Customer Enhancements with BDT

Applies to:
This article and code sample will be Applies to version 4.7.

Summary
The Business Data Toolset (BDT) is a central control tool for maintaining SAP programs. It supports maintenance via the use of dialog techniques, direct input and/or function modules.

The BDT is basically the tool, SAP use when they want to add new fields, change field names, change/add field groups etc for specific customers, but they are now started to allow there customers access to these maintenance transactions.

The BDT also provides generic services for consistently recurring requirements such as occur in change document lists, field groupings and the deletion program. It takes control over these objects as well as generic parts and calls the applications using predefined interfaces. The applications themselves introduce application-specific enhancements, such as writing and reading application tables.

The following topics are dealt with:
- Data Dictionary enhancements
- Dialog control
- Event technique in dialog and program logic
- Enhancements with table appends
- Enhancements with new tables
- GUI menu
- Service programs
- Interfaces

Author(s): Sudheer Junnuthula

Company: Intelligroup Asia Pvt Ltd

Created on: 31 July 2006

Author Bio
I am Sudheer Junnuthula, working in Intelligroup Asia Pvt ltd as an ABAP Developer, and I have been involved in development of various SAP technical areas including ABAP, Scripts, OO ABAP, ALV, ALV(OO) and BDT programming.
## Table of Contents

Preparing Customer Enhancement Implementation ............................................................... 3  
  Prerequisites: .......................................................................................................................... 3  
  Procedure: ............................................................................................................................... 3  
BDT Function Modules ............................................................................................................. 5  
  Direct Input ............................................................................................................................. 6  
  DDIC Structures .................................................................................................................... 6  
Field Groupings ....................................................................................................................... 10  
  Technique ............................................................................................................................... 10  
  Predefined Field Grouping Criteria ....................................................................................... 10  
  Additional Field Grouping Criteria ....................................................................................... 11  
Function Modules .................................................................................................................... 13  
  Program Logic ......................................................................................................................... 14  
    Screen Layout ....................................................................................................................... 19  
Field groups: ......................................................................................................................... 20  
Views ....................................................................................................................................... 20  
  Other Checks on Views ........................................................................................................... 22  
Note: ........................................................................................................................................ 22  
  Procedure for Field Checks .................................................................................................. 23  
Example: ................................................................................................................................... 23  
  Sections .................................................................................................................................. 24  
  Screens .................................................................................................................................. 25  
    Customizing Screen Layout/Screen Sequence with Drag & Drop ....................................... 25  
Action required: ...................................................................................................................... 26  
Future plans: ........................................................................................................................... 26  
Search Helps ............................................................................................................................ 29  
  Creating Search Helps for a Field ......................................................................................... 29  
  Expanding Search Helps for a Field ..................................................................................... 29  
Tables ...................................................................................................................................... 29  
  Sample Example: ................................................................................................................... 30  
Disclaimer and Liability Notice ............................................................................................... 46
Preparing Customer Enhancement Implementation

Prerequisites:

You must clarify the business requirements and, also, the following points:

- Where the additional contract attributes are to be placed on the interface
- Which plausibility checks the attributes must fulfill
- Whether any additional plausibility checks are needed for standard attributes
- Whether the effective validity of customer enhancement data must be independent of that of standard attributes
- How many variations are possible for each customer attribute

Only then can the type of customer enhancement be decided on and implemented

Procedure:

1) Check whether there is more current sample coding available in OSS. To do this, search for notes containing the terms "TCE" (start of function group name of sample coding) and "CACS".

2) **Write a routine.** Data base objects in IS-IS-CS are subject to special period management. The following will help you to understand data communication within commission contract processing:

3) There is a data base record for each version of a commission contract. When a commission contract is read, all versions (data records) are read from the database and stored temporarily in an internal table. The required version must be determined from the table, the valid values (for example, date fields) converted where necessary and sent to screen fields using a routine that you have written.

   - This routine can be a FORM or FUNCTION, see routine "get_val_dyn" in the sample coding

4) To implement customer enhancements, you need to know how some function modules work. If you are not familiar with these, recommending to read the documentation:

   - BUS_MESSAGE_STORE
   - BUS_PARAMETERS_ISSTA_GET
   - CACS_APPL_GET_SET
   - CACS_CSC_CACS_CTRTBU_GET
   - CACS_CSCHANDLE-INFOS (provides different services)

4) Include the area menu CACSBDT in your personal user menu.

5) **BDT Hole Structure**
Diagram 4: Events in the dialog external interface
BDT Function Modules

The BDT provides function modules that have to be called in external maintenance. Figure 4 gives an overview of the modules that are called. Note, however, that the names of the BDT function modules also contain the prefix `BUS_FOREIGN_`.

If several BDT application objects are involved in external maintenance, all function modules per application object (with the exception of `BUS_FOREIGN_INITIALIZE_ALL`) have to be called once.

The function modules are as follows:

- **BUS_FOREIGN_INITIALIZE_ALL**
  The parameters valid for all BDT application objects involved in external maintenance are transferred here. This is what the external maintenance ID is used for. It represents a separate field grouping criterion, which controls field grouping in an external interface. The type of message output decides whether messages that occur are shown directly on screen or in a table.

- **BUS_FOREIGN_INITIALIZE_OBJECT**
  For each application object, various parameters are transferred to the BDT. These are flagged and event `ISSTA` is then processed.

- **Read data**
  This does not mean that a single BDT function module is called. Prior to outputting field contents in an external interface (PBO), the current data must be determined for each table using the `Read data` function module. You will find the names of the modules in the table entries.

- **BUS_FOREIGN_HEADER_CHECK**
  After the header data is entered, call this BDT function module and transfer the content of the header data in the format for DI header data structure to it. This data is distributed to the applications using BDT event DINP1 in order to process the function modules for event `after input` for all header views (indicator within the view definitions). Any messages that may occur are either shown directly on screen or in a message table, as you defined in `BUS_FOREIGN_INITIALIZE_ALL`.

- **BUS_FOREIGN_DATA_READ**
  The data is read following successful input of the header data. To do this, call the function module. In the module, events `ISDAT` and `ISDST` to read the data are called as well as `AUTH1` for the authorization checks.

- **BUS_FOREIGN_DATA_CHECK**
  First initialize the DI structure of the participating table with the NODATA character (call function module `BUS_DI_DATA_INITIALIZE`). Now write the field values input in the screen to the DI structure and convert the content of this structure to the neutral format comprehensible to the BDT. Go through these steps for all tables maintained on the data screen. Then call `BUS_FOREIGN_DATA_CHECK` and transfer the data records to the BDT. In function module `BUS_FOREIGN_DATA_CHECK`, all data views are determined for which at least one field was transferred with a value not equal to NODATA. Event DINP2 is then processed to forward the transferred data to the applications. Function modules `after input` for the views determined above are now called. Any messages that may occur are either shown directly on screen or in a message table, as you defined in `BUS_FOREIGN_INITIALIZE_ALL`.

- **BUS_FOREIGN_FULL_MAINTENANCE**
  You can call BDT maintenance in transfer mode from external maintenance. Changes made in external maintenance are visible in BDT maintenance. Once you leave BDT maintenance, any changes you made there become visible in external maintenance too. This takes place when the data is written from the current memory to the global memory (BDT events `XCHNG`; `DSAVB` and `DTAKE`) in function module `BUS_FOREIGN_FULL_MAINTENANCE` prior to calling BDT maintenance. BDT maintenance is called in transfer mode, where the data for the instance is imported from the global memory to the current memory in BDT event `ISDAT`. When making the transition between BDT maintenance and external maintenance, the same procedure is performed in the reverse order if the user decides to copy the changes that were made. The data from the current memory is copied to the
global memory. When returning to external maintenance, the data is read from the global memory to the current memory.

- **BUS_FOREIGN_CHANGES_EXIST**
  Before saving data and/or before leaving external maintenance, the BDT asks the applications whether changes were made to the data. The external interface provides function module `BUS_FOREIGN_CHANGES_EXIST` for this purpose. The module then says whether changes were made. Internally, BDT event `XCHNG` is processed at this point.

- **BUS_FOREIGN_DATA_COLLECT**
  The application data is collected in the application that owns the table. If changes exist (call BDT event `XCHNG`), events `DSA VB`, `AUTH1` and `DCHCK` are processed internally.

- **BUS_FOREIGN_LOCAL_MEMORY_NEW**
  The content of the current memory is written to the global memory of the application function groups. BDT event `DTAKE` is processed for this purpose.

- **BUS_FOREIGN_SAVE**
  The data is read from the global memory of the application function groups and written to the database. In the function module, events `D SAVC` and `D SAVE` are processed for this purpose. Whether or not data is written to the database with an update task is determined by the setting made during initialization. You can also tell the BDT whether or not it is to perform a COMMIT WORK when you call `BUS_FOREIGN_SAVE`.

**Direct Input**

The direct input (DI) function allows you to execute a high-performance data transfer from a legacy system. The data is checked with the same routines as in a dialog and saved in the database if it contains no errors. Incorrect data is not transferred but is shown in a log.

In addition to creating data from scratch, you can also change the data in existing instances. A NODATA character (default is a slash `/`) was introduced so that the values of all data fields don’t have to be re-entered every time there is a change. If the NODATA character is transferred for a field, the old value remains.

**DDIC Structures**

In order to use direct input (DI), you have to create various DDIC structures and make sure they are recognized by the system.

**Note:** Bear in mind that the functionality of the NODATA character permits only fields belonging to data type CHAR to be included in these structures. For table fields belonging to other data types, you include a CHAR field in which the name and length is the same as the database field.

**Activities:**

- Create DDIC structures
  - Application object-specific header data (abbreviated S1)
    This structure includes all fields that are required for reading the tables (may be the same as the fields on the initial screen in dialog) as well as all fields that also serve as field grouping criteria.
    **Example:** In Business Partner, this structure is called `BUS0DIINIT`.
  - All header data (abbreviated S2)
    In addition to the fields in structure S1, S2 also includes the header data that is always needed independent of the application object. This universally valid header data is located in BDT structure `BUSSIDHDR`. This should be the first structure you insert in S2, followed by the S1 structure you created in step one.
    **Example:** In Business Partner, this structure is called `BUSHDR_D I`.
  - Data per database table (abbreviated S3)
    For each database table, you create a DI structure which contains all the fields from this table that are to be transferred. If multiple records can be maintained for each object instance in a
table (table control in dialog, for instance), you need to include another indicator in this structure that describes the type of change in each data record. Refer to data element BU_CHIND for this field. The following fixed values exist in the domain of the same name:

- U  Change
- I  Insert
- D  Delete
- M  Modify (Create or Change)

For change type 'M', the application program determines within event DINP2 whether a data record was created (record did not previously exist) or changed (record already existed).

**Example:** The following structures are included in Business Partner:

- BUS000_Dl Table BUT000 (general data)
- BUS0BK_Dl Table BUT0BK (bank details)
- BUS020_Dl Tables BUT020/ADRC (addresses)

**Note:** The field names must be unambiguous both in structure S2 and S3, as it is otherwise impossible to activate structure S4 (see below).

- All data (abbreviated S4)
  All data relevant for direct input is grouped in this structure. First insert structure S2 as an include structure, followed by the includes for all S4 structures you created. Example: In Business Partner, this structure is called BUS_DI.

- Define names of DDIC structures in the BDT
  - In the definition of your application object you store
    - the name of structure S2 in the Header data field
    - the name of structure S4 in the All data field
  - In the definition of the tables, you store the name of the associated DI structure S3 for each table.

- Other settings in the BDT
  You have to mark the Header data DI indicator for all views which fields are located in the S1 structure. This ensures that the function modules after input are processed directly after event DINP1.

**Events DINP1 and DINP2**
The same program logic is used for DI as in the dialog. Nearly all events in the dialog are called in the same sequence in DI.
Diagram 3: Events in maintenance without dialog
For runtime reasons, the application data is not written individually to the database for each instance, but in packages of 200 object instances instead. For this reason, the processing of an instance after events DTAKE and DLVE1 is terminated, as in transfer mode. Events DSAVE and DLVE2 are only executed once the data for multiple instances were noted and/or no further data exists.

In addition to the events in dialog, events DINP1 and DINP2 - which take care of the data transport from the BDT to the applications - are both important for DI.

- Event DINP1 (Direct input: Enter data in header fields)
  The content of the application object-specific header data (structure S1) is transferred from the BDT to the applications. The content corresponds to data input on the initial screen in dialog.
  **Runtime:** After ISSTA and before the header data is checked.
  **Note:** In the next step, the BDT starts the function modules at event After input for all header views (indicator ‘Header data DI’ marked in the definition of views), which checks all header data which was flagged.
  **Application area:** All applications that have their own header fields.
  **Naming convention:** <Application>_<Application object>_EVENT_DINP1.
  (Customer: Function module name has the prefix Y_ or Z_).
  **Interface:**
  → I_INIT  Header data
  Action required:
  - Read header data from the BDT in unstructured form (parameter I_INIT) and place it in a field structure with structure S2
  - Flag header data in a function group in fields that will later be subjected to checks
  **Example:** BUP_BUPA_EVENT_DINP1

- Event DINP2 (Direct input: Enter data in data fields)
  One data record for every table (without header data) is transferred from the BDT to the applications. In dialog, this corresponds to data input on the data screens.
  **Runtime:** After reading the old data (event ISDAT and ISDST) and prior to checking the data fields.
  **Application area:** All applications that have data fields in their own views. Applications that use tables.
  **Naming convention:** <Application>_<Application object>_EVENT_DINP2.
  (Customer: Function module name has the prefix Y_ or Z_).
  **Example:** BUP_BUPA_EVENT_DINP2
  **Interface:**
  → I_DATA  Data record for a table
  Action required:
  - Import data into own structure
    The import parameter I_DATA (reference structure BUSIDAT1) contains fields TBNAME and DATA. The name of the DI structure is in field TBNAME. The DATA field contains the data of the DI structure as a string. The application notes the data that is important for it in a structure that has the same construction as the DI structure.
  - Determination of relevant action with indicator ‘M’ (modify)
    For tables with multiple entry and incoming data records with action M, the action to be performed must first be determined. The data record has to be read for the existing data. If a data record already exists, activity U (change) is set, otherwise activity I (insert).
  - Evaluation of the NO-DATA indicator:
    The NO-DATA indicator can be used for changing existing data. All fields that are not to be changed contain this indicator. This indicator is decoded at event DINP2. To do this, function module BUS_DI_DATA_COMBINE has to be called. This module uses both the old and newly transferred data (importing parameters) to determine a completely new dataset (exporting parameters).
  - Noting New Dataset
    The application notes the new dataset in the designated tables and/or structures. The BDT then calls checks for those views in which the Header data DI indicator is not marked.
Field Groupings

Technique

Using field grouping, the BDT can show or hide screen fields, or make these required fields or display fields. Field grouping is done per field group. Different field grouping criteria can be defined by the applications and integrated into the field status determination carried out by the BDT.

For each criterion, the status of the field groups is stored in field status definitions. These consist of four data fields which cross-reference the data element BU_FLDSTLS. The following rules apply when interpreting field status definitions:

- The first field contains the status of field groups 1 to 250, the second contains the status of groups 251 to 500, the third 501 to 750 and the fourth 751 to 1000.
- Each character stands for a field group. In the first field, the first character stands for the field status of field group 1, the second character stands for field group 2 and so on. In the second field, the first character stands for field group 251, the second character stands for field group 252 and so on. This rule also applies in the same way to the third and fourth fields.
- The individual field statuses are represented using the following characters:
  
  + (plus) Required input
  
  . (period) Optional input
  
  * (star) Display
  
  - (minus) Hidden
  
  (SPACE) Not specified

Using set rules, the BDT links the settings for each field group, which have been made on the basis of different criteria.

Note: Of course, the field status is not maintained by the user directly in the field status definition. In the case of further field grouping criteria, the field status can be saved in a different way (for example, a separate data record for each criterion and field group).

Predefined Field Grouping Criteria

The BDT offers field grouping for

- Activity
- Object part

as a service. Each application object can use both of these predefined criteria.

Activities: Only the setting activities

- 0100 field grouping per activity
- 0101 field grouping per object part
must be defined for the application object and must bear a transaction code.

**Note:** Neither predefined criterion is contained in the table of field grouping criteria. Only the additional criteria are entered here.

**Additional Field Grouping Criteria**

Any number of additional field grouping criteria can be created for each application object. Each application can define additional criteria.

**Activities:** The following steps are necessary for defining an additional criterion:

- **Create/expand table in the DDIC**
  
The table key should correspond to the entity of the criterion. Four fields are used in the data part for storing the field status definitions which cross-reference the data element BU_FLDSTLS. The field statuses can, of course, be stored in another form, for example with a separate data record for each field group (the number of the field group then goes into the table key).

- **Create maintenance view in the DDIC**
  
The maintenance view generally consists of:

  - The key fields of the table created/expanded in the first step
  - The fields for storing the field status definitions (maintenance attribute "H": Field does not appear on the maintenance screen) and
  - Possibly a name from the text table belonging to it (maintenance attribute "R": Field is read-only).

  The maintenance status is usually "Read and change" as, in this maintenance view, entries should neither be created nor deleted.

- **Generating maintenance programs**
  
  Using transaction SE54 you can generate the maintenance program for the maintenance view created in the previous step.

- **Adjusting generated maintenance programs**
  
The field status should not be maintained by the user directly in the field status definition. The BDT offers a clearly arranged maintenance interface with selection fields. The generated maintenance program must therefore be adjusted as follows:

  - Define separate menu for the maintenance view

    The standard interface from the program SAPLSVIM is copied to the main program of the generated maintenance program. The new menu option is included in the editing menu and the application toolbar with the text "field grouping". This function must now be activated in various statuses. The function FLDM can be used as a template with the text "field grouping" in the program SAPLBUS2. The event "ST" must be defined for the maintenance view, and also the name of the main program. This means that the main program's interface, which has been expanded to include the "field grouping" function, is used instead of the standard interface SAPLSVIM.
o Calling up the function ‘field status’

In the flow logic of the maintenance view screen, a separate module is inserted in the PAI before the module LISTE_UPDATE_LISTE (applies only to single-level maintenance views). This module calls up the BDT function module BUS_FMOD_CUSTOMIZING which carries out field status maintenance using the selection fields. The module FIELDMOD_V_TB004_01 on screen SAPLBUS2/0840 can be used as a template for creating the PAI module.

- Event FMOD1: Creating a function module for reading the table

The value of the criterion should be known, at the latest, once the initial screen has been executed. The function module to be created will read the table on the basis of this value and sends the contents of the field status definitions as a result to the BDT. If the value is not yet known at one of the runtimes, the field status definitions returned to the BDT should be blank or 0.
Runtime: Before the initial screen and between the initial screen and the first data screen.
Application area: Owner application of the field grouping criterion.
Naming convention: <Application>_<Application object>_EVENT_FMOD1_<Criterion>
(Customer: Function module name has the prefix Y_ or Z_).
Example: BUP_BUPA_EVENT_FMOD2_BUP003

- Enter field grouping criterion

The criterion must now be entered into the control table of field grouping criteria. You should also define the name of the function module here, with which the BDT can call up the field statuses for the criterion.
Function Modules

The procedure for maintenance using function modules corresponds in most ways to the procedure for direct input. In the BDT, events run in exactly the same order as in direct input. Only the collection of data in a format that is both neutral and comprehensible to the BDT as well as the transfer of this data to the BDT had to be realized separately.

1. Create function module per table/table group

A function module is created for each table or group of related tables. The interface generally corresponds to the DI structure (structure or table in multiple entry). The function module converts the existing, structured data into an unstructured form comprehensible to the BDT.

- For header data, you have to convert the data to the format of structure BUSSDI2.
- You convert each data record in a table to the format of structure BUSSDI3. The field TABNAME requires the name of the DI structure and the DATA field needs the transferred data contents in CHAR format.
- You return the data in unstructured form to interface table T_DATA, which references the structure BUSSDI.

Examples:

- BUP_DI_DATA_GLOBAL_DATA
- BUP_DI_DATA_BANK_DETAILS
- BUA_DI_DATA_ADDRESS

2. Function module per application object/object part

If an application object uses divisibility, one function module per object part is usually available. For all other application objects, one function module is generally sufficient to maintain all the data for an application object. This module calls the function modules of all relevant tables/table groups (see 1) and collects the data in unstructured form in a table. The BDT function module BUS_CONTROL_MAIN_DI is then called and the content of this table is transferred.

Examples:

- BUP_DI_ROLE_GLOBAL
- BUP_DI_ROLE_CONTACT_PARTNER

3. Extensibility

If a downstream application adds other fields to a table, it also needs to add these fields to the DI structure for this table with the APPEND function. These fields can then be transferred to the function modules created in step 2, as the structure and/or table in the interface references the complete DI structure.

On the other hand, adding another table requires you to create a separate function module for it (see step 1). In order to provide a function module for each application object/object part (see 2), this application creates an additional layer in the form of its own function module.
Program Logic
Within the dialog flow, events defined by the BDT were used, for which the applications can develop separate program logic in the form of function modules. The names of several function modules can be defined for each event and these modules will then be called up automatically by the BDT.
Figure 1 provides an overview of the flow in the dialog. The events have been illustrated using a darker background.
Diagram 1: Events in dialog, save mode
The most important events for the dialog are described in detail below:

- **ISSTA (Initialization)**
  
The applications initialize their global variables and get the relevant control information from the BDT. Default values may also be assigned to the initial screen here.
  
  **Runtime:** Before the initial screen, the event will be processed in the initial screen when the activity is changed.
  
  **Application area:** All applications

  **Naming convention:** `<Application>_<Application object>_EVENT_ISSTA`

  (Customer: Function module name also has the prefix Y_ or Z_)

  **Examples:**
  - BUP_BUPA_EVENT_ISSTA
  - FI_BUPA_EVENT_ISSTA

  **Action required:**
  - Initialize the function group’s global variables
  - Get control information from the BDT
    - Activity
    - Editing mode (save or transfer mode)
    - Indicator: Display initial screen
    - Selected roles/role groupings
    - Roles to be edited (from the selected roles/role groupings)
    - Etc.

  To do this, call up the BDT function module `BUS_PARAMETERS_ISSTA_GET` and note the information in your function group’s global variables.

  - Set default values for the fields in the initial screen
    - Read default values using the BDT function module `BUS_PARAMETERS_ISSTA_GET` (interface table T_FLDVL)
    - If default values have been set for a field, these should be transferred to the relevant screen fields
    - If no default value has been set for a field, the SET/GET parameters should be read where available

- **ISDAT (reading data)**

  The applications which own the tables (not the applications using the tables!) read them and note the data in their update memory as both the new and the old status. Some application tables however must be read earlier in the PAI function modules for the initial screen views in order to conduct the necessary field checks. These tables do not have to be read again for ISDAT.

  **Runtime:** Between initial screen and first data screen

  **Application area:** Applications which own the tables

  **Naming convention:** `<Application>_<Application object>_EVENT_ISDAT`

  (Customer: Function module name also has the prefix Y_ or Z_)

  **Example:**
  - BUP_BUPA_EVENT_ISDAT
  - FI_BUPA_EVENT_ISDAT

  **Action required:**
  - Determine data from other tables if this is necessary to read own tables. The communication modules can be used to read a table.
o Read own application tables.
  ▪ Read in global memory if the data for the current instance has already been noted in this LUW (transfer mode).
  ▪ If, in this LUW, the data for the current instance has not yet been transferred, it will be read from the database.
  o Note the data in current memory as the old and new status of current instance.
  o Draw temporary number (when creating with internal number assignment). In event DSAVC this number will be replaced by the final number.
  o Determine screen field values for fields of particular data types if the check is not to be carried out directly on the screen.

- **ISDTS (distributing data)**
  The applications using the table determine the contents of the table with the help of the function module for reading data. The data is noted as the old and new status in the current memory of the application using the table.
  Runtime: Between initial screen and first data screen.
  Application area: Applications using the tables.
  Naming convention: <Application>_<Application object>_EVENT_ISDST
  (customer: Function module name has the prefix Y_ or Z_).
  Action required:
    o Determine table data from the application which owns the table using the function module for reading data.
    o Note data in current memory as the old and new status.

- **XCHNG (data changed?)**
  If the user attempts to leave data maintenance, a query dialog box must appear if the data has been changed. Within this event, the BDT determines whether data has been changed.
  This event is only processed in activity Change. In the activity Create the query always appears, whereas in the activity Display no data can be changed and a query is therefore unnecessary.
  Runtime: When leaving data maintenance (activity Change only).
  Application area: All applications.
  Naming convention: <Application>_<Application object>_EVENT_XCHNG
  (Customer: Function module name has the prefix Y_ or Z_).
  Examples:
    o BUP_BUPA_EVENT_XCHNG
    o FI_BUPA_EVENT_XCHNG
  Action required:
    o Compare old and new status of the current memory.
    o Inform the BDT whether data has been changed.

- **DCHCK (checks before saving)**
  Each application can carry out consistency checks before saving. Possible messages appear in a dialog box. If an error message is triggered at this time, it is not possible to save the data until the error has been removed.
  Call time: When saving the data.
  Application area: All applications.
  Naming convention: <Application>_<Application object>_EVENT_DCHCK
  (Customer: Function module name also has the prefix Y_ or Z_)
  Action required: Carry out consistency checks and output possible messages via the Message Handler.

- **DSAVB (collecting data)**
The applications using the table transfer the new data status from their current memory to the application which owns the table. To do this, they call the table function module to collect the data.

**Runtime:** When saving the data.

**Application area:** Applications using the tables

**Naming convention:** `<Application>_ <Application object>_EVENT_DSAVB`  
(Customer: Function module name also has the prefix Y_ or Z_)

**Action required:**
- Transfer new data from the current memory to the application which owns the table. The function module defined for the table is called to collect the data.

- **DTAKE (transferring data to the LM)**

  In the BDT, data is saved using several steps. Data from several instances can be saved together. This is used both in transfer mode and for background maintenance. In this first step, the application which owns the table writes the data from its current memory to its global memory.

  **Runtime:** When saving the data.
  
  **Application area:** Applications which own the tables.
  
  **Naming convention:** `<Application>_ <Application object>_EVENT_DTAKE`  
  (Customer: Function module name has the prefix Y_ or Z_).

  **Example:** BUP_BUPA_EVENT_DTAKE

  **Action required:**
  - Write new data from the current memory to the global memory for new data.
  - Write old data from the current memory to the global memory for old data, if the data for this object instance has been noted for the first time in this LUW.

  **Note:** In the event DSAVE, the global memory for each instance is checked to see whether changes have been made and whether the data therefore needs to be written to the database. This means the old status of the global memory must correspond to the old status of the current memory as it was when the data was first noted in the LUW.

- **DSAVC (completing data)**

  The global memory is prepared for saving the data to the database. All actions for which errors may occur must be carried out here. However, within the event DSAVE, termination messages may only appear if the data is inconsistent or if there are program errors.

  **Runtime:** When saving the data.
  
  **Application area:** Applications which own the tables
  
  **Naming convention:** `<Application>_ <Application object>_EVENT_DSAVC`  
  (Customer: Function module name also has the prefix Y_ or Z_)

  **Example:** BUP_BUPA_EVENT_DSAVC

  **Action required:**
  - Draw object number (only when creating with internal number assignment).
  - Replace temporary number (see event ISDAT) with the number just drawn.

- **DSAVE (saving data to DB)**

  The application which owns the tables writes the data from the global memory to the database. In view of the improved runtime, in particular for the direct input, array operations should always be used here.

  **Runtime:** When saving the data.
  
  **Application area:** Applications which own the tables
  
  **Naming convention:** `<Application>_ <Application object>_EVENT_DSAVE`  
  (Customer: Function module name also has the prefix Y_ or Z_)

  **Example:** BUP_BUPA_EVENT_DSAVE

  **Action required:**
  - Write new data from the global memory to database. The BDT will state, using a parameter of the function module BUS_PARAMETERS_ISSTA_GET, whether this is to be done with or without an update task.
  - Write change documents using the old and the new status from the global memory.
  - Issue success message (only application which owns application object).
• **DLVE1 (initializing the current memory)**
  The current memory is initialized. When returning to the initial screen there is no (!) LEAVE TO TRANSACTION with the result that the current memory must be initialized at this point so that the next data maintenance will start correctly. The application which owns the application object will remove the lock.
  Runtime: When leaving data maintenance.
  Application area: All applications.
  Naming convention: `<Application>_<Application object>_EVENT_DLVE1`.
  (Customer: Function module name has the prefix Y_ or Z_).
  Example: BUP_BUPA_EVENT_DLVE1
  **Action required:**
  - Initialize current memory.
  - Remove lock for current instance (only application which owns application object).

• **DLVE2 (initializing the global memory)**
  Runtime: When leaving data maintenance.
  Description: The global memory is initialized.
  Application area: All applications.
  Naming convention: `<Application>_<Application object>_EVENT_DLVE2`.
  (Customer: Function module name has the prefix Y_ or Z_).
  Example: BUP_BUPA_EVENT_DLVE2
  **Action required:**
  - Initialize global memory.

**Screen Layout**

Screen layout is defined in control tables. Applications from downstream development stages can extend and reconfigure existing screens. The BDT supports the following entities in screen layout:

- **Field group**
- **View**
- **Section**
- **Screen**
Field groups:

Fields whose readiness for input was defined collectively are collected in a single field group. You create settings for each field group with in the Field Groups.

A field group may only contain fields from one application. Customers should neither change nor extend the definition of standard SAP field groups, as this would amount to a program modification. As an alternative, customers can define their own field groups to which they also assign their own views.

Menu path: Control <Object> → Screen layout → Field groups

Procedure: To define your own field group, proceed as follows:

- Define field group
  
  The name of a field group is language-dependent and appears when a field grouping is maintained. If fixed logical dependencies exist, a program can be used to change field groupings set up by customers.

  If Required entry is set for a field group, the BDT checks automatically whether the fields have entries before the data is saved. If this is not the case, an error message is displayed. A different kind of required field check is necessary for some field groups. For these field groups, the automatic check can be turned off and an individual check used instead. One example of this is bank details for a business partner. The individual check will only trigger an error message if no bank details were entered. A specific check for required fields does not take place on the field level.

- Assign Fields to the Field Group
  
  Enter the name of the screen fields that are to be assigned to the field group. Besides input fields, you also need to specify the field names of help texts, work fields, etc. For each field, specify whether it is ready for input or not.

Naming convention: Values from 1 to 1750 are permitted for field groups. While groups 750-1000 are reserved for development partners and groups 600-749 for customers, areas within SAP should be discussed with the development group responsible for the application object. The Business Partner development group is responsible for the business partner (application object BUPA) as well as business partner relationships (application object BUPR).

Views

One or more field groups constitute a view. All attributes that are displayed and checked together are grouped in one view. The fields of a view cannot be separated in screen layout since they are located on the same subscreen.

A view may only contain fields from one application. A downstream application may not extend a view; instead, it should create its own views for its attributes and assign its own subscreens to these views. The same applies to customers as extending a view amounts to a modification.

Menu path: Control <Object> → Screen layout → Views

Procedure: The following steps are necessary for defining a view:

- Create Subscreen
Use the Screen Painter to create a screen. But take note of the following when doing so:

- **Screen attributes:**
  
  Mark the screen type ‘subscreen’

- **Layout:**
  
  Generally, you won’t need to put a frame around your data. The BDT automatically inserts a frame around the fields of a section (see Sections).

  The field name of a pushbutton positioned on the subscreen should adhere to the following naming convention: PUSH_<Menu option>. If this is the case, the BDT’s field grouping automatically hides the pushbutton if the menu option is not active.

- **Flow logic:**
  
  Create a PBO module that calls function module BUS_PBO; call the PBO module from the PBO of each of your subscreens. If your subscreen contains a table control, transfer the data relating to the table control to the BDT using parameter C_TC1 when calling BUS_PBO.

  Create a PAI module that calls function module BUS_PAi. Call this PAI module from the PAI of each of your subscreens.

  Do not carry out any field checks within the flow logic, neither in a module nor in any of the sub-programs called from a module. Checks on a view should generally be carried out in a separate function module whose name is stored in the after-input field (see below).

  Text tables for displaying check texts should be read and other actions related to PBO should be carried out within the function module whose name is to be defined in the prior-to-output field (see next section).

- **Create Function Module Prior to Screen Call**

  This event is triggered by the BDT for all views of a screen if another event is used to reach this screen. Views with step-loop or table control usually need a function module at this point. The developer makes sure that when a screen is re-called,

  - the first page of entries is displayed
  - the entries are output in the correct sorting sequence
  - all entries are deselected

  Example: The view ‘Address overview’ exists in a business partner. A function module was defined for this view at the event Prior to Screen Call.

  Naming convention: <Application>_<Application object>_PBC_<View>

  (Customer: Function module name has the prefix Y_ or Z_).

- **Create Function Module Prior to Output**
This event is triggered for all views of the current screen before the subscreen is called in PBO. Text tables for outputting check texts can be read at this point. Example: The Industry field exists within a business partner. Examples of valid industries are stored in a Customizing table. If an industry ID is entered in the input field, the text table belonging to the Customizing table is selected in such a way that the name of the industry can be displayed behind the input field in the logon language.

Naming convention: &lt;Application&gt; &lt;Application object&gt; _PBO_ &lt;View&gt; (Customer: Function module name has the prefix Y_ or Z_).

- Create Function Module After Input

This event is triggered for all views of the current screen after all the subscreens involved are called in PAI. Carry out your field checks on the view at this point. You can find tips on the procedure for field checks in the sections Views.

Naming convention: &lt;Application&gt; &lt;Application object&gt; _PAI_ &lt;View&gt; (Customer: Function module name has the prefix Y_ or Z_).

- Define view

Description: The program name and screen number of the subscreen, as well as its name, must be specified here. The names of the function modules for the events listed above are also defined here.

Naming convention: &lt;Application&gt; &lt;n&gt;

The ID for a view should always have 6 places. It should start with the application ID followed by a set of numbers.

- Assign View → Field Groups

Define the field groups which belong to the view.

Other Checks on Views

The owner application can store the name of its check function module in the After input field within the definition of a view. If other applications add their own checks to this view, they can add the name of their check function module in the sub-node Other checks. These modules are called in PAI directly after the function module stored by the owner application. More information on data checks is given in the next section.

Naming convention: &lt;Application&gt; &lt;Application object&gt; _PAI_ &lt;View&gt;

(Customer: Function module name has the prefix Y_ or Z_).

Example: A customer request for business partners was to be able to check the last name of a person for the existence of at least one vowel. However, this check could not be implemented as a matter of course as there are names in some countries that have no vowels. The solution for this customer problem is to realize this kind of query using the check for an SAP standard view.

Procedure: In the check function module, the application gets the table content from the application that owns the table. To do this, the application uses the function module defined for the table in the Read data field. Then the check is carried out and, in some cases, a message displayed via the Message Handler (function module BUS_MESSAGE_STORE).

Note: In contrast to a check within event DCHCK, other checks for a view take place immediately after input. Event DCHCK is processed prior to saving.
Procedure for Field Checks

One of the goals for development with the BDT is a strict separation between interface and program logic. This ensures that all checks can be carried out even if you are using direct input and an external interface. Special rules apply to the following checks that are usually carried out by the screen:

- **Foreign Key Check**

  To avoid this check by the screen deselect the field ‘Foreign key’ in the field attributes. As a substitute, program the foreign key check in the function module After input for view.
  Alternatively, you can also use the foreign key check in dialog. However, you then have to program this check in function module After input for direct input (call up indicator for direct input whose value can be determined for event ISSTA).

- **Data Type-Dependent Check**

  These checks are carried out automatically by a screen if there are fields of a certain type on the screen. Examples include CURR (currency-dependent amount field), DATS (date field), INT* (integer) and NUMC (character field only with numbers). You can only get around the data type check by copying a CHAR field of the same length as the original field onto the screen instead of the original.
  The following procedure is one way of dealing with these fields:

  - In a structure, create a work field belonging to data type CHAR whose length corresponds to the output length of the database field.
  - Include this work field in the subscreen instead of the database field.
  - Call one of the BDT service function modules for initialization at event ISDAT. These function modules determine the formatted value for the screen field from the value of the database field. The following function modules are currently available:
    - BUS_DATEFIELD_START for date fields
    - BUS_NUMBERFIELD_START for numbers and currency-dependent amounts.

  - Call the DBT service function module for input check/conversion in the PAI function module for view. The BDT provides the following function modules:
    - BUS_DATEFIELD_PAI for date fields
    - BUS_NUMBERFIELD_PAI for numbers and currency-dependent amounts.

**Example:** The following examples can be found within function group FBU0 (application FI for application object BUPA).

- Field KNB1-DATLZ (date of last interest calculation)
  - Within function module FI_BUPA_EVENT_ISDAT, function module BUS_DATEFIELD_START is called in the form of KNB1_ISDAT.
  - Within function module FI_BUPA_PAI_FI2100, function module BUS_DATEFIELD_PAI is called in the form of DATLZ_CHECK.

- Field KNB1-WEBTR (exchange limit in local currency)
  - Within function module FI_BUPA_EVENT_ISDAT, function module BUS_NUMBERFIELD_START is called in the form of KNB1_ISDAT.
Within function module FI_BUPA_PAI_FI2410, function module BUS_NUMBERFIELD_PAI is called in the form of WEBTR_CHECK.

As with foreign key checks, you also have the option with data types of letting the screen perform the check in dialog. However, you still have to use the procedure outlined above for direct input.

- **Message Output**

Never display messages directly using the message statement. This leads to problems with direct input and external interfaces. Call function module BUS_MESSAGE_STORE instead. You can enter the following information in this module:

  - Message type
  - Message class
  - Message number
  - Message parameters
  - Name of the field on which the cursor is placed
  - Names of fields affected by the message (highlighted)

### Sections

One or more views are grouped together as a section. The BDT automatically puts a frame around each section. The only exception to this rule is the first section of a screen in which the header data appears. According to SAP ergonomic guidelines, a frame should not be put around this data. In addition to the description, you also define a language-dependent title for the section, which is displayed in a dialog in the upper left-hand corner of the frame.

**Menu path:** Control <Object> → Screen layout → Sections

**Procedure:** The following steps are necessary for defining a section:

- **Define Section**

  For each section, fill in **Description** and **Title**. Both are language-dependent. In dialog, the title appears in the upper left-hand corner of the frame.
  
  Naming convention: <Application><n>
  
  The ID for a view should always have 6 places. It should start with the application ID followed by a set of numbers.

- **Assign Section → Views**

  Define the views here which belong to the section. The sequence of views within a section is defined by the position number.

  Naming convention: The position number has 7 places. Each development stage has one place of its own. The places from the left up to your own place are freely assignable. The application uses them to determine the position of the view within a section in relation to the views of the upstream development stages. The numbers to the right of your own place must be '0'.

  The following places are reserved for the development stages:

  - Application basis 1+2
  - Standard applications 3
  - Industry applications 4
  - Development partners 5
Screens

The screen represents the largest unit in screen layout. One or more sections are grouped together as a screen. In addition to screens created with BDT, you can integrate screens that were not created with BDT by using the *External screens* selection.

**Menu path:** *Control* <Object> → *Screen layout* → *Screens*

**Procedure:** The following steps are necessary for defining a screen:

- **Define Screen**

  In addition to the description, you also specify the screen-dependent part of the title as an additional language-dependent text. This is included in the determination of the complete screen title. You can also decide whether the screen is to appear as a full screen or a dialog box (popup).

  Screens not configured using the BDT can also be integrated into the process. To do this, mark the *'External screen'* indicator and enter the name of the function module for calling this screen. This module is called automatically by the BDT as soon as you start to navigate in the external screen. One example of this is the BP relationships of a business partner, whose overview was integrated as an external screen.

  Naming convention: <Application><n>
  The ID for a screen should always have 6 places. It should start with the application ID followed by a set of numbers.

- **Assign Screen → Sections**

  Define the sections here which belong to a screen. The sequence of sections within a screen is defined by the position number.

  Naming convention: The naming convention for the position number corresponds to that for the position number in the section → view assignment (see section).

**Customizing Screen Layout/Screen Sequence with Drag & Drop**

**Description:** Using the Visual Configuration Tool (VCT), customers can change standard SAP screens in the main screen sequence as well as the sequence itself in Customizing by drag&drop. These changes, like all other Customizing activities, connected to transports. Changes made by customers are fully maintained - that is, they are not overwritten by SAP in an upgrade.

The following functions are available for customers:

- **Screen Layout Mode**
  - Move view, also to other screens
  - Move section, also to other screens
  - Change the title of a section (double-click in section outside view)
  - Change the title of a section (double-click in section outside view)
  - Delete view (this flows into the list of unused views)
  - Delete section (the assigned views flow into the list of unused views)
  - Insert view (this flows into the list of unused views)
  - Insert blank section
• **Screen Sequence Mode**
  - Move screen anywhere
  - Change the title of a screen (double-click in screen)
  - Delete screen (the views flow into the list of unused views)
  - Insert blank screen

• **Cross-component functions**
  - Retrieve SAP settings (changes made by customers are lost)

If an application object uses divisibility, you can set screen layout and screen sequence for each object part. If divisibility is not used, only one screen configuration can be created by the customer.

**Action required:** If you want to use configuration for your application object, you have to define just one setting transaction for setting activity 0104.

**Future plans:** Functionality in VCT is to be extended progressively in Releases to come. The next steps include configuration of all screen sequences and/or the additional screens, integration of field grouping as well as the use of VCT by developers.

**Screen sequences**

The screen sequence defines the order in which screens appear. You can go through the sequence with the *Next screen* function. Tab strips take you to screens directly.

Up until now, you could only configure screen sequence with the BDT starting with the initial screen. Now you can use menu options to branch into other screen sequences.

**Menu path:** Control <Object> → <Screen sequence>

**Defining Screen Sequences**

You can define any number of screen sequences to be distributed later over various screen sequence categories.

**Naming convention:** <Application><n>

The ID for a view should always have 6 places. It should start with the application ID followed by a set of numbers.

**Assigning Screen Sequences → Screens**

The sequence of screens is defined by the position number. Screen sequences in the main screen sequence category (see Screen Sequence Categories) include the data screens and the initial screen, which always has to be at the beginning of a screen sequence. All other screen sequences consist exclusively of data screens.

**Naming convention:** The naming convention for the position number corresponds to that for the position number in Assign Section → View (see Sections).

**Note:** Screens on which fields are hidden (by the object part → view assignment or because of a customer field grouping) are automatically skipped over by the BDT in dialog maintenance.
Screen sequence categories

A screen sequence category is a logical place within a dialog flow in which a screen sequence can be processed. The main screen sequence category, whose identification is always SPACE (six blanks), plays a special role. The BDT always starts the dialog flow with the first screen of a screen sequence which is assigned to this category. This is usually the initial screen.

Navigation to other screen sequence categories usually takes place with a GUI function. When defining this function, the screen sequence category is also defined. After the function is selected, the event FCODE is processed and the BDT calls the screen sequence of the defined screen sequence category. If the screen sequence category is not unambiguously assigned to a menu option and for that reason cannot be stored in the definition of this function (for instance if the screen sequence category is decided at run time based on the cursor position), call function module BUS_SCREEN_CALL in event FCODE. The screen sequence category is to be transferred with parameter I_VARTP.

Assigning Screen Sequence Category → Screen Sequences

All valid screen sequences are defined for each screen sequence category, of which one must be indicated as standard.

Screen Sequences for Main Screen Sequence Category: You can assign an unlimited number of screen sequences. The BDT uses the following rules to decide which of the assigned screen sequences will be put into effect:

- Object uses divisibility
  - If just one object part is maintained, the screen sequence defined in it is used. If no screen sequence is defined for the object part, the standard sequence is used.
  - If just one object part grouping is maintained, the screen sequence defined in it is used. If no object part grouping is defined for the object part, the standard sequence is used.
  - If several object parts or object part groupings are maintained and the same screen sequence assigned to all of them, then this is the sequence that will be used.
  - In all other cases, the standard screen sequence is used.

- Object does not use divisibility
  - The standard sequence is always used.

Standard Screen Sequence for Main Screen Sequence Category: Make sure that this sequence includes all views relevant to the main screen sequence. The reason for this is that the BDT sequence is always used in those cases when no sequence variant has been unambiguously assigned (functions as a catch-all).

Number of Screen Sequences for Main Screen Sequence Category: In many cases, one sequence is enough even for application objects that use divisibility. The BDT automatically skips over screens that contain no input fields for the current dialog. You only need to create more sequences for special object parts if:

- several nearly empty screens would appear due to hidden fields (in an additional sequence, the relevant fields can be grouped together on one screen)
- the sequence of the screens is to be changed
Screen Sequences for Other Categories At the moment, only one sequence is to be assigned, which is then marked as standard. The BDT always uses the standard screen sequence for a screen sequence category. In a future Release, it will be possible for the application to determine the screen sequences based on object parts for all screen sequence types.
Search Helps

From release 4.0A, matchcodes will be replaced by new search helps. Up to now, matchcode IDs were limited to the tables and fields defined in the matchcode object. This restriction will not apply with the new search helps. However, adding an elementary search help to a search help defined by a different application will still be a modification in Release 4.5. For this reason, the BDT will provide you with the option starting in Release 4.5 of linking elementary search helps belonging to various search helps within a dialog box.

Menu path: Control <Object> → Search helps

Future plans: From Release 4.6A, Basis will offer an APPEND concept for search helps. As with tables and structures, downstream applications, or customers themselves, can expand the collective search help to include further elementary search helps without modifying the original collective search help. From this release, the BDT table for search helps will no longer be necessary. The search help can be defined

- in the table field or
- in the structure or table or
- in the screen field directly.

The following two chapters describe the procedure up to Release 4.5.

Creating Search Helps for a Field

Using the following steps you can establish the search help for a field, which can later be expanded by other applications without modification.

- Create search help and elementary search help in the Data Dictionary
- Define elementary search helps in the BDT table for search helps
  menu path: Control <Object> → Search helps
- Call up the BDT function module BUS_MCODE for event POV on the screen
- Call up the BDT function module BUS_PAI for event PAI on the screen - this will also support the search help short entry with "=

For all fields wishing to use this search help, the above BDT function modules should be called up in POV and PAI. Only this will ensure that any elementary search helps, added subsequently by other applications (see ‘Expanding Search Helps for a Field’), are available everywhere.

Expanding Search Helps for a Field

Downstream applications or development partners and customers always require additional elementary search helps for a field in another application. The BDT offers this as a fully maintained feature - in other words, without the need for modification.

- Create a collective search help in the data dictionary, giving it its own name, and add elementary search helps.

Define elementary search helps in the BDT table for search helps
(menu path: Control <Object> → Search helps)

Tables

Menu path: Control <Object> → Tables
Description: Each application table you want to maintain must be entered here. The application responsible for the table (determined by the table’s development class) must write two function modules for communicating with other applications. These modules allow applications that participate in tables as well as the BDT to exchange table contents with the responsible application during dialog.

- **Read data**

  This function module is used by other applications as well as the BDT to determine the current content of the table at any time during data maintenance. Refer to the description for event ISDST for more details on how to use it.

  Naming convention: `<application>_<application>_<table name>_GET`

  (Customer: Function module name also has the prefix Y_ or Z_)

  Example:

  - BUP_BUPA_BUT000_GET
  - BUP_BUPA_BUT0BK_GET

- **Collect data**

  This function module allows an application that participates in tables to transfer the values of fields it attached to the application that owns the table. Refer to the description for event DSAVB for more details on how to use it. The function module has to be developed in such a way that only the fields of the application participating in tables in the current memory of the application that owns the table are overwritten. The name of the INCLUDE/APPEND structure in the interface is also transferred.

  Naming convention: `<application>_<application object>_<table name>_COLLECT`

  (Customer: Function module name also has the prefix Y_ or Z_)

  Example: BUP_BUPA_BUT000_COLLECT

**Sample Example:**

Using BDT development to creating a new Tab on the grants management master data transaction (GMGRANT)

- Execute grants management BDT development transaction (GRANT_CUSTOMIZE)

- Grant Master BDT Customizing-> Screen layout and select ‘Field groups’(GMS2)
• Click on 'New Entries'
• Enter details of new field group. For this example it is going to be a copy of the responsibilities tab, which uses a table control.
• Go back to main field group screen, select your new entry and double click on the 'Field Group-> Fields' folder option from the left column. Now enter your fields, for this example I have copied fields from responsibilities group.
Return to main BDT customizing screen and select 'Views'(GMS3). Then click the 'New Entries' button.
Enter view information, once again copied from the responsibilities view, but calling custom created versions of program, screen and function modules, which are essentially copies of the SAP originals functionality.
• Go back to main Views screen, select your new entry and double click on the Views-> Field Groups folder option from the left column. Now enter the new field group you created previously.
• Return to main BDT customizing screen and select ‘Sections’ (GMS4). Then click the 'New Entries' button
- Input new entry and give it a description and title
• Go back to main Sections screen, select your new entry and double click on the Section-> Views folder option from the left column. Now enter the new view you created previously.

• Return to main BDT customizing screen and select ‘Screens’(GMS5). Then click the 'New Entries' button
• Enter details for your new screen
• Go back to main Screen screen, select your new entry and double click on the Screens-> Sections option from the left column. Now enter the new section you created previously.
• Return to main BDT customizing screen and select ‘Screen Sequence’ (GMS6). Then select the screen sequence entry and double click on the Screen Sequence -> Screens folder option from the left column.
• Add your screen to the sequence by clicking on the ‘New Entries’ button
Add your new screen to the screen sequence.

Execute transaction BUSP (click on option from main grant customizing menu), new screen will now appear as a tab on grant master transaction ‘GMGRANT’
Disclaimer and Liability Notice

This document may discuss sample coding or other information that does not include SAP official interfaces and therefore is not supported by SAP. Changes made based on this information are not supported and can be overwritten during an upgrade.

SAP will not be held liable for any damages caused by using or misusing the information, code or methods suggested in this document, and anyone using these methods does so at his/her own risk.

SAP offers no guarantees and assumes no responsibility or liability of any type with respect to the content of this technical article or code sample, including any liability resulting from incompatibility between the content within this document and the materials and services offered by SAP. You agree that you will not hold, or seek to hold, SAP responsible or liable with respect to the content of this document.