Business Data Toolset (BDT)
Developer’s Manual
1 Introduction .......................................................................................................... 4
2 Introduction to the BDT ....................................................................................... 5
  2.1 What is the BDT? ........................................................................................... 5
  2.2 History .......................................................................................................... 5
  2.3 Current Status ............................................................................................... 6
    2.3.1 Users ......................................................................................................... 6
    2.3.2 Advantages ............................................................................................. 7
    2.3.3 Functionality .......................................................................................... 8
  2.4 Future Enhancements ................................................................................... 11
    2.4.1 For Release 4.6 ....................................................................................... 11
    2.4.2 After Release 4.6 ................................................................................... 12
  2.5 Problem Messages/Development Requests .................................................. 13
  2.6 Task Menu BUPT .......................................................................................... 13
3 Function Overview ............................................................................................. 14
  3.1 Basic Settings ................................................................................................. 14
    3.1.1 Differentiation Types .............................................................................. 14
    3.1.2 Application Objects ................................................................................ 15
    3.1.3 Applications ............................................................................................ 18
  3.2 Dialog ............................................................................................................ 18
    3.2.1 Activities ................................................................................................. 18
    3.2.2 Screen Layout ......................................................................................... 19
    3.2.3 Screen Sequences .................................................................................... 26
    3.2.4 Tables ...................................................................................................... 28
    3.2.5 Program Logic ........................................................................................ 30
    3.2.6 Screen Title ............................................................................................. 35
    3.2.7 Menus ...................................................................................................... 36
    3.2.8 Field Groupings ...................................................................................... 39
    3.2.9 Assigning Screen Fields \( \rightarrow \) Database Fields ......................................... 42
    3.2.10 Search Helps .......................................................................................... 43
    3.2.11 Authorizations ....................................................................................... 44
    3.2.12 Divisibility ............................................................................................ 46
    3.2.13 Application Transactions ....................................................................... 48
    3.2.14 Transfer Mode ....................................................................................... 49
  3.3 Change Document Lists ................................................................................ 49
    3.3.1 Events CHGD1 to CHGD4 ..................................................................... 50
    3.3.2 Calling Lists ............................................................................................ 52
  3.4 Data Maintenance Without Dialog ............................................................... 53
    3.4.1 Direct Input ............................................................................................. 53
    3.4.2 Function Modules ................................................................................... 57
  3.5 External Interfaces ........................................................................................ 58
    3.5.1 Process .................................................................................................... 58
    3.5.2 BDT Function Modules .......................................................................... 59
4 Roadmap for Implementation ............................................................................... 62
4.1 Additional Check for Existing View ............................................................ 62
4.2 New Table Fields and New Tables ............................................................... 62
  4.2.1 New Table Field Using Owning Application ......................................... 62
  4.2.2 New Table Field Using Participating Application ................................. 63
  4.2.3 New Table ............................................................................................... 63
4.3 Extend Screen Layout and Screen Sequence ............................................... 64
  4.3.1 Include Field in Existing Field Group .................................................... 64
  4.3.2 Include Field in Existing View ............................................................... 64
  4.3.3 Include New View in Existing Section .................................................. 64
  4.3.4 Include New Section in Existing Screen .............................................. 65
  4.3.5 New Screen in Existing Screen Sequence ............................................ 65
  4.3.6 New Screen Sequence as Main Screen Sequence ............................... 66
  4.3.7 New Additional Screen and/or New Additional Sequence ................. 66
4.4 New Object Part ............................................................................................ 67
4.5 New Field Grouping Criterion ..................................................................... 67
4.6 New Application Object ............................................................................... 67
5 Appendix ............................................................................................................. 68
  5.1 Terminology Definitions .............................................................................. 68
  5.2 Abbreviations ............................................................................................... 69
  5.3 Naming Conventions .................................................................................... 69
1 Introduction

This manual describes the functions in the *Business Data Toolset* (BDT) as of January 1999. Development for Release 4.6A in the standard R/3 system has already begun at this point. The following IBU releases have the same development status:

- IBU Banking Release 4.01B
- IBU Insurance Release 2.1A
- IBU Telecommunications Release 1.2
- IBU Utilities Release 1.2

This manual is divided into five main sections.

After this introduction, **section 2** provides a brief overview of the BDT’s origins, current features and enhancements planned for the future. **Section 3** describes current functionality in detail. In writing this manual, the authors have assumed thorough knowledge of dialog programming and the related tools. **Section 4** is intended as a rough guide to problem-solving: it lists any necessary steps and then makes reference to sections that provide more detailed descriptions. Explanations of special terms and abbreviations as well as an overview of naming conventions (**section 5**) comprise the last section of the manual.

Depending on your interests, different sections will be more or less relevant for you:

- For a **brief overview** of the functions in the BDT, see section 2.
- **Developers and consultants** who plan to use the BDT for implementation but are unfamiliar with it need to read the **entire** manual.
- **Developers and consultants with BDT experience** may only need to consult **section 4** to solve actual problems.
2 Introduction to the BDT

2.1 What is the BDT?
The BDT (Business Data Toolset) is a central control tool for maintaining master
data and simple transaction data. In addition to dialog maintenance, it also
supports maintenance with direct input and/or function modules.
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 (control tables and events). The
applications themselves introduce application-specific enhancements, such as
writing and reading application tables.

Note: The BDT is used at SAP for maintaining several application objects (see
chapter 2.3.1). Development partners and customers can also extend these
application objects via the BDT interfaces. However, objects belonging to
development partners and customers may not be mapped using the BDT, as the
required control has not yet been released.

2.2 History
The Business Data Toolset originated in the Central Business Partner project. The
following demands on the technical aspect of data entry played an important role
in the development of the BDT:

- **Extensibility**
  Although the *Business Partner* project group had realized the central attributes
  of a business partner, (such as name components, addresses and bank details)
  there were other specific attributes in many of the remaining applications.
  Development partners and customers needed a facility for incorporating their
  own attributes into maintenance. In master data for accounts receivable and
  accounts payable, you had to make modifications to do this.
  Because it is impossible to collect and implement all these different attributes
  in one project group, maintenance for downstream enhancements had to be
  extensible without the need for modifications.

- **Configurability**
  Because mid-sized customers in particular tend to suppress most of the
  standard SAP data fields, dialog maintenance becomes tedious when you still
  have to go through screen after screen on which only one or two fields are
  relevant. Switching screens often slows down data entry considerably.
  As a result, it was decided to make screens configurable in order for customers
  to both tailor entry screens to their individual needs and keep the number of
  screens to a minimum.

- **Divisibility**
  If you were to count up all the attributes in the SAP system that are relevant
for a business partner, you would have several hundred fields. Since it is impossible to include all these attributes in each type of maintenance, the maintenance itself must be divisible into parts wherein only those attributes are visible which are relevant in the current business context. These parts are called *roles* in Business Partner.

The necessary technology was first developed in a common program with application data for Business Partner. However, it soon became apparent that the second part of this project - i.e., the business partner relationships - were placing the same technical demands on data maintenance. The requirements listed above were also applicable to other business objects. As SAP restructured with a new industry orientation, extensibility assumed a greater importance for development. Many of the IBUs wanted to extend or enhance application objects from the standard system. As a consequence, the *Business Partner* project group decided to separate the technical part from the application data and then make this technology available to other application objects. This technical part, which was called *BP control or master data control* for a long time, is now known as the *Business Data Toolset*, or *BDT*.

### 2.3 Current Status

#### 2.3.1 Users

The first user of the BDT is SAP Business Partner (see section 2.1). Business Partner’s purpose is to integrate all partner solutions that currently exist in the SAP system. In the first step, the following new applications build on this development:

- Contract Accounts Receivable and Payable
- IBU Banking
- IBU Insurance
- IBU Utilities
- IBU Telecommunications

The integration of the Treasury business partner is currently in process and should be complete by R/3 Release 5.0.

In the meantime, other application objects have already taken advantage of the BDT. The following application objects are currently being realized or developed in conjunction with the BDT:

- **Central Business Partner**
  - Partner maintenance
  - Relationship maintenance
- **Contract Accounts Receivable and Payable**
  - Contract account
2.3.2 Advantages

Similar technical demands are often placed on the development of application objects. By using the BDT, an application object can provide functions without having to realize them itself. The essential advantages of using the BDT are:

- **Extensibility**
  You can extend various dialog parts **without the need for modifications** using downstream applications either within SAP or through development partners or customers. This applies to
  - Screen layout
  - Screen sequence
  - Program logic
  - Menu
  - Search help
  - Field grouping
  - Authorization checks

- Other areas, such as data maintenance without dialog or change document lists, are also extensible.

- Extensions, or enhancements, can be created over multiple levels. As a rule, application development in R/3 has a maximum of five levels:
  - Application basis
  - Standard applications
  - Industry applications
  - Development partners
  - Customers

- **Configurability**
Screen layout and sequence can be configured by application developers and/or customers. While developers use BDT control tables to modify screens, customers can take advantage of a configuration tool developed using Visual Basic for changing standard SAP screen layout and sequence with the drag&drop method.

- **Divisibility**
  The maintenance of large objects can be broken down into smaller parts. You use control tables to define which attributes can be maintained in an object part. The term *object part* can be replaced by a more suitable term for any given application object. The BDT supports two types of divisibility:
  - Each object instance can take on multiple parts (example: roles with a business partner)
  - Each object instance can take on just one part (example: account type with a bank account)

- **Delinking**
  Each application develops within its own function group, thereby delinking individual applications.

- **Use of Other Interfaces**
  Interface and program logic are separate in the BDT. The program logic for the applications is contained fully in function modules that are called by the BDT at predefined times. As a result of both of these factors taken together, the R/3 interface of the BDT can be replaced by a different interface.

- **Faster Development**
  Because the BDT takes control of dialog processes, the applications limit themselves to realizing business functions. The BDT also provides services in which the applications can be included. These factors reduce considerably the time needed to develop applications.

- **Uniformity**
  In all application objects that use the BDT, online navigation takes place using the BDT and is therefore identical. Using the generic object services also contributes to a certain uniformity.

### 2.3.3 Functionality

The following provides you with an overview of functions that have already been realized.

- **Dialog Maintenance**
  - Existing tables can be extended by downstream applications using the APPEND structure technique from Basis. These new table fields as well as completely new tables can be integrated seamlessly into a dialog by SAP applications, development partners or customers.
  - **Screen layout and sequence** can be extended and configured using control
tables (without the need for modification). Customers can adapt standard SAP screens to their needs with drag&drop within customizing. Using the Visual Configuration Tool (VCT), customers can change:
- screen layout - or also group several screens together
- screen sequence
- screen titles
- frame titles

- Program logic can be extended by SAP applications, development partners or customers using event function modules. In this way, each application can:
  - read its tables
  - check its fields
  - carry out additional checks for fields in other applications
  - save its tables

- The **screen title** is composed by the BDT in accordance with SAP ergonomic guidelines. Its elements are:
  - the title of the application object (*business partner*)
  - the activity (*change*)
  - the title of the current screen (*address*)

  - In this example, the BDT created the screen title *Change Business Partner: Address*.
  - You can change the title created by the BDT with event DTITL.

- The **menu** is defined by the application that owns the application object. The central menu options, such as Cancel, Exit, Save and Back, provided by the BDT are part of the menu. You can use control tables to define when a menu option is to be active depending on the:
  - maintenance mode (save or transfer mode)
  - activity (create, change, display)
  - views on the current screen

  - Example: The menu option *Delete bank details* is only to be active in the *Create* or *Change* activity and when the view of bank details is on the current screen.

- **Field groupings** can be made using criteria of your choosing. The BDT supports the application when creating a maintenance transaction for one criterion and links settings for various criteria to the runtime using predefined rules.

- Using a control table, applications can add any number of other elementary search help functions to fields related to **search help**. Starting in Release 4.6A, this service will be provided by Basis in the form of APPEND search help functions.

- You can include notes easily on a screen and, like any other dialog part, place them wherever you like.

- **Authorization checks** can be carried out between the initial screen and the first data screen as well as prior to saving. The BDT provides some of the
recurring authorization checks that can be used by application objects.

- **Authorization for field groups**
  Example: Only user A may maintain names and addresses of business partners, while all users can maintain any remaining fields.

- **Authorization for field values of any fields**
  Example: Authorizations for a business partner are to be granted using the ‘Last name’ field
    - User A may only maintain those business partners whose last names begin with A-K
    - User B may only maintain those business partners whose last names begin with L-Z
  - Every application can also carry out any other authorization checks.

- **Change documents** are written by each application when saving data itself; the BDT provides evaluations. The following evaluation types are available:
  - Field changes (display changes to a field of an instance)
  - Account changes (display changes to all fields of an instance)
  - Display changes to multiple/all instances

- **Transfer mode**: The maintenance dialog is called from the maintenance of another object. The data is saved together with the calling object.
  Example: Maintaining a contract requires you specify both parties to the contract - two business partners. You should be able to create and/or change both business partners from contract maintenance. The contract data and the business partner involved are to be saved together. In order to do this, business partner maintenance must be called from the contract in transfer mode. When you exit business partner maintenance, the data entered is flagged but not yet saved in the database. Once the contract has been saved, the flagged business partner data is written to the database.

- **External interfaces** can be realized for application objects that were developed with the BDT. In this case, the external maintenance transaction only takes over the structuring of the interface and forwards the field contents entered on to the BDT. BDT function modules are called to carry out program logic such as reading, checking and saving data. They trigger the events that call the event functions modules in the applications.

- **Maintenance Without Dialog**
  - **Direct Input (DI)**
    Using the DI tools developed in EIS, data is read from a file and transferred to the BDT. The BDT then forwards that data on to the applications within events DINP1 (header data) and DINP2 (data). Finally, the same events are processed in this type of maintenance transaction as in a dialog. Most of the program logic developed by the applications can be reused.

- **Maintenance Using Function Modules**
  In contrast to DI, the data in this case is transferred in the interface of a
function module instead of being read from a file. Once the data has been transferred to the BDT, the process is the same as that for DI.

### 2.4 Future Enhancements

#### 2.4.1 For Release 4.6

Various enhancements to the BDT are planned for Release 4.6. These will be partially replicated at a later time in the development systems of those IBUs that use the BDT but do not develop in Release 4.6. Any questions you might have should be directed to the Business Partner development group. The most important developments in Release 4.6 are:

- **Time Dependency**
  
  There are numerous ways of creating time dependency, including planned change documents and extending a table key by one or more date fields. It is left up to applications that use the BDT to decide whether and what kind of time dependency they want to use.

  The BDT supports the use of planned change documents with two service function modules, which are called by the applications at certain times.

- **Calculating Data Status in the Past or Future**
  
  A BDT function module uses the database status which contains the current data as a basis for calculating the status at any date in the past or future. As a part of this procedure, the BDT reads the planned (future changes) or actual (changes in the past) change documents.

- **Calculating Data Status for Time Intervals**
  
  Based on the database status, the BDT function module determines the various data statuses in the past or future and then displays the date intervals with the data status valid for each.

  - The application writes the planned change documents during the event that saves the data (DSAVE).

    - If you choose the option of extending the table key by one or more date fields, most of the work is left to the applications. The BDT only provides a function module that uses pre-existing date intervals as well as the current change date and the action to be carried out to determine the new date intervals.

- **Deletion Program**
  
  This program deletes test data prior to production startup. Data should only be archived in production operation (see below). The BDT takes control in the deletion program and uses two events to provide the applications the option to

  - check the deletability of objects

  - delete deletable objects from the view of all applications

- **Archiving**
  
  The ADK (Archive Development Kit) is used for archiving. As in the deletion program, the applications can use events to intervene in the archiving process.
Where-Used List
An employee may sometimes need an overview of all the transactions a company had or is having with a business partner. Ideally, you would be able to include all or only selected areas of a company in this kind of list. Because this service has also been requested for other application objects, the BDT has developed an infrastructure for it. Its various uses are displayed in a user-defined tree structure. Each application can add its own nodes to the tree. The application decides which form of data retention is required on an individual basis for each node.

Reading Data at Runtime
The application stores the name of the function module in the BDT with which data can be imported. The advantage of this method is that redundant data need not be retained.

Saving Data in Redundant Usage Tables
For each application object there is a usage table in which the applications can update their usage at the same time as the operative tables. At runtime, the where-used list extracts the data from this table. This type of data retention generally has a better runtime and should be used for usage from external systems.

There are also two options for display:

Configurable Tree
The BDT can display the usages within one of the configurable trees belonging to an application. Customers can create partial views of this display to see only the usages that they specify. One partial view can be stored as standard for each user group. You can navigate to the maintenance and/or display transaction of an application.

You can also use a separately written program to read the data using a BDT function module and output it in its own display.

Evaluations
The ABAP List Viewer (ALV) can be used to create flexible evaluations of the application data. The BDT takes over control based on the extensibility requested at this point. The applications have the option of integrating their own data into the evaluation within just a few events.

2.4.2 After Release 4.6

Default Values
You will be able to maintain an unlimited number of default variants so that they can be used as a reference when creating a new instance. You will be able to include default values from other sources using a separate event. By using a field grouping, you will be able to decide whether a default value can be overwritten by a user at a later time.

Screen Configuration by Developers with Drag&Drop
Like customers (see section 2.4.1), developers will also be able to configure their screens with VCT and drag&drop.

- **Generating Application Programs**
  Large parts of the function modules to be created by the applications will be developed according to the specifications of the BDT. The important thing is to complete the actions defined for an event within that event. The application code will be generated according to the specifications of the developer responsible. This code can be supplemented and/or replaced by the developer’s own programs, thereby accelerating overall development.

### 2.5 Problem Messages/Development Requests

Problem messages or development requests regarding the BDT should be assigned to the *CA-BP (SAP business partner)* subject area. When finding notes, use key words *BDT* and *CBP.*

### 2.6 Task Menu BUPT

You get to the menu with the transaction code of the same name. This menu fulfills several purposes at once:

- The centrally defined application calls for *Business Partner* can be found in the *Application* submenu. There is also a branching point into the menu for BP-BP relationships, which represent a separate application object. All calls relevant for an application will be included in their respective menu so that there is no need to call from the BUPT menu.

- Cross-application settings (see section 3.1) are located in the *General control* submenu. You can also store differentiation types in addition to defining application objects at this point.

- The *BP control* submenu contains the object-specific settings (see section 3.2) for the *Business Partner* application object. You have to create a separate task menu for each additional object which contains the calls of the settings specific to this object.

- The *Customizing* submenu contains the settings a customer can make in *Business Partner.* You will also find the same points in the IMG under *Cross-Application Settings* → *Central Business Partner.* Documentation for the individual points, which you need to set up your system with the IMG is located here as well.
3 Function Overview

The BDT’s functions are described in detail in this section. Menu paths refer to task menu BUPT explained in section 2.6. Task menus belonging to other application objects will generally look very similar.

3.1 Basic Settings

This section outlines the activities required before you can start any dialog programming.

3.1.1 Differentiation Types

**Menu path:** General control → Differentiation types

**Description:** Application data is generally dependent on the primary key of the application. For a business partner, the primary key is the partner number. However, there can also be other attributes that must be differentiated according to other criteria. If data for an instance is only ever entered as a way of describing it as a criterion, then it would be called a differentiation type. Differentiation types are usually entered on the initial screen. The existence of an instance is also updated in dependence on the differentiation type specified.

Configure the input screens so that:

- the value of the differentiation type is visible in the header data if the underlying data is dependent on it
- divergent views of differentiation types are not grouped together on any one screen

**Examples:** Differentiation types are usually organizational units like company codes or sales areas. For an FI customer, the company code has to be entered on the initial screen. A large part of the data is dependent on the company code. These fields are grouped on separate screens. Because the header data in these screens also contains the value of the company code, it is obvious to the user which data can be given different characteristics on an individual company code basis. When you save, the existence of the customer in the company code entered is updated. If you create the customer in an additional company code, you must re-enter the company code on the initial screen as well as any data dependent on the company code in the data screens.

In addition to organizational units, the business partner also has differentiation types like product type and currency in the Treasury area.

**Note:** The definition of one differentiation type is valid for all application objects. You decide which differentiation types are relevant for each application object (see section 3.1.2.2.).

**Use:** The relevant differentiation types are assigned to each application object (see section 3.1.2.2.). A differentiation type is stored for each view (see section 3.1.2.1.). When configuring the input screens, bear in mind that you should be
able to use them later to check that only views of one differentiation type are grouped on one screen.

**Activities:** Check to see whether differentiation types are relevant for an application object. If this is the case and the differentiation type you need does not exist, contact the Business Partner development group, which is responsible for the central assignment of differentiation types and their associated elements within SAP. To create differentiation types for individual customer purposes, follow the steps below:

**Define Differentiation Types**
Each characteristic used to differentiate data must be entered at this point.

- **Define Differentiation Type Elements**
  Differentiation types are made up of one or more elements. While a company code is made up of only one element - the company code itself -, a sales area is comprised of three elements: sales organization, distribution channel and division. Each differentiation element is normally represented by an input field on the interface.

- **Assign Differentiation Type → Differentiation Type Element**
  Allocate to the differentiation type its elements. You still have to make an assignment even if a differentiation type has only one element.

  **Note:** The application responsible for the differentiation type determines the elements to be assigned. If an application requires a differentiation type with additional elements, do not extend the existing differentiation type - create a new one instead.

**3.1.2 Application Objects**

**Description:** Each master data/transaction data object that uses BDT must be known. The maintenance of application objects is subdivided into the following points:

- Defining application objects
- Assigning application objects → differentiation types
- Settings transactions, task menu

**Menu path:** *General control → Application objects*

**3.1.2.1 Defining Application Objects**

In this step, you register the new application object with the BDT and decide whether you want

- to use divisibility
- tab strips for navigation to be visible
- screen selection to appear on the initial screen
• to use the central authorization checks provided by the BDT

**Examples:** Currently, there are about 15 application objects from IBU Banking, IBU Insurance, Contract Accounts Receivable and Payable and Real Estate Management. Some examples include:

- BUPA  Business partner
- BUPR  Business partner relationships
- BKKA  Bank account
- FICA  Contract account

**Naming convention:** The BDT has been released solely for the development of SAP application objects. It should not be used to develop customer application objects. To register a new application object, contact the *Business Partner* development group. The group keeps a central register that prevents different application objects being given the same name, which could cause problems.

### 3.1.2.2 Assigning Application Objects ➔ Differentiation Types

At this point, you define whether the differentiation types already maintained are relevant for an application object. If you need a differentiation type that does not exist, you have to create it (see section 3.1.1).

### 3.1.2.3 Setting Transactions

The settings described in section 3.2.ff are maintained separately depending on the application object. If, for instance, you define a screen for application object BUPA (business partner), it will only exist within that application object.

By using setting transactions, you can easily create transactions with which the BDT control tables for your application object can be maintained. Later you also create the task menu for your application object (see section 3.1.2.4) from these transactions as well.

The following table provides an overview of current setting activities as well as which activity

- is needed by an application object (required)
- can be used by an application object (optional)

<table>
<thead>
<tr>
<th>Activity</th>
<th>Description</th>
<th>Req’d</th>
<th>Opt</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>Applications</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0002</td>
<td>Field groups</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0003</td>
<td>Views</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0004</td>
<td>Sections</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0005</td>
<td>Screens</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0006</td>
<td>Screen sequences</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0007</td>
<td>Events</td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Activities: Maintain a setting transaction for each setting activity. For each activity, carry out the following steps:

- Create a report transaction and enter the report BUSVIEWS as the start parameters.
  Procedure: Go into the ABAP Workbench menu and choose Development → Other tools → Transactions. Enter the transaction code and choose Create. Now select the Report transaction option and on the screen that follows enter BUSVIEWS in the program field next to the transaction text.

- Enter the setting activity with your application object and assign the transaction code you created in the first step.

When the transaction is started, the maintenance view that is part of the control function or the view cluster for your application object is called automatically.

3.1.2.4 Task Menu
Create a separate task menu for your application object which contains the control transactions. When creating your menu, use the BP control sub-menu in task menu BUPT as a reference.

Note: You can maintain area menus by going into the ABAP Workbench from Development → Other tools → Area menus.
3.1.3 Applications

Each application that is active in the maintenance of an application object has to register itself at this point. They always develop within their own function group, where they create subscreens as well as their own function modules for BDT events.

This encapsulates the application and prepares it for delinking. Avoid grouping heterogeneous components in a single application. Any later delinking of these components that becomes necessary would only be possible with your own expenditure of time and effort - in other words, you would have to split the application and thus also the function group.

Menu path: Control <Object> → Applications

Naming convention: Name ranges Y* and Z* are reserved for customer applications, while development partners can use name range X*. SAP applications register their application with the application responsible for the application object.

The application ID plays an important role in the naming conventions for other BDT objects.

3.2 Dialog

Using the BDT ensures that the dialog created by an SAP application is extensible for other applications both inside and outside SAP. The procedure is changeable. To do this, you would use fully maintained interfaces. Modifications to development objects in other applications are not necessary.

3.2.1 Activities

For each application object, you can define any number of activities in which an object is to be processed. Each application developer can show or hide fields on an individual activity basis. The activity-dependent part of the screen title (see section 3.2.6) is stored within the activity based on language.

Each activity has to refer one of the activity types predefined by the BDT. The following activity types exist for the dialog:

- Create
- Change
- Display

Menu path: Control <Object> → Activities

Naming convention: The following activities - called main activities below - have to be defined for each application object:

- 01 Create → Activity type 01
- 02 Change → Activity type 02
03 Display \( \rightarrow \) Activity type 03

**Examples:** For a standing order in IBU Banking (BCA) there is the *Delete* activity in addition to the three main activities. However, only a deletion flag is set - the associated activity type is actually *Change*.

**Environment:** Direct Input also recognizes activity type *Modify* in addition to the three others listed above. At runtime, the application determines whether a record already exists with the transferred key. If this is the case, the *Change* activity is set. Otherwise, the activity would be *Create*.

**Procedure:** Carry out the following steps for each activity:
- Define the activities for your application object and assign to each an activity type.
- Once the screen layout and screen sequence have been created, hide those field groups that are unnecessary for an activity.

### 3.2.2 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 groups**
- **View**
- **Section**
- **Screen**

#### 3.2.2.1 Field Groups

Fields whose readiness for input was defined collectively are collected in a single field group. You create settings for each field group within the field grouping (see section 3.2.8).

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> \( \rightarrow \) Screen layout \( \rightarrow \) 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 1000 are permitted for field groups. While groups 750-1000 are reserved for development partners and groups 500-749 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).

### 3.2.2.2 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 3.2.2.3).
    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_PAIR*. 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 also gets 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**: `<Application>_<Application object>_PBO_<View>`
  (Customer: Function module name also gets 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 chapter
3.2.2.2.2.
Naming convention: <Application>_ <Application object>_PAI_<View>
(Customers: Function module name also gets 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: <Application> NNN
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: <Application> _ <Application object> _PAI_<View>
(Customers: Function module name also gets 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 FL_BUPA_EVENT_ISDAT, function module BUS_DATEFIELD_START is called in the form of KNB1_ISDAT.
    - Within function module FL_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 FL_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, as this creates
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)

### 3.2.2.3 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 → View**
  Define the views 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
- Customers (central) 6
- Customers (branch) 7

3.2.2.4 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 (see section 3.2.6). 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>NNN
  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 which belong to the 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 3.2.2.3).

3.2.2.5 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

- **General 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 (see section 3.1.2.3).

**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.

**3.2.3 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>*

### 3.2.3.2 Defining Screen Sequences

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

**Naming convention:** <Application>NNN

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

### 3.2.3.2 Assigning Screen Sequences → Screens

The sequence of screens is defined by the position number. Screen sequences in the main screen sequence category (see section 3.2.3.3) 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 the section → view assignment (see section 3.2.2.3).

**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.

### 3.2.3.3 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 menu option (see section 3.2.7). 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.

### 3.2.3.4 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.

### 3.2.4 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. For more details on how to use it, see the description for event ISDST (section 3.2.5).

  **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. For more details on how to use it, see the description for event DSAVB (section 3.2.5). 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>_<_table name>_COLLECT`

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

  **Example:** BUP_BUPA_BUT000_COLLECT
3.2.5 Program Logic

Within the dialog flow, events defined by the BDT were used, for which the applications can develop a 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.

Menu path: Control <Object> → Events

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.

  **User group**: All applications
**Naming convention:** `<Application>_ `<Application object>`_ EVENT_ISSTA

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

**Example:**
- 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 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.
  - 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.
  - Note the data in current memory as the old and new status of current instance.
  - Draw temporary number (when creating with internal number assignment). In event DSAVC this number will be replaced by the final number.
  - Determine screen field values for fields of particular data types if the check is not to be carried out directly on the screen (see section 3.2.2.2).

- **ISDST (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 also has the prefix Y_ or Z_).

- **Action required:**
  - Determine table data from the application which owns the table using the function module for reading data.
  - 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 also has the prefix Y_ or Z_).

Example:
- BUP_BUPA_EVENT_XCHNG
- FI_BUPA_EVENT_XCHNG

- **Action required:**
  - Compare old and new status of the current memory.
  - Inform the BDT whether data has been changed.

- **DSA VB (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 also 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 table 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 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 also 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 also has the prefix Y_ or Z_).

**Example:** BUP_BUPA_EVENT_DLVE2

**Action required:**
- Initialize global memory.

### 3.2.6 Screen Title

#### 3.2.6.1 Determining Standard Titles

**Description:** The standard title is composed by the BDT in accordance with the SAP ergonomics guidelines. It comprises three parts:

- **Object title**
  If the application object uses the divisibility of the BDT and if exactly one object part is maintained, then the title of this object part forms the object title. However, if exactly one object part grouping is maintained, its title will be the object title. In all other cases (maintaining more than one object part/object part grouping or if the application object does not use the BDT’s divisibility) the title defined for the application object will be the object title.

- **Activity name**
  The name of the activity forms the second part of the screen title. It is separated from the other two parts by a colon.

- **Screen title**
  The title fixed when defining the screen forms the third part of the title.

**Example:**
- Change contract partner: Address
- Display business partner: Payment transactions

#### 3.2.6.2 Event DTITL (Changing Screen Titles)

The standard title, generated automatically by the BDT, can still be changed by an application at runtime. This is done using event DTITL, which is processed for each screen’s PBO. Within this event, the application can:
Change any of the title’s three parts.
- Change the title as a whole.

**Application area:** All applications

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

**Interface:**

- `C_TITLE_TOTAL` Total title (Reference: RSEU1-TIT_TEXT)
- `C_TITLE_OBJAP` Title, application object-dependent part (Reference: TBZ1T-OTITL)
- `C_TITLE_ACTVT` Title, activity-dependent part (Reference: TBZ0N-TXT30)
- `C_TITLE_SCREEN` Title, screen-dependent part (Reference: TBZ3B-DTITL)
- `E_XTOTAL` Indicator: Use total title (Reference: BOOLE-BOOLE)
- `E_XCHANGED` Indicator: Change made to title (Reference: BOOLE-BOOLE)

**Example:** no example available as event rarely required

**Action required:**
- Change the standard title.

**Note:** With this event, you can only change the title within the title bar of a screen, meaning that it is not possible to change the text within the screen selection in the initial screen. Each application determines the screen-dependent part of the title for its screens when the screens are defined. (This screen-dependent part of the title is identical to the text in the screen selection).

If customers want to change the title of a screen supplied by SAP, they can do so using the tool for changing the screen layout/screen sequence (see section 3.2.2.5). This change then affects the screen title, screen selection and the tab strip text.

### 3.2.7 Menus

#### 3.2.7.1 Creating Menus

**Description:** The application which owns the application object creates a menu for the application object. The application object menu `BUPA` (Business Partner) may be used as a template. The statuses of this menu (program `SAPLBUDO`) should be copied into the function group of the application which owns the application object, and adjusted there. The OK codes for the control functions (function code `BUS*`) may not be changed. You generally need the following statuses:

- `INITDATF` Initial screen of main screen sequence as full screen
- `INITDATP` Initial screen of main screen sequence as dialog box
- `STNDDATF` Data screen of a screen sequence as full screen
Important note: The central control functions (function code BUS*) are defined by the BDT. You should include these in your menu. The function codes may not be changed as these are used as a basis for programming by the BDT. The following central functions are defined by the BDT.

<table>
<thead>
<tr>
<th>Code</th>
<th>Function text</th>
<th>Menu</th>
<th>Explanations</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUS1</td>
<td>Create</td>
<td>&lt;Object&gt;</td>
<td>On initial screen only, for changing the activity.</td>
</tr>
<tr>
<td>BUS2</td>
<td>Change</td>
<td>&lt;Object&gt;</td>
<td>On initial screen only, for changing the activity.</td>
</tr>
<tr>
<td>BUS3</td>
<td>Display</td>
<td>&lt;Object&gt;</td>
<td>On initial screen only, for changing the activity.</td>
</tr>
<tr>
<td>BUSA</td>
<td>&lt;Changing instance&gt;</td>
<td>Extras</td>
<td>Display change documents for current instance</td>
</tr>
<tr>
<td>BUSB</td>
<td>Back</td>
<td>Goto</td>
<td></td>
</tr>
<tr>
<td>BUSC</td>
<td>Cancel</td>
<td>Edit</td>
<td></td>
</tr>
<tr>
<td>BUSE</td>
<td>&lt;Enter&gt; key</td>
<td>None</td>
<td>Enter</td>
</tr>
<tr>
<td>BUSF</td>
<td>Exit</td>
<td>&lt;Object&gt;</td>
<td></td>
</tr>
<tr>
<td>BUSH</td>
<td>Check input</td>
<td>Edit</td>
<td>Data is checked and the current screen is called up again</td>
</tr>
<tr>
<td>BUSI</td>
<td>Changes to fields</td>
<td>Extras</td>
<td>Display change documents for current instance (for current cursor field only)</td>
</tr>
<tr>
<td>BUSL</td>
<td>Deselect all</td>
<td>Edit</td>
<td>For screen selection on initial screen</td>
</tr>
<tr>
<td>BSUM</td>
<td>Select all</td>
<td>Edit</td>
<td>For screen selection on initial screen</td>
</tr>
<tr>
<td>BUSS</td>
<td>Save</td>
<td>&lt;Object&gt;</td>
<td></td>
</tr>
<tr>
<td>BUSV</td>
<td>Other functions</td>
<td>&lt;Object&gt;</td>
<td>Display additional functions for the &lt;Object&gt; menu</td>
</tr>
<tr>
<td>BUSW</td>
<td>Further editing</td>
<td>Edit</td>
<td>Display additional functions for the Edit menu</td>
</tr>
<tr>
<td>BUSX</td>
<td>Additional goto</td>
<td>Goto</td>
<td>Display additional functions for the Goto menu</td>
</tr>
<tr>
<td>BUSY</td>
<td>Additional extras</td>
<td>Extras</td>
<td>Display additional functions for the Extras menu</td>
</tr>
<tr>
<td>BUSZ</td>
<td>Expanded environment</td>
<td>Environment</td>
<td>Display additional functions for the Environment menu</td>
</tr>
</tbody>
</table>
3.2.7.2 Standard Functions

All menu options defined within the menu created by the application which owns the application object should be entered as standard functions. When defining a standard function, you determine when it is to be active or inactive. This can be done on the basis of:

- The activity
- The editing mode (save or transfer)
- The views of the current screen
- The combination view/screen sequence category

There are three tables which can be used for setting:

- Standard functions
- Standard functions: Active per view
- Standard functions: Active per view/screen sequence category

Menu path: Control <Object> → Menu options → Standard functions

Example: Different requirements can be met using these tables:

- The function Save (function code BUSS) is active only for activity types Create and Change and in save mode.
- The function Delete bank details in Business Partner (function code BUPI) is active only if the view Bank details is in the current screen and for activity types Create and Change.
- The function Address overview in Business Partner (function code BUAO) is active only if the view Address data is in the current screen and in the screen sequence category Main screen sequence, not in the screen sequence category Address details.

Naming convention: <Application><Function>

Application area: Application which owns the application object

3.2.7.3 Additional Functions

All menu options not fixed in the menu can be included in this menu through additional functions. These additional functions can be reached through the following submenus:

- Further object functions (<Object> menu)
- Further editing (editing menu)
- Additional goto destinations (Goto menu)
- Additional extras (Extras menu)

Each of these submenus is active only if at least one additional function is active for the menu bar option. When defining an additional function you determine
when it is to be active or inactive. The table settings available for this correspond to those for the standard functions (see section 3.2.7.2)

**Menu path:** Control <Object> → Menu options → Additional functions

**Naming convention:** <Application><Function>

**Application area:** All applications

**Events DCUAD and DCUAC**

The menu is set in events DCUAD and DCUAC. Menu options can also be set to active or inactive here if the rule which applies to this function cannot be represented using the settings in the control table.

- **Event DCUAD (setting the menu)**

  **Description:** The application determines the GUI status and sends it to the BDT. The BDT also receives the name of the function module which sets the GUI status (command SET PFSTATUS...). This function module must be included in the function group of the application which owns the application object. The naming convention for this function module is <Application>_<Application object>_PFSTATUS_SET.

  **Application area:** Application which owns the application object.

  **Naming convention:** <Application>_<Application object>_EVENT_DCUAD

  **Example:** BUP_BUPA_EVENT_DCUAD.

  **Action required:**
  - Determine GUI status for the screen and send to the BDT.
  - Send the name of the function module for setting the GUI status to the BDT.

- **Event DCUAC (changing the menu)**

  **Description:** Menu options may be set to active or inactive at runtime. This is done here for the menu options whose active/inactive rule cannot be fully represented in the control tables.

  **Application area:** All applications.

  **Naming convention:** <Application>_<Application object>_EVENT_DCUAC

  **Example:** BUP_BUPA_EVENT_DCUAC

  **Action required:**
  - Determine current GUI status with the function module BUS_CUA_STATUS_GET
  - Include inactive standard functions or active additional functions in the relevant tables
  - Return current GUI status to the control with the function module BUS_CUA_STATUS_SET

**3.2.8 Field Groupings**
3.2.8.1 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 (see section 3.2.2.1). 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) Show
  - - (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 (see section 3.2.7.3, “Adjusting generated maintenance programs”). 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).

3.2.8.2 Predefined Field Grouping Criteria

The BDT offers field grouping for
- Activity and
- 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.
3.2.8.3 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**
  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.
  - **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_TBO04_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 also 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.

**3.2.9 Assigning Screen Fields ➔ Database Fields**

In many cases the database fields themselves are not put directly on the screen, but rather the fields of structures for the screen fields, or the fields of program-internal tables (especially for table controls). The names of the screen fields were entered when assigning field groups ➔ fields (see section 3.2.2.1). If the name of a screen field differs from the name of the database field, the assignment must be entered here. The assignment must be clear in both directions so that the BDT can determine the other field name correctly. The following assignments are possible:

- **Assignment for a field**
  Table and field names must be given for both the screen field and the database field.

- **Assignment for all fields in a table**
  The table name must be given both for the screen field and for the database field (for field names enter “*”).

  When determining the name, the BDT first searches for an assignment for the screen field you want to find. If this is not found, the BDT will search for a corresponding assignment for the table (field name = “*”). If this search is also unsuccessful, the BDT will assume that the table field name and the screen field name correspond.
3.2.10 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.

3.2.10.1 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 next chapter), are available everywhere.

3.2.10.2 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)

3.2.11 Authorizations

Authorizations are checked thoroughly

- Between the initial screen and first data screen.
- Before saving

3.2.11.1 Predefined Authorization Checks

The following authorization checks are provided by the BDT:

- **Authorization types**
  - Using this authorization object, authorizations can be defined on the basis of the field value of any data fields. Authorization types can be defined within the IMG for this purpose. For each authorization type, you must determine which field is to be checked. Within the authorizations you can then determine which user is permitted to maintain which field values.
  - Example of the use of authorization types:
    - Sort business partners according to their last names.
      - User A is responsible for business partners whose last names begin with the letters A to K,
      - User B is responsible for business partners whose last names begin with the letters L to Z.
    - Sort business partners according to their postal codes.
      - User A is responsible for business partners whose postal codes begin with numbers 0 to 5,
      - User B is responsible for business partners whose postal codes begin with the numbers 6 to 9.
  - In some cases it may be useful to use authorization types with two fields. This means the user needs authorization for both fields to carry out maintenance tasks. An example of where this may be used:
    - Sort business partners according to the combination country/postal code.
      - User A is responsible for business partners in Germany whose last names begin with the letters A to K
      - User B is responsible for L to Z in Germany
      - User C is responsible for A to F in Austria
  - To use authorization types, the developer must carry out the following steps for an application object:
    - Create authorization field for authorization type
If you use a separate authorization field for each application object, the possible entries will only show the values of the relevant application object in authorization maintenance. The data element BU_AUTYP should be used when defining the authorization field. We recommend the following for foreign key definition:

- Value table TB030
  - Foreign key fields:
    TB030-CLIENT = SYST-MANDT
    TB030-OBJAP = <Name of application object as constant>
    TB030-AUTYP = <Name of authorization field>
  - Example: Business partner authorization field AUTHTYP

- Create authorization object with the fields
  - Authorization type
  - Field value (field 1)
  - Field value (field 2)
  - Activity

  **Example:** Authorization object B_BUPA_ATT in application object BUPA
  - Define the name of the authorization object and the authorization field for authorization types within application object maintenance
  - Generate setting transaction for the setting activity 0102 authorization types and include maintenance in the IMG.

If a customer now wishes to use the authorization types, they must carry out the following steps:

- Create authorization type within the IMG giving the field name
  **Example:** The definition of the authorization type can be found in the business partner’s IMG under
  - Create authorizations, giving the field values, and allocate to the users

**Authorization per field group**

Using this authorization object the customer can determine which user is authorized to maintain which field groups. The authorization-relevant field groups can be determined for this within customizing. This check is not carried out for any other field group. In other words, from the point of view of this authorization check, the field group can be maintained by any user.

To use it the developer must carry out the following steps for an application object:

- Create authorization field for field groups
  If you use a separate authorization field for each application object, the
possible entries will only show the authorization-relevant field groups for an application object in authorization maintenance. The data element BU_FLDGRAU should be used when defining the authorization field. We recommend the following for foreign key definition:

- Value table TB031
  - Foreign key fields:
    
    TB031-CLIENT   = SYST-MANDT
    TB031-OBJAP    = <Name of application object as constant>
    TB031-FLDGR    = <Name of authorization field>
  
  - Example: Business partner authorization field FLDGR

□ Create authorization object with the fields

- Field groups
- Activity
  - Example: Authorization object B_BUPA_FDG in application object BUPA

□ Define the name of the authorization object and the authorization field for field groups within application object maintenance.

3.2.11.2 Additional Authorization Checks

Any other authorization checks can be carried out. Use event AUTH1 for the checks. This will be called up by the BDT:

□ Between the initial screen and first data screen and
□ Before saving.

3.2.12 Divisibility

Particularly for central application objects with a lot of tables and fields, it is easier not to have to maintain the whole object each time, but only parts of the data (object parts) depending on the business environment. The divisibility function within the BDT can be used for this.

The BDT supports two types of divisibility:

□ Simple divisibility
  Each instance can be created in exactly one object part
  The information regarding which object part an instance is created in is an attribute of the primary table. An example of this is a bank account which can only be created in exactly one account type (for example, current account, clearing account etc.).

□ Multiple divisibility
  Each instance can be created in several object parts
  The information regarding which object parts an instance is created in is stored in a separate table. The table key is made up of the object’s primary key and the object part. An example of this is a business partner who can appear in several roles (for example, ordering party, payer, account holder etc.).

In this form of divisibility, a further distinction can be made between whether
it is permitted to maintain the instance in several object parts within a maintenance dialog. If it is permitted, object part grouping can be used to combine several parts for maintenance.

3.2.12.1 Details on Application Objects

The form of divisibility is determined here. Terms such as object part and object part grouping can be replaced by terms from the application object’s language environment. In Business Partner these terms have been replaced by “BP role” and “BP role grouping”.

Menu path: General control → Application objects

Naming convention: <Application>NNN

Object parts should always have 6 places. It should start with the application ID followed by a set of numbers.

3.2.12.2 Object Parts

For each application object, you can define any number of object parts. To define an object part, follow the steps below:

- Define object part
  Along with the name and the title, define the screen sequence and differentiation type. You can also decide here whether or not a screen selection is to be shown on the initial screen.
  
  Naming convention: <Application>NNN
  
  Object parts should always have 6 places. It should start with the application ID followed by a set of numbers.

- Assign object part → Applications
  This allocation will determine which applications provide their own views in this object part or wish to call up the maintenance of this object part.

- Assign object part → Views
  Only some attributes are relevant for each object part. This assignment will enable you to determine which attributes are relevant for an object part. The BDT automatically hides the remaining attributes in the dialog.

Menu path: Control <Object> → <Object parts>

Note: In many cases application transactions (see section 3.2.13) can be used for calling up the maintenance dialog. Otherwise you can call up the dialog using the BDT function module BUS_CONTROL_MAIN.

3.2.12.3 Object Part Groupings

Object parts can be combined using object part groupings. To define an object part grouping, use the following steps:

- Define object part grouping
  Apart from the name and the title, define only the screen sequence for the
dialog here.

**Naming convention:** <Application>NNN
Object part groupings should always have 6 places. It should start with the application ID followed by a set of numbers.

- **Assign object part grouping ➔ Object parts**
  This assignment will enable you to determine which object parts make up the grouping. During the dialog for a grouping, the BDT will show only those views which are assigned to at least one relevant object part.

**Menu path:** *Control* <Object> → <Object part grouping>

### 3.2.13 Application Transactions

With application transactions, the BDT offers an easy way to call up the maintenance dialog for an application object. This covers the majority of maintenance calls.

**Menu path:** *Control* <Object> → *Application transactions*

**Action required:**
- Create a report transaction and enter the report BUSSTART as the start parameter.
- Create an application transaction for each call and enter the details for the following fields:
  - Transaction code
  - Activity
  - Object part or object part grouping (optional)
- Include the transaction in your task level menu

When the application transaction is started, the BDT will determine the parameters you have defined and call up the central dialog module *BUS_CONTROL_MAIN*.

**Note:** Not all maintenance transactions are covered by the application transactions. If, for example, several object parts are to be maintained at the same time or the initial screen is to be filled with reference values and skipped, you will have to call up the BDT function module *BUS_CONTROL_MAIN* directly.
### 3.2.14 Transfer Mode

In addition to save mode, in which the amended data is either saved or discarded when maintenance is being exited, we also have transfer mode. This has the function Transfer, instead of Save, which notes the data in the application’s global memory. The data is written to the database at a later stage.

**Example:** Two business partners are required before a contract can be created. If necessary, these can be created from contract maintenance. However, this would be done in transfer mode. The new business partners are first created together with the contract in the database.

The sequence of events is different from that in save mode. Once the data has been noted in the global memory in event DTAKE, the flow is interrupted. Events DSAVC and DSAVE are processed later for data storage (see figure 2).

![Diagram of events in transfer mode](image)

### 3.3 Change Document Lists

The BDT offers the display of change documents as a generic service. All application objects and their applications can take advantage of this service through events CHGD1, CHGD2, CHGD3 and CHGD4.

The BDT differentiates between two types of list:

1) **Dialog lists**
These are started from the maintenance dialog.

- **Changes to instances**
  Change documents for the instance currently being processed are displayed. The user first receives a dialog box showing all the fields and tables which have been changed, from which a selection can be made for display.

- **Changes to fields**
  Change documents for the instance currently being processed are displayed insofar as they are relevant for the current cursor field.

2) **Report lists**
These lists are started as a separate report (transaction SA38 or SE38). The number of change documents to be displayed can be defined on a generated selection screen according to object key, change date, person who made the changes and the field name. The selection can also be expanded using the differentiation types, where these are used in the application object.

The list of change documents can be re-sorted subsequently, and further restrictions applied to the display.

3.3.1 **Events CHGD1 to CHGD4**

The following events can be used by all applications in an application object to include their tables in the change document list:

- **CHGD1 (Change documents: Transfer selection values)**
  The application recognizes the data in the dialog lists that belongs to the object instance currently being edited and can transfer the relevant object classes and object values to the BDT without further information. The applications get their selections from the BDT. The applications use the selections to determine the relevant object classes and object values and transfer them to the BDT. The BDT uses these specifications to import the change documents.

**Interface:**

- `I_XDIALOG` Indicator: Dialog list
- `I_XGLOBAL` Ind.: General data (without differentiation) relevant
- `T_VALUE_OBJECTID` Selection for the primary key
- `T_VALUE_DIFFTYP` Selection for differentiation types
- `T_OBJECTID` List of relevant object classes/object values

**Activities:**

- Determine list type using import parameter `I_XDIALOG`
- Dialog lists
  - Collect keys from the relevant instances into one table
  - Call BDT function module `BUS_CDOBJECTID_COLLECT` and the transfer
the relevant instances to its interface table \( T_{OBJECTID} \). This creates an entry from the combination of object class (corresponds to the change document object) and object value. If table \( T_{OBJECTID} \) remains blank, the BDT reads the change documents for all object instances.

- Report lists
  - Note value of parameter \( \_I\_XGLOBAL \) for event CHGD4
  - Note value of interface table \( T\_VALUE\_DIFF\_TYP \) for event CHGD4 (only if application object uses differentiation)
  - Read entries on the selection screen for the primary key; determine interface table \( T\_VALUE\_OBJECTID \) and thus the relevant object instances using the parameter
  - Enter the relevant object instances in interface table \( T\_OBJECTID \). This creates an entry from the combination of object class and object value. If table \( T\_OBJECTID \) remains blank, the BDT reads the change documents for all object instances.

- **Application area:** All applications which own tables and write change documents for their tables.
  **Naming convention:** \(<\text{Application}>\_<\text{Application object}>\_EVENT\_CHGD1\).
  (Customer: Function module name has the prefix Y_ or Z_).
  **Example:** BUP_BUPA_EVENT_CHGD1

- **CHGD2 (Change documents: Collect object names)**
  Various texts for the dialog box in which you select tables/fields (dialog list of object changes) as well as for the change document list itself are defined here by the applications.

  **Interface:**
  \(<C\_OBJBEZ>\quad\text{Object name in change document list}\)
  \(<T\_TABBEZ>\quad\text{Names of tables}\)

  **Activities**
  - Define names of individual tables for selecting tables/fields in the list of object changes
  - Define object name for the change document list (only application which owns application object)

- **Application area:** All applications which own tables and write change documents for their tables.
  **Naming convention:** \(<\text{Application}>\_<\text{Application object}>\_EVENT\_CHGD2\).
  (Customer: Function module name has the prefix Y_ or Z_).
  **Example:** BUP_BUPA_EVENT_CHGD2

- **CHGD3 (Change documents: Header relevant?)**
  The BDT reads the change document headers based on the values transferred by the applications into event CHGD1. At event CHGD1, the application could not precisely restrict the relevant change document headers. At event
CHGD3, the application can still prevent non-relevant change document headers from being read.

**Interface:**

- `/I_OBJECTCLAS` Object class
- `/I_OBJECTID` Object value
- `/C_XREL` Indicator: Change document header is relevant

**Application area:** This event is generally not needed by the applications.

**Naming conventions:** `<Application>_ <Application object>_EVENT_CHGD3`

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

- **CHGD4 (Change documents: Item relevant?)**

Once the BDT has imported the change document headers as well as items, the application can decide at event CHGD4 for each item whether it is relevant. If this is the case, you can change the item display using different return parameters.

**Interface:**

- `/I_TABNAME` Table name of item
- `/I_TABKEY` Table key of changed entry
- `/C_XREL` Indicator: Item is relevant
- `/C_TABBEZ` Name of table (for additions or deletions)
- `/C_KEYNAM` Name for the output of an additional key
- `/C_KEYVAL` Additional key

**Activities:**

- Decide whether an item is relevant and tell this to the BDT using parameter `C_XREL`
- Item is relevant

- If additions or deletions are made from a table, the name of the table is output in the *Field name* field. You can define this name using parameter `C_TABBEZ`.
- The output in the *Additional key* field can be defined using parameters `C_KEYNAM` and `C_KEYVAL`.

**Application area:** All applications which own tables and write change documents for their tables.

**Naming convention:** `<Application>_ <Application object>_EVENT_CHGD4`

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

**Example:** BUP_BUPA_EVENT_CHGD4

### 3.3.2 Calling Lists

You can call the dialog lists with the menu options predefined by the BDT. Include the following functions in your menu:
You can replace the function names in your menu with a more fitting description of the application object. For example, in the Business Partner application object, the first function is called Partner changes.

You call the report list using the setting transaction for setting activity 0200 Change Document Lists.

3.4 Data Maintenance Without Dialog

There is a difference between direct input and function modules in data maintenance without dialog. However, the program logic from dialog maintenance is used to the greatest possible extent for both.

The applications recognize data maintenance without dialog by the value of the return parameter E_XDINP in function module BUSI_PARAMETERS_ISRST_AVAIL. If this parameter is set to ‘X’ (marked), it means the data maintenance is without dialog, while an initial value indicates dialog maintenance.

3.4.1 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.

3.4.1.1 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 BUSSDIHDR. 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_DI.

- **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_DI  Table BUT000 (general data)
    - BUS0BK_DI  Table BUT0BK (bank details)
    - BUS020_DI  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.
3.4.1.2 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 (see figure 3).

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 was 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)
- 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 BUSDIDAT1) contains fields TBNAM and DATA. The name of the DI structure is in field TBNAM. 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.

3.4.2 Function Modules
Maintenance using function modules corresponds in most ways to 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 BUSSID2.
   - You convert each data record in a table to the format of structure BUSSID3. 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 structure BUSSID.

   **Examples:**
   - BUP_DI_DATA_GLOBAL_DATA
   - BUP_DI_DATA_BANKDETAILS
   - 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.

### 3.5 External Interfaces

The strict division between interface and program logic as well as the encapsulation of program logic in function modules allows BDT application objects to maintain data with an interface created outside the BDT while using the function modules defined in the BDT as program logic. You can use the Screen Painter in ABAP to create this external interface. You also have the option of using other interface tools such as Visual Basic.

**Note:** The functionality of external interfaces is only released for specially designated projects within SAP.

#### 3.5.1 Process

External interfaces are designed solely in external maintenance transactions (abbreviation: external maintenance). You define how data fields are distributed on the screens of this maintenance transaction. The only restriction is that fields belonging to one view always have to be put together on a data screen since there is a common function module for each view that is used to check data (event *After input*).

The function modules defined for the BDT events are used to read data from the database as well as to check the data and save it. External maintenance determines the current data prior to output using the function modules that read the data (see section 3.2.4).

External maintenance and BDT maintenance can be fully integrated. You can call BDT maintenance in transfer mode from external maintenance. Data can be changed in both types of maintenance. Changes are also visible in each type of maintenance. When making the transition from one kind of maintenance to the other, the data is flagged in the global memory of the application function groups.

The BDT provides varying function modules that are called in external maintenance (procedure described in detail in the next section). These modules in turn process one or more BDT events, which calls event modules in the applications.

Multiple BDT application objects can be maintained at once in external maintenance.

**Note:** In order to be able to work with external interfaces, it is very important that you adhere to the specifications given by the developer for the required actions in events. Correct reading and correct supply of the current and global memory in events ISDAT, DTAKE, DSAVC and DSAVE are particularly important.

**Example:** External interfaces are used in IBU Utilities to maintain move-in documents. A business partner and a contract account, among other objects, are
maintained using the maintenance transactions for move-in documents (transactions EC50, EC51 and EC52).

### 3.5.2 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 and
then flagged. 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 (see section 3.2.4).

- **BUS_FOREIGN_HEADER_CHECK**
  After the header data is input, 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 DSAVB, 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 DSAVC and DSAVE 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.
4 Roadmap for Implementation

This section is intended to assist developers implement the most common requirements. One approach to the problem is described in each sub-section. This is followed by a list of required actions in the order they are to be performed. Reference is frequently made to the associated functional description given in section 3.

The following options will be of particular interest to customers who want to extend standard SAP data maintenance methods:

- Customers add another check for a standard SAP view (see section 4.1)
- Customers extend a standard SAP table with their own fields and integrate them into data maintenance (see section 4.2.2)
- Customers integrate the fields from their own table into data maintenance (see section 4.2.3).

4.1 Additional Check for Existing View

An additional check which is called directly after input into fields and sends a message if necessary is to be added to the view of another application.

Action required:

- Define application if you have not already done so (see section 3.1.3)
- Assign the application to the object part if the application object uses divisibility and the application is not yet assigned (see section 3.2.12)
- Implement the check function module in its own function group
- Store the name of the function module in the view definition in the subnode Additional checks (see section 3.2.2.2.)

Note: Do NOT store the name of your check function module directly in the After input field in the view definition if this view belongs to another application. If you were to do this, it would change the table entry of the other application. The change could then be lost during an upgrade.

4.2 New Table Fields and New Tables

Many of the applications participating in an application object need to be able to extend data maintenance with their own fields or even their own tables. The sections below describe the different options for doing this.

4.2.1 New Table Field Using Owning Application
An application already integrated its table fields into data maintenance. Now there is an additional data field that is also to be included in the dialog.

**Activities:**
- [ ] Extend table in DDIC
- [ ] Extend screen layout/screen sequence with the new field (see section 4.4)

### 4.2.2 New Table Field Using Participating Application

An application wants to attach a field to the table that belongs to another application and make this field maintainable in dialog. The following activities are necessary:

- Create APPEND structure for the table in the DDIC
- Define application if you have not already done so (see section 3.1.3)
- Assign the application to the object part, if this has not already been done and the object uses divisibility (see 3.2.12.2)
- Extend screen layout/screen sequence with the new field (see section 4.4)
- Create program logic for events (see section 3.2.5)
  - [ ] ISSTA  Initialize
  - [ ] ISDST  Distribute data
  - [ ] XCHNG  Were changes made?
  - [ ] DSAVB  Collect data
  - [ ] DLVE1  Initialize current memory

**Note:** SAP policy has it that a table should only be extended by additional fields after first consulting with the owning application. Alternatively, you can also create new tables and integrate them into the dialog.

### 4.2.3 New Table

An application wants to extend the dialog with its own tables. The following activities are necessary:

- Create table in DDIC
- Define application if you have not already done so (see section 3.1.3)
- Enter table in the table directory (see section 3.2.4)
- Extend screen layout/screen sequence with new fields (see section 4.4)
- Create program logic for events (see section 3.2.5)
  - [ ] ISSTA  Initialize
  - [ ] ISDAT  Read data
  - [ ] XCHNG  Were changes made?
Integrate table in change document lists

4.3 Extend Screen Layout and Screen Sequence

New fields can be integrated into screen layout and screen sequence in several different ways.

4.3.1 Include Field in Existing Field Group

If the new field is so closely connected with the fields already created for the field group in your application that these fields can only be shown or hidden together, the new field should be included in this field group. The following activities are necessary:

- Assign the new field to an existing field group (see section 3.2.2.1)
- Change view to which the field group is assigned (see section 3.2.2.2)
  - Include new field in subscreen
  - Assign screen field and database field if necessary (see section 3.2.9)
  - Extend program logic (event After input etc)

Note: A field group may only be assigned fields from one application. Other applications must define separate field groups for their fields.

4.3.2 Include Field in Existing View

If the new field belongs to a fields in a view that already exists in your application in terms of its display and checks, it should be included in this view. You use its own field group to show or hide the new field separate from the others. The following activities are necessary:

- Create field group and assign new field (see section 3.2.2.1)
- Change view to which the field group is assigned (see section 3.2.2.2)
  - Include new field in subscreen
  - Assign screen field and database field if necessary (see section 3.2.9)
  - Extend program logic (event After input etc)

Note: Only fields belonging to the same application can be located in a view. Other applications must define separate views for their fields.

4.3.3 Include New View in Existing Section
If a new field is independent in terms of its display and checks but belongs to existing views in terms of its content, you need to create a separate view for it and assign this view to the existing section. The following activities are necessary:

- Create field group and assign new field (see section 3.2.2.1)
- Create view and assign new field group (see section 3.2.2.2)
- Adapt object part if the application object uses divisibility (see section 3.2.12.2)
  - Assign application to the object part
  - Assign view to the object part
- Assign new view to an existing section (see section 3.2.2.3)

4.3.4 Include New Section in Existing Screen

If a new field is independent in terms of its display and checks as well as in terms of its content, you create a separate view and a separate section for this field. If the section fits the context of an existing screen, the new section should be assigned to this screen. The following activities are necessary:

- Create field group and assign new field (see section 3.2.2.1)
- Create view and assign new field group (see section 3.2.2.2)
- Adapt object part if the application object uses divisibility (see section 3.2.12.2)
  - Assign application to the object part
  - Assign view to the object part
- Create section and assign new view (see section 3.2.2.3)
- Assign new section to an existing screen (see section 3.2.2.4)

4.3.5 New Screen in Existing Screen Sequence

If the field does not fit in the context of an existing screen, create a new screen and integrate it into an existing sequence. The following activities are necessary:

- Create field group and assign new field (see section 3.2.2.1)
- Create view and assign new field group (see section 3.2.2.2)
- Adapt object part if the application object uses divisibility (see section 3.2.12.2)
  - Assign application to the object part
  - Assign view to the object part
- Create section and assign new view (see section 3.2.2.3)
- Create screen and assign new section (see section 3.2.2.4)
• Assign the new screen to an existing screen sequence (see section 3.2.3.2)

**4.3.6 New Screen Sequence as Main Screen Sequence**

If an application object uses divisibility, you have the option of creating a new main screen sequence and assigning it to the object part in which the new field is relevant. Since the views not assigned to an object part are not automatically displayed by the BDT, a new main screen sequence is only necessary if:

• You want a different sequence of screens
• Various screens are to be combined because otherwise there would be a number of nearly blank screens

The following actions are required to create a new main screen sequence:

• Create field group and assign new field (see section 3.2.2.1)
• Create view and assign new field group (see section 3.2.2.2)
• Adapt object part if the application object uses divisibility (see section 3.2.12.2)
  - Assign screen sequence to the object part
  - Assign application to the object part
  - Assign view to the object part
• Create section and assign new view (see section 3.2.2.3)
• Create screen and assign new section (see section 3.2.2.4)
• Create new sequence and assign screens (see section 3.2.3)
• Assign new sequence to category *Main screen sequence* (see section 3.2.3.4)

**4.3.7 New Additional Screen and/or New Additional Sequence**

If you want the new field to be on a separate screen but do not want it to be processed in the main screen sequence, it must be called explicitly using an additional function. Carry out the following activities:

• Create field group and assign new field (see section 3.2.2.1)
• Create view and assign new field group (see section 3.2.2.2)
• Adapt object part if the application object uses divisibility (see section 3.2.12.2)
  - Assign application to the object part
  - Assign view to the object part
• Create section and assign new view (see section 3.2.2.3)
• Create screen and assign new section (see section 3.2.2.4)
• Create new sequence and assign screens (see section 3.2.3.1 and 3.2.3.2)
• Create new sequence category and assign new screen sequence (see section 3.2.3.3 and 3.2.3.4)
• Create new menu option and assign the new sequence category to it (see section 3.2.7)

4.4 New Object Part
If an application object uses divisibility, you can create any number of object parts for it. To include a new object part, you need to carry out the following activities:
• Create new object part (see section 3.2.12.2)
• Create application transactions (see section 3.2.13)

4.5 New Field Grouping Criterion
If the field grouping criteria predefined by BDT are not sufficient (section 3.2.8.2), additional criteria may be added (see section 3.2.8.3 for the procedure).

4.6 New Application Object
The BDT can be used by various application objects. Before you start doing any development, register the new object with the Business Partner development group in IBU Insurance. We then recommend you use the following procedure to create a new application object:
• Create application object (see section 3.1.2.1)
• Create setting transaction and use this to create a task menu (see sections 3.1.2.3 and 3.1.2.4)
• Define applications
• Create subscreens for related fields
• Determine screen layout and screen sequence
• Create program logic

Note: Using the BDT for development has only been approved for registered application objects in SAP.
5 Appendix

5.1 Terminology Definitions

**Current memory:** The current memory contains the data on the instance that is being edited at the moment. To do this, the owning application as well as the participating applications in their function group create globally valid field structures (only one record per instance possible) or tables (several records possible) for each table. Old and new datasets are kept separately in the current memory.

The old set is made up of data that exists on the initial screen in maintenance. This data is generally flagged at event ISDAT (table-owning application) or ISDST (table-participating application). The new set reflects the data currently entered and is also flagged for the first time at events ISDAT and/or ISDST. The status is then updated after each entry.

**Global memory:** This memory contains data for all instances that have already been edited but have not yet been saved to the database. The owning application creates only one memory for each table. The data from the participating application is also flagged in the table. Regardless of whether one or more records can exist for each instance in the table, the owning application creates a globally valid table within its function group. As in the current memory, a distinction is made in the global memory between the old and new statuses, which are also flagged separately. The old status represents the current database status and is only created in the first data maintenance for an instance for this reason. This takes place by transferring the data (event DTAK) from the old status of the current memory. If this instance is maintained again in transfer mode without saving in the meantime, you cannot change the old status at this point. The new status is updated every time the data is transferred. The new status is copied from the new status of the current memory.

**Applications which own the tables:** Application that created the table and in whose development class the table is located. This application is responsible for reading the data on the initial screen (event ISDAT) as well as for writing the data to the database (event DSAVE).

**Applications using the tables:** Application that adds data fields to the table of another application using the APPEND structure. This application takes data from the application that owns the tables (event ISDST) and transfers the new status of its own data fields to this application prior to saving the data (event DSAVB).

**Events:** Events are defined in different places in the BDT. Each application can define the name of its own function modules in the BDT’s control tables at these events. Once the process reaches an event, the BDT automatically calls all the function modules in the applications whose names are entered for this event. Applications can use this technique to include their own program logic.
5.2 Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADK</td>
<td>Archive Development Kit</td>
</tr>
<tr>
<td>ALV</td>
<td>ABAP List Viewer</td>
</tr>
<tr>
<td></td>
<td>Tool for creating flexible evaluations</td>
</tr>
<tr>
<td>BDT</td>
<td>Business Data Toolset</td>
</tr>
<tr>
<td>DDIC</td>
<td>Data Dictionary</td>
</tr>
<tr>
<td>DI</td>
<td>Direct Input</td>
</tr>
<tr>
<td>FM</td>
<td>Function Module</td>
</tr>
<tr>
<td>BP</td>
<td>Business Partner</td>
</tr>
<tr>
<td>PAI</td>
<td>Process After Input</td>
</tr>
<tr>
<td>PBO</td>
<td>Process Before Output</td>
</tr>
<tr>
<td>SAP-BP</td>
<td>SAP Business Partner</td>
</tr>
<tr>
<td></td>
<td>Project for integrating all BP data into R/3</td>
</tr>
<tr>
<td>VCT</td>
<td>Visual Configuration Tool</td>
</tr>
<tr>
<td></td>
<td>Tool for configuring screen layout and sequence using drag&amp;drop</td>
</tr>
<tr>
<td>CAM</td>
<td>Central Address Management</td>
</tr>
<tr>
<td></td>
<td>Project for integrating all address data into R/3</td>
</tr>
<tr>
<td>CBP</td>
<td>Central Business Partner</td>
</tr>
<tr>
<td></td>
<td>Previous project name of SAP BP</td>
</tr>
</tbody>
</table>

5.3 Naming Conventions

The applications of an application object can add their own data and/or their own program logic. To do this, they add entries to the BDT control tables. To a certain extent, these applications complete implementation in different systems. For this reason, each application must adhere to the predefined naming conventions when making entries in control tables. Otherwise, entries in other applications would be overwritten. There is also a recommendation for naming event function modules.

You will find the relevant naming convention for each topic in chapter 3. The table below provides you with an overview of the naming conventions needed when using the BDT. Adhere to these rules wherever possible.

### Naming conventions:

<table>
<thead>
<tr>
<th>Object</th>
<th>SAP</th>
<th>Dev. partner</th>
<th>Customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Differentiation types</td>
<td>0*..9*, A*..W* (central assignment)</td>
<td>X*</td>
<td>Y*, Z*</td>
</tr>
<tr>
<td>Differentiation type elements</td>
<td>0*..9*, A*..W* (central assignment)</td>
<td>X*</td>
<td>Y*, Z*</td>
</tr>
<tr>
<td>Application objects</td>
<td>A*..W* (central assignment)</td>
<td>May only be used for SAP objects</td>
<td>May only be used for SAP objects</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------</td>
<td>----------------------------------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>Applications</td>
<td>Aa .. Wa</td>
<td>Xa</td>
<td>Ya, Zaa</td>
</tr>
<tr>
<td></td>
<td>Aaa .. Waa</td>
<td>Xaa (Xaa)</td>
<td>Yaa, Zaa (Yaa, Zaa)</td>
</tr>
<tr>
<td>Field groups</td>
<td>001 – 599 (central assignment)</td>
<td>750 – 1000</td>
<td>600 - 749</td>
</tr>
<tr>
<td>Views</td>
<td>&lt;pp&gt;nnnn</td>
<td>&lt;pp&gt;nnnn</td>
<td>&lt;pp&gt;nnnn</td>
</tr>
<tr>
<td></td>
<td>&lt;ppp&gt;nn</td>
<td>&lt;ppp&gt;nn</td>
<td>&lt;ppp&gt;nn</td>
</tr>
<tr>
<td></td>
<td>&lt;pppp&gt;nn</td>
<td>&lt;pppp&gt;nn</td>
<td>&lt;pppp&gt;nn</td>
</tr>
<tr>
<td>Sections</td>
<td>&lt;pp&gt;nnnn</td>
<td>&lt;pp&gt;nnnn</td>
<td>&lt;pp&gt;nnnn</td>
</tr>
<tr>
<td></td>
<td>&lt;ppp&gt;nn</td>
<td>&lt;ppp&gt;nn</td>
<td>&lt;ppp&gt;nn</td>
</tr>
<tr>
<td></td>
<td>&lt;pppp&gt;nn</td>
<td>&lt;pppp&gt;nn</td>
<td>&lt;pppp&gt;nn</td>
</tr>
<tr>
<td>Assignment</td>
<td>Section → Views</td>
<td>Naming conventions for item numbers (see below)</td>
<td></td>
</tr>
<tr>
<td>Screens</td>
<td>&lt;pp&gt;nnnn</td>
<td>&lt;pp&gt;nnnn</td>
<td>&lt;pp&gt;nnnn</td>
</tr>
<tr>
<td></td>
<td>&lt;ppp&gt;nn</td>
<td>&lt;ppp&gt;nn</td>
<td>&lt;ppp&gt;nn</td>
</tr>
<tr>
<td></td>
<td>&lt;pppp&gt;nn</td>
<td>&lt;pppp&gt;nn</td>
<td>&lt;pppp&gt;nn</td>
</tr>
<tr>
<td>Assignment</td>
<td>Screen → Sections</td>
<td>Naming conventions for item numbers (see below)</td>
<td></td>
</tr>
<tr>
<td>Screen sequences</td>
<td>&lt;pp&gt;nn</td>
<td>&lt;pp&gt;nn</td>
<td>&lt;pp&gt;nn</td>
</tr>
<tr>
<td></td>
<td>&lt;ppp&gt;n</td>
<td>&lt;ppp&gt;n</td>
<td>&lt;ppp&gt;n</td>
</tr>
<tr>
<td></td>
<td>&lt;pppp&gt;</td>
<td>&lt;pppp&gt;</td>
<td>&lt;pppp&gt;</td>
</tr>
<tr>
<td>Assignment</td>
<td>Screen sequence → Screens</td>
<td>Naming conventions for item numbers (see below)</td>
<td></td>
</tr>
<tr>
<td>Screen sequence</td>
<td>&lt;pp&gt;nn</td>
<td>&lt;pp&gt;nn</td>
<td>&lt;pp&gt;nn</td>
</tr>
<tr>
<td>categories</td>
<td>&lt;ppp&gt;n</td>
<td>&lt;ppp&gt;n</td>
<td>&lt;ppp&gt;n</td>
</tr>
<tr>
<td></td>
<td>&lt;pppp&gt;</td>
<td>&lt;pppp&gt;</td>
<td>&lt;pppp&gt;</td>
</tr>
</tbody>
</table>
## Event function modules

<table>
<thead>
<tr>
<th>&lt;pp&gt;<em>&lt;apo&gt;</em>*</th>
<th>&lt;pp&gt;<em>&lt;apo&gt;</em>*</th>
<th>Y_&lt;pp&gt;<em>&lt;apo&gt;</em>*</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;ppp&gt;<em>&lt;apo&gt;</em>*</td>
<td>&lt;ppp&gt;<em>&lt;apo&gt;</em>*</td>
<td>Z_&lt;ppp&gt;<em>&lt;apo&gt;</em>*</td>
</tr>
<tr>
<td>&lt;pppp&gt;<em>&lt;apo&gt;</em>*</td>
<td>&lt;pppp&gt;<em>&lt;apo&gt;</em>*</td>
<td>Y_&lt;pppp&gt;<em>&lt;apo&gt;</em>*</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Z_&lt;pppp&gt;<em>&lt;apo&gt;</em>*</td>
</tr>
</tbody>
</table>

### Assignment Event → FMs

<table>
<thead>
<tr>
<th>Menu options</th>
<th>Naming conventions for item numbers (see below)</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;pp&gt;nn</td>
<td></td>
</tr>
<tr>
<td>&lt;ppp&gt;n</td>
<td></td>
</tr>
<tr>
<td>&lt;pppp&gt;</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field grouping criteria</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;ppp&gt;nnnnnnn</td>
<td></td>
</tr>
<tr>
<td>&lt;ppp&gt;nnnnnn</td>
<td></td>
</tr>
<tr>
<td>&lt;ppp&gt;nnnnnn</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Object parts</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;pp&gt;nnnn</td>
<td></td>
</tr>
<tr>
<td>&lt;ppp&gt;nnn</td>
<td></td>
</tr>
<tr>
<td>&lt;pppp&gt;nn</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Object part groupings</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;pp&gt;nnnn</td>
<td></td>
</tr>
<tr>
<td>&lt;ppp&gt;nnn</td>
<td></td>
</tr>
<tr>
<td>&lt;pppp&gt;nn</td>
<td></td>
</tr>
</tbody>
</table>

### Key:

- Capital letters ==> constants
- A...Z
- Numeric ==> Alphabetic
- 0...9
- application object (2,3 or 4 places)
- application (2,3 or 4 places)

### Naming conventions for item numbers:

The example Assign screen → sections will be used in the following section to explain naming conventions for item numbers. This is similar to item numbers in other control tables. The purpose of the naming convention is so the application can add its own sections to any part of a screen regardless of the development level of the application. However, you have to ensure that any existing entries are not
overwritten by a downstream application later. For that reason, it is important to follow the rules listed below for determining the item numbers:

- Place X in a 7-place item number is assigned to each development level; only the SAP application basis is assigned two places X (1 and 2).
- The applications decide the values for the places left of X. They are used to define at which item the new section is to be inserted.
- For place X, values 1-9 are permitted, but 0 is not. In other words, a minimum of nine new sections can be added to each item.
- 0 is the only permitted value for the places right of X. This prevents entries by later development levels from being overwritten.

**Naming conventions for item numbers**

<table>
<thead>
<tr>
<th>Development level</th>
<th>Place X</th>
<th>Examples for numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAP: Application basis</td>
<td>1,2</td>
<td>0100000, 0200000, 0900000, 1000000, 1100000</td>
</tr>
<tr>
<td>SAP: Standard applications</td>
<td>3</td>
<td>0120000, 0160000, 0240000</td>
</tr>
<tr>
<td>SAP: IBUs</td>
<td>4</td>
<td>0123000, 0125000, 0248000</td>
</tr>
<tr>
<td>Development partners</td>
<td>5</td>
<td>0123200, 0123600, 0247500</td>
</tr>
<tr>
<td>Customers (central)</td>
<td>6</td>
<td>0123230, 0123270, 0247510</td>
</tr>
<tr>
<td>Customers (branch)</td>
<td>7</td>
<td>0123234, 0123238, 0247515</td>
</tr>
</tbody>
</table>