDALEDC-RCVPRN
*"     VALUE(RCVPRT) TYPE  BDALEDC-RCVPRT
*"     VALUE(SNDPFC) TYPE  BDALEDC-SNDPFC
                                       DEFAULT SPACE
*"     VALUE(SNDPRN) TYPE  BDALEDC-SNDPRN
*"     VALUE(SNDPRT) TYPE  BDALEDC-SNDPRT
*"  EXPORTING
*"     VALUE(CREATED_COMM_IDOCS) LIKE  SY-TABIX
*"  CHANGING
*"     REFERENCE(TE_IDOC_CONTROL) TYPE  EDIDC_TT
*"----------------------------------------------------------
```

Data definitions

```
  DATA: BEGIN OF f_idoc_header .
          INCLUDE STRUCTURE edidc .
  DATA: END OF f_idoc_header .
  DATA: BEGIN OF t_idoc_data OCCURS 10 .
          INCLUDE STRUCTURE edidd .
  DATA: END OF t_idoc_data .
  DATA: BEGIN OF t_idoc_comm_control OCCURS 10 .
          INCLUDE STRUCTURE edidc .
  DATA: END OF t_idoc_comm_control .
  DATA: comm_control_lines LIKE sy-tabix .
  DATA: idoc_must_be_sent .
  DATA: wa_zlzsmh TYPE zlzsmh .
  DATA: wa_zsmh TYPE zsmh .
  DATA: wa_zlzsmp TYPE zlzsmp .
  DATA: wa_zsmp TYPE zsmp .
  DATA: it_zsmp TYPE TABLE OF zsmp .
  DATA: wa_zlzsmq TYPE zlzsmq .
* Initialize
  CLEAR   t_idoc_comm_control .
  REFRESH t_idoc_comm_control .
  CLEAR   t_idoc_data .
  REFRESH t_idoc_data .
* Check ALE distribution model
```

Listing 4.9 Data Declarations

Reading the distribution model

If you work with ALE distribution, check — before you generate the IDoc — whether the predefined combination of sender, receiver, and message type is allowed, that is, whether it's maintained in the customer distribution model. You can use the standard module ALE_MODEL_DETERMINE_IF_TO_SEND provided by SAP for this purpose (see Listing 4.10).

```
  CALL FUNCTION ‘ALE_MODEL_DETERMINE_IF_TO_SEND’
     EXPORTING
          message_type           = ‘ZSMNACH’
*             sending_system         = ‘ ‘
*             receiving_system       = ‘ ‘
              receiving_system       = rcvprn
          VALIDDATE              = SY-DATUM
     IMPORTING
          idoc_must_be_sent      = idoc_must_be_sent
     EXCEPTIONS
          own_system_not_defined = 1
          OTHERS                 = 2 .
  IF idoc_must_be_sent IS INITIAL .
     EXIT .
  ENDIF .
```

Listing 4.10 Determining the Data of the Distribution Model

Generating segments

If you get a positive result, the root segment is generated next (see Listing 4.11). Using the `Move-Corresponding` or `Corresponding Fields` statements always has the benefit that potential new fields don't lead to changes in the program.

```
* ZSMH ==> Generate E1ZSMH HEADER segment
  SELECT SINGLE * FROM zsmh
     INTO CORRESPONDING FIELDS OF wa_zsmh
     WHERE key1 = objkey-key1 .
  MOVE-CORRESPONDING wa_zsmh TO wa_z1zsmh .
```

Listing 4.11 Filling the Header Segment — Application Data

The segment generated so far is now transferred to the data record table (see Listing 4.12). Only the segment name from the control area needs to be transferred, everything else will be generated at a later stage.

```
  CLEAR t_idoc_data .
  t_idoc_data-segnam = 'Z1ZSMH' .
  t_idoc_data-sdata  = wa_z1zsmh .
```

Listing 4.12 Filling the Header Segment — Control Data

Reduction activated?

If reduction is enabled for your message type, the following function module checks which fields are active for the segment that is supposed to be processed in the reduced type (see Listing 4.13). The `ZSMNACH` type in our example is no reduction, so it will bring back all fields and segments as relevant; nevertheless, the call has been executed here for illustration purposes.

```
*   Reducing segments
  CALL FUNCTION 'IDOC_REDUCTION_FIELD_REDUCE'
    EXPORTING
      message_type = 'ZSMNACH'
      segment_type = 'Z1ZSMH'
      segment_data = t_idoc_data-sdata
    IMPORTING
      segment_data = t_idoc_data-sdata .
```

Listing 4.13 Checking if the Function Module Is Supposed to Be Reduced

The append to the IDoc data table is implemented in the next step:

```
  APPEND t_idoc_data .
```

Item segment

Afterward, the system immediately creates the next segment (see Listing 4.14).

```
* ZSMP ==> E1ZSMP
  SELECT * FROM zsmp
     INTO CORRESPONDING FIELDS OF TABLE it_zsmp
     WHERE key1 = objkey-key1 .
  LOOP AT it_zsmp INTO wa_zsmp .
    MOVE-CORRESPONDING wa_zsmp TO wa_z1zsmp .
```

Listing 4.14 Generating the Item Segment — Application Data

This is the first field with digits that are supposed to be transferred as alphanumeric characters. A `Condense` is executed:

```
    CONDENSE wa_z1zsmp-menge .
```

Converting units of measurement

Also a field exists that is supposed to be sent in the ISO code and not in the SAP unit (see Listing 4.15). It's the unit of measure. The SAP function module for the conversion direction from SAP unit to ISO code is `UNIT_OF_MEASURE_SAP_TO_ISO`.

```
    CALL FUNCTION ‘UNIT_OF_MEASURE_SAP_TO_ISO’
      EXPORTING
        sap_code    = wa_z1zsmp-meina
      IMPORTING
        iso_code    = wa_z1zsmp-meina
      EXCEPTIONS
        not_found   = 01
        no_iso_code = 02 .
    CLEAR t_idoc_data .
    t_idoc_data-segnam = ‘Z1ZSMP’ .
    t_idoc_data-sdata  = wa_z1zsmp .
```

Listing 4.15 Determining the ISO Units of Measure

Here, the reduction is missing, which is already contained in the code for the header segment in Listing 4.13. The append immediately follows:

```
    APPEND t_idoc_data.
```

Qualifying segment

In addition, an optional qualifying segment is provided that is derived from the same data record (see Listing 4.16). The system checks the three possible cases, FELDA, FELDB, or FELDC, and generates the segments if the corresponding data is available.

```
    IF NOT wa_zsmp-felda IS INITIAL .
      wa_z1zsmq-qual     = ‘A’ .
      wa_z1zsmq-feld     = wa_zsmp-felda .
    CLEAR t_idoc_data.
      t_idoc_data-segnam = ‘Z1ZSMQ’ .
      t_idoc_data-sdata  = wa_z1zsmq .
    APPEND t_idoc_data .
    ENDIF .
    IF NOT wa_zsmp-feldc IS INITIAL .
      wa_z1zsmq-qual     = ‘C’ .
      wa_z1zsmq-feld     = wa_zsmp-feldc .
    CLEAR t_idoc_data .
      t_idoc_data-segnam = ‘Z1ZSMQ’ .
      t_idoc_data-sdata  = wa_z1zsmq .
    APPEND t_idoc_data .
    ENDIF .
    IF NOT wa_zsmp-feldb IS INITIAL .
      wa_z1zsmq-qual     = ‘B’ .
      wa_z1zsmq-feld     = wa_zsmp-feldb .
    CLEAR t_idoc_data .
      t_idoc_data-segnam = ‘Z1ZSMQ’ .
      t_idoc_data-sdata  = wa_z1zsmq .
    APPEND t_idoc_data .
    ENDIF .
  ENDLOOP .
```

Listing 4.16 Generating the Optional Qualifying Segments

Control record — required fields

Next, you need your control record; however, there are only a few fields that you need to fill in yourself. In our example (see Listing 4.17), these mandatory fields are filled for sender information, receiver information, and possibly required serialization information. The system automatically specifies the timestamp, the creating or modifying person, and the like.

```
* Design control record.
  CLEAR f_idoc_header .
  f_idoc_header-mestyp = ‘ZSMNACH’ .
  f_idoc_header-idoctp = ‘ZSMTYP01’ .
  f_idoc_header-sndpfc = sndpfc .
  f_idoc_header-sndprn = sndprn .
  f_idoc_header-sndprt = sndprt .
  f_idoc_header-rcvpfc = rcvpfc .
  f_idoc_header-rcvprn = rcvprn .
  f_idoc_header-rcvprt = rcvprt .
  f_idoc_header-serial = space .
```

Listing 4.17 Filling the Mandatory Fields in the Control Record

Transfer to the communication layer

The system transfers the finally complete master IDoc to the communication layer (see Listing 4.18). During this process, the hierarchy for the individual segments is set up according to the IDoc type definition.

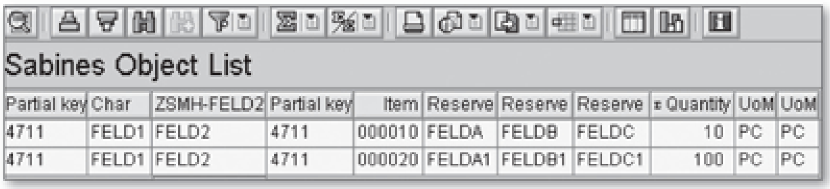
```
* Transfer IDoc to communication layer
CALL FUNCTION 'MASTER_IDOC_DISTRIBUTE'
    EXPORTING
     master_idoc_control            = f_idoc_header
    TABLES
     communication_idoc_control     = t_idoc_comm_control
     master_idoc_data               = t_idoc_data
    EXCEPTIONS
     error_in_idoc_control          = 01
     error_writing_idoc_status      = 02
     error_in_idoc_data             = 03
     sending_logical_system_unknown = 04 .
    DESCRIBE TABLE t_idoc_comm_control
    LINES comm_control_lines.
    created_comm_idocs              = comm_control_lines .
    te_idoc_control[]               = t_idoc_comm_control[] .
ENDFUNCTION .
```

Listing 4.18 Sending the Generated IDocs

Note that, for better readability, this code doesn't include some recurring aspects with which developers are usually familiar. For example, you may want to use more variables where hard code was used in this example. This example also doesn't check if the segments are filled before they are appended, which you should always do.

Complete custom IDoc

You've now created the data records for your tables. The quick view in Figure 4.109 displays the data object that is supposed to be sent as an example here.



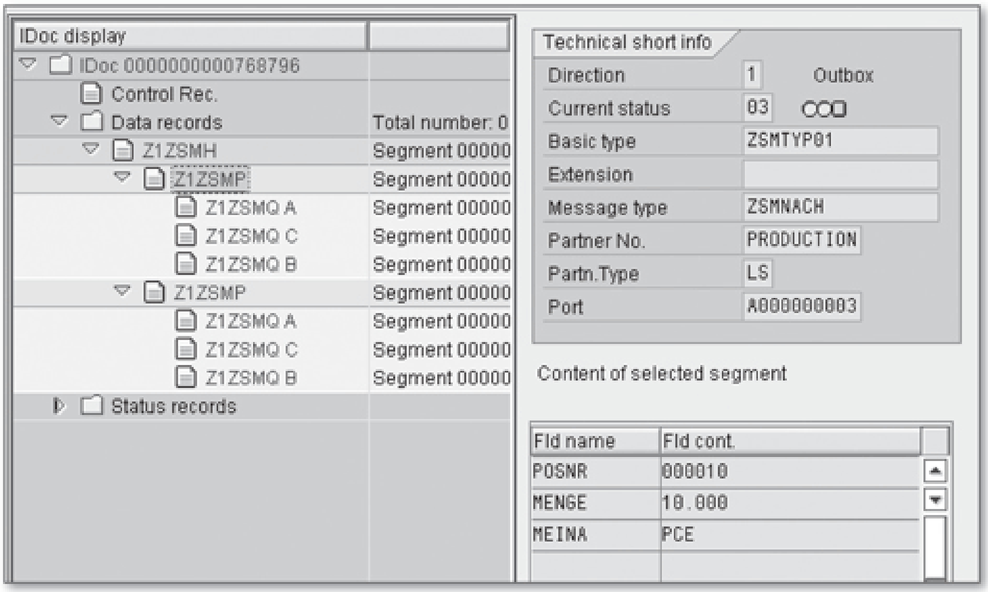
Sabines Object List

Partial key	Char	ZSMH-FELD2	Partial key	Item	Reserve	Reserve	Reserve	± Quantity	UoM	UoM
4711	FELD1	FELD2	4711	000010	FELDA	FELDB	FELDC	10	PC	PC
4711	FELD1	FELD2	4711	000020	FELDA1	FELDB1	FELDC1	100	PC	PC

Figure 4.109 Data Object to Be Sent

Individual display

Transaction BD87 lists the IDoc that has been generated from the data object (see Figure 4.110).



IDoc display

- IDoc 0000000000768796
  - Control Rec.
  - Data records — Total number: 0
    - Z1ZSMH — Segment 00000
      - Z1ZSMP — Segment 00000
        - Z1ZSMQ A — Segment 00000
        - Z1ZSMQ C — Segment 00000
        - Z1ZSMQ B — Segment 00000
      - Z1ZSMP — Segment 00000
        - Z1ZSMQ A — Segment 00000
        - Z1ZSMQ C — Segment 00000
        - Z1ZSMQ B — Segment 00000
  - Status records

Technical short info

Direction	1	Outbox
Current status	03	
Basic type	ZSMTYP01	
Extension		
Message type	ZSMNACH	
Partner No.	PRODUCTION	
Partn.Type	LS	
Port	A000000003	

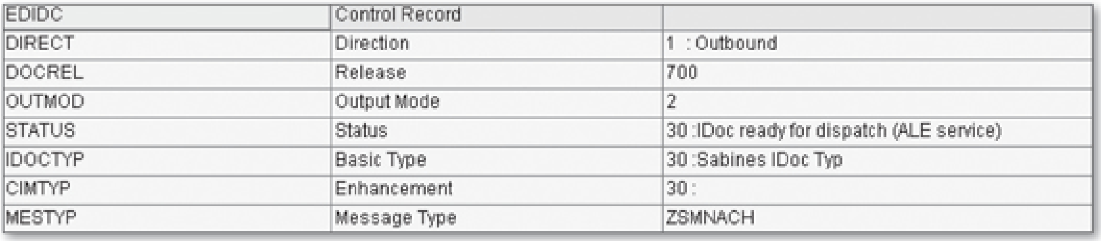
Content of selected segment

Fld name	Fld cont.
POSNR	000010
MENGE	10.000
MEINA	PCE

Figure 4.110 Sample IDoc in Transaction BD87

Control record

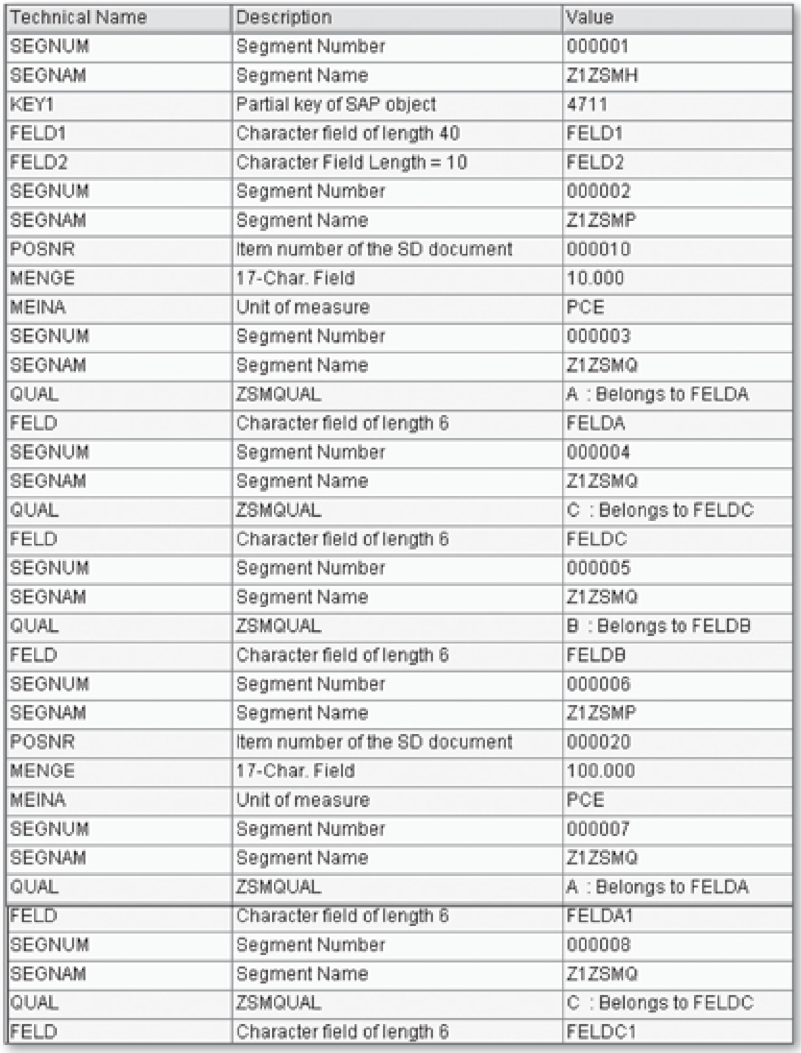
The control record (see Figure 4.111) indicates that your IDoc type (here: `Sabines IDoc Type`) and your message type (here: `ZSMNACH`) were actually used.



EDIDC	Control Record	
DIRECT	Direction	1 : Outbound
DOCREL	Release	700
OUTMOD	Output Mode	2
STATUS	Status	30 :IDoc ready for dispatch (ALE service)
IDOCTYP	Basic Type	30 :Sabines IDoc Typ
CIMTYP	Enhancement	30 :
MESTYP	Message Type	ZSMNACH

Figure 4.111 Section Taken from the Control Record

The data of the individual segments now also clearly shows the conversion of the data from "PC," as the internal unit of measure piece, to "PCE" as the corresponding ISO unit of measure in the MEINA field (see Figure 4.112). The list additionally includes the qualifying segment, Z1ZSMQ, which has been created as an example.



Technical Name	Description	Value
SEGNUM	Segment Number	000001
SEGNAM	Segment Name	Z1ZSMH
KEY1	Partial key of SAP object	4711
FELD1	Character field of length 40	FELD1
FELD2	Character Field Length = 10	FELD2
SEGNUM	Segment Number	000002
SEGNAM	Segment Name	Z1ZSMP
POSNR	Item number of the SD document	000010
MENGE	17-Char. Field	10.000
MEINA	Unit of measure	PCE
SEGNUM	Segment Number	000003
SEGNAM	Segment Name	Z1ZSMQ
QUAL	ZSMQUAL	A : Belongs to FELDA
FELD	Character field of length 6	FELDA
SEGNUM	Segment Number	000004
SEGNAM	Segment Name	Z1ZSMQ
QUAL	ZSMQUAL	C : Belongs to FELDC
FELD	Character field of length 6	FELDC
SEGNUM	Segment Number	000005
SEGNAM	Segment Name	Z1ZSMQ
QUAL	ZSMQUAL	B : Belongs to FELDB
FELD	Character field of length 6	FELDB
SEGNUM	Segment Number	000006
SEGNAM	Segment Name	Z1ZSMP
POSNR	Item number of the SD document	000020
MENGE	17-Char. Field	100.000
MEINA	Unit of measure	PCE
SEGNUM	Segment Number	000007
SEGNAM	Segment Name	Z1ZSMQ
QUAL	ZSMQUAL	A : Belongs to FELDA
FELD	Character field of length 6	FELDA1
SEGNUM	Segment Number	000008
SEGNAM	Segment Name	Z1ZSMQ
QUAL	ZSMQUAL	C : Belongs to FELDC
FELD	Character field of length 6	FELDC1

Figure 4.112 Data Records

4.4.3 Updating an IDoc

The inbound processing is now supposed to process the message and IDoc types. For this purpose, you need the appropriate function module again.

Function Module Restrictions

The sample module only contains the data that is necessary for IDoc processing. Usually, the system is supposed to evaluate all return codes, check the user authorizations, and set lock entries. Furthermore, the example uses only one IDoc per call (theoretically, multiple IDocs are feasible) and none of the ALE services is deployed. Comprehensive checks for the completeness and correctness of the data are missing as well. Because these checks are application-specific, you must consult the person responsible for the module and integrate them yourself.

You can obtain the variables that recur in this sample module by making the following entry in the TOP include of your function group:

```
include mbdconwf. "Report containing the ALE constants.
```

Among other things, the `mbdconwf` include contains the constants that are listed in Table 4.2 and used later. This is merely a subset of the variables from `mbdconwf`.

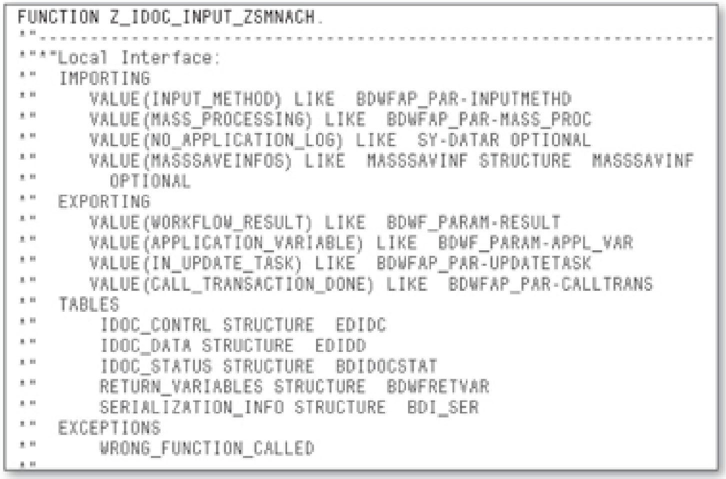
Fixed values in the SAP include

Variable	Fixed Value
C_WF_RESULT_ERROR	99999
C_WF_RESULT_DELETE_IDOC	99998
C_WF_RESULT_WI_COMPLETE	99997
C_WF_RESULT_OK	0
C_WF_RESULT_RETRY_IDOC	1
C_WF_RESULT_CONTINUE_IDOC	2
C_IDOC_STATUS_OK	53
C_IDOC_STATUS_ERROR	51

Table 4.2 Fixed Values from the "mbdconwf" Include

Inbound function module: Signature

While the function modules aren't subject to further restrictions in the outbound processing, inbound function modules must have a uniform signature because they are automatically called by the ALE communication layer. Figure 4.113 shows the necessary parameters and their typing. Parameters of the `Tables` type are no longer common in more recent releases; however, here they have to be used to meet the requirements of the ALE communication layer. If required, you can copy one of the existing IDoc inbound function modules.



```
FUNCTION Z_IDOC_INPUT_ZSMNACH.
*"----------------------------------------------------------------
*"*"Local Interface:
*"  IMPORTING
*"     VALUE(INPUT_METHOD) LIKE  BDWFAP_PAR-INPUTMETHD
*"     VALUE(MASS_PROCESSING) LIKE  BDWFAP_PAR-MASS_PROC
*"     VALUE(NO_APPLICATION_LOG) LIKE  SY-DATAR OPTIONAL
*"     VALUE(MASSSAVEINFOS) LIKE  MASSSAVINF STRUCTURE  MASSSAVINF
*"       OPTIONAL
*"  EXPORTING
*"     VALUE(WORKFLOW_RESULT) LIKE  BDWF_PARAM-RESULT
*"     VALUE(APPLICATION_VARIABLE) LIKE  BDWF_PARAM-APPL_VAR
*"     VALUE(IN_UPDATE_TASK) LIKE  BDWFAP_PAR-UPDATETASK
*"     VALUE(CALL_TRANSACTION_DONE) LIKE  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
*"  EXCEPTIONS
*"      WRONG_FUNCTION_CALLED
*"
```

Figure 4.113 Signature of an Inbound Function Module

Checking the validity of the function modules

To not use resources unnecessarily, you must first check if the function module corresponds to the IDoc that is supposed to be processed. For this purpose, you can check the message type if the message types are non-reducible (see Listing 4.19).

```
IF idoc_contrl-mestyp <> 'ZSMNACH'
   RAISE wrong_function_called .
ENDIF .
```

Listing 4.19 Checking the Message Type

For reducible IDocs, a different message type can be used so that you have to check the IDoc type(s) that are permitted for the reducible message type in this case (see Listing 4.20).

```
IF  idoc_contrl-idoctp <> 'ZSMTYP01' .
    RAISE wrong_function_called .
ENDIF.
```

Listing 4.20 Checking the IDoc Type

This is followed by the variable definitions (see Listing 4.21).

```
  DATA: subrc LIKE sy-subrc .
* Structure for the header data
  DATA: wa_zsmh TYPE zsmh .
* Variables for the items
  DATA: it_zsmp TYPE TABLE OF zsmp .
```

Listing 4.21 Declaration of the Required Variables

Starting the subroutine

Then, the system reads the IDoc and copies the data to transfer the structures (see Listing 4.22). In this example, the function module can process one IDoc only, so the transferred table, which contains merely one data record, is always read using `Index 1`. If you want to use a mass-compatible module, the processing takes place in a loop.

```
READ TABLE idoc_contrl INDEX 1 .
PERFORM idoc_process_zsmnach TABLES idoc_data
  idoc_status it_zsmp
  USING idoc_contrl
  CHANGING subrc  wa_zsmh .
```

Listing 4.22 Reading the Control Record

The content of the form is structured as illustrated in Listing 4.23. To provide an example that considers the new technologies for the table declaration, which are available as of Release 4.0, the table type is included here.

Transferring the IDoc data

Variables for the application data

```
FORM   idoc_process_zsmnach
TABLES t_idoc_data STRUCTURE edidd
       t_idoc_status STRUCTURE bdidocstat
it_zsmp structure zsmp
USING f_idoc_contrl STRUCTURE edidc
CHANGING subrc LIKE sy-subrc
wa_zsmh like zsmh.
* Structure for the header data
  DATA: key1 TYPE zsmh-key1 .
* Variables for the items
  DATA: wa_zsmp TYPE zsmp .
  DATA: posi TYPE zsmp-posnr .
  DATA: tabix TYPE sy-tabix .
```

Variables for the segments

Retrieving data from the IDoc

```
* Data for the IDoc segments:
  DATA: wa_z1zsmh TYPE z1zsmh .
  DATA: wa_z1zsmp TYPE z1zsmp .
  DATA: wa_z1zsmq TYPE z1zsmq .
* Transfer IDoc data
  LOOP AT t_idoc_data WHERE docnum = f_idoc_contrl-docnum .
    CASE t_idoc_data-segnam .
      WHEN ‘Z1ZSMH’ .
        CLEAR key1  .
        wa_z1zsmh = t_idoc_data-sdata .
        MOVE-CORRESPONDING wa_z1zsmh TO wa_zsmh .
        key1 = wa_zsmh-key1 .
      WHEN ‘Z1ZSMP’ .
        CLEAR posi .
        wa_z1zsmp = t_idoc_data-sdata .
        MOVE-CORRESPONDING wa_z1zsmp TO wa_zsmp .
        wa_zsmp-key1 = key1 .
```

Listing 4.23 Transferring the User Data from the IDoc Segments

Converting the unit of measure from ISO to SAP

In this case, the unit of measure is also converted, this time from the ISO code to the SAP unit. *Importing parameter unique*, which isn't evaluated here, is returned to X if exactly one appropriate value has been found (see Listing 4.24).

```
CALL FUNCTION ‘UNIT_OF_MEASURE_ISO_TO_SAP’
         EXPORTING
            iso_code      = wa_zsmp-meina
         IMPORTING
           SAP_CODE       = wa_zsmp-meina
*          UNIQUE         =
         EXCEPTIONS
           NOT_FOUND      = 1
           OTHERS         = 2 .
         APPEND wa_zsmp TO it_zsmp .
           posi           = wa_zsmp-posnr .
```

Listing 4.24 Converting from ISO Units of Measure to Internal Units of Measure

Qualified segment in inbound processing

This is followed by the evaluation of the next segment, the qualified segment (see Listing 4.25).

```
      WHEN ‘Z1ZSMQ’ .
        wa_z1zsmq = t_idoc_data-sdata .
      READ TABLE it_zsmp WITH KEY key1 = key1
        posnr = posi INTO  wa_zsmp .
        tabix = sy-tabix .
      CASE wa_z1zsmq-qual .
      WHEN ‘A’ .
        wa_zsmp-felda = wa_z1zsmq-feld .
      WHEN ‘B’ .
        wa_zsmp-feldb = wa_z1zsmq-feld .
      WHEN ‘C’ .
        wa_zsmp-feldc = wa_z1zsmq-feld .
      ENDCASE .
      MODIFY it_zsmp FROM wa_zsmp INDEX tabix .
      CLEAR wa_zsmp .
    ENDCASE .
  ENDLOOP .
ENDFORM .                     "IDOC_PROCESS_ZSMNACH
```

Listing 4.25 Processing of the Qualifying Segment, “Z1ZSMQ”

If the data contains errors, for example, if mandatory segments are missing or data needs to be checked against Customizing and doesn't match, modify subrc to trigger an error in the calling module.

Posting using the application module

Afterward, if the return value is correct, the system transfers the data to the actual update module, which is always called in the update mode (see Listing 4.26). The ALE communication layer then triggers the Commit Work and ensures that the IDoc status and the application object are updated or discarded simultaneously.

```
* Everthing ok so far? Then post
  IF subrc = 0 .
    CALL FUNCTION ‘Z_ZSM_CREATE’
      EXPORTING
        header = wa_zsmh
        posis  = it_zsmp
      EXCEPTIONS
        OTHERS = 1 .
```

Listing 4.26 Calling the Update Module of the Application

Now, the system evaluates the return code of the update module and sets the status value of the IDoc accordingly (see Listing 4.27). The error and success messages of this example are rather basic, of course. You should use a nice message class and also transfer the `t_idoc_status-msgid` and `t_idoc_status-msgno` values, which are indicated by asterisks, as created in your message class.

Setting the error status

Setting the success status

```
  IF sy-subrc <> 0 .
    subrc = 1 .
* Error? Then copy to status text
      PERFORM status_fill_sy_error
      TABLES idoc_status
      USING idoc_data
      sy 'Error' 'My module' .
    ELSE .
* Ok? Then success status
      idoc_status-docnum = idoc_contrl-docnum .
      idoc_status-status = c_idoc_status_ok .
      idoc_status-msgty  = 'S' .
*      t_idoc_status-msgid = your_msgid. "Global variable.
*      t_idoc_status-msgno = msgno_success."Global variable.
      idoc_status-msgv1  = 'ok' .
      APPEND idoc_status .
    ENDIF . "if sy-subrc <> 0 .
  ENDIF .
* Fill the ALE export parameters
```

Listing 4.27 Checking the Status of the IDoc Update and Message Assignment

Workflow handling

If successful, you must now ensure — if a connection to the workflow is given — that an eventually existing error workflow is canceled; if an error occurs, the corresponding workflow needs to be triggered. This is done by the ALE communication layer, which uses the values that are transferred here (see Listing 4.28) and the Customizing settings that are described in Section 4.4.5, Error Workflow for Custom IDocs.

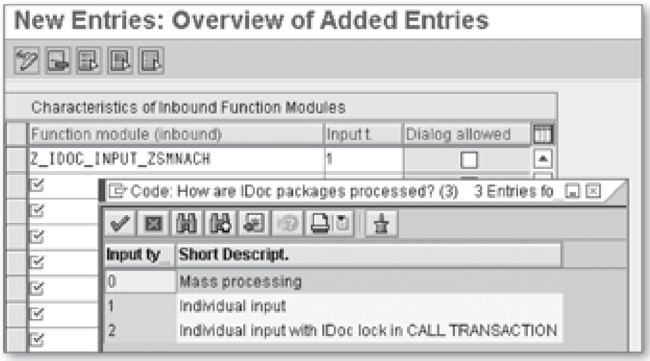
```
  CLEAR in_update_task .
  CLEAR call_transaction_done . "Call Transaction not used .
  IF subrc <> 0 . "Error occurred
    workflow_result             = c_wf_result_error .
    return_variables-wf_param   = c_wf_par_error_idocs .
    return_variables-doc_number = idoc_contrl-docnum .
    APPEND return_variables .
  ELSE . "IDoc processed successfully
    workflow_result              = c_wf_result_ok .
    return_variables-wf_param    = c_wf_par_processed_idocs .
    return_variables-doc_number = idoc_contrl-docnum .
    APPEND return_variables .
    return_variables-wf_param    = c_wf_par_appl_objects .
    APPEND return_variables .
  ENDIF .
```

Listing 4.28 Transferring the Data to the Error Workflow

You usually don't have to program the actual update module yourself because the application must already have developed something that creates the same data via dialog transaction. To ensure that the data resulting from updating an IDoc and the data that a user has manually created is identical, you should always have both procedures access the same function module. Here, the checks, locks, and authorization queries defined by the application are also coded.

Assignment to the process code

In outbound processing, you only have to assign function modules to process code if you use output determination. In inbound processing, this assignment is always required. Further activities therefore involve specifying the characteristics of the function module in Transaction BD51. Figure 4.114 displays the selection of the characteristics. Option 2 (Individual Input with IDoc Lock in Call Transaction) is rarely selected due to performance reasons. Our sample function module uses option 1, Individual Input. The code examples already indicated the requirements for using Mass Processing (option 0). Basically, it's all about integrating everything that has been implemented only once into a loop via a complete table.



New Entries: Overview of Added Entries

Characteristics of Inbound Function Modules

Function module (inbound)	Input t.	Dialog allowed
Z_IDOC_INPUT_ZSMNACH	1	☐

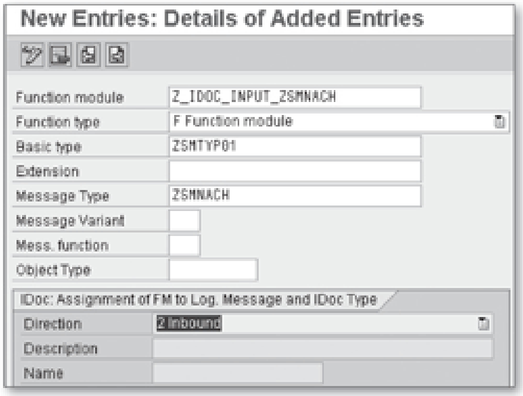
Code: How are IDoc packages processed? (3) 3 Entries fo

Input ty	Short Descript.
0	Mass processing
1	Individual input
2	Individual input with IDoc lock in CALL TRANSACTION

Figure 4.114 Processing Type of the Module

Assignment to the application

Afterward, you assign the function module to its message type and all possible IDoc types using Transaction WE57 (see Figure 4.115). In the Object Type field, enter an object type from the Business Object Repository (BOR), for example, `BUS1001006` for the material. It's this assignment that enables the system to write links between the IDoc and the business object that has been created and display them in Transaction BD87. Describing the development of business object types for application objects is beyond the scope of this book, however.



New Entries: Details of Added Entries

Function module	Z_IDOC_INPUT_ZSMNACH
Function type	F Function module
Basic type	ZSMTYP01
Extension	
Message Type	ZSMNACH
Message Variant	
Mess. function	
Object Type	

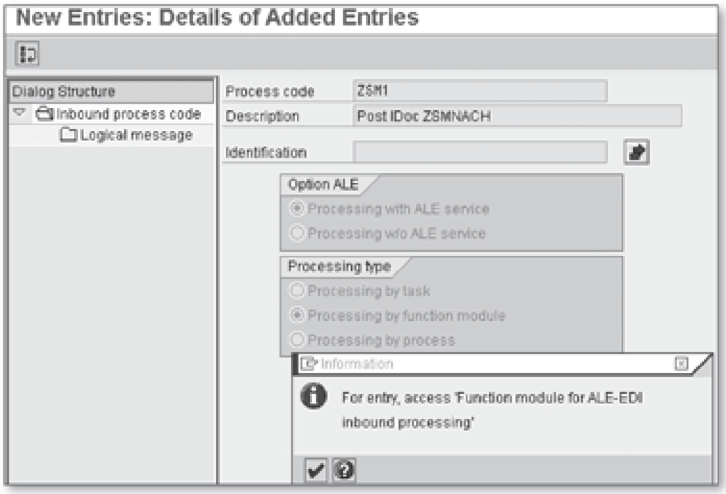
IDoc: Assignment of FM to Log. Message and IDoc Type

Direction	2 Inbound
Description	
Name	

Figure 4.115 Additional Characteristics of the Inbound Function Module

Creating process code

After assigning the function module, you create the process code in Transaction WE42. In the Option ALE tab, you can choose if you want to work with or without ALE services, and the Processing Type tab enables you to select the procedure for the process code (see Figure 4.116).



New Entries: Details of Added Entries

Dialog Structure
- Inbound process code
  - Logical message

Process code	ZSM1
Description	Post IDoc ZSMNACH
Identification	

Option ALE
- ● Processing with ALE service
- ○ Processing w/o ALE service

Processing type
- ○ Processing by task
- ● Processing by function module
- ○ Processing by process

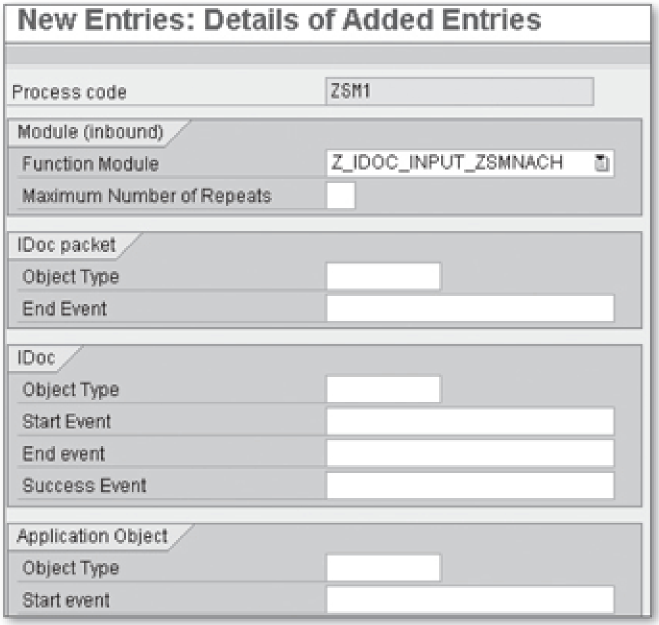
Information

For entry, access 'Function module for ALE-EDI inbound processing'

Figure 4.116 Process Code in Inbound Processing

Characteristics of the module

When creating a new process code, the system automatically navigates you to the input screen for additional characteristics if you select the Processing by Function Module OPTION. Later, you can go there by clicking on the ➡ icon. Figure 4.117 shows these additional characteristics. Here, you also specify the objects and events for handling errors via workflow. In our example, nothing is defined here, so errors can currently only be processed via Transaction BD87. Maximum Number of Repeats specifies the maximum number of attempts for successfully posting an IDoc to an application. If this isn't successful within this number of attempts, manual error handling is required.



New Entries: Details of Added Entries

Process code: ZSM1

Module (inbound)

Function Module: Z_IDOC_INPUT_ZSMNACH

Maximum Number of Repeats:

IDoc packet

Object Type:

End Event:

IDoc

Object Type:

Start Event:

End event:

Success Event:

Application Object

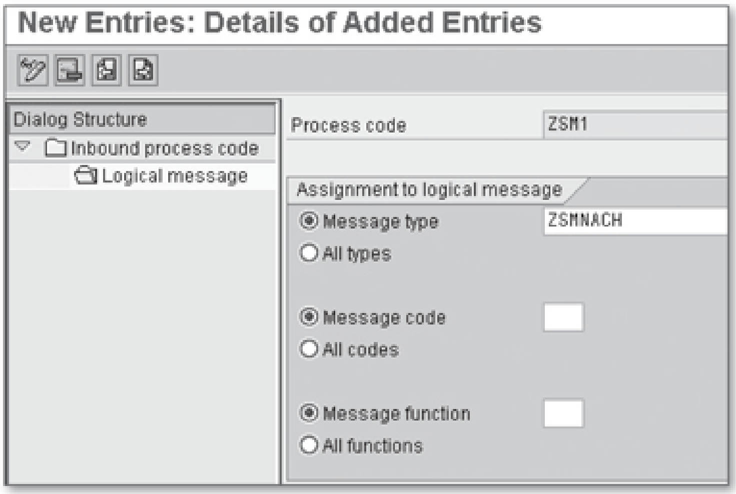
Object Type:

Start event:

Figure 4.117 Assigning the Inbound Function Module

Assigning the message type to a logical message

Finally, you can assign your message type to a logical message in Transaction WE41 (see Figure 4.118), which concludes your development work.



New Entries: Details of Added Entries

Dialog Structure
- Inbound process code
  - Logical message

Process code: ZSM1

Assignment to logical message

● Message type: ZSMNACH
○ All types

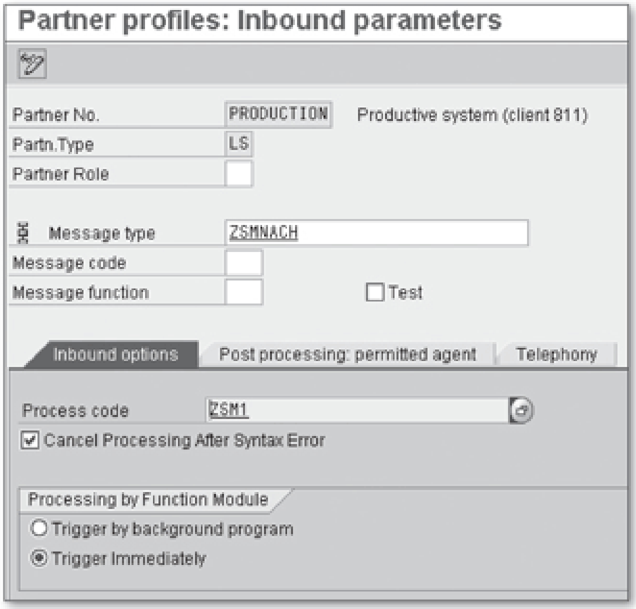
● Message code:
○ All codes

● Message function:
○ All functions

Figure 4.118 Assigning the Message Type

All you need now is the appropriate partner profile in Transaction WE20 that uses your process code to enable you to receive and update IDocs (see Figure 4.119).

Partner profile with custom process code



Partner profiles: Inbound parameters

Partner No.: PRODUCTION Productive system (client 811)
Partn.Type: LS
Partner Role:

Message type: ZSMNACH
Message code:
Message function: □ Test

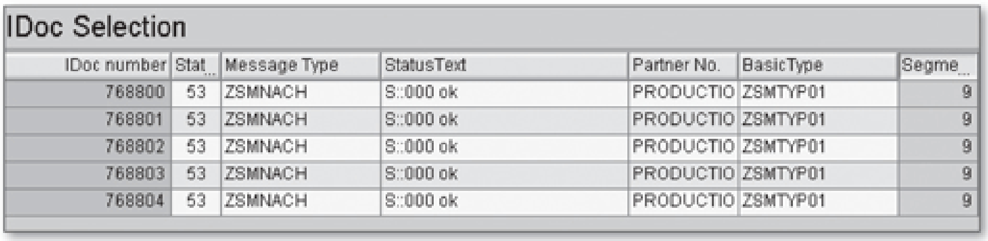
Process code: ZSM1
☑ Cancel Processing After Syntax Error

Processing by Function Module
○ Trigger by background program
● Trigger Immediately

Figure 4.119 Partner Profile — Inbound Parameters

Sample IDocs

Figure 4.120 shows some successfully posted IDocs with a proper success message.



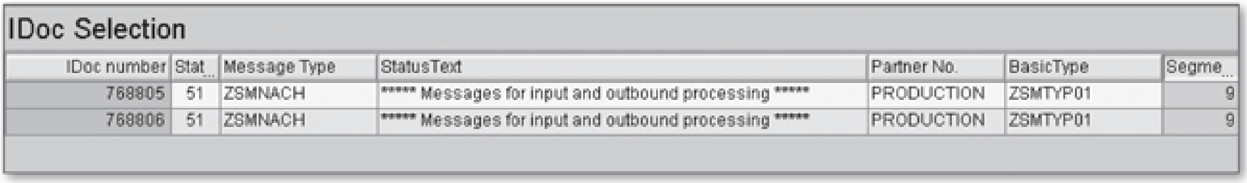
IDoc Selection

IDoc number	Stat...	Message Type	StatusText	Partner No.	BasicType	Segme...
768800	53	ZSMNACH	S::000 ok	PRODUCTIO	ZSMTYP01	9
768801	53	ZSMNACH	S::000 ok	PRODUCTIO	ZSMTYP01	9
768802	53	ZSMNACH	S::000 ok	PRODUCTIO	ZSMTYP01	9
768803	53	ZSMNACH	S::000 ok	PRODUCTIO	ZSMTYP01	9
768804	53	ZSMNACH	S::000 ok	PRODUCTIO	ZSMTYP01	9

Figure 4.120 Successfully Updated IDocs

Faulty IDocs

Figure 4.121, in turn, shows faulty IDocs. The status text, however, is somewhat basic. You should specify errors as detailed as possible to facilitate the work for persons who monitor the IDocs if errors occur.



IDoc Selection

IDoc number	Stat...	Message Type	StatusText	Partner No.	BasicType	Segme...
768805	51	ZSMNACH	***** Messages for input and outbound processing *****	PRODUCTION	ZSMTYP01	9
768806	51	ZSMNACH	***** Messages for input and outbound processing *****	PRODUCTION	ZSMTYP01	9

Figure 4.121 IDocs with Application Errors

4.4.4 Generating IDoc Function Modules

Generating suitable ALE interfaces

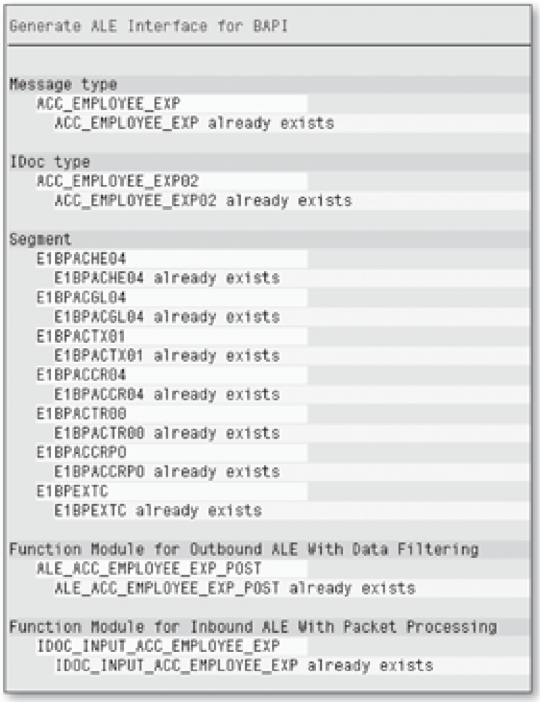
There are two options to have the system automatically generate modules for creating and updating an IDoc. The first option is to generate suitable IDoc interfaces to BAPIs. In this process, the message type and the IDoc type are generated. The second option is based on a function module. Both methods are described in the following sections.

Having the System Generate IDoc Modules from BAPI

Segment generation in Transaction BDBG

The segments are built from import and change parameters of the signature. (The export parameters aren't required here because IDocs never result in a response.) For this purpose, as many individual variables as possible become a segment, structures are provided with a separate segment that only occurs once, and table types obtain a separate segment that occurs multiple times. Then, a module is generated respectively for generating as well as for updating the corresponding IDoc.

Transaction BDBG generates the corresponding IDoc interface to a BAPI, and in Figure 4.122, you can view an ALE interface that has already been delivered by SAP in the Function display.



```
Generate ALE Interface for BAPI

Message type
   ACC_EMPLOYEE_EXP
     ACC_EMPLOYEE_EXP already exists

IDoc type
   ACC_EMPLOYEE_EXP02
     ACC_EMPLOYEE_EXP02 already exists

Segment
   E1BPACHE04
     E1BPACHE04 already exists
   E1BPACGL04
     E1BPACGL04 already exists
   E1BPACTX01
     E1BPACTX01 already exists
   E1BPACCR04
     E1BPACCR04 already exists
   E1BPACTR00
     E1BPACTR00 already exists
   E1BPACCRPO
     E1BPACCRPO already exists
   E1BPEXTC
     E1BPEXTC already exists

Function Module for Outbound ALE With Data Filtering
   ALE_ACC_EMPLOYEE_EXP_POST
     ALE_ACC_EMPLOYEE_EXP_POST already exists

Function Module for Inbound ALE With Packet Processing
   IDOC_INPUT_ACC_EMPLOYEE_EXP
     IDOC_INPUT_ACC_EMPLOYEE_EXP already exists
```

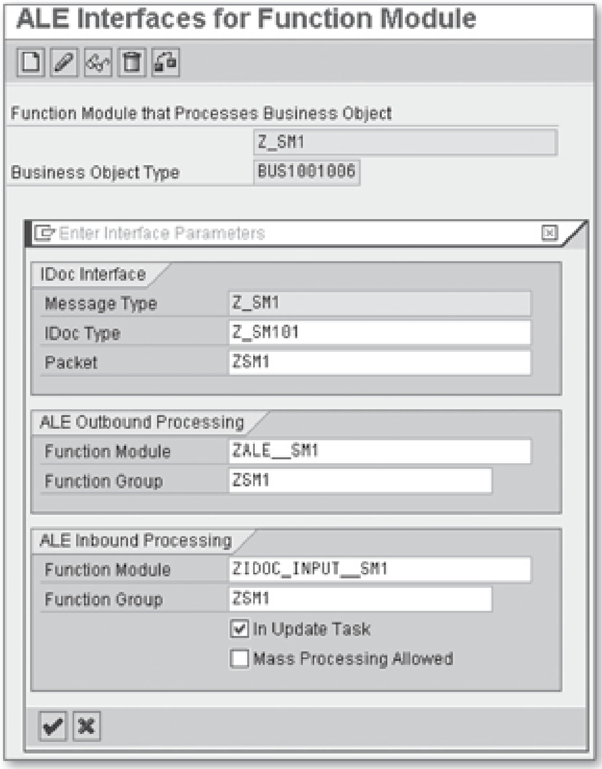
Figure 4.122 Display of an Already Existing Interface

The update via a generated module is always carried out with the same process code: BAPI for individual processing and BAPP for multiprocessing. So, you don't need a custom process code.

Having the System Generate IDoc Modules from Function Modules

ALE interface for function modules

The second option to automatically generate IDoc modules is based on a function module that hasn't been implemented as a BAPI. More restrictions apply here. The function module must contain only one import and one export parameter. However, the import parameter may be a complex variable. The export parameter must be of the BAPIRETM type if you have a simple import variable, and it must be of the `BAPIRETS` type if you have a complex transfer variable. Transaction BDFG is required for this purpose, and although it isn't a BAPI, the reference to a business object type from the BOR is mandatory. Then you can specify the names, packages, and properties of your objects using Transaction BDFG (see Figure 4.123).



ALE Interfaces for Function Module

Function Module that Processes Business Object: Z_SM1

Business Object Type: BUS1001006

Enter Interface Parameters

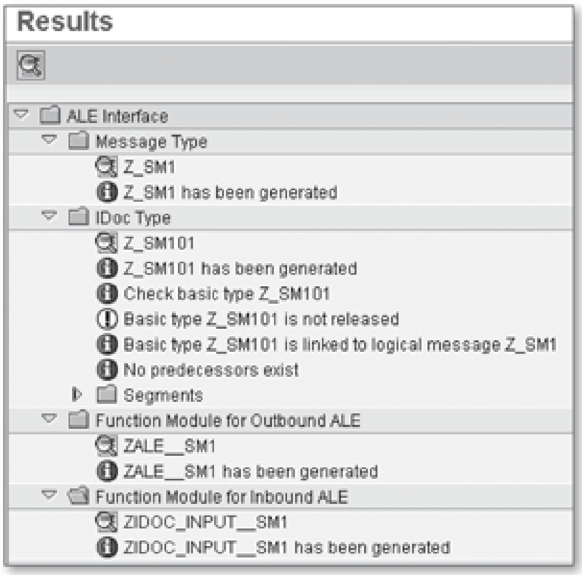
IDoc Interface
- Message Type: Z_SM1
- IDoc Type: Z_SM101
- Packet: ZSM1

ALE Outbound Processing
- Function Module: ZALE__SM1
- Function Group: ZSM1

ALE Inbound Processing
- Function Module: ZIDOC_INPUT__SM1
- Function Group: ZSM1
- ☑ In Update Task
- □ Mass Processing Allowed

Figure 4.123 Entries in Transaction BDFG

The system then uses these specifications to generate everything that is required as illustrated in Figure 4.124. So it can pay off to develop a function module with the required properties around the application's module and to spare the rest of the work.



```
Results
▽ ALE Interface
   ▽ Message Type
         Z_SM1
         Z_SM1 has been generated
   ▽ IDoc Type
         Z_SM101
         Z_SM101 has been generated
         Check basic type Z_SM101
         Basic type Z_SM101 is not released
         Basic type Z_SM101 is linked to logical message Z_SM1
         No predecessors exist
      ▷ Segments
   ▽ Function Module for Outbound ALE
         ZALE__SM1
         ZALE__SM1 has been generated
   ▽ Function Module for Inbound ALE
         ZIDOC_INPUT__SM1
         ZIDOC_INPUT__SM1 has been generated
```

Figure 4.124 Generated Objects in Transaction BDFG

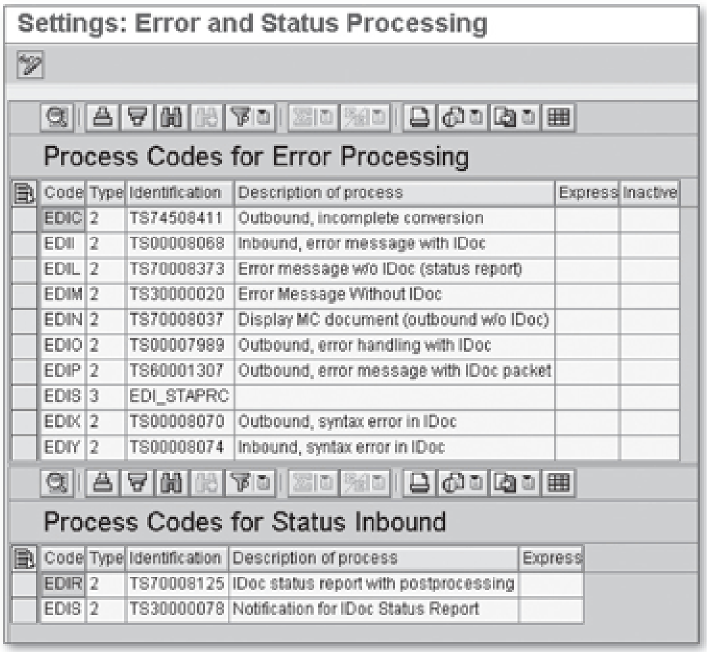
4.4.5 Error Workflow for Custom IDocs

The error processing is basically implemented via workflow tasks, and some of the programming examples already ensure that workflow tasks are started or completed or that the necessary events are triggered.

General error: workflow tasks

In error handling, a distinction is made among errors of the IDoc interface, errors of the external system/EDI subsystem, and errors of the SAP application. SAP provides error tasks for the IDoc interface and the external system, which you can find in Transaction WE40. Figure 4.125 shows them for both the inbound and the outbound processing. Here, `EDIS` is the default error code for the external system, but you can also create your own error tasks. This involves error handling that is independent of the application and is therefore generally valid.

For the errors in the application, a separate error workflow exists for each message type. If you program IDocs yourself, you must create this error workflow completely. This is described here for the `ZSMNACH` message type that was created previously.



Settings: Error and Status Processing

Process Codes for Error Processing

Code	Type	Identification	Description of process	Express	Inactive
EDIC	2	TS74508411	Outbound, incomplete conversion		
EDII	2	TS00008068	Inbound, error message with IDoc		
EDIL	2	TS70008373	Error message w/o IDoc (status report)		
EDIM	2	TS30000020	Error Message Without IDoc		
EDIN	2	TS70008037	Display MC document (outbound w/o IDoc)		
EDIO	2	TS00007989	Outbound, error handling with IDoc		
EDIP	2	TS60001307	Outbound, error message with IDoc packet		
EDIS	3	EDI_STAPRC			
EDIX	2	TS00008070	Outbound, syntax error in IDoc		
EDIY	2	TS00008074	Inbound, syntax error in IDoc		

Process Codes for Status Inbound

Code	Type	Identification	Description of process	Express
EDIR	2	TS70008125	IDoc status report with postprocessing	
EDIS	2	TS30000078	Notification for IDoc Status Report	

Figure 4.125 Technical Error Process Codes

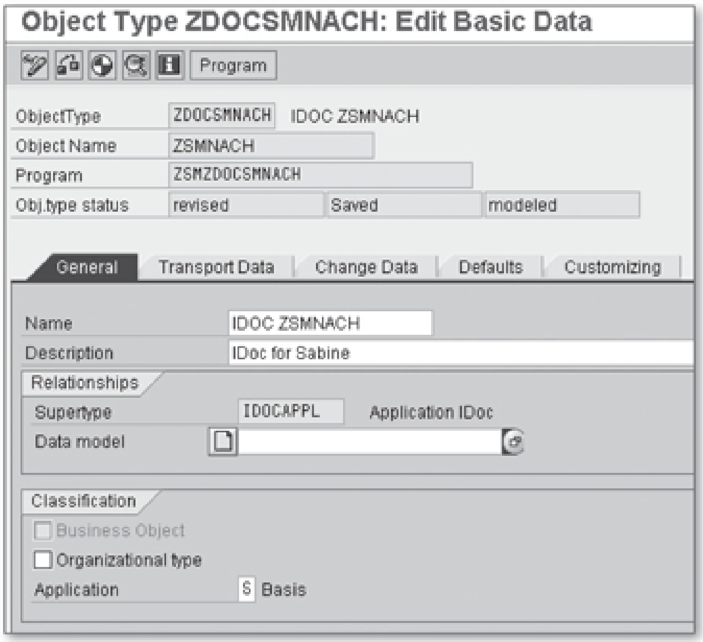
First of all, you must create a custom business object type for the IDoc in Transaction SWO1. This business object type must inherit from the IDOCAPPL type so that all methods required are available. Again, the name of the business object type must be within your namespace. Figure 4.126 shows the business object type of this example.

Context with BOR

Business Object Types for IDocs

Note that there can be two business object types in the context of IDocs. One of them is mandatory and is used for the error workflow. This is the one that must inherit from IDOCAPPL and provides a few, always identical methods.

The second business object type refers to the actual object in the application whose data is part of the IDoc data. This business object type is optional. It's assigned to the message type using Transaction BDA4, and if it's maintained appropriately in the inbound process code, this assignment results in an update of the object links.



Object Type ZDOCSMNACH: Edit Basic Data

ObjectType	ZDOCSMNACH	IDOC ZSMNACH	
Object Name	ZSMNACH		
Program	ZSMZDOCSMNACH		
Obj.type status	revised	Saved	modeled

Name: IDOC ZSMNACH

Description: IDoc for Sabine

Relationships

Supertype: IDOCAPPL Application IDoc

Data model:

Classification

□ Business Object

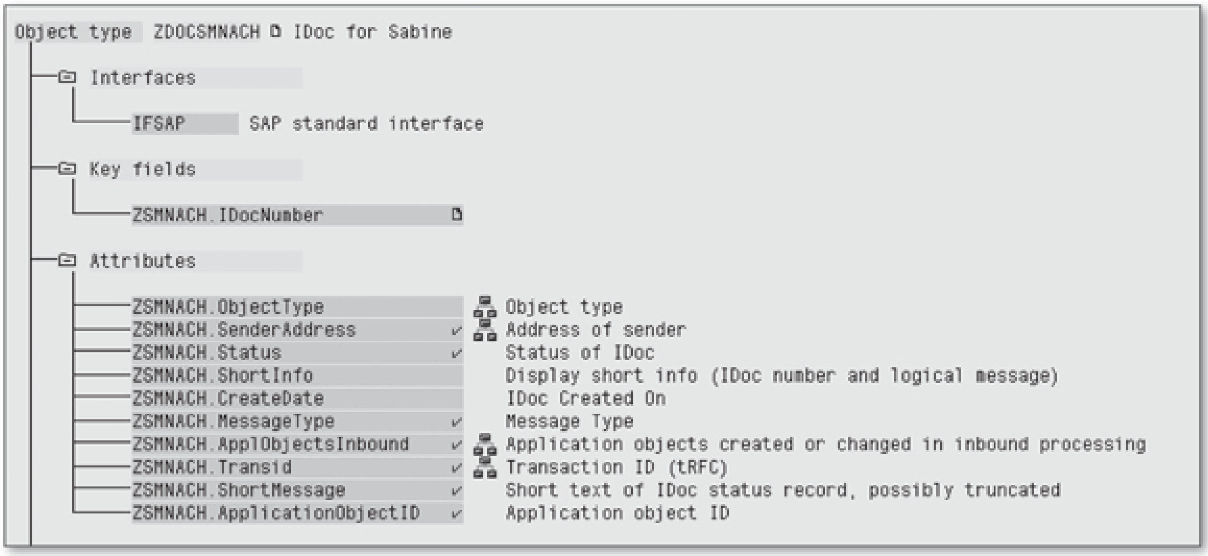
□ Organizational type

Application: S Basis

Figure 4.126 Business Object Type for Error Workflow

The IDoc number is always the key for such IDoc error objects, and additional attributes are the most important attributes from the header table, which is illustrated in Figure 4.127. The red highlighting indicates that the `ZDOCSMNACH` IDoc has inherited from the `IDOCAPPL` supertype.

Attributes

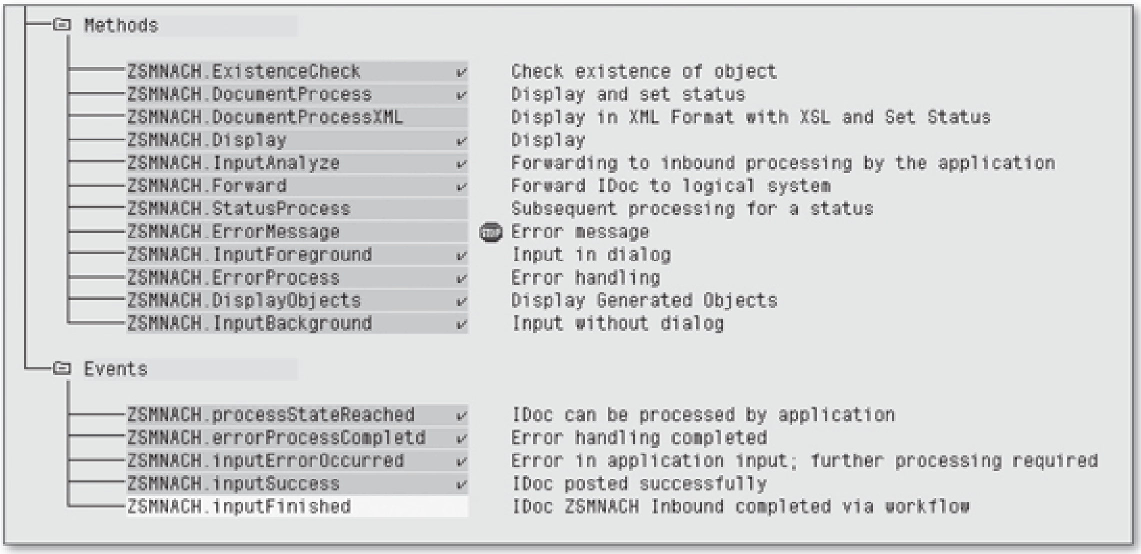


```
Object type  ZDOCSMNACH  IDoc for Sabine
  Interfaces
     IFSAP            SAP standard interface
  Key fields
     ZSMNACH.IDocNumber
  Attributes
     ZSMNACH.ObjectType                  Object type
     ZSMNACH.SenderAddress          ✓    Address of sender
     ZSMNACH.Status                 ✓    Status of IDoc
     ZSMNACH.ShortInfo                   Display short info (IDoc number and logical message)
     ZSMNACH.CreateDate                  IDoc Created On
     ZSMNACH.MessageType            ✓    Message Type
     ZSMNACH.ApplObjectsInbound     ✓    Application objects created or changed in inbound processing
     ZSMNACH.Transid                ✓    Transaction ID (tRFC)
     ZSMNACH.ShortMessage           ✓    Short text of IDoc status record, possibly truncated
     ZSMNACH.ApplicationObjectID    ✓    Application object ID
```

Figure 4.127 Attributes

Methods and events

ZDOCSMNACH also inherits the necessary methods and events from the supertype, whereas the inputFinished event must be overwritten. If you don't have any special requests, you can simply leave the code as it is. Figure 4.128 displays our sample business object type.



```
Methods
  ZSMNACH.ExistenceCheck          ✓  Check existence of object
  ZSMNACH.DocumentProcess         ✓  Display and set status
  ZSMNACH.DocumentProcessXML         Display in XML Format with XSL and Set Status
  ZSMNACH.Display                 ✓  Display
  ZSMNACH.InputAnalyze            ✓  Forwarding to inbound processing by the application
  ZSMNACH.Forward                 ✓  Forward IDoc to logical system
  ZSMNACH.StatusProcess              Subsequent processing for a status
  ZSMNACH.ErrorMessage               Error message
  ZSMNACH.InputForeground         ✓  Input in dialog
  ZSMNACH.ErrorProcess            ✓  Error handling
  ZSMNACH.DisplayObjects          ✓  Display Generated Objects
  ZSMNACH.InputBackground         ✓  Input without dialog
Events
  ZSMNACH.processStateReached     ✓  IDoc can be processed by application
  ZSMNACH.errorProcessCompletd    ✓  Error handling completed
  ZSMNACH.inputErrorOccurred      ✓  Error in application input; further processing required
  ZSMNACH.inputSuccess            ✓  IDoc posted successfully
  ZSMNACH.inputFinished              IDoc ZSMNACH Inbound completed via workflow
```

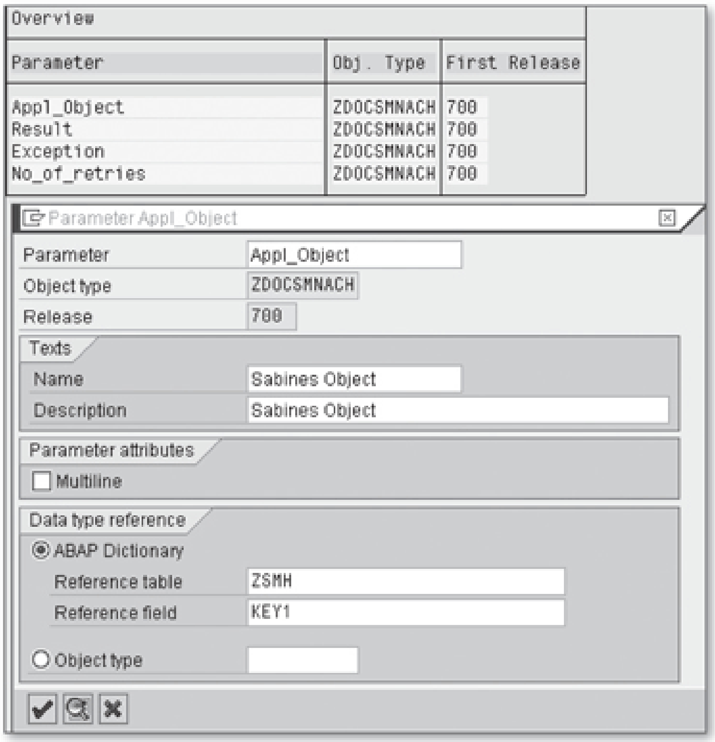
Figure 4.128 Methods and Events

Assigning the fields from the application

However, you must make a minor change to the Appl_Object parameter (see Figure 4.129), that is, refer to the application. This reference must be specified separately for each object; it can't be inherited. In the Data Type Reference tab, you specify the custom Reference Table (here: ZSMH) and the Reference Field (here: KEY1).

Workflow task

Next, you create the actual workflow task. Transaction PFTC_INS is required here (see Figure 4.130). In the input screen, select the task type, and enter a name for your workflow task. The task in this example is of the *TS type* (standard task) and the Abbreviation for the workflow task — in whatever abbreviated form — usually consists of your message type and the term "Error" (here ZSMNAC_Error). In Object Method in the Object Type field, you refer to your business object type (here: ZDOCSMNACH) and in the Method field to the inherited method (here: INPUTFOREGROUND). This is the task that is executed in the foreground in IDoc single processing of an IDoc has run on an error status. Listing 4.28 in Section 4.4.3, Updating an IDoc, shows how this error is triggered and transferred to the workflow runtime.



Overview

Parameter	Obj. Type	First Release
Appl_Object	ZDOCSMNACH	700
Result	ZDOCSMNACH	700
Exception	ZDOCSMNACH	700
No_of_retries	ZDOCSMNACH	700

Parameter Appl_Object

Parameter	Appl_Object
Object type	ZDOCSMNACH
Release	700

Texts

Name	Sabines Object
Description	Sabines Object

Parameter attributes

☐ Multiline

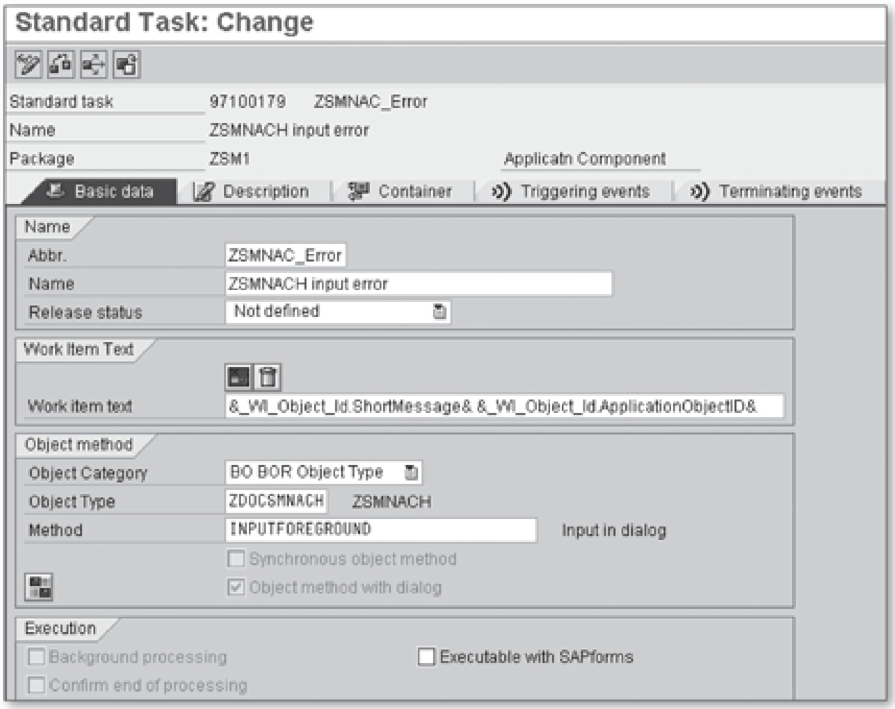
Data type reference

● ABAP Dictionary

Reference table	ZSMH
Reference field	KEY1

○ Object type

Figure 4.129 Adapting the Object Reference



Standard Task: Change

Standard task	97100179 ZSMNAC_Error	
Name	ZSMNACH input error	
Package	ZSM1	Applicatn Component

Name

Abbr.	ZSMNAC_Error
Name	ZSMNACH input error
Release status	Not defined

Work Item Text

Work item text: &_WI_Object_Id.ShortMessage& &_WI_Object_Id.ApplicationObjectID&

Object method

Object Category	BO BOR Object Type	
Object Type	ZDOCSMNACH	ZSMNACH
Method	INPUTFOREGROUND	Input in dialog

☐ Synchronous object method

☑ Object method with dialog

Execution

☐ Background processing

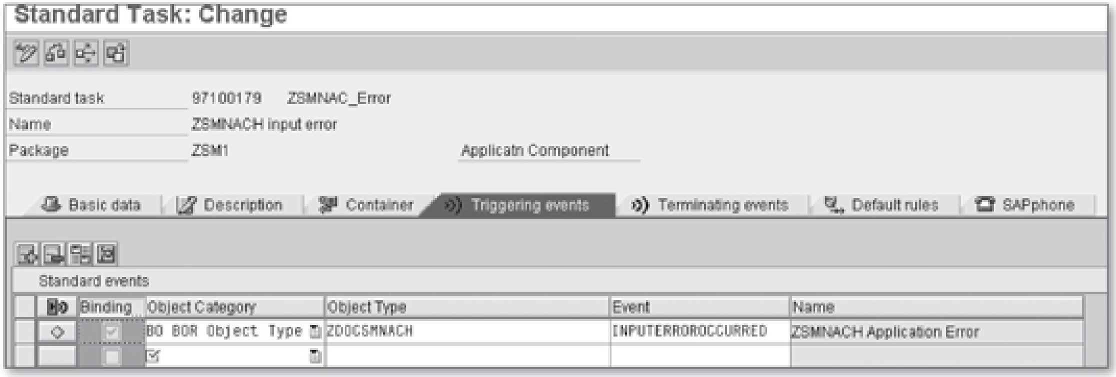
☐ Executable with SAPforms

☐ Confirm end of processing

Figure 4.130 Workflow Task for Error Handling

Assigning the start event

The event that triggers the workflow task is the INPUTERROROCCURRED event that is inherited by IDOCAPPL in the business object type (see Figure 4.131).



Standard Task: Change

Standard task	97100179	ZSMNAC_Error	
Name	ZSMNACH input error		
Package	ZSM1		Applicatn Component

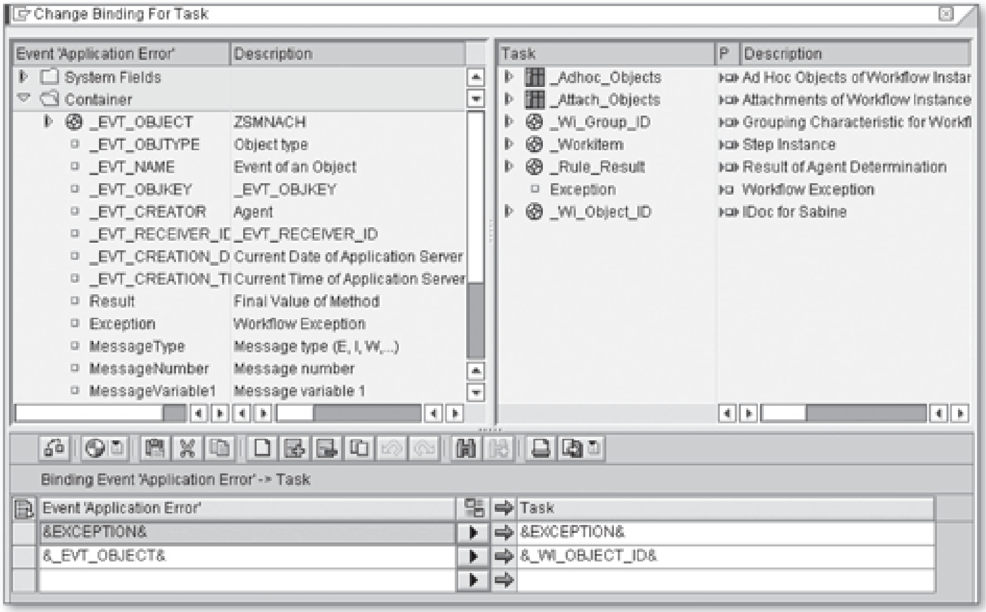
Standard events

Binding	Object Category	Object Type	Event	Name
☑	BO BOR Object Type	ZDOCSMNACH	INPUTERROROCCURRED	ZSMNACH Application Error
□				

Figure 4.131 Start Event

Binding

A workflow task requires a binding that you can use to transfer the data of the object to the data of the task. The system can create this binding automatically. For this purpose, you must click on the binding view. The binding of this example ensures that the workflow is provided with information from the IDoc. Everything is okay if your binding appears as shown in Figure 4.132.



Change Binding For Task

Event 'Application Error'	Description
System Fields	
Container	
_EVT_OBJECT	ZSMNACH
_EVT_OBJTYPE	Object type
_EVT_NAME	Event of an Object
_EVT_OBJKEY	_EVT_OBJKEY
_EVT_CREATOR	Agent
_EVT_RECEIVER_ID	_EVT_RECEIVER_ID
_EVT_CREATION_D	Current Date of Application Server
_EVT_CREATION_TI	Current Time of Application Server
Result	Final Value of Method
Exception	Workflow Exception
MessageType	Message type (E, I, W,...)
MessageNumber	Message number
MessageVariable1	Message variable 1

Task	P	Description
_Adhoc_Objects		Ad Hoc Objects of Workflow Instar
_Attach_Objects		Attachments of Workflow Instance
_WI_Group_ID		Grouping Characteristic for Workfl
_WorkItem		Step Instance
_Rule_Result		Result of Agent Determination
Exception		Workflow Exception
_WI_Object_ID		IDoc for Sabine

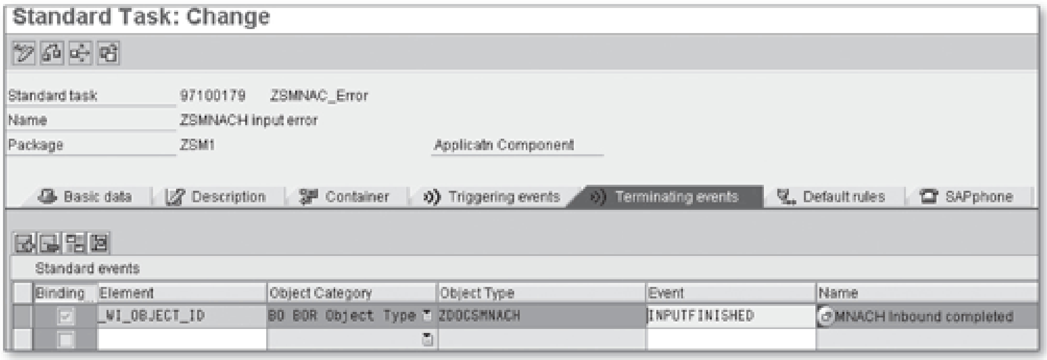
Binding Event 'Application Error' -> Task

Event 'Application Error'		Task
&EXCEPTION&	⇨	&EXCEPTION&
&_EVT_OBJECT&	⇨	&_WI_OBJECT_ID&
	⇨	

Figure 4.132 Binding for Start Event

Terminating event

The successful update of the IDoc is always the end event, irrespective of whether it comes from the error handling of Transaction DB87 or is carried out via a program that is scheduled in the background. `INPUTFINISHED` is the name of the end event, which is triggered in the previously created programs for processing the `ZSMNACH` message type (see Listing 4.28 in Section 4.4.3, Updating an IDoc). As you can see in Figure 4.133, a binding is not available for the Terminating event.



Standard Task: Change

Standard task 97100179 ZSMNAC_Error
Name ZSMNACH input error
Package ZSM1
Applicatn Component

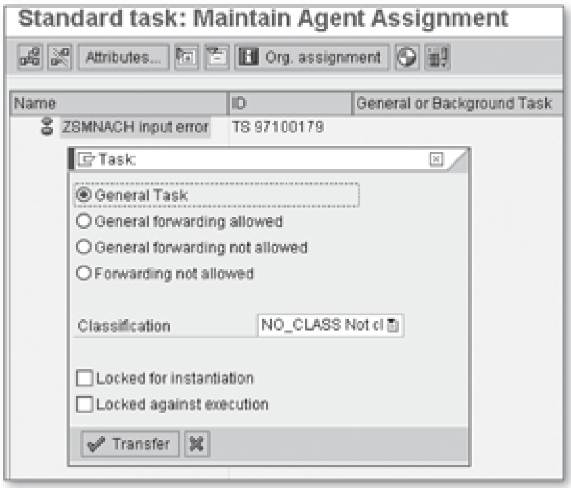
Standard events

Binding	Element	Object Category	Object Type	Event	Name
☑	_WI_OBJECT_ID	BO BOR Object Type	ZDOCSMNACH	INPUTFINISHED	MNACH inbound completed
☐					

Figure 4.133 Terminating Event

Agent assignment

Finally, you must specify who may carry out this task from the error workflow. This depends on how your enterprise handles workflows in general; in the standard version, this involves a General Task that may be performed by any employee (see Figure 4.134). The settings in Transaction WE20 in the inbound partner profile for the agent with postprocessing authorizations determine to whom the workflow is actually sent.



Standard task: Maintain Agent Assignment

Name	ID	General or Background Task
ZSMNACH input error	TS 97100179	

Task:

● General Task
○ General forwarding allowed
○ General forwarding not allowed
○ Forwarding not allowed

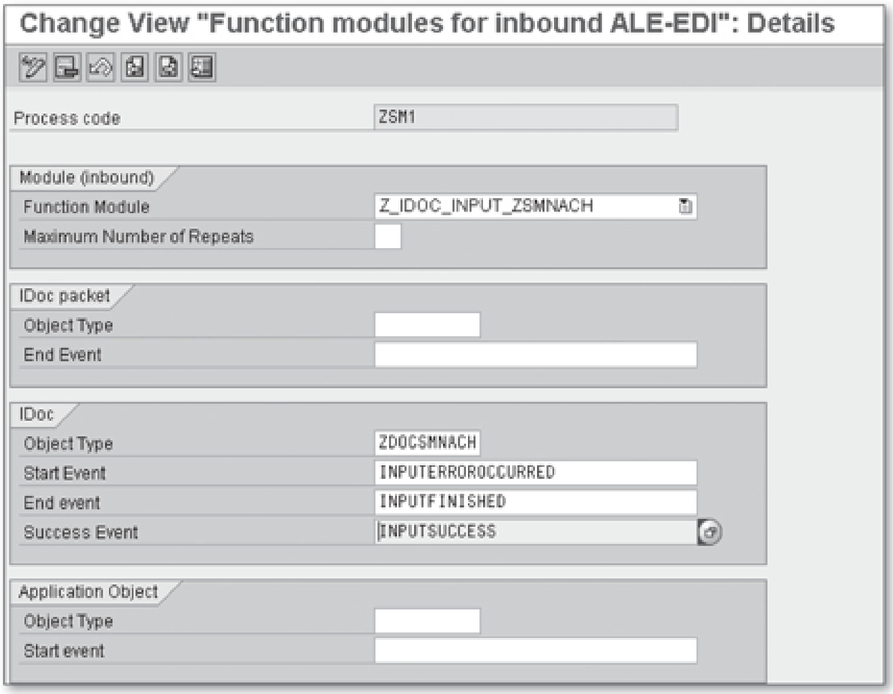
Classification NO_CLASS Not cl

☐ Locked for instantiation
☐ Locked against execution

Figure 4.134 Agent Assignment

Extending the process code by error handling

Call Transaction WE42 again. Up to now, you used this transaction only for linking the process code and the function module. Now you assign your business object type (see Figure 4.135) and specify the events to be processed. The ALE communication layer in combination with your program then ensures that the corresponding events are used (see Listing 4.28 in Section 4.4.3, Updating an IDoc).



Change View "Function modules for inbound ALE-EDI": Details

Process code	ZSM1

Module (inbound)

Function Module	Z_IDOC_INPUT_ZSMNACH
Maximum Number of Repeats	

IDoc packet

Object Type	
End Event	

IDoc

Object Type	ZDOCSMNACH
Start Event	INPUTERROROCCURRED
End event	INPUTFINISHED
Success Event	INPUTSUCCESS

Application Object

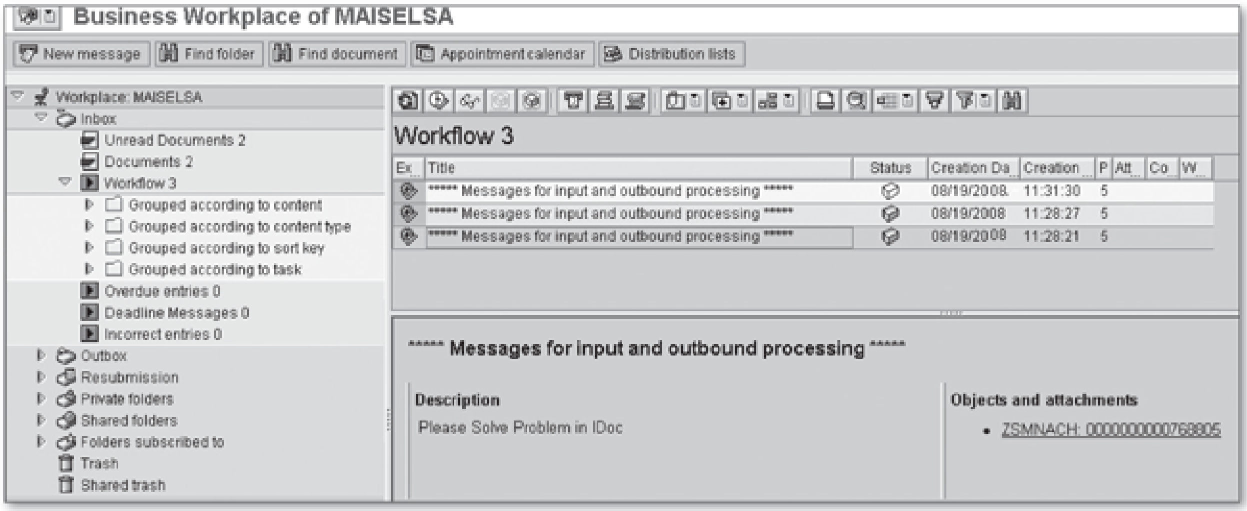
Object Type	
Start event	

Figure 4.135 Assigning the Error Handling to the Process Code

Transaction SBWP, the central workplace, lists the faulty IDocs in the Workflow Tasks in the inbox. In Figure 4.136 under WORKPLACE • INBOX • WORKFLOW, you can find three faulty IDocs for which I was specified as the agent in the partner profile.

Assuming the workflow task

Double-click on the text of the task, and the system navigates to the actual processing menu where you can set, edit, and view deletion flags (see Figure 4.137).



Workflow 3

Ex	Title	Status	Creation Da	Creation	P	Att	Co	W
	***** Messages for input and outbound processing *****		08/19/2008	11:31:30	5			
	***** Messages for input and outbound processing *****		08/19/2008	11:28:27	5			
	***** Messages for input and outbound processing *****		08/19/2008	11:28:21	5			

***** Messages for input and outbound processing *****

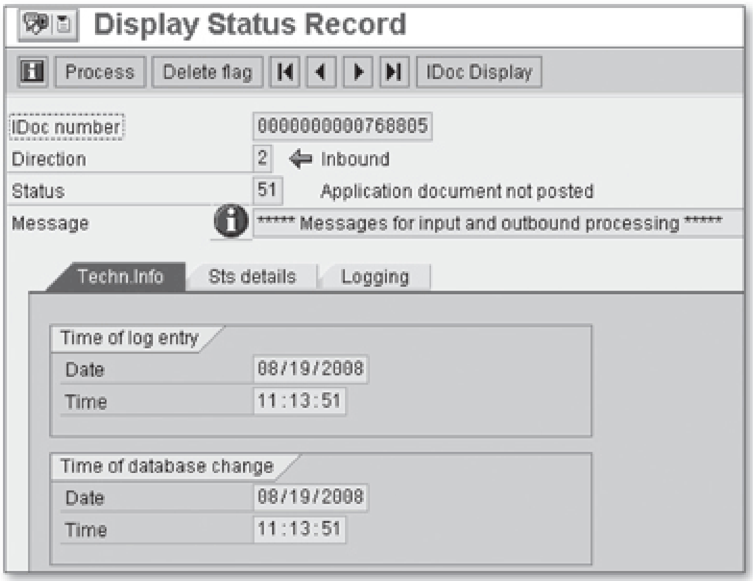
Description

Please Solve Problem in IDoc

Objects and attachments

- ZSMNACH: 0000000000768805

Figure 4.136 Faulty IDocs in the Inbox



IDoc number	0000000000768805
Direction	2 Inbound
Status	51 Application document not posted
Message	***** Messages for input and outbound processing *****

Time of log entry

Date	08/19/2008
Time	11:13:51

Time of database change

Date	08/19/2008
Time	11:13:51

Figure 4.137 View After Assuming the Workflow Task

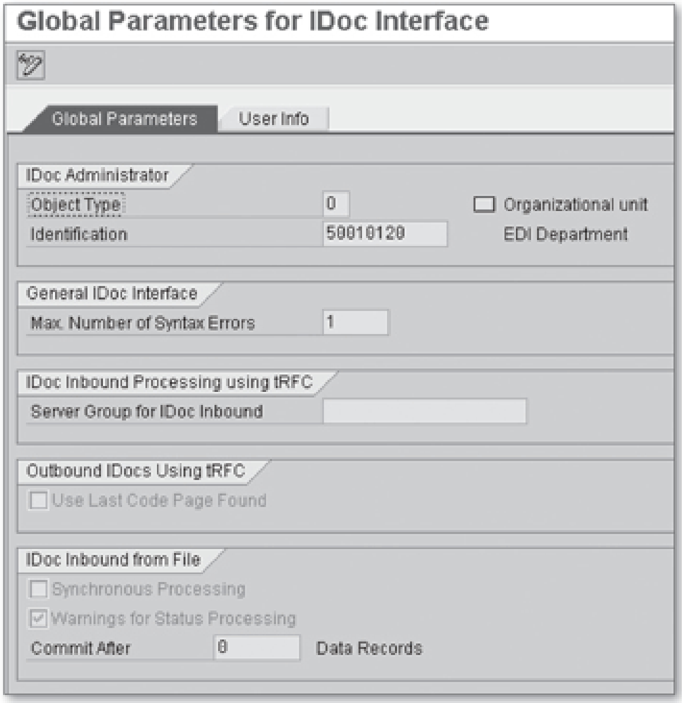
Agent Assignment of Error Tasks

Agent assignment

You assign agents to error tasks in the partner profiles. Initially, you search for the actual message in the individual display; if you can't find anything there, the settings for the entire partner are scanned, and if you can't find anything there either, the error is sent to the IDoc administrator. The system checks whether a specific user is defined or belongs to the organizational unit that is specified there and whether the user is allowed to execute the task. Every user may carry out a task of the *general task* type or a task that was assigned to the user via the organizational management in SAP ERP HCM. The tasks are either specified in the job description or assigned directly. Coordinate the settings in the organizational management with the employees responsible because these settings can also influence other elements in the SAP system.

IDoc administrator

Use Transaction OYEA to maintain the IDoc administrator and some other general settings. Here, you can also enter default settings for frequently requested aspects, such as the maximum number of syntax errors per IDoc (see Figure 4.138).



Global Parameters for IDoc Interface

IDoc Administrator

Object Type: 0 □ Organizational unit

Identification: 50010120 EDI Department

General IDoc Interface

Max. Number of Syntax Errors: 1

IDoc Inbound Processing using tRFC

Server Group for IDoc Inbound:

Outbound IDocs Using tRFC

□ Use Last Code Page Found

IDoc Inbound from File

□ Synchronous Processing

☑ Warnings for Status Processing

Commit After: 0 Data Records

Figure 4.138 Global Settings

4.4.6 Useful Function Modules

Some of the necessary function modules for processing custom-programmed IDocs have already been illustrated in the programming examples. Table 4.3 lists the most important function modules again for your reference. This list also includes those conversion modules that aren't contained in the examples.

Function Module	Function
ALE_MODEL_DETERMINE_IF_TO_SEND	Check whether a specific message may be generated according to the distribution model.
ALE_SERIAL_KEY2CHANNEL	Determine/set object channel for serialization.
CLOI_CONVERSION_PERFORM	Conversion from internal to external formats.
CLOI_PUT_SIGN_IN_FRONT	Put minus sign in front.
CURRENCY_AMOUNT_IDOC_TO_SAP	Convert currency amounts to SAP format.
CURRENCY_AMOUNT_SAP_TO_IDOC	Convert currency amounts to external format.
CURRENCY_CODE_ISO_TO_SAP	Convert currencies from ISO code to SAP format.
CURRENCY_CODE_SAP_TO_ISO	Convert currencies from SAP format to ISO code.
IDOC_DATE_TIME_GET	Get status information from the receiver.
IDOC_REDUCTION_FIELD_REDUCE	Determine active fields in the reduced message type.
IDOC_SERIALIZATION_CHECK	Check timestamp in the serialization field of the IDoc header.
IDOC_SERIAL_POST	Update the serialization table.
IDOC_STATUS_WRITE_TO_DATABASE	Change IDoc status.
LANGUAGE_CODE_ISO_TO_SAP	Convert language code from ISO code to SAP format.
LANGUAGE_CODE_SAP_TO_ISO	Convert language code from SAP format to ISO code.
MASTER_IDOC_DISTRIBUTE	Transfer IDoc to the communication layer.
MMODEL_INT_VALID_GET	Determine in the distribution model who sends which message to whom.
OWN_LOGICAL_SYSTEM_GET	Find the custom logical system name.
RFC_DATA_DETERMINE_FOR_CHECKS	Determine an RFC destination of a partner.
UNIT_OF_MEASURE_ISO_TO_SAP	Convert unit of measure from ISO code to SAP format.
UNIT_OF_MEASURE_SAP_TO_ISO	Convert unit of measure from SAP format to ISO code.

Table 4.3 Function Modules for IDoc Processing

4.5 Summary

Now you've seen all the development tasks for managing company specialties in IDocs. We started with the adaption of standard IDocs with Customizing tasks, because this is the easiest way. The next step is the development of changes to standard IDocs with the help of enhancement techniques. To do so, we reviewed the techniques available for the non-interactive IDoc processing.

At the end we went through the complete process of developing customer IDocs starting with the application tables and ending with the complex process of IDoc error handling.

IDoc communication is essentially asynchronous — the sending system doesn't receive information about how the IDoc is processed in the receiving system. But, this information can be transferred from the receiver to the sender. This chapter will outline the various options available here.

5 Confirmations

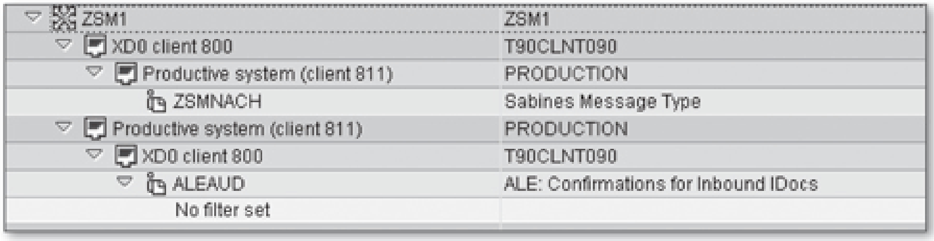
IDoc communication is fundamentally asynchronous. In other words, the sender doesn't receive any information about how the receiver processes the IDoc. However, if required, the receiver can actively send this information back to the sender.

5.1 "ALEAUD" IDocs

ALEAUD IDocs are generally deployed when two SAP systems are in use. This scenario (a sender sends an application IDoc and a receiver sends back an ALEAUD IDoc) is known as *ALE audit*. In SAP NetWeaver Process Integration (SAP NetWeaver PI), the IDoc adapter also processes the ALEAUD IDoc as an *application acknowledgment*.

Distribution model for ALE audit

To use an ALEAUD IDoc, you need a suitable distribution model (see Figure 5.1) that instructs the receiver of the original IDoc to send an ALEAUD IDoc to the sender of the original IDoc.

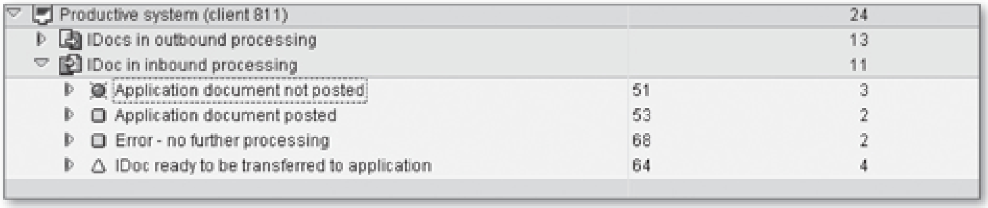


ZSM1	ZSM1
XD0 client 800	T90CLNT090
Productive system (client 811)	PRODUCTION
ZSMNACH	Sabines Message Type
Productive system (client 811)	PRODUCTION
XD0 client 800	T90CLNT090
ALEAUD	ALE: Confirmations for Inbound IDocs
No filter set	

Figure 5.1 Partner Model for the "ALEAUD" IDoc

Initial status values

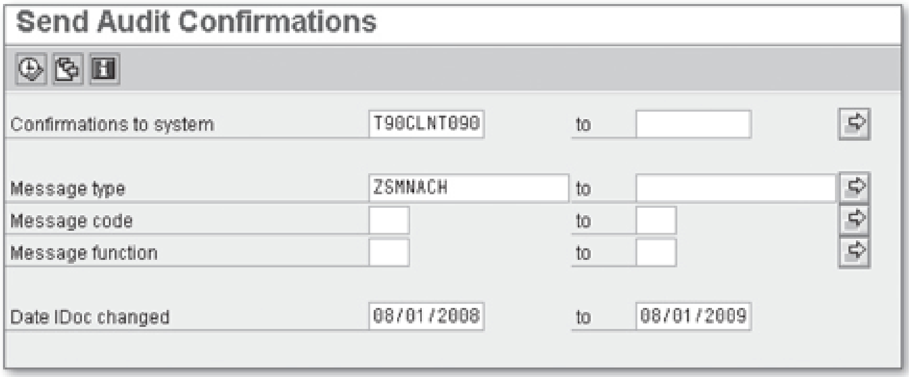
The ALEAUD IDoc confirms the status of the IDoc in the target system. Figure 5.2 shows some possible status values. For example, an IDoc is assigned status *53* if it's posted successfully. In addition, a deletion flag can be set to permanently exclude an IDoc from processing (status *68*) or an IDoc can have an intermediate processing status (e.g., status *64: Waiting* and *51: Incorrect processing*).



Productive system (client 811)		24
IDocs in outbound processing		13
IDoc in inbound processing		11
Application document not posted	51	3
Application document posted	53	2
Error - no further processing	68	2
IDoc ready to be transferred to application	64	4

Figure 5.2 IDoc Status Values in Receiving System

We'll now use the report RDBSTATE to send our confirmations. This report must be scheduled on a regular basis. Figure 5.3 shows the required entries, that is, for which you want ALEAUD IDocs to be generated, for which message type, and for which period.



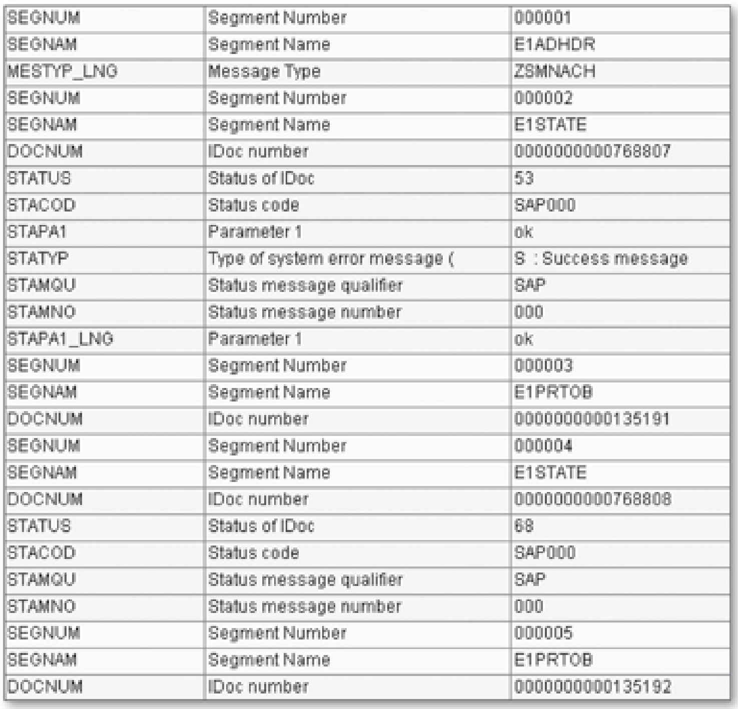
Send Audit Confirmations

Confirmations to system	T90CLNT090	to	
Message type	ZSMNACH	to	
Message code		to	
Message function		to	
Date IDoc changed	08/01/2008	to	08/01/2009

Figure 5.3 "RBDSTATE" Report

"ALEAUD" IDoc

The ALEAUD IDoc contains a header segment, which provides information about the selection made by the user. In our example, it states that we'll send confirmations for the message type ZSMNACH (see Figure 5.4). This is followed by one segment for each IDoc in the selection period, which provides information about the status of the IDoc in the receiving system as well as the IDoc number in both systems.

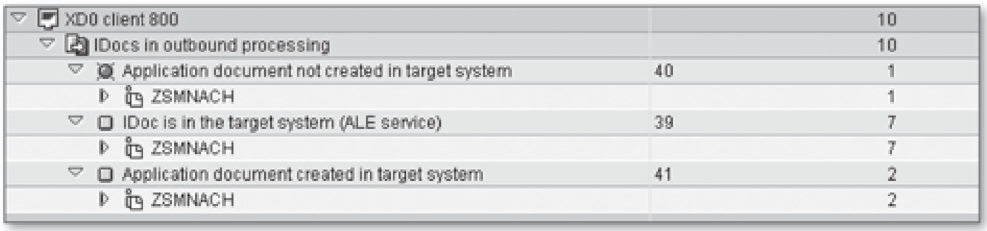


SEGNUM	Segment Number	000001
SEGNAM	Segment Name	E1ADHDR
MESTYP_LNG	Message Type	ZSMNACH
SEGNUM	Segment Number	000002
SEGNAM	Segment Name	E1STATE
DOCNUM	IDoc number	0000000000768807
STATUS	Status of IDoc	53
STACOD	Status code	SAP000
STAPA1	Parameter 1	ok
STATYP	Type of system error message (	S : Success message
STAMQU	Status message qualifier	SAP
STAMNO	Status message number	000
STAPA1_LNG	Parameter 1	ok
SEGNUM	Segment Number	000003
SEGNAM	Segment Name	E1PRTOB
DOCNUM	IDoc number	0000000000135191
SEGNUM	Segment Number	000004
SEGNAM	Segment Name	E1STATE
DOCNUM	IDoc number	0000000000768808
STATUS	Status of IDoc	68
STACOD	Status code	SAP000
STAMQU	Status message qualifier	SAP
STAMNO	Status message number	000
SEGNUM	Segment Number	000005
SEGNAM	Segment Name	E1PRTOB
DOCNUM	IDoc number	0000000000135192

Figure 5.4 Sample "ALEAUD" IDoc

Status values in sending system

The result shown in the sending system comprises three new status values (*39*, *40*, and *41*), as shown in Figure 5.5. Status value *40: Application document not created in target system* is red because even if the sender no longer needs to take action here, it should be clear that the partner didn't post these IDocs but assigned them status *68: Error - no further processing* instead. The following two status values are green: *41*, which, in the sending system, means that the receiving system successfully processed the IDoc (and therefore obtained status *53*), and status *39*, which means that this IDoc is still in an intermediate stage. All IDocs that have a value other than *53* or *68* in a partner system are assigned status *39* because it's assumed that the partner will continue to process all IDocs that don't have status *53* or *68* until they obtain status *53* or *68*.



XD0 client 800		10
IDocs in outbound processing		10
Application document not created in target system	40	1
ZSMNACH		1
IDoc is in the target system (ALE service)	39	7
ZSMNACH		7
Application document created in target system	41	2
ZSMNACH		2

Figure 5.5 Status Values in Sending System After ALE Audit

IDocs with audit messages

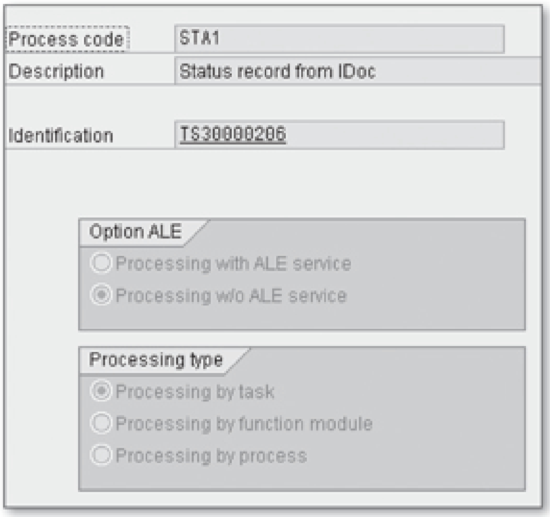
If the status of an IDoc changes in a partner system, and if the report RBDSTATE is executed again, a new entry is sent within an ALEAUD IDoc.

5.2 "STATUS" IDocs

When ALEAUD IDocs are used to send confirmations, all three status values result in the sender no longer being able to process or resend the IDoc. Here, it's generally assumed that the receiver will resolve any errors that occur. However, in the case of EDI subsystems, in particular, you may see additional status values in the EDI subsystem, or you may want to resend an IDoc because an error occurred in the EDI subsystem. You're already familiar with status processing from Chapter 3, Section 3.3, Processing Status Files. Here, an IDoc is used to create a new status value for the original IDoc.

Processing by workflow task

Figure 5.6 shows process code STA1, which is used to post the STATUS IDoc. As you can see, processing by (workflow) task is specified under Processing Type if your system has not changed since it was delivered by SAP. For this reason, you should only use STATUS IDocs if you really need the other status value. Workflow processing is more performance-intensive than processing by function module. Where possible, you should work with the ALEAUD scenario. You should also only work with STATUS IDocs if you want to process IDocs again.



Process code: STA1
Description: Status record from IDoc

Identification: TS30000206

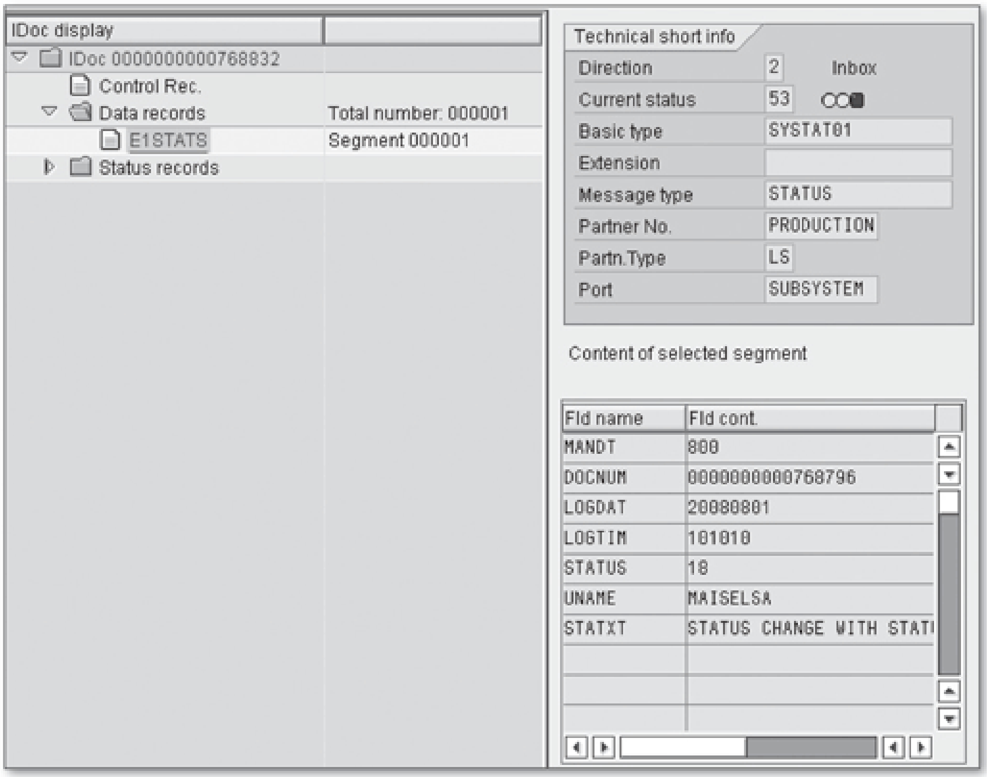
Option ALE
○ Processing with ALE service
● Processing w/o ALE service

Processing type
● Processing by task
○ Processing by function module
○ Processing by process

Figure 5.6 Processing "STATUS" IDocs

"STATUS" message type

The STATUS IDoc doesn't transfer the status of the IDoc in the receiving system. Instead, you specify the new status that you want the IDoc to obtain in the sending system, along with a suitable piece of text. In our example, this text is always hard-coded in the IDoc. You can also reference a message in a message class, but this must exist in the original sending system. The latter option facilitates the use of more user-friendly, language-specific messages. Figure 5.7 shows the main fields in the STATUS IDoc, the reference to the IDoc that is to receive a new status value, the new status value itself, and the comment text.



IDoc display

- IDoc 0000000000768832
  - Control Rec.
  - Data records — Total number: 000001
    - E1STATS — Segment 000001
  - Status records

Technical short info

Direction	2	Inbox
Current status	53	○○●
Basic type	SYSTAT01	
Extension		
Message type	STATUS	
Partner No.	PRODUCTION	
Partn.Type	LS	
Port	SUBSYSTEM	

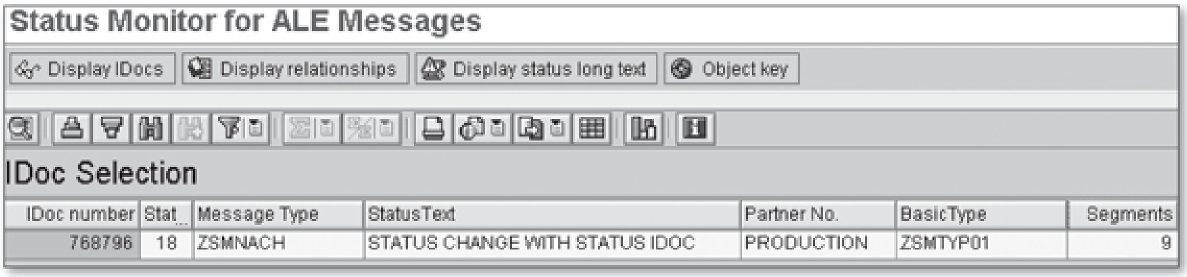
Content of selected segment

Fld name	Fld cont.
MANDT	800
DOCNUM	0000000000768796
LOGDAT	20080801
LOGTIM	101010
STATUS	18
UNAME	MAISELSA
STATXT	STATUS CHANGE WITH STAT

Figure 5.7 Sample "STATUS" IDoc

Implemented status

Figure 5.8 shows the result of posting the SYSTAT IDoc. In our case, status value *18, EDI subsystem initialization OK* is assigned. Similarly, this status doesn't permit further processing.



Status Monitor for ALE Messages

IDoc Selection

IDoc number	Stat...	Message Type	StatusText	Partner No.	BasicType	Segments
768796	18	ZSMNACH	STATUS CHANGE WITH STATUS IDOC	PRODUCTION	ZSMTYP01	9

Figure 5.8 Modified IDoc in the Sending System

5.3 Summary

You've seen now the different possibilities for bringing information about the IDoc back to the original sender. This helps you to overcome one of the disadvantages of asynchronous communication: not knowing what happened at your partner's site. Also, with the help of the STATUS IDoc or the Status File, you are able to resend an IDoc that was already sent successfully before.

In the next chapter, we'll review the more complicated processes, where the sequence of IDocs becomes an important factor, the serialization methods for IDocs.

Even though normal IDoc communication makes provision for the use of timestamps to sort system IDocs in the correct sequence before they are dispatched or posted, IDocs may "queue-jump" during parallel processing and when terminations occur. If you don't want this to happen, you must ensure that IDocs are always processed in exactly the correct sequence.

6 Serializing IDocs

In some situations, it's necessary to adhere to the exact posting sequence for IDocs. One such situation involves goods movements. If queue-jumping occurs among IDocs that have goods movements for the same material in the same storage bin, this will result in numerous different postings, which are not desirable from a financial audit or system load perspective. So, SAP software supports several types of IDoc serialization:

Serialization options

- Serialization using serialization groups
- Serialization using timestamps
- Serialization using business objects
- Serialization using qRFC

Not all serialization types are always possible. Also, they can have various different effects. In the next section, we'll describe how to use each serialization type, how to find the message types supported, and how to configure the entire process.

6.1 Serialization Using Groups

When using groups for serialization purposes, you assume that more than one IDoc is being dispatched for a particular process, so numerous different message types are used. In the case of a material master, for example, you also have the option of making a classification when you create a material. Classifications are then sent with the message type `CLFMAS`, while materials are sent with the already familiar message type `MATMAS`.

Processing sequence

A material master is required to successfully process the `CLFMAS` IDoc. If a material is created, posting problems may occur if the `CLFMAS` IDoc queue-jumps and overtakes the `MATMAS` IDoc. Even though the `CLFMAS` IDoc can be successfully posted if the IDoc is restarted after the MATMAS IDoc arrived and was posted successfully, it's better, in terms of system performance, if such queue-jumping is prevented. For this reason, you have the option of grouping the message types into a serialization group in the sending and receiving systems as well as specifying the required processing sequence. A prerequisite for this procedure is the use of a change pointer to evaluate the relevant IDocs, so this procedure concerns master data.

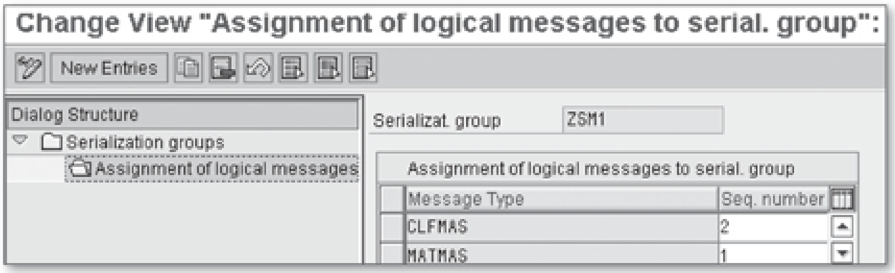
Evaluation report "RBDSER01"

Instead of using the report `RBDMIDOC` to evaluate the change pointers or Transaction BD21, which evaluates all of the change pointers for a message type, you use the report `RBDSER01`, which selects all of the change pointers for a group of message types and ensures that all IDocs with the first message type are created first, followed by all IDocs with the second message type, and so on. If individual IDocs contain errors, they are ignored and added later, in the usual way, via error handling.

Serialization groups

To work with serialization groups, it's important that processing in both the sending and receiving systems is set to Wait for Background Program in the partner profiles in Transaction WE20.

The first step when working with serialization groups is to use Transaction BD44 to create the serialization group in the sending and receiving systems. Figure 6.1 shows the settings for our classification example. The material masters (`MATMAS`) will be processed first, followed by the classification assignments (`CLFMAS`).



Change View "Assignment of logical messages to serial. group":

Serializat. group: ZSM1

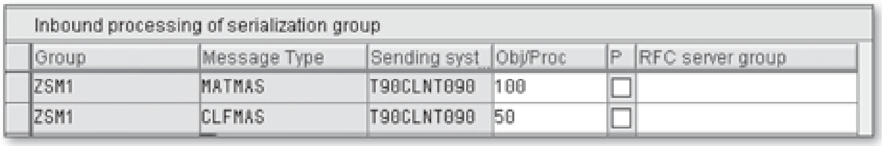
Assignment of logical messages to serial. group

Message Type	Seq. number
CLFMAS	2
MATMAS	1

Figure 6.1 Creating a Serialization Group

Parallel processing

These settings must be configured in both the sending and receiving systems. In the receiving system, you can also specify additional information about parallel processing in Column P. In our example (see Figure 6.2), however, the flag for parallel processing was not set because you would then need to specify a server group. It's best to leave such specifications to your system administrator.



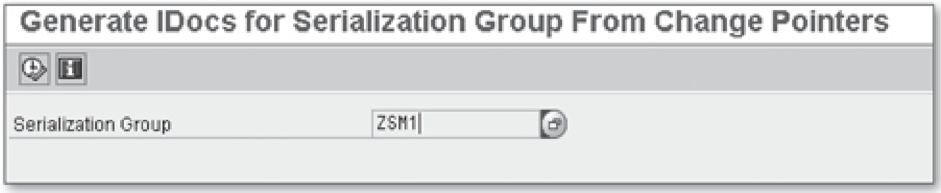
Inbound processing of serialization group

Group	Message Type	Sending syst	Obj/Proc	P	RFC server group
ZSM1	MATMAS	T90CLNT090	100	□	
ZSM1	CLFMAS	T90CLNT090	50	□	

Figure 6.2 Processing in the Receiving System

Creating IDocs for a group

In the sending system, you then use the report `RBDSER01` or Transaction BD40 to create IDocs for a particular group. Here, you only need to specify the serialization group (see Figure 6.3). This is also the case for the reports described next.

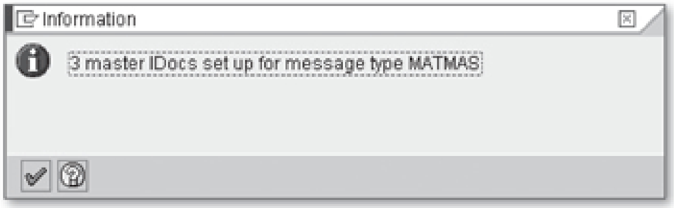


Generate IDocs for Serialization Group From Change Pointers

Serialization Group: ZSM1

Figure 6.3 Creating IDocs for a Group

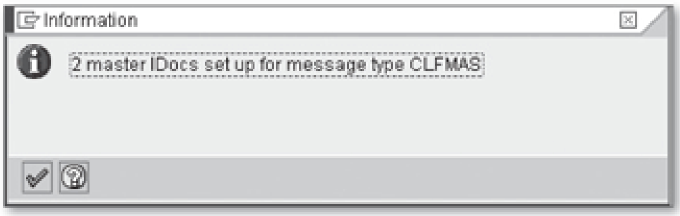
First, the system generates all of the `MATMAS` IDocs (see Figure 6.4).



3 master IDocs set up for message type MATMAS

Figure 6.4 First Message Type Created

Then, it generates the classification IDocs (see Figure 6.5). In both cases, the customer distribution model is relevant for determining which partner will receive the IDocs. For each change, you can also create several IDocs for various different partners, if the use of multiple receivers is permitted.

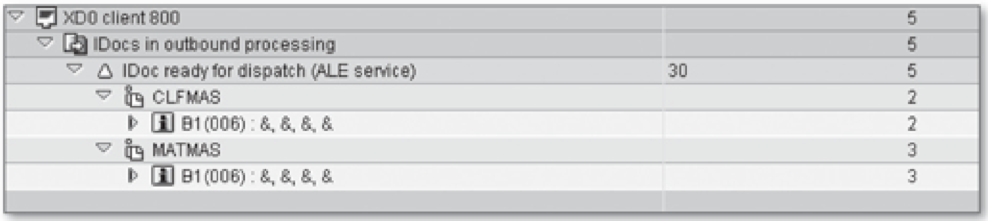


2 master IDocs set up for message type CLFMAS

Figure 6.5 Second Message Type Created

Serialization group in outbound processing

In the sending system, all of these new IDocs obtain status *30: Ready for dispatch*, as shown in Figure 6.6. To retain the sequence, it's important that these IDocs are not included in conventional dispatch jobs (report `RSEOUT00`). The variants here must be selected so that these messages are excluded.

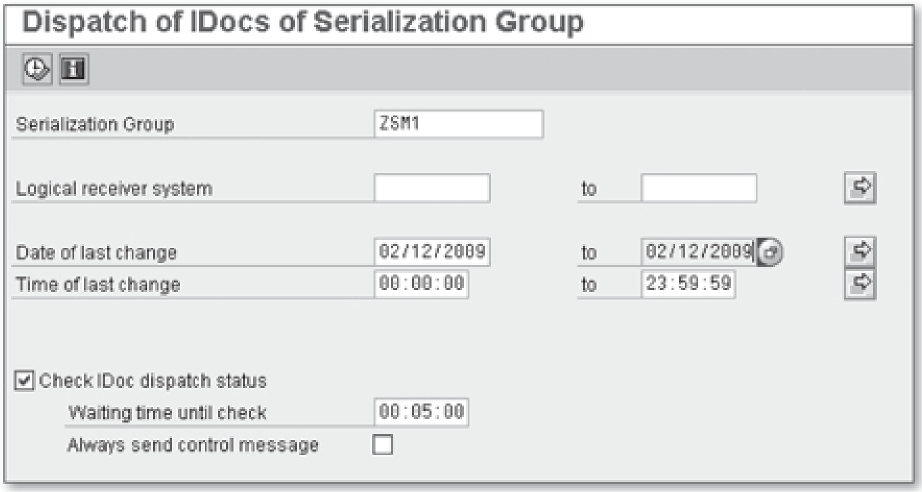


XD0 client 800		5
IDocs in outbound processing		5
IDoc ready for dispatch (ALE service)	30	5
CLFMAS		2
B1(006) : &, &, &, &		2
MATMAS		3
B1(006) : &, &, &, &		3

Figure 6.6 Creating IDocs But Not Dispatching Them Yet

Dispatching the group

You can now start the report `RBDSER02` or Transaction BD41 to dispatch the IDocs. You can use the flag Always Send Control Message to determine whether or not to send the control message directly (see Figure 6.7).



Dispatch of IDocs of Serialization Group

Serialization Group	ZSM1		
Logical receiver system		to	
Date of last change	02/12/2009	to	02/12/2009
Time of last change	00:00:00	to	23:59:59

☑ Check IDoc dispatch status

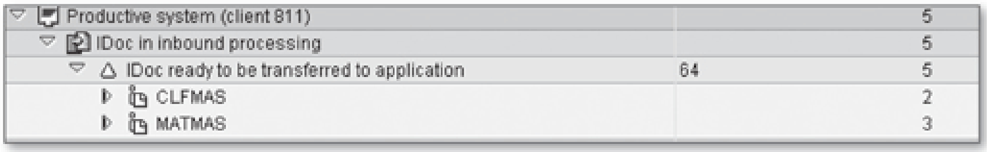
Waiting time until check 00:05:00

Always send control message □

Figure 6.7 Dispatching IDocs

Inbound processing in partner system

When these IDocs arrive in the target system, their status is *64: IDoc ready to be transferred to application*. Our example shows a large number of unprocessed IDocs because we selected a long period of time in which the system would collect change pointers (see Figure 6.8).

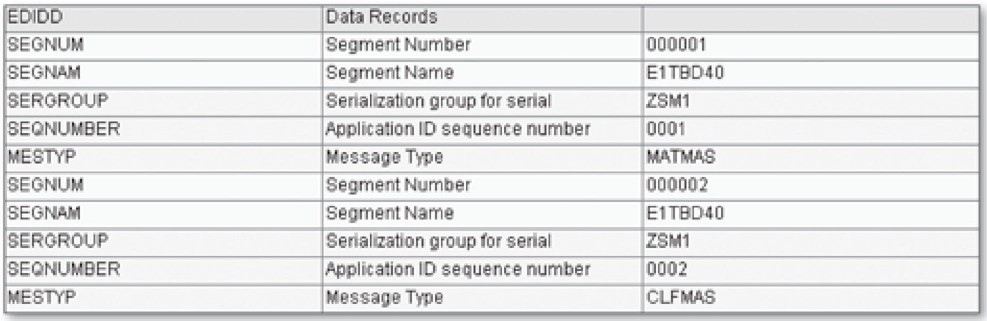


Productive system (client 811)		5
IDoc in inbound processing		5
IDoc ready to be transferred to application	64	5
CLFMAS		2
MATMAS		3

Figure 6.8 Unprocessed IDocs in Target System

To post the IDocs in a useful manner, you can start processing when the last IDoc in this group and change pointer evaluation has arrived. You, as the sender, can use the report `RBDSER03` or Transaction BD42 to instruct the receiver that posting can now commence. This information, which has its own message type `SERDAT`, is then transferred to the sender if you haven't already used the report `RBDSER02` to generate a control message.

The serialization information is transferred within the `SERDAT` IDoc. You get a line item segment for each message type in the serialization group (see Figure 6.9).



EDIDD	Data Records	
SEGNUM	Segment Number	000001
SEGNAM	Segment Name	E1TBD40
SERGROUP	Serialization group for serial	ZSM1
SEQNUMBER	Application ID sequence number	0001
MESTYP	Message Type	MATMAS
SEGNUM	Segment Number	000002
SEGNAM	Segment Name	E1TBD40
SERGROUP	Serialization group for serial	ZSM1
SEQNUMBER	Application ID sequence number	0002
MESTYP	Message Type	CLFMAS

Figure 6.9 SERDAT IDoc, Which Starts Posting

You can also start posting without sending a `SERDAT` IDoc. To do this, use the report `RBDSER04` or Transaction BD43 in the receiving system.

6.2 Serialization Using Timestamps

Serialization using timestamps, also known as "serialization at IDoc level," ensures that if two IDocs queue-jump and contain different data for the same object, the older IDoc that queue-jumped can no longer be posted.

Serialization with table "BDSER"

For this purpose, the last timestamp posted is entered for each object in the table BDSER. An IDoc for the same object checks its timestamp from the SERIAL control record field against the timestamp in the above mentioned table BDSER, and it's only posted if its timestamp is later. Otherwise, the IDoc obtains status *51*, and it can be flagged for deletion. This only concerns serialization in the receiving system. IDocs created using the generated BAPI interface don't support this function because the generation program doesn't implement the corresponding function modules.

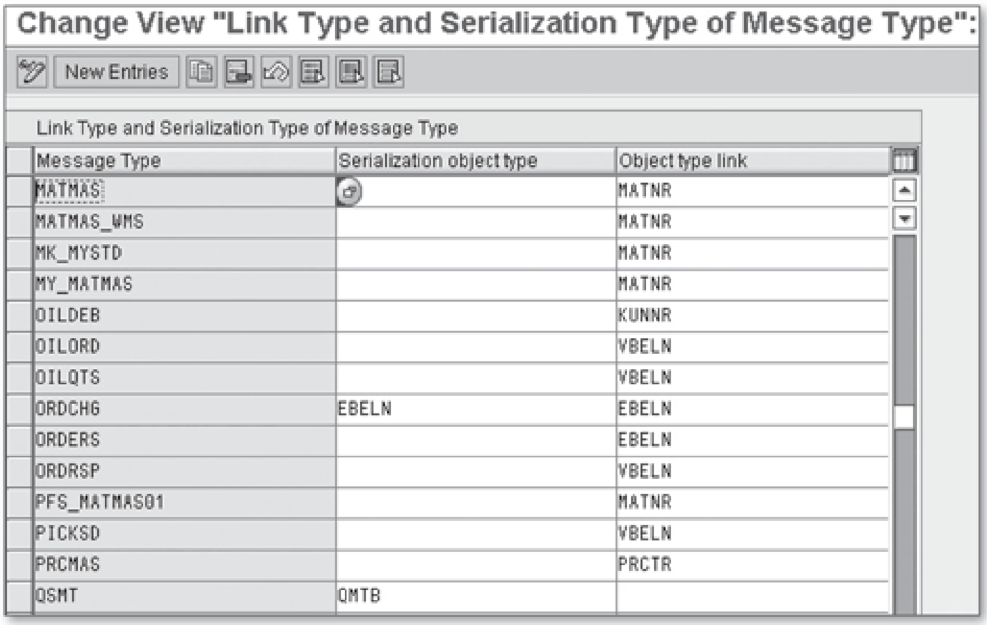
Customizing for timestamp serialization

In Transaction BD95, you assign IDoc fields to certain fields in the database (see Figure 4.9 in Chapter 4, Section 4.1.3, Filtering Segments). These fields are also the basis for determining object assignment for timestamp serialization.

Transaction BD57, shown in Figure 6.10, contains the message types delivered by SAP, which support serialization using timestamps. This transaction also shows which field is important for the sequence. One example of such a field is the document number EBELN in the message type ORDCHG.

Timestamp in table "BDSER"

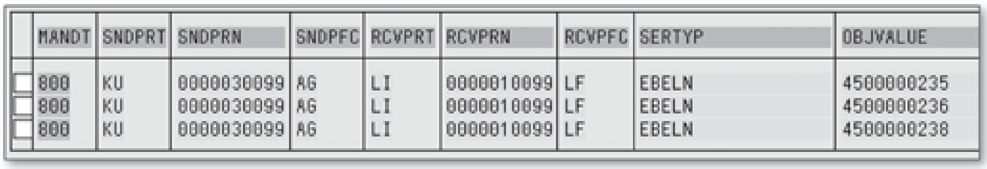
Table BDSER (see Figure 6.11) contains the associated entries with the IDocs that were posted for an object as well as the timestamp indicating when posting took place. The SERIAL field here contains the timestamp from the SERIAL field in the control record for the last successfully posted IDoc. IDocs that are based on the same object but have an earlier timestamp must not be posted.


Change View "Link Type and Serialization Type of Message Type":

Link Type and Serialization Type of Message Type

Message Type	Serialization object type	Object type link
MATMAS		MATNR
MATMAS_WMS		MATNR
MK_MYSTD		MATNR
MY_MATMAS		MATNR
OILDEB		KUNNR
OILORD		VBELN
OILQTS		VBELN
ORDCHG	EBELN	EBELN
ORDERS		EBELN
ORDRSP		VBELN
PFS_MATMAS01		MATNR
PICKSD		VBELN
PRCMAS		PRCTR
QSMT	QMTB	

Figure 6.10 IDocs That Support Timestamp Serialization



	MANDT	SNDPRT	SNDPRN	SNDPFC	RCVPRT	RCVPRN	RCVPFC	SERTYP	OBJVALUE
□	800	KU	0000030099	AG	LI	0000010099	LF	EBELN	4500000235
□	800	KU	0000030099	AG	LI	0000010099	LF	EBELN	4500000236
□	800	KU	0000030099	AG	LI	0000010099	LF	EBELN	4500000238

Figure 6.11 Table "BDSER"

You can use the report `RBDSRCLR` to remove obsolete timestamps from the Table BDSER.

Function Modules for Serialization

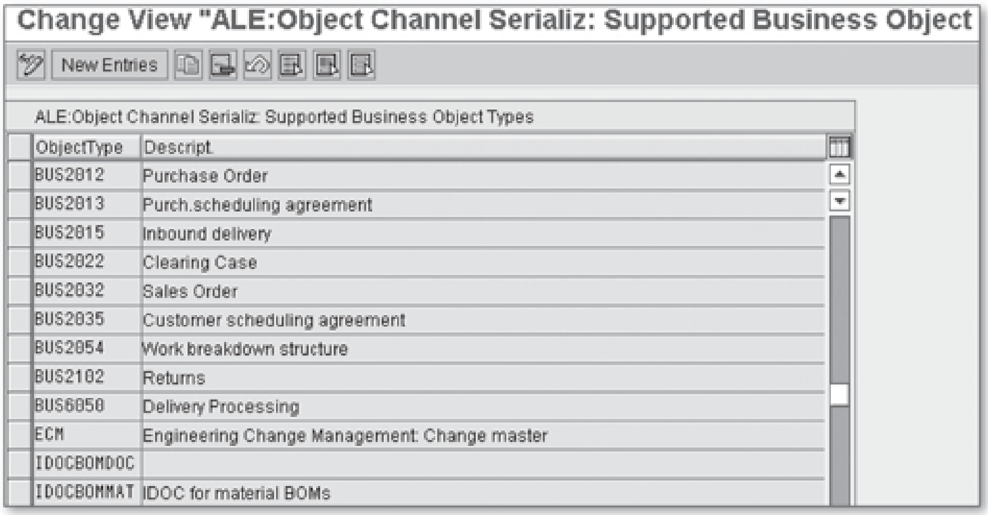
SAP delivers two function modules for timestamp serialization: `IDOC_SERIALIZATION_CHECK` for checking timestamps and `IDOC_SERIAL_POST` for updating the serialization table after a new IDoc has been successfully posted. If you want to use this method to serialize IDocs, go to Transaction BD57, maintain the field to be used, and then use the enhancement technology to implement the two function modules in the inbound function module.

6.3 Serialization Using Business Objects

For serialization using business objects, one common channel number is assigned to all IDocs for the same object type. A *channel number* is a message attribute generated using the function module `ALE_SERIAL_KEY2CHANNEL` or assigned in the application program.

Object channel serialization

This type of serialization is sometimes also known as *object channel serialization*. Here, serialization occurs at the object type level (e.g., based on all BOM IDocs). The business object type for this purpose is `IDOCBOMMAT`. Because SAP doesn't deliver any object channel serialization for material masters, we'll use BOMs in our example here. Figure 6.12 shows Transaction BD105, which contains all objects for which SAP delivers serialization using business objects. You can maintain this table yourself, but you must use exits to build the associated implementation into the processing routines.



Change View "ALE:Object Channel Serializ: Supported Business Object

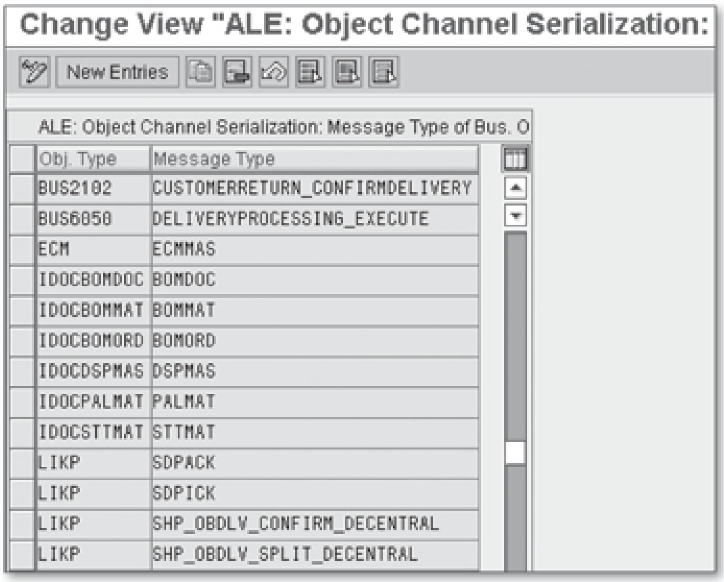
ALE:Object Channel Serializ: Supported Business Object Types

ObjectType	Descript.
BUS2012	Purchase Order
BUS2013	Purch.scheduling agreement
BUS2015	Inbound delivery
BUS2022	Clearing Case
BUS2032	Sales Order
BUS2035	Customer scheduling agreement
BUS2054	Work breakdown structure
BUS2102	Returns
BUS6050	Delivery Processing
ECM	Engineering Change Management: Change master
IDOCBOMDOC	
IDOCBOMMAT	IDOC for material BOMs

Figure 6.12 Objects Connected to Object Channel Serialization

Assignment to message type

A message type that can be serialized is then assigned to each of these objects. You use Transaction BD104 to access this assignment. For our sample BOM, the message type is `BOMMAT`, as shown in Figure 6.13.



Change View "ALE: Object Channel Serialization:

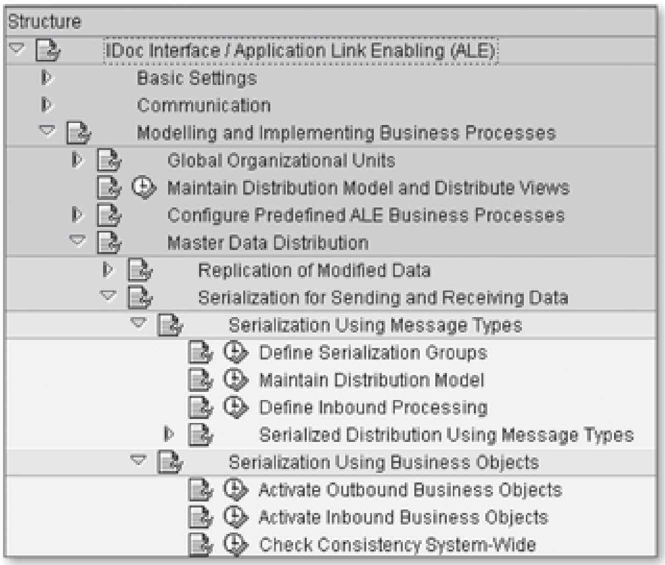
ALE: Object Channel Serialization: Message Type of Bus. O

Obj. Type	Message Type
BUS2102	CUSTOMERRETURN_CONFIRMDELIVERY
BUS6050	DELIVERYPROCESSING_EXECUTE
ECM	ECMMAS
IDOCBOMDOC	BOMDOC
IDOCBOMMAT	BOMMAT
IDOCBOMORD	BOMORD
IDOCDSPMAS	DSPMAS
IDOCPALMAT	PALMAT
IDOCSTTMAT	STTMAT
LIKP	SDPACK
LIKP	SDPICK
LIKP	SHP_OBDLV_CONFIRM_DECENTRAL
LIKP	SHP_OBDLV_SPLIT_DECENTRAL

Figure 6.13 Objects and Their IDocs

Activating serialization

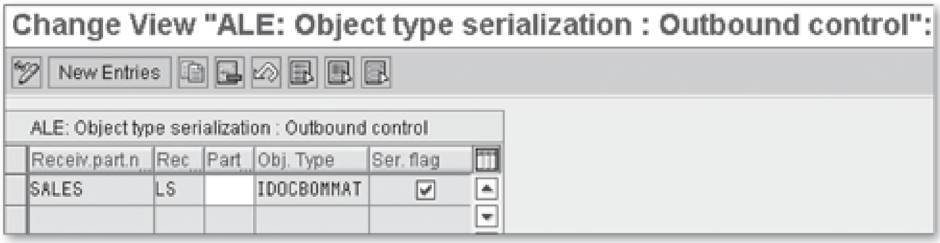
The functions for activating object channel serialization are only available in ALE Customizing. Surprisingly, they are located under the Master Data Distribution menu option even though serialization is wholly independent of the type of data used and has nothing to do with the Shared Master Data tool (SMD). Figure 6.14 shows an extract from ALE Customizing.



Structure
IDoc Interface / Application Link Enabling (ALE)
Basic Settings
Communication
Modelling and Implementing Business Processes
Global Organizational Units
Maintain Distribution Model and Distribute Views
Configure Predefined ALE Business Processes
Master Data Distribution
Replication of Modified Data
Serialization for Sending and Receiving Data
Serialization Using Message Types
Define Serialization Groups
Maintain Distribution Model
Define Inbound Processing
Serialized Distribution Using Message Types
Serialization Using Business Objects
Activate Outbound Business Objects
Activate Inbound Business Objects
Check Consistency System-Wide

Figure 6.14 Path to Serialization Settings in ALE Customizing

To activate serialization, the serialization flag SFLAG is set in the sending system for the receiver (here: SALES) and the business object `IDOCBOMMAT`, as shown in Figure 6.15.



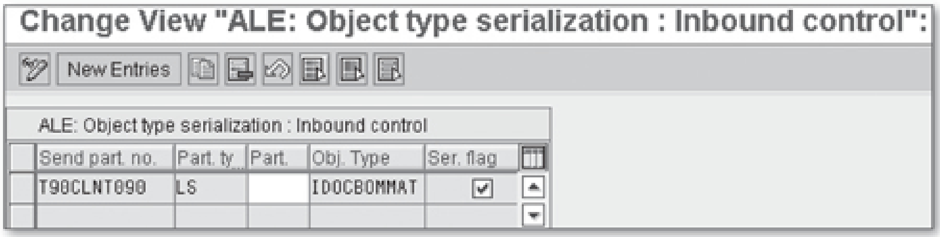
Change View "ALE: Object type serialization : Outbound control":

ALE: Object type serialization : Outbound control

Receiv.part.n	Rec	Part	Obj. Type	Ser. flag
SALES	LS		IDOCBOMMAT	☑

Figure 6.15 Settings in Sending System

The settings must be configured in the same way in the receiving system. In our example, the sender is `T90CLNT090` (see Figure 6.16).



Change View "ALE: Object type serialization : Inbound control":

ALE: Object type serialization : Inbound control

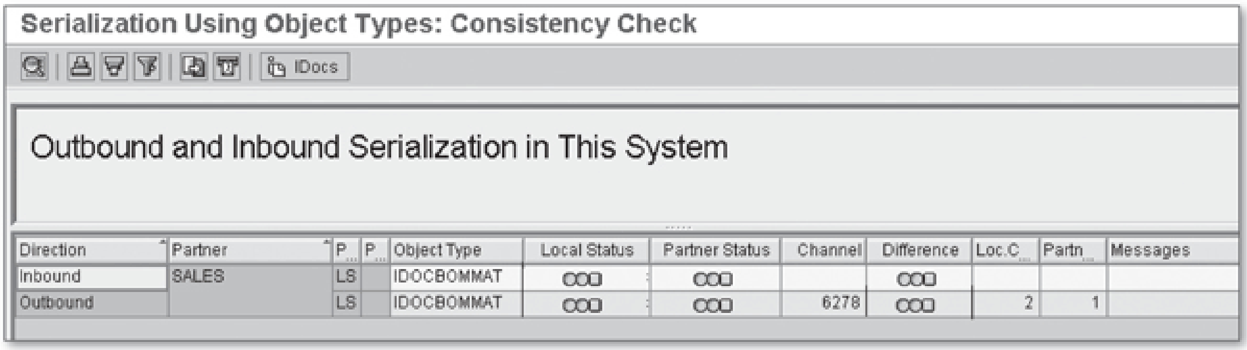
Send part. no.	Part. ty.	Part.	Obj. Type	Ser. flag
T90CLNT090	LS		IDOCBOMMAT	☑

Figure 6.16 Settings in Receiving System

Checking the settings

After you've finished configuring the settings in both systems, you can perform a consistency check in Transaction BD101. Here, you see whether serialization using object types is activated. The status traffic lights in the Local Status and Partner Status fields also indicate whether Customizing is okay and consistent. The traffic light in the Difference field then shows whether some IDocs queue-jumped. This is the case in our example in Figure 6.17. If all of the IDocs in both systems have been posted in the correct sequence, the values in the Counter field should also correspond here.

You can use Transaction BD100 to specifically view only those IDocs that will be processed using object channel serialization. Figure 6.18 shows the three IDocs that have already been dispatched in our particular example.

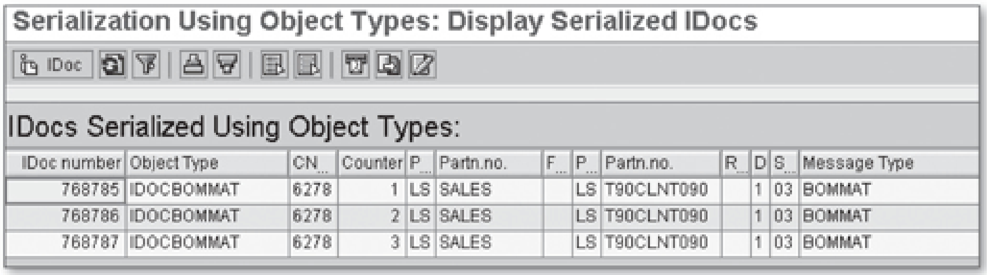


Serialization Using Object Types: Consistency Check

Outbound and Inbound Serialization in This System

Direction	Partner	P...	P...	Object Type	Local Status	Partner Status	Channel	Difference	Loc.C...	Partn...	Messages
Inbound	SALES	LS		IDOCBOMMAT	○○○	○○○		○○○			
Outbound		LS		IDOCBOMMAT	○○○	○○○	6278	○○○	2	1	

Figure 6.17 Check Function for Object Channel Serialization



Serialization Using Object Types: Display Serialized IDocs

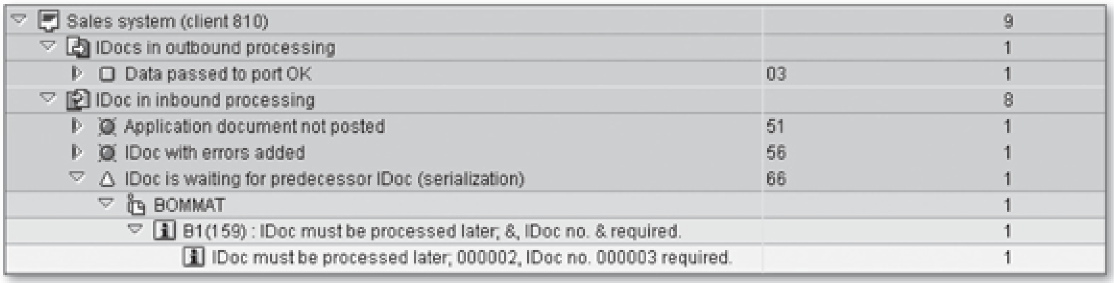
IDocs Serialized Using Object Types:

IDoc number	Object Type	CN...	Counter	P...	Partn.no.	F...	P...	Partn.no.	R...	D	S...	Message Type
768785	IDOCBOMMAT	6278	1	LS	SALES		LS	T90CLNT090		1	03	BOMMAT
768786	IDOCBOMMAT	6278	2	LS	SALES		LS	T90CLNT090		1	03	BOMMAT
768787	IDOCBOMMAT	6278	3	LS	SALES		LS	T90CLNT090		1	03	BOMMAT

Figure 6.18 IDocs with Object Channel Serialization

Wait status for IDocs that queue-jump

In the receiving system, individual processing of specific IDocs ensures that our second IDoc is posted first. The system "notices" that it's not yet its own number's turn in the relevant channel, so it changes its status to waiting status *66* (see Figure 6.19). If the first IDoc arrives and is successfully processed, it can be rescheduled and processed again.

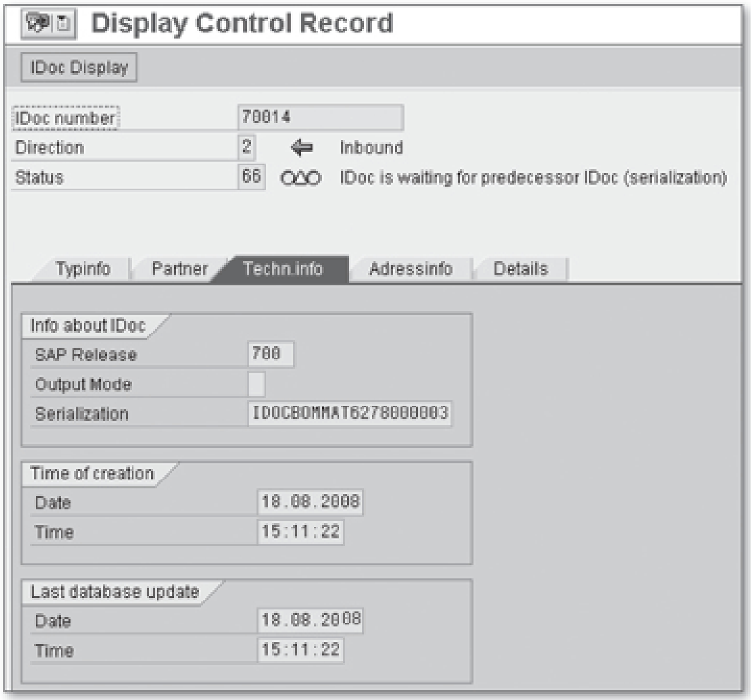


Sales system (client 810)		9
IDocs in outbound processing		1
Data passed to port OK	03	1
IDoc in inbound processing		8
Application document not posted	51	1
IDoc with errors added	56	1
IDoc is waiting for predecessor IDoc (serialization)	66	1
BOMMAT		1
B1(159) : IDoc must be processed later; &, IDoc no. & required.		1
IDoc must be processed later; 000002, IDoc no. 000003 required.		1

Figure 6.19 Wait Status for IDoc That Queue-Jumped

"Serial" field in control record

Once again, the SERIAL field in the IDoc control record determines the sequence here. However, this field doesn't contain a timestamp, which is usually the case, but rather a combination of the object name, object channel number (of the channel used for the transfer), and the transfer number. This is the value in the Counter field, which you can also see in the IDoc display screen in Transaction BD100. Figure 6.20 shows the control record of the sample IDoc that queue-jumped and has the counter "3." The IDocs with the counter "1" and "2" haven't arrived yet, so our IDoc has been assigned status 66, which you can also see here.



Display Control Record

IDoc number: 70014
Direction: 2 Inbound
Status: 66 IDoc is waiting for predecessor IDoc (serialization)

Info about IDoc
SAP Release: 700
Output Mode:
Serialization: IDOCBOMMAT6278000003

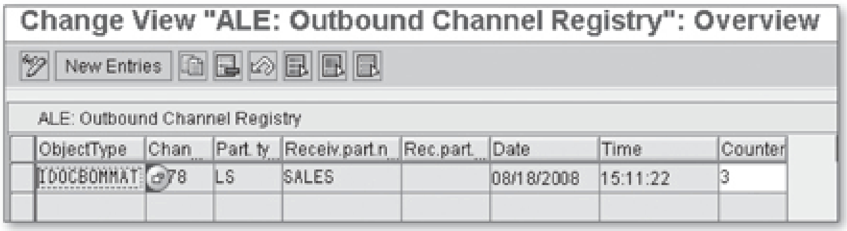
Time of creation
Date: 18.08.2008
Time: 15:11:22

Last database update
Date: 18.08.2008
Time: 15:11:22

Figure 6.20 Control Record for a Serialized IDoc

Data channel registry

To get an overview of the statuses of the serialized IDocs, you can view the *data channel registry* (in outbound processing, you do this in Transaction BD102). The result is displayed in Figure 6.21. Three IDocs have already been sent.



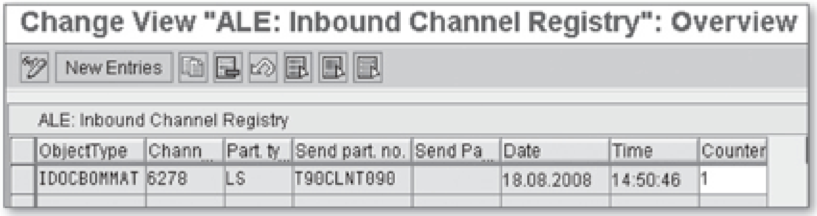
Change View "ALE: Outbound Channel Registry": Overview

ALE: Outbound Channel Registry

ObjectType	Chan	Part. ty	Receiv.part.n	Rec.part.	Date	Time	Counter
IDOCBOMMAT	78	LS	SALES		08/18/2008	15:11:22	3

Figure 6.21 View of Registry in Sending System

You can use Transaction BD103 to view the same process in the receiving system and see that, so far, just one IDoc has arrived (see Figure 6.22). It's exactly this situation that produces this standby position in IDoc processing, in other words, different values in the Counter field in the receiving and sending systems.


Change View "ALE: Inbound Channel Registry": Overview

ALE: Inbound Channel Registry

ObjectType	Chann…	Part. ty…	Send part. no.	Send Pa…	Date	Time	Counter
IDOCBOMMAT	6278	LS	T90CLNT090		18.08.2008	14:50:46	1

Figure 6.22 View of Registry in Receiving System

6.4 Serialization Using qRFC

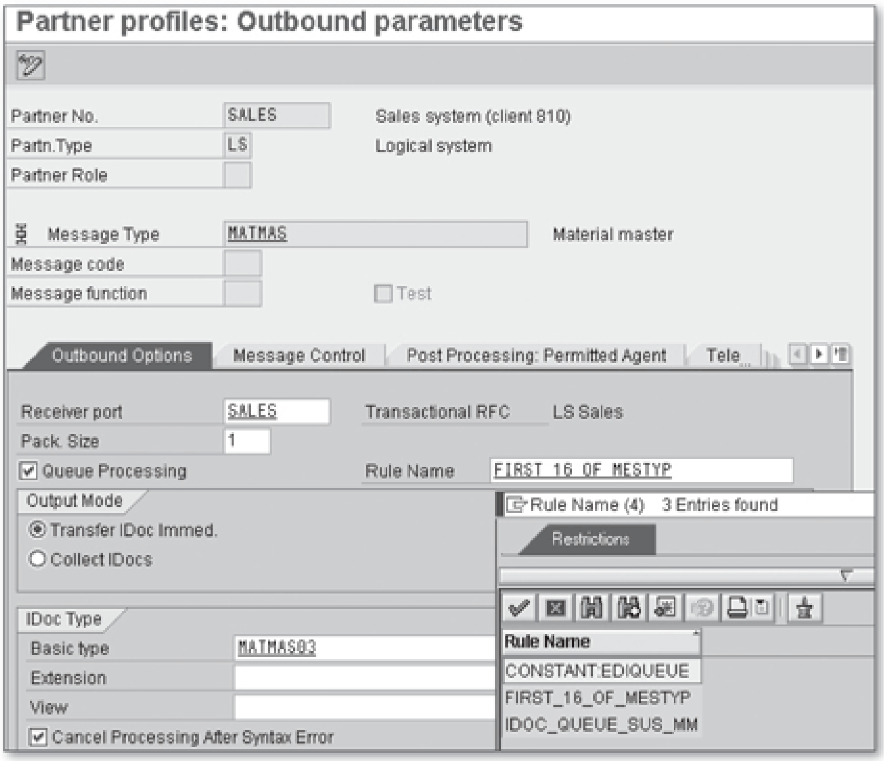
Serialization using qRFC (*Queued RFC*), in which the sequence of the IDocs is also retained, has only been available since SAP NetWeaver 6.40. Instead of the function module IDOC_INBOUND_ASYNCHRONOUS, the function module IDOC_INBOUND_IN_QUEUE is called in the target system. This type of serialization is possible for all IDocs. In SAP NetWeaver PI, it can also be used to implement the *Quality of Service* EOIO (*Equally Once In Order*) in the IDoc adapter.

Serialization using qRFC

You configure serialization using qRFC in the outbound partner profile by setting the Queue Processing flag on the Outbound Options tab page (see Figure 6.23). This opens a new field in which you specify how you want the queue name to be created using rules that reference a function module. Figure 6.23 shows three of the rules delivered by SAP (bottom right). The rule IDOC_QUEUE_SUS_MM requires the use of an IDoc of the type ORDERS or ORDRSP.

Rules for queue names

The function modules for creating queue names can be assigned a rule name in Transaction WE85. Figure 6.24 shows this assignment for the three rules shown in Figure 6.23: CONSTANT:EDIQUEUE, FIRST_16_OF_MESTYP, and IDOC_QUEUE_SUS_MM.



Partner profiles: Outbound parameters

Partner No.	SALES	Sales system (client 810)
Partn.Type	LS	Logical system
Partner Role		
Message Type	MATMAS	Material master
Message code		
Message function		□ Test

Receiver port	SALES	Transactional RFC	LS Sales
Pack. Size	1		
☑ Queue Processing		Rule Name	FIRST 16 OF MESTYP

Output Mode
● Transfer IDoc Immed.
○ Collect IDocs

IDoc Type

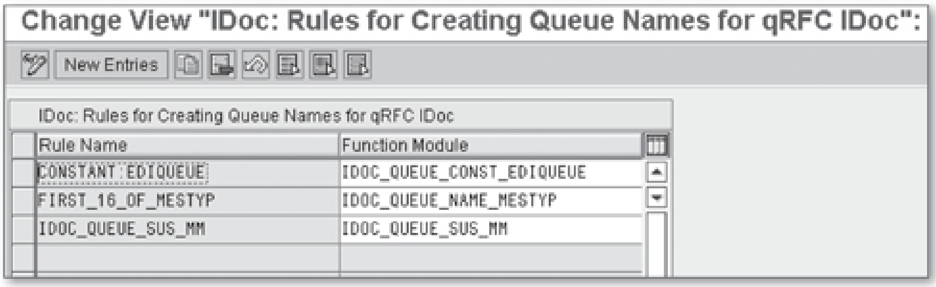
Basic type	MATMAS03
Extension	
View	

☑ Cancel Processing After Syntax Error

Rule Name (4) 3 Entries found

Rule Name
CONSTANT:EDIQUEUE
FIRST_16_OF_MESTYP
IDOC_QUEUE_SUS_MM

Figure 6.23 Partner Profile with Queue Processing



Change View "IDoc: Rules for Creating Queue Names for qRFC IDoc":

IDoc: Rules for Creating Queue Names for qRFC IDoc

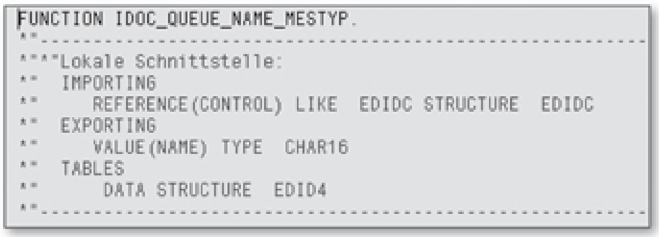
Rule Name	Function Module
CONSTANT:EDIQUEUE	IDOC_QUEUE_CONST_EDIQUEUE
FIRST_16_OF_MESTYP	IDOC_QUEUE_NAME_MESTYP
IDOC_QUEUE_SUS_MM	IDOC_QUEUE_SUS_MM

Figure 6.24 Generating Queue Names

If you're working with your own function modules, these require a suitable module signature (see Figure 6.25).

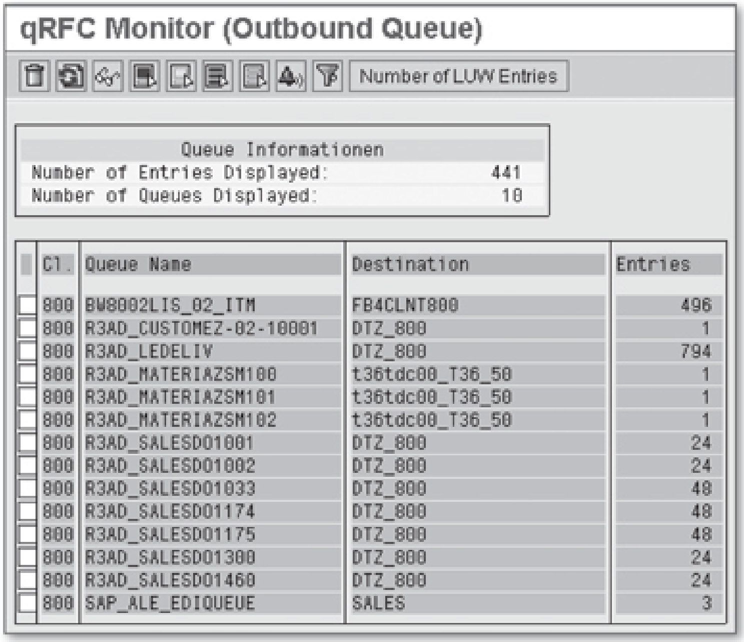
Queue monitor

Figure 6.26 shows our example (at the very bottom): The queue has been given a fixed name (SAP_ALE_EDIQUEUE), the RFC destination addressed by this queue is called SALES, and the transaction that you can use to monitor the outbound queue is called WEOUTQUEUE.



```
FUNCTION IDOC_QUEUE_NAME_MESTYP.
*"----------------------------------------------------------------------
*"*"Lokale Schnittstelle:
*"  IMPORTING
*"     REFERENCE(CONTROL) LIKE  EDIDC STRUCTURE  EDIDC
*"  EXPORTING
*"     VALUE(NAME) TYPE  CHAR16
*"  TABLES
*"      DATA STRUCTURE  EDID4
*"----------------------------------------------------------------------
```

Figure 6.25 Signature of a Function Module for Creating Queues



qRFC Monitor (Outbound Queue)

Queue Informationen

Number of Entries Displayed: 441

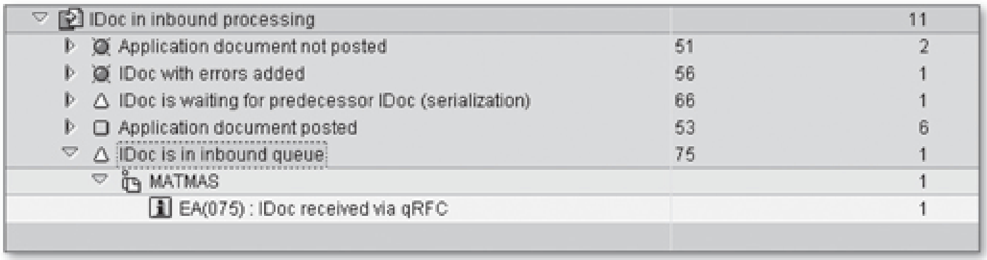
Number of Queues Displayed: 10

Cl.	Queue Name	Destination	Entries
800	BW8002LIS_02_ITM	FB4CLNT800	496
800	R3AD_CUSTOMEZ-02-10001	DTZ_800	1
800	R3AD_LEDELIV	DTZ_800	794
800	R3AD_MATERIAZSM100	t36tdc00_T36_50	1
800	R3AD_MATERIAZSM101	t36tdc00_T36_50	1
800	R3AD_MATERIAZSM102	t36tdc00_T36_50	1
800	R3AD_SALESD01001	DTZ_800	24
800	R3AD_SALESD01002	DTZ_800	24
800	R3AD_SALESD01033	DTZ_800	48
800	R3AD_SALESD01174	DTZ_800	48
800	R3AD_SALESD01175	DTZ_800	48
800	R3AD_SALESD01300	DTZ_800	24
800	R3AD_SALESD01460	DTZ_800	24
800	SAP_ALE_EDIQUEUE	SALES	3

Figure 6.26 Outbound SALES Queue

Stopped queue in status monitor

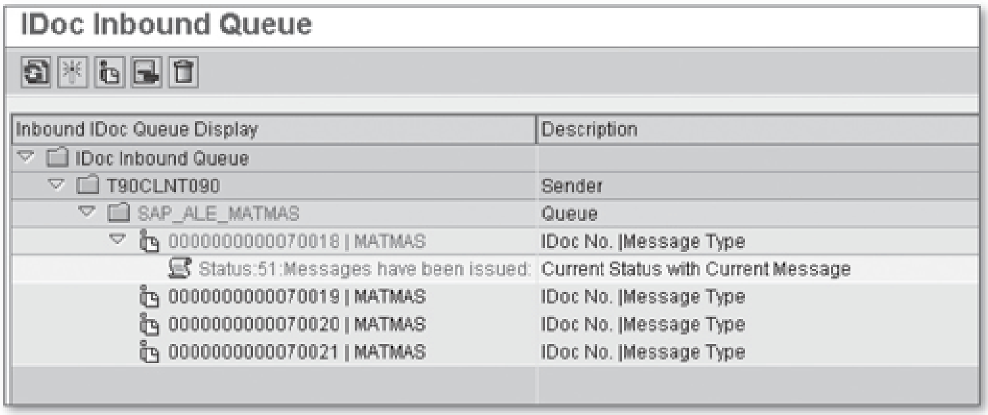
So, if you have created the queue, and, for example, an IDoc in the receiving system now encounters an error, subsequent IDocs can no longer be posted and each obtain status 75: *IDoc is in inbound queue* (see Figure 6.27).



IDoc in inbound processing		11
Application document not posted	51	2
IDoc with errors added	56	1
IDoc is waiting for predecessor IDoc (serialization)	66	1
Application document posted	53	6
IDoc is in inbound queue	75	1
MATMAS		1
EA(075) : IDoc received via qRFC		1

Figure 6.27 IDocs in Target System

The error scenario is also shown in the inbound queue, which is monitored using Transaction WEINBQUEUE (see Figure 6.28).



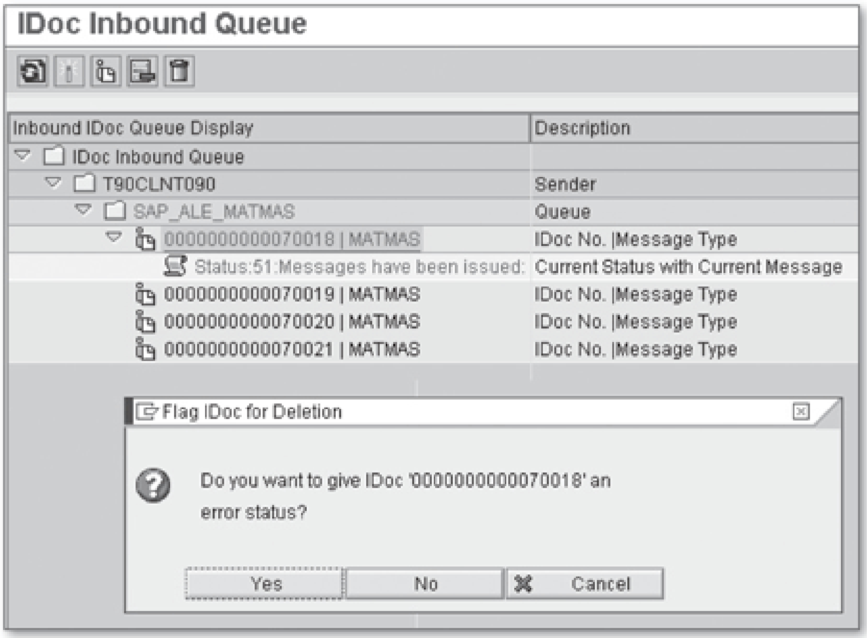
IDoc Inbound Queue

Inbound IDoc Queue Display	Description
IDoc Inbound Queue	
T90CLNT090	Sender
SAP_ALE_MATMAS	Queue
0000000000070018 | MATMAS	IDoc No. |Message Type
Status:51:Messages have been issued:	Current Status with Current Message
0000000000070019 | MATMAS	IDoc No. |Message Type
0000000000070020 | MATMAS	IDoc No. |Message Type
0000000000070021 | MATMAS	IDoc No. |Message Type

Figure 6.28 Displaying IDocs in a Queue

Error handling in an inbound queue

You can postprocess the IDoc here, ideally in such a way that it can be posted successfully. However, you can also use the "trashcan" icon to flag the IDoc blocking the queue for deletion (status *68*) (see Figure 6.29).



IDoc Inbound Queue

Inbound IDoc Queue Display	Description
IDoc Inbound Queue	
T90CLNT090	Sender
SAP_ALE_MATMAS	Queue
0000000000070018 | MATMAS	IDoc No. |Message Type
Status:51:Messages have been issued:	Current Status with Current Message
0000000000070019 | MATMAS	IDoc No. |Message Type
0000000000070020 | MATMAS	IDoc No. |Message Type
0000000000070021 | MATMAS	IDoc No. |Message Type

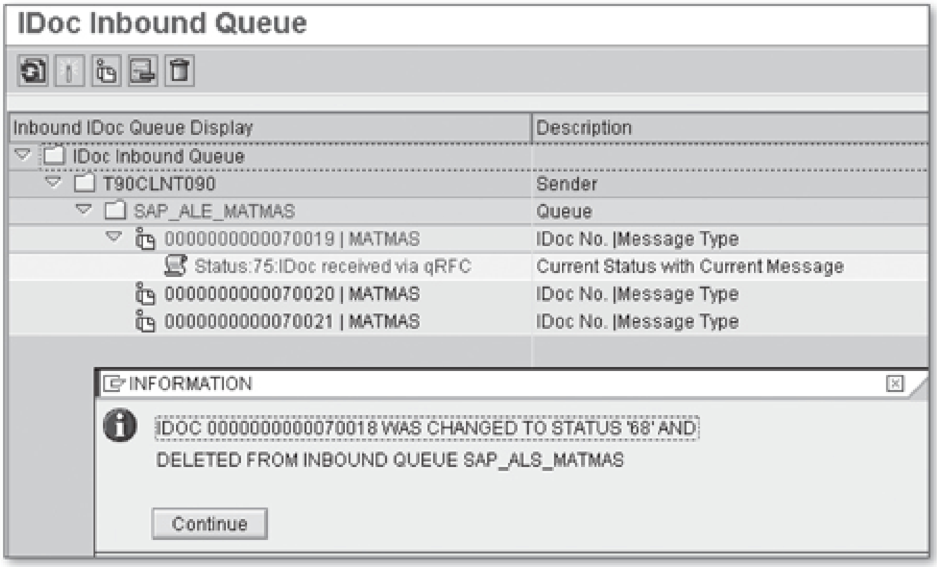
Flag IDoc for Deletion

Do you want to give IDoc '0000000000070018' an error status?

Figure 6.29 Deleting IDocs from a Queue

Restarting a queue

After you've processed or removed the IDoc that contains errors (see Figure 6.30), you can restart the queue. All other IDocs are now posted (Figure 6.31).



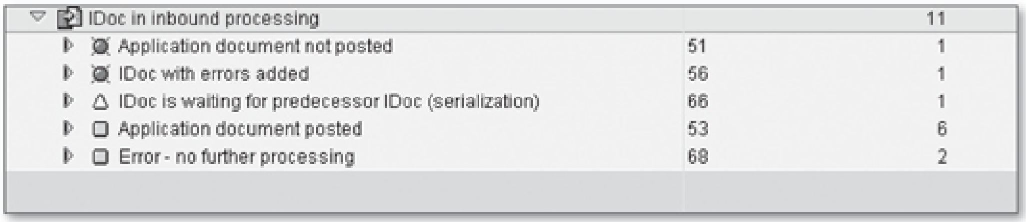
IDoc Inbound Queue

Inbound IDoc Queue Display	Description
IDoc Inbound Queue	
T90CLNT090	Sender
SAP_ALE_MATMAS	Queue
0000000000070019 | MATMAS	IDoc No. |Message Type
Status:75:IDoc received via qRFC	Current Status with Current Message
0000000000070020 | MATMAS	IDoc No. |Message Type
0000000000070021 | MATMAS	IDoc No. |Message Type

INFORMATION

IDOC 0000000000070018 WAS CHANGED TO STATUS '68' AND DELETED FROM INBOUND QUEUE SAP_ALS_MATMAS

Figure 6.30 Result of Delete Operation



IDoc in inbound processing		11
Application document not posted	51	1
IDoc with errors added	56	1
IDoc is waiting for predecessor IDoc (serialization)	66	1
Application document posted	53	6
Error - no further processing	68	2

Figure 6.31 Result After Restarting a Queue

6.5 Summary

In this chapter, you've seen all possibilities for serialization of IDocs, and learned about the places to find which serialization type is implemented for your IDoc. The different types of serialization may lead to additional work if they are out of sync, so make sure you only use serialization where it is really needed for your process.

This chapter deals with administrative issues, specifically, IDoc-related information that is written to the database and regular background jobs.

7 Administration

This chapter gives you an overview of the administration options for IDocs. One option discussed in this chapter is how to link IDocs to display or search for information. Another is to increase the performance of your system by scheduling regular reports that you would otherwise have to run manually. This chapter also summarizes IDoc functionalities and the associated transaction codes, and explains how IDocs are archived and their status converted.

7.1 IDoc Links

There are two types of links: links between IDocs and the actual business objects when a business object type is assigned; and links between the sender's IDoc and the recipient's IDoc. The latter type of link is known only to the recipient.

IDoc links have their own database table called IDOCREL (see Table 7.1). The table contains both data items that are related to each other; they are identified by their internal number.

"SRRELROLES" table

The SRRELROLES table contains the objects that are associated with these internal numbers. There are several different link types in this table. Table 7.2 shows the entry types that are relevant to IDocs.

Link Type	ROLE A	ROLE B
IDC0	OUTBELEG	OUTIDOC
IDC1	INIDOC	INBELEG
IDC4	INIDOC	OUTIDOC
IDC8	INIDOC	INTID
IDC9	OUTIDOC	INBELEG
IDCA	OUTIDOC	OUTID
IDCB	INIDOC	OUTBELEG

Table 7.1 IDoc-Related Entries in the "IDOCREL" Database Table

Abbreviation in the ROLETYPE Field	Description
INBELEG	Recipient document
INIDOC	Recipient IDoc
INTID	Transaction ID that came with the IDoc
OUTBELEG	Sender document
OUTID	Sender transaction ID
OUTIDOC	Sender IDoc

Table 7.2 Entries in the "SRRELROLES" Table

For an example, let's look at an IDoc and a material master in this table (see Figure 7.1). The `OBJKEY` in each case is the unique object key, for example, the material number or the IDoc number. For documents, the `OBJTYPE` is always the associated business object type; for IDocs, on the other hand, it's always "IDOC"; and for transaction IDs, it's "TRANSID".

If an IDoc is created for this material, there will also be an entry for this in the `SRRELROLES` table (see Figure 7.2). In both objects, the unique number that is referred to in the Table `IDOCREL` is included at the field `ROLEID`.

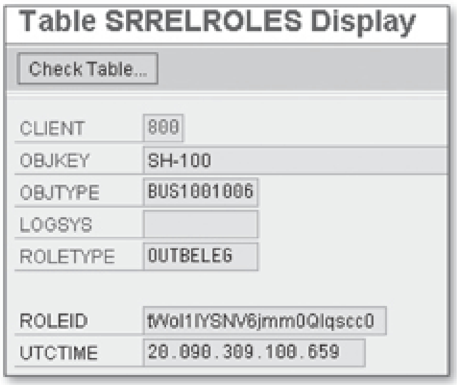


Table SRRELROLES Display

CLIENT	800
OBJKEY	SH-100
OBJTYPE	BUS1001006
LOGSYS	
ROLETYPE	OUTBELEG
ROLEID	fWol1IYSNV6jmm0Qlqscc0
UTCTIME	20.090.309.100.659

Figure 7.1 Example of a Material

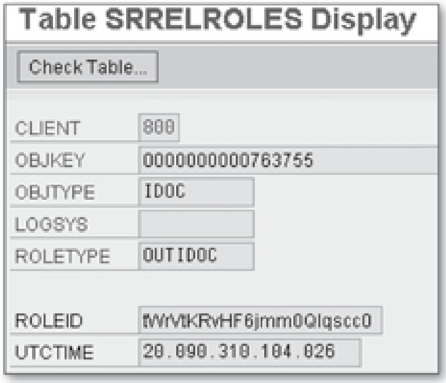


Table SRRELROLES Display

CLIENT	800
OBJKEY	0000000000763755
OBJTYPE	IDOC
LOGSYS	
ROLETYPE	OUTIDOC
ROLEID	fWrVtKRvHF6jmm0Qlqscc0
UTCTIME	20.090.310.104.026

Figure 7.2 Example of the IDoc to Be Included in a Link

Outbound material

Our example objects, IDoc `763755` and material `SH-100`, have the link type `IDC0` because we're on the outbound side of the link. Figure 7.3 shows the relevant table entry in Transaction SE11.

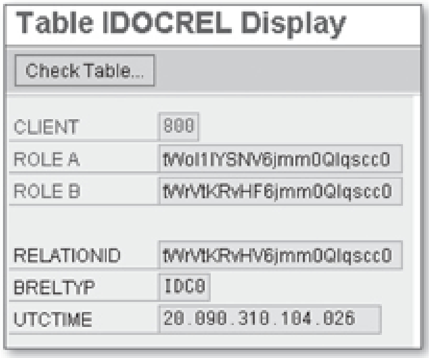


Table IDOCREL Display

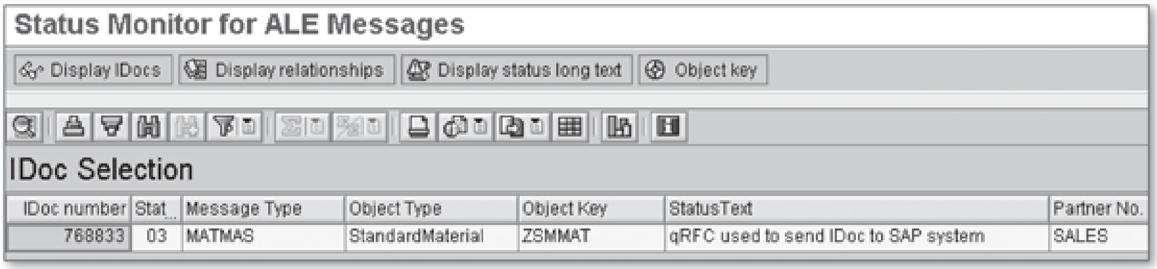
CLIENT	800
ROLE A	fWol1IYSNV6jmm0Qlqscc0
ROLE B	fWrVtKRvHF6jmm0Qlqscc0
RELATIONID	fWrVtKRvHV6jmm0Qlqscc0
BRELTYP	IDC0
UTCTIME	20.090.310.104.026

Figure 7.3 Entry in Database Table "IDOCREL"

All other links are managed in a similar way. They can be viewed from within IDoc monitoring in Transaction BD87 or in the individual IDoc display by means of the object services.

Object key

The link to the object type can also be used to display the key values of the IDoc contents. To do this, use the Object Key function in Transaction BD87 (see Figure 7.4).



Status Monitor for ALE Messages

IDoc Selection

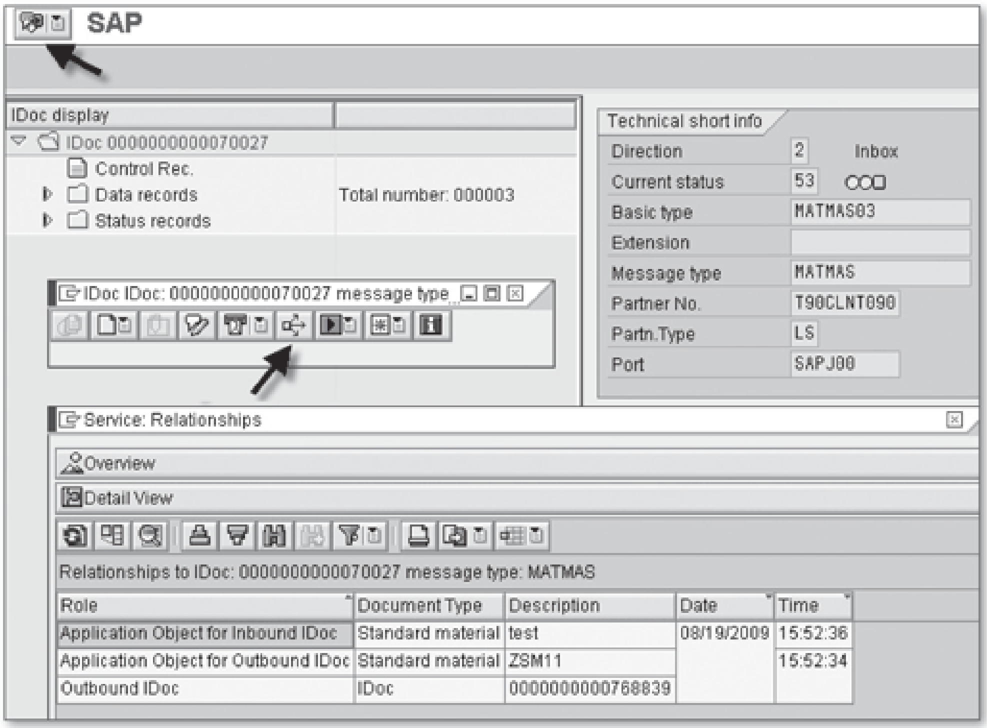
IDoc number	Stat...	Message Type	Object Type	Object Key	StatusText	Partner No.
768833	03	MATMAS	StandardMaterial	ZSMMAT	qRFC used to send IDoc to SAP system	SALES

Figure 7.4 Object Key in Transaction BD87

The other links can be displayed from within the individual display. You can also view a related object in the individual display by simply double-clicking it. If you've edited an IDoc, you'll also see the link to the original IDoc here. This original object is retained for the sake of completeness. Figure 7.5 shows the recipient system (because this system contains a higher number of links).

Deactivate link

Also Business objects that are part of links can be displayed directly from here. You can access this function via the object services, as before (top arrow in Figure 7.5). To get to the actual links for the IDoc, use the IDoc links icon (lower arrow in Figure 7.5). If you don't want to have all links written to the database Transaction WENOLINKS is used to specify which links are written for which message type. The lower part of Figure 7.6 shows a deactivated link for `MATMAS`.



IDoc display

- IDoc 0000000000070027
  - Control Rec.
  - Data records — Total number: 000003
  - Status records

IDoc IDoc: 0000000000070027 message type

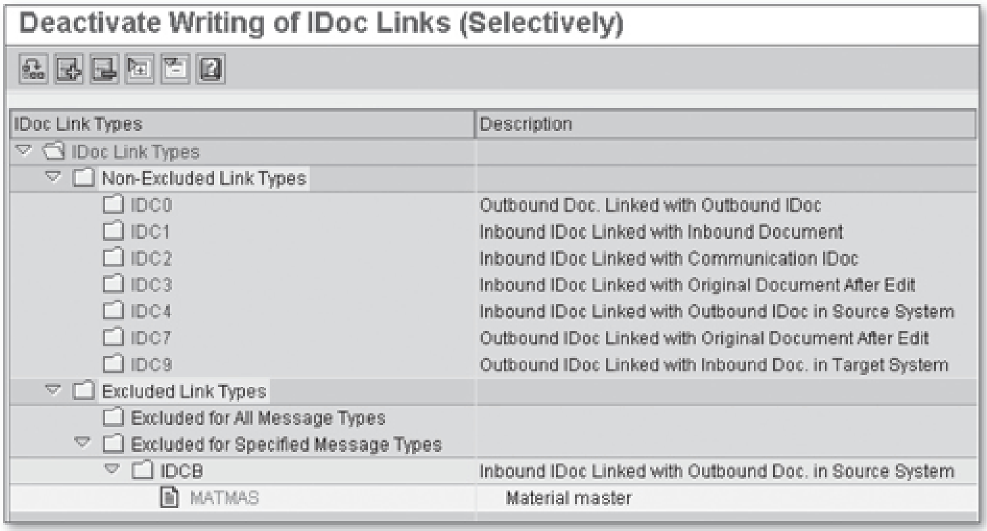
Technical short info		
Direction	2	Inbox
Current status	53	
Basic type	MATMAS03	
Extension		
Message type	MATMAS	
Partner No.	T90CLNT090	
Partn.Type	LS	
Port	SAPJ00	

Service: Relationships

Relationships to IDoc: 0000000000070027 message type: MATMAS

Role	Document Type	Description	Date	Time
Application Object for Inbound IDoc	Standard material	test	08/19/2009	15:52:36
Application Object for Outbound IDoc	Standard material	ZSM11		15:52:34
Outbound IDoc	IDoc	0000000000768839		

Figure 7.5 Links in Recipient System


Deactivate Writing of IDoc Links (Selectively)

IDoc Link Types	Description
IDoc Link Types	
Non-Excluded Link Types	
IDC0	Outbound Doc. Linked with Outbound IDoc
IDC1	Inbound IDoc Linked with Inbound Document
IDC2	Inbound IDoc Linked with Communication IDoc
IDC3	Inbound IDoc Linked with Original Document After Edit
IDC4	Inbound IDoc Linked with Outbound IDoc in Source System
IDC7	Outbound IDoc Linked with Original Document After Edit
IDC9	Outbound IDoc Linked with Inbound Doc. in Target System
Excluded Link Types	
Excluded for All Message Types	
Excluded for Specified Message Types	
IDCB	Inbound IDoc Linked with Outbound Doc. in Source System
MATMAS	Material master

Figure 7.6 Deactivating Links

Report RSRLDREL is used to delete written links from the system.

7.2 Regular Jobs

During the process of setting up and testing IDoc links, many things are done manually that are processed by means of regular jobs in the production system. Likewise, in test systems, the processing mode for sending and processing IDocs is often set to Process Immediately in the partner profiles. For performance reasons, this isn't recommended in production systems and is listed by SAP as incorrect in go-live checks.

Important reports

Table 7.3 lists some important reports and their functions. Most of these reports can be regularly scheduled.

Report Name	Report Function
IDX_DELETE_IDOCTYP_ WITHOUT_IS	Delete IDoc type from SAP NetWeaver PI exceptions table (only available in SAP NetWeaver PI)
IDX_SELECT_IDOCTYP_ WITHOUT_IS	Copy IDoc type to SAP NetWeaver PI exceptions table (only available in SAP NetWeaver PI)
RBDAGAI2	Post IDocs after ALE error
RBDAGAIN	Send IDocs after an ALE error
RBDAPP01	IDoc passed to application
RBDAUD01	ALE audit: statistical analyses
RBDAUD02	Reorganizing the audit database
RBDCPCLR	Delete change pointers
RBDMANIN	Reposting of IDocs (ALE)
RBDMIDOC	Create IDocs from change pointers
RBDMOIND	Convert IDoc status
RBDSER01	Create IDocs of a serialization group from change pointers
RBDSER02	Dispatch IDocs of a serialization group
RBDSER03	Check the send status of the IDocs of a serialization group

Table 7.3 Programs for Regular Scheduling

Report Name	Report Function
RBDSER04	Inbound processing of IDocs of a serialization group
RBDSRCLR	Delete serialization data
RBDSTATE	ALE audit: sending the confirmations
RSARFCEX	Restart "frozen" tRFCs
RSEOU00	Send IDocs in status 30
RSRLDREL	Delete IDoc links
SAPLBDRC	ALE: determine recovery objects

Table 7.3 Programs for Regular Scheduling (Cont.)

7.3 Transaction Code Overview

Table 7.4 contains the transaction codes for all of the functionalities used in this book. Menu information has been deliberately omitted from this list, where possible, as the SAP menus sometimes change from one release to the next. The text also makes direct reference to the transaction, where possible.

General transaction codes

Table 7.4 contains the general transaction codes, while Table 7.5 contains the transaction codes that, while related to IDocs, refer directly to the objects of a specific module and aren't for general use.

Transaction Code	Function
BD20	IDoc passed to application
BD21	Select change pointer
BD22	Delete change pointer
BD23	Delete serialization data
BD40	Read change pointer for group
BD41	Dispatch IDocs for group
BD42	Check IDocs for group
BD43	Post IDocs for group

Table 7.4 Main General Transaction Codes

Transaction Code	Function
BD44	Assign message type to serialization group
BD47	Dependencies between methods
BD48	Dependency method – message
BD50	Activate change parameters for message type
BD51	Maintain function modules (inbound)
BD52	Activate change pointer per change document item
BD53	Reduce message types
BD54	Maintain logical systems
BD55	Assign rule to message type
BD56	Filter IDoc segments
BD57	Maintain link and serialization type
BD58	Convert organizational units
BD59	Maintain filter objects
BD60	Maintain function module for analysis
BD61	Activate change pointers generally
BD62	Create rule
BD63	Transport ALE table of message type
BD64	Maintain distribution model
BD65	Define mandatory fields
BD66	Assign IDoc field to change document
BD67	Maintain methods (inbound)
BD68	Assign classes to recipient logical system
BD69	Assignment of message type to IDoc
BD79	ALE IDoc segments: conversion rules
BD81	Filter objects: parameter filtering
BD82	Generate partner profiles
BD83	Send IDocs after an ALE error
BD84	Post IDocs after ALE error
BD85	Rules for creating queue names for qRFC IDocs

Table 7.4 Main General Transaction Codes (Cont.)

Transaction Code	Function
BD87	Status monitor
BD95	Change ALE object type, assign filter objects segment (IDoc) and serialization objects
BD96	Filter objects: recipient determination (BAPI)
BD97	Determine RFC destinations for method calls
BD99	Message type dependencies
BD100	IDoc display object channel view
BD101	Consistency check
BD102	Outbound registry
BD103	Inbound registry
BD104	Assign IDocs to business object
BD105	Maintain supported business objects
BDA4	Assign message type to object type
BDBG	Generate ALE interface
BDCCC	ALE Basis Customizing data: check center
BDCCV	Maintain items to be checked
BDLSM	Convert ALE Basis Customizing data: conversion matrix
BDLSS	Convert logical system names after client is copied (not in production systems)
BDLST	Convert ALE Basis Customizing data: execution (note: starts immediately without warning)
BDM2	Monitoring: recipient IDocs (IDoc tracking)
BDM5	Technical consistency check
BDM7	ALE audit: statistical analyses
BDM8	ALE audit: sending the confirmations
BDM9	Reorganize the audit database
BDMO	ALE CCMS group administration
BDMONIC	Maintain ALE CCMS monitoring objects (definition)
BDMONIC2	Maintain ALE CCMS monitoring objects (group definition)

Table 7.4 Main General Transaction Codes (Cont.)

Transaction Code	Function
BDMONIC3	ALE CCMS monitor branch
BDR1	Display application log for recovery
BDR2	Reorganize recovery data
BDRC	ALE: determine recovery objects
BDRL	ALE: process recovery objects
BDTP	Business process – maintain templates
BF01	Library of "publish and subscribe" business transaction events
BF05	Library of process business transaction events
CMOD	Project management of SAP extensions
FIBF	SAP Business Framework: business transaction events
OB72	Create global company code
OYEA	Global parameters for IDoc interface
PFTC_CHG	Modify workflow task
PFTC_COP	Copy workflow task
PFTC_DEL	Delete workflow task
PFTC_DIS	Display workflow task
PFTC_INS	Create workflow task
SALE	ALE Customizing
SARA	Archive administration
SARI	Archive information system: central administration
SARJ	Archive retrieval configurator
SE11	ABAP Dictionary: initial screen
SE18	BAdI Builder: definition maintenance initial screen
SE19	BAdI Builder: implementation maintenance initial screen
SE38	ABAP Editor: initial screen
SE80	Object Navigator
SE84	Repository info system
SMOD	SAP Enhancement Management

Table 7.4 Main General Transaction Codes (Cont.)

Transaction Code	Function
WE02	Display IDocs
WE05	IDoc lists
WE06	Active IDoc monitoring
WE07	IDoc statistics
WE08	Status of file interface
WE09	Search for IDoc based on business content
WE10	Search for IDoc in database release 4.6C and earlier releases
WE11	Delete IDocs (note: use only after obtaining agreement)
WE12	Receipt of modified outbound file
WE14	Outbound processing from IDoc
WE15	Outbound processing from MC
WE16	Original inbound file
WE17	Process status file
WE18	Create status file
WE19	Test tool
WE20	Partner profiles
WE21	Ports for IDoc processing
WE23	Verification of IDoc processing
WE24	Default values for outbound parameters
WE27	Default values for inbound parameters
WE30	IDoc types
WE31	IDoc segments
WE32	IDoc views
WE34	Objects for display of XML IDocs
WE40	Settings for errors and status processing
WE41	Process codes, outbound
WE42	Process codes, inbound
WE46	Error and status processing (like WE40)

Table 7.4 Main General Transaction Codes (Cont.)

Transaction Code	Function
WE47	Maintain status values
WE54	Function modules for changing file names
WE55	Create file names
WE57	Assign message for application object
WE58	Status of process codes: texts
WE59	Status of process codes: modify
WE60	Documentation for IDoc types
WE61	Documentation for IDoc record types
WE62	Documentation for segments
WE63	Documentation for IDoc types (like WE60)
WE64	Documentation for message types
WE70	Conversion: basic types
WE71	Conversion: extensions
WE72	Conversion: IDoc types
WE73	Conversion: logical messages
WE81	Logical message types
WE82	Assignment of message type to IDoc type
WE84	Assignment of IDoc fields and application fields
WEDI	IDoc basis and EDI basis
WEINBQUEUE	Monitoring program for IDoc inbound queue
WEOUTQUEUE	Monitoring program for IDoc outbound queue
WENOLINKS	Switch off links
WELI	Maintain status groups

Table 7.4 Main General Transaction Codes (Cont.)

Application-specific transaction codes

Application-Specific Transaction Codes

Most of these transaction codes are from SMD and aren't mentioned explicitly in this book, but you'll likely find this table useful.

Transaction Code	Function
BD10	Send material
BD11	Get material
BD12	Send customer
BD13	Get customer
BD14	Send vendor
BD15	Get vendor
BD16	Send cost center
BD17	Get cost center
BD18	Send general ledger account
BD19	Get general ledger account
BD24	Send cost elements
BD25	Send activity type
BD26	Get activity type
BD27	Send cost center activity prices
BD28	Send object/cost center type control data
BD30	Distribute material object list
BD31	Distribute document object list
BD32	Distribute plant allocations (material BOMs)
BD33	Distribute material variants (ALE)
BD34	Distribute order BOM
BD35	Send business process groups
BD36	Send business processes
BD37	Send business process prices
BD85	Consistency check for transfer
BD86	Consistency check for sales
BD91	Send characteristic
BD92	Send class
BD93	Send classification
BDA5	Distribute documents

Table 7.5 Main Module-Specific Transaction Codes

Transaction Code	Function
BDD5	Application consistency check (SD)
BDFDF	Request fund
BDFDS	Send fund
BDMC	Upload info structures

Table 7.5 Main Module-Specific Transaction Codes (Cont.)

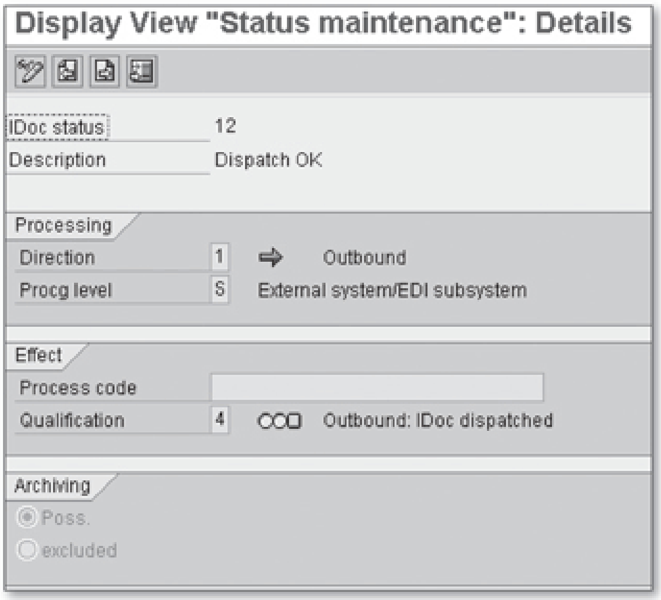
7.4 Archiving

GDPdU

IDocs are documents that, under German law at least, may have to be archived in accordance with the GDPdU (German Principles on Data Access and the Examination of Digital Documents). This legal instrument always applies in communication with partners when the IDoc is the first electronic document that you receive in a process. If you're reading the IDocs from a file that was sent by a partner, this file, too, must be archived. Likewise, in companies to which *Good Manufacturing Practice* (GMP) applies, all changes to the status of warehouse stock have to be archived. Thus, you can see that in most cases, IDocs cannot be simply deleted from the system; they have to be archived using SAP's proprietary archiving system.

Archivable status values

One IDoc-specific rule is that you have to specify for every possible IDoc status value whether IDocs with this status are archivable or not. Of course, SAP provides in archivable form those values that normally indicate that an IDoc has a "success" status and the associated document already exists (in inbound processing), or that the associated document was successfully transferred to the recipient (in outbound processing). The transaction for maintaining status values is WE47. Figure 7.7 shows the status maintenance screen for status value *12*, which indicates that an IDoc was sent successfully via RFC in outbound processing.



Display View "Status maintenance": Details

IDoc status: 12
Description: Dispatch OK

Processing
Direction: 1 ⇨ Outbound
Procg level: S External system/EDI subsystem

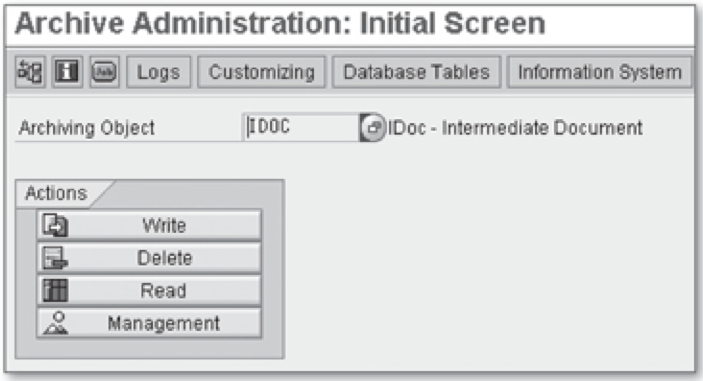
Effect
Process code:
Qualification: 4 Outbound: IDoc dispatched

Archiving
● Poss.
○ excluded

Figure 7.7 Setting Archiving Options per Status

Transaction SARA

The actual process of archiving is completed in Transaction SARA, as with all objects to be archived. The archiving object for IDocs is also called IDOC and contains the usual Write, Read, Management, and Delete functions (see Figure 7.8).



Archive Administration: Initial Screen

Archiving Object: IDOC IDoc - Intermediate Document

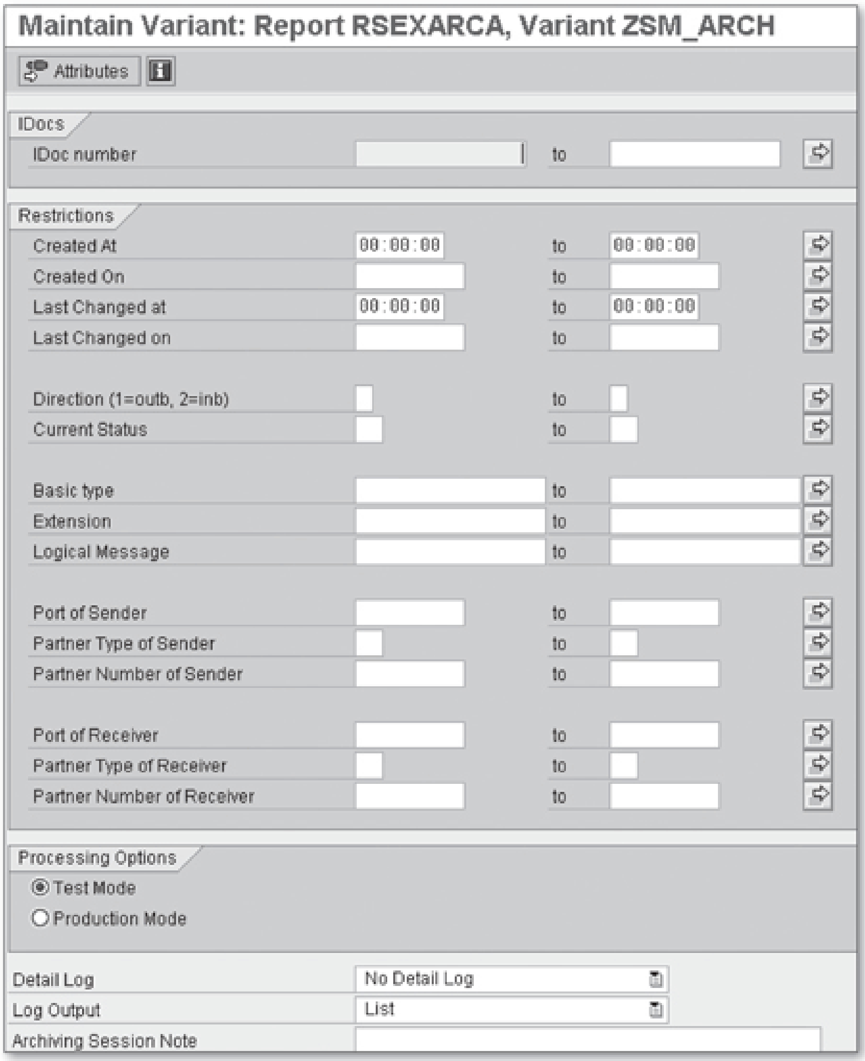
Actions

Figure 7.8 Archiving Object for IDocs

Archiving Criteria for IDocs

When it comes to the archiving report itself, which is started using the Write button, you can specify in great detail which IDocs are to be archived (see Figure 7.9). When test mode is active (shown as a flag on the Flow Control tab page), you're presented with an archive file, but the deletion report doesn't delete the archived IDocs from the database.

As a general rule, you shouldn't wait too long to archive IDocs. The fewer IDocs you have in the database, the faster your search processes will run, and viewing IDocs or searching field content will be just as easy after archiving.



Maintain Variant: Report RSEXARCA, Variant ZSM_ARCH

IDocs

IDoc number		to	

Restrictions

Created At	00:00:00	to	00:00:00
Created On		to	
Last Changed at	00:00:00	to	00:00:00
Last Changed on		to	
Direction (1=outb, 2=inb)		to	
Current Status		to	
Basic type		to	
Extension		to	
Logical Message		to	
Port of Sender		to	
Partner Type of Sender		to	
Partner Number of Sender		to	
Port of Receiver		to	
Partner Type of Receiver		to	
Partner Number of Receiver		to	

Processing Options

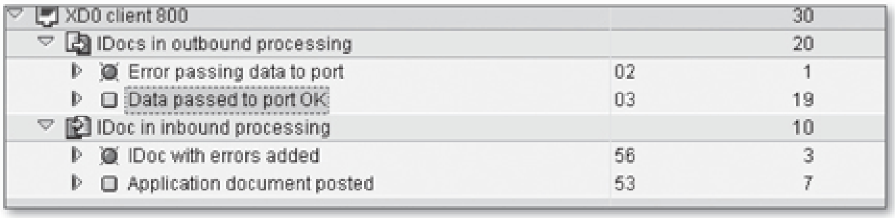
● Test Mode

○ Production Mode

Detail Log	No Detail Log
Log Output	List
Archiving Session Note	

Figure 7.9 Criteria for IDoc Archiving

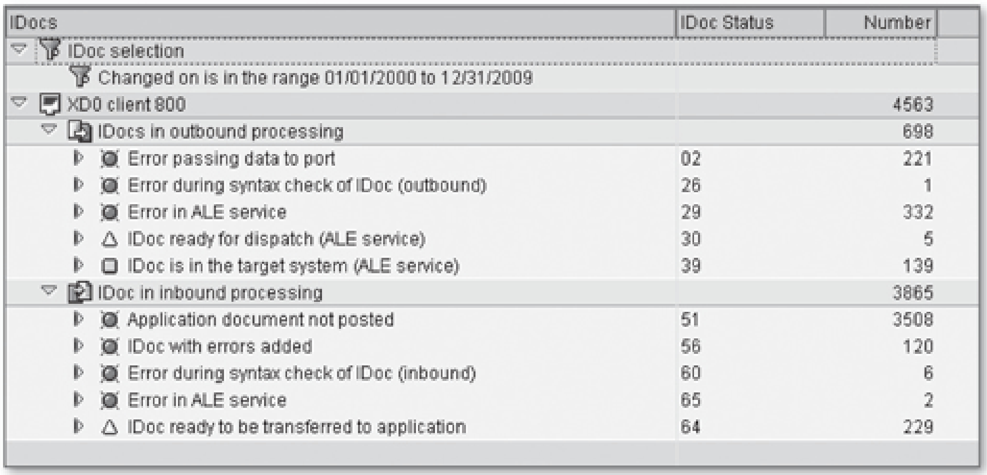
Our example contains both IDocs with a successful status and IDocs with an unsuccessful status (see Figure 7.10). The SAP standard settings for archiving status values specify that IDocs with errors remain active because they cannot be archived.



XD0 client 800		30
IDocs in outbound processing		20
Error passing data to port	02	1
Data passed to port OK	03	19
IDoc in inbound processing		10
IDoc with errors added	56	3
Application document posted	53	7

Figure 7.10 IDocs Before Archiving

Now, the archiving process is started with the All IDocs That Have an Archivable Status Will Be Archived setting (this happens if you haven't made any specific selection). Figure 7.11 shows the result.

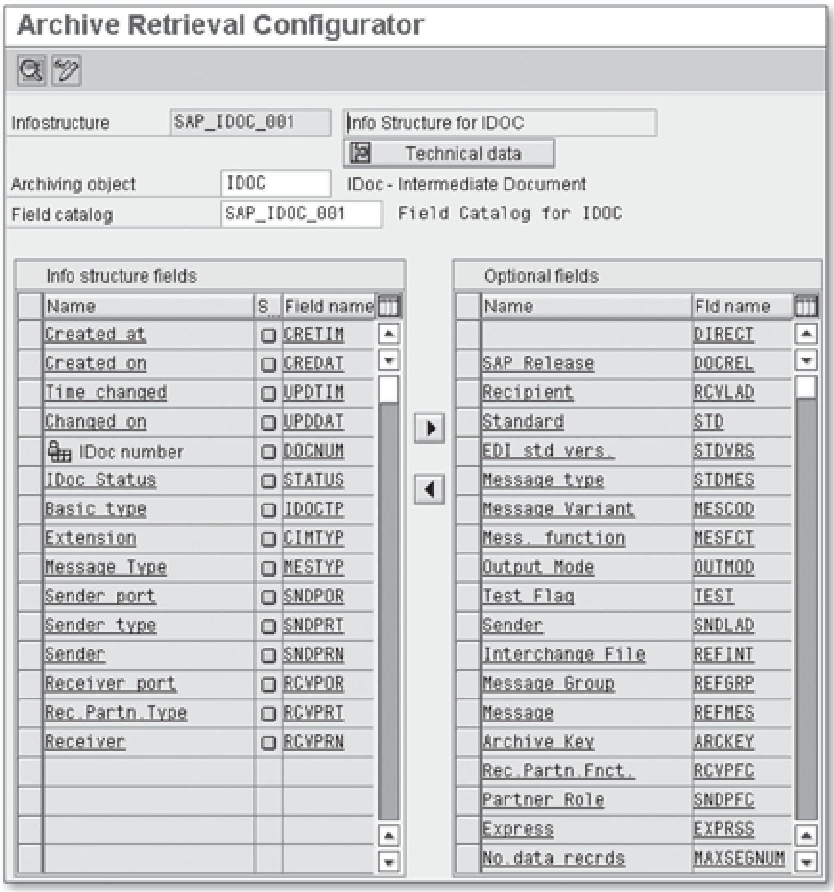


IDocs	IDoc Status	Number
IDoc selection		
Changed on is in the range 01/01/2000 to 12/31/2009		
XD0 client 800		4563
IDocs in outbound processing		698
Error passing data to port	02	221
Error during syntax check of IDoc (outbound)	26	1
Error in ALE service	29	332
IDoc ready for dispatch (ALE service)	30	5
IDoc is in the target system (ALE service)	39	139
IDoc in inbound processing		3865
Application document not posted	51	3508
IDoc with errors added	56	120
Error during syntax check of IDoc (inbound)	60	6
Error in ALE service	65	2
IDoc ready to be transferred to application	64	229

Figure 7.11 IDocs After Archiving

Activating info structure

As mentioned previously, it's also possible to search archived IDocs. A prerequisite for this is that you have activated the info structure provided by SAP, SAP_IDOC_001, or your own info structure, in Transaction SARJ. These info structures are used to establish the context for the data in the archive. For this purpose, part of the data, which should be kept as small as possible, is retained in the database while the remaining data is in the archive. For IDocs, it's sufficient to retain a small amount of data from the control record in the database; the vast majority is then deleted after archiving. Figure 7.12 shows the SAP-specific structure SAP_IDOC_001. For active structures, the corresponding data records are created automatically after archiving.



Archive Retrieval Configurator

Infostructure SAP_IDOC_001 Info Structure for IDOC

Archiving object IDOC IDoc - Intermediate Document

Field catalog SAP_IDOC_001 Field Catalog for IDOC

Info structure fields

Name	S	Field name
Created at	□	CRETIM
Created on	□	CREDAT
Time changed	□	UPDTIM
Changed on	□	UPDDAT
IDoc number	□	DOCNUM
IDoc Status	□	STATUS
Basic type	□	IDOCTP
Extension	□	CIMTYP
Message Type	□	MESTYP
Sender port	□	SNDPOR
Sender type	□	SNDPRT
Sender	□	SNDPRN
Receiver port	□	RCVPOR
Rec.Partn.Type	□	RCVPRT
Receiver	□	RCVPRN

Optional fields

Name	Fld name
	DIRECT
SAP Release	DOCREL
Recipient	RCVLAD
Standard	STD
EDI std vers.	STDVRS
Message type	STDMES
Message Variant	MESCOD
Mess. function	MESFCT
Output Mode	OUTMOD
Test Flag	TEST
Sender	SNDLAD
Interchange File	REFINT
Message Group	REFGRP
Message	REFMES
Archive Key	ARCKEY
Rec.Partn.Fnct.	RCVPFC
Partner Role	SNDPFC
Express	EXPRSS
No.data recrds	MAXSEGNUM

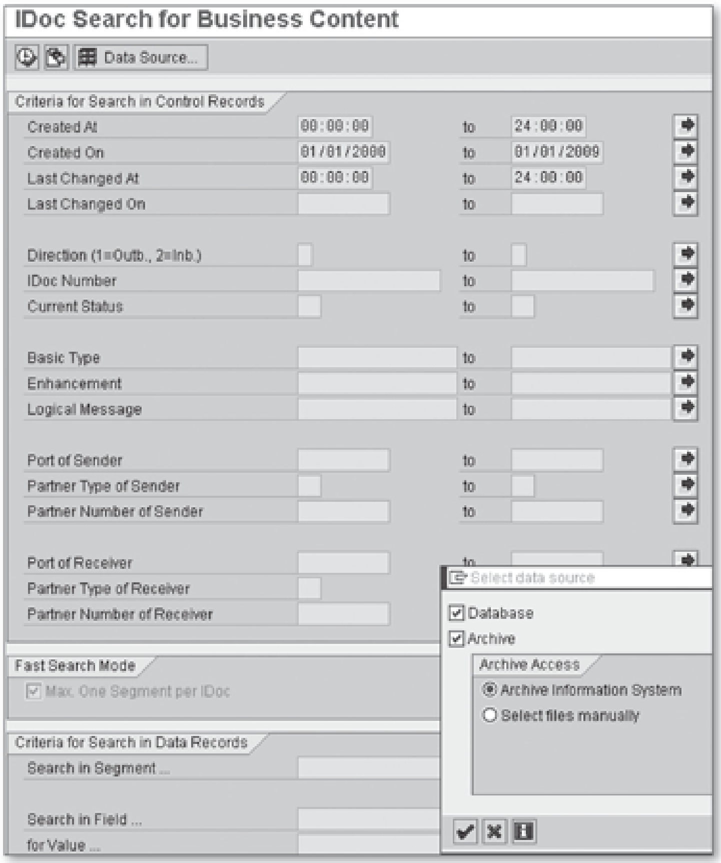
Figure 7.12 Archive Info Structure "SAP_IDOC_001"

Search for archived IDocs

You can use this info structure in Transaction SARI to search for archived IDocs. This can be done for all archived objects, although the result is simply a tabular view of the individual fields. For IDocs, the normal IDoc search in Transaction WE09 is structured so that you can choose whether to search in the archive, in the database, or in both. The advantage of this is that you can search for archived IDocs in the usual manner, and the data is displayed in the same way for database IDocs and archived IDocs. In other words, it doesn't make any difference where the data comes from. Click the Select Data Source button to select the data source (bottom right in Figure 7.13). The alternative of selecting files manually isn't recommended because you have to know which IDocs are located in which archive file.

Display IDocs from archive

Figure 7.14 shows an IDoc that was found in WE09 (517747). It has status *53*, which shows that it comes from the archive.



IDoc Search for Business Content

Data Source...

Criteria for Search in Control Records

Created At	00:00:00	to	24:00:00	➔
Created On	01/01/2000	to	01/01/2009	➔
Last Changed At	00:00:00	to	24:00:00	➔
Last Changed On		to		➔
Direction (1=Outb., 2=Inb.)		to		➔
IDoc Number		to		➔
Current Status		to		➔
Basic Type		to		➔
Enhancement		to		➔
Logical Message		to		➔
Port of Sender		to		➔
Partner Type of Sender		to		➔
Partner Number of Sender		to		➔
Port of Receiver		to		➔
Partner Type of Receiver				
Partner Number of Receiver				

Fast Search Mode

☑ Max. One Segment per IDoc

Criteria for Search in Data Records

Search in Segment ...

Search in Field ...

for Value ...

Select data source

☑ Database

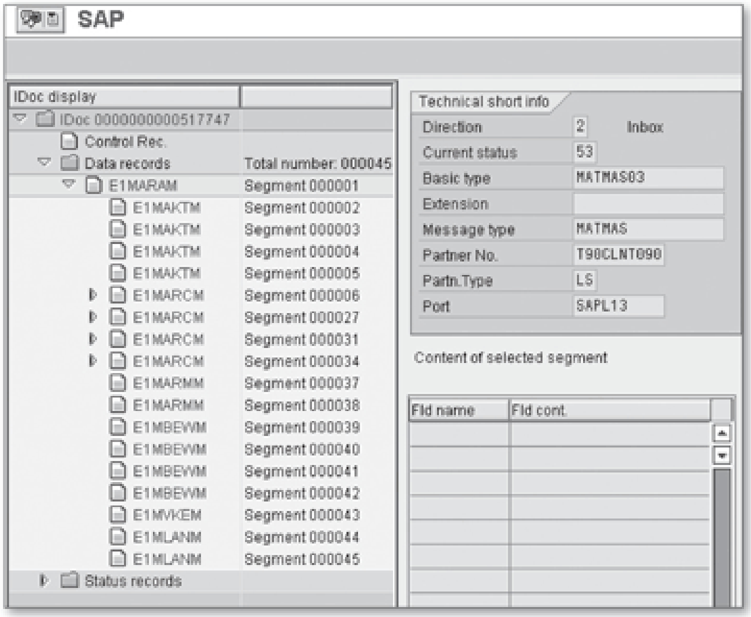
☑ Archive

Archive Access

● Archive Information System

○ Select files manually

Figure 7.13 IDoc Search for Field Content



SAP

IDoc display	
IDoc 0000000000517747	
Control Rec.	
Data records	Total number: 000045
E1MARAM	Segment 000001
E1MAKTM	Segment 000002
E1MAKTM	Segment 000003
E1MAKTM	Segment 000004
E1MAKTM	Segment 000005
E1MARCM	Segment 000006
E1MARCM	Segment 000027
E1MARCM	Segment 000031
E1MARCM	Segment 000034
E1MARMM	Segment 000037
E1MARMM	Segment 000038
E1MBEWM	Segment 000039
E1MBEWM	Segment 000040
E1MBEWM	Segment 000041
E1MBEWM	Segment 000042
E1MVKEM	Segment 000043
E1MLANM	Segment 000044
E1MLANM	Segment 000045
Status records	

Technical short info

Direction	2	Inbox
Current status	53	
Basic type	MATMAS03	
Extension		
Message type	MATMAS	
Partner No.	T90CLNT090	
Partn.Type	LS	
Port	SAPL13	

Content of selected segment

Fld name	Fld cont.

Figure 7.14 Individual IDoc Display in Transaction WE09

Transaction WE10 in older releases

Procedure in Older Releases

Up to Release 4.6C, Transaction WE09 read data from the database only. There was an additional transaction, WE10, for reading data from the archive. The two functions have been combined in one easy-to-use transaction in more recent releases.

Delete IDocs

Deleting IDocs

You'll sometimes have to delete IDocs. You can do so in Transaction WE11. However, whether or not you should be permitted to delete IDocs in a production system is a matter to be clarified with the person who is responsible for GDPdU (or equivalent legislation) in your organization because deleted data cannot be recovered.

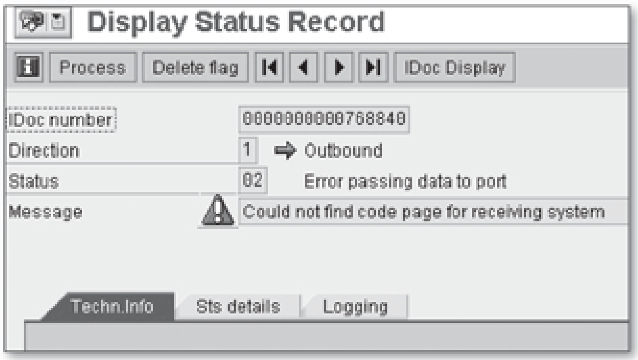
7.5 Status Conversion

There are two automatic options for status conversion in the SAP standard. Both options use SAP-specific IDocs, ALEAUD and SYSTAT. The ALEAUD IDoc assigns the status values *39*, *40*, or *41*, in accordance with the recipient's status, while the SYSTAT IDoc can assign any new status with its own failure or success text. Chapter 5, Confirmations, describes the functionality of both of these IDocs in detail. It's also possible to set IDocs that still contain errors to a status that prevents them from any further processing, in both inbound and outbound processing. This status also allows the affected IDocs to be archived. The corresponding status is 31 in outbound processing and 68 in inbound processing. Note that consistency is guaranteed on a system-wide basis only if the actions that should have been executed by the IDoc are executed in some other way (either by another IDoc that doesn't contain the error, or manually).

Manual delete flag

Setting the status values to mean Error, no further processing is done in the error workflow using the Delete Flag button (see Figure 7.15).

The status change itself is made after a security prompt and for only one IDoc at a time. There is no facility in the SAP standard for making mass changes to status values, nor is there likely to be such a facility. The next section contains a sample program for making mass changes.



IDoc number 0000000000768840
Direction 1 ⇨ Outbound
Status 02 Error passing data to port
Message ⚠ Could not find code page for receiving system

Figure 7.15 Setting the Delete Flag in the Workflow

Sample Program for Mass Changes

Mass changes

If you have the problem, particularly in test systems, that a large number of IDocs cannot be posted, you can set all of these IDocs to an archivable status at one time. Note, however, you must ensure that this is done using the original SAP tools and that the associated error workflow has been closed.

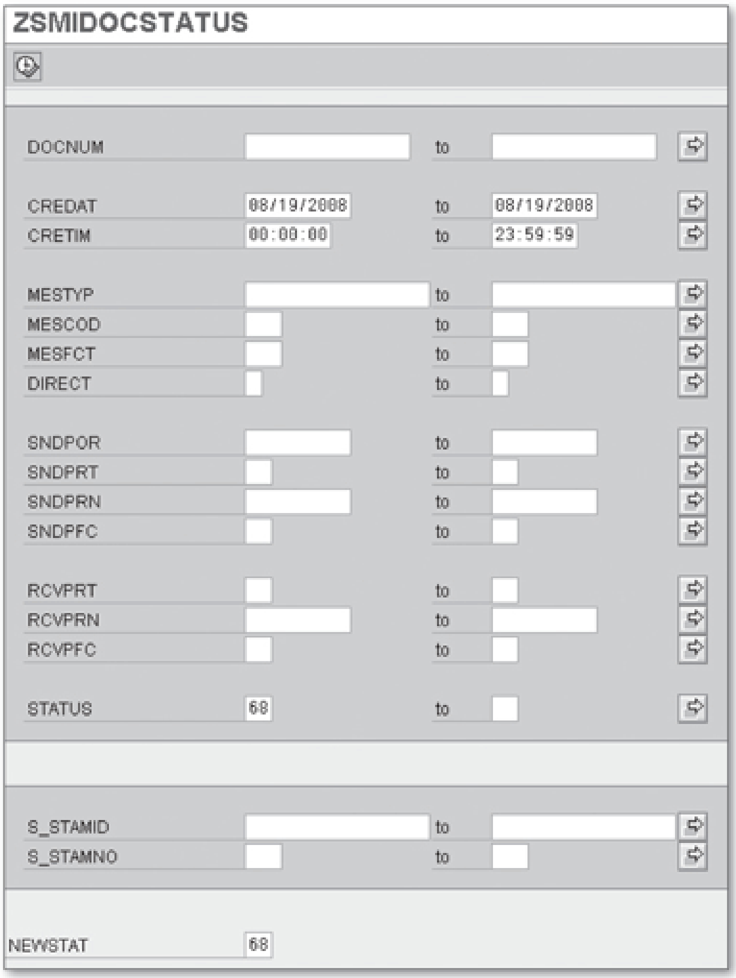
Because the program can also be used to set another status, it may be necessary to trigger an error workflow. This program represents a major intervention into the SAP system, so you should add a lot of high-quality comments to the code and secure it with a high level of authorization checks. Our sample program (see Listing 7.1) uses a status text that shows which user made the status change.

Notes on the Code

This code is intended as an example only and doesn't contain authorization checks. Also carefully consider whether you really want to make every status change shown in this example. One alternative would be to change to statuses *31* and *68* only. In particular, note that resetting a successful IDoc to a status that permits further processing can lead to duplicate postings and inconsistencies.

Selection fields

The program itself was programmed using the new *ABAP List Viewer* (ALV). This has the advantage that the display dynpro and the GUI status don't have to be created, which makes it much easier for you to work with this example. Figure 7.16 shows which selection fields you can use. Here, as before, it's of course up to you whether you incorporate additional requirements.



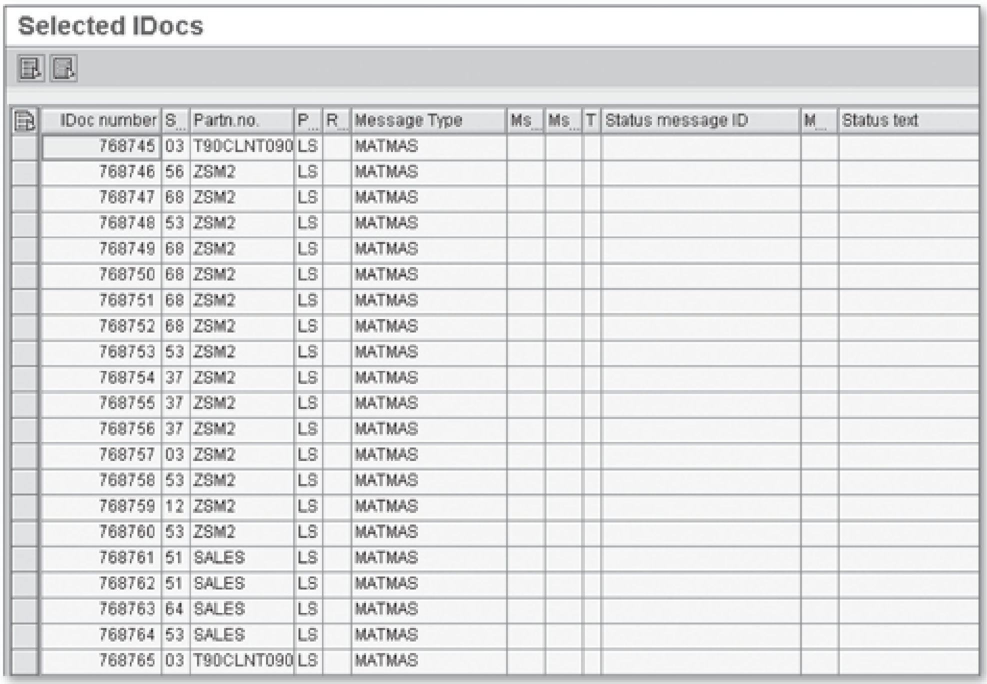
ZSMIDOCSTATUS

DOCNUM		to	
CREDAT	08/19/2008	to	08/19/2008
CRETIM	00:00:00	to	23:59:59
MESTYP		to	
MESCOD		to	
MESFCT		to	
DIRECT		to	
SNDPOR		to	
SNDPRT		to	
SNDPRN		to	
SNDPFC		to	
RCVPRT		to	
RCVPRN		to	
RCVPFC		to	
STATUS	68	to	
S_STAMID		to	
S_STAMNO		to	
NEWSTAT	68		

Figure 7.16 Initial Dynpro of the Sample Program

Selection list

The input from Figure 7.16 is used to list all of the possible IDocs (see Figure 7.17). You can limit the IDocs by placing checkmarks in the appropriate selection column. Then, without a button, when every action is executed, the list of selected IDocs is converted. You can also apply special controls to this conversion — using a button, for example. In general, this can be done with both the old and the new ALV.



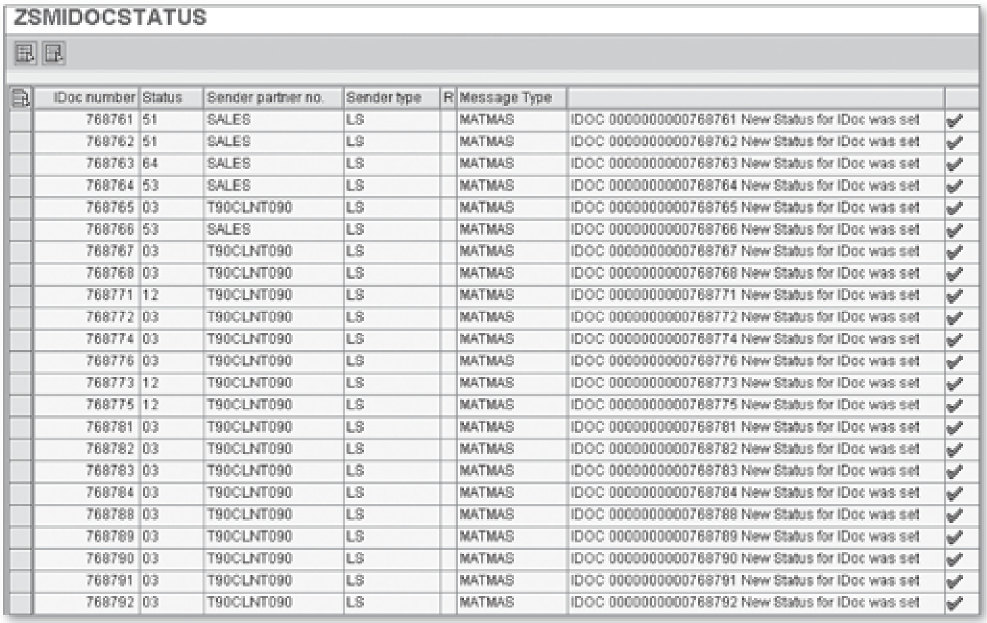
Selected IDocs

IDoc number	S...	Partn.no.	P...	R...	Message Type	Ms...	Ms...	T	Status message ID	M...	Status text
768745	03	T90CLNT090	LS		MATMAS						
768746	56	ZSM2	LS		MATMAS						
768747	68	ZSM2	LS		MATMAS						
768748	53	ZSM2	LS		MATMAS						
768749	68	ZSM2	LS		MATMAS						
768750	68	ZSM2	LS		MATMAS						
768751	68	ZSM2	LS		MATMAS						
768752	68	ZSM2	LS		MATMAS						
768753	53	ZSM2	LS		MATMAS						
768754	37	ZSM2	LS		MATMAS						
768755	37	ZSM2	LS		MATMAS						
768756	37	ZSM2	LS		MATMAS						
768757	03	ZSM2	LS		MATMAS						
768758	53	ZSM2	LS		MATMAS						
768759	12	ZSM2	LS		MATMAS						
768760	53	ZSM2	LS		MATMAS						
768761	51	SALES	LS		MATMAS						
768762	51	SALES	LS		MATMAS						
768763	64	SALES	LS		MATMAS						
768764	53	SALES	LS		MATMAS						
768765	03	T90CLNT090	LS		MATMAS						

Figure 7.17 Results of the IDoc Search

Results list

The list shown in Figure 7.18 shows the results. In our example, all of the IDocs were converted successfully.

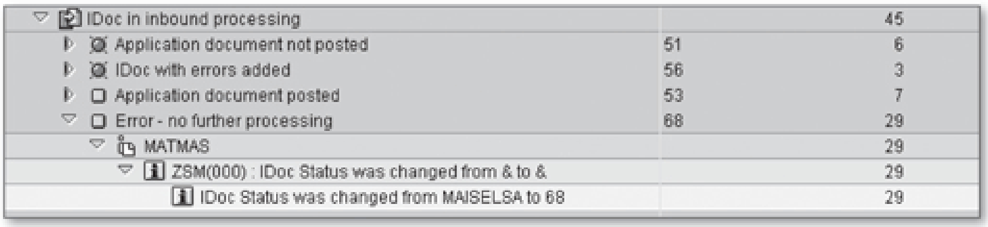


ZSMIDOCSTATUS

IDoc number	Status	Sender partner no.	Sender type	R	Message Type		
768761	51	SALES	LS		MATMAS	IDOC 0000000000768761 New Status for IDoc was set	✓
768762	51	SALES	LS		MATMAS	IDOC 0000000000768762 New Status for IDoc was set	✓
768763	64	SALES	LS		MATMAS	IDOC 0000000000768763 New Status for IDoc was set	✓
768764	53	SALES	LS		MATMAS	IDOC 0000000000768764 New Status for IDoc was set	✓
768765	03	T90CLNT090	LS		MATMAS	IDOC 0000000000768765 New Status for IDoc was set	✓
768766	53	SALES	LS		MATMAS	IDOC 0000000000768766 New Status for IDoc was set	✓
768767	03	T90CLNT090	LS		MATMAS	IDOC 0000000000768767 New Status for IDoc was set	✓
768768	03	T90CLNT090	LS		MATMAS	IDOC 0000000000768768 New Status for IDoc was set	✓
768771	12	T90CLNT090	LS		MATMAS	IDOC 0000000000768771 New Status for IDoc was set	✓
768772	03	T90CLNT090	LS		MATMAS	IDOC 0000000000768772 New Status for IDoc was set	✓
768774	03	T90CLNT090	LS		MATMAS	IDOC 0000000000768774 New Status for IDoc was set	✓
768776	03	T90CLNT090	LS		MATMAS	IDOC 0000000000768776 New Status for IDoc was set	✓
768773	12	T90CLNT090	LS		MATMAS	IDOC 0000000000768773 New Status for IDoc was set	✓
768775	12	T90CLNT090	LS		MATMAS	IDOC 0000000000768775 New Status for IDoc was set	✓
768781	03	T90CLNT090	LS		MATMAS	IDOC 0000000000768781 New Status for IDoc was set	✓
768782	03	T90CLNT090	LS		MATMAS	IDOC 0000000000768782 New Status for IDoc was set	✓
768783	03	T90CLNT090	LS		MATMAS	IDOC 0000000000768783 New Status for IDoc was set	✓
768784	03	T90CLNT090	LS		MATMAS	IDOC 0000000000768784 New Status for IDoc was set	✓
768788	03	T90CLNT090	LS		MATMAS	IDOC 0000000000768788 New Status for IDoc was set	✓
768789	03	T90CLNT090	LS		MATMAS	IDOC 0000000000768789 New Status for IDoc was set	✓
768790	03	T90CLNT090	LS		MATMAS	IDOC 0000000000768790 New Status for IDoc was set	✓
768791	03	T90CLNT090	LS		MATMAS	IDOC 0000000000768791 New Status for IDoc was set	✓
768792	03	T90CLNT090	LS		MATMAS	IDOC 0000000000768792 New Status for IDoc was set	✓

Figure 7.18 Results List After Status Value Conversion

In Transaction BD87, if the basic view is open, you can see the generic error text with placeholders (see Figure 7.19).

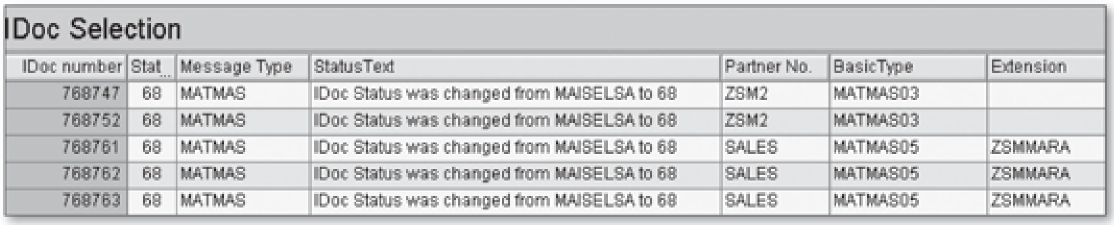


```
▽ IDoc in inbound processing                                        45
    ▷ Application document not posted                  51            6
    ▷ IDoc with errors added                           56            3
    ▷ Application document posted                      53            7
    ▽ Error - no further processing                    68           29
        ▽ MATMAS                                                    29
            ▽ ZSM(000) : IDoc Status was changed from & to &        29
                IDoc Status was changed from MAISELSA to 68         29
```

Figure 7.19 Message Text in the Basic List

Converted IDoc

In the individual view of an IDoc, you can see exactly who made the conversion (see Figure 7.20).



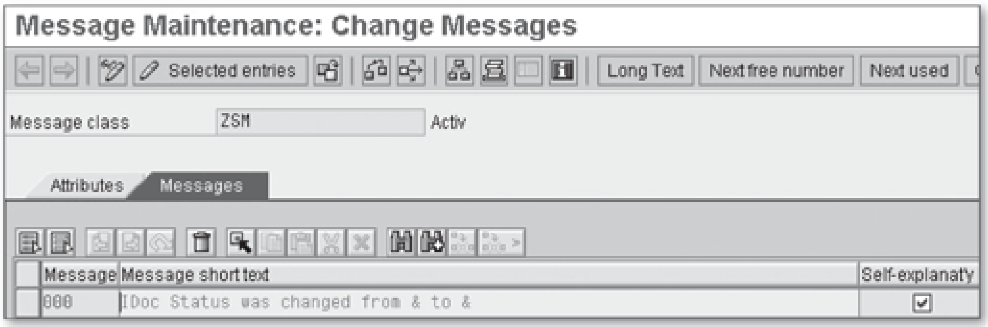
IDoc Selection

IDoc number	Stat...	Message Type	StatusText	Partner No.	BasicType	Extension
768747	68	MATMAS	IDoc Status was changed from MAISELSA to 68	ZSM2	MATMAS03	
768752	68	MATMAS	IDoc Status was changed from MAISELSA to 68	ZSM2	MATMAS03	
768761	68	MATMAS	IDoc Status was changed from MAISELSA to 68	SALES	MATMAS05	ZSMMARA
768762	68	MATMAS	IDoc Status was changed from MAISELSA to 68	SALES	MATMAS05	ZSMMARA
768763	68	MATMAS	IDoc Status was changed from MAISELSA to 68	SALES	MATMAS05	ZSMMARA

Figure 7.20 Message Text in the IDoc Individual View

Message class

For the example program used here, a message class with the most important messages has to be set up. Figure 7.21 shows the text used for this purpose.



Message Maintenance: Change Messages

Message class ZSM Activ

Message	Message short text	Self-explanat'y
000	IDoc Status was changed from & to &	☑

Figure 7.21 Message Class of the Sample Program

Listing 7.1 contains the basic code required for the examples described in this chapter. Again, note that only the code that is relevant to IDocs is given here, and the code shouldn't be used in unmodified form; authorization checks, for example, should be added. Also, the program contains text elements that you'll have to add, such as `text-b01` for one of the blocks on the selection screen. Where these don't appear on the selection screen, they are indicated in the code in each case — for example, `'IDoc could not be locked' (001)` — and so no further screenshots are used here.

```
*&---------------------------------------------------------------*
*& Report  ZSMIDOCSTATUS
*&---------------------------------------------------------------*
REPORT  ZSMIDOCSTATUS .
INCLUDE <icon> .
TABLES: edidc .
TABLES:
  edids,   tbd05 .
DATA: wa_tbd52        TYPE tbd52 .
DATA: wa_edp21        TYPE edp21 .
DATA: partner         TYPE edidc-sndprn,
      nrlines         TYPE sy-tabix .
Data: alv             TYPE REF TO cl_salv_table .
DATA: lr_selections   TYPE REF TO cl_salv_selections .
DATA: i_Selected_rows type salv_t_row .
DATA: wa_row          type line of salv_t_row .
Data: exc             TYPE REF TO cx_root .
DATA: msg             TYPE string .
```

Type definitions

```
Types: BEGIN OF r_edidc,
        docnum LIKE edidc-docnum,
        status LIKE edidc-status,
        sndprn LIKE edidc-sndprn,
        sndprt LIKE edidc-sndprt,
        sndpfc LIKE edidc-sndpfc,
        mestyp LIKE edidc-mestyp,
        mescod LIKE edidc-mescod,
        mesfct LIKE edidc-mesfct,
        test   LIKE edidc-test,
        stamid LIKE edids-stamid,
        stamno LIKE edids-stamno,
        statxt LIKE edids-statxt,
        msg(80),
        icon(4),
        box(1),
      END OF r_edidc .
```

```
DATA: i_edidc  type table of r_edidc .
data: wa_edidc type r_edidc .
DATA: idoc_status LIKE bdidocstat OCCURS 0 WITH HEADER LINE .
DATA: c_event_object_type_idocappl LIKE
      swetypecou-objtype VALUE ‘IDOCAPPL’ .
DATA: c_event_err_process_completed
      LIKE swetypecou-event VALUE ‘ERRORPROCESSCOMPLETD’ .
CONSTANTS: c_wf_result_delete_idoc LIKE bdwf_param-result
VALUE ‘99998’,
           c_wf_result_wi_complete LIKE bdwf_param-result
      VALUE ‘99997’ .
 Data: g_repid LIKE sy-repid,
       g_save(1) TYPE c VALUE ‘A’,
       g_variant LIKE disvariant .
DATA:
  $idoc_ges(6)    TYPE n,
  $idoc_ok(6)     TYPE n,
  $idoc_fehler(6) TYPE n .
* Select IDocs to be modified with pre-selection
```

Selection screen structure

```
SELECTION-SCREEN BEGIN OF BLOCK 01 WITH FRAME TITLE text-b01 .
SELECT-OPTIONS: docnum FOR edidc-docnum .
SELECTION-SCREEN SKIP 1 .
SELECT-OPTIONS: credat FOR edidc-credat DEFAULT sy-datum TO sy-
datum .
SELECT-OPTIONS: cretim FOR edidc-cretim DEFAULT ‘000000’ TO
                                                ‘235959’ .
SELECT-OPTIONS: mestyp FOR edidc-mestyp .
SELECT-OPTIONS: mescod FOR edidc-mescod .
SELECT-OPTIONS: mesfct FOR edidc-mesfct .
SELECT-OPTIONS: direct FOR edidc-direct .
SELECT-OPTIONS: sndpor FOR tbd05-sndsystem .
SELECT-OPTIONS: sndprt FOR edidc-sndprt .
SELECT-OPTIONS: sndprn FOR edidc-sndprn .
SELECT-OPTIONS: sndpfc FOR edidc-sndpfc .
SELECT-OPTIONS: rcvprt FOR edidc-rcvprt .
SELECT-OPTIONS: rcvprn FOR edidc-rcvprn .
SELECT-OPTIONS: rcvpfc FOR edidc-rcvpfc .
SELECT-OPTIONS: status FOR edidc-status DEFAULT ‘68’ .
SELECTION-SCREEN END OF BLOCK 01 .
SELECTION-SCREEN BEGIN OF BLOCK 02 WITH FRAME TITLE text-b02 .
SELECT-OPTIONS:
  s_stamid FOR edids-stamid,
  s_stamno FOR edids-stamno .
```

```
SELECTION-SCREEN END OF BLOCK 02 .
SELECTION-SCREEN SKIP 1 .
PARAMETERS: newstat LIKE edidc-status DEFAULT ‘68’ .
START-OF-SELECTION .
```

Data selection

```
  SELECT: docnum status sndprn sndprt sndpfc mestyp mescod
          mesfct test
          INTO corresponding fields of table i_edidc
          FROM edidc     WHERE docnum IN docnum AND
                               upddat IN credat AND
                               updtim IN cretim AND
                               mestyp IN mestyp AND
                               mescod IN mescod AND
                               mesfct IN mesfct AND
                               sndpor IN sndpor AND
                               sndprt IN sndprt AND
                               sndprn IN sndprn AND
                               sndpfc IN sndpfc AND
                               direct IN direct AND
                               rcvprt IN direct AND
                               rcvpfc IN direct AND
                               rcvprn IN direct AND
                               status IN status .
  PERFORM get_edids .
END-OF-SELECTION .
  PERFORM alv .
*-----------------------------------------------------------*
* Change status
*-----------------------------------------------------------*
FORM status_change .
```

Change status

```
  i_selected_rows = LR_Selections->GET_SELECTED_ROWS( ) .
  loop at i_selected_rows into wa_row .
  read table i_edidc into wa_edidc index wa_row .
*   IDOC processing
```

Set lock

```
    CALL FUNCTION ‘ENQUEUE_ES_EDIDOCE’
         EXPORTING
              mode_edidc     = ‘E’
              mandt          = sy-mandt
              docnum         = wa_edidc-docnum
         EXCEPTIONS
              foreign_lock   = 1
              system_failure = 2
              OTHERS         = 3 .
```

```
IF sy-subrc <> 0 .
  CONCATENATE ‘IDOC’(003) wa_edidc-docnum
              ‘IDoc could not be locked’ (001)
              INTO wa_edidc-msg SEPARATED BY space .
  wa_edidc-icon = icon_failure .
ELSE .
  CLEAR: idoc_status .
  REFRESH: idoc_status .
  MOVE wa_edidc-docnum   TO idoc_status-docnum .
  MOVE newstat           TO idoc_status-status .
  MOVE ‘S’               TO idoc_status-msgty .
  MOVE ‘ZSM’             TO idoc_status-msgid .
  MOVE ‘000’             TO idoc_status-msgno .
  MOVE sy-uname          TO idoc_status-msgv1 .
  MOVE newstat           TO idoc_status-msgv2 .
  MOVE sy-uname          TO idoc_status-uname .
  MOVE sy-repid          TO idoc_status-repid .
  APPEND idoc_status .
  CALL FUNCTION ‘IDOC_STATUS_WRITE_TO_DATABASE’
       EXPORTING
            idoc_number               = wa_edidc-docnum
       TABLES
            idoc_status               = idoc_status
       EXCEPTIONS
            idoc_foreign_lock         = 1
            idoc_not_found            = 2
            idoc_status_records_empty = 3
            idoc_status_invalid       = 4
            db_error                  = 5 .

  IF sy-subrc EQ 0  .
    CONCATENATE ‘IDOC’(003) wa_edidc-docnum
       ‘New status set for IDoc’(002)
       INTO wa_edidc-msg SEPARATED BY space .
    wa_edidc-icon = icon_checked .
  ELSE .
    CONCATENATE ‘IDOC’(003) wa_edidc-docnum
       ‘Status conversion error’(004)
        INTO wa_edidc-msg SEPARATED BY space .
    wa_edidc-icon = icon_failure .
    MODIFY i_edidc from wa_edidc index wa_row .
    CONTINUE .
  ENDIF .
```

```
      CASE newstat .
        WHEN ‘68’ .
* IDoc was set to a status that does not permit further
* processing; therefore, trigger event that ends the
* error workflow.
SELECT SINGLE * FROM edp21 INTO wa_edp21
       WHERE sndprn = wa_edidc-sndprn
         AND sndprt = wa_edidc-sndprt
         AND sndpfc = wa_edidc-sndpfc
         AND mestyp = wa_edidc-mestyp
         AND mescod = wa_edidc-mescod
         AND mesfct = wa_edidc-mesfct
         AND test   = wa_edidc-test .
IF sy-subrc <> 0 .
CONCATENATE ‘IDOC’(003) wa_edidc-docnum
             ‘for’(005) ‘message’(008) edidc-sndprn
             edidc-sndprt edidc-sndpfc edidc-mestyp
             ‘no entry found in EDP21.’(006)
             INTO wa_edidc-msg SEPARATED BY space .
ELSE .
  SELECT SINGLE * FROM tbd52 INTO wa_tbd52
         WHERE evcode = wa_edp21-evcode .
IF sy-subrc <> 0 .
   CONCATENATE ‘IDOC’(003) wa_edidc-docnum ‘for’(005)
     ‘process code’(009) wa_edp21-evcode
     ‘no entry found in TBD52.’(007)
     INTO wa_edidc-msg SEPARATED BY space .
ELSE .
```

Start workflows, if required

```
PERFORM start_event(saplbd20) USING wa_edidc-docnum
        wa_tbd52-event_end  wa_tbd52-idocobjtyp
        c_wf_result_delete_idoc .
COMMIT WORK .
PERFORM start_event(saplbd20) USING wa_edidc-docnum
        wa_tbd52-event_end wa_tbd52-idocobjtyp
        c_wf_result_wi_complete .
COMMIT WORK .
ENDIF .
ENDIF .
```

End workflow

```
        WHEN ‘31’ .
          PERFORM event_for_task_end_create(saplbd16)
                  USING wa_edidc-docnum .
          COMMIT WORK .
        WHEN OTHERS .
```

```
      ENDCASE .
      CALL FUNCTION ‘DEQUEUE_ES_EDIDOCE’
           EXPORTING
                mode_edidc = ‘E’
                mandt      = sy-mandt
                docnum     = wa_edidc-docnum .
    ENDIF .
    MODIFY i_edidc from wa_edidc index wa_row .
  ENDLOOP .
  IF sy-subrc = 0 .
*  Entries were selected and processed
    CLEAR: $idoc_ok, $idoc_error .
    LOOP AT i_edidc TRANSPORTING NO FIELDS
                    WHERE icon = icon_checked .
      ADD 1 TO $idoc_ok .
    ENDLOOP .
    LOOP AT i_edidc TRANSPORTING NO FIELDS
                    WHERE icon = icon_failure .
      ADD 1 TO $idoc_error .
    ENDLOOP .
   ENDIF .
ENDFORM .
*----------------------------------------------------------*
* Retrieve data
*----------------------------------------------------------*
FORM get_edids .
  DATA:    h_str(150),
           it_edids TYPE TABLE OF edids,
           wa_edids LIKE edids .
  CHECK NOT s_stamno IS INITIAL .
  LOOP AT i_edidc into wa_edidc .
    REFRESH it_edids .
```

Get status values

```
    SELECT * FROM edids APPENDING TABLE it_edids
                        WHERE docnum = wa_edidc-docnum
                        AND   status = wa_edidc-status
                        AND   stamid IN s_stamid
                        AND   stamno IN s_stamno .
    IF sy-subrc NE 0 .
      DELETE i_edidc .
      CONTINUE .
    ENDIF .
    SORT it_edids BY docnum
                     logdat DESCENDING
                     logtim DESCENDING
                     countr DESCENDING .
    DELETE ADJACENT DUPLICATES FROM it_edids
           COMPARING docnum .
```

Set texts

```
READ TABLE it_edids INDEX 1 INTO wa_edids .
    wa_edidc-stamid = wa_edids-stamid .
    wa_edidc-stamno = wa_edids-stamno .
    h_str           = wa_edids-statxt .
    REPLACE '&' WITH wa_edids-stapa1 INTO h_str .
    CONDENSE h_str .
    REPLACE '&' WITH wa_edids-stapa2 INTO h_str .
    CONDENSE h_str .
    REPLACE '&' WITH wa_edids-stapa3 INTO h_str .
    CONDENSE h_str .
    REPLACE '&' WITH wa_edids-stapa4 INTO h_str .
    CONDENSE h_str .
    wa_edidc-statxt = h_str .
    MODIFY i_edidc from wa_edidc .
  ENDLOOP .
ENDFORM .
```

Title ALV

```
*---------------------------------------------------------*
* Title ALV
*---------------------------------------------------------*
FORM title USING title TYPE lvc_title .
  DATA settings TYPE REF TO cl_salv_display_settings .
  settings = alv->get_display_settings( ) .
  settings->set_list_header( titel ) .
ENDFORM .
```

ALV selection options

```
*---------------------------------------------------------*
* ALV selection options
*---------------------------------------------------------*
FORM set_selections  USING p_alv TYPE REF TO cl_salv_table .
* get the SELECTIONS object
  lr_selections = p_alv->get_selections( ) .
* set the selection mode
  lr_selections->set_selection_mode(
      value  = if_salv_c_selection_mode=>cell ) .
ENDFORM .                    " set_selections
*---------------------------------------------------------*
* ALV output
*---------------------------------------------------------*
```

Get ALV reference

```
FORM alv .
    TRY .
```

```
      cl_salv_table=>factory(
        IMPORTING
          r_salv_table   = alv
        CHANGING
          t_table        = i_edidc ) .
    CATCH cx_salv_msg INTO exc .
      msg = exc->get_text( ) .
      MESSAGE msg TYPE 'A' .
  ENDTRY .
  PERFORM title           USING 'Selected IDocs' .
  perform set_selections  USING alv .
  alv->display( ) .
  perform status_change .
  alv->display( ) .
ENDFORM .
```

Listing 7.1 Status Conversion of Selected IDocs

7.6 Summary

This chapter gave you an overview of administrative tasks that are related to IDocs. Depending on the structure of your company, some of this work might be done by a special department, but you should know which work has to be done. We've also included some hints for daily work and error handling.

This chapter provides some tips on using SAP NetWeaver Process Integration to send or receive IDocs.

8 IDocs in Conjunction with SAP NetWeaver Process Integration

Communication in SAP NetWeaver Process Integration (SAP NetWeaver PI) using IDocs is implemented via the IDoc adapter. The requirement in the creation of the IDoc adapter in SAP NetWeaver PI was that no difference may arise for the backend systems if they communicate with SAP NetWeaver PI instead of with another backend system. For IDoc developers, this means that actually nothing needs to be done — nice, right? But it's important to know what happens in SAP NetWeaver PI so this rule can be followed.

Business systems in SAP NetWeaver PI

As you've learned in the course of this book, IDocs work with logical systems or with partner roles. In SAP NetWeaver PI, you work with business systems that approximately have the same meaning. One difference, however, is that while partners and logical systems are limited to 10 characters, business systems can have a length of 256 characters. In SAP NetWeaver PI, a conversion from the logical system to the business system and vice versa occurs. The following sections describe the conversion options.

8.1 Conversion of Logical Systems to Business Systems

"SNDPOR" field in SAP systems

In the SNDPOR field of the control record of an IDoc, SAP systems always send the combination of `SAP<SID>`, for example, `SAPJ00` for an SAP system that is called J00. In the MANDT field, you then have the client from which the IDoc was generated.

SLD

In the SLD (*System Landscape Directory*) of SAP NetWeaver PI, you initially register the SAP systems as a technical system with their system ID, or they are created manually. All available clients are known then, and the logical system name from the client management is adopted or maintained for all clients.

Reference to the logical system

When you create a business system, you then reference exactly one client so that the logical system name is automatically determined for each business system. If an IDoc is received, you use the system ID from SNDPOR to determine the SAP system and the client to determine the business system. This client is then used as the sender for further processing.

The SNDPRN field for the name of the logical system, which also exists in the sending system, isn't considered. If the sending system isn't an SAP system, the value is read from the SNDPOR field, too. In this case, a name that matches SNDPOR is searched among the logical system names that are assigned to the business systems, and then the associated business system is used.

This means that non-SAP partners must possibly prepare your IDocs differently if they are processed via SAP NetWeaver PI. If the sending is directly to an SAP system, the SNDPRN field would be relevant. Ideally, your partner sends the same name in both fields. For IDocs that leave SAP NetWeaver PI, the SNDPRN field is populated according to the partner's requirements.

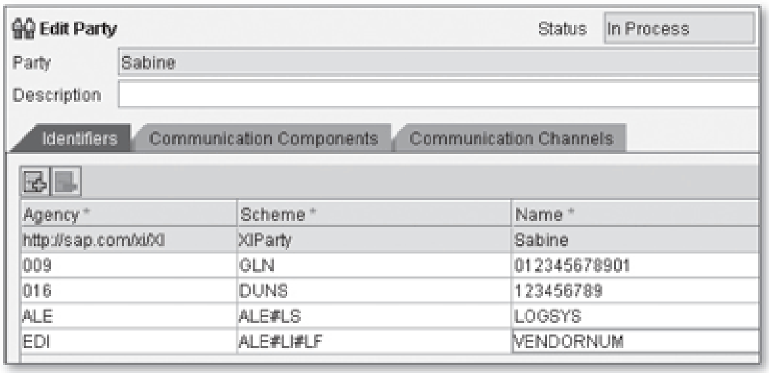
8.2 Conversion of IDoc Partner Roles

Parties in SAP NetWeaver PI

If you don't communicate with a logical system but with a party outside your enterprise, for example, with a customer or a vendor, the sender and receiver data in the IDoc is also converted into the parties of SAP NetWeaver PI. This is done using the *Party* object that is available there (see Figure 8.1).

Partner conversion

The agency is specified as *http://sap.com/xi/XI* for the actual name in SAP NetWeaver PI. For EDI conversions it's specified with the values "016" and "009" and can be selected as such. For other values, you can assign the agency itself. If an IDoc is received, the value for the scheme is `ALE#<partner type>#partner role>`.



Status: In Process
Party: Sabine
Description:

Agency *	Scheme *	Name *
http://sap.com/xi/XI	XIParty	Sabine
009	GLN	012345678901
016	DUNS	123456789
ALE	ALE#LS	LOGSYS
EDI	ALE#LI#LF	VENDORNUM

Figure 8.1 Parties in SAP NetWeaver PI

So for a logical system as the sender, "ALE#LS" is specified in the Scheme field because no partner roles are known there, and the Name field includes the logical system name. If you have a vendor in the role of the vendor, "ALE#LI#LF" is entered in the Scheme field, and the SAP vendor number from Transaction WE20 is in the Name field. SAP NetWeaver PI then converts it into the SAP NetWeaver PI-specific partner number, in this case, "SABINE."

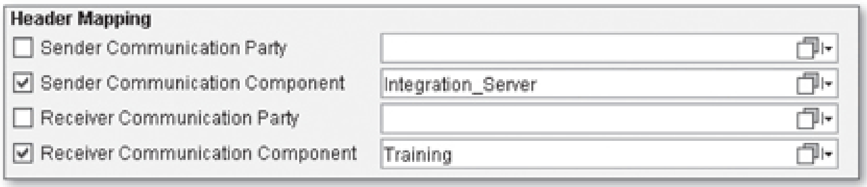
8.3 Header Mapping

Communication channel

In the outbound processing, the IDoc adapter requires a communication channel, which is called a *receiver communication channel*. The naming is from the business system perspective and not from the SAP NetWeaver PI perspective with regard to the directions. The communication channel knows how it can reach the receiver of the IDoc. A receiver agreement specifies which communication channel is supposed to be used for which message type.

Receiver agreement

In the receiver agreement, you now have the option to have the system perform a *header mapping*. For this purpose, a value from SAP NetWeaver PI is specified for the sender or receiver components or for partner or communication components. In this example (see Figure 8.2), this was converted for the two communication components. SAP NetWeaver PI then searches the appropriate logical system instead of the actual sender or receiver and writes it to the control record of the IDoc.



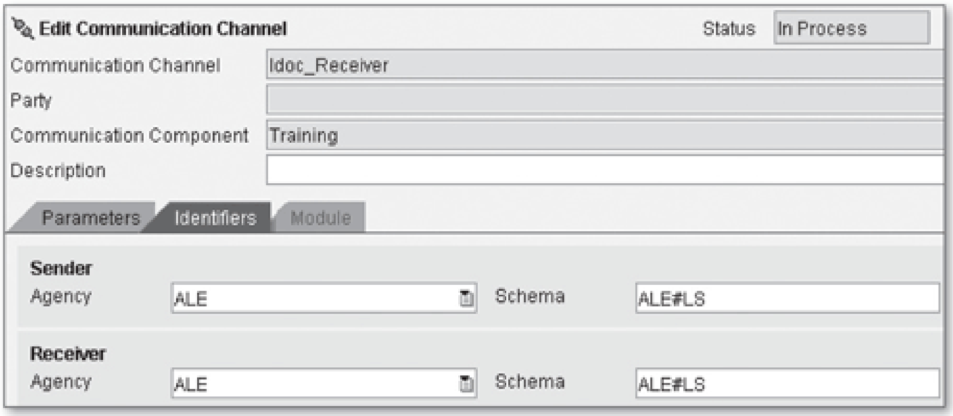
Header Mapping

□ Sender Communication Party	
☑ Sender Communication Component	Integration_Server
□ Receiver Communication Party	
☑ Receiver Communication Component	Training

Figure 8.2 Header Mapping in the Receiver Agreement

xPath

When you click on Advanced during editing, instead of a predefined sender from the SLD, you can specify an *xPath* that points to a specific field in the payload. Then, you must also specify in the associated communication channel how this field is supposed to be sent. The syntax corresponds to the party definition. Figure 8.3 shows an example.



Edit Communication Channel

Status: In Process

Communication Channel: Idoc_Receiver

Party:

Communication Component: Training

Description:

Sender

Agency: ALE Schema: ALE#LS

Receiver

Agency: ALE Schema: ALE#LS

Figure 8.3 Suitable Identifiers in the xPath Variant

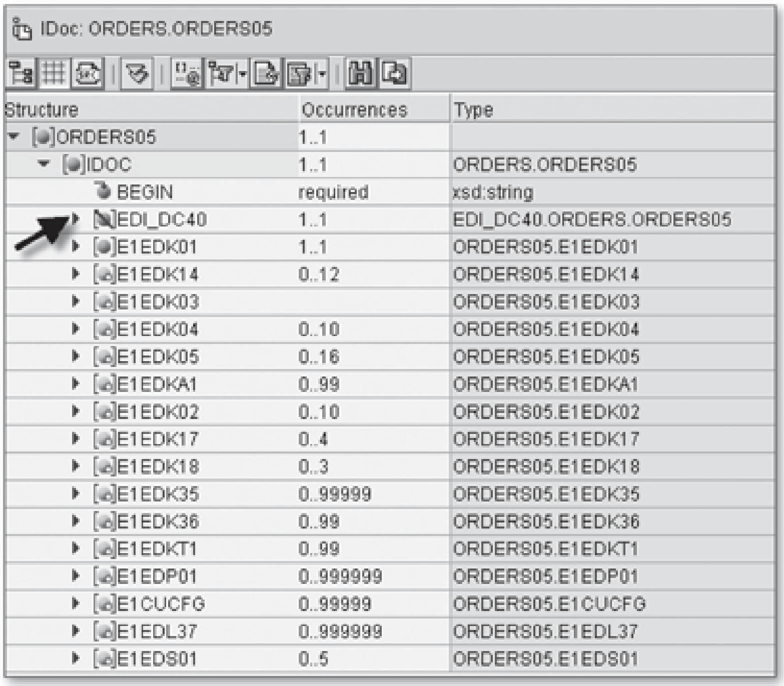
8.4 Handling the Control Record in SAP NetWeaver PI

Control record in mapping

In SAP NetWeaver PI, you can process the control record in the mapping yourself or have it created by SAP NetWeaver PI. In mapping, you can label the corresponding node as irrelevant for mapping (see Figure 8.4).

Deactivating nodes in the mapping

Even if the corresponding node is a mandatory segment or contains mandatory fields, the mapping can now be activated without having implemented an assignment. The crossed-through node is displayed in green so that it's apparent that everything is okay here.



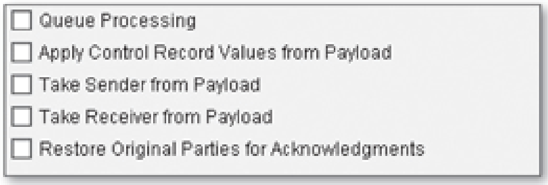
IDoc: ORDERS.ORDERS05

Structure	Occurrences	Type
ORDERS05	1..1	
IDOC	1..1	ORDERS.ORDERS05
BEGIN	required	xsd:string
EDI_DC40	1..1	EDI_DC40.ORDERS.ORDERS05
E1EDK01	1..1	ORDERS05.E1EDK01
E1EDK14	0..12	ORDERS05.E1EDK14
E1EDK03		ORDERS05.E1EDK03
E1EDK04	0..10	ORDERS05.E1EDK04
E1EDK05	0..16	ORDERS05.E1EDK05
E1EDKA1	0..99	ORDERS05.E1EDKA1
E1EDK02	0..10	ORDERS05.E1EDK02
E1EDK17	0..4	ORDERS05.E1EDK17
E1EDK18	0..3	ORDERS05.E1EDK18
E1EDK35	0..99999	ORDERS05.E1EDK35
E1EDK36	0..99	ORDERS05.E1EDK36
E1EDKT1	0..99	ORDERS05.E1EDKT1
E1EDP01	0..999999	ORDERS05.E1EDP01
E1CUCFG	0..99999	ORDERS05.E1CUCFG
E1EDL37	0..999999	ORDERS05.E1EDL37
E1EDS01	0..5	ORDERS05.E1EDS01

Figure 8.4 Ignoring the Control Record in the Mapping

Settings in the IDoc communication channel

You can also make some settings in the receiver communication channel (see Figure 8.5). For example, if you want to use the IDoc serialization via qRFC, which has not been available until SAP NetWeaver 6.40, you can set the Queue Processing flag. The target system then calls the `IDOC_INBOUND_IN_QUEUE` function module instead of the `IDOC_INBOUND_ASYNCHRONOUS` function module.



□ Queue Processing
□ Apply Control Record Values from Payload
□ Take Sender from Payload
□ Take Receiver from Payload
□ Restore Original Parties for Acknowledgments

Figure 8.5 IDoc Receiver Communication Channel

Control record from payload

You set the Apply Control Record Values from Payload flag if you populate the control record in mapping yourself. If you don't populate the control record in mapping at all (as in the previous example with the crossed-through control record), this flag must never be set. However, if you want to use parts of the payload to determine the party, you must set this flag, and the control record must be populated accordingly in mapping.

Instead of a conversion of the senders and receivers, you can take them directly from payload if the corresponding flags are set. Of course, this saves time because the conversion of an IDoc into XML is time-consuming. Finally, you can also restore the original parties for sending acknowledgments instead of converting them in SAP NetWeaver PI.

8.5 Updating IDocs Directly in SAP NetWeaver PI

SAP NetWeaver PI as the actual IDoc receiver

If you want to send an IDoc directly to SAP NetWeaver PI and if it's supposed to be updated in the local database, that is, if SAP NetWeaver PI is the actual data receiver and doesn't just assume the typical mediator role, you can use the `IDXIDOCINB` table. All IDocs that are entered in this table are updated by SAP NetWeaver PI itself instead of forwarding them.

You can insert IDocs using the `IDX_SELECT_IDOCTYP_WITHOUT_IS` report. You can remove already-entered IDocs using `IDX_DELETE_IDOCTYP_WITHOUT_IS`.

8.6 Summary

This chapter has described what you have to consider when you use IDocs in SAP NetWeaver PI, and which settings facilitate your work. Having IDoc interfaces already in all SAP Releases as well as the structure of IDocs where control information is part of the message itself lead to some special IDoc functionality inside of SAP NetWeaver PI.

The Author

Sabine Maisel has worked as an SAP consultant since 1996, having gained her initial SAP R/3 experience as an employee of a beta customer of SAP. She also holds training courses for SAP in Switzerland and Germany on all aspects of SAP NetWeaver interface technologies and the SAP Basis. In addition, Sabine has undertaken development work for the AFS (*Apparel and Footwear Solution*) industry solution and support work for customer projects. Her specializations are interface technologies, SAP NetWeaver PI/XI, ABAP development, and warehouse management.

Index

A

ABAP List Viewer → ALV, 134
Acknowledgment, 187
Agency, 244
Agent assignment, 184
ALE, 15
  business process, 15, 18
  communication layer, 35
  Customizing → SALE, 31
  scenario, 15, 18
  service, 45, 73, 84, 134
ALE00001, 119
ALEAUD, 187, 230
ALE interface
  for BAPI, 173
  for function modules, 174
ALE_MODEL_DETERMINE_IF_ TO_SEND, 155
ALE_PROCESSING, 50
ALEREQ01, 42
ALE_SERIAL_KEY2CHANNEL, 200
ALV, 231
ANSI ASC X12, 27
API (Application Programming Interface), 18
Application, 43
Application acknowledgment, 187
Application Link Enabling → ALE, 15
Application record, 21
Archiving object, 225
Asynchronous interface, 17
a_t_complex_matnr, 135

B

BAdI, 91
  classic, 103
  implementing class, 113
  interface, 91, 113
  new, 110
BAPI, 18, 47
  release, 19
BAPI_IDOC_INPUT1, 47
BAPI_IDOC_INPUTP, 47
BAPIRETM, 174
BAPIRETS, 174
BAPI_TRANSACTION_COMMIT, 19
BAPI_TRANSACTION_ROLLBACK, 19
BAPP, 47
Basic type, 126
BD10, 40
BD40, 195
BD41, 196
BD42, 197
BD43, 197
BD44, 194
BD51, 168
BD52, 32
BD53, 38, 40
BD54, 25
BD55, 84
BD56, 77
BD57, 198, 199
BD58, 89
BD59, 74
BD60, 34, 36, 69, 70, 135, 151
BD64, 26, 41, 67
BD65, 37
BD66, 34
BD68, 72
BD79, 81
BD87 → IDoc-Monitor, 24
BD95, 73, 198
BD97, 48
BD100;BD97a, 202, 204
BD101;BD97b, 202
BD102;BD97c, 204
BD103;BD97d, 205
BD104;BD97e, 200
BD105;BD97f, 200
BDA4, 176
BDBG, 24, 47, 173, 174

BDCP, 36
BDCP2, 36
BDCP_BEFORE_WRITE, 121
BDCPS, 36
BDFG, 174
BDSER, 198
BF01, 100
BF05, 100
Binding, 181
BOMMAT, 200
BOR, 18, 168
Breakpoint, 110
BTE, 90
BUS1001006, 168
Business Add-In → BAdI, 91
Business Application Programming Interface → BAPI, 18
Business Object Repository → BOR, 18
Business object type, 18
Business-related error, 20
Business system, 243
Business transaction event → BTE, 90
Business workflow, 18

C

CALL CUSTOMER-FUNCTION, 93
C file, 23
Change document, 30
CHANGEDOCUMENT_CLOSE, 30
CHANGEDOCUMENT_OPEN, 30
Change pointer, 31
activate generally, 32
for message type, 32
for reduced message types, 40
CHANGE_POINTERS_CREATE, 31
CHANGE_POINTERS_CREATE_LONG, 31
Channel number, 200
CIMTYP, 93
CL02, 72
Class, 71
Classic BAdI, 103
Classification, 69, 72
CL_EXITHANDLER, 103
CMOD, 94
COMMIT, 92
Communication channel, 245
Company code, cross-system, 85
CONDENSE, 153, 157
Condition technique, 43
Conversion exit, 83
CONVERSION_EXIT_ALPHA_ OUTPUT, 83
Conversion routine, 83
Cross-system business area, 85
Cross-system company code, 85
Customer
distribution model, 26, 67
management object, 101
project, 95
segment, 122
Customer exit, 90, 93

D

Data channel registry, 204
Data Dictionary → DDIC, 124
Data record, 21
Data Source, 228
Date field, 153
DDIC, 124
Debugging, 55
Decimal number, 153
Description
semantic, 16
technical, 16
Dialog call, 48
Distribution class type, 70
Document number, 198
DTD (Document Type Description), 23

E

EDI, 26
EDIFACT, 22, 27, 144
EDI_PROCESSING, 44, 50
EDIS, 65, 175
Electronic Data Interchange → EDI, 26

Enhancement, 91, 95, 107
check, 129
dynamic, 118
explicit, 107
implement, 108
implementation, 111
implicit, 115
partner profile, 131
point, 107
project, 90
release, 128
section, 107, 108
spot, 91, 107, 108, 110
static, 118
technology, 67
type, 125
EOIO (Equally Once In Order), 205
Error
business-related, 20
status, 167
technical, 20
Error workflow, 175, 176
connection, 167
Exit, 90
EXIT_SAPLBD11_001, 119
Extensible Markup Language → XML, 23
Extension, 126

F

Fetch IDoc, 41
FIBF, 101
FIDCMT, 89
File, 23
FILE, 140
File port, 56, 138
Filter group, 68
Filter object, 67, 68, 73
Floating point number, 153
Function module
evaluation of change pointers, 34
file name generation, 138
for queue names, 206
test, 55

G

GDPdU, 224
General method call, 48
General rule, 83
General task, 181, 184
Generate control record, 158
German Principles on Data Access and the Examination of Digital Documents, 224
GET*BADI, 110
GMP (Good Manufacturing Practice), 224

H

Header mapping, 245
Header record, 21
Header table, 141
HTML (Hypertext Markup Language), 23
HTTP (Hypertext Transfer Protocol), 23

I

IDoc, 18
adapter, 243
administrator, 184
monitor, 24, 160, 168
request, 41
status file, 62
suppress, 68, 76
tracking, 24
type, 22
IDOCAPPL, 176
IDOCBOMMAT, 200
IDOC_DATE_TIME_GET, 24
IDOC_INBOUND_ASYNCHRONOUS, 23
IDOC_INBOUND_IN_QUEUE, 205
IDOC_INPUT_MATMAS01, 133
IDOC_REDUCTION_FIELD_REDUCE, 156
IDOCREL, 211, 212
IDOC_SERIALIZATION_CHECK, 199

IDOC_SERIAL_POST, 199
IDX_DELETE_IDOCTYP_WITHOUT_IS, 248
IDXIDOCINB, 248
IDX_SELECT_IDOCTYP_WITHOUT_IS, 248
Importing parameter unique, 165
Inbound interface, 15
Info structure, 227
Initial, 153
INPUTERROROCCURRED, 179
inputFinished, 178, 181
INPUTFOREGROUND, 179
Interface
asynchronous, 17
synchronous, 16
Intermediate Document → IDoc, 18
ISO code, 144, 153
Item table, 141

K

KEY, 33

L

Link, 211
Logical message, 151
Logical system (LS), 25

M

Mandatory field, 37
Mandatory segment, 127, 148
Master data, 29
MASTERIDOC_CREATE_MATMAS, 93, 107
MASTERIDOC_CREATE_SMD_MATMAS, 135
MASTER_IDOC_DISTRIBUTE, 159
MATFET, 26, 41
MATMAS, 22, 26
mbdconwf, 162
Message
code, 44
control, 29, 43
determination, 44
function, 44
Message type, 22, 151
reduced, 37
Meta information, 22
Method call, 48
MGV00001, 95
MGV00100, 100
Modification, 90
MOVE, 81

N

NACE, 43
Naming rule, 148
customer object, 47
own object, 39
New BAdI, 110
NO_DATA character, 30

O

O1CL, 70
Object channel serialization, 200, 201
Object Key, 214
Object services, 214
OBJTYPE, 212
OPEN_FI_PERFORM_XXXXXXXX_E';OPEN_FI_PERFORM_XXXXXXXX_E, 99
ORDCHG, 22, 198
ORDERS, 22
Organizational management, 184
Outbound interface, 15
Outbound partner profile, 44
Output
type, 43
OYEA, 184

P

Partner, 26
- *inbound profile, 24*
- *outbound profile, 24*
- *profile, 45, 79*
- *role, 43, 145*

Party, 244
PFTC_INS, 178
PI → SAP NetWeaver PI, 187
Port management, 138
Process code, 168
- *BAPI, 174*
- *BAPP, 174*

Processing layer, 63
Process Integration → SAP NetWeaver PI, 187
Process interface, 91
Proposal for rule, 81
Proxy interface, 16
P/S interface, 91
Publish&Subscribe → P/S interface, 91

Q

qRFC, 205
Qualifier, 144, 145
Quality of Service, 205
Queue, 209
Queued RFC → qRFC, 205

R

RBDMIDOC, 35
RBDSER01, 194, 195
RBDSER02, 196
RBDSER03, 197
RBDSER04, 197
RBDSRCLR, 199
RDBSTATE, 188
Receiver, 26
Receiver communication channel, 245, 247
Reduced message type, 37
Release security, 15, 19
Remote Function Call → RFC, 16
Request, 16
Request IDocs, 41
Response, 16
RFC, 16
- *queued → qRFC, 205*
- *synchronous → sRFC, 19*
- *transactional → tRFC, 19*

RFM (RFC-enabled function module), 18
ROLEID, 212
ROLLBACK, 92
Root segment, 78, 79
RSEOUT00, 59, 196
RSNAST00, 44, 58
RSRLDREL, 215
Rule, 67, 80
- *general, 83*

S

SALE, 25, 31
Sample file, 56
SAP_ALE_EDIQUEUE, 206
SAP_IDOC_001, 227
SAP NetWeaver PI, 205, 243
- *agency, 244*
- *header mapping, 245*
- *partner, 26*
- *party, 244*

SARA, 225
SARI, 228
SBWP, 183
SE11, 124
SE18, 103
SE19, 103, 110
SE80, 111
SE84, 91
Segment
- *definition, 124*
- *filter, 76*
- *qualifying, 145*
- *release, 125*
- *type, 23, 124*
- *version, 84*

Sender, 26

Sender field
convert, 83
copy, 82
SERDAT, 197
SERIAL, 198, 203
Serialization, 193
business objects, 200
flag, 202
group, 193, 194
timestamp, 198
Shared Master Data Tool → SMD, 29
Signature, 154
inbound function module, 162
SLD, 244
SM59, 102
SMD, 29
SNDPOR, 243
SNDPRN, 244
SOAP, 16
Special method call, 48
sRFC, 19
SRRELROLES, 211
Standard inbound processing, 54
Standard interface, 15
Standard outbound processing, 57
Standard task, 178
Status 42<normal>, 57
Status 53<normal>, 188
Status 66, 204
Status 68<normal>, 188
Status 74<normal>, 61
STATUS IDoc, 62, 190
Status record, 21
Success status, 167
SWO1, 176
Synchronous interface, 16
Synchronous RFC → sRFC, 19
SYSTAT, 62, 230
System Landscape Directory → SLD, 244
System name, logical, 25
System, technical, 244

T

Task, general, 181, 184
Technical error, 20
Technical system, 244
Templates, 45
Test
flag, 54
function module, 55
transaction, 51
Time, 153
Transactional RFC → tRFC, 19
Transaction data, 29
Transaction security, 17
TRANSID, 212
Transmission medium 6, 43, 44
Transmission medium A, 43, 50
Transport IDoc, 23
tRFC, 19

U

UN/EDIFACT → EDIFACT, 22
Unicode, 56
UNIT_OF_MEASURE_ISO_TO_SAP, 165
UNIT_OF_MEASURE_SAP_TO_ISO, 157
Update technique, 19

V

Version conversion, 84, 119
View, 77

W

WE09, 228
WE14, 58
WE16, 60
WE17, 65
WE18, 63
WE19, 52, 65
WE20, 24, 50, 119, 194
WE21, 138
WE24, 45
WE27, 45
WE30, 126, 148
WE31, 124
WE32, 78

WE41, 44
WE42, 169
WE47, 63, 224
WE55, 139
WE57, 133, 168
WE60, 150
WE62, 81
WE81, 151
WE82, 136, 151
WE85, 205
WEDI, 51
WEINBQUEUE, 208
WENOLINKS, 214
WEOUTQUEUE, 206
WMTORD, 46

X

XML, 23
xPath, 246