BC ABAP Workbench Tutorial

HELP.BCDWBTUT

Release 4.6B

BC ABAP Workbench Tutorial

Copyright

BC ABAP Workbench Tutorial

Icons

Icon	Meaning
	Caution
	Example
	Note
	Recommendation
	Syntax
	Tip

BC ABAP Workbench Tutorial

Contents

BC ABAP Workbench Tutorial

BC ABAP Workbench Tutorial

BC ABAP Workbench Tutorial

BC ABAP Workbench Tutorial

Note to the Reader

This note to the reader contains information you should know before going on to read the *ABAP Workbench Tutorial.* The note includes the following topics:

About the Tutorial

This tutorial introduces you to the tools in the ABAP Development Workbench. The Workbench contains the tools you use to create an ABAP application.

The tutorial contains six lessons. These lessons are intended to introduce you to the concepts of creating an application with the Development Workbench. After you complete the lessons, you should have a good understanding of the role each tool takes in the application development process.

ABAP Workbench Tutorial does not teach you how to program in *ABAP*. Learning More [Page 15] provides information about what you should read when you are done with this tutorial.

Where to Start

The tutorial lessons are designed to quickly acquaint you with the process of creating an application with the Development Workbench. You are introduced to each of the Workbench tools through the implementation of a single ABAP transaction. The transaction is a simplified flight-reservation application. Because each lesson builds on information learned in the previous lessons, you should work through the lessons in order. The following lessons are included:

- Lesson 1: Understanding the Workbench Tools [Page 16]

  introduces you to the tools found in the Workbench. You learn how to open and close the Workbench. The chapter teaches you how to use the Object Browser tool. You learn how to display both public and private object lists. Finally, you learn how to create both a program object and a transaction.

- Lesson 2: Working with Tables [Page 27]

  introduces you to the SAP Data Dictionary tool. This lesson teaches you how to create a table object. You learn what components make up a table. This lesson also teaches you how to use the Workbench tools to release objects to everyone in your R/3 System.

- Lesson 3: Designing Screens [Page 41]

  teaches you how to use the Screen Painter tool. You use the tool to lay out screen elements. You also learn how to refine a simple screen so that it has the look and feel of a commercial interface.

- Lesson 4: Specifying a GUI Status [Page 51]

  uses the Menu Painter tool to define a GUI status. You use the Menu Painter to define the tool bars in an application window. You learn how to specify menus, window titles, and function keys.

- Lesson 5: Coding the Transaction [Page 59]

  teaches you how to complete a transaction by writing code with the ABAP Editor tool. You learn how to create global variables and subprograms. You also use the Editor tool to add messages to an existing program. You learn how to run your transaction. Finally, you learn how to debug a program with the debugger.

- Lesson 6: Working in a Team [Page 70]

  teaches you how to use the Workbench Organizer tool. You learn how to set access to objects during development and how to release program code to everyone in your R/3 system.

About the Tutorial

You can work through the lessons at your own pace. Each lesson begins with an overview of what you will learn. Lessons are divided into exercises. You are given an opportunity to check your work after each exercise. At the end of every lesson, you are provided with a review of what you learned and a an overview of the next lesson.

When you complete these lessons, you should continue reading in the ABAP Workbench Tools [Ext.] to learn about the tools in detail.

Prerequisites

This tutorial is written for readers who are familiar with the R/3 System. You should have read and understood the information contained in the *Getting Started* documentation. It helps if you have some knowledge about using other areas of the R/3 system. Finally, you should have some experience with ABAP, though in-depth knowledge is not required.

Before you can use the tutorial, you must:

- Be able to log on to the R/3 system

  If you are reading this tutorial online, you probably already have an SAP logon. Otherwise, ask your R/3 system administrator for a user ID, a client, and a password. *Getting Started* documentation supplies more information about requirements for an SAP logon.

- Have Release 3.0 of the R/3 System

  If you are unsure about what version you are using, ask your system administrator.

- Be working on a UNIX, Windows 95, or Windows NT platform

  SAP's graphical full screen editor runs only on UNIX, Windows 95, or Windows NT platforms. With other platforms, you must use an alphanumeric full screen editor.

- Have access to the SDW6 development class

  If you are not sure whether this development class exists in your system, ask your system administrator for confirmation.

Terminology

Getting Started with the ABAP Workbench tutorial assumes you are already familiar with how to use the R/3 interface. If you are not, you should take the “Tour through the R/3 Window” in the *Getting Started* documentation. This tutorial uses the following terminology:

Term	Description
dialog box (or **dialog**)	A window that the system displays on top of the application window. Dialogs appear when the R/3 System needs more information from you or when it wants to give you a message.
enter	To type information in a field provided in a screen or in a dialog.
screen	The arrangement of menus, buttons, and fields that appear in a window. A single ABAP application might have several screens. The title of a screen appears in the title bar.
choose	A method for selecting options in the system using a mouse or function key. How you choose an item in the R/3 system will depend on the look and feel of your windowing system and the type of mouse you have. If you are unfamiliar with how to choose items in your environment, you should review the introductory material that accompanied your system.
select	An action that makes a radio button, list item, or text field active.
window	A graphical object on your display that contains an application. In the case of the lessons, the window contains the ABAP application.

A Word About Interfaces

The Windows NT 3.5 interface version of the Development Workbench was used to produce the screen captures you see in this tutorial. The pictures of screens and buttons in this tutorial is thus the same as as they would in Windows NT 3.5 or Windows 3.1. If you are using another interface, for example Motif or Windows 95, your screens will appear slightly different. Regardless of your system's look and feel, the procedures and examples in this tutorial work in the same manner.

Choosing Names for SAP Objects

Throughout this tutorial you are asked to name new ABAP objects. It is important that you understand and follow two R/3 naming conventions. First, all names for the objects you define should begin with `Y` or `Z`. This convention ensures that the names you select are different from the R/3 system versions.

Secondly, the names you select must be unique. The system will tell you if a name you choose already exists. Many development objects are system-wide and this convention provides an extra safeguard against loss or damage to data.

Finally, for the purposes of this tutorial, it is a good idea to use your initials as the distinguishing feature of your object names. Using your initials prevents any conflicts between you and other users who may also be working through the tutorial.

For example, if Ann Jones is asked to name an object using the form `Y<xx>ID`, she would name her object: `YAJID`. This would distinguish Ann´s object from Bob Smith´s object: `YBSID`.

Failure to follow the naming conventions can result in loss of data. Be sure you understand the conventions before continuing.

Learning More

To learn more about ABAP programming and the ABAP Development Workbench, refer to the following:

- ABAP User's Guide [Ext.]**.** This documentation provides a detailed explanation about the parts of an ABAP program. Read this, to learn more about
  - ABAP basics
  - Programming reports
  - Programming transactions
- Advanced techniques.
- ABAP Workbench Tools [Ext.]. This documentation discusses each Workbench Tool and provides detailed information about the code produced by the various tools.
- ABAP Dictionary [Ext.]. This documentation explains ABAP data. Basic objects and aggregated objects are discussed. Information is provided on maintaining dictionary objects, and several special subjects are covered.
- Workbench Organizer [Ext.]. This documentation explains how to organize large projects in the R/3 System. Setting up the organizer and the transport system are described in detail. Information is provided about version control and modification support.
- Extended Application Function Library [Ext.]. This documentation addresses such topics as:
  - Standardized dialogs
  - Central address administration
  - Application logs
  - Archiving
- Basis Programming Interfaces [Ext.]. This documentation describes the programming interfaces for ‘R/3 components. These interfaces include the background processing system and the batch input system.

Lesson 1: Understanding the Workbench Tools

In this lesson, you learn basic concepts and procedures associated with the Development Workbench. This lesson contains the following information:

Introduction to Lesson 1

This lesson introduces you to the ABAP Development Workbench and the tools it contains. When you have completed this lesson, you will be able to:

- Identify concepts underlying the Workbench
- Open the Workbench in the SAP window
- Close the Workbench
- Identify the Workbench tools and their functions
- Navigate through a program
- Create a new program
- Create a new transaction

Before you continue with this lesson, be sure you have read the Note to the Reader [Page 8].

Workbench Concepts

You use the ABAP Development Workbench to create application programs. The Workbench is a graphical programming environment [Ext.]. You access the programming tools using buttons, dialogs, and windows that appear on your computer display. In the R/3 System, the parts of a program a programmer creates are called development objects [Ext.].

Each ABAP application is either a transaction [Ext.] or a report [Ext.]. The example application you create in this tutorial is a transaction. A transaction is an end-user application [Ext.]. Transactions retrieve data from users and then perform one or more relevant actions. For example, an application that creates purchase orders is a transaction. Unlike transactions, reports are applications that require little or no user interaction.

Underlying each transaction is a module pool [Ext.] program. Module pool is the term used to describe the collection of ABAP language entities that drive a transaction.

Exercise 1: Opening the Workbench

If you have not already done so, log on to the R/3 System. Then, choose *Tools* → *ABAP Workbench* from the menu bar. The system displays the *ABAP Development Workbench* screen [Ext.].

You can return to the initial screen at any time using the *Back* button. You can exit the R/3 System at any time by selecting *System* → *Log off*. When you logoff, the system prompts you to confirm your choice.

Check Your Work

Experiment with the Workbench buttons and menus. You should feel comfortable using the buttons on the toolbar. If you need a review of interface basics, read through the *Getting Started* documentation.

As you work through the tutorial, you can stop, save your work, and exit the R/3 System. You should feel comfortable doing this. Practice exiting from different parts of the Workbench.

Exercise 2: Learning the Workbench Tools

In the last exercise, you learned how to open and how to close the Workbench. This exercise acquaints you with each Workbench tool. If you have not already done so, open the Workbench. Each tool in the Workbench serves a particular function in building an ABAP program.

The ABAP Dictionary stores system-wide data definitions. When you create a new data definition, the Dictionary tool does all the processing necessary to create the definition. You can use the Dictionary tool to look up the “definition” of objects in your R/3 System.

You use the ABAP Editor to create new code or change existing code. The ABAP editor can check to make sure you use the correct syntax in your program. Once your program is syntactically correct, you can generate, run, and debug your program from the Editor.

The Function Library is a repository of library routines. When you create a new routine, the Function Library tool does all the processing necessary to create the new routine.

You use the Screen Painter and the Menu Painter to design a graphical user interface (GUI) for your program. You use the Screen Painter to add fields, buttons, and other elements to a screen. You use the Menu Painter to create the menus that go around a screen.

The Object Browser

The ABAP Development Workbench contains a special tool, the Object Browser. The Object Browser provides a programming context. A carpenter building a cabinet has the cabinet’s physical materials, such as the wood and nails, visible in front of him. As he builds, the carpenter’s eyes provide a context to view the relationships between a cabinet’s materials.

Because a program consists of relationships between data, it is hard for a programmer to see the relationships among separate data components. The Object Browser corrects this problem by supplying the context for viewing programming relationships.

You can use the Object Browser to navigate through a list of development objects. Development objects are the components you use to build an application. You can also view a single development object. In the next exercise, you will learn how to navigate through an object list with the Object Browser.

When you use the Object Browser, it automatically calls other tools when your actions require them. For example, you create a new data definition from the Object Browser screen. The Browser calls the Data Dictionary and, after you create your definition, returns you to the Browser screen.

You can create an entire application using the Object Browser without directly calling any of the other tools. In fact, the recommended method for creating an application is from the Browser because you can see what you build. The lessons in this tutorial use this method.

Check Your Understanding

In this exercise, you learned a little about what each of the tools in the Development Workbench can do. Take a minute to check your understanding of what you have just learned. Match each of the following tasks with the Workbench tool that you would use to accomplish it:

- Debug a reporting program.
- Look up a data definition.
- Position a button on a dialog.

Exercise 2: Learning the Workbench Tools

- View a list of the screens in a program.

This exercise also introduced the Object Browser. You learned that creating a program from the Object Browser is the recommended method for creating ABAP programs.

Exercise 3: Navigating an Object List

An object list is graphical organization of related development objects. For example, a program object list contains all the objects in a program. The appearance of an object list [Ext.] is similar to a directory listing in a graphical file manager. In this exercise, you learn how to open and navigate through an object list. If you have not already done so, open the Development Workbench. Then, proceed as follows:

1. Choose *Object Browser* to start the Browser.

   The system displays the *Object Browser: Initial Screen*. At this point, you must choose whether you want to look at an object list or a single object. For this exercise, you will look at an object list.

2. Select *Program* in the *Object list* area and place your cursor in the adjacent field.

   The system displays a possible entries push button beside the field. This button indicates there are several possible entries.

3. Enter the name **`tutprog`** in the *Program* field.

4. Choose *Display*.

   The Browser displays the program development objects. You can see that the **`tutprog`** program contains several types of objects. Each object type has a folder icon beside it. A plus sign (+) in a folder means the folder is closed.

5. Open the *Global data* folder by clicking the folder icon.

   The Browser displays a list of the global data found in the **`tutprog`** folder. Notice that the *Global data* folder now has a minus sign (-) to tell you that it is open.

6. Open *ANSWER*.

   The Object Browser starts the ABAP Editor. If you have not used the Editor before, it asks you to confirm the mode. Choose *Continue* to confirm PC-mode. The Editor opens the **`tutprog`** program at the point where the global data *ANSWER* is declared [Ext.].

7. Use *Back* to return to the *Object Browser: Initial Screen*.

Check Your Understanding

In this exercise, you learned how to open an object list for a specific program. You used the Object Browser to navigate through the object list to the **`tutprog`** program. The **`tutprog`** program contains the example transaction you will create with this tutorial. Take a moment to open some of the objects in the program object list.

If you like, you can execute the program by selecting the TUTPROG node and choosing *Development Object* → *Test/execute*. When the system prompts you for an execution type, select *Direct*.

Exercise 4: Creating a Program

This exercise teaches you how to create a program. The program you create will eventually make up a complete application in the R/3 System. If you have not already done so, log onto your R/3 System and open the Workbench to the *Object Browser: Initial Screen*. To create a program, proceed as follows:

1. Select *Local priv objects*.

   Each user has a local private object list. By default, the Browser assumes you want to look at your own object list.

2. Choose *Display*.

   The system opens your object list. If this is the first time you opened the list, it contains only the *Development class object types* node. This node is interactive.

3. Double-click *Development class object types*.

   The system prompts you for the type of object you want to create.

4. Ensure that *Program object* is selected and choose *Continue*.

   The system displays a list of all the possible types of program objects.

5. Choose a program name.

   The recommended format for your program name is **SAPMZ<bbb>** where **<bbb>** are your initials.

6. Enter the program name you choose in the field provided [Ext.].

7. Ensure that *Program* is selected and choose *Create*.

   The system prompts you to confirm your selection.

8. Ensure that the program name is correct and that *With TOP INCL.* is set.

9. Choose *Continue*.

   The system prompts you for the name of the top include file. The convention for naming include files is **MZ<bbb>TOP** where **<bbb>** are your initials.

10. Enter a name for the include file and choose *Continue* to accept it.

    The system displays the program attribute screen.

11. Enter attribute values for your program as follows:

Title	**Create Flight Module Pool**
Type	**M**
Application	*****

12. Save your program attributes.

    The system creates both your program and the include file.

Check Your Work

In this exercise, you created a new program and an include file. You will use these objects later on in the tutorial. Right now, take a moment to check you work. Use the Object Browser to open

your local private object list. Alternatively, if you are still in the program attributes screen, use the *Back* button.

Use the *Object Browser* to open the *Programs* and *Includes* folders. Your object list should contain the program and the include you created  [Ext.].

Exercise 5: Setting a Marker

In this exercise, you learn how to set a marker. A marker is similar to a bookmark in that it lets you get to a specific list within the Browser quickly. To set a marker, proceed as follows:

1. Select *Goto* → *Markers*.

   The system displays a list of markers. The initial marker defines the first screen the system displays when you start the Object Browser. By default, the system sets *the Workbench Selection Screen* as the initial marker. You are going to set your private object list as the default marker.

2. Select *Development Class $TMP*.

3. Choose *Define init position*.

   The system places the *Development Class $TMP* in the initial position.

4. Press ENTER.

Check Your Work

Choose the *Back* button to return to the initial *ABAP Development Workbench* screen. Then, open the Object Browser. The system takes you immediately to your private object list without displaying the *Initial Selection* screen.

From the Object Browser menu bar, you can also select *Goto* → *Initial Screen* to return to the initial screen of the Browser.

Exercise 6: Creating a Transaction

Entwicklungsobjekt [Ext.]

This exercise teaches you how to create a transaction. A transaction is a module. To create a transaction, go to your private object list and proceed as follows:

1. Open the *Programs* folder and double-click your new program.

   The system displays the program object list. The list displays the objects related to your program. At this point, the list should be empty.

2. Double-click the *Program object types* folder.

   The system prompts you for a development object type.

3. Select *Transaction*.

   At this point, you must choose a transaction name. Use the format **ZF<xx>** where **<xx>** are your initials. Make sure you follow the SAP naming conventions [Page 14].

4. Enter a transaction name in the field provided and choose *Create*.

   The system prompts you for a transaction type.

5. Select *Dialog Transaction* and choose *Continue*.

   The system prompts you for some additional information.

6. Enter the following information:

Transaction Text	`Create Flight Data`
Program	The name of the program you created in exercise 4.
Screen number	`100`

7. Save your work.

   The system creates the new transaction.

Check Your Work

Your program object list should now contain a *Transactions* folder. Open your new transaction and note the information associated with a transaction.

Review of Lesson 1

Lesson 1 introduced you to the ABAP Development Workbench and the tools it contains. You learned that the Workbench is a programming environment for the ABAP language. This lesson taught you how to open and how to exit the Workbench.

The names of the six major Workbench tools should now be familiar to you:

- Object Browser
- ABAP Dictionary
- ABAP Editor
- Function Library
- Screen Painter
- Menu Painter

This lesson placed particular emphasis on the Object Browser. You learned that this tool provides a context for viewing your development environment. Using the Browser you looked at the different parts of an existing program and created a new program. You also learned how to set the initial marker.

In the Next Lesson...

Lesson 2 teaches you how to create a table. You will learn about concepts and more about some important parts of the ABAP Dictionary.

Lesson 2: Working with Tables

In this lesson, you learn how to create a table. This lesson contains the following information:

Introduction to Lesson 2

Tables play a key role in the R/3 System. Tables define a database's structure. You need a good understanding of how to work with tables before going on to design and create complex tables. After you complete this lesson, you will be able to:

- List the steps needed to create a table.
- Recognize the key components of a table.
- Identify the tool that creates tables.
- Create a table from the Object Browser.
- Activate a table.
- Add a field to a table.
- Define a value list.
- Specify how the system should handle a table.

When complete, your table will define the data used by your transaction (for example, departure time, flight number, and arrival date).

Components of a Table

Each table in the R/3 System is composed of several components. These components are the following:

table object	Represents a table in the ABAP Dictionary.
fields	Define information stored in a table.
data element objects	Describe field content and determine how a field is displayed to the end-user. Data elements appear as objects in the Dictionary. Because they are objects in their own right, you can reuse data elements within the same table or among fields in several tables.
domain objects	Describe valid values for a field. A domain specifies information like data type or number of positions in a field. Like data elements, domains are stored as objects in the Dictionary. You can reuse domains just as you can reuse data elements.
technical settings	Specify how the R/3 System handles a table.

Summary of Table Design Process

Normally, before creating a table you would take a moment and plan what kind of data you require. Your plan would include an understanding of the relationships between data. For this lesson, the planning was done for you. Once a plan is in place, you proceed as follows to create a table:

- Create a table object
- Specify fields
- Define data elements and domains

- Specify technical settings
- Activate the table

As you create your table, you should save your work. When you save an object, it is placed in the SAP database. The table status is set to saved. Other users can view saved objects, but they cannot access them in ABAP programs.

After you complete a table object, you activate it in the database with the *Activate* operation. When you activate a table, the system does the following:

- Checks the syntax.
- Updates the table status to active.
- Compiles a runtime version of the table.

Once a table is active, other programs and users can access it.

Exercise 1: Creating a Table Object

If you are not already there, use the ABAP Object Browser to display your table object. Go to your private object list and proceed as follows to create a table:

1. Select *Development class object types*.

   The system prompts you for an object type.

2. Double-click the *Dictionary objects* and choose *Continue.*

   The system prompts you for the type of Dictionary object you want to create.

   At this point, the system is processing your input using the ABAP Dictionary tool. You did not have to leave the Object Browser; the system opened the Dictionary for you.

3. Select *Table* and enter a table name in the adjacent field.

   Dictionary objects are global to the SAP system. The name you select must be unique. You should select a name in the form `Z<xx>FL`, where `<xx>` are your initials [Ext.].

4. Choose *Create*.

   The system prompts you for some additional information:

5. Enter the following information:

Short Text	`Flight Table`
Delivery Class	`A`

6. Set *Tab Maint. Allowed*.

7. Save your table.

   The system creates a table object and adds it to your local object list.

Check Your Work

Take a moment to check your work. Use *Back* to return to your local object list. You should now have a *Dictionary Objects* folder. Open the folder to view your new table object.

Exercise 2: Specifying Table Fields

In the previous exercise, you created an empty table object. In this exercise, you add the table fields. Recall that table fields define the information stored in a table. Normally, a table has one or more fields. You can create a table that has no fields but it would have little use in the R/3 System.

To identify a field, go to your private object list and proceed as follows:

1. Select your new table object.
2. Choose *Change*.

   The system displays the *Change Table/Structure Attributes* screen. You specify fields by entering information in the *Field Attributes* section of the screen.
3. Enter **`FLID`** for the name of the first field.

   The **`FLID`** field is the key field for the table.
4. Set the *Key* check box.

   When a checkbox is set, the box becomes filled [Ext.].
5. Save your changes.

   The system removes empty fields and saves your table.

Define the Remaining Fields

You now have a single field in your table object. The **`FLID`** field is the table KEY. Take a moment and identify the remaining fields in your table. You do not need to set the *Key* checkbox for the remaining fields. Select *New fields* to display some empty fields. Then, define the following table fields:

`LVCITY`

`LVDATE`

`LVTIME`

`REGLR`

`CHRTR`

`MOVIE`

`SNACK`

`FMEAL`

`ARCITY`

`ARDATE`

`ARTIME`

Check Your Work

At this point, you have created a table object and identified the table´s fields [Ext.]. You can look at your table from the Dictionary. Use *Environment* → *ABAP Dictionary* to open the dictionary from an Object Browser screen.

Exercise 2: Specifying Table Fields

Exercise 3: Defining Data Elements and Domains

In the previous exercise, you identified the fields you want in your finished table. This exercise teaches you how to define data elements and domains. Remember, data elements describe field content and determine how a field appears on a screen. Domains describe valid values for a field.

Before you begin, ensure that your table is open on your SAP Workbench. If the table is not open, use the *Object Browser* to reache and open your table object. To define a data element and a domain for a field, proceed as follows:

1. Ensure that the table is in change mode.

   To put the table in change mode, choose *Display↔Change*.

2. Choose a data element name for the **FLID** field.

   Because data elements are objects, they must have unique names. Choose a name in the form **Z<xx>_FLID**, where **<xx>** are your initials.

3. Enter the name you choose in the **FLID** *Data elem.* field.

4. Double-click the *Data elm.* field.

   The system asks you to confirm that you want to create the data element.

5. Choose *Continue*.

   The system displays the *Change Data Element* screen.

6. Enter the following information:

Short text	**Flight ID**
Domain name	**z<xx>_FLID**

   You can use the data element name for the *Domain name* field [Ext.].

7. Enter the following in the *Texts* group box of the screen [Ext.]:

Short	9	**Flight ID**
Medium	13	**Flight ID Num**
Long	16	**Flight ID Number**

   The information in the *Texts* group box is used later by the system to label fields in your interface automatically.

8. Save the data element.

9. Double-click the *Domain name* field.

   The system confirms your selection.

10. Choose *Continue.*

    The system displays the *Change Domain* screen.

11. Enter the following information:

Short text	`Flight ID`
Data type	`Char`
Field length	`5`

12. Choose *Activate*.

    The system saves the domain object and activates it. Activating an object makes it visible in the SAP database. The status of the domain should be *active*.

13. Go back to the *Dictionary: Change Data Element* screen.

14. Choose *Activate* to add the new data-element object to the SAP database.

15. Go back to the *Dictionary: Table/Structure Change Fields* screen and *Save* your work.

Check Your Work

You have defined two new objects: a data element object and a domain object. These objects now appear in your local private-object list. Open your local private object list. In the *Dictionary objects* folder, you have two new folders: *Data elements* and *Domains*. Open these new folders to view your new data element and domain.

Exercise 4: Reusing Domains

In the previous exercise, you defined a data element and domain for a single field in a table. From the *Change Table/Structure* screen you should see the completed information for the **Z<xx>_FLID** field. In this exercise, you will define the data elements and domains for the remaining fields in your table.

This exercise also teaches you how to reuse existing domain definitions. Recall that a field is associated with a single data element and single domain. You can reuse data elements and domains for fields in the same table or for fields in other tables. This exercise shows you how to share domains and data elements between fields in the same table.

Specify Additional Data Elements and Domains

If you are not already there, open the Change Table/Structure dialog. Then, using the procedure you learned in exercise 3, create data elements and domains for the following fields:

Field	Data Elm	Short Text	Domain	Data Type	Field Length
LVCITY	Z<xx>_LVCITY	Dep. City	Z<xx>_CITY	CHAR	5
LVDATE	Z<xx>_LVDATE	Dep. Date	Z<xx>_DATE	DATS	8
LVTIME	Z<xx>_LVTIME	Dep. Time	Z<xx>_TIME	TIMS	6

About the Data Elements: This exercise does not provide you with values for the *Texts* group box. Enter values for this box that make sense to you.

Reuse Domains with Remaining Data Elements

Now, use the 3 new domains you just defined (**Z<xx>_CITY**, **Z<xx>_DATE**, **Z<xx>_TIME**) and the system defined domain, CHAR1, to specify the remaining data elements and domains. Proceed as follows to define the remaining data elements:

1. Place your table in change mode if necessary.
2. Enter a data-element name in the *Data elem.* field.

   Because data elements are objects, they must have unique names. Choose a name in the form **Z<xx>_REGL**, where **<xx>** are your initials.
3. Select the *Data elem.* field.

   The system asks you to confirm that you want to create the data element.
4. Select *Cont.*

   The system displays the *Change Data Element* screen.
5. Enter the following information:

   Short text **Regular Flight**
   Domain name **CHAR1**

   CHAR1 is a system-defined domain.
6. Complete the *Texts* group box of the screen [Ext.].

Exercise 4: Reusing Domains

7. Save the data element.
8. Choose *Activate* to add the new data element to the Dictionary.
9. Go back to the *Change Table/Structure* screen and *Save* your work.
10. Repeat steps 2 through 9 for the remaining fields using the following information:

Field	Data Element	Short Text	Domain
CHRTR	z<xx>_CHRTR	Charter Flight	CHAR1
MOVIE	z<xx>_MOVIE	Inflight Movie	CHAR1
SNACK	z<xx>_SNACK	Light Snack	CHAR1
FMEAL	z<xx>_FMEAL	Full Meal	CHAR1
ARCITY	Z<xx>_ARCITY	Arrival City	Z<xx>_CITY
ARDATE	Z<xx>_ARDATE	Arrival Date	Z<xx>_DATE
ARTIME	Z<xx>_ARTIME	Arrival Time	Z<xx>_TIME

Do not forget to fill in the *Texts* group box for each data element.

Check Your Work

In this exercise, you have learned how to reuse existing domains. You learned that system defined domains also exist and that you can use them as well. Use the Object Browser to go to your local private-object list and view the new objects you created.

In your list, compare the number of new data elements to the number of domains. The number of data elements exceeds the number of domains. Notice that each domain only appears once in the list even though it is used by several fields. Also, you can see that system-defined domains (for example, CHAR1) do not appear in your domain list. This is because these objects are defined system-wide, not locally.

Exercise 5: Defining a Value List

In the last exercise, you defined domains for your table fields. Recall that a domain describes valid field values. Often, you can narrow down the acceptable values for a field to a set of fixed values. This set is called a fixed-value list [Ext.].

In this exercise, you learn how to define a fixed-value list for the **Z<xx>_CITY** domain. To define a fixed-value list for the **Z<xx>_CITY** domain, go to your private object list and proceed as follows:

1. Open the *Domains* folder.
2. Select your city domain and choose *Change*.

   The system displays the *Dictionary: Maintain Domain* screen.
3. Choose *Fixed Values*.
4. Enter **MUC** in the first lower limit column.
5. Enter **Munich** for the *Short text*.
6. Add the following values:

Value	Short Text
TXL	Tegel Field, Berlin
THF	Tempelhof, Berlin
DEN	Denver
EWR	Newark, New Jersey
JFK	New York
CDG	Paris
YYZ	Toronto

7. Save the list.
8. Go back to the *Dictionary: Maintain Domain* screen.
9. Choose *Activate* to save your changes and update the SAP database.

Check Your Work

Use the Object Browser to view your domains. Select the **Z<xx>_CITY** domain and check for the value list you just created.

Exercise 6: Specifying Technical Settings

This exercise teaches you how to define technical settings for your table. Recall technical settings influence how the system handles a table. If you have not already done so, open the *Table/Structure* screen. To define technical settings, ensure your table is in change mode and proceed as follows:

1. Choose *Technical settings*.
2. Ensure that the *Technical Settings* screen is in change mode.
3. Enter **`APPL1`** in the *Data class* field.

   Data c*lass* of **`APPL1`** indicates the table is updated frequently.
4. Enter a **`1`** in the *Size Category*.

   Size category of **`1`** specifies that the table is small.
5. Save your settings.
6. Go back to the *Dictionary: Table/Structure Change* screen.
7. Save your table.

Check Your Work

You have finished specifying the technical settings for your table. If you like, display the *Technical Settings* screen to verify your work.

Exercise 7: Activating a Table

In the previous exercises, you constructed a complete table object. This exercise teaches you how to activate a table. Recall that, after you activate a table, other programs can reference the table. To activate your table, proceed as follows:

1. Open your table object in change mode.
2. Choose *Activate*.

   The system compiles the underlying table code, creates a runtime version of the table, and updates the table status to active.

Check Your Work

When the compile is finished, the system changes the table *Status* field to *Active*  [Ext.].

Review of Lesson 2

In Lesson 2, you created a new table. This table defines the data used by your transaction, for example, departure time, flight number, and arrival date. You learned that tables have 5 components:

- A table object
- Fields
- Data element objects
- Domain objects
- Technical settings

Without these 5 components, you cannot add a table to the SAP database. The lesson also taught you that Dictionary objects like tables, data elements, and domains are system-wide objects.

During the course of this lesson, you went through the steps for creating a table. These steps are:

- Create a table object.
- Identify fields.
- Define data elements and domains.
- Specify technical settings.
- Activate the table.

You also learned that data elements and domains are re-usable. You can use data elements and domains you define for one table in many tables.

In the Next Lesson...

Lesson 3 teaches you how to use the Screen Painter tool. You will use the Screen Painter to design your transaction screen. Additionally, Lesson 3 teaches you how to layout gadgets and buttons. Finally, you will learn how to specify screen field attributes.

Lesson 3: Designing Screens

Lesson 3 teaches you about the Screen Painter. This lesson contains the following information:

Introduction to Lesson 3

In the previous lesson, you created a table that described your application's data. Lesson 3 teaches you how to create a screen where an end-user can enter data. A screen is an arrangement of graphical elements that appear in a window [Ext.]. After you complete this lesson, you will be able to:

- Identify major concepts associated with an ABAP screen.
- Create an initial screen.
- Arrange the elements in a screen.
- Characterize screen elements.
- Run a prototype of a screen.

Screens and menus make up the graphical user interface (GUI) for an ABAP application. Lesson 4 teaches you how to create menus. Before you complete this lesson, you must have successfully completed Lesson 2.

Screen Concepts

To create a screen, you must understand what components underlie a screen's graphical elements and the screen itself. Some examples of screen elements are push buttons, radio buttons, labels, and boxes. Each element has associated with it:

attributes	Describe a screen. Screen attributes include things like a description, a type, and position.
layout	Refers to the arrangement of elements on a screen.
field attributes	Describe an element. For example, a particular field accepts only character input.
flow logic	Describes the relationship between a screen element and its underlying application. Flow logic is a series of instructions.

You create and maintain all ABAP screen elements in the Screen Painter.

The Screen Painter has a fullscreen editor. You use this editor to layout screens. The fullscreen editor works in two modes: graphical and alphanumeric. In graphical mode, you use dialogs and a mouse to identify and create elements. Graphical mode is available only on Unix/Motif, Windows 95, and Windows NT platforms.

Alphanumeric mode, the default, is available on all platforms. This lesson uses the graphical mode.

Exercise 1: Creating a Screen

In this exercise, you create a screen and identify its attributes. Before you begin, ensure that you are in your program object list [Ext.] and proceed as follows:

1. Double-click *Program object types*.

   The browser displays the list of possible program objects.

2. Select *Screen*.

   At this point, you must specify a screen number.

3. Enter `100` for the screen number.

4. Choose *Create*.

   The system displays the *Change Screen Attributes* screen.

5. Enter the following information:

Short Description	`Create Flight Data`
Screen type	`Normal`
Next screen	`100`

   ABAP programs can determine the screen sequence dynamically. This means it is not necessary to specify a different screen for the *Next Screen* attribute.

6. Save the screen and return to the program object list.

Check Your Work

In this exercise, you created a single screen. You should now have a *Screens* folder in your program object list. In subsequent exercises, you will populate your new screen with graphical elements.

Exercise 2: Placing an Element on a Screen

In this exercise, you use the Screen Painter fullscreen editor to add an element to your screen. Ensure that you are in the object list for your program and proceed as follows:

1. Select *Screen 100* and choose *Change*.

   The browser displays the *Flow Logic Display* screen.

2. Ensure that you have the set the fullscreen editor to graphical mode.

   You can set the mode by choosing *Settings* → *Graph. Fullscreen*.

3. Choose *Fullscreen*.

   The system opens the *Screen Painter* fullscreen editor [Ext.].

4. Choose *Dict/Prog Fields*.

   The Screen Painter displays the *Dict./program fields* dialog. You use this dialog to copy existing fields from a table or program into your screen.

5. Enter your table name in the *Table/field name* field.

6. Choose *Get from Dict*.

   The system retrieves the list of fields from your table and lists them for you.

7. Select the flight ID field.

8. Set *Template* and set *Key word* to *Average*.

9. Choose *Copy*.

   The Screen Painter window moves into the foreground.

10. Place the table field by positioning your cursor in the work area and pressing the left mouse button.

    The flight ID appears with a label and an entry field in the work area. Once an element is placed, you can reposition it by selecting it again and dragging it to a new location [Ext.].

11. Save your changes.

Add the Remaining Fields

Try some new techniques to add the rest of the elements to your dialog. You should have the following elements remaining:

- Arrival/departure city
- Arrival/departure date
- Arrival/departure time
- Regular flight
- Charter flight
- Snack
- Meal

- Movie

Add several elements at once by selecting them in the *Dict./program fields* dialog. Add the departure city, date, and time using this method. Select the fields as a group [Ext.] and choose *Copy* to add them as a group [Ext.].

Add the remaining fields to your dialog. Group the arriving city, date, and time in the same manner as you grouped the departure elements. Add the regular and charter fields as a group. Finally, add the meal, snack, and movie fields as a block.

If you need to, you can position several elements by using a “rubber-band” to select a whole group of elements. Then, drag the group to another place in the screen [Ext.].

Check Your Work

You have created a simple interface using the Screen Painter. Try looking at your interface as it will appear to your users. While still in the fullscreen editor, choose *Screen* → *Test*. The system prompts you for the window coordinates.

Your new screen appears in the SAP application window. Experiment with your interface. Try to enter values for each of your fields. You will find that, for some fields, the domain helped determine interface behavior. For example, try entering a departure time using the possible entries button.

Exercise 3: Refining a Screen

In this exercise, you learn how to refine the look of your screen. You will change two existing text fields into radio buttons and you will add labels and boxes to define different areas of your screen. Before you begin, open the Screen Painter fullscreen editor to screen `100`.

Convert Fields to Radio Buttons

The *Regular* and *Charter* fields on your screen describe two types of flights. A flight is either one type or the other. In an interface, radio buttons are used to indicate a single choice among a set of mutually exclusive choices. To convert the text fields to radio buttons, proceed as follows:

1. Use a “rubber-band” to select both the *Regular* and *Charter* fields.

   The Screen Painter indicates the fields are selected.

2. Choose *Edit* → *Convert* → *Radio button* → *Button left*.

   The Screen Painter converts both fields to radio buttons. Now, you need to group the buttons so that they are mutually exclusive.

3. Select *Edit* → *Radio button group* → *Define*.

   The Screen Painter groups the buttons. Small handles in the shape of diamonds appear along the sides of the group [Ext.]. Use the square handle at the top to move the group. Use the diamond handles to reshape the group if necessary.

4. Save your changes.

Add a Box

You can add a box around the schedule elements on your screen to indicate that they are related:

1. Select the *Frame* tool from the palette.
2. Move your cursor into the work area.

   The cursor changes shape to indicate you are about to draw a box.

3. Draw a box around the flight information in your screen.

   Position the cursor to the left of the group. Hold down the left mouse button and drag the cursor down and to the right [Ext.]. The painter creates a frame around the group. When you release the left mouse button, the box remains selected.

4. While the box is selected, use the keyboard to type `Schedule`.

   The system creates a title for your new box.

5. Repeat steps 1 through 3 to create a box around the radio group.
6. Title the new box `Classification`.
7. Save your changes.

Convert Fields to Check Boxes

Convert some existing fields to check boxes. Check boxes are used for choices that are either set or not set. When a choice is set, a check mark appears in the box. Proceed as follows to add check boxes:

1. Select the *Snack, Meal,* and *Movie* fields.

   The Screen Painter indicates both fields are selected.

2. Choose *Edit* → *Convert* → *Check box* → *Button left*.

   The Screen Painter converts the fields to check boxes. Because each check box is an individual choice that does not rely on the other choices, you do not need to group the check boxes.

3. Save your changes.

Check Your Work

Take a moment to examine what you have accomplished. Test your newly refined dialog. Your screen in simulation mode appears as it will to your users [Ext.]. Recall that, by definition, radio buttons have logic. If one button is selected, the others must not be. As you test your dialog, you will notice the Screen Painter added the radio button logic for you.

Exercise 4: Checking the Screen Layout

In this exercise, you learn how to check the layout of a screen and use the *Attributes* dialog to position an element. It is a good idea to follow the R/3 style guidelines when creating your interface. Open the Screen Painter fullscreen editor to screen 100. Then check the layout of your screen by choosing, *Screen* → *Check* → *Layout*. The system lists the errors in your layout.

You can use the *Attributes* dialog to quickly fix the position of an element. To use the dialog, proceed as follows:

1. Select the element.
2. Press *Attributes*.

   The system displays the current attributes for the element.
3. Enter a new value in the *Line* field.

   This value determines the vertical position of the element.
4. Enter a new value in the *Column* field.

   This field determines the horizontal position of the element.
5. Choose *Close*.

   The system changes the position of the element.

Check Your Work

Test your update to screen 100. You should notice the difference in the position of your screen elements [Ext.].

Exercise 5: Setting the OK Field

In this exercise, you leave the fullscreen editor and use the *Field list* view to set the OK field. Each SAP screen has a single OK code. This OK code is used to pass information from the screen to the underlying application. To set the OK code, open your screen to the *Flow Logic Display* screen and proceed as follows:

1. Choose *Field list* or, if the screen is already open in change mode, select *Goto* → *Field list views* → *List field types*.

   The system displays the field list for your screen. This list is a table representation of the same data available through the *Attributes* dialog.

2. Page down until you find a *FType* specification of *OK*.

   The *Field name* for this specification is empty [Ext.].

3. Enter `OK-CODE` for the *Field Name*.

   The `OK-CODE` field is available with every screen. It is an invisible element and so does not appear in the Screen Painter fullscreen editor. You will use the OK code to pass information from the interface back to your application.

4. Save your changes.

Check Your Work

To check your work, display the *General Field Attribute* list for your screen. The `OK-CODE` field appears at the bottom of the list.

Review of Lesson 3

Lesson 3 taught you how to use the Screen Painter. You learned that a screen is an arrangement of graphical elements that appear in a window. Graphical elements include things like radio buttons, push buttons, text fields, and check boxes.

You learned that graphical elements have attributes. Attributes include things like the name of the element and the position of the element on the screen.

This lesson taught you how to refine a screen by adding boxes and labels to different areas of a screen. Finally, you learned how to use the Screen Painter to view a prototype of your new interface.

In the Next Lesson...

Screens and menus make up the graphical user interface (GUI) for an ABAP application. In lesson 4 you learn how to create menus.

Lesson 4: Specifying a GUI Status

This lesson introduces you to the process of creating a GUI status. The lesson contains the following information:

Introduction to Lesson 4

Lesson 4 teaches you how to use the Menu Painter tool to create a GUI status and its accompanying menu bars. In the previous lesson, you created screens [Ext.]. In this lesson, you add menu bars to the screens. After you complete this lesson, you will be able to:

- Define the concepts underlying ABAP menus.
- Create a menu bar for a screen.
- Define function keys.
- Create a tool bar for a screen.
- Specify window titles.

To complete this lesson, you must have first successfully completed Lesson 3.

Menu Concepts

Within ABAP you use two tools to create a GUI. You use the Screen Painter to create screens that contain radio buttons, check boxes, text fields, and push buttons. You use the Menu Painter to create the interface components. These components are the following:

status	Defines the combination of menu bars, menu lists, F-key settings, and functions available to an interface. For example, an Editor application might have two statuses: edit and view. In edit status, the cut function is available and with the view status the cut function is unavailable.
menu bars	Define functions available to the user. Where the functions appear depend on the dialog. If the dialog is modal, the functions appear at the bottom of the interface as a row of buttons [Ext.]. In the primary window, the functions can appear both as a row of menus and as buttons in a tool bar.
menu list	Lists the items in a specific menu. For example, an edit menu might contain items like copy, cut, and replace.
F-key settings	Define keyboard keys associated with a particular interface function.
Functions	Define individual functions such as cut, copy, and replace.
Titles	Define window titles for an interface.

You can share components between statuses. For example, you can define a delete function. Then, you can use this function in the statuses belonging to an editor application, a file manager, and an accounting application.

After you create a GUI status, you must generate [Ext.] it. When you generate a menu, the system creates a runtime version of the status. The runtime form is used when a user executes an application.

Exercise 1: Create a GUI Status

A GUI status [Ext.] is associated with a particular screen or set of screens. In this exercise, you will define a GUI status for your screens. To define a status, do the following:

1. Open your program object list.
2. Double-click on *Program Object Types*.

   The system prompts you for an object type.
3. Select *GUI Status*.
4. Enter `100` in the status field.

   The value you enter in the status field is used as the status identifier. If you like, you can specify an identifier that suggest the function of the status. For example, you might call the status `createflt`.
5. Choose *Create*.

   The system displays the *Create Status* dialog.
6. Enter `Create Flight Data` in the *Short Text* field.
7. Select *Screen*.
8. Choose *Continue*.

   The system displays the *Maintain Status* dialog.
9. Save your new status.

Check Your Work

Take a moment and ensure that you successfully created your first GUI status. Because a status is a programming object, the system created a new *GUI status* folder in your program's object list. Check the contents of the folder to make sure your new status is there.

Exercise 2: Add Menus to Your Interface

Exercise 2 teaches you how to add menu bars to the screens you created in Lesson 3. If you have not already done so, open the browser to your program object list. Then, do the following:

1. Open the status you created in exercise 1.

   The system displays the *Maintain Status* dialog.

2. Ensure that the status is in change mode.

   For this example, you will use the standard default menus.

3. Click on the *Display standards* button.

   The system displays the standard default menus [Ext.].

4. Replace the <*Object*> menu with **`Flight`**.

   To do this, place your cursor in the menu field and enter **`Flight`**.

5. Double-click on **`Flight`**.

   The system displays a list of the standard menu items. To define menu items, you must enter valid values in the function column.

6. Enter the following menu items:

   ```
   CREA  Create
   UPDA  Change
   DISP  Display
   DELE  Delete
   ```

7. Save your changes.

Check Your Work

Test your new menu bar along with your *Create Flight Data* screen. Ensure that the GUI status 100 is open and select *User Interface* → *Test status*. In the *Status Simulation* dialog, enter `100` for the screen number.

When you run your simulation, notice that the system has automatically provided two menus: *System* and *Help*.

Exercise 3: Define Function Keys

In this exercise, you define function keys for your menus. If you have not already done so, open status `100` in your program. Ensure the status is in change mode and do the following:

1. Enter **`Flight F-Keys`** in the *F key assignment* field.
2. Scroll down to the *Recommended function key settings* area.
3. Enter **`DELE`** as the function and for the *Shift-F2* key.
4. Scroll down to the *Freely assigned function keys* area.
5. Fill in the first three keys with the following:

F5	**`UPDA`**	**`Change`**
F7	**`DISP`**	**`Display`**
F2	**`CREA`**	**`Create`**

6. Save your changes.

Check Your Work

Take a moment to test your latest changes. Right now, you cannot test the function keys except to check that choosing them does not cause the system to return an error.

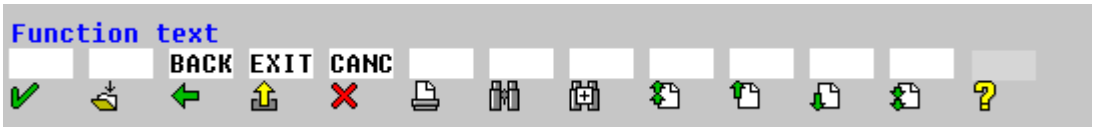
Exercise 4: Specify PushButtons

This exercise teaches you how to specify pushbuttons in the application and standard toolbars. You can specify a push button for any function that is defined for a function key. For example, in the last exercise you created an **Create** function key. In this exercise, you create a corresponding *Create* button. Open status 100 in change mode and do the following:

1. Select the first *Application toolbar* setting.
2. Enter the **CREA** function and press ENTER.

   The system automatically displays the text **Create** and an appropriate icon. Recall that you entered this text when you defined the function key.
3. Enter the remaining functions:

   DISP
   UPDA
   DELE
4. Add the SAP default functions (back, exit, cancel) to the *Standard toolbar* area:



5. Save your changes.

Check Your Work

Use the *Test* function to view the new push buttons.

Exercise 5: Finish Up

In the previous exercise, you define a status for a dialog window. In this exercise, you learn how to create a title for a screen or a window. You also learn how to generate [Ext.] your GUI status. Before you begin, ensure that you are in your program's object list.

Title Your Interface

To specify a title for your interface, do the following:

1. Select the *Program object types* folder.

   The system displays a list of the object types you can create.

2. Select *GUI Title*.
3. Enter `100` for the title code.
4. Choose *Create*.

   The system displays the *Create Title* dialog.

5. Enter `Create Flight Data` for the title.
6. Choose *Save*.

Generate Your Status

Recall that when you generate a status, the system prepares a load version of the menu for the user. To generate your new status, ensure that you are in your program's object list and do the following:

1. Select GUI status `100`.
2. Choose *Generate/activate*.

   The system sends a message to the status line telling you status `100` was generated [Ext.].

Check Your Work

The system created a new folder for you in your program's object list. This folder contains all the GUI titles. You can use the test function to test your completed interface: the titles, the screens, and the menus. This time, you can specify both a screen number and a title number.

Review of Lesson 4

Lesson 4 taught you about GUI statuses. Each ABAP interface has at least one GUI status [Ext.]. A GUI status is associated with a menu bar. You learned that a GUI status describes menu bar elements and their availability.

You learned how to use the Menu Painter tool to create menu bars. You used the Menu Painter to create menus, function keys, and push buttons. The Menu Painter's test feature allowed you to run a prototype of your new interface.

Finally, you created a GUI title. You also learned that a complete interface requires a screen, a status, and a title.

In the Next Lesson...

The next lesson introduces you to writing code. You learn how to tie your interface to an underlying transaction. You also learn how to provide feedback to your users through messages. Lesson 5 concludes with a short example of how to use the debugging tool.

Lesson 5: Coding the Transaction

Lesson 5 teaches you how to use some of the features in the ABAP Editor. This lesson covers the following topics:

Introduction to Lesson 5

In this lesson, you use the editor tool to enter ABAP code. After you complete this lesson, you will be able to:

- Identify underlying coding concepts.
- Create screen flow logic.
- Generate a screen interface.
- Create code modules.
- Copy code modules from existing modules.
- Check syntax.
- Create messages.

Coding Concepts

To complete your application, you must create the instructions that tell the R/3 System how to process information your users enter. These instructions consist of the screen [Ext.] flow logic and the ABAP modules [Ext.].

Flow logic is associated with each screen. Screen flow logic is divided into processing that occurs before output and processing that takes place after the user enters data in a screen. There are only 20 screen keywords for writing screen flow logic. Keywords can, in turn, refer to ABAP modules.

The process before output (PBO) modules and the process after input (PAI) modules are specific ABAP modules that direct processing. The PBO and PAI modules are stored in the module pool of the transaction.

In this lesson, you also create a message class. A message class groups messages used by a particular application or group of applications.

Exercise 1: Writing Flow Logic

This first exercise teaches you how to write flow logic for your screens. Flow logic is written using less than 20 keywords. These keywords provide pointers to ABAP code modules. Before you can begin, you must be in your program object list.

1. Open screen 100.

   The system places you in the Screen Painter Flow Logic Editor automatically.

2. Ensure the screen is in change mode.

3. Enter the following below the *process before output* entry [Ext.]:

   ```
   module initialize_100.
   ```

   The MODULE keyword identifies the ABAP module that defines the processing.

4. Insert several blank lines after the *process after input* entry.

5. Enter the following keywords and values:

   ```
   field <tablename>-<idfield>.
   module fcode_100.
   ```

   <tablename> is the name of your table. **<idfield>** is the name of the flight ID field. Remember to enter a period after each statement.

6. Choose *Screen* → *Generate*.

   Generating the screen saves your changes and prepares a runtime version [Ext.] of your screen.

Check Your Work

You have now completed the screen flow logic for your screen. Use the Screen → *Check* → *Syntax* function of the Screen Painter to make sure you did not make any typing errors when entering your logic code.

Exercise 2: Creating Modules

In the last exercise, you created screen flow logic that referenced specific ABAP modules. Exercise 2 teaches you how to create these modules.

1. Open screen 100 in change mode.
2. Open the `initialize_100` module by double-clicking it.

   The system asks you if you want to create a new module. If the module already existed, the system would take you to it automatically.
3. Choose *Yes*.

   The system displays the *Create PBO Module* dialog. This dialog asks for the name of an include file to place the module in.
4. Select *New include*.

   At this point, you must choose a name for the include file. The name should be in the form `MZ<bb>O01`. Be sure that you use the SAP naming conventions [Page 14].
5. Enter the name of your new include file and choose *Continue*.

   The system displays the include file with the new `initialize_100` module.
6. Save the new include file and return to the flow logic screen.
7. Repeat steps 1 through 6 for the `fcode_100` module but use the form `MZ<bb>I01` for the include file name.

Check Your Work

Test your changes. When you double-click a module definition in the flow logic for a screen, the system automatically opens the code that the definition references. Try this with your new modules. You can also open your table automatically by selecting the field reference in the screen flow logic.

Exercise 3: Specifying Global Variables

In this exercise, you specify global variable declarations. You add these declarations to your program top include file. Recall that you created the top include file [Page 22] in lesson 1. To specify global variables, ensure that you are in your program object list and proceed as follows:

1. Select your top include file and choose *C*hange.

   The system opens your file for editing. The *program* declaration must be the first one in the file. The system added this declaration automatically when you created the include. You should enter your changes below the program declaration line.

2. Declare your table with the following statement:

   ```
   tables: <tablename>.
   ```

   Substitute your table name for the `<tablename>` variable. You must declare a variable for each screen element that accepts input and from which your program takes data.

3. Add the following declarations to the file:

   ```
   data answer.
   data: ok-code(4), fcode(4).
   ```

   At this point, you must choose a name for your message class. The name should be in the form `z<x>` or `y<x>`. Be sure that you use the SAP naming conventions [Page 14].

4. Extend the PROGRAM statement by adding a message ID.

   ```
   program sapmzaaa message-id <xx>.
   ```

5. Save your changes.

Check Your Work

Test and make sure you entered your global data correctly. Select *Program* → *Check* → *Current Program* to check your program syntax. The system displays any errors.

You can also double-click the table name. When you double-click the table name, the system opens the table for you.

Exercise 4: Coding the Modules

In the previous exercise, you used the ABAP Editor to add code to your top include file. In this exercise, you enter code using the keyboard and special features of the Workbench environment both to copy code and automatically insert predefined functions.

Entering Code with the Editor

1. Open the *PBO modules* folder in your program object list.
2. Select the *INITIALIZE_100* module and choose *Change*.

   The system opens the module in change mode.
3. Enter the following in the line following *module initialize_100 output* [Ext.]:

```
set pf-status '100'.
set titlebar '100'.
```

4. Save your changes and return to your program list.

Copy Code from Another Program

Programmers often create new programs by copying and then modifying existing programs. You can do this with the Workbench tools.

1. Return to the Object Browser initial screen.
2. Select *Program* and enter **TUTPROG** for the program name.

   The system takes you to the example program.
3. Choose *Display*.
4. Open PAI module **FRAGMENT**.

   The system opens the module in display mode.
5. Place your cursor in the line containing the *fcode = ok-code* entry.
6. Choose *Select*.

   The system marks the line [Ext.].
7. Select the *endcase* entry on line 97 and choose *Select*.
8. Select *Block/clipboard* → *Copy to clipboard*.
9. Exit the module and return to your program object list.

   If you like, you can use the *Markers* feature to return to your object list.
10. Select your **FCODE_100** module and choose *Change*.
11. Place your cursor in the empty line below **module fcode_100 input**.
12. Choose *Block/clipboard* → *Insert from clipboard*.

    The system inserts the section from **FRAGMENT**.
13. Use the *Edit* → *Replace* function to replace **wbtable** with your table name [Ext.]:
14. Save your changes.

Insert a System Function Automatically

You can use the built-in capabilities of the ABAP Editor to insert function templates automatically into your code. You then customize these templates to fit your application. Insert a function automatically into your code by doing the following:

1. Open your **fcode_100** module in change mode.
2. Find the **endif** entry that occurs immediately before the **when space** entry.

   Look for the **endif** entry appears around line 97.
3. Insert a blank line before the **endif** line.
4. Select *Edit* → *Insert statement*.

   The system displays the insert statement dialog.
5. Select *CALL FUNCTION*.
6. Enter **POPUP_TO_CONFIRM_LOSS_OF_DATA** in the space provided.
7. Choose *Continue*.

   The system enters a template for the function.
8. Add code to the IF statement and finish the function template.

   When you have finished, the code should look similar to the following The red text indicates the information you must add:

```
message e004 with <tablename>-flid.
else.
      call function 'POPUP_TO_CONFIRM_LOSS_OF_DATA'
            exporting
                  textline1 = 'Delete Flight?'
                  TEXTLINE2 = ' '
                  titel = 'Attention'
                  START_COLUMN = 25
                  START_ROW = 6
      importing
                  answer = answer.
      check answer ne 'N'.
      delete <tablename>.
      clear <tablename>.
      message s003 with <tablename>-flid.
endif.
```

9. Save your changes.

Check Your Work

Use the *Program* → *Check* function to check the syntax in your module.

Exercise 5: Creating a Message Class

This exercise teaches you how to create a message class. You use a message class to deliver appropriate messages to your enduser. To create a message class, open your program object list and proceed as follows:

1. Open your top include file.
2. Double-click the message ID.

   The system asks you to confirm that you want to create a new message class.
3. Choose *Yes*.

   The system displays the *Maintain Message Class* screen.
4. Enter the following in the *Short text* field:

   `Flight Application Messages`
5. Save your changes.

   The system displays the *Maintain Object Catalog Entry* dialog.
6. Choose Local *o*bject.

   The system returns you to the *Maintain Message Class* screen.
7. Choose *Messages*.

   The system displays the *Maintain Messages* screen.
8. Choose *Maintain All* to edit the messages.
9. Enter the following [Ext.]:

`000`	`Flight already exists`
`001`	`Flight & was CREATED successfully.`
`002`	`Flight & was UPDATED successfully.`
`003`	`Flight & was DELETED successfully.`
`004`	`Flight & does not exist.`

   At runtime, the system replaces the ampersand (`&`) with the appropriate flight ID number.
10. Save your changes.

Check Your Work

Open the `fcode_100` module. Double-click a message ID, for example `e000`. If you created your messages successfully, the system takes you to the message definition.

Exercise 6: Testing Your Transaction

In this example, you test your transaction in a separate SAP session. To run your transaction, go to your program object list and proceed as follows:

1. Select your program name and choose *Development object* → *Generate/activate*.

   If your program contains syntax errors, the system displays an error message.

2. Open a second SAP session by choosing *System* → *Create session*.
3. Enter your transaction name in the command field [Ext.].

   The system starts your transaction in the session window.

4. Create a flight.
5. Return to your first SAP session and go to your private object list.
6. Open the *Dictionary Objects* folder.
7. Select your table and choose *Execute*.

   The system displays the *Data Browser Selection* Screen.

8. Select *Program* → *Execute* to display your table.

   The system displays all the flights in your application database.

Check Your Work

Experiment with each of the functions on your interface. You might want to:

- Create a flight bound for Peru.
- Delete an existing flight.
- Display a flight you know does not exist.
- Change an existing flight.

Be sure to check and make sure each of your error messages appears at an appropriate point in each interaction. When you have finished experimenting, review your flight database again.

Exercise 7: Running the Debugger

You can use the debugger to identify problems in your application code. You can start the debugger when you are in your transaction by entering `/h` in the command prompt. For this exercise, you start your debugger from your program object list:

1. Select your transaction and choose *Development Object* → *Test Execute*.

   The system displays the *Execution Types* dialog.

2. Select *Debugging* and choose *Continue*.

   The system opens your program in debugging mode.

3. Double-click the flight ID field declaration.

   The system places the variable in the *Variables* group box. Initially, there is no flight ID.

4. Choose *Single-Step*.

   The system starts executing your program. After each step, it returns you to the debugger.

5. Step through the remainder of your program.

Check Your Work

You can examine your flight table internally to check your work. From your program object list, select your table object and choose *Environment* → *Data Browser* → *Contents.* Once you are in the data browser screen, you can choose *Execute* to view a table of your entries.

Review of Lesson 5

Lesson 5 taught you how to use the editor tool to enter ABAP code. You learned about some of the operations you can use to create your own applications. In this lesson, you used the editor to:

- Create screen flow logic.
- Generate a screen interface.
- Create PAI and PBO modules.
- Copy code modules from existing modules.
- Insert and modify a function template.

You also learned how to create a message class and add messages to the class.

In the Next Lesson...

Lesson 6 completes your work with the create flight transaction example. It introduces you to the tools and concepts you will need when programming in a team environment.

Lesson 6: Working in a Team

This lesson introduces you to the tools and the concepts you need to develop ABAP applications in a team environment. The following topics are discussed:

Introduction to Lesson 6

By now, you should have a good idea of the role each Workbench tool plays in application development. This lesson introduces you to the tool and the concepts you will need to develop an application with a team of programmers. When you complete this lesson, you will be able to:

- Understand concepts underlying team development with ABAP
- Create a development class in the R/3 system
- Track changes to your development class
- Release an object

Team ABAP Development Concepts

ABAP allows you to divide work on large projects among several programmers. Consider an accounting application project with an accounts payable module and an accounts receivable module. The ABAP environment helps you to create a work area in the system for the project. You can then assign tasks to each programmer and follow their work as it progresses.

The tool you use for tracking development projects is called the Workbench Organizer. The steps for creating a large project in the ABAP development group, are:

- **Create a development class.**

  A development class [Ext.] groups objects that are logically related, such as the objects that make up an accounting application. A development class is a type of development object [Ext.].

- **Create a change request.**

  A change request [Ext.] records the changes made to a development object. For example, creating a program in a development class is considered a change to a development object. A change request is associated with a single ABAP user.

- **Program the project.**

  ABAP programs consist of transactions, reports, screens, and other development objects. If the components of a project are constructed by more than one programmer, you must assign each programmer a task [Ext.] under the change request. Tasks help you to track who made changes to a program.

- **Release the change request.**

  While a change request or a task is associated with a development object, the object is locked. The lock also prevents other users from editing the object. Once changes are complete and tested, a programmer releases the object.

In a standard SAP installation, a single machine acts as the development system and another machine serves as the production system. New applications are created in the development system and transported to the production system. Daily work takes place in the production system.

The division between production and development systems is necessary because of when application changes take effect. When a change is made to an existing ABAP application, the change takes immediate effect. The division prevents development work adversely affecting daily work flow.

Exercise 1: Creating a Development Class

In this exercise, you create a development class for a fictitious accounting application. Recall that a development class groups objects that are logically related. As a by-product of this exercise, you will also create a change request. Change requests record an application's development and lock the objects in the class from modification by unauthorized users.

Go to the object browser initial screen and proceed as follows:

1. Select *Development Class*.

   At this point, you must choose a name for your development class. For this example, use a name in the form **z<xxx>** where **<xxx>** are your initials.

2. Enter the name of a development class.
3. Choose *Display*.

   The system confirms the class does not exist and asks you if you want to create it.

4. Choose *Yes*.

   The system displays the *Change View "Development Classes" Details* screen.

5. Enter descriptive short text for your new development class [Ext.].
6. Choose *Save*.

   The system prompts you for a change request.

7. Choose *Create request*.

   The system displays the *Create Request* screen. At this point, you must choose a descriptive name for the change request. It is a good idea to choose a name that reflects the development stage, for example, initial development or release 1.0 [Ext.].

8. Enter a short description and choose *Save*.

   The system returns you to the *Change Request* dialog.

9. Choose *Enter*.

   The system places you at the top of your new development class list.

Check Your Work

At this point, your development list contains a single folder for development class objects. You can check the information associated with the class by double-clicking the development class name.

Exercise 2: Examining Change Request List

In this exercise, you learn how to look at the change requests associated with your user name. Recall that a change request records the changes made to a development object. Each request is associated with a single ABAP user. To look at the change requests associated with your user name, go to your new development class and proceed as

follows:

1. Choose *Environment* → *Workbench Organizer.*

   The system opens the *Workbench Organizer: Initial Screen*.

2. Ensure that the following values are set:

   Requests for user

   Modifiable

   Local

3. Choose *Display* from the *Selection* group box.

   The system displays the change requests associated with your name.

Check Your Work

Open all the folders in your change request list  [Ext.]. Your change request list contains a single change request and a task. There is a user name associated with a change task. Your list should show that you have performed only a single action in association with your new development class; that action is the creation of the class itself.

Exercise 3: Adding Another Programmer

In the last exercise, you learned how to display your own change request [Ext.] list. Right now, only your name appears in association with your list. However, three programmers will construct the accounting application. To track the activities of another programmer under the same change request list, you create a task [Ext.] for the user by adding the user to the change request list.

1. Open the change request list.
2. Select the change request.

   If you do not select the change request, the system does not allow you to add a user.
3. Choose *Add user*.

   The system displays the *Add User* screen.
4. Enter a user name.
5. Choose *Continue*.

   The system displays the user in the list.

Check Your Work

At this point, your change request list should contain two programmers [Ext.]. Because the new user has not performed any actions associated with his or her task, no folder appears besides his or her name.

Exercise 4: Creating a Program

You are already familiar with how to create a program from your work with Lesson 1 [Page 22]. However, the procedure for creating a program in a team environment is slightly different. When you create a program or any development objects in a team environment, you must identify both a development class [Ext.] and a To create a program in a team environment, proceed as follows:

1. Open the development object list you created in exercise 1.
2. Double-click *Development class object types*.

   The system prompts you for the type of object you want to create.
3. Ensure that *Program Object* is selected and choose *Cont*.

   The system displays a list of all the possible types of program objects.
4. Choose a program name.

   Recall that there are specific SAP naming conventions [Page 14] you should follow. The recommended format for your program name is **SAPMZ<bb>** where **<bb>** are your initials.
5. Enter the program name you choose in the field provided.
6. Ensure that *Program* is selected and choose *Create*.

   The system prompts you to confirm your selection.
7. Ensure that the program name is correct.
8. Turn off *With TOP INCL*.
9. Choose *Continue*.

   The system displays the program attribute screen.
10. Enter attribute values for your program as follows:

Title	**Tutorial: Accounts Payable**
Type	**M**
Application	*****

11. Choose *Save*.

    The system displays the *Query for Change Request* screen.
12. Enter change request you created in Exercise 1 [Page 73].
13. Choose *Continue*.

    The system returns you to your program attributes screen.

Check Your Work

Display your change request list. Your personal task contains an ABAP program folder. Within this folder, you can find your new program. If you like, try opening your program by double-clicking its label.

Exercise 5: Releasing the Change Request

In this exercise, you release the change request you created earlier. When you release a change request, you make new programs or new features available to other users in the system. After a request is released, other users can change the objects it contains. You cannot release a change request without first releasing all the associated tasks. To release your change request, display your change request list and proceed as follows:

1. Select the task that belongs to you.
2. Press *Release*.

   The system displays the *Document request/task* screen.
3. Enter a short description of the task you are releasing.

   For example, your description could appear as follows:

   ```
   Initial release of accounting program.
   ```
4. Choose *Save final version*.

   The task is stored in preparation for release.
5. Choose *Back*.

   The system returns you to your change request list. Because you cannot release a task that does not belong to you, you must first associate all the tasks in the request with your user name. You can do this because you own the change request.
6. Select the task belonging to the other user.
7. Choose *Change owner*.

   The system displays the *Change User Name* dialog.
8. Enter your user name in the *New User* field.
9. Choose *Confirm*.

   The system reassigns the task to your user name. Because this task has no work associated with it, you should delete it rather than release it.
10. Select the task and choose *Request/task* → *Delete*.

    The system asks you to confirm your request.
11. Choose *Yes*.

    The system deletes the empty task.
12. Choose the change request folder.
13. Choose *Release*.

    The system releases all the changes into the system.

Check Your Work

Take a minute to check your changes. Go to the *Workbench Organizer: Initial Screen* and set *Modifiable* and *Released* [Ext.]. When you next display the change requests associated with your user name, both the released and modifiable (current) change requests appear. You can view only released requests by turning *Modifiable* off.

Exercise 5: Releasing the Change Request

Review of Lesson 6

Lesson 6 introduced you to the main concepts you need to develop ABAP applications in a team environment. You should now be familiar with the following terms:

- Development class
- Change request
- Task

You learned that a development class groups objects that are logically related. For example, all the development objects associated with an accounting application are logically related. You also learned that you track an application's development using change requests and tasks. Both change requests and tasks are associated with a single user.

You learned how to use the Workbench Organizer to display all the change requests associated with a user. You used the organizer to add a user to an existing change request. Later, you changed the owner of a task so that you could delete it. Finally, you used the organizer to release your program for use in the system at large.

Where to Go From Here

You have completed the lessons included with the *ABAP Workbench Tutorial*. You should have a good idea of the role each tool plays in the development of an ABAP application. At this point, if you are familiar with ABAP programming, you should go on to read the ABAP Development Workbench Tools [Ext.] documentation. This documentation discusses each Workbench tool in detail.

For information on programming in ABAP, see the ABAP User's Guide [Ext.].