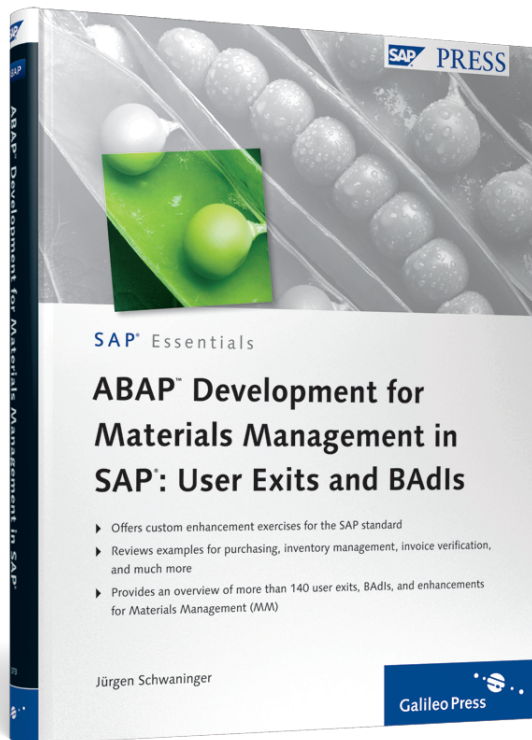


Jürgen Schwaninger

ABAP™ Development for Materials Management in SAP®: User Exits and BADIs



Contents at a Glance

1	Introduction	13
2	General Information on User Exits and BAdIs	17
3	User Exits and BAdIs in Purchasing	37
4	User Exits and BAdIs in External Services Management	83
5	User Exits and BAdIs in Inventory Management	93
6	User Exits and BAdIs in the Valuation and Account Assignment Area	139
7	User Exits and BAdIs in Logistics Invoice Verification	145
8	Validation and Substitution of Accounting Documents	171
A	User Exits and BAdIs in SAP Materials Management	193
B	The Author	257

Contents

Preface	11
1 Introduction	13
1.1 Objectives	13
1.2 Structure and Content	13
1.3 Target Audience	14
1.4 Prerequisites	14
2 General Information on User Exits and BAdIs	17
2.1 Using User Exits	17
2.1.1 Finding and Viewing Enhancements	18
2.1.2 Creating a Project and Assigning Enhancements	18
2.1.3 Using Components of the Project	19
2.1.4 Activating and Deactivating Projects	21
2.2 Use of Classic BAdIs	22
2.2.1 Finding and Viewing Enhancements	22
2.2.2 Creating a BAdI Implementation	24
2.2.3 Working with Methods	25
2.2.4 Activating and Deactivating BAdIs	27
2.2.5 Enhanced Editing Options	28
2.3 Use of New BAdIs (Enhancement Spots)	29
2.3.1 SAP Enhancement Framework	29
2.3.2 Finding and Viewing Enhancement Spots	30
2.3.3 Creating Enhancement Implementations	31
2.3.4 Working with Methods	33
2.3.5 Activating and Deactivating BAdIs	34
3 User Exits and BAdIs in Purchasing	37
3.1 Customized Fields in Purchase Orders	37
3.1.1 Overview of the Implementation	38
3.1.2 Implementation of Custom Purchase Order Data and Function Group	42

3.1.3	Integration of Custom Fields into the BAdIs	54
3.1.4	Integration of Customer Fields into the Business Logic ...	61
3.1.5	Initializing, Reading, and Updating Data	68
3.1.6	Display of Error Messages	69
3.2	Customizing the Document Overview in Purchase Requisitions or Purchase Orders	72
3.2.1	Removal of a Standard Selection Variant	72
3.2.2	Inserting Custom Selection Variants	75

4 User Exits and BAdIs in External Services Management 83

4.1	Prepopulating Account Assignment for Service Lines	83
4.2	Input Check of the Service Lines	86
4.2.1	Prepopulating Fields in EXIT_SAPLMLSP_030	87
4.2.2	Input Check in EXIT_SAPLMLSP_031	88
4.3	Prepopulation of the Header Data in the Data Entry Sheet	89

5 User Exits and BAdIs in Inventory Management 93

5.1	Custom Fields in Transaction MIGO	93
5.1.1	Custom Fields: An Overview	94
5.1.2	Preparations in the ABAP Dictionary	97
5.1.3	Preparation of the Function Group	98
5.1.4	Preparation and Status Management in MB_MIGO_BADI	104
5.1.5	Activation of Custom Header Data	109
5.1.6	Activation of Custom Item Data	112
5.1.7	Updating the Data	117
5.2	Other Functions of the BAdI MB_MIGO_BADI	118
5.2.1	Noting Custom Data	119
5.2.2	Input Checks in Transaction MIGO	123
5.3	Checking and Prepopulating Standard Fields	125
5.3.1	Prepopulation of Storage Location and Text	125
5.3.2	Checking the Standard Fields	126
5.4	Check of the Earliest Delivery Date	127
5.5	Tolerance Limits for Scheduling Agreements	129
5.5.1	Overwriting Overdelivery Quantity	129
5.5.2	Overwriting Default Quantity	132

5.6	Enhancement of Reservations	134
5.6.1	Prepopulating Fields	135
5.6.2	Checking Entries	137
6	User Exits and BADIs in the Valuation and Account Assignment Area	139
6.1	GR/IR Clearing Account	139
6.2	Overriding the Account Determination in the User Exit	141
7	User Exits and BADIs in Logistics Invoice Verification	145
7.1	Custom Fields in Transaction MIRO	145
7.1.1	Overview of the Solution via BAdI	146
7.1.2	BAdI in Detail — Customizations in the ABAP Dictionary	148
7.1.3	Creating a Custom Dynpro with Table Control	151
7.1.4	Preparation of the Data in the BAdI	154
7.1.5	Back to the Dynpro	158
7.2	Overriding Tolerance Checks	164
7.2.1	Tolerance Limits in Customizing	165
7.2.2	Use of the Enhancement	166
8	Validation and Substitution of Accounting Documents	171
8.1	Validation of Accounting Documents	172
8.1.1	Callup Points	172
8.1.2	Steps	173
8.1.3	Example without Exit Routine	174
8.1.4	Example with Exit Routine	177
8.2	Substitution of Accounting Documents	182
8.2.1	Substitution without Exit Routine	183
8.2.2	Substitution with Exit Routine	185
8.2.3	Read Access to Data of the Source Document	188

Appendices	193
A User Exits and BADIs in SAP Materials Management	193
A.1 Purchasing	194
A.1.1 Purchase Order Requisitions in General	194
A.1.2 Purchase Orders in General	199
A.1.3 Outline Agreements (Scheduling Agreements/ Contracts)	205
A.1.4 Pricing	208
A.1.5 Commitment Functions	210
A.1.6 Cross-Document	211
A.1.7 Vendor Evaluation	215
A.1.8 IDoc Processing	216
A.1.9 Logistics Information Systems	218
A.1.10 Archiving	218
A.2 External Services Management	221
A.3 Inventory Management	227
A.3.1 Material Documents in General	227
A.3.2 Goods Receipt	233
A.3.3 Reservations	235
A.3.4 Archiving	236
A.4 Valuation and Account Assignment	238
A.5 Logistics Invoice Verification	240
A.5.1 General	240
A.5.2 Archiving	254
A.5.3 Conventional Invoice Verification	254
B The Author	257
Index	259

5 User Exits and BAdIs in Inventory Management

The concept of Inventory Management in SAP Materials Management (MM) basically involves summarizing the management of all warehouse stocks with regard to value and quantity as well as the associated goods movements. Because the conceivable processes in this core area are very comprehensive, there are also many setting options in the system. This is also reflected in the number of possible program-related enhancements. In this chapter, you will learn about the most important enhancements for Inventory Management.

5.1 Custom Fields in Transaction MIGO

With the implementation of the central Transaction MIGO for all goods movements, a powerful BAdI is provided: MB_MIGO_BADI. With this BAdI, you can integrate and post custom fields as custom tabstrips at the header and item level. You can also carry out input checks in your custom fields or populate many standard fields with default values.

Display of Custom Data

If you execute the following example, you may find that the subsequent display of the data of a posted material document at item level doesn't work. This is the case when SAP Note 1029951 has been manually imported into your system or support packages have been imported.

As a result of the changes from this note, the method `LINE_MODIFY`, which is actually provided for reading the item data from the database, will no longer run when material documents are displayed. Unfortunately, it isn't possible to use another method for this because the associated standard item data (structure `GOITEM`) isn't provided in all other relevant methods, and you therefore cannot assign your data to a specific document line.

With SAP Note 1477221, the method `LINE_MODIFY` will run again. If this note hasn't yet been released for customers when you read these lines, you can modify the program location relevant for the call of the method, as described in SAP Note 1136344. The example given here will be fully functional in both cases.

5.1.1 Custom Fields: An Overview

The capability of the BAdI MB_MIGO_BADI results in a certain amount of complexity. There are 17 methods that are called at many different times. Nevertheless, to make the next example as clear as possible, you'll only view the implementation of custom fields at the header and item level (see Figure 5.1). This reduces the BAdI to 11 methods with which the minimum requirements can be fulfilled. In a second step, you'll then learn about the function of the remaining methods in a brief overview.

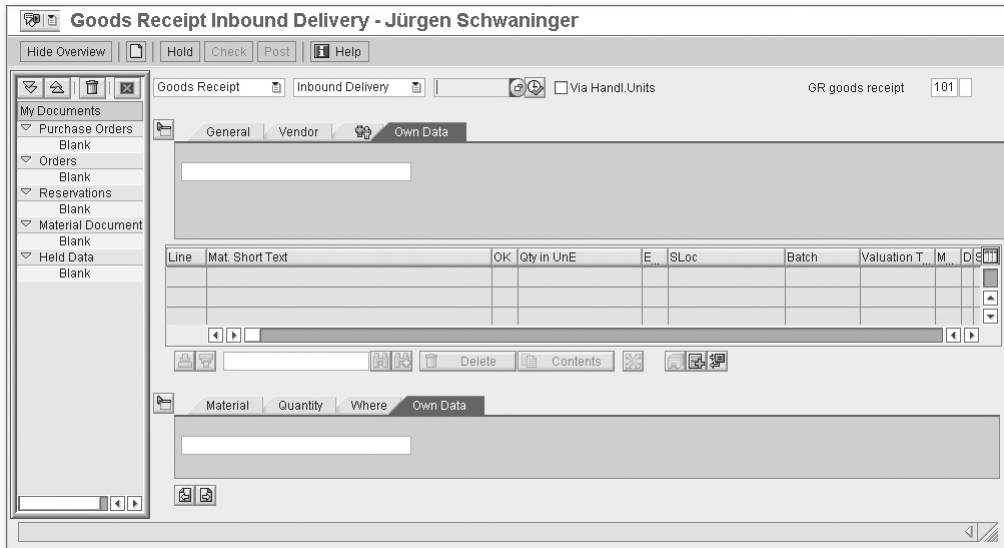


Figure 5.1 Custom Fields in Transaction MIGO

Function Group for the Management of Dynpros

Just as with the custom fields in the purchase order (see Section 3.1, Custom Fields in Purchase Orders, in Chapter 3), you first need a function group to include the dynpros and some function modules for the exchange of data and for updating custom fields. However, this function group can be implemented for the BAdI MB_MIGO_BADI much more easily and must assume fewer tasks.

You have to work with two BADIs in the purchase order to activate the custom fields. The function group therefore not only has to assume the communication with the dynpros but also must map the communication between the BADIs. You can, however, execute all the necessary steps directly in the BAdI MB_MIGO_BADI for

Transaction MIGO, so additional communication with another BAdI isn't needed. You can create a dynpro in the function group for the header and item data and create two function modules for each dynpro to store and retrieve the data in the dynpro. You also need an update module to save your custom data. To post your custom data, you need to define custom tables in the ABAP Dictionary.

Usage of the BAdI MB_MIGO_BADI

As soon as you've created the function group, you can begin with the implementation of the BAdI. You keep the data on the custom fields for runtime in the attributes of the BAdI. First, these areas need to be defined.

MB_MIGO_BADI can be implemented many times. However, there are only five custom tabstrips for the header and item levels. For this reason, precisely five active implementations are allowed. The method `INIT` is activated when Transaction MIGO is started, and it's used to register an implementation and populate a tabstrip.

With the initialization of a document (new document or display of an existing document), the methods will run from Table 5.1. The methods `MODE_SET` and `STATUS_AND_HEADER` are also triggered when the document status changes.

Method	Description
RESET	This method is used to initialize all custom data in the attributes of the BAdI implementation.
MODE_SET	In this method, you obtain information on the action (goods receipt, goods issue, etc.) and the chosen reference (purchase order, reservation, etc.) chosen by the user; these are the fields that are always available in Transaction MIGO in the top-left side. You can evaluate this information, and store it in the attributes of the implementation for subsequent use.
STATUS_AND_HEADER	This method is mainly used to fill the custom header data when an existing document is read.

Table 5.1 Initialization of a Document and Status Change

The methods are also executed with each dialog step for header data from Table 5.2 and with each dialog step on item data from Table 5.3. The method `LINE_MODIFY` is also executed when a line is added either when the user makes an entry or when an existing material document is read from the database.

Method	Description
PBO_HEADER	This method is called before the header data are displayed. You must transfer the data to be displayed to the function group and specify the dynpro to be displayed. The tabstrip header is also specified here.
PAI_HEADER	This method runs as soon as the user has entered the data. You need to retrieve the possibly changed data from the function group.

Table 5.2 Methods in the Dialog Step on Header Data

Method	Description
PBO_DETAIL	This method is called before the item data is displayed. In the parameter <code>I_LINE_ID</code> , you obtain the current item and must transfer the associated data in the function group. You also need to specify the dynpro to be displayed as well as the header of the tabstrip here.
PAI_DETAIL	You retrieve the data from the dynpro after the user has entered it. First, check whether data has changed, and if applicable, set the parameter <code>E_FORCE_CHANGE</code> . The method <code>LINE_MODIFY</code> is activated when the parameter has been set.
LINE_MODIFY	This method is activated when a line has been changed (see <code>PAI_DETAIL</code>) or when a new line has been inserted. When adding new lines, you must initialize your custom fields, or if the document is being read in the display mode, you must read the data from the database. If required, you have the option here of writing standard fields that you've also possibly integrated in your dynpro back to the standard items.
LINE_DELETE	This method runs when the user has deleted a line of the material document. In this case, you also need to delete the associated custom data.

Table 5.3 Methods in the Dialog Step for Item Data

Finally, you must use the method `POST_DOCUMENT` to format your custom data and call the update module from your function group. The final line number for each document line is set only in this method, as contained in Table `MSEG` — that is, in

the table in which the standard item data is stored. You should convert your custom data accordingly so that you can assign it more easily later on.

5.1.2 Preparations in the ABAP Dictionary

At this point, it makes sense to refer again to the general data definitions in the ABAP Dictionary. Both header and item data are implemented in this example, and two suitable database tables are required to store the data. The following example again only uses a simple variant for this, and only one text field is used as a custom field.

1. Switch to the ABAP Dictionary (Transaction SE11), and create the table for the header data. Besides the actual text field ZK_FELD1, you also need the key fields; that is, the material document number and posting year. The table name ZMB_MIGOHEAD has been used in the example. You can view the associated fields in Table 5.4.

Field Name	Key Field	Data Element
MANDT	Yes	MANDT
MBLNR	Yes	MBLNR
MJAHR	Yes	MJAHR
ZK_FELD1		CHAR32

Table 5.4 Fields of the Table ZMB_MIGOHEAD

2. Create the table for the item data. A line number is necessary besides the previously used key fields. The custom field is called ZP_FELD1 here. The table name ZMB_MIGOITEM is used. The associated fields can be viewed in Table 5.5.

Field Name	Key Field	Data Element
MANDT	Yes	MANDT
MBLNR	Yes	MBLNR
MJAHR	Yes	MJAHR
ZEILE	Yes	MBLPO
ZP_FELD1		CHAR32

Table 5.5 Fields of the Table ZMB_MIGOITEM

5.1.3 Preparation of the Function Group

You next need to deal with the function group and the dynpro. In this example, both the header and the item data are implemented. A dynpro is also required here. Moreover, two function modules for the data communication and one update module are necessary in each case. Another module informs the function group on the current status of Transaction MIGO. If this isn't in the display mode, the fields in the dynpros aren't ready for input.

1. Switch to Transaction SE80. Select the FUNCTION GROUP option in the Repository Browser, and specify ZMB_MIGO as the name. Press . Confirm in the dialog that this function group should now be generated.

Dynpro Variations

In this example, a dynpro is defined for the header and item data in each case. You certainly have the option, however, of preparing different situations as well as different dynpros. Your requirement can, for example, be different in goods receipt than in goods issue. You decide which dynpro is actually displayed at runtime via your programming.

In the method `MODE_SET` of the BAdI `MB_MIGO_BADI`, you learn which action (displays, goods receipt, goods issue, etc.) has just been selected. You can keep this information in an attribute of the class and then dynamically set the dynpro in method `PBO_HEADER` or `PBO_DETAIL` based on this information.

2. Create the dynpro for the header data. Now right-click ZMB_MIGO in the object browser, and select `CREATE • DYNPRO` from the context menu. The dynpro number should be 0100. In the properties, you must set `SUBSCREEN` as the `DYNPRO TYPE` because other types cannot be integrated with tabstrips, and you would get a short dump upon implementation.
3. Click the `LAYOUT` button to switch to the Screen Painter. Here you choose `DICTIONARY/PROGRAM FIELDS` in the `GO TO • SECONDARY WINDOW` menu, or press . In the window displayed, specify `ZMB_MIGOHEAD` as the table name and press . Mark the field `ZK_FELD1`, and press again (see Figure 5.2). Using the mouse, now position the field in the top-left corner of the dynpro. Save and activate the dynpro, and then exit the Screen Painter.
4. Create the dynpro 0200 for the item data. Choose the field `ZP_FELD1` from Table `ZMB_MIGOITEM`, and then proceed as in the previous step.

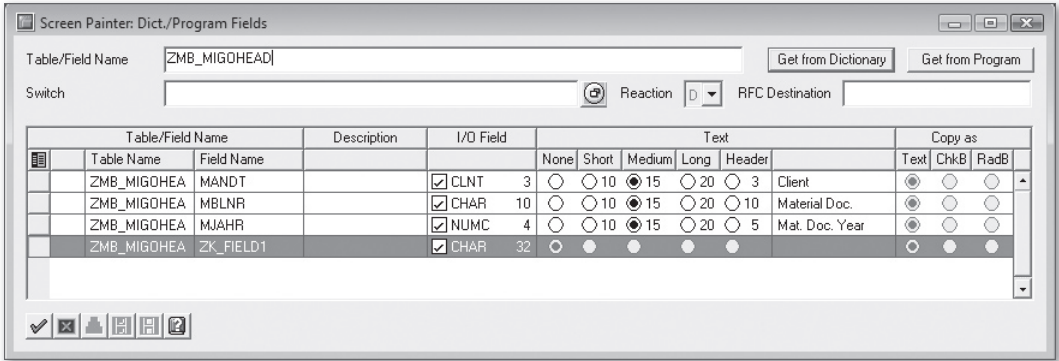


Figure 5.2 Dict./Program Fields Window

5. So that you are able to subsequently fill the fields in these dynpros with data, fields with an identical name must be available in the global data. Create these with the keyword TABLES. Furthermore, the fields are then ready for input only if the document isn't in a display mode. The mode is to be subsequently set via a function module. You can create it now, however, as a flag (gv_outputonly, data type C) in the global data.

You switch to the global data via the Object Browser by navigating to the include LZMB_MIGOTOP in the INCLUDES section. An example is given in Listing 5.1.

```

FUNCTION-POOL ZMB_MIGO.                "MESSAGE-ID ..
* Work structures for dynpros
TABLES: zmb_migohead,
        zmb_migoitem.

* Display mode?
DATA gv_outputonly TYPE c.
    
```

Listing 5.1 Global Data in LZMB_MIGOTOP

6. If the flag gv_outputonly is set, then the fields in the dynpro aren't ready for input. Therefore, change to the flow logic of dynpro 0100. Remove the comments in the PROCESS BEFORE OUTPUT time from the STATUS_0100 module. Double-click the module name. To generate the module, use the suggested include name.

Set the value of SCREEN-INPUT based on the field gv_outputonly with a LOOP AT SCREEN. An example is given in Listing 5.2.

```

*-----*
***INCLUDE LZMB_MIG0001 .
*-----*
MODULE status_0100 OUTPUT.
  LOOP AT SCREEN.
    IF gv_outputonly IS INITIAL.
      screen-input = 1.
    ELSE.
      screen-input = 0.
    ENDIF.

    MODIFY SCREEN.
  ENDLOOP.
ENDMODULE.                " STATUS_0100  OUTPUT

```

Listing 5.2 Flow Logic on Dynpro 0100

7. Dynpro 0200 is also not ready for input in the display mode. Switch to dynpro 0200, and proceed as in the previous step.

Next you can begin with the function modules:

1. An overview of the required modules is given in Table 5.6. Begin with the module ZMB_MIGO_SETSTATUS. You only need a flag as an input parameter; from this, you overwrite the global field `gv_outputonly` (see Listing 5.3).

Function Module	Description
ZMB_MIGO_SETSTATUS	Sets the flag <code>gv_outputonly</code>
ZMB_MIGO_HEAD_SET	Sets data for dynpro 100
ZMB_MIGO_HEAD_GET	Returns data from dynpro 100
ZMB_MIGO_ITEM_SET	Sets data for dynpro 200
ZMB_MIGO_ITEM_GET	Returns data from dynpro 200
ZMB_MIGO_POST	Updates header and item data

Table 5.6 Function Modules for Data Exchange

```

FUNCTION ZMB_MIGO_SETSTATUS.
*-----*
***"Local interface:
**  IMPORTING
**    REFERENCE(I_OUTPUTONLY) TYPE  C

```

```

*“-----
* Set input status
  gv_outputonly = i_outputonly.

ENDFUNCTION.

```

Listing 5.3 Coding on ZMB_MIGO_SETSTATUS

2. Create the modules ZMB_MIGO_HEAD_SET and ZMB_MIGO_HEAD_GET. You need a structure with Table ZMB_MIGOHEAD as a data type to use it as an import or export parameter. Simply copy the data in the respective direction between the parameters and the work structure previously defined with TABLES (see Listing 5.4 and Listing 5.5).

```

FUNCTION ZMB_MIGO_HEAD_SET.
*“-----
*“*“Local interface:
*“  IMPORTING
*“    REFERENCE(I_HEAD) TYPE  ZMB_MIGOHEAD
*“-----
* Prepare data for dynpro
  zmb_migohead = i_head.

ENDFUNCTION.

```

Listing 5.4 Coding on ZMB_MIGO_HEAD_SET

```

FUNCTION zmb_migo_head_get.
*“-----
*“*“Local interface:
*“  EXPORTING
*“    REFERENCE(E_HEAD) TYPE  ZMB_MIGOHEAD
*“-----
* Return data from dynpro
  e_head = zmb_migohead.

ENDFUNCTION.

```

Listing 5.5 Coding on ZMB_MIGO_HEAD_GET

3. For the function modules ZMB_MIGO_ITEM_SET and ZMB_MIGO_ITEM_GET, proceed exactly as before. Only use ZMB_MIGOITEM as a reference (see Listing 5.6 and Listing 5.7).

```

FUNCTION zmb_migo_item_set.
*-----
***Local interface:
*  IMPORTING
*    REFERENCE(I_ITEM) TYPE  ZMB_MIGOITEM
*-----
* Prepare data for dynpro
  zmb_migoitem = i_item.

ENDFUNCTION.

```

Listing 5.6 Coding on ZMB_MIGO_ITEM_SET

```

FUNCTION ZMB_MIGO_ITEM_GET.
*-----
***Local interface:
*  EXPORTING
*    REFERENCE(E_ITEM) TYPE  ZMB_MIGOITEM
*-----
* Return data from dynpro
  e_item = zmb_migoitem.

ENDFUNCTION.

```

Listing 5.7 Coding on ZMB_MIGO_ITEM_GET

4. Create the function module ZMB_MIGO_POST. The module must be marked in the PROCESSING TYPE section as an UPDATE MODULE (START IMMED.) (see Figure 5.3).

Create an import parameter (suggestion I_HEAD) as a structure for Table ZMB_MIGOHEAD. Activate the PASS VALUE option for this parameter because no reference parameters are allowed in update modules. Then define a table parameter (suggestion T_ITEMS) with reference to Table ZMB_MIGOITEM.

5. With regard to update modules, you should always integrate several logical tests to prevent erroneous updates as much as possible. You need to check whether all key fields of the header and item data are filled. Otherwise, an exception will be triggered. To do this, create the DATA_ERROR exception in the EXCEPTIONS tabstrip.

If the writing of the data fails, however, an exception will also be triggered. You create this exception as INSERT_ERROR. Triggering an exception results in an

update termination, and if there's an error, the complete material document is not posted.

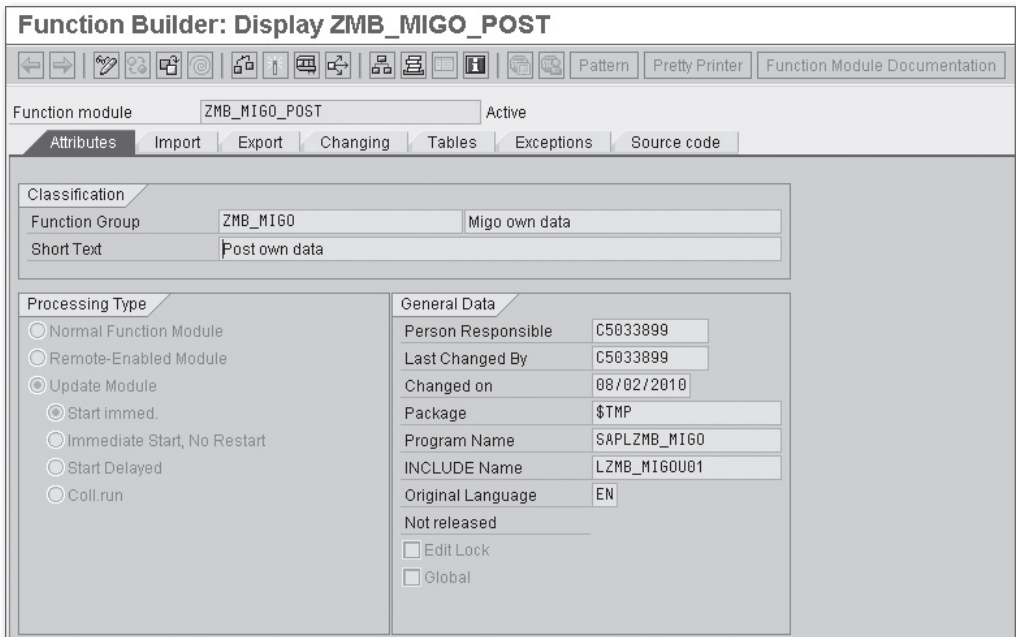


Figure 5.3 Properties of the Update Module

6. Switch to the SOURCE TEXT, and begin with the programming. Check whether the key fields of the header data are filled (parameter I_HEAD, fields MBLNR and MJAHR). Proceed exactly as before in a loop via the item data (Table T_ITEMS). Check also the line number (field ZEILE). If an error occurs, trigger the exception DATA_ERROR by using the command RAISE.
7. If no error occurs, update the data using the command INSERT. If an error occurs here (SY-SUBRC <> 0), trigger the exception INSERT_ERROR (see Listing 5.8).

```
FUNCTION zmb_migo_post.
*-----
*""Update module:
*
*""Local interface:
*  IMPORTING
*    VALUE(I_HEAD) TYPE  ZMB_MIGOHEAD
*  TABLES
```

```

*''      T_ITEMS STRUCTURE  ZMB_MIGOITEM
*''  EXCEPTIONS
*''      INSERT_ERROR
*''      DATA_ERROR
*''-----

* Check transferred data
  IF i_head-mblnr IS INITIAL OR
     i_head-mjahr IS INITIAL.
     RAISE data_error.
  ENDIF.

  LOOP AT t_items.
    IF t_items-mblnr IS INITIAL OR
       t_items-mjahr IS INITIAL OR
       t_items-zeile IS INITIAL.
      RAISE data_error.
    ENDIF.
  ENDLOOP.

* Write header data
  INSERT zmb_migohead FROM i_head.
  IF sy-subrc <> 0.
    RAISE insert_error.
  ENDIF.

* Write item data
  INSERT zmb_migoitem FROM TABLE t_items.
  IF sy-subrc <> 0.
    RAISE insert_error.
  ENDIF.
ENDFUNCTION.

```

Listing 5.8 Coding on ZMB_MIGO_POST

8. Save all changes, and check whether all components of the function group have been activated.

5.1.4 Preparation and Status Management in MB_MIGO_BADI

From now on, you can completely focus on the BAdI. All preparations have been made. As described earlier, all necessary data are kept in the attributes of the class. These are therefore created first.

1. Switch to Transaction SE19, and create a new implementation for the BAdI MB_MIGO_BADI. Double-click the name of the implementing class ZCL_IM_MB_MIGO_BADI, and then switch to the ATTRIBUTES tabstrip.
2. Create the attributes according to Table 5.7. Consider the following notes:
 - ▶ GC_CLASS
Each implementation of the BAdI MB_MIGO_BADI must clearly be identified toward Transaction MIGO. This happens via a constant of the type MIGO_CLASS_ID, which you must define in the attributes because you use these in several areas. In the example given, the constant has been specified with the initial value 'MIGO_OWN'. Note that the single quotation marks are necessary here.
 - ▶ GV_LINEID
In this line number, always note the item transferred last in the dynpro.
 - ▶ GV_ACTION
This attribute contains the currently chosen action (see Table 5.8).
 - ▶ GV_REFDOC
This attribute contains the currently chosen reference document type (see Table 5.9).
 - ▶ GS_HEADER
This structure contains the current header data on your tabstrip.
 - ▶ GT_ITEM
This internal table contains all items on the current document. You can only define internal tables in the attributes when you refer to a table type defined in the ABAP Dictionary or you create a local type. In the following example, a local type (TT_MIGOITEM) has been used that is still currently unknown.

Attribute	Type	Visibility	Reference Type
GC_CLASS	Constant	Private	MIGO_CLASS_ID
GV_LINEID	Instance Attribute	Private	GOITEM-GLOBAL_COUNTER
GV_ACTION	Instance Attribute	Private	GOACTION
GV_REFDOC	Instance Attribute	Private	REFDOC
GS_HEADER	Instance Attribute	Private	ZMB_MIGOHEAD
GT_ITEM	Instance Attribute	Private	TT_MIGOITEM

Table 5.7 Attributes of the Implementing Class

Action/Operation	Description
A01	Goods receipt
A02	Return delivery
A03	Cancellation
A04	Display
A05	Release Goods receipt (GR) blocked stock
A06	Subsequent delivery
A07	Goods issue
A08	Transfer posting
A09	Remove from storage
A10	Place in storage
A11	Subsequent adjustment

Table 5.8 List of Actions in Transaction MIGO

Reference Document	Description
R01	Purchase order
R02	Material document
R03	Delivery note
R04	Inbound delivery
R05	Outbound delivery
R06	Transport
R07	Transport ID code
R08	Order
R09	Reservation
R10	Other

Table 5.9 List of the Possible Reference Documents in Transaction MIGO

- Switch to the TYPES tabstrip to create the data type TT_MIGOITEM. To define a table, you need the direct type input (see Figure 5.4).

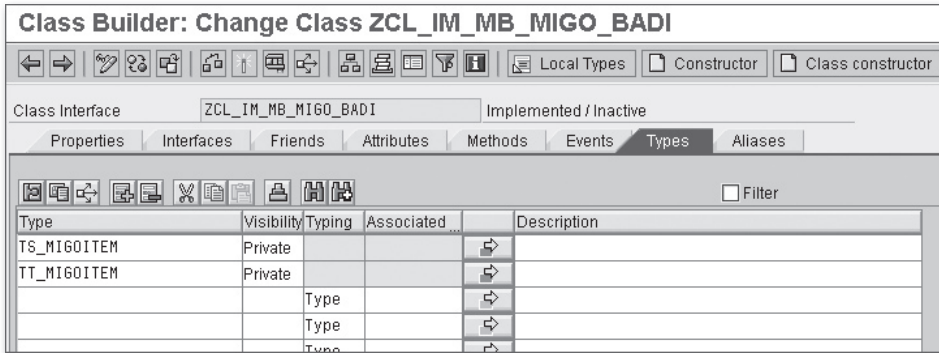


Figure 5.4 Enhanced Type Definition in Classes

4. Enter the name of the data type: TT_MIGOITEM.
5. Choose the PRIVATE entry in the VISIBILITY column because this type is only used within the class.
6. Click the yellow arrow icon on the right next to the reference type (direct type input). You're now in the ABAP Editor and can define the type as required.
7. Besides the actual data from table ZMB_MIGOITEM, the internal table requires another field LINE_ID that displays the internal number during the entry. Create a type that is made up of the field LINE_ID (type MB_LINE_ID) and ZMB_MIGOITEM. Name this type TS_MIGOITEM.
8. Define the type TT_MIGOITEM as an internal table for TS_MIGOITEM. The complete section should appear as follows:

```
PRIVATE SECTION.
  TYPES BEGIN OF ts_migoitem.
    TYPES line_id TYPE mb_line_id.
    INCLUDE TYPE zmb_migoitem.
  TYPES END OF ts_migoitem.

  TYPES tt_migoitem TYPE TABLE OF ts_migoitem.
```

9. Save and activate your changes.

The first methods can now be programmed. Start with the method INIT. As described earlier, this method is used to announce the implementation in Transaction MIGO. A maximum of five implementations may be active at the same time.

You can now prepare the methods `RESET` and `MODE_SET` (refer to Table 5.1). You'll focus on the method `STATUS_AND_HEADER` later.

1. Switch to the `METHODS` tabstrip of the class `ZCL_IM_MB_MIGO_BADI`, and then navigate to method `INIT`. Here you must only transfer defined constants `GC_CLASS` to the internal Table `CT_INIT` in the attributes. This table may already contain other implementations. Therefore, don't overwrite the table; instead, attach your constant via `APPEND`. Because Table `CT_INIT` only consists of one field, you can attach the constant directly. You don't need a local structure (see Listing 5.9).

```
METHOD if_ex_mb_migo_badi~init.
* Register implementation
  APPEND gc_class TO ct_init.
ENDMETHOD.
```

Listing 5.9 Coding on the Method `INIT`

2. Edit the method `RESET`. This method is called when a new document is entered or loaded. Initialize the instance attributes of your class (see Listing 5.10).

```
METHOD if_ex_mb_migo_badi~reset.
* Initialize instance attributes
  CLEAR:   gv_lineid,
           gv_action,
           gv_refdoc,
           gs_header.
  REFRESH gt_item.
ENDMETHOD.
```

Listing 5.10 Coding on the Method `RESET`

3. Switch to the method `MODE_SET`. In this method, you obtain the chosen action (parameter `I_ACTION`; refer to Table 5.8) and the chosen reference document type (parameter `I_REFDOC`; refer to Table 5.9).

Your custom fields are only integrated when this allows the current action. You've already prepared the function module `ZMB_MIGO_SETSTATUS`, which you now call. Set the parameter `I_OUTPUTONLY` when the action `A04` (Display) or `A03` (Cancellation) is selected. Also note the current action in the attribute `GV_ACTION` and the reference document type in the attribute `GV_REFDOC` to be used later (see Listing 5.11).

```

METHOD if_ex_mb_migo_badi~mode_set.
* Local data declarations
  DATA lv_outputonly TYPE c.

* No input for action A04 or A03
  IF    i_action = 'A04' " Anzeige
    OR i_action = 'A03'. "Storno
    lv_outputonly = 'X'.
  ENDIF.

* Set current status
  CALL FUNCTION 'ZMB_MIGO_SETSTATUS'
    EXPORTING
      i_outputonly = lv_outputonly.

* Hold action
  gv_action = i_action.
* Hold reference document type
  gv_refdoc = i_refdoc.

ENDMETHOD.

```

Listing 5.11 Coding on the Method `MODE_SET`

5.1.5 Activation of Custom Header Data

Now let's move on to the relatively simple header data. At this point, your dynpro is displayed as a tabstrip, and data is exchanged between the BAdI and your dynpro. The reading of custom data is already implemented here when a posted document is open. You post the header data later, together with the item data.

1. Switch to the method `PBO_HEADER`. Start with a mandatory check: For technical reasons, the methods from all implementations are always called in an undefined sequence when one BAdI method is called to a method with several active implementations. At the same time, the method `PBO_HEADER` is called once for each of the five possible tabstrips. All implemented methods are therefore started each time. For this reason, you're informed of the class ID in the input parameter `I_CLASS_ID` (you've registered this in the method `INIT`; see Figure 5.5) that is associated with this call. You must make sure that you check whether the content of `I_CLASS_ID` corresponds to the class ID used in this implementation from constant `GC_CLASS`. Otherwise, unforeseeable side effects may arise.

- The dynpro is registered next by transferring `E_CPROG` to the parameters and `E_DYNNR` to the program name and the dynpro number. In the parameter `E_HEADING`, specify the name of the tabstrip. In the example given, the dynpro 0100 is used in the program `SAPLZMB_MIGO`.

As an example, the tabstrip is only displayed when the operation refers to a purchase order document. You can check via the attribute `GV_REFDOC` whether you have filled this in method `MODE_SET`. The reference key for purchase orders is `R01` (refer to Table 5.9).

Class Builder: Class ZCL_IM_MB_MIGO_BADI Change

Ty.	Parameter	Type spec.	Description
▷	I_CLASS_ID	TYPE MIGO_CLASS_ID	Klassen-ID für MIGO-Komponenten (externer Detail-Screen)
▷	E_CPROG	TYPE SYCPROG	ABAP-Programm, Aufrufer in externen Prozeduren
▷	E_DYNNR	TYPE SYDYNNR	ABAP-Programm, Nummer des aktuellen Dynpros
▷	E_HEADING	TYPE MIGO_BADI_HEADING	Tabstrip Überschrift

Method: IF_EX_MB_MIGO_BADI~PBO_HEADER Inactive

```

1  METHOD if_ex_mb_migo_badi~pbo_header.
2  * Bezieht sich der Aufruf auf diese Implementierung?
3  CHECK i_class_id = gc_class.
4  * Dynpro registrieren, falls Referenz zu Bestellung
5  IF gv_refdoc = 'R01'.
6     e_cprog = 'SAPLZMB_MIGO'.
7     e_dynnr = '0100'.
8     e_heading = 'Eigene Daten'.
9  *   Aktuellen Stand der Kopfdaten auf Dynpro übertragen
10  CALL FUNCTION 'ZMB_MIGO_HEAD_SET'
11     EXPORTING
12        i_head = gs_header.
13
14  ENDIF.
15  ENDMETHOD.                                     "IF EX MB MIGO BADI~PBO HEADER

```

Figure 5.5 Interface of the Method PBO_HEADER

- Finally, you still need to call the function module `ZMB_MIGO_HEAD_SET` and transfer the current MIGO content of the header data to the dynpro. The current status can always be found in the attribute `GS_HEADER` (see Listing 5.12).

```

METHOD if_ex_mb_migo_badi~pbo_header.
* Does the call refer to this implmentation?
CHECK i_class_id = gc_class.

* Register dynpro if reference to purchase order
IF gv_refdoc = 'R01'.
    e_cprog = 'SAPLZMB_MIGO'.

```



```

e_dynnr   = '0100'.
e_heading = 'Eigene Daten'.

* Transfer current status of the header data to dynpro
CALL FUNCTION 'ZMB_MIGO_HEAD_SET'
  EXPORTING
    i_head = gs_header.

ENDIF.
ENDMETHOD.

```

Listing 5.12 Coding on the Method PBO_HEADER

4. Now process the method PAI_HEADER. You only need to write the data from the dynpro back to the attribute GS_HEADER at this point. You do this by calling the function module ZMB_MIGO_HEAD_GET (see Listing 5.13).

```

METHOD if_ex_mb_migo_badi~pai_header.
* Retrieve data from the dynpro
CALL FUNCTION 'ZMB_MIGO_HEAD_GET'
  IMPORTING
    e_head = gs_header.

ENDMETHOD.

```

Listing 5.13 Coding on the Method PAI_HEADER

5. You must also read the custom data from the database if required, and transfer it to the attribute GS_HEADER. You carry this out in method STATUS_AND_HEADER. Rereading the data is only necessary when an already-posted document is displayed or canceled. Therefore, you can check the action (attribute GV_ACTION) again here.

You read the data from the table ZMB_MIGOHEAD by using key fields MBLNR (material document number) and MJAHR (posting year) that are available in the IS_GOHEAD parameter (see Listing 5.14).

```

METHOD if_ex_mb_migo_badi~status_and_header.
* If an already posted document is to be displayed
* or canceled, read custom fields
* also from database.

IF   gv_action = 'A04' " Anzeige
  OR gv_action = 'A03'. " Storno

```

```

*   Read data from database
SELECT SINGLE *
      FROM ZMB_MIGOHEAD
      INTO gs_header
      WHERE mblnr = is_gohead-mblnr
      AND   mjahr = is_gohead-mjahr.
ENDIF.
ENDMETHOD.

```

Listing 5-14 Coding on the Method STATUS_AND_HEADER

Note Data

You can also set the parameter `E_HOLD_DATA_DISABLE` as an option in the method `STATUS_AND_HEADER` to prohibit the function `HOLD`. If you continue to allow the noting of material documents, you should program the methods `HOLD_DATA_SAVE`, `HOLD_DATA_LOAD`, and `HOLD_DATA_DELETE` so that your custom data can also be noted. You can obtain further information on this in Section 5.2.1, Noting Custom Data.

5.1.6 Activation of Custom Item Data

The actual activation of the item tabstrip works in a similar way to that of the header tabstrip. The communication with the dynpro and the management of the item data is slightly more time-consuming. Ultimately, there are several items in a document that must always be correctly assigned.

1. Start with the method `PBO_DETAIL` to register the item tabstrip and prepare the data. It's also important to check the class ID just as you did before with the header data.
2. In the parameter `I_LINE_ID`, you can find the line number that is being processed. All subsequent actions are only carried out when the parameter has a content unequal to zero.

You need to also note this line number in the attribute `GV_LINEID` so that you can check later which line is actually being displayed in the dynpro.

3. There are also three parameters here, `E_CPROG`, `E_DYNNR`, and `E_HEADING`, to specify the program name, the dynpro number, and the caption of the tabstrip. The item tabstrip always appears in the example regardless of the reference document type.

4. You must then read the current line according to parameter `I_LINE_ID` from your internal table defined in the attributes `GT_ITEM` and transfer it to your function group by calling the function module `ZMB_MIGO_ITEM_SET`. Make sure that you copy the data via `MOVECORRESPONDING` into a structure suitable for the function module because your table with additional line ID has been defined in the attributes (see Listing 5.15).

```

METHOD if_ex_mb_migo_badi~pbo_detail.
* Local declarations
  DATA ls_item TYPE ts_migoitem.
  DATA ls_dynpro TYPE zmb_migoitem.

* Does the call refer to this implementation?
  CHECK i_class_id = gc_class.

* Has a line been set?
  CHECK i_line_id <> 0.

* Hold line
  gv_lineid = i_line_id.

* Register dynpro
  e_cprog   = 'SAPLZMB_MIGO'.
  e_dynnr   = '0200'.
  e_heading = 'Eigene Daten'.

* Read line from internal table
  READ TABLE gt_item
    INTO ls_item
    WITH KEY line_id = i_line_id.

* Copy necessary fields
  MOVE-CORRESPONDING ls_item TO ls_dynpro.
* Prepare line in dynpro
  CALL FUNCTION ZMB_MIGO_ITEM_SET
    EXPORTING
      i_item = ls_dynpro.

ENDMETHOD.

```

Listing 5.15 Coding on the Method `PBO_DETAIL`

5. Switch to the method `PAI_DETAIL` in which you retrieve your data from the dynpro, and check whether the content has changed. If it has, set the flag `E_FORCE_CHANGE` through which the method `LINE_MODIFY` is triggered. Then carry out the actual handling of the data. The advantage to this is that the complete item management can be found in one central position (see Listing 5.16).

```
METHOD if_ex_mb_migo_badi~pai_detail.
* Local declarations
  DATA: ls_olddata TYPE zmb_migoitem,
         ls_newdata TYPE zmb_migoitem,
         ls_item TYPE ts_migoitem.

* Has a line been set?
  CHECK i_line_id <> 0.

* Retrieve old status from internal table
  READ TABLE gt_item
    INTO ls_item
    WITH KEY line_id = i_line_id.
  MOVE-CORRESPONDING ls_item TO ls_olddata.

* Retrieve new status from dynpro
  CALL FUNCTION 'ZMB_MIGO_ITEM_GET'
    IMPORTING
      e_item = ls_newdata.

* Check whether the data has been changed
  IF ls_olddata <> ls_newdata.
*   Trigger execution of LINE_MODIFY
  e_force_change = 'X'.
ENDIF.

ENDMETHOD.
```

Listing 5.16 Coding on the Method `PAI_DETAIL`

6. Now it's time for the method `LINE_MODIFY`. As mentioned earlier, in this method, you manage not only possible changes to existing lines but also the insertion of new document lines in one central position.

As a parameter, you obtain the number of the current line in `I_LINE_ID`. With this line number, you must first of all check which case is actually present. Read the table `GT_ITEM` using this key. If the line already exists, possible changes must be transferred. If this line is not yet contained in `GT_ITEM`, you need to initialize the line and add the table. If the document is currently being read, at this point also read your custom data from the database.

Take a look at the case of a change. To ensure that the line being handled in the BAdI is also the line that is currently in the dynpro, compare the parameter `I_LINE_ID` with the previously noted line in the attribute `GV_LINEID`. You carry out the following steps only when the comparison is successful:

1. Retrieve the data from the dynpro by calling the function module `ZMB_MIGO_ITEM_GET` again. Copy the data received into a work structure that fits Table `GT_ITEM` (data type `TS_MIGOITEM`), and enhance the line at the current `LINE_ID`. Then write the changes back to Table `GT_ITEM`.
2. If the line doesn't yet exist, you must first check whether it's a line that has already been posted. In this case, the fields `CS_GOITEM-MBLNR` (material document number), `CS_GOITEM-MJAHR` (posting year), and `CS_GOITEM-ZEILE` (document item) are filled. Read your data from Table `ZMB_MIGOITEM` using these fields as a key, and copy these again into a suitable work structure for Table `GT_ITEM`.

If the line doesn't yet exist in the database, leave the work structure empty, apart from the line `LINE_ID`, which you still need to enhance in both cases. Then write the new line in Table `GT_ITEM` (see Listing 5.17).

```
METHOD if_ex_mb_migo_badi~line_modify.
* Local declarations
  DATA lv_tabix TYPE sy-tabix.
  DATA ls_item TYPE ts_migoitem.
  DATA ls_newdata TYPE zmb_migoitem.

* Does the line already exist?
  READ TABLE gt_item INTO ls_item
    WITH KEY line_id = i_line_id.
  lv_tabix = sy-tabix.

  IF sy-subrc = 0.
*   Line exists already, does change
*   correspond to line in the BAdI of the dynpro line?
    CHECK i_line_id = gv_lineid.
```

```

* Retrieve data from dynpro
CALL FUNCTION 'ZMB_MIGO_ITEM_GET'
  IMPORTING
    e_item = ls_newdata.

* Format changes
MOVE-CORRESPONDING ls_newdata TO ls_item.
ls_item-line_id = i_line_id.

* Write back in table gt_item
MODIFY gt_item FROM ls_item INDEX lv_tabix.
ELSE.
* Line doesn't yet exist, insert
IF cs_goitem-mblnr IS NOT INITIAL AND
  cs_goitem-mjahr IS NOT INITIAL AND
  cs_goitem-zeile IS NOT INITIAL.
* Line refers to an existing
* material document, retrieve data from database
SELECT SINGLE * FROM zmb_migoitem
  INTO ls_newdata
  WHERE mblnr = cs_goitem-mblnr AND
        mjahr = cs_goitem-mjahr AND
        zeile = cs_goitem-zeile.
  IF sy-subrc = 0.
* Copy data
  MOVE-CORRESPONDING ls_newdata TO ls_item.
  ENDIF.
ENDIF.
* Build line with line ID and copy
* to internal table
ls_item-line_id = i_line_id.
APPEND ls_item TO gt_item.
ENDIF.

ENDMETHOD.

```

Listing 5.17 Coding on the Method LINE_MODIFY

- Now you need to program the method LINE_DELETE. The method is called when a document item is deleted. In this case, you must also delete the associated custom data from Table GT_ITEM. Parameter I_LINE_ID is provided for you to identify the suitable line (see Listing 5.18).

```

METHOD if_ex_mb_migo_badi~line_delete.
* Delete line
  DELETE gt_item WHERE line_id = i_line_id.
ENDMETHOD.

```

Listing 5.18 Coding on the Method LINE_DELETE

5.1.7 Updating the Data

Now that you've fully programmed the internal handling of the data, you must ensure that your data is also posted when the material document is updated. To accomplish this, the method `POST_DOCUMENT` runs while the data is being posted.

1. Switch to the method `POST_DOCUMENT`. Before you can call your update modules, you must prepare the data. The structure `GS_HEADER`, which has been defined in the attributes, must still be enhanced in the document number. You obtain this structure via the parameter `IS_MKPF`. You can copy the values via a simple `MOVE-CORRESPONDING`.
2. Regarding item data, so far you have managed the data via the `LINE_ID`, that is, the internal line number. As you know, only the items for which the user has selected the OK field in Transaction MIGO are updated while being posted. Therefore, the table to be updated, `MSEG`, which is transferred to you as the parameter `IT_MSEG`, can have a different line numbering. The item number here is specified via the field `ZEILE`, and you must now convert your data to this line number. However, because Table `IT_MSEG` also contains the original `LINE_ID` besides the line number, this isn't a problem.

Simply process all your custom item data from the internal Table `GT_ITEM` in a loop, and read the associated line from Table `IT_MSEG` using the field `LINE_ID`. You can then fill a structure for Table `ZMB_MIGOITEM` from both data structures. From the structure for Table `IT_MSEG`, copy the key fields, and from the structure on `GT_ITEM`, copy the custom fields. Add the result of a local internal table that also uses the type `ZMB_MIGOITEM`.

3. You now only need to call the function module `ZMB_MIGO_POST` and transfer the formatted header data and item data. Don't forget the addition `IN UPDATE TASK`, so that the function module in the updating is called (see Listing 5.19).

```

METHOD if_ex_mb_migo_badi~post_document.
* Local data declarations
  DATA: ls_item TYPE ts_migoitem,

```

```

ls_mseg TYPE mseg,
ls_migoitem TYPE zmb_migoitem,
lt_migoitem TYPE TABLE OF zmb_migoitem.

* Prepare header data
* The key fields (material document number, posting year)
* are copied from the standard data (IS_MKPF)
MOVE-CORRESPONDING is_mkpf TO gs_header.

* Prepare item data
LOOP AT gt_item INTO ls_item.
* Determine corresponding line in IT_MSEG.
* Conversion between LINE_ID and ZEILE
* LINE_ID: internal line ID during entry
* ZEILE: Line number in table MSEG
READ TABLE it_mseg
  INTO ls_mseg
  WITH KEY line_id = ls_item-line_id.
IF sy-subrc = 0.
  MOVE-CORRESPONDING ls_item TO ls_migoitem.
  MOVE-CORRESPONDING ls_mseg TO ls_migoitem.
  APPEND ls_migoitem TO lt_migoitem.
ENDIF.
ENDLOOP.

* Updating of the data
CALL FUNCTION 'ZMB_MIGO_POST'
  IN UPDATE TASK
  EXPORTING
    i_head = gs_header
  TABLES
    t_items = lt_migoitem.

ENDMETHOD.

```

Listing 5.19 Coding on the Method POST_DOCUMENT

5.2 Other Functions of the BAdI MB_MIGO_BADI

After you've implemented custom fields via the BAdI MB_MIGO_BADI in the previous chapter, you already know all the fundamentals on this BAdI. Using other

methods, you can still implement additional functions, particularly to further extend the functionality behind your custom fields.

5.2.1 Noting Custom Data

If you haven't prohibited the function `Note` in the method `STATUS_AND_HEADER` (see Section 5.1.5, Activation of Custom Header Data), you should ensure that your custom fields are also noted. Because a noted document isn't actually posted yet, it still doesn't have any material document number under which it can be stored. The data are instead clearly identified by a 22-digit GUID (Global Unique Identifier). Standard data are stored under this key in Table `MMIM_PRED`.

Three methods are available for storing your custom data: `HOLD_DATA_SAVE`, `HOLD_DATA_LOAD`, and `HOLD_DATA_DELETE`. The GUID, which has also been used for the standard fields, will transfer each of these methods as input parameters. Whether you now store your data, that is, the header data that are in the structure `GS_HEADER` and the item data in internal Table `GT_ITEM`, is up to you.

You have several options for storing the data:

- ▶ You can copy Tables `ZMB_MIGOHEAD` and `ZMB_MIGOITEM` and provide them with a GUID field as the key field. This makes saving somewhat more time-consuming, however, and when you add another custom field later to the original tables, you must also implement such enhancements in the `Hold` function.
- ▶ One alternative allows the comfortable, dynamic storage of both objects without any additional customization effort. You have the option in ABAP to serialize complex variables such as structures or internal tables; that is, to convert them into a character string. This string must have a very special data type. Use the type `XSTRING` in ABAP and type `RAWSTRING` in the ABAP Dictionary. You then convert the data via the command `EXPORT ... TO DATA BUFFER`. You can store the result of the conversion in a field of a database line, regardless of whether or not this object has displayed an internal table with many lines beforehand. If the data is subsequently retrieved, using the command `IMPORT ... FROM DATA BUFFER` performs a reconversion in the actual object.

Based on the example of custom fields in Section 5.1, Custom Fields in Transaction MIGO, the following example shows you how simply you can temporarily store your data:

1. Your header and item data must be stored in a custom table. The data is saved under a GUID, which you obtain in the methods. You also need a field of the type `RAWSTRING` for the header data and the item data.

Navigate to the ABAP Dictionary (Transaction SE11), and create a new table. Name this table `ZMB_MIGOHOLD`, for example.

2. Maintain the fields according to Table 5.10. The data element `ZMB_RAW` doesn't yet exist. You can create this by double-clicking. Don't use any domains to generate the data element, but choose the `INTEGRATED TYPE` option in the `DATA TYPE` tabstrip, and enter `RAWSTRING` here. Activate your data element, and then navigate back to the table.

Field Name	Key Field	Data Element
MANDT	Yes	MANDT
GUID	Yes	GUID
HHEAD		ZMB_RAW (Definition: Integrated type RAWSTRING)
HITEMS		ZMB_RAW

Table 5.10 Structure of the Table `ZMB_MIGOHOLD`

3. Maintain the technical settings of the table, and then activate the table.
4. Switch to your BAdI implementation, and navigate to the method `HOLD_DATA_SAVE`. Create a local structure for Table `ZMB_MIGOHOLD`, and fill the `GUID` field from the input parameter `I_GUID`.
5. Fill the fields `HHEAD` and `HITEMS` from your attributes `S_HEADER` and `GT_ITEM` via the command `EXPORT`, which has the following structure:

```
EXPORT <id> FROM <variable> TO DATA BUFFER
  <ziel> COMPRESSION ON.
```

`<id>` is any ID, `<variable>` is the source object that is to be converted, and `<target>` is the target variable of the type `RAWSTRING`. The addition, `COMPRESSION ON`, is optional and compresses the data. This saves space in the database. The subsequent `IMPORT` automatically recognizes whether the data has been compressed, and extracts the data accordingly.

6. Finally, save your local structure via `INSERT` in the database (see Listing 5.20).

```

METHOD if_ex_mb_migo_badi~hold_data_save.
* Local work structure for ZMB_MIGO_HOLD
  DATA ls_migohold TYPE zmb_migohold.

* Copy GUID
  ls_migohold-guid = i_guid.

* Convert header data in RAWSTRING/XSTRING
  EXPORT header FROM gs_header TO DATA BUFFER
    ls_migohold-hhead COMPRESSION ON.

* Convert item data in RAWSTRING/XSTRING
  EXPORT item FROM gt_item TO DATA BUFFER
    ls_migohold-hitems COMPRESSION ON.

* Store data in database
  INSERT into zmb_migohold values ls_migohold.

ENDMETHOD.

```

Listing 5.20 Coding on the Method HOLD_DATA_SAVE

Now focus on the method HOLD_DATA_LOAD, which is called when a held document is loaded back.

1. Here you must load back the data from Table ZMB_MIGO_HOLD using the retransferred GUID, and fill the attributes GS_HEADER and GT_ITEM from this.
2. Create a new local work structure for Table ZB_MIGO_HOLD, and read the data via SELECT using the key in parameter I_GUID from your table.
3. You perform the reconversion from the fields HHEAD and HITEMS via the IMPORT function with the following structure:

```
IMPORT <id> TO <variable> FROM DATA BUFFER <quelle>.
```

<id> stands for the ID that you've used with regard to the EXPORT; <variable> is the target variable in which the result is to be written, that is, GS_HEADER or GT_ITEM; and <source> is the RAWSTRING field from your work structure for ZMB_MIGO_HOLD. You can find an example on this method in Listing 5.21.

```

METHOD if_ex_mb_migo_badi~hold_data_load.
* Local work structure for ZMB_MIGO_HOLD
  DATA ls_migohold TYPE zmb_migohold.

* Read held data using I_GUID

```

```

SELECT SINGLE * FROM zmb_migohold
      INTO ls_migohold
      WHERE guid = i_guid.

IF sy-subrc = 0.
*   Convert header data in original object
  IMPORT header TO gs_header FROM DATA BUFFER
      ls_migohold-hhead.

*   Convert item data in original object
  IMPORT item TO gt_item FROM DATA BUFFER
      ls_migohold-hitems.
ENDIF.
ENDMETHOD.

```

Listing 5.21 Coding on the Method HOLD_DATA_LOAD

4. If you delete the marked data, you must also remove your noted data. To do this, the method HOLD_DATA_DELETE is used. You also get the GUID as an input parameter again. Simply delete the associated record in Table ZMB_MIGOHOLD (see Listing 5.22).

```

METHOD if_ex_mb_migo_badi~hold_data_delete.
*   Delete held data
  DELETE FROM zmb_migohold WHERE guid = i_guid.
ENDMETHOD.

```

Listing 5.22 Coding on the Method HOLD_DATA_DELETE

Changes to the Data Structure

As you've seen, you can easily convert ABAP structures in a RAWSTRING using the commands EXPORT and IMPORT, and copy these back to a structure. However, what happens when you change such a packed structure later on?

For example, if you add another field to Table ZMB_MIGOHEAD for the header data, data could still exist in the HHEAD field of the ZMB_MIGOHOLD table, which does not contain this new field. However, this generally doesn't constitute any problem. The command IMPORT copies the suitable part back to the work structure while reading the noted data, and the new field simply remains empty.

Vice versa is a bit more difficult. If you delete the second field again at a later stage and still contain noted data in the RAWSTRING HHEAD of this, you obtain a short dump when you IMPORT. You can avoid this dump by using the ACCEPTING TRUNCATION addition:

```
IMPORT <id> TO <variable> FROM DATA BUFFER <source> ACCEPTING TRUNCATION.
```

For major structure changes, you should delete the content of Table ZMB_MIGO_HOLD as soon as you transport your changes to the production system.

5.2.2 Input Checks in Transaction MIGO

Two more methods are provided for checking your custom data: `CHECK_HEADER` and `CHECK_ITEM`. These methods are only carried out when the user chooses the `CHECK` or `POST` function. Follow the example below to activate the checks.

You need to carry out the checks on the attributes of your class in both cases. To do this, you obtain the input parameter `I_LINE_ID` for the items. If a warning or error message is displayed, this isn't allowed to happen directly via the `MESSAGE` command. Instead, you need to fill a return-table. You might know about this from using the BAPI function modules. The name of this return-table is in both methods: `ET_BAPIRET2`. Your messages are therefore displayed in the standard error log (see Figure 5.6) of Transaction MIGO, and the item number for item-related messages, to which the message refers, is displayed by default.

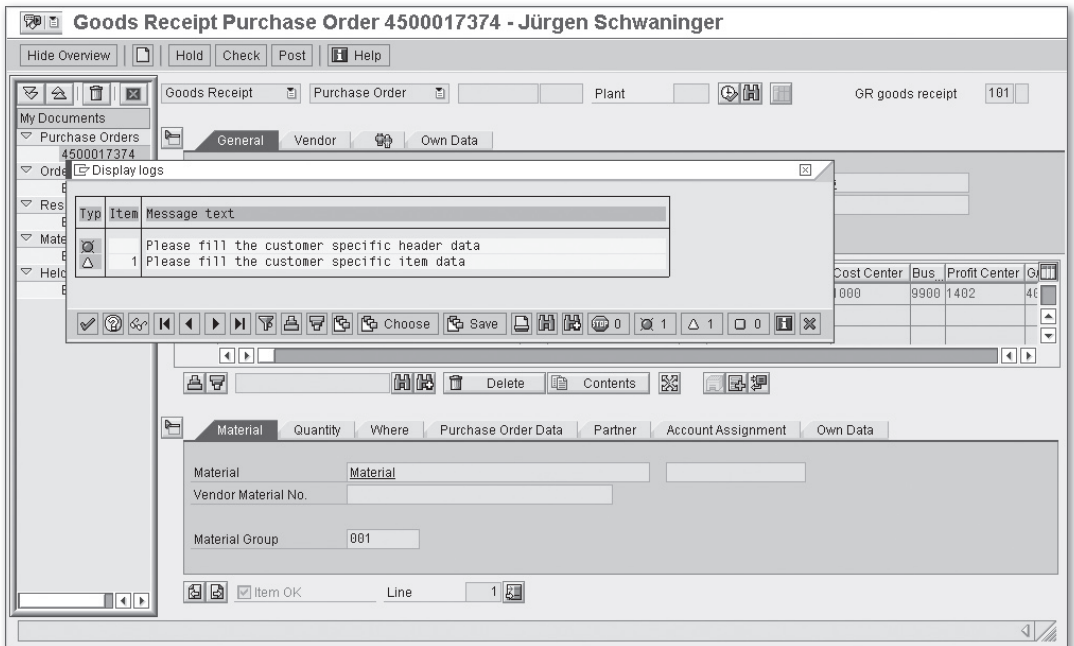


Figure 5.6 Custom Messages in the Error Log

1. Switch to the method `CHECK_HEADER`. For the return of messages, you must first define a work structure for the return-table by using the type `BAPIRET2`. This type is the line type that has been used to define internal Table `ET_BAPIRET2`. This type therefore contains the fields required for a message (see Table 5.11).
2. Now check whether the `ZK_FELD1` field is filled from the attribute `GS_HEADER`. If this isn't the case, an error message appears.
3. To display the message, fill the fields of your work structure according to Table 5.11. Then copy the structure to internal Table `ET_BAPIRET2`, which is defined in the interface of the method (see Listing 5.23).

Field	Description
TYPE	Type of message, for example, 'E' for error messages or 'W' for warning messages
ID	Message class
NUMBER	Message number
MESSAGE_V1 – MESSAGE_V4	Optional message variables 1-4 that can be included

Table 5.11 Fields of the Return-Table

```

METHOD if_ex_mb_migo_badi~check_header.
* Local declaration on the return table
  DATA: ls_bapiret TYPE bapiret2.

* Header field must be filled!
  IF gs_header-zk_feld1 IS INITIAL.
*   Configure error message
    ls_bapiret-type      = 'E'.
    ls_bapiret-id       = 'ZMB'.
    ls_bapiret-number   = '050'.
    APPEND ls_bapiret TO et_bapiret2.
  ENDIF.

ENDMETHOD.

```

Listing 5.23 Coding on the Method `CHECK_HEADER`

4. Switch to the method `CHECK_ITEM`. You basically proceed here as you did with checking the header data. However, you must also define a local work structure

for your items, and read the suitable item from Table GT_ITEM using the input parameter I_LINE_ID.

5. You can then carry out your check again. The user should also fill the field ZP_FELD1. However, only a warning message is displayed if the field is empty (see Listing 5.24).

```

METHOD if_ex_mb_migo_badi~check_item.
* Local declarations
  DATA: ls_item TYPE ts_migoitem,
         ls_bapiret TYPE bapiret2.

* Read item from GT_ITEM
  READ TABLE gt_item INTO ls_item
    WITH KEY line_id = i_line_id.

  IF ls_item-zp_feld1 IS INITIAL.
*   Configure warning message
    ls_bapiret-type      = 'W'.
    ls_bapiret-id        = 'ZMB'.
    ls_bapiret-number    = '051'.
    APPEND ls_bapiret TO et_bapiret2.
  ENDIF.
ENDMETHOD.

```

Listing 5.24 Coding on the Method CHECK_ITEM

5.3 Checking and Prepopulating Standard Fields

Standard fields aren't provided in the check methods of the BAdI MB_MIGO_BADI. However, you can use the BAdI MB_MIGO_ITEM_BADI to also check these fields. Furthermore, in this BAdI, you can simply prepopulate the storage location or the item text. If other fields are prepopulated, you can take another look at the method LINE_MODIFY in the BAdI MB_MIGO_BADI, which provides more options in this context.

5.3.1 Prepopulation of Storage Location and Text

The only method of the BAdI MB_MIGO_ITEM_BADI, ITEM_MODIFY is called when new items are added or the user chooses the CHECK or POST function. When you fill the

export parameter `E_STGE_LOC` (storage location) or `E_ITEM_TEXT` (item text) in the method, these values are set for all new additional items.

5.3.2 Checking the Standard Fields

If the user chooses the `CHECK` or `POST` functions, you can carry out custom checks and prevent posting if applicable. For this, the header data in parameter `IS_GOHEAD` and the item data table `IS_GOITEM` are provided.

If a message is displayed, you also need a work structure for a return-table with the type `BAPIRET2`. Fill the work structure according to Table 5.11 in Section 5.2.2, Input Checks in Transaction `MIGO`, and then append the message to internal Table `ET_RETURN`, which is also defined in the interface. Because the method `ITEM_MODIFY` could be called repeatedly, and Table `ET_RETURN` is already filled by the previous call, you should first delete the content; otherwise, the same message could possibly appear twice in the message log (see Listing 5.25).

```
METHOD if_ex_mb_migo_item_badi~item_modify.
* Local work structure for the return- table
  DATA: ls_bapiret TYPE bapiret2.

* For plant 1000 storage location 0001 is always
* to be suggested.
  IF is_goitem-werks = '1000'.
    e_stge_loc = '0001'.
  ENDIF.

* Reset return table
  REFRESH et_return.
* Header text must be populated
  IF is_gohead-bktxt IS INITIAL.
*   Configure message, warning
    ls_bapiret-type      = 'W'.
    ls_bapiret-id        = 'ZMB'.
    ls_bapiret-number    = '060'.
    APPEND ls_bapiret TO et_return.
  ENDIF.

* Item text must be filled
  IF is_goitem-sgtxt IS INITIAL.
*   Configure message, error
```



```

ls_bapiret-type      = 'E'.
ls_bapiret-id        = 'ZMB'.
ls_bapiret-number    = '061'.
APPEND ls_bapiret TO et_return.
ENDIF.
ENDMETHOD.

```

Listing 5.25 Coding on the Method ITEM_MODIFY

5.4 Check of the Earliest Delivery Date

You have the option in the purchase order to specify a delivery date. However, vendors often don't keep to the delivery date and deliver goods too early. When this is better controlled, you can set the message M7 254 ("Earliest Possible Delivery Date is &") as shown in Figure 5.7) as an error message in Customizing. To do this, choose the MATERIALS MANAGEMENT • INVENTORY MANAGEMENT AND PHYSICAL INVENTORY • DEFINE ATTRIBUTES OF SYSTEM MESSAGES • SETTINGS FOR SYSTEM MESSAGES setting in Transaction SPRO, and define the message M7 254 as type 'E'.

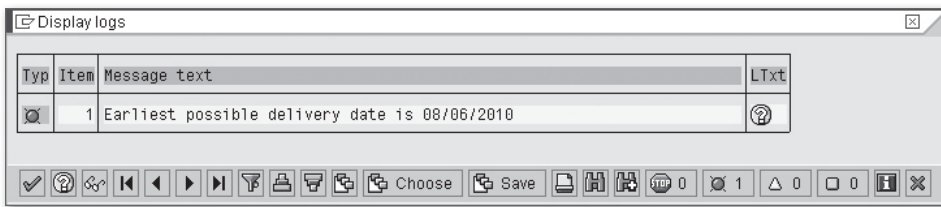


Figure 5.7 Error Message M7 254

As a result, all goods receipts that are supposed to be posted at an earlier date are rejected with the error message. However, you can post the goods receipt in the GR blocked stock (transaction type 103).

In certain circumstances, the settings for such an error message are too strict. With the enhancement MEFLD004 (EXIT_SAPLEINR_004), you can specify the earliest delivery date. Let's assume you have activated the error message. Vendor 1002 is allowed to deliver before the delivery date specified in the purchase order, but not before the purchase order date. The following applies to all other vendors: For all items with material group 00, a delivery of up to seven days prior to the delivery

date is possible. For other goods deliveries, a delivery of up to three days prior to the delivery date is permitted.

1. Create a new project in Transaction CMOD, and include the enhancement MEFLD004.
2. Switch to the user exit EXIT_SAPLEINR_004, and create the include ZXM06U54.
3. In the interface of the exit, the purchase order header (EKKO) and the purchase order item to be checked (EKPO) are transferred. Based on this data, you can overwrite the earliest delivery date (parameter FRLFD). The field FRLFD already contains the delivery date from the purchase order. You can therefore leave the content unchanged, if no special rule applies.

Check whether the vendor (EKKO-LIFNR) has the number 1002 (look out for leading zeros); if yes, the purchase order date (EKKO-BEDAT) will apply as the earliest delivery date.

4. Check whether the purchase order item has the material group 003; if yes, a goods receipt of up to seven days prior to the delivery date is expected to be possible. Because the field FRLFD already contains the delivery date from the purchase order, you can simply deduct seven days.

For all other items, the goods receipt can take place three days earlier. Also carry out the respective calculation (see Listing 5.26). Don't forget to activate the project so that your user exit will run.

```
*&-----*
*&  Include           ZXM06U54
*&-----*
**"*Local interface:
**  IMPORTING
**    VALUE(EKKO) TYPE  EKKO
**    VALUE(EKPO) TYPE  EKPO
**  CHANGING
**    VALUE(FRLFD) LIKE  EBEFU-FRLFD
**-----
* Determine earliest delivery date dynamically

IF ekko-lifnr = '0000001002'.
* Vendor 1002 is allowed to deliver earlier indefinitely,
* however, not before the purchase order date
  frlfd = ekko-bedat.
```

```

ELSE.
* Item for material group 003?
  IF ekpo-matk1 = '003'.
*   Goods receipt can take place seven days earlier
    frlfd = frlfd - 7.
ELSE.
*   Goods receipt can take place three days earlier
    frlfd = frlfd - 3.
ENDIF.
ENDIF.

```

Listing 5.26 Coding on User-Exit EXIT_SAPLEINR_004

5.5 Tolerance Limits for Scheduling Agreements

Using the current date, the default quantity and tolerance check quantity are determined from the schedule lines when goods receipts are posted according to scheduling agreements. If no tolerances are permitted in the scheduling agreement, and the vendor occasionally delivers a few days too early, you receive an error message in the goods receipt (see Figure 5.8).

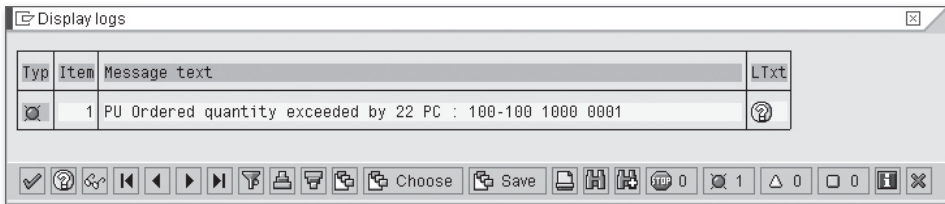


Figure 5.8 Exceeding the Tolerance Levels

If your vendor is allowed to deliver the goods, for example, up to three days earlier, you may use the user exit EXIT_SAPLEINR_001 of the enhancement MEVME001 to determine the default quantity and tolerance levels dynamically.

5.5.1 Overwriting Overdelivery Quantity

Let's first take a look at the scheduling agreement schedule in Figure 5.9. Items 1 and 2 have each a scheduled quantity of 10 pieces. A goods receipt of more than 12 pieces has already been posted. Item 1 is therefore complete; for item 2, there is still an open quantity of 8 pieces.

Maintain Sch. Agmt Schedule : Delivery Schedule for Item 00010										
Agreement		5500000139		Quantity		90		PC		
Material		100-100		Case						
Last Transm.		00:00:00		Next Transmission Number		1				
Cum. Rec. Qty		12		Old Qty		0				
C	Delivery Date	Scheduled Quantity	Time	F	C	StDelDate	Cum. Sch. Qty	GR Qty	Open Quan.	Purchase
D	07/01/2010	10				R07/01/2010	12	10		
D	07/02/2010	10				R07/02/2010	20	2	8	
D	07/05/2010	5				R07/05/2010	25		5	
D	07/08/2010	30				R07/08/2010	55		30	
D	07/11/2010	15				R07/11/2010	70		15	
D	07/15/2010	20				R07/15/2010	90		20	

Figure 5.9 Scheduling Agreement Schedule

On July 6, 2010, another goods receipt of more than 43 pieces is now supposed to be posted. The open quantity consists of the 5 pieces scheduled for delivery on July 5, 2010, and also of the 8 pieces still open from the previous items. A total of 13 pieces are open. The goods receipt is therefore rejected due to an overbooking of 30 pieces (43 pc – 13 pc). The vendor has already added the 30 pieces, which are expected on July 8, 2010 – that is, in the future. Because this is within the 3 days for which you allow an earlier delivery, the goods receipt is expected to go through.

1. To resolve this situation, you must program the aforementioned user exit `EXIT_SAPLEINR_001`. Create a project in Transaction `CMOD`, and add the enhancement `MEVME001`.
2. Switch to the user exit `EXIT_SAPLEINR_001`, and create the include `ZXM06U28`.
3. You get two parameters in the interface: The parameter `POT` contains the item data from the scheduling agreement; the internal table `CETT` contains all schedule lines. Based on this data, you can perform your own calculations and fill the parameters according to Table 5.12. Keep in mind that these parameters in the documentation on the enhancement `MEVME001` are not completely and uniquely defined. The parameters `F3` and `F4` have only been added with SAP Note 737495; the documentation has not been customized, however.

Parameter	Description
F1	WE default quantity
F2	Open quantity on key date
F3	Underdelivery quantity
F4	Overdelivery quantity

Table 5.12 Output Parameter EXIT_SAPLEINR_001

4. Because the overdelivery quantity is customized, you need to determine a new value for parameter F4. The calculation for this works very simply. The parameter contains the complete scheduled quantity, starting from the first item of the scheduling to the last scheduling prior to the key data in total. The system automatically deducts goods receipts already carried out, so you don't need to take this into consideration.

Calculate in a loop via Table CETT the total of all schedule quantities with one vendor smaller than the date "today + 3 days". You write the result of the calculation in the field F4 (see Listing 5.27).

```
*&-----*
*&  Include          ZXM06U28
*&-----*
*""*Local interface:
*""  IMPORTING
*""    VALUE(POT)
*""  TABLES
*""    CETT STRUCTURE  EKET
*""  CHANGING
*""    VALUE(F1)
*""    VALUE(F2)
*""    VALUE(F3) OPTIONAL
*""    VALUE(F4) OPTIONAL
*""-----
* Local declarations
DATA lv_menge TYPE eket-menge.
DATA lv_checkdate TYPE eket-eindt.

* Consider schedules until today + 3 days
lv_checkdate = sy-datlo + 3.

* Total all schedule quantities
```

```

LOOP AT cett WHERE eindt <= lv_checkdate.
  lv_menge = lv_menge + cett-menge.
ENDLOOP.

```

```

* Transfer of the calculated value as maximum limit
f4 = lv_menge.

```

Listing 5.27 Coding for EXIT_SAPLEINR_001

5.5.2 Overwriting Default Quantity

After the goods receipt that includes 43 pieces has been posted, the new situation in the scheduling agreement is shown in Figure 5.10. As you can see, if there is no longer any quantity open until and including July 8, 2010, the next goods receipt is not expected again until July 11, 2010.

Maintain Sch. Agmt Schedule : Delivery Schedule for Item 00010											
Agreement		5500000139		Quantity		90		PC			
Material		100-100		Case							
Last Transm.		00:00:00		Next Transmission Number		1					
Cum. Rec. Qty		55		Old Qty		0					
C	Delivery Date	Scheduled Quantity	Time	F	C	St.DelDate	Cum. Sch. Qty	GR Qty	Open Quantity	Item	Purch
D	07/01/2010	10				R07/01/2010	55	10			
D	07/02/2010	10				R07/02/2010	55	10			
D	07/05/2010	5				R07/05/2010	55	5			
D	07/08/2010	30				R07/08/2010	55	30			
D	07/11/2010	15				R07/11/2010	70		15		
D	07/15/2010	20				R07/15/2010	90		20		

Figure 5.10 Scheduling Agreement Schedule following Goods Receipt

If you now receive a new delivery of more than 15 pieces on July 11, 2010, and these are supposed to be posted as goods receipt, you receive the message "Document contains no selectable item" if the user hasn't activated the PROPOSE ALL ITEMS option. You can react using the user exit: If an open scheduling exists within the next three days, the first open quantity found is suggested. In the example given, this quantity refers to the 15 pieces from July 11, 2010.

1. Navigate again to the user exit EXIT_SAPLEINR_001, and enhance the coding for the current case. The field F2 contains the quantity of all schedulings up to and

including today, minus the goods receipts already carried out. If this value is zero, the line is not suggested. Check the value F2. If no quantity is suggested here, you must calculate a new value.

2. Because the system has already checked all schedulings up to and including the current date, you must check Table CETT again in a loop only from the day "today + 1" until "today + 3". The schedule quantity, as before in the field MENGE, was carried out in the goods receipts in field WEMNG. Calculate the open quantity from this. As soon as you find an open quantity, you can leave the loop.
3. You don't need to assign the result of the calculation to the field F2. In fact, the field F1 again contains the complete schedule quantity from the first day of the scheduling agreement until today. The system deducts the goods receipts already carried out only later from this figure. Simply add the result of your calculation to the field F1. The line with your determined quantity is therefore suggested (see Listing 5.28).

```

...
* Additional determination of the default quantity
* Further data declarations
DATA lv_morgen TYPE eket-eindt.

IF F2 = 0.
* No default quantity available, check whether an open
* schedule exists in the period today + 1 until today + 3
  lv_morgen = sy-datlo + 1.
  clear lv_menge.

  LOOP AT cett WHERE eindt <= lv_checkdate AND
                eindt >= lv_morgen.
    lv_menge = cett-menge - cett-wemng.
    IF lv_menge > 0.
*      Open schedule available, exit loop
      EXIT.
    ENDIF.
  ENDLOOP.

* Increase expected quantity for found value
  F1 = F1 + lv_menge.

ENDIF.

```

Listing 5.28 Further Coding for EXIT_SAPLEINR_001

- Based on the situation from Figure 5.10, if you post a goods receipt on July 10, 2010, an item with a quantity of 15 pieces is suggested to you, which is the value of the schedule of July 11, 2010.

5.6 Enhancement of Reservations

For reservations, you can't use Transaction MIGO. As previously stated, reservations are created or changed in Transactions MB21 and MB22. Therefore, you can't use any enhancements for reservations, for example, the BAdI MB_MIGO_BAdI, to prepopulate fields or to carry out input checks. Even here there is some assistance — the BAdI MB_RESERVATION_BAdI.

This BAdI is essentially suitable for two situations. First, there is the method `DATA_MODIFY`, which is called when an item is entered before the detail screen view (see Figure 5.11) is displayed. You can use this method to prepopulate individual fields with values. The method `DATA_CHECK` is then called. Here you have the option to carry out custom checks and display, if applicable, a warning or error message.

The screenshot displays the SAP 'Create Reservation: Details 0001 / 0001' screen. The interface includes a menu bar (Reservation, Edit, Goto, Environment, System, Help) and a toolbar. The main content area is divided into several sections:

- Basic Data:**
 - Movement Type: 201
 - Plant: 1000
 - Material: 100-100
 - Stor. Location: (empty)
 - GI for cost center: (empty)
 - Werk Hamburg: (empty)
 - Casings: (empty)
- Quantity in:**
 - Unit of Entry: 2 (PC)
 - Stockkeepg Unit: 2 (PC)
 - Qty is Fixed
- Further Information:**
 - Reqmt Date: 08/02/2010
 - Mvt Allowed
 - Business Area: 9900
 - Cost Center: 1000 (Corporate Services)
 - G/L Account: 890000
 - Business Area: 9900
 - Recipient: (empty)
 - Unl. Point: (empty)
 - Text: (empty)

Figure 5.11 Detailed Entry in the Reservation

For both cases, a small example will help you implement these enhancements.

5.6.1 Prepopulating Fields

The prepopulation of fields via the method `DATA_MODIFY` works relatively simply. However, you can't overwrite all of the fields of the document. All fields of Table 5.13 can basically be overwritten. However, this list is further restricted by the current field status. Fields that haven't been integrated may also not be overwritten by the BAdI method. Therefore, internal Table `IT_CHANGEABLE` is transferred as a parameter that contains all changeable fields. All changes you make in the BAdI to fields not contained there are subsequently not copied.

Field	Description
RESB-MATNR	Material number
RESB-WERKS	Plant
RESB-LGORT	Storage location
RESB-CHARG	Batch
RESB-SAKNR	GL account number
RESB-ERFMG	Quantity in unit of entry
RESB-ERFMG	Unit of entry
RESB-BDMNG	Requirement quantity
RESB-FMENG	Characteristic: Fixed quantity
RESB-KZEAR	Characteristic: Final issue of the reservation
RESB-XLOEK	Item deleted
RESB-WEMPF	Ship-to party
RESB-ABLAD	Unloading point
RESB-SGTXT	Item text
RESB-BDMNG	Requirement date of the components
RESB-XWAOK	Characteristic: Goods movement to the reservation permitted

Table 5.13 Fields of the Reservation that Can Be Overwritten

Furthermore, the parameter `I_NEW_ITEM` exists, which means that this item will run the first time in the method. If your prepopulation is only supposed to be a

suggestion that can be overwritten by the user at any time via other values, then it makes sense to check this parameter and to overwrite the field content only when it is run for the first time.

You can find the complete parameter interface of the method `DATA_MODIFY` in Table 5.14.

Parameter	Description
<code>IT_CHANGEABLE</code>	Table of the changeable fields.
<code>I_NEW_ITEM</code>	Current item is new and is handled the first time in the method.
<code>CS_RKPF</code>	Header data of the reservation. The fields of this structure cannot be changed.
<code>CS_RESB</code>	Structure on an item of the reservation; the fields contained may be overwritten if the field is contained in Table <code>IT_CHANGEABLE</code> .

Table 5.14 Parameters of the Method `DATA_MODIFY`

In a small example, the field GL account (`RESB-SAKNR`) is populated for all reservations in plant 1000 with the GL account number 400000.

1. Create an implementation for the BAdI `MB_RESERVATION_BADI` in Transaction SE19. You can choose, for example, `ZMB_RESERVATION_BADI` as the name of the implementation.
2. Now navigate to the method `DATA_MODIFY`. The GL account is only a default value that can be overwritten by the user at any time. For this reason, the method is executed for an item only on the first run (i.e., before the detail screen for an item may have been displayed the first time). Therefore, check the content of `I_NEW_ITEM`, and exit the method if necessary.
3. Next, you must check whether the field may at all be changed. For this, check whether the field is available in Table `IT_CHANGEABLE`.

This internal table is defined via a table type `TDTAB_C132` in the ABAP Dictionary; however, this is done without using a structure but instead via an integrated data type. For this reason, the only field of this internal table doesn't have any defined name. However, you can always access the internal table via the field name `TABLE_LINE`.

4. When the checks are successful, you can overwrite the field value in the structure CS_RESB. A suitable example is given in Listing 5.29.

```

METHOD if_ex_mb_reservation_badi~data_modify .
* Prepopulate GL account
* Prepopulation is only to be a default value, therefore
* only execute item on first run
  CHECK i_new_item IS NOT INITIAL.

* Check whether GL account is changeable
  READ TABLE it_changeable WITH KEY table_line = 'RESB-SAKNR'
  TRANSPORTING NO FIELDS.

* Suggest GL account for plant 1000
  IF sy-subrc = 0 AND
    cs_resb-werks = '1000'.
    cs_resb-saknr = '400000'.
  ENDIF.
ENDMETHOD.

```

Listing 5.29 Coding on the Method DATA_MODIFY

5.6.2 Checking Entries

Custom input checks are also simple to implement. To implement your checks, the header and item data are provided. To trigger a warning or error message, you can work perfectly normally with the MESSAGE command, but you must trigger the exception EXTERNAL_MESSAGE.

You can find the parameter interface of the method DATA_CHECK in Table 5.15.

Parameter	Description
IT_CHANGEABLE	This internal table again contains the list of current changeable fields. This doesn't usually play any major role for this method.
IS_RKPF	The header data of the reservation.
IS_RESB	Structure on an item of the reservation.
I_NEW_ITEM	Current item is new and is handled the first time in the method.

Table 5.15 Parameters of the Method DATA_CHECK

As an example, the fields are supposed to be checked for the ship-to party (RESB-WEMPF) and the unloading point (RESB-ABLAD). The specification of a ship-to party is mandatory. If this specification is missing, it must be confirmed by an error message. If there is no entry in the unloading point field, this is indicated by a warning message.

1. Navigate to the implementation of the BADIs that you created in the previous example on method `DATA_MODIFY`. Then, switch to the method `DATA_CHECK`.
2. The method is only called for an item when the item is new or has been changed. Therefore, the evaluation of the parameter `I_NEW_ITEM` isn't absolutely necessary. Check whether the ship-to party is filled, and display an error message. To trigger the exception, you need the addition `RAISING EXTERNAL_MESSAGE`.
3. So that the message only appears for the unloading point, which is just a warning when the item is entered, but not in the case of each subsequent change, you can at this point also check the parameter `I_NEW_ITEM`. If the parameter is set, and the unloading point hasn't been completed, a warning message is displayed. You must also use the addition `RAISING EXTERNAL_MESSAGE` with regard to warning messages.

By displaying the warning message, you can always navigate to the detail screen in which the unloading point can be entered. This is usually only shown when an important piece of information is missing. You can find the relevant example in Listing 5.30.

```
METHOD if_ex_mb_reservation_badi~data_check .
* The ship-to party must be filled.
  IF is_resb-wempf is initial.
*   Message: "Please specify a ship-to party"
    MESSAGE e100(ZMM) RAISING external_message.
  ENDIF.

* The unloading point should be filled
* Check only on initial run of item
  IF i_new_item is not initial AND
    is_resb-ablad IS INITIAL.
*   Message: "Unloading point was not entered!"
    MESSAGE w101(ZMM) RAISING external_message.
  ENDIF.
ENDMETHOD.
```

Listing 5.30 Coding on the Method `DATA_CHECK`

Index

A

ABAP Dictionary, 45, 97, 148

ABAP list, 78

AC03, 222

Acceptance, 90

Account

assignment, 139, 238

assignment data, 71

assignment field, 146

determination, 140

modification constant, 139, 240

type, 149

Chart of accounts, 140

GR/IR account, 139

Accounting document, 171

Action, 95

Append structure, 150

Archiving, 218, 236, 254

Archiving object

MM_INVBEL, 237

MM_MATBEL, 237

ARC_MM_EBAN_CHECK, 219

ARC_MM_EBAN_PRECHECK, 218

ARC_MM_EBAN_WRITE, 219

ARC_MM_EINA_CHECK, 219

ARC_MM_EINA_WRITE, 220

ARC_MM_EKKO_WRITE, 220

ARC_MM_INVBEL_CHECK, 237

ARC_MM_INVBEL_WRITE, 237

ARC_MM_MATBEL_CHECK, 236

ARC_MM_MATBEL_WRITE, 237

ARC_MM_REBEL_CHECK, 254

ARC_MM_REBEL_WRITE, 254

Assignment operator, 63

Attribute, 28, 55, 63, 104

B

Background check, 151

BAdI, 22, 93

Activate, 27, 34

ARC_MM_EBAN_CHECK, 219

ARC_MM_EBAN_PRECHECK, 218

ARC_MM_EBAN_WRITE, 219

ARC_MM_EINA_CHECK, 219

ARC_MM_EINA_WRITE, 220

ARC_MM_EKKO_CHECK, 220

ARC_MM_EKKO_WRITE, 220

ARC_MM_INVBEL_CHECK, 237

ARC_MM_INVBEL_WRITE, 237

ARC_MM_MATBEL_CHECK, 236

ARC_MM_MATBEL_WRITE, 237

ARC_MM_REBEL_CHECK, 254

ARC_MM_REBEL_WRITE, 254

Classic, 22, 29

Deactivate, 27, 34

Definition, 23

Documentation, 24, 31

Filter-dependent, 24

Implementation, 23, 24

MB_ACCOUNTING_DISTRIBUTE, 233

MB_CHECK_LINE_BADI, 229

MB_DOCUMENT_BADI, 227

MB_GOODSMOUMENT_DCI, 233

MB_INSMK_WIP_CHANGE, 229

MB_MIGO_BADI, 28, 118, 228

MB_MIGO_ITEM_BADI, 228

MB_RESERVATION_BADI, 235

ME_BAPI_PO_CUST, 199

ME_BAPI_PR_CUST, 195

ME_CHDOC_ACTIVE, 212

ME_CHECK_ALL_ITEMS, 200

ME_CIP_ALLOW_CHANGE, 212

ME_COMMITMENT_PLAN, 210

ME_COMMITMENT_RETURN, 211

ME_DEFINE_CALCTYPE, 208

ME_GUI_PO_CUST, 28, 37, 201

ME_HOLD_PO, 22, 200

MEOUT_BAPI_CUST, 205

ME_POHIST_DISP_CUST, 30, 200

ME_PO_PRICING_CUST, 209

ME_PROCESS_OUT_CUST, 205

- ME_PROCESS_PO_CUST*, 37, 194, 202
ME_PROCESS_REQ_CUST, 38
ME_PURCHDOC_POSTED, 201
ME_RELEASE_CREATE, 206
ME_REQ_HEADER_TEXT, 196
ME_REQ_OI_EXT, 211
ME_REQ_POSTED, 196
ME_TAX_FROM_ADDRESS, 203
MM EDI DESADV IN, 216
MMSRV_SM_BAPI_CUST, 221
MMSRV_SM_MAIN, 221
MMSRV_SM_NOTIFY, 222
MRM_BLOCKREASON_DELETE_CUST, 249
MRM_DOWNPAYMENT, 248
MRM_ERS_HDAT_MODIFY, 241
MRM_ERS_IDAT_MODIFY, 241
MRM_HEADER_CHECK, 242
MRM_HEADER_DEFAULT, 242
MRM_INVOICE_UPDATE, 247
MRM_ITEM_CUSTFIELDS, 145, 241
MRM_MRIS_HDAT_MODIFY, 243
MRM_MRIS_IDAT_MODIFY, 243
MRM_MRKO_HDAT_MODIFY, 243
MRM_PARTNER_CHECK, 247
MRM_PAYMENT_TERMS, 244
MRM_RELEASE_CHECK, 244
MRM_RETENTIONS, 248
MRM_TOLERANCE_GROUP, 244
MRM_TRANSACT_DEFAULT, 245
MRM_UDC_DISTRIBUTE, 245
MRM_VARIANCE_TYPE, 246
MRM_WT_SPLIT_UPDATE, 245
New, 29
SMOD_MRFLB001, 207
Usable multiple times, 23
WRF_MRM_ASSIGN_TEST, 246
WRF_PREPAY_INVOICE, 246
- Balance sheet valuation, 238
- BAPI
- BAPI_CONTRACT_CHANGE*, 205, 206
BAPI_CONTRACT_CREATE, 205, 206
BAPI_PO_CHANGE, 199, 202
BAPI_PO_CREATE1, 199, 202
BAPI_PR_CHANGE, 195
BAPI_PR_CREATE, 195
BAPI_PR_GETDETAIL, 195
BAPI_REQUISITION_CREATE, 198
BAPI_SAG_CHANGE, 205
BAPI_SAG_CREATE, 205
BAPI_SERVICE_CHANGE, 221
BAPI_SERVICE_CREATE, 221
BAPI_USER_GET_DETAIL, 84
 BAPIRET2, 124, 126
 Basic list, 78
 BASO0001, 222
 Batch master, 231
 BEXCLUDE, 182
 BKPF, 172
 BLAREL, 217
 Boolean class, 173, 183
 B_RESULT, 179
 BSEG, 172
 Business Add-In -> see BAdI, 22, 93
 Business logic, 61
 Business partner, 247
- ## C
-
- CALL FUNCTION ... IN UPDATE TASK, 118
C_EXIT_PARAM_CLASS, 179
C_EXIT_PARAM_FIELD, 179, 185
C_EXIT_PARAM_NONE, 179
 CHAIN, 164
 Change
 Message class, 174
 Change document, 17, 204
 CHANGING Parameters, 50
 Check, 40, 71, 173
 CHECK_HEADER, 123, 124
 CHECK_ITEM, 123, 124
 CI_DRSEG_CUST, 147
 Class, 155
 Boolean, 173, 183
 CL_EXITHANDLER, 158
 Implementing, 33, 55, 57
 interface, 24
 Classic BAdI, 22, 29
 CL_EXITHANDLER, 158
 CLOSE, 41
 CMOD, 18, 21
 COBL_MRM, 150

Commitment, 199
 Function, 210
 Plan, 210
 COMMIT WORK, 52
 Communication, 145
 Company code, 168
 Component, 18, 19
 Composite enhancement implementation, 31
 Configuration data, 212
 Consignment settlement, 243
 Consignment stock processing, 252
 Constant
 C_EXIT_PARAM_CLASS, 179
 C_EXIT_PARAM_FIELD, 179, 185
 C_EXIT_PARAM_NONE, 179
 Constant value
 Substitution, 184
 Contract, 205
 release order, 207
 Conventional Invoice Verification, 254
 CS_GOITEM, 115
 CUSTOM_DATA_GET, 147, 158
 CUSTOMDATA_MODIFY, 147, 156
 CUSTOM_DATA_TRANSFER, 158
 Customer namespace, 148, 167
 Custom field, 93

D

Data
 communication, 158
 entry sheet, 89
 exchange, 148
 Global, 43, 167
 type, 167
 Database, 50, 147
 table, 46, 50
 DATA_CHECK, 134
 Data field
 OK-CODE, 44
 DATA_MODIFY, 134, 135
 Default quantity, 129, 132
 Default value, 93
 Definition, 23, 42
 Degree of activation, 176, 185
 Deletion Indicator, 79

DELFOR, 217
 DELINS, 217
 Delivery address, 215
 Delivery costs, 241, 245
 Delivery date, 127
 DELVRY01, 216
 DESADV, 216, 217
 Direct type input, 107
 Display mode, 99, 100, 161
 Display transaction, 54
 Distribution indicator, 86
 Document item, 46
 Document number, 52
 Down payment, 248
 DRSEG, 147
 DRSEG_CI, 147, 149
 Dynamic selection, 75
 Dynpro, 39, 54, 94, 146

E

Enhancement, 18, 37
 BASO0001, 222
 IQSM0007, 232
 LIFO0040, 238
 LMEKO001, 209
 LMEKO002, 210
 LMELA002, 233
 LMELA010, 217, 233
 LMEQR001, 213
 LMEXF001, 210
 LMR1M001, 250
 LMR1M002, 240
 LMR1M003, 250
 LMR1M004, 250
 LMR1M005, 251
 LMR1M006, 251
 LWBON001, 218
 LWSUS001, 213
 M06B0001, 196
 M06B0002, 197
 M06B0003, 197
 M06B0004, 198
 M06B0005, 197
 M06E0004, 203
 M06E0005, 203

- MB_CF001*, 230
 - MBCF0002*, 230
 - MBCF0005*, 230
 - MBCF0006*, 234
 - MBCF0007*, 236
 - MBCF0009*, 231
 - MBCFC003*, 231
 - MBCFC004*, 231
 - ME590001*, 204
 - MEETA001*, 208
 - MEFLD004*, 234
 - MEQUERY1*, 212
 - MEREQ001*, 38, 198
 - MEVME001*, 235
 - MM06E001*, 217
 - MM06E003*, 213
 - MM06E004*, 204
 - MM06E005*, 37, 214
 - MM06E007*, 17, 204
 - MM06E008*, 207
 - MM06E009*, 214
 - MM06E010*, 215
 - MM06E011*, 199
 - MM06L001*, 216
 - MM08R001*, 255
 - MM08R002*, 249, 255
 - MMDA0001*, 215
 - MMFAB001*, 207
 - MRFLB001*, 208
 - MRMH0001*, 252
 - MRMH0002*, 252
 - MRMH0003*, 253
 - MRMN0001*, 253
 - NIWE0000*, 238
 - NIWE0001*, 239
 - NIWE0002*, 239
 - NIWE0003*, 240
 - RMVKON00*, 252
 - SRVDET*, 222
 - SRVEDIT*, 223
 - SRVESI*, 223
 - SRVESKN*, 223
 - SRVESLL*, 224
 - SRVESSR*, 224
 - SRVEUSCR*, 224
 - SRVKNTTP*, 225
 - SRVLIMIT*, 225
 - SRVMAIL1*, 226
 - SRVMSTLV*, 226
 - SRVREL*, 226
 - SRVSEL*, 227
 - XMBF0001*, 232
 - Enhancement category, 149
 - Enhancement implementation, 31
 - Enhancement spot, 29, 30
 - ES_BADI_INVOICE_UPDATE*, 247
 - ES_BADI_ME_BAPI*, 195, 199, 205
 - ES_BADI_ME_POHIST*, 30, 200
 - ES_BADI_MRM_DOWNPAYMENT*, 248
 - ES_BADI_MRM_PARTNER*, 247
 - ES_BADI_MRM_RETENTION*, 248
 - ES_COMMITMENT_PLAN*, 210
 - MB_GOODSMOUMENT*, 229, 233
 - ME_PROCESS_OUT*, 205
 - MRM_BLOCKREASON_DELETE*, 249
 - Enjoy purchase order, 37
 - Error message, 69
 - ERS procedure, 241
 - EXECUTE, 40
 - Exit
 - Substitution*, 184
 - Exit routine
 - Substitution*, 185
 - Validation*, 177
 - Expert mode, 175
 - Explicit enhancement spot, 29
 - EXPORT ... COMPRESSION ON, 120
 - Export parameter, 157
 - EXPORT ... TO DATA BUFFER, 119
 - External Services Management, 83, 221
- ## F
-
- Field
 - BEXCLUDE*, 182
 - Custom*, 93
 - Property*, 45
 - SGTXT*, 228, 230, 250
 - Status*, 59
 - Symbol*, 190
 - Field-field assignment
 - Substitution*, 184
 - FIELDSELECTION_*, 41

Flow logic, 45, 99, 148, 157, 163
 Flow module, 154
 Form
 get_exit_titles, 178, 185
 Forward declaration, 57
 Function group, 38, 94
 Function module, 46, 49, 94
 BAPI_USER_GET_DETAIL, 84

G

GB01, 182
get_exit_titles, 178, 185
 GET_INSTANCE_FOR_SUBSCREENS, 158
 GL account, 135, 136
 GL account tab, 145
 Global data, 43, 167
 Goods issue, 95
 Goods movement, 93
 Goods receipt, 95, 127, 130, 233
 Goods receipt/issue slip, 230
 GR block stock, 127
 GR/IR-account, 139
 GR/IR account determination, 240
 GUID, 119

H

Header data, 43
 HOLD_DATA_DELETE, 119, 122
 HOLD_DATA_LOAD, 119, 121
 HOLD_DATA_SAVE, 119, 120

I

I_CLASS_ID, 109
 IDoc
 Basic DELVRY01, 216
 IDOC_INPUT_SRVMAS, 222
 processing, 216
 IF_PURCHASE_ORDER_ITEM_MM, 62
 IF_PURCHASE_ORDER_MM, 62, 67
 Implementation, 23, 24, 95, 155
 Implementing class, 33, 55, 57

Implicit enhancement spot, 29
 IMPORT ... ACCEPTING TRUNCATION, 123
 Import data, 204
 IMPORT ... FROM DATA BUFFER, 119
 Import parameter, 49, 52
 Inbound delivery, 216
 Include, 20
 LMEVIEWSF01, 44
 Incoming invoice, 141
 InfoSet, 76
 INIT, 95, 107
 INITIALIZE, 40
 Input check, 93, 123
 INSERT, 120
 Instance, 158
 Instance attribute, 108
 Interface, 31, 62, 155, 159
 IF_PURCHASE_ORDER_ITEM_MM, 62
 IF_PURCHASE_ORDER_MM, 62, 67
 Interface tab, 24
 Internal table, 48
 IN UPDATE TASK, 52
 INVALIDATE, 70
 Inventory document, 237
 Inventory Management, 227
 Invoice, 165
 INVOICE_DATA_GET, 157, 159
 INVOICE_DATA_TRANSFER, 147, 156
 Invoice document, 145, 146, 147
 Item, 156
 Header, 156
 Invoice Verification, 240
 Conventional, 254
 Invoicing plan settlement, 243
 IQSM0007, 232
 Item data, 42
 ITEM_MODIFY, 125
 Item number, 162
 Item text, 125, 135, 174, 214, 228, 230, 250

L

LIFO
 0040, 238
 Valuation, 238
 Limit check, 225

LINE_DELETE, 96
 LINE_MODIFY, 96, 114
 LMEKO001, 209
 LMEKO002, 210
 LMELA002, 233
 LMELA010, 217, 233
 LMEQR001, 213
 LMEVIEWSF01, 44
 LMEXF001, 210
 LMR1M001, 250
 LMR1M002, 240
 LMR1M003, 250
 LMR1M004, 250
 LMR1M005, 251
 LMR1M006, 251
 Logistics Information Systems (LIS), 218
 Logistics Invoice Verification, 145, 164, 175, 240
 LOOP AT SCREEN, 99
 Lowest value comparison, 238
 LWBON001, 218
 LWSUS001, 213

M

M06B0001, 196
 M06B0002, 197
 M06B0003, 197
 M06B0004, 198
 M06B0005, 197
 M06E0004, 203
 M06E0005, 203
 MAP_DYNPRO_FIELDS, 39, 57
 Market price analysis, 239
 Material document, 95, 227
 number, 97
 Material group, 166
 Materials Management (MM), 37
 MB21, 134
 MB22, 134
 MB_ACCOUNTING_DISTRIBUTE, 233
 MB_CF001, 230
 MBCF0002, 230
 MBCF0005, 230
 MBCF0006, 234
 MBCF0007, 236
 MBCF0009, 231
 MBCFC003, 231
 MBCFC004, 231
 MB_CHECK_LINE_BADI, 229
 MB_DOCUMENT_BADI, 227
 MB_GOODSMOVMovement, 229, 233
 MB_GOODSMOVMovement_DCI, 233
 MB_INSMK_WIP_CHANGE, 229
 MB_MIGO_BADI, 28, 118, 228
 MB_MIGO_ITEM_BADI, 228
 MB_RESERVATION_BADI, 235
 ME23N, 54
 ME31K, 210
 ME31L, 210
 ME59, 204
 ME84, 207
 ME590001, 204
 ME_BAPI_PO_CUST, 199
 ME_BAPI_PR_CUST, 195
 ME_CHDOC_ACTIVE, 212
 ME_CHECK_ALL_ITEMS, 200
 ME_CIP_ALLOW_CHANGE, 212
 ME_COMMITMENT_PLAN, 210
 ME_COMMITMENT_RETURN, 211
 ME_DEFINE_CALCTYPE, 208
 MEETA001, 208
 MEFLD004, 234
 ME_GUI_PO_CUST, 28, 37, 201
 MAP_DYNPRO_FIELDS, 54
 TRANSPORT_FROM_DYNP, 62
 TRANSPORT_FROM_MODEL, 61
 TRANSPORT_TO_DYNP, 61
 TRANSPORT_TO_MODEL, 62
 ME_HOLD_PO, 22, 200
 MEOUT_BAPI_CUST, 205
 ME_POHIST_DISP_CUST, 30, 200
 ME_PO_PRICING_CUST, 209
 ME_PROCESS_OUT, 205
 ME_PROCESS_OUT_CUST, 205
 ME_PROCESS_PO_CUST, 37, 202
 CLOSE, 68
 FIELDSELECTION_HEADER, 54
 FIELDSELECTION_ITEM, 54
 INITIALIZE, 68
 OPEN, 68
 POST, 68
 ME_PROCESS_REQ_CUST, 38, 194

- ME_PURCHDOC_POSTED, 201
- MEQUERY1, 212
- ME_RELEASE_CREATE, 206
- MEREQ001, 38, 198
- ME_REQ_HEADER_TEXT, 196
- ME_REQ_OI_EXT, 211
- ME_REQ_POSTED, 196
- Message, 137, 173
 - BLAREL, 217
 - DELFOR, 217
 - DELINS, 217
 - DESADV, 217
 - ORDCHG, 217
 - ORDERS, 217
 - ORDRSP, 217
 - REQOTE, 217
- Message class
 - Change, 174
- Message log, 126
- Metafield, 54, 57
- ME_TAX_FROM_ADDRESS, 203
- Method, 24, 25
 - CHECK_HEADER, 123, 124
 - CHECK_ITEM, 123, 124
 - CUSTOM_DATA_GET, 158
 - CUSTOMDATA_MODIFY, 156
 - CUSTOM_DATA_TRANSFER, 158
 - DATA_CHECK, 134
 - DATA_MODIFY, 134, 135
 - GET_INSTANCE_FOR_SUBSCREENS, 158
 - HOLD_DATA_DELETE, 119, 122
 - HOLD_DATA_LOAD, 119
 - HOLD_DATA_SAVE, 119, 120
 - INIT, 95, 107
 - INVOICE_DATA_GET, 157
 - INVOICE_DATA_TRANSFER, 156
 - LINE_DELETE, 96
 - LINE_MODIFY, 96, 114
 - MODE_SET, 95, 108
 - PAI_DETAIL, 96, 114
 - PAI_HEADER, 96
 - PBO_DETAIL, 96, 112
 - PBO_HEADER, 96
 - POST_DOCUMENT, 96, 117
 - RESET, 95, 108
 - STATUS_AND_HEADER, 95, 119
 - TABPAGE_LABEL_SET, 147, 158
- MEVME001, 235
- MIGO, 93, 95, 228
 - Action, 106
 - check, 123
 - Note, 112, 119
 - post, 123
 - Reference document, 106
- MIGO_CLASS_ID, 105
- MIR4, 151
- MIR6, 151
- MIR7, 175
- MIRA, 151
- MIRO, 145, 147, 175
- ML81N, 89
- ML86, 223, 226
- ML87, 223, 226
- MLS5, 226
- MM06E001, 217
- MM06E003, 213
- MM06E004, 204
- MM06E005, 37, 214
- MM06E007, 17, 204
- MM06E008, 207
- MM06E009, 214
- MM06E010, 215
- MM06E011, 199
- MM06L001, 216
- MM08R001, 255
- MM08R002, 249, 255
- MMDA0001, 215
- MM EDI_DESADV_IN, 216
- MMFAB001, 207
- MM_INVBEL, 237
- MM_MATBEL, 237
- MM_MESSAGES_MAC, 70
- MMMFD, 57
- MMPUR_DYNAMIC_CAST, 62
- MMPUR_MESSAGE_FORCED, 70
- MMPUR_METAFIELD, 57, 70
- MMSRV_SM_BAPI_CUST, 221
- MMSRV_SM_MAIN, 221
- MMSRV_SM_NOTIFY, 222
- MODE_SET, 95, 108
- Modification, 29
- MODULE, 159
- MODULE ... ON CHAIN-REQUEST, 164
- Module pool, 151

MOVE-CORRESPONDING, 113
 MRBP, 151
 MRBR, 244
 MRFLB001, 208
 MRIS, 243
 MRKO, 243
 MRM_BLOCKREASON_DELETE, 249
 MRM_BLOCKREASON_DELETE_CUST, 249
 MRM_DOWNPAYMENT, 248
 MRM_ERS_HDAT_MODIFY, 241
 MRM_ERS_IDAT_MODIFY, 241
 MRMH0001, 252
 MRMH0002, 252
 MRMH0003, 253
 MRM_HEADER_CHECK, 242
 MRM_HEADER_DEFAULT, 242
 MRM_INVOICE_UPDATE, 247
 MRM_ITEM_CUSTFIELDS, 145, 241
 MRM_MRIS_HDAT_MODIFY, 243
 MRM_MRIS_IDAT_MODIFY, 243
 MRM_MRKO_HDAT_MODIFY, 243
 MRMN0001, 253
 MRM_PARTNER_CHECK, 247
 MRM_PAYMENT_TERMS, 244
 MRM_RELEASE_CHECK, 244
 MRM_RETENTIONS, 248
 MRM_TOLERANCE_GROUP, 244
 MRM_TRANSACT_DEFAULT, 245
 MRM_UDC_DISTRIBUTE, 245
 MRM_VARIANCE_TYPE, 246
 MRM_WT_SPLIT_UPDATE, 245
 MSEG, 96
 MSEG-ZEILE, 117
 Multiple account assignment, 86

N

NIWE0000, 238
 NIWE0001, 239
 NIWE0002, 239
 NIWE0003, 240
 Number range, 197, 213, 250

O

OB28, 172, 174
 OBBH, 182
 Object, 62
 OBYC, 140
 OK-CODE, 44
 OPEN, 40
 ORDCHG, 217
 Order, 22
 Hold, 22
 Order confirmation, 217
 ORDERS, 217
 ORDRSP, 217
 Outline agreement, 205
 Overdelivery quantity, 129, 131
 OXK3, 146

P

PAI_DETAIL, 96, 114
 PAI_HEADER, 96
 Parameter
 Interface, 26
 Parameters
 CHANGING, 50
 Interface, 31
 Parameter type
 C_EXIT_PARAM_CLASS, 179
 C_EXIT_PARAM_FIELD, 179, 185
 C_EXIT_PARAM_NONE, 179
 Parked document, 175
 Partner role, 247
 Pass by value, 50
 Pass By Value, 102
 PBO_DETAIL, 96, 112
 PBO_HEADER, 96, 109
 Pipeline
 Processing, 252
 Settlement, 243
 POST, 41
 POST_DOCUMENT, 96, 117
 Posting transaction
 WRX, 140
 Prerequisite, 173, 181
 Price variance, 164

Pricing, 200, 208
 PROCESS_ACCOUNT, 40, 71
 PROCESS_AFTER INPUT, 148
 PROCESS_BEFORE OUTPUT, 148
 PROCESS_HEADER, 40, 71
 Processing
 IDoc, 216
 Processing type, 102
 PROCESS_ITEM, 40, 71
 PROCESS_SCHEDULE, 40, 71
 Program
 RGGBR000, 177
 RGGBS000, 185
 RM06BD70, 219
 RM06BV70, 218
 RM06BW70, 219
 RM06ED47, 220
 RM06EFLB, 208
 RM06EW47, 220
 RM06ID47, 220
 RM06IW47, 219, 220
 RM07IARCS, 237, 238
 RM07IDELS, 238
 RM07MADES, 237
 RM07MARCS, 236, 237
 SAPLMR1M, 189
 Program table, 154
 Project, 18, 21
 Purchase order, 37, 199
 Document overview, 212
 Input check, 40, 69
 Message collector, 69
 Purchase order date, 128
 Purchase order history, 34, 200
 Purchase order number, 47
 Selection variant, 72
 Purchase order requisition, 194
 Purchase requisition, 38
 Selection variant, 72
 Purchasing, 37
 Purchasing information record, 219

Q

Quantity variance, 164
 Query, 76
 Query Painter, 76

R

RAWSTRING, 119, 120, 122
 RBDRSEG, 150
 RBMA, 150
 Reference, 62, 95
 Reference key, 183
 Release procedure, 196
 Release strategy, 203
 Report
 RGUGBR00, 174, 183
 RGUGBR28, 174
 RM06EFAB, 207
 RMMR1MRB, 253
 Repository Browser, 98, 152
 REQOTE, 217
 Request for quotation (RFQ), 37
 Reservation, 134, 235
 RESET, 95, 108
 Retention amount, 248
 Return item, 211
 Return-table, 123
 RGGBR000, 177
 RGGBS000, 185
 RGUGBR00, 174, 183
 RGUGBR28, 174
 RM06BD47, 219
 RM06BV70, 218
 RM06BW47, 219
 RM06BW70, 219
 RM06ED47, 220
 RM06EFAB, 207
 RM06EFLB, 208
 RM06EW47, 220
 RM06ID47, 220
 RM06IW47, 219, 220
 RM07IARCS, 237, 238
 RM07IDELS, 238
 RM07MADES, 237
 RM07MARCS, 236, 237
 RMMR1MRB, 253
 RMVKON00, 243, 252
 RSEG, 150
 Runtime behavior, 34

S

- SAP Enhancement Framework, 29
- SAPLMR1M, 189
- SAP Query, 75
- Schedule, 129
- Scheduling agreement, 129, 205
- Scheduling agreement schedule, 129
- Scheduling data, 71
- SCREEN-NAME, 163
- Screen Painter, 98, 152
- SE11, 42, 120
- SE18, 22, 30
- SE19, 23, 24, 31, 155
- SE24, 59
- SE80, 43, 98
- Selection
 - dynamic*, 75
 - list*, 82
 - variant*, 75
- Serial number, 232
- Service catalog, 226
- Service entry, 222
 - entry sheet*, 223
- Service line, 83, 87
- Service master, 221
- Service specification, 223
- SGTXT, 228, 230, 250
- Shipping notification, 217
- Ship-to party, 135, 138
- Signature, 26
- SMOD, 18
- SMOD_MRFLB001, 207
- Source, 213
- Source document
 - Substitution*, 188
- SQ01, 76
- SQ02, 76
- SRVDET, 222
- SRVEDIT, 223
- SRVESI, 223
- SRVESKN, 223
- SRVESLL, 224
- SRVESSR, 224
- SRVEUSCR, 224
- SRVKNTP, 225
- SRVLIMIT, 225
- SRVMAIL1, 226
- SRVMSTLV, 226
- SRVREL, 226
- SRVSEL, 227
- STATUS_AND_HEADER, 95, 119
- Step, 173
- Stock determination, 232
- Storage location, 125, 231
- Structure, 101
 - CI_DRSEG_CUST*, 147
 - DRSEG*, 147
 - DRSEG_CI*, 147, 149
- Subcontracting, 212, 234
- Subcontracting component, 234
- Subscreen, 98, 145, 152, 155
- SUBSCRIBE, 39
- Substitution, 171, 182
 - Constant value*, 184
 - Exit*, 184
 - Exit routine*, 185
 - Field-field assignment*, 184
 - Source document*, 188

T

- T80D, 177, 185
- Table
 - BKPF*, 172
 - BSEG*, 172
 - GB01*, 182
 - MSEG*, 96
 - Parameter*, 50
 - T80D*, 177, 185
 - Work area*, 154
 - YDRSEG*, 190
- Table control, 147, 152, 160
- Table Control Wizard, 152
- TABLE_LINE, 136
- TABPAGE_LABEL_SET, 147, 158
- Tabstrip, 39
- Text field, 97
- Text symbol, 158

Tolerance check, 145, 164, 249

Tolerance key, 165, 166

Tolerance limit, 129, 164

Transaction

AC03, 222

CMOD, 18, 21

MB21, 134

MB22, 134

ME23N, 54

ME31K, 210

ME31L, 210

ME59, 204

ME84, 207

MIGO, 93, 106, 112, 119, 228

MIR4, 151

MIR6, 151

MIR7, 175

MIRA, 151

MIRO, 145, 147, 175

ML81N, 89

ML86, 223, 226

ML87, 223, 226

MRBP, 151

OB28, 172, 174

OBH, 182

OBYC, 140

OXK3, 146

SE11, 42, 120

SE18, 22, 30

SE19, 23, 24, 31, 155

SE24, 59

SE80, 98

SMOD, 18

SQ01, 76

SQ02, 76

TRANSPORT_FROM_DYNPRO, 40

TRANSPORT_FROM_MODEL, 39

TRANSPORT_TO_DYNPRO, 39

TRANSPORT_TO_MODEL, 40

Type, 28, 106

DESADV, 216

Type group, 57

Type input

Direct, 107

U

Underdelivery quantity, 131

Unlimited variance, 169

Unloading point, 135, 138

Upcast, 62

Update module, 50, 95, 102

Update termination, 103

Updating, 50, 117

User exit, 17

Activate, 21

Deactivate, 21

EXIT_RM06EFAB_001, 208

EXIT_RM06EFLB_001, 208

EXIT_RM06LBAT_001, 216

EXIT_RM06LBEW_001, 216

*EXIT_RMMR1MRB_**, 253

*EXIT_RMVKON00_**, 252

*EXIT_SAPLBASO_**, 222

*EXIT_SAPLEBND_**, 197, 203, 226

EXIT_SAPLEBNE_001, 198

*EXIT_SAPLEBNF_**, 196, 203

EXIT_SAPLEINL_001, 208

*EXIT_SAPLEINM_**, 217

*EXIT_SAPLEINR_**, 127, 129, 234, 235

EXIT_SAPLIE01_007, 232

EXIT_SAPLKONT_011, 141, 240

*EXIT_SAPLLIFS_**, 238

EXIT_SAPLMBMB_001, 230

*EXIT_SAPLMDBF_**, 232

EXIT_SAPLME59_001, 204

*EXIT_SAPLMEKO_**, 209, 210

EXIT_SAPLMELO_001, 216

EXIT_SAPLMEQR_001, 213

*EXIT_SAPLMEQUERY_**, 75, 80, 213

*EXIT_SAPLMEREQ_**, 198

EXIT_SAPLMEXF_001, 210

EXIT_SAPLMIGO_001, 234

EXIT_SAPLMLS_001, 83, 84, 224

EXIT_SAPLMLSL_001, 225

*EXIT_SAPLMLSP_**, 87, 88, 223, 224, 227

*EXIT_SAPLMLSR_**, 89, 223, 224, 225

EXIT_SAPLMLST_001, 226

*EXIT_SAPLMLSX_**, 223, 225, 226

EXIT_SAPLMMDA_001, 215

*EXIT_SAPLMR1M_**, 167, 249, 250

EXIT_SAPLMRM_BAPI_001, 251
*EXIT_SAPLMRMC_**, 167, 250, 251
EXIT_SAPLMRME_003, 250
*EXIT_SAPLMRMH_**, 252, 253
*EXIT_SAPLMRMN_**, 253, 254
*EXIT_SAPLMRMP_**, 167, 250, 251
*EXIT_SAPLNIWO_**, 239
*EXIT_SAPLNIW1_**, 239
*EXIT_SAPLNIW3_**, 240
*EXIT_SAPLNIWE_**, 239
EXIT_SAPLWN08_001, 218
EXIT_SAPLWN12_001, 218
EXIT_SAPLWN35_001, 218
EXIT_SAPLWSUS_001, 213
*EXIT_SAPM07DR_**, 231
EXIT_SAPMM06B_001, 197
*EXIT_SAPMM06E_**, 20, 199, 204, 205,
207, 214, 215
EXIT_SAPMM06L_001, 216
*EXIT_SAPMM07M_**, 230, 231, 232, 234
EXIT_SAPMM07R_001, 236

User group, 76
User master, 84

V

Validation, 171, 172
 Change of message class, 174
 Exit routine, 177
Valuation, 139, 238

Value contract, 207
Variable
 B_RESULT, 179
Variance, 167
 Unlimited, 169
Vendor, 127
 address, 215
 evaluation, 215
Visibility, 107

W

Work area, 75, 76, 177, 185
WRF_MRM_ASSIGN_TEST, 246
WRF_PREPAY_INVOICE, 246
WRX, 140

X

XMBF0001, 232
XML invoice, 251
XSTRING, 119

Y

YDRSEG, 190