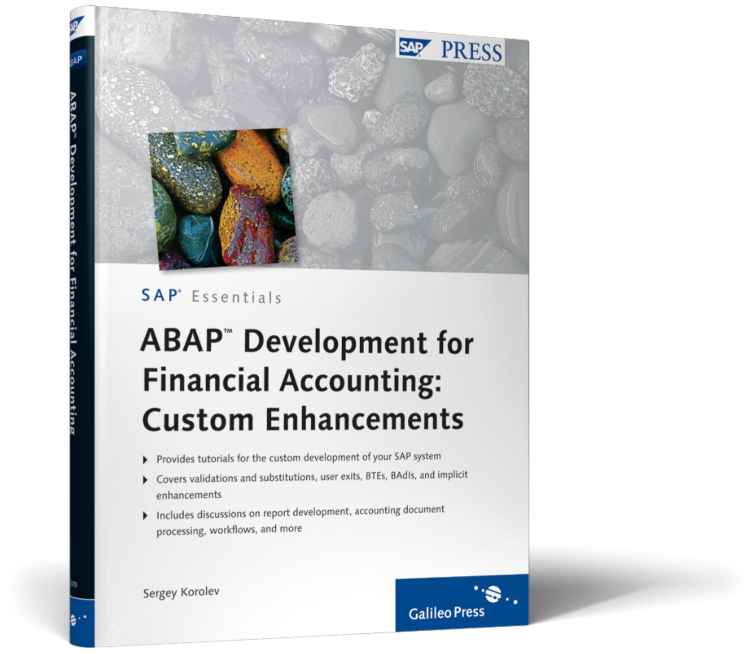
Sergey Korolev

ABAP™ Development for Financial Accounting

Custom Enhancements


SAP PRESS
SAP® Essentials
ABAP™ Development for Financial Accounting: Custom Enhancements
Provides tutorials for the custom development of your SAP system
Covers validations and substitutions, user exits, BTEs, BAdIs, and implicit enhancements
Includes discussions on report development, accounting document processing, workflows, and more
Sergey Korolev
Galileo Press

Galileo Press

Bonn • Boston

Contents at a Glance

Contents

As today's corporate ERP system landscape becomes more and more distributed, you have to be prepared for different kinds of data that can flow to and from external systems. With this in mind, the focus of this chapter is inbound scenarios in Financial Accounting.

5 Inbound Scenarios in Financial Accounting

In this chapter, we consider data processing scenarios when the SAP system receives accounting data from external systems. This can be master data from legacy systems or posting data from, for example, an external payroll system. This chapter describes how you can intervene in this process using various user exits.

5.1 Master Data Migration and Distribution

There could be no SAP ERP implementation project without an initial data migration procedure. Imagine how painful it would be if a company started its trading activity by implementing SAP ERP and then entered its existing customers and vendors one by one. As a rule, the moment a company implements SAP ERP, the customer/vendor list (which is in some other legacy system) has to be prepared. There are also scenarios in which accounting master data are loaded from external systems on a regular basis.

In the following subsections, we'll discuss several ways to load master data into an SAP system and how to seamlessly penetrate the standard data flow to address specific requirements.

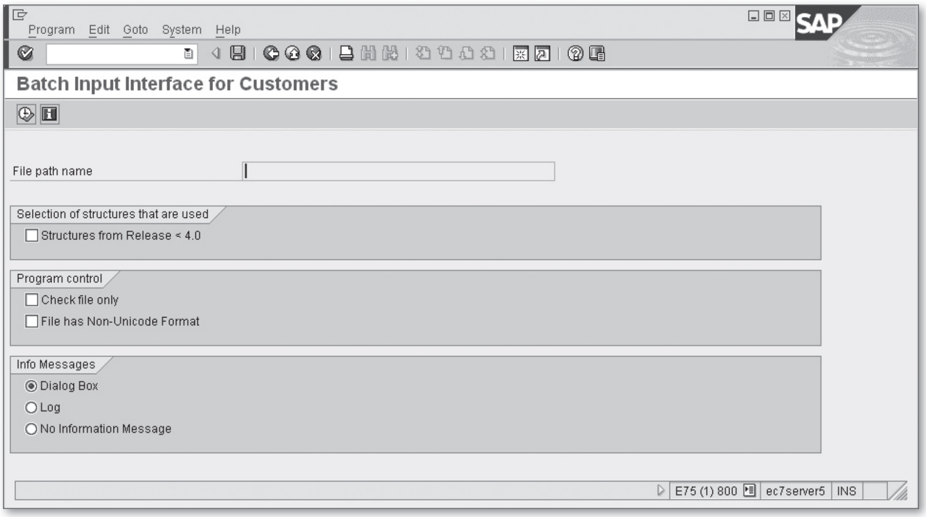
5.1.1 Batch Input

If you are familiar with the SAP Legacy System Migration Workbench (LSMW) and have completed data migration projects, you probably recognize these standard SAP programs for the mass uploading of customer and vendor master records: reports RFBIDE00 and RFBIKR00.

Both reports have the same selection screen as shown in Figure 5.1. Input data for the report must be presented as a flat file located on the application server.

Note

You can also pass a logical file name into the report by passing it through invisible parameter `LDS_NAME`, which can be used in a `SUBMIT` statement. In this case, the value of the visible file path name parameter is ignored.



Batch Input Interface for Customers

File path name

Selection of structures that are used
☐ Structures from Release < 4.0

Program control
☐ Check file only
☐ File has Non-Unicode Format

Info Messages
● Dialog Box
○ Log
○ No Information Message

E75 (1) 800 | ec7server5 | INS

Figure 5.1 Selection Screen of Report RFBIDE00

By default, the maximum length of an input file line is 2,000 characters—this is the length of dictionary structure `BDIFIBIWA`. If your input file has longer lines, you can extend structure `BDIFIBIWA` by using customer include structure `CI_BDIFIBIWA`.

Keep in mind, however, that structure `BDIFIBIWA` only defines the length of an input file line, whereas the actual structure of the data being processed is defined according to the first 31 characters of the line (see the structure shown in Figure 5.2).

The first character of each file line is a *record type*, which can take one of three values: 0, 1, or 2. Record type 0 marks the beginning of a session, record type 1 is the beginning of one customer (or vendor) data for one transaction code, and record type 2 is a data record. The next 30 characters of a file line contain a dictionary structure name. For record type 0, the structure name is always `BGR00`; for record type 1, the structure name is always `BKN00` for customers and `BLF00` for vendors.

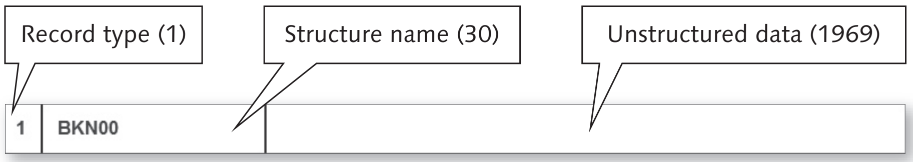


Figure 5.2 The Structure of a Flat File Line

In the record with structure BGR00, you can denote the transaction code that will be used to process the data. The record with structure BKN00 contains the customer number and corresponding organizational assignment, such as company code, sales market data, credit control area, and so on. In the record with structure BLF00, the data contains information for the vendor number, company code, purchasing organization, and so on.

File lines with record type 2 can contain standard and nonstandard structures. Standard batch-input structures mainly comply with the following naming convention: character B followed by one of the master data table names. For example, BKNA1 is a batch-input structure for Table KNA1, BLFA1 is the batch-input counterpart for LFA1, and so on.

Note

The full list of all standard batch-input structures and supported transactions can be found in SAP online help for reports RFBIDE00 and RFBIKR00.

In the next subsection, you'll see learn to extend data and amend its processing using BAdIs. You'll also see a step-by-step example of loading extended data with a standard SAP program.

Data Enhancement

You can enhance batch-input data either by defining your own fields in the corresponding customer include, which you can find in all standard batch-input structures (e.g., CI_BKNA1 in BKNA1), or by defining your own data structures.

If you choose the second option, follow the same conventions found in the standard structure:

- The first two fields of the customer include should be the same as in the standard structure (STYPE and TABNAME).
- All fields must be characters (no numbers).

Note

To make the customer-defined batch-input structure available in SAP LSMW, you must insert a corresponding entry in the customizing table SXDA2.

Using BAdIs

If your custom-defined fields are part of an additional screen layout (see Chapter 2, Master Data Enhancements), then you have to apply user exits to make the system process additional data in customer or vendor loading reports.

Customer loading report RFBIDE00 uses the following BAdI definitions and methods:

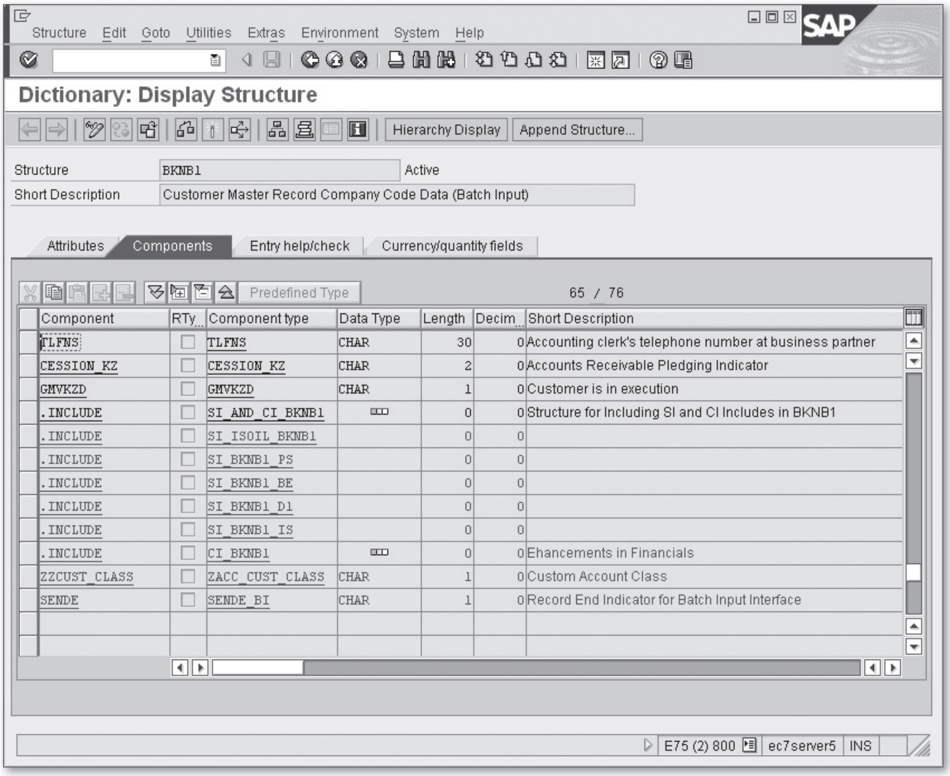
- **Definition:** `CUSTOMER_ADD_DATA`
  - Method `CHECK_ADD_ON_ACTIVE` is called in the initialization phase of the report. Other BAdI methods are called only if at least one add-on is active.
- **Definition:** `CUSTOMER_ADD_DATA_BI`
  - Method `CHECK_DATA_ROW` is called for any nonstandard file line with record type 2 and an unknown structure name. The method can be used to check the input contents for nonstandard structures.
  - Method `FILL_FT_TABLE_USING_DATA_ROWS` is called at the end of transactions processing (only for Transactions XD01 and XD02). The method can be used to amend or extend generated batch-input screens and field sequences to incorporate add-on screens and fields.

Vendor loading report RFBIKR00 uses the following BAdI definitions: `VENDOR_ADD_DATA` and `VENDOR_ADD_DATA_BI`. Method names and their purposes are the same as in report RFBIDE00; and logical method `FILL_FT_TABLE_USING_DATA_ROWS` is only called for Transactions XK01 and XK02.

Example

To illustrate the enhancement usage in our IDES system, let's incorporate the example from Chapter 2, where we enhanced customer master data, into the standard loading program, RFBIDE00. We extended the company code view of the customer master data by an additional field: Custom Account Class (with technical name KNB1-ZZCUST_CLASS).

First, we extended the dictionary structure (BKNB1) by defining the customer include (CI_BKNB1). As a result, the BKNB1 definition in Transaction SE11 should look like Figure 5.3.



Dictionary: Display Structure

Structure: BKNB1 Active
Short Description: Customer Master Record Company Code Data (Batch Input)

65 / 76

Component	RTy	Component type	Data Type	Length	Decim	Short Description
TLFNS	☐	TLFNS	CHAR	30	0	Accounting clerk's telephone number at business partner
CESSION_KZ	☐	CESSION_KZ	CHAR	2	0	Accounts Receivable Pledging Indicator
GMVKZD	☐	GMVKZD	CHAR	1	0	Customer is in execution
.INCLUDE	☐	SI_AND_CI_BKNB1	▭	0	0	Structure for Including SI and CI Includes in BKNB1
.INCLUDE	☐	SI_ISOIL_BKNB1		0	0	
.INCLUDE	☐	SI_BKNB1_PS		0	0	
.INCLUDE	☐	SI_BKNB1_BE		0	0	
.INCLUDE	☐	SI_BKNB1_D1		0	0	
.INCLUDE	☐	SI_BKNB1_IS		0	0	
.INCLUDE	☐	CI_BKNB1	▭	0	0	Ehancements in Financials
ZZCUST_CLASS	☐	ZACC_CUST_CLASS	CHAR	1	0	Custom Account Class
SENDE	☐	SENDE_BI	CHAR	1	0	Record End Indicator for Batch Input Interface

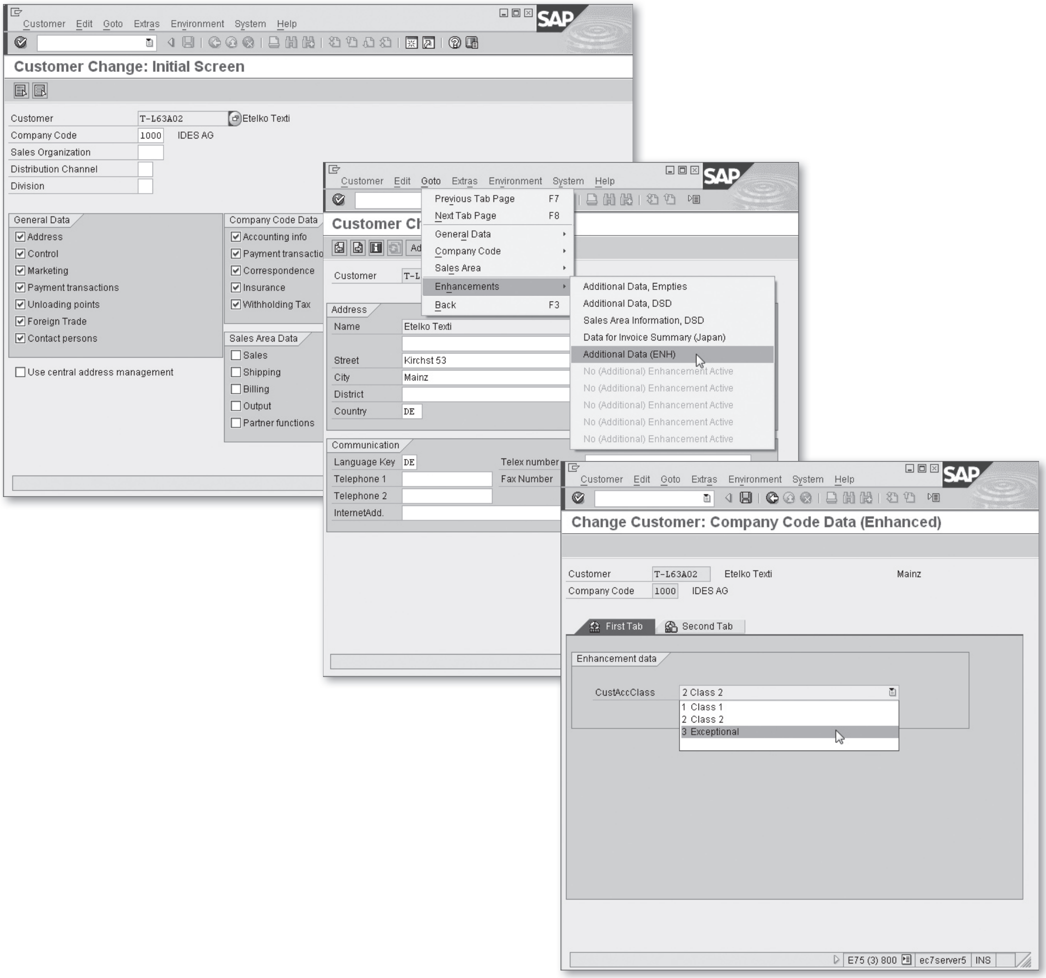
Figure 5.3 Extended BKNB1 Dictionary Structure

When preparing the example for Chapter 2, we implemented BAdI CUSTOMER_ADD_DATA. Now we need to use BAdI definition CUSTOMER_ADD_DATA_BI. Because we haven't created our own batch-input structure, but extended a standard structure instead, we don't need to implement the CHECK_DATA_ROW method. We do need to code an addition to the screen and field sequence, which will save our data into the customer master record. To do this, we need to examine how the screen sequence might look by using an old batch-input recording, which can be found in Transaction SHDB.

We record the following actions of Transaction XD02 with the following steps:

1. Enter the customer number and company code.
2. Select the enhanced screen layout (defined in Chapter 2).
3. Change the value in the CustAccClass field (no matter from which to which; we just need a value change).
4. Save.

Figure 5.4 shows the combined sequence of screenshots of these steps.


Customer Edit Goto Extras Environment System Help
Previous Tab Page F7
Next Tab Page F8
General Data
Company Code
Sales Area
Enhancements
Back F3
Additional Data, Empties
Additional Data, DSD
Sales Area Information, DSD
Data for Invoice Summary (Japan)
Additional Data (ENH)
No (Additional) Enhancement Active
No (Additional) Enhancement Active
No (Additional) Enhancement Active
No (Additional) Enhancement Active
No (Additional) Enhancement Active
First Tab
Second Tab

Customer Change: Initial Screen

Customer	T-L63A02 Etelko Texti
Company Code	1000 IDES AG
Sales Organization	
Distribution Channel	
Division	

General Data: ☑ Address ☑ Control ☑ Marketing ☑ Payment transactions ☑ Unloading points ☑ Foreign Trade ☑ Contact persons

Company Code Data: ☑ Accounting info ☑ Payment transactio ☑ Correspondence ☑ Insurance ☑ Withholding Tax

Sales Area Data: ☐ Sales ☐ Shipping ☐ Billing ☐ Output ☐ Partner functions

☐ Use central address management

Customer Ch

Customer T-L

Address

Name	Etelko Texti
Street	Kirchst 53
City	Mainz
District	
Country	DE

Communication

Language Key	DE	Telex number	
Telephone 1		Fax Number	
Telephone 2			
InternetAdd.			

Change Customer: Company Code Data (Enhanced)

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

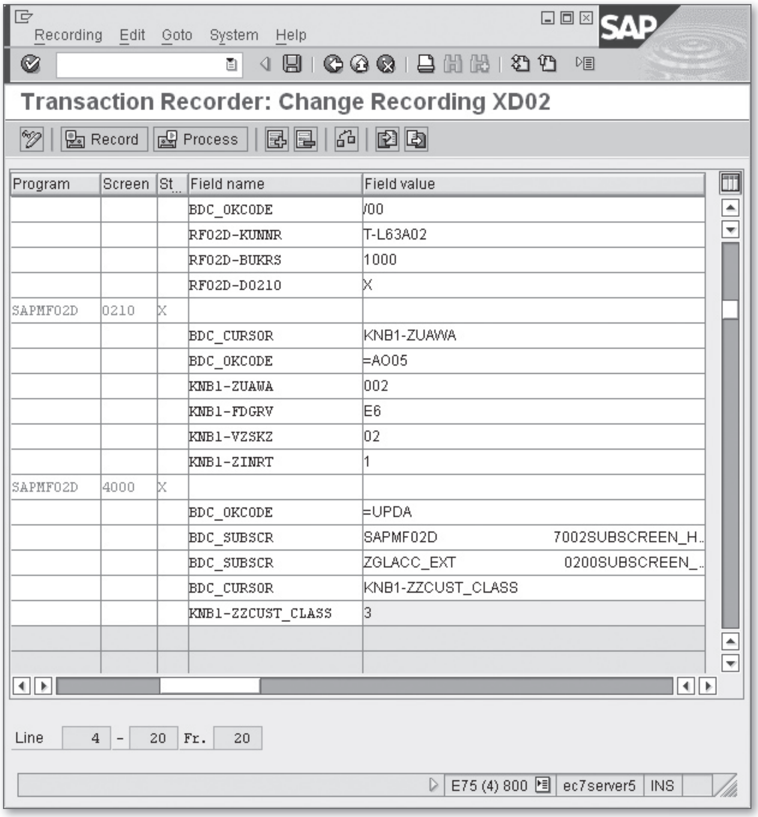
Enhancement data

CustAccClass 2 Class 2
1 Class 1
2 Class 2
3 Exceptional

E75 (3) 800 ec7server5 INS

Figure 5.4 Recorded Screen Sequence of Transaction XD02

The result of the recording is shown in Figure 5.5.



Transaction Recorder: Change Recording XD02

Program	Screen	St...	Field name	Field value
			BDC_OKCODE	/00
			RF02D-KUNNR	T-L63A02
			RF02D-BUKRS	1000
			RF02D-D0210	X
SAPMF02D	0210	X		
			BDC_CURSOR	KNB1-ZUAWA
			BDC_OKCODE	=AO05
			KNB1-ZUAWA	002
			KNB1-FDGRV	E6
			KNB1-VZSKZ	02
			KNB1-ZINRT	1
SAPMF02D	4000	X		
			BDC_OKCODE	=UPDA
			BDC_SUBSCR	SAPMF02D 7002SUBSCREEN_H..
			BDC_SUBSCR	ZGLACC_EXT 0200SUBSCREEN_..
			BDC_CURSOR	KNB1-ZZCUST_CLASS
			KNB1-ZZCUST_CLASS	3

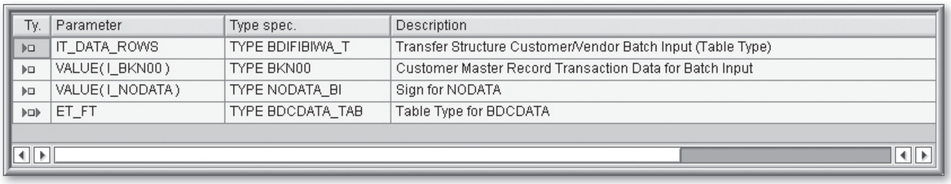
Line 4 - 20 Fr. 20

E75 (4) 800 ec7server5 INS

Figure 5.5 The Recording of Transaction XD02

As you analyze the recording, you see that on the starting data screen SAPMFD02/200, we executed function code `AO05`, which has taken us into the enhanced screen layout. There we entered a value of 3 into the field KNB1-ZZCUST_CLASS and clicked Save (function code `UPDA`).

Now we are ready to implement the code of method `FILL_FT_TABLE_USING_DATA_ROWS`.



Ty.	Parameter	Type spec.	Description
▸▫	IT_DATA_ROWS	TYPE BDIFIBIWA_T	Transfer Structure Customer/Vendor Batch Input (Table Type)
▸▫	VALUE(I_BKN00)	TYPE BKN00	Customer Master Record Transaction Data for Batch Input
▸▫	VALUE(I_NODATA)	TYPE NODATA_BI	Sign for NODATA
▸▫▸	ET_FT	TYPE BDCDATA_TAB	Table Type for BDCDATA

Figure 5.6 The Interface of Method FILL_FT_TABLE_USING_DATA_ROWS

Figure 5.6 shows the interface of method `FILL_FT_TABLE_USING_DATA_ROWS`. You can see that we have current `BKN00` data (with customer number and other organizational assignment data) as input parameter `I_BKN00`; we also have all file lines related to the current transaction in input parameter `IT_DATA_ROWS`. Finally, we have one export table typed parameter, `ET_FT`, which we will amend according to our logic. `ET_FT` has line type of `BDCDATA` structure, which is a well-known structure used in batch-input statement `CALL TRANSACTION USING`.

The algorithm should do the following:

- Find the first entry of structure `BKNB1` in the file data.
- Insert function code `AO05` into the previous screen: BDC data.
- Start a new screen in BDC data.
- Set new field values according to `BKNB1` contents that were found.

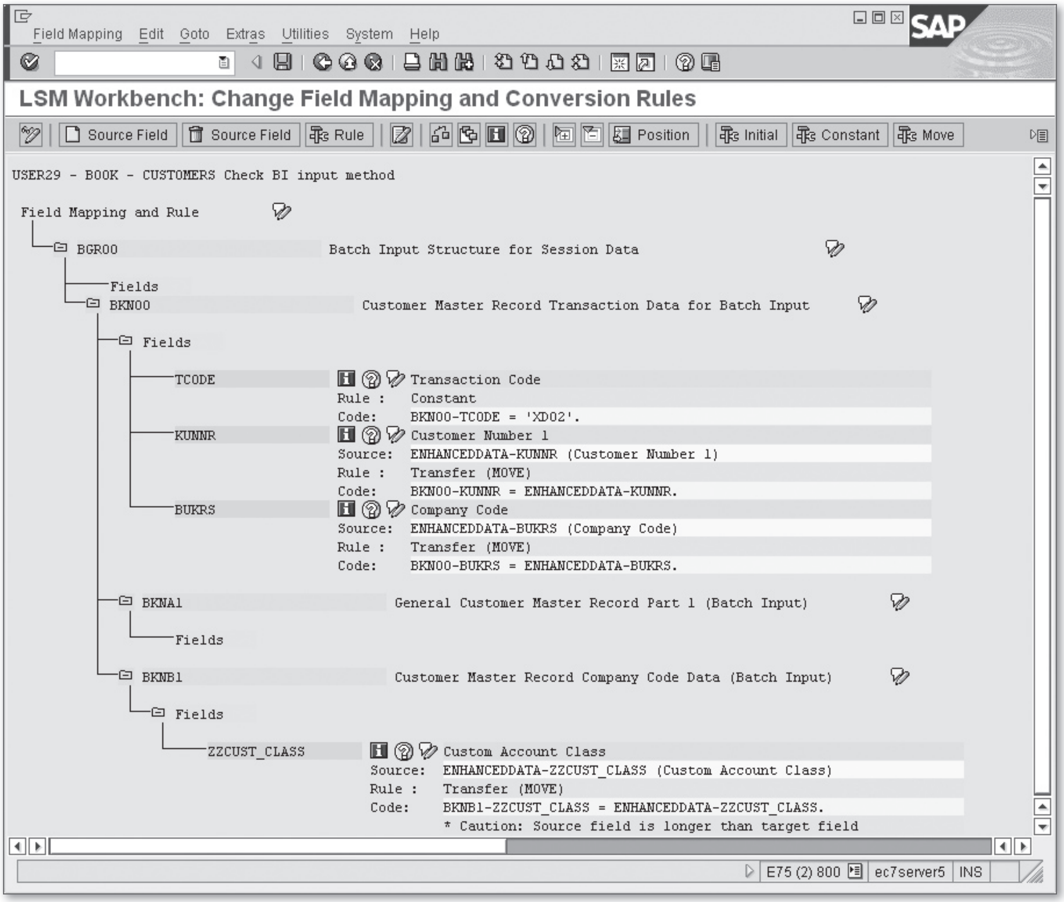
Always keep in mind that there can be other active BAdI implementations, so you shouldn't include any function codes in the batch input because this can end the transaction. In our example, we don't insert the function code `UPDA`, which is seen in our sample recording (refer back to Figure 5.5).

Listing 5.1 shows the source code of our method implementation.

```
METHOD if_ex_customer_add_data_bi~fill_ft_table_using_data_rows.
  FIELD-SYMBOLS: <wa> TYPE bknb1.

  DATA: ft TYPE bdcdata.
  LOOP AT it_data_rows ASSIGNING <wa> CASTING.
    CHECK <wa>-stype = '2' AND <wa>-tbnam = 'BKNB1'.

*   Insert function code to select Enhanced screen layout
*   This will be added to the last processed screen in BDC data
    CLEAR ft.
    ft-fnam = 'BDC_OKCODE'.
    ft-fval = '=AO05'.
    APPEND ft TO et_ft.

*   Start new screen
    CLEAR ft.
    ft-program = 'SAPMF02D'.
    ft-dynpro = '4000'.
    ft-dynbegin = 'X'.
    APPEND ft TO et_ft.

*   Enter field value on the custom defined screen
    CLEAR ft.
    ft-fnam = 'KNB1-ZZCUST_CLASS'.
    ft-fval = <wa>-zzcust_class.
    APPEND ft TO et_ft.

    EXIT.
  ENDLOOP.
ENDMETHOD.
```

Listing 5.1 Method FILL_FT_TABLE_USING_DATA_ROWS Source

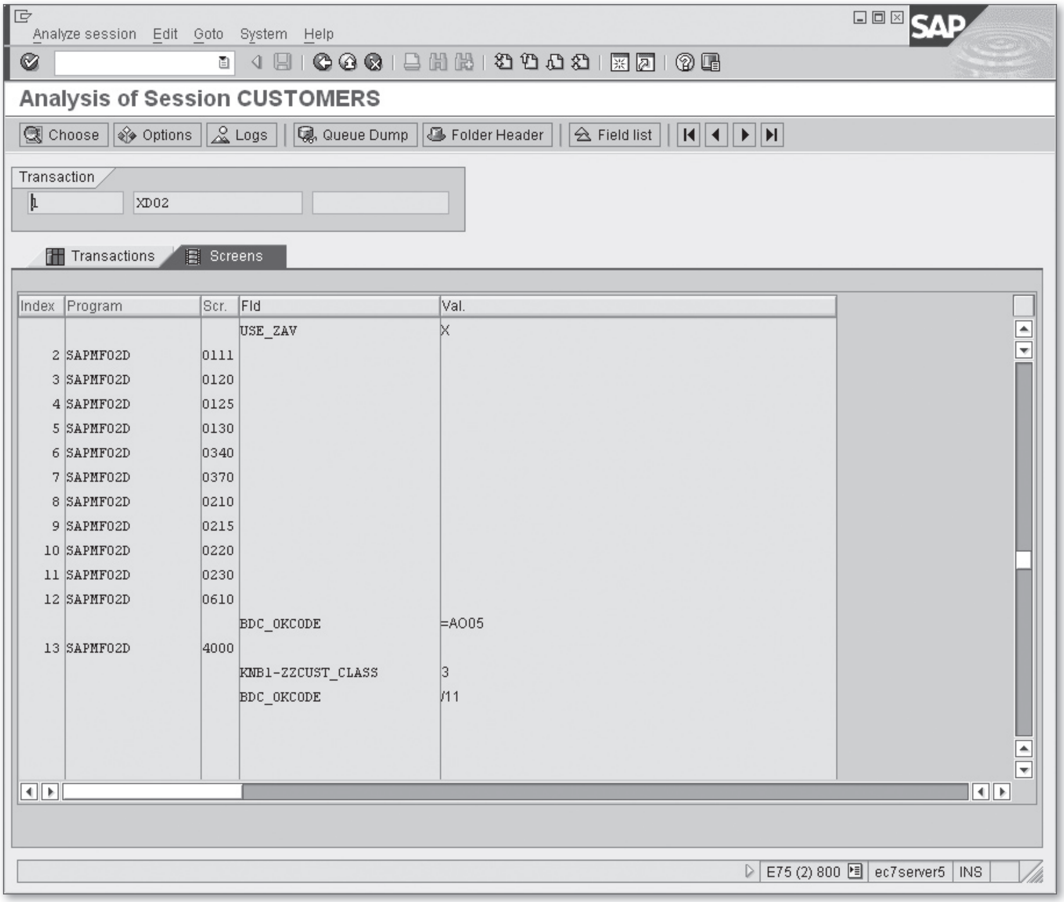
After activating the BAdI implementation, we can now test the new fields with a small SAP LSMW project. The goal of this project is to update field `KNB1-ZZCUST_CLASS` using the batch-input loading program RFBIDE00. After defining the appropriate target object and source structure, you can see in the SAP LSMW field-mapping step that our field is included in the target structure (see Figure 5.7). Note that all uninitialized fields are turned off to make the view more compact.



```
USER29 - BOOK - CUSTOMERS Check BI input method

Field Mapping and Rule
  BGR00                 Batch Input Structure for Session Data
    Fields
    BKN00               Customer Master Record Transaction Data for Batch Input
      Fields
        TCODE           Transaction Code
                        Rule :   Constant
                        Code:    BKN00-TCODE = 'XD02'.
        KUNNR           Customer Number 1
                        Source:  ENHANCEDDATA-KUNNR (Customer Number 1)
                        Rule :   Transfer (MOVE)
                        Code:    BKN00-KUNNR = ENHANCEDDATA-KUNNR.
        BUKRS           Company Code
                        Source:  ENHANCEDDATA-BUKRS (Company Code)
                        Rule :   Transfer (MOVE)
                        Code:    BKN00-BUKRS = ENHANCEDDATA-BUKRS.
      BKNA1             General Customer Master Record Part 1 (Batch Input)
        Fields
      BKNB1             Customer Master Record Company Code Data (Batch Input)
        Fields
          ZZCUST_CLASS  Custom Account Class
                        Source:  ENHANCEDDATA-ZZCUST_CLASS (Custom Account Class)
                        Rule :   Transfer (MOVE)
                        Code:    BKNB1-ZZCUST_CLASS = ENHANCEDDATA-ZZCUST_CLASS.
                                 * Caution: Source field is longer than target field
```

Figure 5.7 LSMW Field Mapping View for the Customer Master

The Create Batch Input Session step in the SAP LSMW project is actually a call of the program RFBIDE00. We tested it with only one record in the input file to update customer T-L63A02 in company code 1000. Now change the CustAccClass field to 3. After generating the batch-input session, we can inspect it in Transaction SM35. Figure 5.8 shows the screen list of the session with an opened field value list. Our added field is in its place.



Analysis of Session CUSTOMERS

Transaction: 1 | XD02

Index	Program	Scr.	Fld	Val.
			USE_ZAV	X
2	SAPMF02D	0111		
3	SAPMF02D	0120		
4	SAPMF02D	0125		
5	SAPMF02D	0130		
6	SAPMF02D	0340		
7	SAPMF02D	0370		
8	SAPMF02D	0210		
9	SAPMF02D	0215		
10	SAPMF02D	0220		
11	SAPMF02D	0230		
12	SAPMF02D	0610		
			BDC_OKCODE	=AO05
13	SAPMF02D	4000		
			KNB1-ZZCUST_CLASS	3
			BDC_OKCODE	/11

E75 (2) 800 | ec7server5 | INS

Figure 5.8 Batch-Input Session Analysis in Transaction SM35

5.1.2 HR Master Data

In some HR payroll instances, an employee has his own HR master record, which generates a corresponding vendor master record or customer master record for that employee in the financials department of the company. From the formal accounting point of view, when the company pays the salary to that employee, he should be treated as a company vendor because that employee sells his services to the company (in the form of an everyday job). If HR Payroll and FI are installed as separate systems, you must set up a task of regularly distributing HR employee data into an FI system to form vendor or customer master records.

In brief, the standard process of HR data distribution, which is based on ALE (application link enabling) technology, looks as follows:

1. Several structures of employee data (called *infotypes*) from the external HR system are copied into the FI system, in the form of an IDoc (depending on the HR system version, it can be an IDoc type from `HRMD_A01` to `HRMD_A07`).
2. The receiving FI system regularly runs report RPRAPA00, which prepares the locally available HR data for loading with the standard report RFBIKR00.
3. Inside report RPRAPA00, a BAdI definition `BADI_EXITS_RPRAPA00` is used to intercept the standard logic when preparing a data file for the following run of the report RFBIKR00. The list of available BAdI methods is shown in Table 5.1.

Method	Description
`SET_VALUES_FOR_BLFBW`	Exit for `BLFBW`: Vendor master, withholding tax types
`SET_VALUES_FOR_BLF00`	Exit for `BLF00`: Vendor master
`SET_VALUES_FOR_BLFA1`	Exit for `BLFA1`: Vendor master, general data part 1
`SET_VALUES_FOR_BLFBK`	Exit for `BLFBK`: Vendor master, bank details
`SET_VALUES_FOR_BLFB1`	Exit for `BLFB1`: Vendor master, company code data
`SET_VALUES_FOR_BLFB5`	Exit for `BLFB5`: Vendor master, dunning data
`SET_VALUES_FOR_BGR00`	Exit for `BGR00`: Batch-input structure for session data

Table 5.1 Interface Methods of the BAdI Definition BADI_EXITS_RPRAPA00

Each method has an employee number (`PERNR`) as an input parameter and a respective batch-input structure as a changing parameter. The structure name is clearly shown by the method name.

Because report RPRAPA00 works on the local HR data, you can use standard HR functionality to access employee infotypes. All the BAdI methods are called in the end of each employee number processing.

5.1.3 ALE/IDoc

The batch-input data loading techniques discussed earlier are based on a file as a data carrier. This is a somewhat outdated technology, and while it is robust and stable, it's less flexible and less secure compared to ALE/IDoc technology. IDoc processing logic is completely separated from the data transferring media, which is much more suitable to the modern distributed environments with its variety of data transferring protocols. In essence, ALE/IDoc technology is more welcome in modern integration projects involving B2B (business to business), A2A (application to application), and mobile scenarios.

When it comes to making a decision on what type of technology to employ in an integrating project of almost any nature, we recommend choosing IDocs over files. ALE/IDoc technology is highly configurable, and depending on corporate-specific requirements, you can completely intercept the IDoc processing of any individual type.

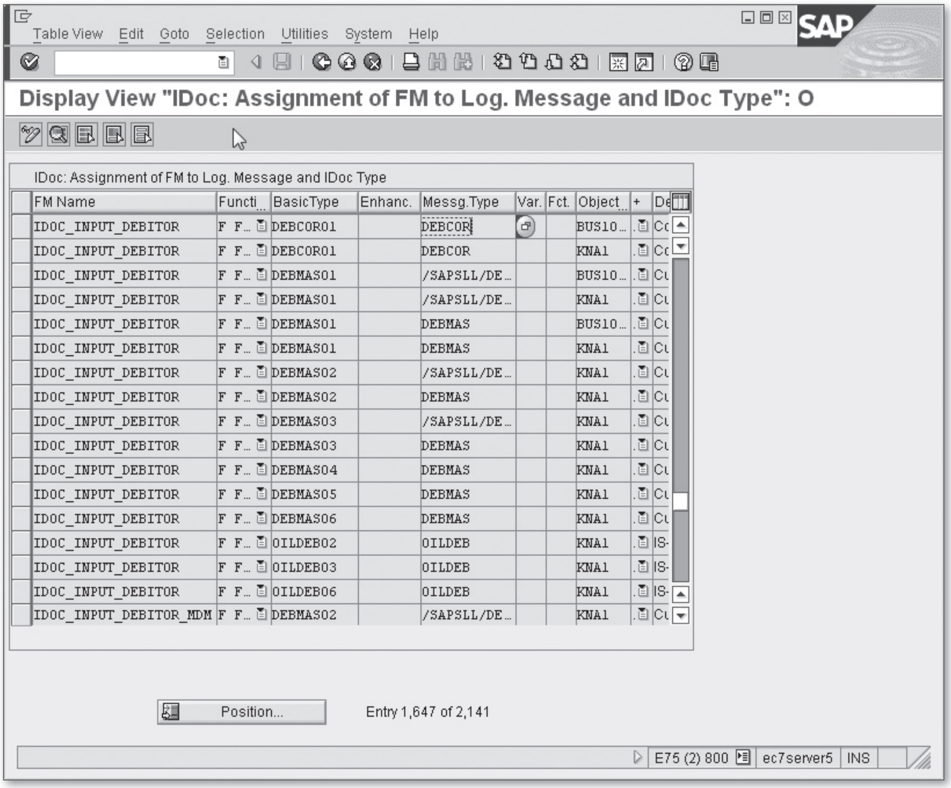
The Structure of an IDoc in a Nutshell

The structure of an IDoc is identified by its basic type, which is an ordered set of segments. For simplicity, the notion of an IDoc segment can be treated as an equivalent of the dictionary structure. Basic type defines not only a simple order of its segments but also their hierarchy relations, cardinality, and necessity. In other words, the basic type defines the syntax of IDoc, which is controlled by the runtime ALE system layer. The IDoc basic type structure can be displayed using Transaction WE30.

For the sake of simplicity, we can also say that a pair of objects—logical message code and basic type—together define IDoc processing logic via assignment to a specific ABAP function module, workflow template, or task. These assignments are stored in configuration table EDIFCT, which is accessible via Transaction WE57.

SAP delivers the following logical messages for master data distribution via ALE: `CREMAS` and `CRECOR` for vendors, and `DEBMAS` and `DEBCOR` for customers. Figure 5.9 shows the IDoc processing module configuration for customer-related messages and IDoc types.

If you look into the default IDoc configuration table EDIFCT (via Transaction WE57), you can see that standard processing logic for inbound IDoc transferring customer and vendor master data is hidden in two function modules: `IDOC_INPUT_DEBITOR` and `IDOC_INPUT_CREDITOR`. These function modules are assumed to process IDoc basic types from `CREMAS01` to `CREMAS05`, and from `DEBMAS01` to `DEBMAS06`, `CRECOR01`, and `DEBCOR01`. In this notation, the numeric suffix is the version of the IDoc structure.


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

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

FM Name	Functi…	BasicType	Enhanc.	Messg.Type	Var.	Fct.	Object…	+	De
IDOC_INPUT_DEBITOR	F F…	DEBCOR01		DEBCOR			BUS10…	.	Cc
IDOC_INPUT_DEBITOR	F F…	DEBCOR01		DEBCOR			KNA1	.	Cc
IDOC_INPUT_DEBITOR	F F…	DEBMAS01		/SAPSLL/DE…			BUS10…	.	Cu
IDOC_INPUT_DEBITOR	F F…	DEBMAS01		/SAPSLL/DE…			KNA1	.	Cu
IDOC_INPUT_DEBITOR	F F…	DEBMAS01		DEBMAS			BUS10…	.	Cu
IDOC_INPUT_DEBITOR	F F…	DEBMAS01		DEBMAS			KNA1	.	Cu
IDOC_INPUT_DEBITOR	F F…	DEBMAS02		/SAPSLL/DE…			KNA1	.	Cu
IDOC_INPUT_DEBITOR	F F…	DEBMAS02		DEBMAS			KNA1	.	Cu
IDOC_INPUT_DEBITOR	F F…	DEBMAS03		/SAPSLL/DE…			KNA1	.	Cu
IDOC_INPUT_DEBITOR	F F…	DEBMAS03		DEBMAS			KNA1	.	Cu
IDOC_INPUT_DEBITOR	F F…	DEBMAS04		DEBMAS			KNA1	.	Cu
IDOC_INPUT_DEBITOR	F F…	DEBMAS05		DEBMAS			KNA1	.	Cu
IDOC_INPUT_DEBITOR	F F…	DEBMAS06		DEBMAS			KNA1	.	Cu
IDOC_INPUT_DEBITOR	F F…	OILDEB02		OILDEB			KNA1	.	IS-
IDOC_INPUT_DEBITOR	F F…	OILDEB03		OILDEB			KNA1	.	IS-
IDOC_INPUT_DEBITOR	F F…	OILDEB06		OILDEB			KNA1	.	IS-
IDOC_INPUT_DEBITOR_MDM	F F…	DEBMAS02		/SAPSLL/DE…			KNA1	.	Cu

Entry 1,647 of 2,141

Figure 5.9 The Contents of Table EDIFCT

Both function modules work the same way. They first analyze the system type; if it's an ERP system, the function modules call an ERP-specific function: `ERP_IDOC_INPUT_CREDITOR` for a vendor and `ERP_IDOC_INPUT_DEBITOR` for a customer. There is also a function call for a standalone HR system, but it's quite simple. Because HR doesn't need any advanced customer or vendor master data manipulations, you'll find just a direct update of the corresponding tables.

The main secret of standard IDoc processing logic is that it updates or creates individual master record by means of batch input. If you dive into the source code of `ERP_IDOC_INPUT_DEBITOR` or `ERP_IDOC_INPUT_CREDITOR`, you'll find the corresponding `CALL TRANSACTION` statement. In a way, they repeat the logic of reports RFBIDE00 and RFBIKR00; but instead of a flat file, these functions process IDocs, and each segment can be treated as an equivalent of a file line. You can also see that after processing IDoc segments, the function gathers information into an internal table of structure `BDIFIBIWA`.

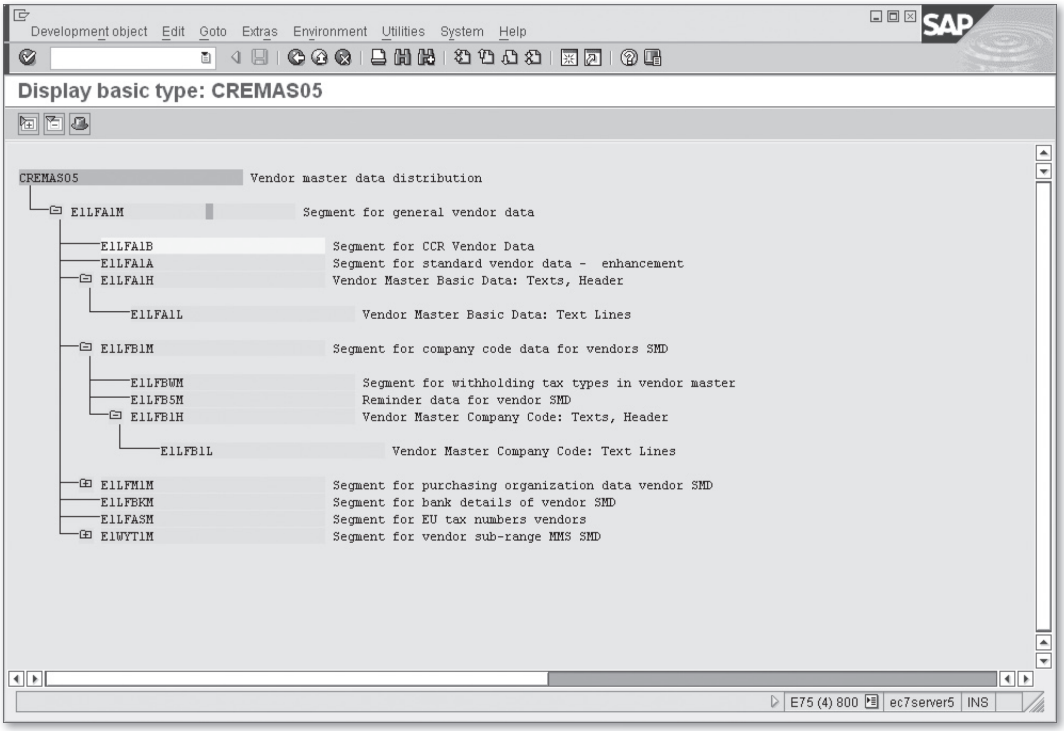
Note

In IDoc processing, SAP provides calling moments for the same BAdI definition as in `RFBIDE00` and `RFBIKR00`.

Next, we'll discuss working with IDoc data structures—segments—and how you can affect the processing logic in standard SAP functions.

Working with Segments

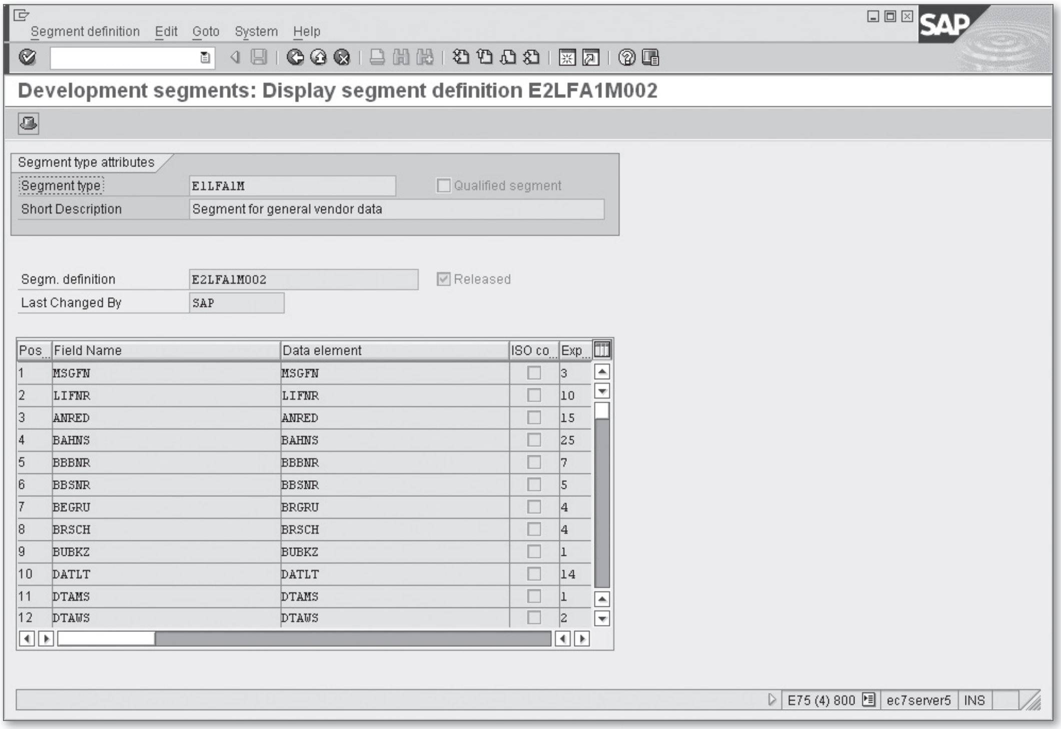
The structure of the IDoc type you are planning to process can be displayed in Transaction WE30. Figure 5.10 shows the structure of IDoc basic type `CREMAS05`. As you can see, there are three levels of segment hierarchy.


Development object Edit Goto Extras Environment Utilities System Help
Display basic type: CREMAS05
E75 (4) 800 ec7server5 INS

```
CREMAS05                    Vendor master data distribution
└─ E1LFA1M                  Segment for general vendor data
   ├─ E1LFA1B               Segment for CCR Vendor Data
   ├─ E1LFA1A               Segment for standard vendor data -  enhancement
   ├─ E1LFA1H               Vendor Master Basic Data: Texts, Header
   │  └─ E1LFA1L            Vendor Master Basic Data: Text Lines
   ├─ E1LFB1M               Segment for company code data for vendors SMD
   │  ├─ E1LFBWM            Segment for withholding tax types in vendor master
   │  ├─ E1LFB5M            Reminder data for vendor SMD
   │  └─ E1LFB1H            Vendor Master Company Code: Texts, Header
   │     └─ E1LFB1L         Vendor Master Company Code: Text Lines
   ├─ E1LFM1M               Segment for purchasing organization data vendor SMD
   ├─ E1LFBKM               Segment for bank details of vendor SMD
   ├─ E1LFASM               Segment for EU tax numbers vendors
   └─ E1WYT1M               Segment for vendor sub-range MMS SMD
```

Figure 5.10 The Structure of IDoc Type CREMAS05

By double-clicking on an arbitrary segment name, you can drill down to the segment editor where you can see the list of segment fields. You can see an example of segment structure in the segment editor in Figure 5.11.



Development segments: Display segment definition E2LFA1M002

Segment type attributes

Segment type	E1LFA1M	Qualified segment
Short Description	Segment for general vendor data	

Segm. definition	E2LFA1M002	✓ Released
Last Changed By	SAP	

Pos	Field Name	Data element	ISO co	Exp
1	MSGFN	MSGFN	☐	3
2	LIFNR	LIFNR	☐	10
3	ANRED	ANRED	☐	15
4	BAHNS	BAHNS	☐	25
5	BBBNR	BBBNR	☐	7
6	BBSNR	BBSNR	☐	5
7	BEGRU	BRGRU	☐	4
8	BRSCH	BRSCH	☐	4
9	BUBKZ	BUBKZ	☐	1
10	DATLT	DATLT	☐	14
11	DTAMS	DTAMS	☐	1
12	DTAWS	DTAWS	☐	2

E75 (4) 800 | ec7server5 | INS

Figure 5.11 The Structure of the Segment E1LFA1M

When you develop a brand new segment, the final point of the development is the act of releasing the segment. At the moment of release, the system generates a dictionary structure with the same name and all of the segment's fields, which means that the segment can be used officially. All standard segments also have a dictionary structure of the same name. So if an IDoc type defined in your system contains a segment `E1LFA1M`, you can declare a variable in your program of the type `E1LFA1M`.

IDoc has a single primary key field—its 16-digit number. We recommend accessing an individual IDoc by the standard function module `IDOC_READ_COMPLETELY`. Besides the control data (which are outside of our current discussion), the function returns all of the IDoc segments in the form of an internal table of structure, `EDIDD`.

Each record contains exactly one segment; the segment's name is stored in field SEGNAM, while segment data are located in an unstructured field, SDATA. An example of a code snippet for IDoc segment processing is provided in Listing 5.2.

```
DATA: lt_edidd TYPE TABLE OF edidd,
      IDoc_number TYPE edidc-docnum,
      ls_e1lfa1m_segment TYPE e1lfa1m,
      ls_e1lfb1m_segment TYPE e1lfb1m.

FIELD-SYMBOLS: <edidd> TYPE edidd.

CALL FUNCTION 'IDOC_READ_COMPLETELY'
  EXPORTING
    document_number = IDoc_number
  TABLES
    int_edidd       = lt_edidd
  EXCEPTIONS
    OTHERS          = 3.

IF sy-subrc <> 0.
  MESSAGE ID sy-msgid TYPE sy-msgty NUMBER sy-msgno
          WITH sy-msgv1 sy-msgv2 sy-msgv3 sy-msgv4.
ENDIF.

LOOP AT lt_edidd ASSIGNING <edidd>.
  CASE <edidd>-segnam.
    WHEN 'E1LFA1M'.
      ls_e1lfa1m_segment = <edidd>-sdata.
*     Processing...

    WHEN 'E1LFB1M'.
      ls_e1lfb1m_segment = <edidd>-sdata.
*     Processing...

      ......

    WHEN OTHERS.
*     Processing non-standard segments...

  ENDCASE.
ENDLOOP.
```

Listing 5.2 IDoc Segment Processing Code

Note that you can freely use direct assignment between unstructured field EDIDD-SDATA and the structured field of the segment despite the Unicode. This is possible because IDoc segment structure contains only character fields; EDIDD-SDATA is character typed as well.

Available BAdIs in Customer Data IDoc Processing

SAP standard batch-input program RFBIDE00 is called from within the function module ERP_IDOC_INPUT_DEBITOR. First, it calls method CHECK_ADD_ON_ACTIVE of BAdI definition CUSTOMER_ADD_DATA. All other methods of the BAdI definition CUSTOMER_ADD_DATA_BI are only called if there is at least one active add-on.

During the IDoc processing, function ERP_IDOC_INPUT_DEBITOR invokes the following methods of the BAdI definition CUSTOMER_ADD_DATA_BI:

- PASS_NON_STANDARD_SEGMENT
  This method is called when the system encounters an unknown segment during the main loop of IDoc segments processing. This call allows you to convert a nonstandard segment into an internal structure for later processing. The segment name and segment data are passed to the method as import parameters.
- MODIFY_BI_STRUCT_FROM_STD_SEG
  This method is called after fulfilling all standard processing for each standard segment. The method uses the segment name and segment data as import parameters, and one changing parameter with an already known structure, BDIFIBIWA. By the moment of the call, structure BDIFIBIWA is filled with standard values, and you can change it according to your requirements.
- FILL_BI_TABLE_WITH_OWN_SEGMENT
  This method is called when all standard batch-input data are saved into the internal table of structure BDIFIBIWA. This method has a changing table parameter with this structure and an import parameter of dictionary structure CUSTOMER_ORG_DATA. When this method is called, you should process the data that were prepared earlier and saved by the PASS_NON_STANDARD_SEGMENT method.
- CHECK_DATA_ROW
  When all segments are processed and all of the data gathered into the batch-input table of structure BDIFIBIWA, the system checks the data before starting the batch input. This method is called for each line of batch-input data if it contains the name of a nonstandard structure. The method has import parameter of structure BDIFIBIWA and a flag parameter for passing the data check status ("X"

for success and a blank space for failure). If some of the data have not passed the check, the method can return an error message through the corresponding export parameters.

- FILL_FT_TABLE_USING_DATA_ROWS
  This method is called just before calling the transaction in batch-input mode. It allows the user to make final alterations into the batch-input screen and field value sequence. Note that this method is only called if the transaction to be called is either XD01 or XD02. This method has a changing table parameter typed with structure BDCDATA.

Available BAdIs in Vendor Data IDoc Processing

Function module ERP_IDOC_INPUT_CREDITOR works with BAdIs in a slightly different way: It calls BAdI VENDOR_ADD_DATA and method CHECK_ADD_ON_ACTIVE after gathering information into an intermediary internal table of structure BDIFIBIWA, instead of at the beginning of IDoc processing.

The following methods of BAdI definition VENDOR_ADD_DATA_BI are called during the processing of the vendor master IDoc:

- PASS_NON_STANDARD_SEGMENT
- MODIFY_BI_STRUCT_FROM_STD_SEG
- FILL_BI_TABLE_WITH_OWN_SEGMENT
- CHECK_DATA_ROW
- FILL_FT_TABLE_USING_DATA_ROWS

Enhancement Spots

Function group VV02 has two entries of enhancement spot ES_SAPLVV02CORE. One source code plug-in entry of this spot is located in the top include of the function group and allows you to use your own includes here. Another spot entry can be found at the beginning of the function code ERP_IDOC_INPUT_DEBITOR. On the vendor side, there is an enhancement spot—ES_SAPLKD02—with the same functionality.

Function Module Exits

There are also components of old-styled function module exit VSV00001, which you can examine in Transaction SMOD. Customer function EXIT_SAPLKD02_001 is called after the vendor data IDoc is completely processed and allows you to save additional data in the database. Customer function `EXIT_SAPLVV02_001` has the same purpose; it is called after processing the customer data IDoc.

5.2 Postings Inbound Scenarios

Now let's examine how accounting document data can come from the external world and what we can do with it.

5.2.1 Batch-Input or Direct Input

As with master data, an initial stage of an SAP ERP implementation project virtually always requires loading initial accounting transaction data. A traditional tool for this activity is the standard SAP report RFBIBL00. Input data for the report are provided in the form of a flat file located on the application server. The report is suitable for use with SAP LSMW, which effectively hides all the file preparation issues.

Internally, the report uses function modules of group `FIPI`, which are listed in Table 5.2.

Name	Description
`POSTING_INTERFACE_CLEARING`	Post with clearing (`FB05`) using internal posting interface.
`POSTING_INTERFACE_DOCUMENT`	Post document using the internal posting interface.
`POSTING_INTERFACE_END`	The ending function of the group. Should be called in the end of the process.
`POSTING_INTERFACE_RESET_CLEAR`	Reset clearing via posting interface.
`POSTING_INTERFACE_REVERSE_DOC`	Cancel document via posting interface.
`POSTING_INTERFACE_START`	Initial information for internal accounting interface.

Table 5.2 FIPI Function Group Modules

These functions actually make postings through batch input by generating sessions or calling a transaction directly. The function modules also have detailed system documentation. Unfortunately, report RFBIBL00 does not contain a user-exit call, although you can rely on the user exits available inside the transactions that are called during processing.

5.2.2 Payroll Results

Note that the payroll result posting interface is fully equipped with specific user exits. However, it's worth seeing the overall process outline so that you can understand where and when the process should (or should not) be intercepted, depending on your business requirements.

If the company has SAP HR Payroll implemented, then in every payroll period (weekly or monthly), there must be an interface running that posts payroll results to the Financials department. SAP recommends implementing HR as a separate system to improve data security because payroll data are among the most sensitive corporate data.

If you are implementing a payroll results posting from SAP HR into SAP FI, then in the end, the posting will be performed with the same tools.

The whole process of HR payroll posting looks like this:

1. The responsible person in HR creates a payroll posting run with report RPCIPE00. The report creates a preliminary posting document stored in Tables PPDHD, PPDIT, and others.
2. Someone then checks and approves all of the resulting posting documents (they are not accounting documents) by editing particular payroll runs with Transaction PCP0.
3. Finally, someone runs report RPCIPP00 to transfer values into accounting.

The last step can be performed either via ALE/IDoc interfaces (if HR Payroll works as a separate system), or locally—by direct call of an accounting BAPI. By default, all of HR Payroll IDocs are processed in the receiving system by the same BAPI. Let's trace the chain.

The HR system generates three types of postings:

- **Employee expenses**
  For example, travel and accommodation when on a business trip.
- **Employee vendor items**
  For example, an employee can be treated as a corporate vendor or service provider to justify salary payment; thus the document is generated as an Account Payables item.

- **Employee customer items**
  If an employee has debts that are not settled, he might appear in the role of a corporate customer; the document is generated as an Account Receivables item.

If an HR Payroll component is implemented as a separate system, then it generates three types of IDocs: `ACC_EMPLOYEE_PAY02`, `ACC_EMPLOYEE_REC02`, and `ACC_EMPLOYEE_EXP02`. In the receiving system, these IDocs are linked by default via the ALE/BAPI-generated interface to the following function modules:

- `IDOC_INPUT_ACC_EMPLOYEE_EXP` for employee expenses
- `IDOC_INPUT_ACC_EMPLOYEE_PAY` for employee payments
- `IDOC_INPUT_ACC_EMPLOYEE_REC` for employee debts

The accounting documents are generated with BAPI calls:

- `BAPI_ACC_EMPLOYEE_EXP_POST` for employee expenses
- `BAPI_ACC_EMPLOYEE_PAY_POST` for employee payments
- `BAPI_ACC_EMPLOYEE_REC_POST` for employee debts

Finally, each of the BAPIs call function modules `AC_DOCUMENT_CREATE` and `AC_DOCUMENT_POST` as a low-level accounting interface utility. Thus, you can employ any user exit (BAdI or BTE) appearing in the `AC_DOCUMENT_CREATE` function module (see Chapter 3, Posting to Accounting), including substitutions and validations.

At the call point of a user exit during the document generation, you can distinguish SAP standard HR Payroll postings from any others by the contents of the field `BKPF-GLVOR`:

- `HRP1` for employee expenses
- `HRP3` for employee payments (Account Payables)
- `HRP2` for employee debts (Accounts Receivable)

5.2.3 Postings via IDoc

The SAP system delivers dozens of IDoc types to be used for posting different flavors of accounting documents: direct posting to a general ledger account, posting of incoming vendor invoice, and so on. You can find corresponding IDoc types in Transaction WE30 (Executing the Search Help with Mask ACC*). However, if you look into the processing function modules, you'll notice that they aren't equipped with user exits. If you thoroughly trace the chain of calls, you'll see that this chain is ended at the same function modules mentioned in the previous section: AC_DOCUMENT_CREATE and AC_DOCUMENT_POST. Thus, you should rely on already-known user exits discussed in Chapter 3.

5.2.4 Electronic Bank Statement

The process of loading a bank statement file consists of two phases: importing the bank statement file in Transaction FF_5, and posting the bank statement through Transaction FEBP.

Importing the Bank Statement File

Loading program RFEBKA00, which is linked to Transaction FF_5, parses incoming bank files according to a selected format, such as Multicash or SWIFT MT940, which are widely used in bank communication. Each individual file format is parsed in an external program, although the code in report RFEBKA00 that is responsible for choosing the format parsing program is quite static; there is just a CASE statement with no configuration.

However, if you look into the source code of format SWIFT MT940 parsing routine program RFEKA400, you can discover an old-fashioned user exit, EXIT_RFEKA400_001, belonging to function module exit FEB00004. The enhancement can be used for preprocessing raw file data, which is passed to the user exit in the form of a table parameter with a length of 512 unstructured lines. Listing 5.3 shows the interface of the user exit.

```
FUNCTION EXIT_RFEKA400_001.
*"--------------------------------------------------------------
*"*"Lokale Schnittstelle:
*"       TABLES
*"              T_RAW_DATA STRUCTURE  RAW_DATA
*"       EXCEPTIONS
*"              ERROR_OCCURED
*"--------------------------------------------------------------


INCLUDE ZXF01U06 .


ENDFUNCTION.
```

Listing 5.3 EXIT_RFEKA400_001 Interface

You can also see that `EXIT_RFEKA400_001` has one exception, which signals the host program to stop processing the file any further.

Report RFEBKA00 gathers parsed data into the following bank statement database tables:

- FEBKO (electronic bank statement header records)
- FEBEP (electronic bank statement line items)
- FEBRE (reference record for electronic bank statement line item)

Posting the Bank Statement

When you link report RFEBKA30 to Transaction FEBP, it interprets data in bank statement tables and makes an accounting posting. A bank statement is a list of operations of what the bank did with your money on your behalf, such as company payments to vendors, bank charges for its services, interest payments, payments from your customers, and so on. All of these operations should be correctly reflected in the company's financial accounting to make sure that the money flow is consistent and correct.

At the same time, the bank's statement can use different identification for the same objects presented in your system; also, it's possible that some valuable data in the context of your SAP ERP system may be omitted in the statement for one reason or another. During the interpretation phase, report RFEBKA30 is trying to fill these gaps automatically, for example, to determine the business partner number for the bank transaction or even more important to determine the clearing reference (e.g., payment against invoice) document numbers.

Report RFEBKA30 actually is only a wrapper for another report, RFEBBU10, which performs the interpretation. The algorithm runs through header-item relation of two tables, FEBKO and FEBEP. For each FEBEP internal loop run, the report calls different user exits that can help discover missing statement data.

Now let's walk through the available BTEs you can employ during the processing of a bank statement.

BTE 00002810 and Process 00002820

First, the system calls BTE `00002810` (you can see its interface in Listing 5.4). The event has a pair of parameters for the header record and for the line item of the bank statement that is being processed. The parameter with suffix `EXT` contains fields with external data (records that were sent by the bank), whereas suffix INT signifies that this data is internal. As a result of its run, each function module that is subscribed to the 00002810 event must return a registration flag in one of two export parameters: E_REGISTER_AREA_1 or E_REGISTER_AREA_2.

```
*"----------------------------------------------------------------------
*"*"Lokale Schnittstelle:
*"  IMPORTING
*"     VALUE(I_FEBKO_EXT) LIKE  FEBKOXT_BF STRUCTURE  FEBKOXT_BF
*"     VALUE(I_FEBEP_EXT) LIKE  FEBEPXT_BF STRUCTURE  FEBEPXT_BF
*"     VALUE(I_FEBKO_INT) LIKE  FEBKOIN_BF STRUCTURE  FEBKOIN_BF
*"     VALUE(I_FEBEP_INT) LIKE  FEBEPIN_BF STRUCTURE  FEBEPIN_BF
*"     VALUE(I_TESTRUN) TYPE  XFLAG OPTIONAL
*"  EXPORTING
*"     VALUE(E_REGISTER_AREA_1) LIKE  BOOLE-BOOLE
*"     VALUE(E_REGISTER_AREA_2) LIKE  BOOLE-BOOLE
*"     VALUE(E_SUPPR_STD_AREA_1) LIKE  BOOLE-BOOLE
*"     VALUE(E_SUPPR_STD_AREA_2) LIKE  BOOLE-BOOLE
*"  TABLES
*"      T_FEBRE STRUCTURE  FEBRE_BF
*"      T_FEBCL STRUCTURE  FEBCL_BF
*"----------------------------------------------------------------------
```

Listing 5.4 The Interface of BTE 00002810

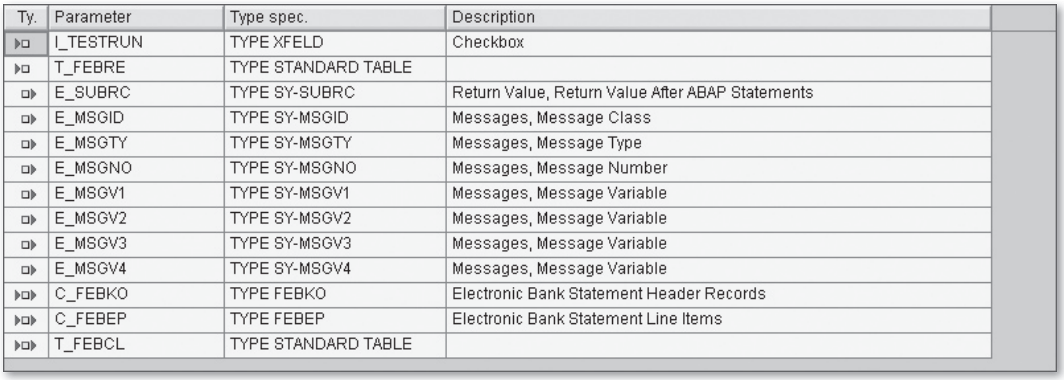
Note that subscribers to event 00002810 are called from within function FEB_OPEN_FI_CALL_1. This function allows only one application ID to be registered for each of the two areas. The application ID in the BTE framework is used to distinguish SAP internal and partner application areas. Customer-defined P&S modules and processes can have blank application IDs. Therefore, you should make sure that for this particular line item of the bank statement, your function is the only one registered, or an error will be reported. Another pair of event flag parameters, E_SUPPR_STD_AREA_1 and E_SUPPR_STD_AREA_1, will prevent execution of interpretation algorithm if they are assigned X.

Process 00002820 is called just after the event and only for registered application IDs. You can see the process interface in Listing 5.5. Note that there are export parameters to allow changing values in bank statement headers and items. Note that your changed data will be taken into account only if you assign X to the export parameter E_UPDATE_FEB.

```
*"----------------------------------------------------------------
*"*"Lokale Schnittstelle:
*"  IMPORTING
*"     VALUE(I_FEBKO_EXT) LIKE  FEBKOXT_BF STRUCTURE  FEBKOXT_BF
*"     VALUE(I_FEBEP_EXT) LIKE  FEBEPXT_BF STRUCTURE  FEBEPXT_BF
*"     VALUE(I_FEBKO_INT) LIKE  FEBKOIN_BF STRUCTURE  FEBKOIN_BF
*"     VALUE(I_FEBEP_INT) LIKE  FEBEPIN_BF STRUCTURE  FEBEPIN_BF
*"     VALUE(I_TESTRUN) TYPE  XFLAG OPTIONAL
*"  EXPORTING
*"     VALUE(E_FEBKO_EXT) LIKE  FEBKOXT_BF STRUCTURE  FEBKOXT_BF
*"     VALUE(E_FEBEP_EXT) LIKE  FEBEPXT_BF STRUCTURE  FEBEPXT_BF
*"     VALUE(E_FEBKO_INT) LIKE  FEBKOIN_BF STRUCTURE  FEBKOIN_BF
*"     VALUE(E_FEBEP_INT) LIKE  FEBEPIN_BF STRUCTURE  FEBEPIN_BF
*"     VALUE(E_UPDATE_FEB) LIKE  BOOLE-BOOLE
*"  TABLES
*"      T_FEBRE STRUCTURE  FEBRE_BF
*"      T_FEBCL STRUCTURE  FEBCL_BF
*"----------------------------------------------------------------
```

Listing 5.5 The Interface of BTE Process 00002820

Besides header and item data, you can also fill in clearing data in table parameter T_FEBCL.



Ty.	Parameter	Type spec.	Description
▶▫	I_TESTRUN	TYPE XFELD	Checkbox
▶▫	T_FEBRE	TYPE STANDARD TABLE	
▫▶	E_SUBRC	TYPE SY-SUBRC	Return Value, Return Value After ABAP Statements
▫▶	E_MSGID	TYPE SY-MSGID	Messages, Message Class
▫▶	E_MSGTY	TYPE SY-MSGTY	Messages, Message Type
▫▶	E_MSGNO	TYPE SY-MSGNO	Messages, Message Number
▫▶	E_MSGV1	TYPE SY-MSGV1	Messages, Message Variable
▫▶	E_MSGV2	TYPE SY-MSGV2	Messages, Message Variable
▫▶	E_MSGV3	TYPE SY-MSGV3	Messages, Message Variable
▫▶	E_MSGV4	TYPE SY-MSGV4	Messages, Message Variable
▶▫▶	C_FEBKO	TYPE FEBKO	Electronic Bank Statement Header Records
▶▫▶	C_FEBEP	TYPE FEBEP	Electronic Bank Statement Line Items
▶▫▶	T_FEBCL	TYPE STANDARD TABLE	

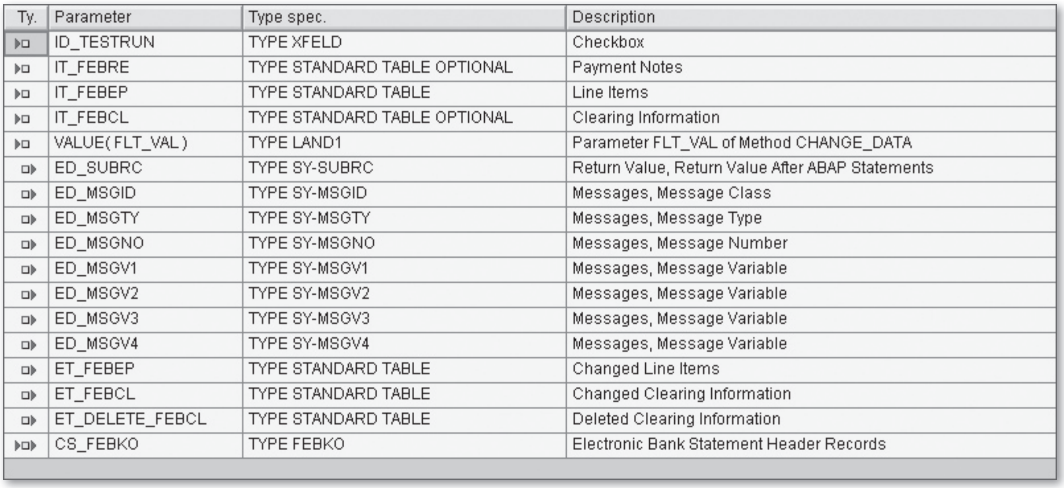
Figure 5.12 The Signature of Method CHANGE_DATA of BAdI FIEB_CHANGE_BS_DATA

BAdI Definitions

Progressing to business transaction events and processes, the system calls BAdI definition FIEB_CHANGE_BS_DATA and method CHANGE_DATA. Figure 5.12 shows the interface (or signature) of the method. Notice that the method has three changing parameters: `C_FEBKO` and `C_FEBEP` for the header and item of the bank statement, and table parameter `T_FEBCL` for clearing data from the statement.

The method can also return error code and error message attributes to be reported in the log and prevent the statement from being processed further.

Another BAdI definition, `FIEB_CHANGE_STATEMNT`, is called after all of the interpretation is executed, and the system has done everything it can. You can see the interface of the BAdI method `CHANGE_DATA` in Figure 5.13.



Ty.	Parameter	Type spec.	Description
▶□	ID_TESTRUN	TYPE XFELD	Checkbox
▶□	IT_FEBRE	TYPE STANDARD TABLE OPTIONAL	Payment Notes
▶□	IT_FEBEP	TYPE STANDARD TABLE	Line Items
▶□	IT_FEBCL	TYPE STANDARD TABLE OPTIONAL	Clearing Information
▶□	VALUE(FLT_VAL)	TYPE LAND1	Parameter FLT_VAL of Method CHANGE_DATA
□▶	ED_SUBRC	TYPE SY-SUBRC	Return Value, Return Value After ABAP Statements
□▶	ED_MSGID	TYPE SY-MSGID	Messages, Message Class
□▶	ED_MSGTY	TYPE SY-MSGTY	Messages, Message Type
□▶	ED_MSGNO	TYPE SY-MSGNO	Messages, Message Number
□▶	ED_MSGV1	TYPE SY-MSGV1	Messages, Message Variable
□▶	ED_MSGV2	TYPE SY-MSGV2	Messages, Message Variable
□▶	ED_MSGV3	TYPE SY-MSGV3	Messages, Message Variable
□▶	ED_MSGV4	TYPE SY-MSGV4	Messages, Message Variable
□▶	ET_FEBEP	TYPE STANDARD TABLE	Changed Line Items
□▶	ET_FEBCL	TYPE STANDARD TABLE	Changed Clearing Information
□▶	ET_DELETE_FEBCL	TYPE STANDARD TABLE	Deleted Clearing Information
▶□▶	CS_FEBKO	TYPE FEBKO	Electronic Bank Statement Header Records

Figure 5.13 The Signature of Method CHANGE_DATA of the BAdI FIEB_CHANGE_STATEMNT

Customer-Defined Interpretation Algorithm

After calling BTEs and the first BAdI, the system runs the interpretation proper. Each bank statement item can have its own interpretation algorithm, which is defined by the field `FEBEP-INTAG` value. Therefore, the individual item algorithm can be set during a user exit run: either BTE or BAdI.

A full list of interpretation algorithm numbers and descriptions can be found in the `INTAG_EB` domain fixed values. `INTAG_EB` is numeric 3. It is assumed that all SAP system algorithms belong to the range of `INTAG` values from `000` to `899`, and everything above `900` is a customer-defined interpretation.

To implement a customer-defined interpretation, you have to create a function module with a predefined name structure—Z_FIEB_NNN_ALGORITHM—where NNN is the algorithm number.

This function module must have the interface shown in Listing 5.6.

```
FUNCTION Z_FIEB_901_ALGORITHM.
*"----------------------------------------------------------------------
*"*"Local Interface:
*"  IMPORTING
*"     REFERENCE(I_NOTE_TO_PAYEE) TYPE  STRING
*"     VALUE(I_COUNTRY) TYPE  T001-LAND1
*"  TABLES
*"      T_AVIP_IN STRUCTURE  AVIP
*"      T_AVIP_OUT STRUCTURE  AVIP
*"      T_FILTER1
*"      T_FILTER2
*"----------------------------------------------------------------------

ENDFUNCTION.
```

Listing 5.6 Sample Interpretation Algorithm Function

Based on the payment note passed to the function in parameter I_NOTE_TO_PAYEE and document references in T_AVIP_IN, the algorithm is expected to produce reasonable results in table structure T_AVIP_OUT, which has the structure of the payment advice line item. Table structure T_AVIP_OUT is then used to update the clearing reference data for the statement item.

Function Module Exit

After the interpretation algorithm and just before the second BAdI call, the system invokes a component (function module) EXIT_RFEBBU10_001 of the old-fashioned function module exit FEB00001. Its interface is shown in Listing 5.7.

```
FUNCTION EXIT_RFEBBU10_001.
*"----------------------------------------------------------------------
*"*"Lokale Schnittstelle:
*"  IMPORTING
*"     VALUE(I_FEBEP) LIKE  FEBEP STRUCTURE  FEBEP
*"     VALUE(I_FEBKO) LIKE  FEBKO STRUCTURE  FEBKO
*"     VALUE(I_TESTRUN) TYPE  XFLAG
*"  EXPORTING
*"     VALUE(E_FEBEP) LIKE  FEBEP STRUCTURE  FEBEP
*"     VALUE(E_FEBKO) LIKE  FEBKO STRUCTURE  FEBKO
*"     VALUE(E_MSGTEXT) LIKE  FEBMKA-MESSG
*"     VALUE(E_MSGTYP) LIKE  FEBMKA-MSTYP
*"     VALUE(E_UPDATE) LIKE  FEBMKA-MSTYP
*"  TABLES
*"      T_FEBCL STRUCTURE  FEBCL
*"      T_FEBRE STRUCTURE  FEBRE
*"----------------------------------------------------------------------

  INCLUDE ZXF01U01.

ENDFUNCTION.
```

Listing 5.7 The Interface of EXIT_RFEBBU10_001

This is another point where you can intercept the standard flow of the bank statement processing.

5.3 Summary

In this chapter, we discussed several inbound interfaces of Financial Accounting, which cover some of the general corporate activities. Thanks to the SAP design in all of these scenarios, you can find ways to seamlessly tailor the standard process for specific corporate needs.

In the next chapter, you'll see what user-exit techniques are available for development in outbound scenarios when the system sends accounting data to external systems.

Index